Science of Computer Programming 243 (2025) 103269

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Generalizing neural network verification to the family of piece-wise linear activation functions

László Antal *, Erika Ábrahám, Hana Masara

RWTH Aachen University, 52074 Aachen, Germany

ARTICLE INFO

Keywords:
Neural network verification
Piece-wise linear activation functions
Star-based reachability analysis
Exact analysis
Over-approximate analysis
Convex relaxations
Unbounded input sets

ABSTRACT

In this paper, we extend an available neural network verification technique to support the full class of *piece-wise linear* activation functions. Furthermore, we extend the algorithms, which provide in their original form exact, respectively, over-approximative results for bounded input sets represented as star sets, to allow also *unbounded* input sets. We implemented our algorithms and demonstrate their effectiveness on some case studies.

1. Introduction

In the area of artificial intelligence, *feed-forward neural networks (FNNs)* [1] enjoy increasing popularity. An FNN can be trained [74] to learn a function $f : \mathbb{R}^n \to \mathbb{R}^m$ from a set of input-output samples, and predict outputs also for previously unseen inputs. This way, FNNs can tackle problems that would otherwise require very complex solutions [2].

Nowadays, a wide range of applications use FNNs, such as autonomous vehicles [3], speech- and object-recognition systems [4,5] or robot vision [6], just to mention a few. While FNNs are impressively effective, their reliability in safety-critical situations is still questionable [7–9]. Hence, verification methods play an important role in providing guarantees about their behavior. In this work, we focus on the *reachability problem* for FNNs, which is the problem of determining which output values (*reachable set*) an FNN computes for inputs from a given set.

Related work An early example of the application of *formal methods* [10–13] to verify neural networks was given in [14]. Since then, the verification of neural networks has gained significant attention from the formal methods research community [15–26].

Verification techniques are broadly categorized into exact (complete) methods and relaxed (incomplete) methods [27].

i. Exact verifiers commonly solve the verification problem using MILP solvers [28–30] or Satisfiability Modulo Theories (SMT) solvers [31,32,25,17,19]. These methods are typically restricted to networks with piece-wise linear activation functions, such as ReLU, as the reachability problem for other activation types, like sigmoid or tanh, is in most cases undecidable. Furthermore, the NP-completeness of the problem makes scaling to large networks challenging.

ii. Relaxed verifiers, on the other hand, employ polynomially-solvable techniques like convex optimization or linear programming (LP) [33–35,15,16,22,36]. These methods are faster and scalable, relying on propagation-based approaches. However, this efficiency comes at the cost of reduced tightness, leading to higher false-negative rates and potentially failing to certify safety even when

* Corresponding author.
E-mail addresses: antal@informatik.rwth-aachen.de (L. Antal), abraham@informatik.rwth-aachen.de (E. Ábrahám), hana.masara@rwth-aachen.de (H. Masara).
URLs: https://ths.rwth-aachen.de/people/laszlo-antal/ (L. Antal), https://ths.rwth-aachen.de/people/erika-abraham/ (E. Ábrahám).

https://doi.org/10.1016/j.scico.2025.103269
Received 24 June 2024; Received in revised form 30 December 2024; Accepted 29 January 2025

Available online 5 February 2025

present. The effectiveness of relaxed methods is strongly dependent on the tightness of the relaxation, which determines how closely the relaxed model approximates the original [27,37].

Most current methods for relaxed verification focus on convex, single-neuron relaxations, that consider only the input and output domain of a single neuron [27]. These methods balance a trade-off between tightness (effectiveness) and computational efficiency. Examples include zonotopic relaxations [38,39], non-parallel lower and upper bounds [40], or tight convex relaxations like the Δ-relaxation [17]. Recently, researchers have explored multi-neuron relaxation techniques, which achieve tighter approximations [41,42]. However, these approaches often introduce an exponential number of constraints, significantly increasing computational complexity.

We apply a propagation-based technique and build on previous work [43,44,22] to compute reachable sets, where *star sets* are used to represent subsets of $\mathbb{R}^k$ for any $k \in \mathbb{N}$ with $k > 0$, like sets of input and output values. In those works the authors present two methods, one with exact computations, and one relaxed method which over-approximates the reachable sets.

Contributions The contributions of this paper, which is an extended version of the FMAS 2023 workshop paper [45], are the following:

1. We extend the set of activation functions supported by [43,44,22] to include *leaky ReLU*, *hard tanh*, *hard sigmoid*, and *unit step*. For each of these activation functions, we provide comprehensive formalisms and present the corresponding reachability analysis algorithms using both exact and over-approximative computations. While some of these functions had partial implementations in existing tools, our work offers complete formalizations and extends support to more general, parameterized versions of these activation functions. Furthermore, we introduce a general solution for arbitrary piece-wise linear functions, both for the exact and for the relaxed technique.
2. While previous work was restricted to bounded input sets, we provide extensions to allow also *unbounded* input sets.
3. Using the open-source library HyPro[1] [46] for the star-set representation, we developed a C++ *implementation* of both the exact and the over-approximative analysis methods, covering all the above activation functions. This includes also an extension of HyPro with an NNET parser to input FNN models in NNET file format.
4. We propose some novel benchmarks (thermostat and sonar classifier) with the aim of supporting the formal methods community. Using our implementation, we provide *experimental evaluation* on these two proposed benchmarks and two other existing benchmarks.

Besides minor extensions and improvements regarding the presentation, the extensions with respect to [45] cover (i) the generalized methodology to handle any piece-wise linear activation function, and (ii) further strengthening of the theoretical basis through additional proofs. Moreover, we extended our implementation to support both the exact and the relaxed analysis for unbounded input sets, and the exact reachability analysis for general piece-wise linear activation functions. To evaluate these new functionalities, we introduced some additional (unbounded) safety properties in our benchmarks, carried out experiments with them, and analyzed the obtained results.

Outline The rest of this paper is structured as follows. We present in Section 2 the fundamentals of this work, including feedforward neural networks (FNN), star sets, and the reachability analysis of FNN with the rectified linear unit (ReLU) activation function. Then, in Section 3, we propose exact and relaxed analysis methods for several other activation functions, followed by a general solution for arbitrary piece-wise linear activation functions, considering both bounded and unbounded input sets. In Section 4, we present and evaluate experimental results on four different benchmarks. Finally, in Section 5, we conclude the paper and discuss future work.

2. Preliminaries

We use $\mathbb{N}$ to denote the set of all natural numbers including 0, $\mathbb{R}$ for the reals, and use lower indices to specify subsets thereof, e.g. $\mathbb{R}_{>0}$ for the positive reals. We consider elements from $\mathbb{R}^n$ (for any $n \in \mathbb{N}_{>1}$) to be column vectors.

2.1. Feedforward neural networks

A *feedforward neural network (FNN)* [47,48] is a directed weighted graph annotated with some data. It has a finite set of nodes called *neurons*, which are grouped into $k \in \mathbb{N}_{\geq 2}$ disjoint non-empty ordered sets $l_1, \ldots, l_k$ called *layers*. We call l_1 the *input layer*, l_k the *output layer*, while the others are *hidden layers*. Let $\langle i \rangle$ denote the number of neurons $|l_i|$ in layer $i = 1, \ldots, k$, which we call its *size*. There is a directed edge from each neuron n in each non-output layer l_{i-1} to each neuron n' in the next layer l_i, weighted by $w_{n',n} \in \mathbb{R}$; let $\mathbf{W}_i \in \mathbb{R}^{\langle i \rangle \times \langle i-1 \rangle}$ be the matrix whose entry in row r and column c is the weight of the edge from the cth neuron in layer $i-1$ to the rth neuron in layer i. In addition, each neuron n in each non-input layer is annotated with a *bias* $b_n \in \mathbb{R}$ and an *activation function* $act_n : \mathbb{R} \to \mathbb{R}$. For each non-input layer i with neurons $l_i = \{n_1, \ldots, n_{\langle i \rangle}\}$, let $\mathbf{b}_i = (b_{n_1}, \ldots, b_{n_{\langle i \rangle}})^T$ and $\mathbf{act}_i : \mathbb{R}^{\langle i \rangle} \to \mathbb{R}^{\langle i \rangle}$ with $\mathbf{act}_i(\mathbf{y}) = (act_{n_1}(y_1), \ldots, act_{n_{\langle i \rangle}}(y_{\langle i \rangle}))^T$ for any $\mathbf{y} = (y_1, \ldots, y_{\langle i \rangle})^T \in \mathbb{R}^{\langle i \rangle}$. A frequently used activation function is the *Rectified Linear Unit (ReLU)*, defined as $ReLU(x) = max(0, x)$ for $x \in \mathbb{R}$. An example of FNN is shown in Fig. 1.

[1] The implementation is available online at https://github.com/hypro/hypro. For reproducing the experimental results, please check the Case Studies/Neural Network Verification subsection of HyPro's GitHub page: see the `README.md` file.

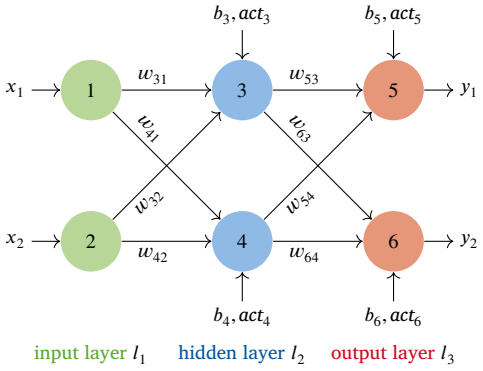


Fig. 1. Illustration of a feed-forward fully-connected neural network, consisting of one input layer (green), one hidden layer (blue), and one output layer (red). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

For an *input* $\mathbf{x}_1 = (x_1, \ldots, x_{\langle 1 \rangle})^T \in \mathbb{R}^{\langle 1 \rangle}$, the *state* $\mathbf{x}_i$ of each non-input layer l_i is defined recursively as

$$\mathbf{x}_i = \mathbf{act}_i \left(\mathbf{W}_i \, \mathbf{x}_{i-1} + \mathbf{b}_i \right) \ . \tag{1}$$

Thus, an FNN can be seen as a function $f : \mathbb{R}^{\langle 1 \rangle} \to \mathbb{R}^{\langle k \rangle}$, assigning to each input the output layer's state, which we call the *output*. For a given FNN and a set $\mathcal{R}_1 \subseteq \mathbb{R}^{\langle 1 \rangle}$ of possible inputs, the *FNN reachability problem* is the problem to compute all possible states for each of the layers $1 < i \leq k$ [44]:

$$\mathcal{R}_i = \left\{ \mathbf{act}_i(\mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b}_i) \,\middle|\, \mathbf{x}_{i-1} \in \mathcal{R}_{i-1} \right\} \ . \tag{2}$$

Solving the FNN reachability problem allows us to check properties of interest, e.g. safety properties (whether the output set is disjoint from a set of unsafe outputs) or stability (whether the distance between possible outputs is below a threshold for a given input set).

In this work, as input sets we consider convex polyhedra $\mathcal{R}_1 = \left\{ \mathbf{x} \in \mathbb{R}^{\langle 1 \rangle} \,\middle|\, \mathbf{A}\mathbf{x} \leq \mathbf{c} \right\}$ for some $p \in \mathbb{N}_{\geq 1}$, $\mathbf{A} \in \mathbb{R}^{p \times \langle 1 \rangle}$, and column vector $\mathbf{c} \in \mathbb{R}^p$.

2.2. Stars

To compute $\mathcal{R}_i$ via Equation (2), the two main operations that need to be applied on the state sets are the activation function $\mathbf{act}_i$ and *affine transformations* using the weights $\mathbf{W}_i$ and biases $\mathbf{b}_i$ of layer i. For implementing these calculations efficiently, different state set representations have been proposed [49]. Under these, star sets (or short stars) turned out to be very useful, as they provide efficient operations needed in neural network verification, such as affine transformations (see Proposition 2.2), half-space intersections (see Proposition 2.3), emptiness checking (see Proposition 2.4) and linear optimization (see Proposition 2.5).

Definition 2.1 *(Star).* For any $n, m \in \mathbb{N}_{\geq 1}$, an (n, m)-dimensional *star* is a tuple $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ of (i) a *center* $\mathbf{c} \in \mathbb{R}^n$, (ii) a *generator matrix* $\mathbf{V} \in \mathbb{R}^{n \times m}$ whose columns $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(m)} \in \mathbb{R}^n$ are called the *basis vectors* or *generators*, and (iii) a *predicate* $P \subseteq \mathbb{R}^m$. The star θ *represents* the set $[\theta] = \left\{ \mathbf{c} + \sum_{j=1}^{m} (\alpha_j \mathbf{v}^{(j)}) \mid (\alpha_1, \ldots, \alpha_m)^T \in P \right\}$.

As in [44], we restrict P to be a convex polyhedron $P = \{ \alpha \in \mathbb{R}^m \mid \mathbf{C}\alpha \leq \mathbf{d} \}$ for some $p \in \mathbb{N}_{\geq 1}$, $\mathbf{C} \in \mathbb{R}^{p \times m}$ and $\mathbf{d} \in \mathbb{R}^p$. The following star properties, whose proofs are included in Appendix A.1, will be used to solve the FNN reachability problem.

Proposition 2.1 *(Convex polyhedra). For any $m, p \in \mathbb{N}_{\geq 1}$, $\mathbf{C} \in \mathbb{R}^{p \times m}$ and $\mathbf{d} \in \mathbb{R}^p$, the convex polyhedron $\mathcal{P} = \left\{ \mathbf{x} \in \mathbb{R}^m \mid \mathbf{C}\mathbf{x} \leq \mathbf{d} \right\}$ can be represented by a star.*

Proposition 2.2 *(Affine transformation). Assume an (n, m)-dimensional star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ and let $\mathbf{W} \in \mathbb{R}^{k \times n}$ and $\mathbf{b} \in \mathbb{R}^k$. Then the affine transformation $\left\{ \mathbf{W}\mathbf{x} + \mathbf{b} \mid \mathbf{x} \in [\theta] \right\}$ of θ is represented by $\bar{\theta} = \langle \bar{\mathbf{c}}, \bar{\mathbf{V}}, P \rangle$ with $\bar{\mathbf{c}} = \mathbf{W}\mathbf{c} + \mathbf{b}$ and $\bar{\mathbf{V}} \in \mathbb{R}^{k \times m}$ with columns $\mathbf{W}\mathbf{v}^{(1)}, \ldots, \mathbf{W}\mathbf{v}^{(m)}$.*

Proposition 2.3 *(Intersection with a halfspace). Assume an (n, m)-dimensional star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ and a half-space $\mathcal{H} = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{h}^T \mathbf{x} \leq g \right\}$ with some $\mathbf{h} \in \mathbb{R}^n$ and $g \in \mathbb{R}$. Then the intersection $[\theta] \cap \mathcal{H}$ is represented by the star $\bar{\theta} = \langle \mathbf{c}, \mathbf{V}, P \cap P' \rangle$ with $P' = \left\{ \alpha \in \mathbb{R}^m \mid (\mathbf{h}^T \mathbf{V})\alpha \leq g - \mathbf{h}^T \mathbf{c} \right\}$.*

Proposition 2.4 *(Emptiness check). A star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ is empty if and only if P is empty.*

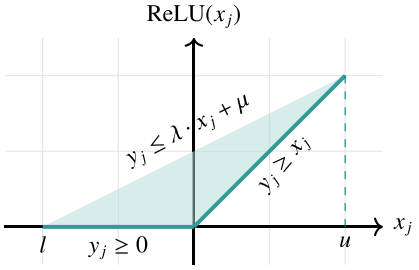


Fig. 2. Relaxation of the ReLU function for bounded input sets with $\lambda = \frac{u}{u-l}$ and $\mu = -\frac{lu}{u-l}$, where l and u are the lower respectively upper bounds on the input variable x_j [50]. Dark lines represent the exact set, the light area shows the approximate set.

Proposition 2.5 *(Bounding box). Assume an (n,m)-dimensional star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ such that $\mathbf{c} = (\mathbf{c}_1, \dots, \mathbf{c}_n)^T$, and let $\mathbf{V}_{(i)}$ be the i^{th} row of $\mathbf{V}$. Let furthermore the star's bounding box defined as $B = \left\{(x_1, \dots, x_n)^T \in \mathbb{R}^n \mid \bigwedge_{i=1}^{n} lb_i \leq x_i \leq ub_i \right\}$ with $lb_i = \mathbf{c}_i + \min_{\alpha \in P} \mathbf{V}_{(i)}\alpha$ and $ub_i = \mathbf{c}_i + \max_{\alpha \in P} \mathbf{V}_{(i)}\alpha$ for $i = 1, \dots, n$. Then $[\theta] \subseteq B$.*

2.3. Reachability analysis for FNNs with ReLUs

Next, we present two algorithms proposed in [44,75] to solve the reachability problem for FNNs with the ReLU activation function for bounded polyhedral input sets. These algorithms form the basis for our work. The first algorithm is exact, complete, and it can synthesize counter-examples. The second algorithm is relaxed and it can be used to over-approximate reachability, but it is not suited to create counter-examples.

We note that alongside ReLU, the work [22] includes some other activation functions but no complete formalizations were available. In Section 3, we will extend these algorithms to support some specific, but also any arbitrary piece-wise linear activation functions, as well as unbounded input sets.

Exact analysis

The exact algorithm first constructs a star from the input set which is required to be a polyhedron (see Proposition 2.1). Then, correspondingly to Equation (2), it propagates the star through the network, layer-by-layer, until we get the reachable set $\mathcal{R}_k$ of the output layer. This propagation involves two main operations.

(1) For each non-input layer i and each star representing possible state sets of the previous layer, to compute the reachable states of layer i, we first apply an affine transformation on the star, using the weight matrix $\mathbf{W}_i$ and the bias vector $\mathbf{b}_i$. Thus, from a star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ we obtain a new star $\theta' = \langle \mathbf{c}', \mathbf{V}', P \rangle$ with $\mathbf{c}' = \mathbf{W}_i\mathbf{c} + \mathbf{b}_i$ and $\mathbf{V}' = \mathbf{W}_i\mathbf{V}$ (see Proposition 2.2). Note that during the affine transformation, the predicate does not change.

(2) Then the activation function is applied on the intermediate star θ' dimension-wise to represent $\mathcal{R}_i = \mathbf{act}_{n_{\langle i \rangle}}\left(\dots \mathbf{act}_{n_1}\left([\theta']\right) \dots\right)$, where, $n_1, \dots, n_{\langle i \rangle}$ are the neurons in layer i. Since we consider the ReLU activation function, the $\mathbf{act}_{n_j}(\cdot)$ operation at neuron n_j is defined as $ReLU(x_j) = max(0, x_j)$; instead of $\mathbf{act}_{n_j}(\cdot)$ we write $\mathbf{act}_j^{\mathrm{R}}(\cdot)$ to denote that the ReLU function is applied in dimension j (i.e., at the jth neuron of a layer). To compute $\mathbf{act}_j^{\mathrm{R}}(\theta)$ for a star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$, we decompose θ into two stars $\theta_1 = \langle \mathbf{c}, \mathbf{V}, P_1 \rangle$ and $\theta_2 = \langle \mathbf{c}, \mathbf{V}, P_2 \rangle$ such that $[\theta_1] = [\theta] \cap \left\{(x_1, \dots, x_n)^T \in \mathbb{R}^n \mid x_j < 0\right\}$ and $[\theta_2] = [\theta] \cap \left\{(x_1, \dots, x_n)^T \in \mathbb{R}^n \mid x_j \geq 0\right\}$ (see Proposition 2.3).

On the negative branch, i.e., when $x_j < 0$, the ReLU function sets the corresponding values to zero. Thus, all the resulting elements of the star θ_1 should have the value zero in dimension j. We can obtain this result by applying the mapping matrix $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{j-1}, \mathbf{0}, \mathbf{e}_{j+1}, \dots \mathbf{e}_n]$ on θ_1, where $\mathbf{e}_i \in \mathbb{R}^n$ is the ith n-dimensional unit vector (with 1 at position i and 0s otherwise). On the positive branch $x_j \geq 0$, the ReLU function does not change the set elements of θ_2.

Thus, the application of ReLU results in the union of two stars $\mathbf{act}_j^{\mathrm{R}}(\theta) = \langle \mathbf{Mc}, \mathbf{MV}, P_1 \rangle \cup \langle \mathbf{c}, \mathbf{V}, P_2 \rangle$. Note that if the values in $[\theta]$ in the given dimension j are purely positive or purely negative, then the result of $\mathbf{act}_j^{\mathrm{R}}(\theta)$ is just a single star (the other one being empty).

Relaxed analysis

While the exact algorithm is complete, it suffers from scalability issues since, during analysis, the number of stars grows *exponentially* with the number of ReLU activations. To tackle this problem, one solution is to side-step to over-approximative computations, which makes the analysis more *scalable*, however, it sacrifices the *completeness* of the method.

The over-approximative method from [44] also builds on Equation (2), but the application of the activation functions is different: the original $\mathbf{act}_j^{\mathrm{R}}(\cdot)$ operation is replaced by an over-approximating one $\overline{\mathbf{act}}_j^{\mathrm{R}}(\cdot)$, which produces only a single star as output as follows. A fresh variable α_{m+1} and three more constraints are added to the predicate P of the star, with the purpose of capturing the over-approximation of the ReLU function at neuron n_j (see Fig. 2).

The three new constraints are: $\alpha_{m+1} \geq 0$, $\alpha_{m+1} \geq x_j$, and $\alpha_{m+1} \leq \frac{u(x_j - l)}{u - l}$, where l and u are the lower and upper bounds, respectively, for variable x_j in $[\theta]$ (see Proposition 2.5). Finally, since we want the new variable α_{m+1} to hold the over-approximation of x_j, after

introducing the new variable and constraints to the predicate, we need to update the center $\mathbf{c}$ and basis $\mathbf{V}$ of the star θ correspondingly. First, the old values of x_j are projected out using the mapping matrix $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{j-1}, \mathbf{0}, \mathbf{e}_{j+1}, \dots \mathbf{e}_n]$. Then, a new generator $\mathbf{e}_j$ is added to the basis, to link x_j to α_{m+1}.

Formally, for an (n, m)-dimensional star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ we define $\overline{\mathbf{act}}_j^{\mathrm{R}}(\theta) = \langle \bar{\mathbf{c}}, \bar{\mathbf{V}}, \bar{P} \rangle$, where $\bar{\mathbf{c}} = \mathbf{M}\mathbf{c}$, $\bar{\mathbf{V}} = [\mathbf{M}\mathbf{v}^{(1)}, \mathbf{M}\mathbf{v}^{(2)}, \dots, \mathbf{M}\mathbf{v}^{(m)}, \mathbf{e}_j]$ and $\bar{P} = \left\{ (\alpha_1, \dots, \alpha_{m+1}) \in \mathbb{R}^{m+1} \;\middle|\; (\alpha_1, \dots, \alpha_m) \in P \wedge \alpha_{m+1} \geq 0 \wedge \alpha_{m+1} \geq x_j \wedge \alpha_{m+1} \leq \frac{u(x_j - l)}{u - l} \right\}$.

In case $l \geq 0$ or $u \leq 0$, the introduction of a new variable is not necessary, and we can proceed in a similar way as in the exact case, i.e., for positive domain we keep the set as it is, for negative domain we project out the variable x_j. Note that this over-approximative method yields the tightest possible relaxation that we can achieve for the ReLU function [27].

3. FNN reachability analysis for piece-wise linear activation functions

Neural networks offer flexibility in choosing different activation functions. In this section, we first present extensions of the reachability analysis algorithm from the previous section to support (1) unbounded input sets and (2) the *leaky rectified linear unit (leaky ReLU)*, *hard hyperbolic tangent (HardTanh)*, *hard sigmoid (HardSigmoid)*, and *unit step* activation functions. Afterwards, we further generalize our formalisms and present the respective algorithms for analyzing *arbitrary piece-wise linear* activation functions.

We remark, that we handle the leaky ReLU function for bounded sets identically as it is done in the NNV tool [22]; we include the corresponding formalisms for completeness, and we extend them to support unbounded input sets.

Below we define each of the above functions and their application to a given input star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$.

3.1. Unbounded input sets

During the analysis of an FNN, it may happen that one or more variables x_j of a star θ become unbounded. That is, it has no lower bound (i.e., $l = -\infty$) or it has no upper bound (i.e., $u = \infty$). In the following, we present how to handle unbounded input sets as well. Essentially, the exact reachability analysis of any piece-wise linear activation function does not change in case of unbounded input sets. The same steps are applied as per the exact analysis of bounded sets, i.e., (1) splitting the input set into multiple subsets based on the cases of the activation function, and (2) applying the corresponding transformations to each subset.

However, the relaxed analysis does work differently for unbounded input sets. In the following, we show for each activation function how their convex relaxations can be adjusted to achieve the *tightest* possible single neuron relaxation in case of unbounded inputs. We distinguish for each function three cases of unboundedness of a variable x_j: either it has no lower bound ($l = -\infty$ and $u \in \mathbb{R}$), it has no upper bound ($l \in \mathbb{R}$ and $u = \infty$), or it has neither of the bounds ($l = -\infty$ and $u = \infty$). We do not formalize the unbounded case for ReLU explicitly, but define it as a special case of the leaky ReLU with $\gamma = 0$.

3.2. Leaky ReLU layer

Due to the dead neuron problem [51,52] caused by the ReLU function, its alternative, the leaky ReLU function proposed by Mass et al. [53], is used in many applications.

Definition 3.1 *(Leaky ReLU [54]).* The *leaky ReLU* activation function with scaling parameter $\gamma \in [0, 1) \subset \mathbb{R}$ is defined for each $x \in \mathbb{R}$ as

$$LeakyReLU(x) = max(\gamma \cdot x, x) = \begin{cases} x & \text{if } x \geq 0 \\ \gamma \cdot x & \text{otherwise} \end{cases}. \tag{3}$$

Exact analysis

The application of the leaky ReLU activation function is similar to the previously presented algorithm for the ReLU activation function, but they handle the negative inputs differently: While the ReLU function completely projects the input to zero, the leaky ReLU just scales the input down by $\gamma \in (0, 1)$. Thus, the application $\mathbf{act}_j^{\mathrm{L}}(\theta)$ of leaky ReLU on a star θ can be computed as follows.

First, we split the star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ into two subsets $\theta_1 = \langle \mathbf{c}, \mathbf{V}, P_1 \rangle$ and $\theta_2 = \langle \mathbf{c}, \mathbf{V}, P_2 \rangle$ with negative resp. non-negative x_j-values. Then we apply the corresponding transformations for both subsets. As previously, in case of the positive subset θ_2, no transformation is needed, since the leaky ReLU acts as an identity function for non-negative inputs. However, in case of the negative subset θ_1, we apply the scaling matrix $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \gamma\mathbf{e}_j, \dots \mathbf{e}_{n-1}, \mathbf{e}_n]$. Thus, the final result of the $\mathbf{act}_j^{\mathrm{L}}(\cdot)$ operation at neuron n_j is the union of two stars: $\mathbf{act}_j^{\mathrm{L}}(\theta) = \langle \mathbf{M}\mathbf{c}, \mathbf{M}\mathbf{V}, P_1 \rangle \cup \langle \mathbf{c}, \mathbf{V}, P_2 \rangle$.

The same observations apply here, that if the domain of a variable x_j is only negative (i.e., $u \leq 0$) or only positive (i.e., $l \geq 0$), then the final result of the $\mathbf{act}_j^{\mathrm{L}}(\cdot)$ operation is a single star: either $\theta_1 = \langle \mathbf{M}\mathbf{c}, \mathbf{M}\mathbf{V}, P_1 \rangle$ or $\theta_2 = \langle \mathbf{c}, \mathbf{V}, P_2 \rangle$.

Relaxed analysis

The relaxed analysis of the leaky ReLU is also similar to the one for ReLU. For bounded inputs, correspondingly to the Planet relaxation [50], we also try to find an enclosing triangle, which is the tightest convex relaxation that we can achieve for leaky ReLUs (see Fig. 3).

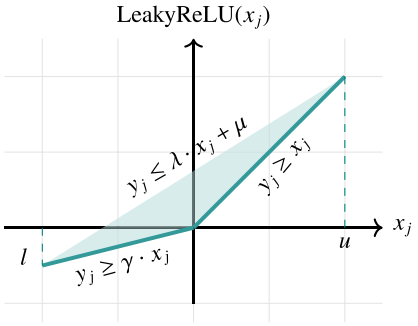
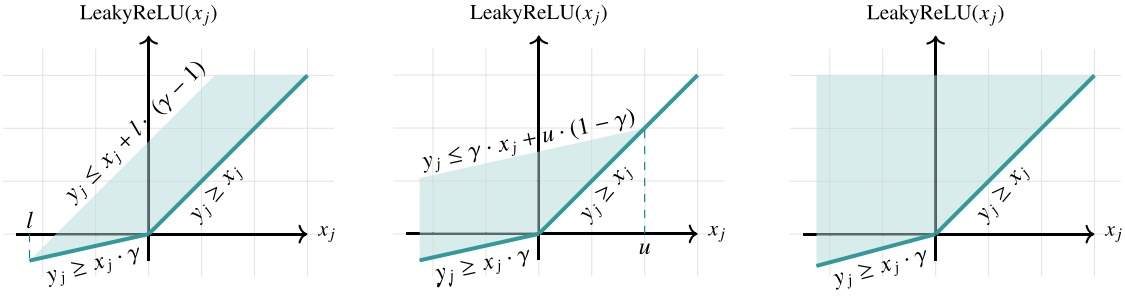
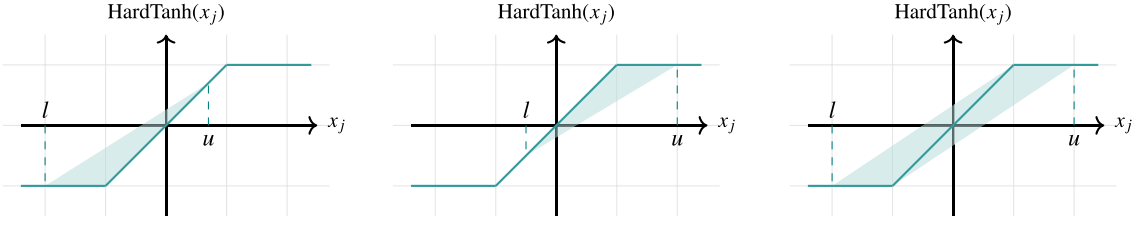


Fig. 3. Relaxation for the leaky ReLU function for bounded input sets, where l and u are the lower respectively upper bounds on the input variable x_j, $\lambda = \frac{u-\gamma \cdot l}{u-l}$ and $\mu = \frac{u \cdot l \cdot (\gamma-1)}{u-l}$. The dark line shows the exact set and the light area the approximate set.



(a) $x_j \in (l, \infty)$ (b) $x_j \in (-\infty, u)$ (c) $x_j \in (-\infty, \infty)$

Fig. 4. Convex relaxations of the leaky ReLU function with three cases of an unbounded input set.



(a) $l < V_{\min} \wedge u \in [V_{\min}, V_{\max}]$ (b) $l \in [V_{\min}, V_{\max}] \wedge u > V_{\max}$ (c) $l < V_{\min} \wedge u > V_{\max}$

Fig. 5. Relaxation for the hard tanh function for bounded input sets, where l and u are the lower respectively upper bounds on the input variable x_j. The dark lines show the exact set (non-convex) and the light area shows the approximate set (convex and linear).

The three constraints on the freshly introduced variable α_{m+1} are the following: (1) $\alpha_{m+1} \geq \gamma \cdot x_j$, (2) $\alpha_{m+1} \geq x_j$, and (3) $\alpha_{m+1} \leq \frac{u-\gamma \cdot l}{u-l} x_j + \frac{u \cdot l \cdot (\gamma-1)}{u-l}$. At this point, the result of the $\overline{\mathbf{act}}_j^{\mathrm{L}}(\cdot)$ operation is a single star with one more variable and three more constraints than the original input star. We note that if the domain of variable x_j is fully positive (i.e., $l \geq 0$) or fully negative (i.e., $u \leq 0$), then the resulting star is the same as described for the exact approach.
For unbounded input sets θ, we distinguish the cases (1) $x_j \in (-\infty, u)$, (2) $x_j \in (l, \infty)$, and (3) $x_j \in (-\infty, \infty)$. The analysis for unbounded input is similar to the bounded case, but the introduced constraints change, as visualized in Fig. 4. Note that these are the tightest linear, convex relaxations that can be achieved.

3.3. Hard tanh layer

The hard hyperbolic tangent function, commonly known as the hard tanh function and illustrated in Fig. 5, is a linearized variant of the hyperbolic tangent activation function. In our work, we have generalized this function by introducing the parameters $V_{\min}$ and $V_{\max}$, which replace the original values of -1 and 1, respectively [55]. This modification allows us to flexibly adapt the function according to our specific needs and requirements.

Definition 3.2 *(Hard hyperbolic tangent).* The *hard hyperbolic tangent (HardTanh)* activation function with parameters $V_{\min} \in \mathbb{R}$ and $V_{\max} \in \mathbb{R}_{>V_{\min}}$ is defined for each $x \in \mathbb{R}$ by

$$HardTanh(x) = \begin{cases} V_{\min} & \text{if } x < V_{\min} \\ x & \text{if } V_{\min} \leq x \leq V_{\max} \\ V_{\max} & \text{if } x > V_{\max} \end{cases}. \tag{4}$$

Exact analysis

For the analysis of FNNs with the hard tanh activation function at neuron n_j, which we denote as $\mathbf{act}_j^{\mathrm{H}}(\cdot)$, we split the result of the affine transformation $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ into three subsets: $\theta_1 = \langle \mathbf{c}, \mathbf{V}, P_1 \rangle$ is the intersection of θ with the hyperplanes $V_{\min} \leq x_j \leq V_{\max}$, $\theta_2 = \langle \mathbf{c}, \mathbf{V}, P_2 \rangle$ with $x_j < V_{\min}$, and $\theta_3 = \langle \mathbf{c}, \mathbf{V}, P_3 \rangle$ with $x_j > V_{\max}$ (see Proposition 2.3).

According to Equation (4), $\mathbf{act}_j^{\mathrm{H}}(\cdot)$ leaves the elements of θ_1 unchanged since x_j is in the range between $V_{\min}$ and $V_{\max}$. For θ_2, all of its elements get the value $V_{\min}$ in dimension j since $x_j < V_{\min}$, hence, we project the star onto $V_{\min}$ in the dimension j. To achieve this, we apply the mapping matrix $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{j-1}, \mathbf{0}, \mathbf{e}_{j+1}, \dots \mathbf{e}_n]$. Additionally, we set the j^{th} dimension of the center to $V_{\min}$ by adding the shifting vector $\mathbf{s}_{\min} = [0, \dots, V_{\min}, \dots, 0]^T$ to the center. For θ_3, we do the same by mapping the set with the mapping matrix, but instead, we set the center to $V_{\max}$ by adding the shifting vector $\mathbf{s}_{\max} = [0, \dots, V_{\max}, \dots, 0]^T$ to it. Thus, we project the star onto $V_{\max}$ in the dimension j. Accordingly, the $\mathbf{act}_j^{\mathrm{H}}(\theta)$ operation at neuron j results in the union of three stars: $\mathbf{act}_j^{\mathrm{H}}(\cdot) = \langle \mathbf{c}, \mathbf{V}, P_1 \rangle \cup \langle \mathbf{Mc} + \mathbf{s}_{\min}, \mathbf{MV}, P_2 \rangle \cup \langle \mathbf{Mc} + \mathbf{s}_{\max}, \mathbf{MV}, P_3 \rangle$.

Note that some of the intersections of the input star θ with the halfspaces $V_{\min} \leq x_j \leq V_{\max}$, $x_j < V_{\min}$, and $x_j > V_{\max}$ may be empty (see Proposition 2.4). In that case, we can spare the computation for the empty subsets, and continue the reachability analysis only with the non-empty resulting stars.

Relaxed analysis

In the relaxed analysis, the $\overline{\mathbf{act}}_j^{\mathrm{H}}(\theta)$ operation should yield a single star. Thus, we aim to find an enclosing triangle or trapezoid, which is the tightest convex, linear relaxation that we can achieve for hard tanh, as illustrated in Fig. 5.

For bounded inputs, we make a case distinction. If the lower bound (in the bounding box of θ in dimension j, see Proposition 2.5) is less than $V_{\min}$, and the upper bound is between $V_{\min}$ and $V_{\max}$, then the three constraints on the newly introduced variable α_{m+1} are the following: (1) $\alpha_{m+1} \geq x_j$, (2) $\alpha_{m+1} \geq V_{\min}$ and (3) $\alpha_{m+1} \leq \frac{u - V_{\min}}{u - l} \cdot x_j - \frac{u \cdot (l - V_{\min})}{u - l}$. For the opposite case, when the lower bound is between $V_{\min}$ and $V_{\max}$ and the upper bound is greater than $V_{\max}$, we introduce the new variable α_{m+1} and three constraints: (1) $\alpha_{m+1} \leq V_{\max}$, (2) $\alpha_{m+1} \leq x_j$ and (3) $\alpha_{m+1} \geq -\frac{l - V_{\max}}{u - l} \cdot x_j - \frac{l \cdot (V_{\max} - u)}{u - l}$. When the star's domain contains $V_{\min}$ and $V_{\max}$ (i.e., $l < V_{\min} \wedge u > V_{\max}$), we introduce the new variable α_{m+1} and four additional constraints: (1) $\alpha_{m+1} \geq V_{\min}$, (2) $\alpha_{m+1} \leq V_{\max}$, (3) $\alpha_{m+1} \leq \frac{V_{\max} - V_{\min}}{V_{\max} - l} \cdot x_j - \frac{V_{\max} \cdot (l - V_{\min})}{V_{\max} - l}$ and (4) $\alpha_{m+1} \geq \frac{V_{\min} - V_{\max}}{V_{\min} - u} \cdot x_j - \frac{V_{\min} \cdot (V_{\max} - u)}{V_{\min} - u}$.

It is important to highlight that when the bounds of variable x_j fall inside the domain of a single branch of the hard tanh (i.e., they are between $V_{\min}$ and $V_{\max}$; or less than $V_{\min}$; or greater than $V_{\max}$), the result is again a single star and is computed the same way as described in the exact approach.

When dealing with an unbounded input set θ we again distinguish the three cases (1) $x_j \in (-\infty, u)$, (2) $x_j \in (l, \infty)$, and (3) $x_j \in (-\infty, \infty)$. The cases (1) and (2) are further divided into two subcases each, hence we obtain five different cases, each one presented in Table 1, coupled with the corresponding constraints and illustrations.

3.4. Hard sigmoid layer

The hard sigmoid activation function, illustrated in Fig. 6, is a linearized alternative of the sigmoid function. Since the hard sigmoid function has different variants in use (e.g. PyTorch and TensorFlow implementations, see link in footnote for further details[2]) [56–59], we generalize it by adding parameters to cover all those variants.

Definition 3.3 *(Hard sigmoid function).* The *hard sigmoid (HardSigmoid)* function with parameters $V_{\min} \in \mathbb{R}$ and $V_{\max} \in \mathbb{R}_{>V_{\min}}$ is defined for each $x \in \mathbb{R}$ by

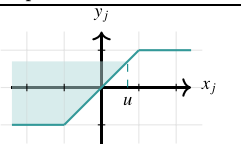
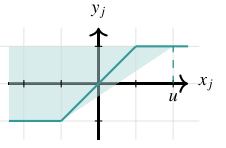
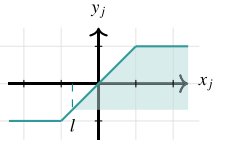
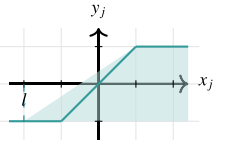
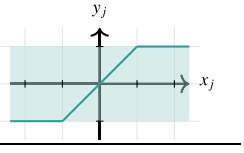
$$HardSigmoid(x) = \begin{cases} 0 & \text{if } x \leq V_{\min} \\ \frac{x}{(V_{\max} - V_{\min})} + \frac{V_{\min}}{(V_{\min} - V_{\max})} & \text{if } V_{\min} < x < V_{\max} \\ 1 & \text{if } x \geq V_{\max} \end{cases}. \tag{5}$$

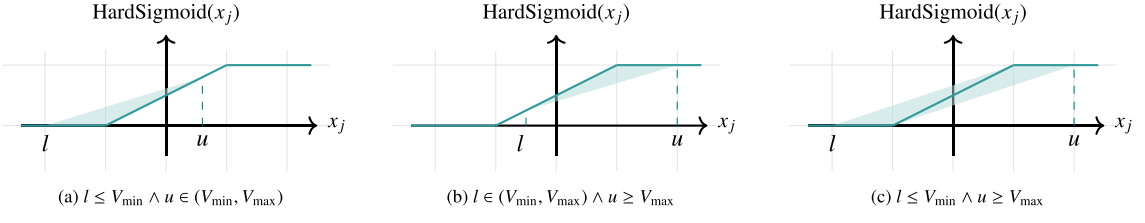
Exact analysis

The analysis of the hard sigmoid works similarly to the one of the hard tanh function. The difference is that instead of the star remaining the same in the range between $V_{\min}$ and $V_{\max}$, we scale the star according to Equation (5). To compute $\mathbf{act}_j^{\mathrm{S}}(\theta)$, the star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ is partitioned into three subsets θ_1, θ_2 and θ_3, covering the partitions with $V_{\min} < x_j < V_{\max}$, $x_j \leq V_{\min}$ and $x_j \geq V_{\max}$, respectively. We scale θ_1 by applying the scaling matrix $\mathbf{M}_{sc} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \frac{1}{V_{\max} - V_{\min}} \mathbf{e}_j, \dots \mathbf{e}_{n-1}, \mathbf{e}_n]$ and shift the center with

[2] Discussion on hard sigmoid implementation: https://stackoverflow.com/questions/35411194/how-is-hard-sigmoid-defined.

Table 1
Approximation rules for the hard tanh function, considering an unbounded input set. We distinguish five cases in total, for each we show the case itself, the introduced constraints and a graphical illustration.

Domain of x_j	Introduced constraints	Graphical illustration
$l = -\infty \wedge u \in [V_{\min}, V_{\max}]$	$\alpha_{m+1} \geq V_{\min}$ $\alpha_{m+1} \geq x_j$ $\alpha_{m+1} \leq u$	
$l = -\infty \wedge u > V_{\max}$	$\alpha_{m+1} \geq V_{\min}$ $\alpha_{m+1} \leq V_{\max}$ $\alpha_{m+1} \geq \frac{V_{\min}-V_{\max}}{V_{\min}-u} \cdot x_j - \frac{V_{\min}\cdot(V_{\max}-u)}{V_{\min}-u}$	
$l \in [V_{\min}, V_{\max}] \wedge u = \infty$	$\alpha_{m+1} \leq V_{\max}$ $\alpha_{m+1} \leq x_j$ $\alpha_{m+1} \geq l$	
$l < V_{\min} \wedge u = \infty$	$\alpha_{m+1} \leq V_{\max}$ $\alpha_{m+1} \geq V_{\min}$ $\alpha_{m+1} \leq \frac{V_{\max}-V_{\min}}{V_{\max}-l} \cdot x_j - \frac{V_{\max}\cdot(l-V_{\min})}{V_{\max}-l}$	
$l = -\infty \wedge u = \infty$	$\alpha_{m+1} \geq V_{\min}$ $\alpha_{m+1} \leq V_{\max}$	



(a) $l \leq V_{\min} \wedge u \in (V_{\min}, V_{\max})$ (b) $l \in (V_{\min}, V_{\max}) \wedge u \geq V_{\max}$ (c) $l \leq V_{\min} \wedge u \geq V_{\max}$

Fig. 6. Relaxation for the hard sigmoid function with a bounded input set, where l and u are the lower respectively upper bounds on the input variable x_j. The dark lines show the exact set (non-convex) and the light area shows the approximate set (convex and linear).

the translation vector defined as $\mathbf{s}_{sc} = [0, \ldots, \frac{V_{\min}}{V_{\min}-V_{\max}}, \ldots, 0]^T$. Furthermore, the elements of θ_2 are set to zero in dimension j by applying the mapping matrix $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{j-1}, \mathbf{0}, \mathbf{e}_{j+1}, \ldots \mathbf{e}_n]$. Finally, the elements of θ_3 are set to one by using the same projection $\mathbf{M}$, plus setting the center to one by the shifting vector $\mathbf{s}_{one} = [0, \ldots, 1, \ldots, 0]^T$. Consequently, the result is the union of three stars: $\mathbf{act}_j^S(\theta) = \langle \mathbf{M}_{sc}\mathbf{c} + \mathbf{s}_{sc}, \mathbf{M}_{sc}\mathbf{V}, P_1 \rangle \cup \langle \mathbf{Mc}, \mathbf{MV}, P_2 \rangle \cup \langle \mathbf{Mc} + \mathbf{s}_{one}, \mathbf{MV}, P_3 \rangle$.

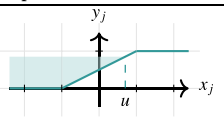
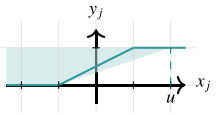
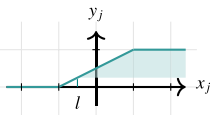
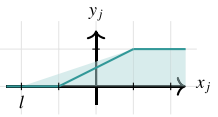
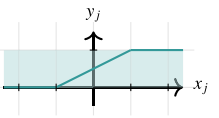
Again, when intersecting the star θ with $V_{\min} < x_j < V_{\max}$, $x_j \leq V_{\min}$ respectively $x_j \geq V_{\max}$, certain resulting subsets may become empty (see Proposition 2.4) and thus their further processing can be omitted.

Relaxed analysis

Using the relaxed analysis for hard sigmoid with bounded input, we consider cases where a convex triangle or trapezoid is applicable based on the input. The $\overline{\mathbf{act}}_j^S(\theta)$ operation introduces a new variable α_{m+1} regardless of which case occurs.

If the lower bound is less than $V_{\min}$ and the upper bound is between $V_{\min}$ and $V_{\max}$, then three new constraints are introduced: (1) $\alpha_{m+1} \geq 0$, (2) $\alpha_{m+1} \geq \frac{1}{V_{\max}-V_{\min}} \cdot x_j - \frac{V_{\min}}{V_{\max}-V_{\min}}$, and (3) $\alpha_{m+1} \leq \frac{u-V_{\min}}{V_{\max}-V_{\min}} \cdot \frac{(x_j-l)}{u-l}$. In the dual scenario when the lower bound is between $V_{\min}$ and $V_{\max}$ while the upper bound exceeds $V_{\max}$, we encounter the constraints: (1) $\alpha_{m+1} \leq 1$, (2) $\alpha_{m+1} \leq \frac{1}{V_{\max}-V_{\min}} \cdot x_j - \frac{V_{\min}}{V_{\max}-V_{\min}}$, and (3) $\alpha_{m+1} \geq 1 + \frac{V_{\max}-l}{V_{\max}-V_{\min}} \cdot \frac{x_j-u}{u-l}$. Lastly, when in dimension j the star is between $V_{\min}$ and $V_{\max}$, then we introduce four constraints: (1) $\alpha_{m+1} \leq 1$, (2) $\alpha_{m+1} \geq 0$, (3) $\alpha_{m+1} \leq \frac{1}{V_{\max}-l} \cdot x_j - \frac{l}{V_{\max}-l}$, and (4) $\alpha_{m+1} \geq \frac{1}{u-V_{\min}} \cdot x_j - \frac{V_{\min}}{u-V_{\min}}$. It is important to highlight that when

Table 2
Approximation rules for hard sigmoid, when the input set is unbounded. We distinguish five cases, for each we show the case itself, the introduced constraints and a graphical illustration.

Domain of x_j	Introduced constraints	Graphical illustration
$l = -\infty \wedge u \in [V_{\min}, V_{\max}]$	$\alpha_{m+1} \geq 0$ $\alpha_{m+1} \geq \frac{x_j}{V_{\max}-V_{\min}} + \frac{V_{\min}}{V_{\min}-V_{\max}}$ $\alpha_{m+1} \leq u$	
$l = -\infty \wedge u > V_{\max}$	$\alpha_{m+1} \geq 0$ $\alpha_{m+1} \leq 1$ $\alpha_{m+1} \geq \frac{x_j - V_{\min}}{u - V_{\min}}$	
$l \in [V_{\min}, V_{\max}] \wedge u = \infty$	$\alpha_{m+1} \leq 1$ $\alpha_{m+1} \leq \frac{x_j}{V_{\max}-V_{\min}} + \frac{V_{\min}}{V_{\min}-V_{\max}}$ $\alpha_{m+1} \geq l$	
$l < V_{\min} \wedge u = \infty$	$\alpha_{m+1} \leq 1$ $\alpha_{m+1} \geq 0$ $\alpha_{m+1} \leq \frac{x_j - l}{V_{\max} - l}$	
$l = -\infty \wedge u = \infty$	$\alpha_{m+1} \geq 0$ $\alpha_{m+1} \leq 1$	

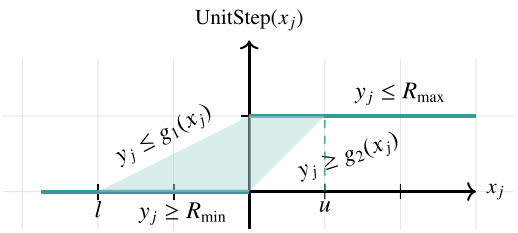


Fig. 7. Relaxation for the unit step function with bounded input sets, where l and u are the lower resp. upper bounds on the input variable x_j. The dark line shows the exact set and the light area the approximate set. The constraints $y_j \leq g_1(x_j)$ and $y_j \geq g_2(x_j)$ correspond to constraints (3) and (4).

the domain of variable x_j is between $V_{\min}$ and $V_{\max}$, less than $V_{\min}$ (i.e., $u \leq V_{\min}$) or greater than $V_{\max}$ (i.e., $l \geq V_{\max}$), the resulting stars remain the same as described in the exact approach.

When dealing with an unbounded input set θ we again distinguish the cases (1) $x_j \in (-\infty, u)$, (2) $x_j \in (l, \infty)$, and (3) $x_j \in (-\infty, \infty)$. The cases (1) and (2) are again divided into two subcases, hence we obtain five different cases, each one presented in Table 2, coupled with the corresponding constraints and illustrations.

3.5. Unit step function layer

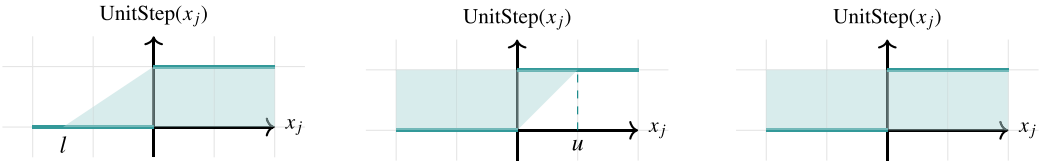
The unit step activation function, also called the Heaviside function and illustrated in Fig. 7, is widely used in neural networks. In this work, we generalize the unit step function, by introducing three parameters with commonly used values $val = 0$, $R_{\min} = 0$, and $R_{\max} = 1$.

Definition 3.4 *(Unit step function [60])*. The *unit step* function with *separator* $val \in \mathbb{R}$, *lower limit* $R_{\min} \in \mathbb{R}$ and *upper limit* $R_{\max} \in \mathbb{R}_{> R_{\min}}$ is defined for each $x \in \mathbb{R}$ by

$$UnitStep(x) = \begin{cases} R_{\min} & \text{if } x < val \\ R_{\max} & \text{if } x \geq val \end{cases}. \tag{6}$$

Exact analysis

The result $\mathbf{act}_j^{\mathrm{U}}(\theta)$ of applying unit step on a star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ in dimension j is obtained as follows. First, θ is decomposed into two parts θ_1 and θ_2 that result from the intersection of θ with $x_j < val$ and $x_j \geq val$. Then, the values in the jth dimension are set to



(a) $x_j \in (l, \infty)$ (b) $x_j \in (-\infty, u)$ (c) $x_j \in (-\infty, \infty)$

Fig. 8. Convex relaxations of the unit step function with three cases of an unbounded input set.

$R_{\min}$ and $R_{\max}$, respectively in the stars θ_1 and θ_2. We achieve this by using the projection matrix $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{j-1}, \mathbf{0}, \mathbf{e}_{j+1}, \dots \mathbf{e}_n]$ and translation vectors $\mathbf{s}_{\min} = [0, \dots, R_{\min}, \dots, 0]^T$ and $\mathbf{s}_{\max} = [0, \dots, R_{\max}, \dots, 0]^T$. The resulting stars are $\langle \mathbf{Mc} + \mathbf{s}_{\min}, \mathbf{MV}, P_1 \rangle$ and $\langle \mathbf{Mc} + \mathbf{s}_{\max}, \mathbf{MV}, P_2 \rangle$.

Note that if the domain $[l, u]$ of x_j does not contain the value val, then the case splitting is not necessary and only one of the stars is the final result, correspondingly to the non-empty intersection with one of the halfspaces.

Relaxed analysis

When the input set is bounded, the relaxed analysis for the unit step function uses a linear, convex trapezoid as shown in Fig. 7, which is again the tightest over-approximation that we can achieve. The $\overline{\mathbf{act}}^{\mathrm{U}}_j(\theta)$ operation also introduces a new variable α_{m+1} and, in this case, four new constraints, which define the trapezoid.

The four constraints are as follows: (1) $\alpha_{m+1} \geq R_{\min}$, (2) $\alpha_{m+1} \leq R_{\max}$, (3) $\alpha_{m+1} \leq \frac{R_{\max} - R_{\min}}{val - l} \cdot x_j + \frac{val \cdot R_{\min} - l \cdot R_{\max}}{val - l}$, and (4) $\alpha_{m+1} \geq \frac{R_{\max} - R_{\min}}{u - val} \cdot x_j + \frac{u \cdot R_{\min} - val \cdot R_{\max}}{u - val}$.

As previously, the result of $\overline{\mathbf{act}}^{\mathrm{U}}_j(\theta)$ is a single star which over-approximates the exact resulting stars. In case the domain of θ in dimension j lies completely in either $(-\infty, val)$ or $[val, \infty)$, then the resulting star is either $\theta_1 = \langle \mathbf{s}_{\min} + \mathbf{Mc}, \mathbf{MV}, P \rangle$ or $\theta_2 = \langle \mathbf{s}_{\max} + \mathbf{Mc}, \mathbf{MV}, P \rangle$, respectively.

Finally, for relaxed analysis with unbounded input stars θ, we again distinguish the cases (1) $x_j \in (-\infty, u)$, (2) $x_j \in (l, \infty)$, and (3) $x_j \in (-\infty, \infty)$. The procedure is similar to the bounded case, but the introduced constraints change as shown in Fig. 8.

3.6. General solution to any piece-wise linear function

In this subsection we present a general solution to any piece-wise linear activation function that is composed from a finite number of linear pieces. The function does not need to be continuous, but we expect that its domain covers the whole set of real numbers (i.e., $f : \mathbb{R} \to \mathbb{R}$).

We consider a finite decomposition of $\mathbb{R}$ into $n \in \mathbb{N}_{\geq 2}$ non-empty disjoint intervals $\mathbb{I}_1, \mathbb{I}_2, \dots, \mathbb{I}_n$, such that $\bigcup_{i=1}^{n} \mathbb{I}_i = \mathbb{R}$ and $\mathbb{I}_i \cap \mathbb{I}_j = \emptyset$ for all $i, j \in \{1, 2, \dots, n\}$ with $i \neq j$. We use a_i and b_i to refer to the lower resp. upper bound of the interval $\mathbb{I}_i$. Each interval might be either open or closed in each direction, but we assume w.l.o.g. that the intervals are ordered according to the order on $\mathbb{R}$: $\mathbb{I}_1$ is unbounded from below (i.e. $\mathbb{I}_1$ is left-open with $a_1 = -\infty$), $\mathbb{I}_n$ is unbounded from above (i.e. $\mathbb{I}_n$ is right-open with $b_n = \infty$), and $b_i = a_{i+1}$ for $i = 1, \dots, n-1$ (note that the property of decomposition implies that $\mathbb{I}_i$ is right-closed if and only if $\mathbb{I}_{i+1}$ is left-open).

A piece-wise linear function is composed of n linear functions, each of which is applied over one of these intervals.

Definition 3.5 *(Piece-wise linear function).* A *piece-wise linear function* is a function of type $f : \mathbb{R} \to \mathbb{R}$ such that there exists a finite decomposition of $\mathbb{R}$ into $n \in \mathbb{N}_{>0}$ non-empty intervals $\mathbb{I}_1, \mathbb{I}_2, \dots, \mathbb{I}_n$ with

$$f(x) = \begin{cases} f_1(x) & \text{if } x \in \mathbb{I}_1 \\ f_2(x) & \text{if } x \in \mathbb{I}_2 \\ \vdots & \vdots \\ f_n(x) & \text{if } x \in \mathbb{I}_n \end{cases} \tag{7}$$

for some linear functions $f_i : \mathbb{I}_i \to \mathbb{R}$, $f_i(x) = m_i \cdot x + s_i$ with $m_i, s_i \in \mathbb{Q}$ for $i = 1, \dots, n$. For any interval $[l, u] \subseteq \mathbb{R}$, we call

$$F = \{(x, f(x))^T \mid x \in [l, u] \subseteq \mathbb{R}\} \tag{8}$$

the *graph* of f over $[l, u]$.

Exact analysis

In order to compute the result of the set operation $\mathbf{act}^{\mathrm{G}}_j(\theta)$ (i.e., the resulting set of stars by considering the exact analysis of a general activation function in the dimension j), corresponding to the piece-wise linear activation function f, we consider each linear piece separately. For an input set $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ having lower and upper bounds l and u in the dimension j (see Proposition 2.5), one needs to apply the following operations.

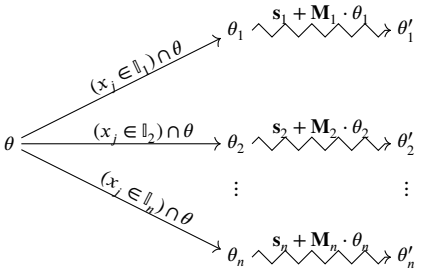


Fig. 9. Illustrating the exact analysis of the input star θ, in a specified dimension j, considering a piece-wise linear activation function f with n pieces.

First, the input star θ needs to be decomposed into n subsets, such that each subset θ_i covers the domain of the corresponding linear piece f_i. Formally, $\theta_i \equiv \theta \cap (x_j \in \mathbb{I}_i)$, considering that $x_j \in \mathbb{I}_i$ can be expressed as the conjunction of two linear constraints, that correspond to the lower bound and upper bound of $\mathbb{I}_i$ (i.e. $a_i \trianglelefteq x_j \trianglelefteq b_i$ and $\trianglelefteq \in \{<, \leq\}$).

In case $[l,u] \cap \mathbb{I}_i = \emptyset$, the corresponding linear piece f_i with its domain $\mathbb{I}_i$ does not intersect the original input star's domain $[l,u]$ in dimension j. Therefore, the result of the intersection θ_i is empty, meaning that θ_i can be discarded.

As a second step, on each non-empty θ_i we have to apply an affine transformation that corresponds to the linear function f_i applied in dimension j. This happens using the linear mapping matrix $\mathbf{M}_i = [\mathbf{e}_1, \mathbf{e}_2, \dots, m_i \cdot \mathbf{e}_j, \dots \mathbf{e}_{n-1}, \mathbf{e}_n]$, that is the identity matrix except column j, which is multiplied by m_i; and translation vector $\mathbf{s}_i = [0, \dots, s_i, \dots, 0]^T$, that is the zero vector, except the jth entry, which is set to s_i. Formally, $\theta'_i = \mathbf{s}_i + \mathbf{M}_i \cdot \theta_i$ and the final result of the operation $\mathbf{act}_j^{\mathrm{G}}(\theta)$ at neuron j can be calculated as $\mathbf{act}_j^{\mathrm{G}}(\theta) = \bigcup_{i=1}^{n} \theta'_i$. This process is illustrated in Fig. 9. Furthermore, as it was mentioned before, having an unbounded input set (i.e., $l = -\infty$ or $u = \infty$) does not have an effect on the exact reachability analysis method.

Relaxed analysis

Let us consider first the case of bounded input sets. We aim at a suitable over-approximation of the transformation of the input set θ under a piece-wise linear activation function f consisting of n pieces, applied in the dimension j. Our over-approximative operation $\overline{\mathbf{act}}_j^{\mathrm{G}}(\theta)$ returns a single star, with one new variable α_{m+1} and at most $2 \cdot n$ new constraints in the predicate (see Proposition Appendix A.6, [77]). Intuitively, we intersect each linear piece of f with the star's domain in dimension j, and collect the endpoints of all resulting segments (that are not empty). The new constraints specify the convex hull of these endpoints, i.e. the smallest convex set that contains for each linear piece its portion within the star's domain in dimension j.

Definition 3.6 *(Convex hull [49]).* The *convex hull* of a finite set $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\} \subseteq \mathbb{R}^d$ of n d-dimensional points is defined as $conv(\mathcal{V}) = \emptyset$ if $\mathcal{V} = \emptyset$, and as

$$conv(\mathcal{V}) = \left\{ \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \;\middle|\; \left(\bigwedge_{i=1}^{n} \lambda_i \in \mathbb{R}_{\geq 0} \right) \wedge \sum_{i=1}^{n} \lambda_i = 1 \right\}$$

otherwise.

In the two-dimensional case, the convex hull of a finite set of points can be easily computed via various algorithms, such as the Graham scan [61], with $\mathcal{O}(n \log n)$ time complexity.

To efficiently define the point set whose convex hull we want to consider, we identify the two indices L and U with $L \leq U$ such that $l \in \mathbb{I}_L$ and $u \in \mathbb{I}_U$. In other words, we identify the two pieces f_L and f_U of f whose domains contain the lower bound l resp. the upper bound u of the star domain. Then, the point set consists of the endpoints of the linear pieces $f_{L+1}, \dots, f_{U-1}$, plus the endpoints of the segments of f_L and f_U that lay in the star domain $[l,u]$, formally defined as:

$$\mathcal{V} = \left\{ (l, f_L(l)), (b_L, f_L(b_L)) \right\} \cup \left\{ (a_i, f_i(a_i)), (b_i, f_i(b_i)) \;\middle|\; L < i < U \right\} \cup \left\{ (a_U, f_U(a_U)), (u, f_U(u)) \right\} \tag{9}$$

The convex hull of $\mathcal{V}$ is the smallest convex set that contains for each of the linear segments $f_L, f_{L+1}, \dots, f_U$ their portion between the lower and upper bounds l and u. Therefore, it gives us the smallest sound over-approximation of the application of f to the input set θ in dimension j. The resulting star $\bar{\theta} = \overline{\mathbf{act}}_j^{\mathrm{G}}(\theta)$ has one new predicate variable α_{m+1} and at most $2 \cdot n$ new linear constraints, which specify the convex hull of $\mathcal{V}$. Fig. 10 illustrates this over-approximation for a piece-wise linear function f.

Proposition 3.1 *(Minimality of the over-approximation in the bounded case). Assume an input star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ with a bounded domain $[l,u]$ in the dimension j, and a piece-wise linear function $f : \mathbb{R} \to \mathbb{R}$ composed from n linear functions $f_i : \mathbb{I}_i \to \mathbb{R}$, $i = 1, \dots, n$. Let F be the graph of f in $[l,u]$ as defined in Eq. (8), and let $\mathcal{V}$ be the finite point set defined for l, u and f in Eq. (9). Then the convex hull $conv(\mathcal{V})$ of $\mathcal{V}$ is the smallest convex superset of F.*

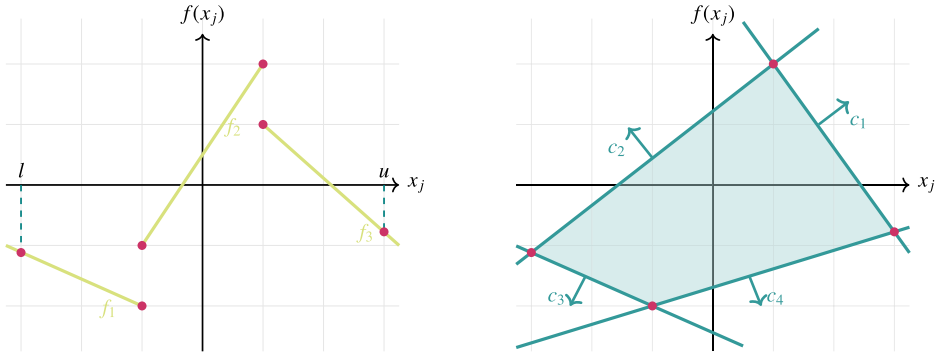


Fig. 10. An example for computing the over-approximation rules for a piece-wise linear activation function f consisting of three linear pieces f_1, f_2 and f_3 and a bounded input set, as illustrated by the maygreen lines on the left. The bounds of the input star in dimension j are l and u. The resulting constraints c_1, c_2, c_3 and c_4, determining the over-approximation area, are illustrated on the right (teal halfplanes), and they are computed by taking the convex hull of endpoints $\mathcal{V}$ (red dots on the left).

Proof. i. We first show that $conv(\mathcal{V})$ is an over-approximation of F. For an input value $x \in [l,u]$, the corresponding point $\mathbf{x} = (x, f(x))^T$ can be expressed as the linear combination of two points from $\mathcal{V}$. Let $i \in \{1,\dots,n\}$ be the index with $x \in \mathbb{I}_i$, and let $a'_i = l$ if $i = L$ and $a'_i = a_i$ otherwise, and analogously $b'_i = u$ if $i = U$ and $b'_i = b_i$ otherwise. Then $x = (1-\lambda)\cdot a'_i + \lambda \cdot b'_i$. Furthermore, $f(x) = (1-\lambda)\cdot f_i(a'_i) + \lambda \cdot f_i(b'_i)$, with $\lambda = \frac{x-a'_i}{b'_i - a'_i}$. Since $a'_i \le x \le b'_i$, clearly $\lambda \in [0,1]$. Thus, the convex hull $conv(\mathcal{V})$ contains all points $(x, f(x))$ for $x \in [l,u]$.

ii. Next we show that $conv(\mathcal{V})$ is the smallest convex over-approximation of F. Let $\mathcal{S}$ be a superset of F, and let $\mathbf{x}^* \in conv(\mathcal{V})$. We show that $\mathbf{x}^* \in \mathcal{S}$. Since $\mathcal{S}$ is a superset of F, it needs to contain all points from $\mathcal{V}$, which are the endpoints of the segments of the graph of f over $[l,u]$. Note that the point $\mathbf{x}^*$ can be expressed as the convex combination of the points in $\mathcal{V}$, i.e. $\mathbf{x}^* = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i$ for some $\lambda_i \in [0,1]$ such that $\sum_{i=1}^{n} \lambda_i = 1$ and $n = |\mathcal{V}|$. Since $\mathcal{S}$ must be convex and contains $\mathcal{V}$, it thus must contain $\mathbf{x}^*$ by the definition of convexity.

From i. and ii. it follows, that the over-approximation $conv(\mathcal{V})$ contains all points from the graph of f over the domain $[l,u]$, and it is the minimal convex set containing these points. □

Once more, if the bounds of the star in dimension j fall inside the domain of a single linear piece f_i, i.e. $[l,u] \subseteq \mathbb{I}_i$, then the result would be a single convex star $\theta' = \mathbf{s}_i + \mathbf{M}_i \cdot \theta$, and it can be computed exactly, therefore there is no need for over-approximating the result.

The unbounded case with $l = -\infty$ or $u = \infty$ is not as straightforward as the bounded case, since we cannot use the convex hull to compute unbounded convex polyhedra. In this case, the over-approximation will be still convex and linear, but also unbounded in negative, positive or both directions, depending on whether the input star θ is unbounded in negative, positive or both directions in dimension j. We first recall some further concepts from geometry.

Definition 3.7 *(Cone and conical hull [62]).* A non-empty set of vectors $C \subseteq \mathbb{R}^d$ is a *cone* if

$$\forall \mathbf{x}, \mathbf{y} \in C.\ \forall \gamma_x, \gamma_y \in \mathbb{R}_{\ge 0}.\ \gamma_x \cdot \mathbf{x} + \gamma_y \cdot \mathbf{y} \in C.$$

For a finite subset $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_k\} \subseteq \mathbb{R}^d$ with k elements, we define the *conical hull*, i.e., $cone(\mathcal{Y})$, as the smallest cone in $\mathbb{R}^d$ that contains $\mathcal{Y} \subseteq \mathbb{R}^d$. If $\mathcal{Y} = \emptyset$, then $cone(\mathcal{Y}) = \{\mathbf{0}\}$. Otherwise, if $\mathcal{Y} \neq \emptyset$, then

$$cone(\mathcal{Y}) = \left\{ \sum_{i=1}^{k} \gamma_i \mathbf{y}_i \;\middle|\; \bigwedge_{i=1}^{k} \gamma_i \in \mathbb{R}_{\ge 0} \right\}.$$

Definition 3.8 *(Minkowski sum of two sets [49]).* The Minkowski sum of two sets $\mathcal{P}, \mathcal{Q} \subseteq \mathbb{R}^d$ is defined as

$$\mathcal{P} \oplus \mathcal{Q} := \left\{ \mathbf{x} + \mathbf{y} \;\middle|\; \mathbf{x} \in \mathcal{P} \wedge \mathbf{y} \in \mathcal{Q} \right\}.$$

Theorem 3.1 *(Minkowski–Weyl theorem [63]). Let $\mathcal{P} \subseteq \mathbb{R}^d$ be a convex polyhedron. Then there exist two finite sets of vectors $\mathcal{V}, C \subseteq \mathbb{R}^d$, such that*

$$\mathcal{P} = conv(\mathcal{V}) \oplus cone(C). \tag{10}$$

Table 3
Enlisting of the elements of the finite endpoints $\mathcal{V}$ and basic vector(s) C of the conical hull, conditioned on the three cases of unboundedness. Note that in the first two cases, since the basis of C consists of a single vector, the obtained cone, i.e. $cone(C)$, is a degenerate cone, consisting of a single half unbounded line (ray). For our formalism, the piece-wise linear activation function f, consists of n pieces, $f_1 = s_1 + m_1 \cdot x_j$ being the first piece and $f_n = s_n + m_n \cdot x_j$ is the last piece.

Bounds of x_j	Elements of $\mathcal{V}$	Basis vectors of C
$l \leq x_j < \infty$	$\{(l, f_L(l)), (b_L, f_L(b_L)) \mid l \in \mathbb{I}_L\} \cup$ $\{(a_i, f_i(a_i)), (b_i, f_i(b_i)) \mid l \in \mathbb{I}_L \wedge L < i < n\} \cup$ $\{(a_n, f_n(a_n))\}$	$\{(1, m_n)^T\}$
$-\infty < x_j \leq u$	$\{(b_1, f_1(b_1))\} \cup$ $\{(a_i, f_i(a_i)), (b_i, f_i(b_i)) \mid u \in \mathbb{I}_U \wedge 1 < i < U\} \cup$ $\{(a_U, f_U(a_U)), (u, f_U(u)) \mid u \in \mathbb{I}_U\}$	$\{(-1, m_1)^T\}$
$-\infty < x_j < \infty$	$\{(b_1, f_1(b_1))\} \cup$ $\{(a_i, f_i(a_i)), (b_i, f_i(b_i)) \mid 1 < i < n\} \cup$ $\{(a_n, f_n(a_n))\}$	$\{(-1, m_1)^T, (1, m_n)^T\}$

An implication of the above theorem is that $\mathcal{P}$ can be written as a finite set of k linear constraints, i.e., for some real matrix $\mathbf{A} \in \mathbb{R}^{k \times d}$ and real vector $\mathbf{b} \in \mathbb{R}^k$, $\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$. To achieve this, considering the finite sets $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ and $C = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$, Equation (10) can be re-written as:

$$\mathcal{P} = \left\{ \sum_{i=1}^{n} \lambda_i \mathbf{v}_i + \sum_{i=1}^{m} \gamma_i \mathbf{d}_i \;\middle|\; \left(\bigwedge_{i=1}^{n} \lambda_i \in \mathbb{R}_{\geq 0} \right) \wedge \sum_{i=1}^{n} \lambda_i = 1 \wedge \left(\bigwedge_{i=1}^{m} \gamma_i \in \mathbb{R}_{\geq 0} \right) \right\}.$$

With the application of quantifier elimination methods [64,65], it is possible to obtain from the above description an equivalent set expressed as a finite conjunction of linear constraints.

For finding the minimal convex polyhedron that defines the over-approximation of the unbounded piece-wise linear activation function f, we compute the Minkowski sum of the convex hull of the set of bounded endpoints $\mathcal{V}$ and the conical hull C constructed from the half-unbounded linear pieces (see Theorem 3.1). Since there could be potentially three subcases of unboundedness, Table 3 summarizes how we obtain $\mathcal{V}$ and C in each subcase. As stated by Theorem 3.1, the obtained result is a convex polyhedron, and it can be written as a set of linear constraints (see Fig. 11). This is the minimal f, as stated in the following proposition.

Proposition 3.2 *(Minimality of our over-approximation in the unbounded case). Assume an input star $\theta = \langle \mathbf{c}, \mathbf{V}, P \rangle$ with an unbounded domain with lower bound l and upper bound u (i.e., $l = -\infty$ or $u = \infty$) in the dimension j, and let f be a piece-wise linear activation function consisting of n pieces. Then the convex polyhedron obtained via Equation (10), with the set of finite endpoints $\mathcal{V}$ and set of direction vectors C as defined in Table 3, is the least conservative over-approximation of the function f over the unbounded domain of θ.*

Proof. i. Assume a fixed but arbitrary input value $l \leq x \leq u$, and let $\mathbf{x} = (x, f(x))^T$. We distinguish two cases. First, if $x \in \mathbb{I}_k$ and $\mathbb{I}_k$ is a bounded interval, then the point is included in the convex hull $conv(\mathcal{V})$ (see Proposition 3.1). Otherwise, if $\mathbb{I}_k$ is an unbounded interval, then we consider the finite bound v of $\mathbb{I}_k$, the point $\mathbf{v} = (v, f_k(v))^T \in \mathcal{V}$ and the corresponding direction vector $\mathbf{d} \in C$. Then, $\mathbf{x} = \mathbf{v} + \gamma \mathbf{d}$, such that $\gamma = \frac{\mathbf{d}^T(\mathbf{x}-\mathbf{v})}{|\mathbf{d}|^2}$, which is a non-negative value. Thus, $conv(\mathcal{V}) \oplus cone(C)$ contains $(x, f(x))^T$.

ii. Let S be a convex and linear over-approximation such that $S \subseteq (conv(\mathcal{V}) \oplus cone(C))$. Furthermore, let $\mathbf{x}^* \in (conv(\mathcal{V}) \oplus cone(C))$. Since S must contain all points from $\mathcal{V}$, and it also contains the points of the unbounded rays with direction vector $\mathbf{d}_i$, then $\mathbf{x}^*$ is expressible as $\mathbf{x}^* = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i + \sum_{k=1}^{m} \gamma_k \cdot \mathbf{d}_k$ for some $\lambda_i \in [0, 1]$ and $\gamma_k \in \mathbb{R}_{>0}$, $n = |\mathcal{V}|$, $m = |C|$. Thus, if S is convex, then $(conv(\mathcal{V}) \oplus cone(C)) \subseteq S$. We conclude that $(conv(\mathcal{V}) \oplus cone(C)) = S$.

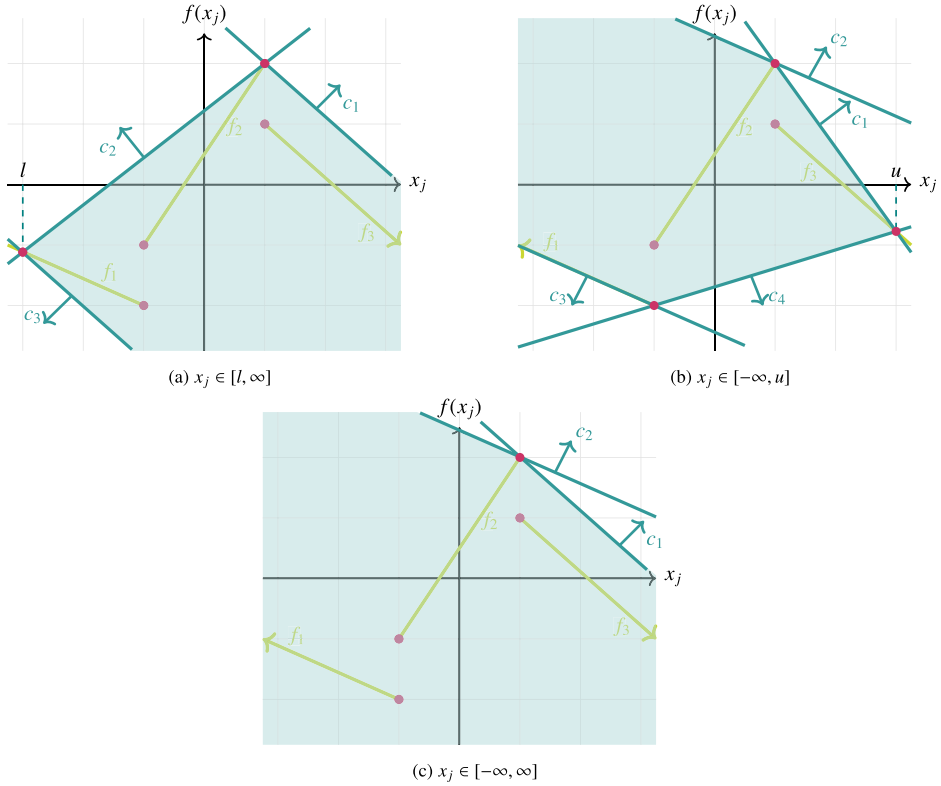
From i. and ii. it follows, that $\mathcal{P} = conv(\mathcal{V}) \oplus cone(C)$ contains the graph of f over θ's domain, and it is the minimal convex set containing these points. □

4. Experimental evaluation

We implemented our proposed algorithms using the open-source C++ tool HyPro [46] and evaluated them on four different benchmark families. We include a brief summary about the networks used in each benchmark in Table 4. The ACAS Xu and drone hovering benchmarks contain only ReLU activations while the thermostat controller and sonar classifier benchmarks use the unit step and hard sigmoid activation functions besides ReLU. Both the exact and the relaxed approaches are evaluated. The evaluations were performed on RWTH Aachen University's HPC Cluster [70] using Rocky Linux 8 as the operating system. Each execution ran on an individual node equipped with 16 GB RAM and two Intel Xeon Platinum 8160 "SkyLake" processors with a total of 16 cores. A 48-hour time limit was set for each experiment.

4.1. ACAS Xu

The Airborne Collision Avoidance System Xu (ACAS Xu) is a mid-air collision avoidance system focusing on unmanned aircrafts. The ACAS Xu networks (ACAS Xu DNNs) provide advisories for horizontal maneuvers to avoid collisions while minimizing unnecessary



(a) $x_j \in [l, \infty]$ (b) $x_j \in [-\infty, u]$

(c) $x_j \in [-\infty, \infty]$

Fig. 11. Visualization of the convex relaxation of a piece-wise linear activation function f, considering three cases of unboundedness. The maygreen lines show the linear pieces f_1, f_2 and f_3. The teal halfplanes are the resulting over-approximation constraints. The red vertices are the endpoints of each (bounded / half bounded) linear piece.

Table 4
Specifics of the four benchmarks used in the experimental results section. Further information about the individual networks can be Found in the corresponding subsections.

ACAS Xu [66]	Drone hovering [67]
• 45 feed-forward, fully-connected neural networks (FFNNs) • 5 input, 5 output neurons • 6 hidden layers, 50 neurons each • ReLU activation after each hidden layer • 10 safety properties	• 8 FFNNs • 12 input, 1 output neurons • number of hidden layers and their size varies (see Table 7) • ReLU activation after each hidden layer • 2 robustness property per network
Thermostat controller [68]	**Sonar binary classifier [69]**
• 1 FFNN • 2 input, 1 output neurons • 2 hidden layers, both with 10 neurons • ReLU activation after hidden layers • UnitStep function after the output neuron • 2 safety properties	• 1 FFNN • 60 input, 1 output neurons • 1 hidden layer with 60 neurons and ReLU activation • HardSigmoid + UnitStep functions after the output neuron • 1 robustness property for each of the 208 binary classification instances

alerts. The ACAS Xu benchmark consists of a set of 45 feedforward neural networks, each with seven fully connected layers, comprising a combined count of 300 neurons. Each network possesses five inputs (see Fig. 12) and five outputs. For further information about the ACAS Xu benchmark see [71,31].

In our experiments, we first compute the reachable set of the networks. Afterward, we check whether the reachable set is fully included in the safe zone. If yes then the FNN is safe, otherwise we can conclude unsafety only for the exact analysis. We check the *safety verification time* (*VT*) in seconds, using the ten safety properties $\phi_1, \phi_2, \ldots, \phi_{10}$ from [66].

According to the condensed results, which are shown in Table 5, we can conclude that the approach with star representation is able to correctly verify the safety properties. We marked with boldface numbers, where the given property could be verified on all the

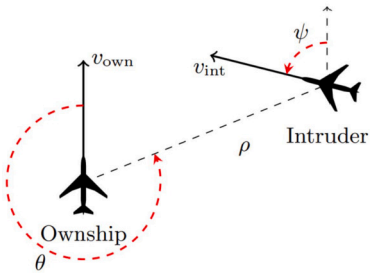


Fig. 12. Vertical view of the inputs of ACAS Xu networks [31].

Table 5
Average verification results for properties $\phi_1, \phi_2, \phi_3, \phi_4$ in seconds. The exact method is executed using parallelization with 8 threads, while the relaxed approach is single threaded.

Prop.	Exact	Overapprox.
	AVG VT(s)	AVG VT(s)
ϕ_1	**35244.9**	2293.4
ϕ_2	44715.5	2316.2
ϕ_3	**279.4**	12.4
ϕ_4	**98.0**	11.4

relevant networks. In case of exact analysis of ϕ_2, the verification results were correct, but in case of 3 networks, timeout occurred. The relaxed analysis could verify correctly only a subset of the networks. We refer to Appendix A.2 for the detailed results, where we show the reachability result and safety verification times of each property and network combinations. A comparison with the NNV tool [22], would be meaningful, but it is implemented in MATLAB and, unfortunately, we do not own a MATLAB license.

4.1.1. Generalized activation function analysis

The popular benchmarks used in neural network verification do not include a wide range of piece-wise linear activation functions, except the more common ones that we presented in Sections 2 and 3. Therefore, to test our implementation of the generalized exact reachability method, we reutilize the ACAS benchmark. We take the 45 FFNNs and interpret each ReLU activation function as a *general* piece-wise linear activation function. Then we use properties ϕ_3 and ϕ_4 of ACAS Xu to compare the reachability algorithms of the generalized analysis and the customized analysis for ReLU presented in Section 2.3.

On all 90 verification instances, the number of stars, variables, and verification results are identical for both approaches, demonstrating the correctness of our exact generalized reachability analysis. Both methods have comparable runtimes, though the customized approach is notably faster for some instances (see Fig. 13). This is likely due to the overhead in computing linear constraints and affine mappings for each linear piece in the generalized method, compared to the precomputed constraints and mappings in the specialized approach.

Moreover, implementing the relaxed reachability analysis for general activation functions would likely exacerbate this runtime disparity, as computing the linear constraints for general piece-wise linear functions is substantially harder to compute than leveraging precomputed constraints as in the customized analysis.

4.1.2. Unbounded analysis

The current state-of-the-art in neural network verification does not address the challenges of verifying networks with unbounded input state sets; i.e., the available, more common benchmarks were designed for bounded analysis. In this subsection, we present an *unbounded* benchmark and the experimental evaluation of our algorithms using this benchmark.

Since verifying adversarial robustness properties of neural networks accounts for *slight* input perturbations, an unbounded input set would not be meaningful for such properties. Nonetheless, the safety properties of the ACAS Xu benchmark are not robustness properties, but rather *functional safety properties*. Moreover, in the definition of some ACAS Xu properties in [66] (for example ϕ_4), the authors did not specify an explicit upper-bound for some input variables. However, in practice these variables are all bounded, the bounds being obtained from the min and max input range parameters of the networks. Therefore, by retaining the original formulation of these ACAS Xu properties, one can obtain unbounded variants of the aforementioned properties.

In our experiments, we use property ϕ_4 of ACAS Xu as defined in [66], and we consider that the variable v_{own} (i.e., the velocity of the ownship) has no upper bound, hence we obtain a five dimensional input set, from which one of the variables has no upper bound. Regarding the safety checking of the output, we do not change the output constraints. We refer to this new unbounded property as $\overline{\phi}_4$. Our experiments that utilize $\overline{\phi}_4$ are listed in Table 6. Each ACAS network $N_{i,j}$ is verified against ϕ_4 and $\overline{\phi}_4$, in order to have comparative measures on how the unboundedness affects our reachability algorithms.

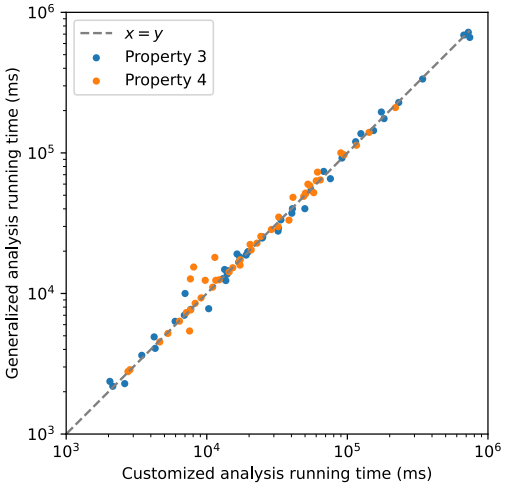


Fig. 13. Exact reachability analysis times of the generalized versus the customized approaches, for 45 ACAS FFNNs and properties ϕ_3 (blue) and ϕ_4 (orange). The gray dashed line is the bisector of the first quadrant. Axes are log-scaled.

Based on the experiment, a general, but also quite expected result is that the unbounded analysis is less efficient (i.e. slower) than the bounded analysis. The reason behind this, is that having an unbounded input set increases the number of case splittings (exact) or number of fresh predicate variables (relaxed), both being essential components of runtime efficiency.

Another key observation regarding the unbounded analysis is that in most cases, the relaxed method is less efficient than the exact method. This is mainly because of the way how an unbounded input set affects the two types of analysis. While at each ReLU node the unbounded relaxed method produces only a single intermediate star, this star is guaranteed to be unbounded, due to the nature of the unbounded relaxation rules. Moreover, at each node where over-approximation happens (i.e., $l < 0 < u$), the algorithm introduces a fresh predicate variable α_{m+1}, which is also unbounded. On the other hand, when working with the exact analysis, at nodes where case splitting happens, the unbounded input set is decomposed into two subsets, but only one of them remains unbounded. That is, at any intermediate neuron, only one of the output stars will be unbounded throughout the analysis.

Lastly, another aspect of the unbounded relaxed analysis is that in our experiments, it did not prove safety for any of the networks, even if the exact unbounded analysis managed to succeed. This is related to the previous observation. While for the exact method, only one star is unbounded and the rest are all bounded, for the relaxed method the resulting star is unbounded, but it is a convex over-approximation of the exact set, hence having very large over-approximation that covers the unbounded exact result and maintains convexity.

We conducted some experiments comparing the increase in the number of stars and number of variables in verifying the bounded ϕ_4 and unbounded $\overline{\phi}_4$, on the network ACAS Xu $N_{3,5}$. The results of these experiments are illustrated in Fig. 14. Our experiments support the fact that the unbounded relaxed analysis has a very large over-approximation error due to the unboundedness of each freshly introduced variable. For the same reason, much more fresh variables are introduced earlier during the relaxed analysis, compared to the bounded relaxed method, thus increasing greatly the running time.

4.2. Drone hovering

Autonomous drone control revolves around launching a drone into the air and enabling it to hover at a desired altitude [67,72]. This benchmark consists of eight neural networks. The first four consist of two, and the other four networks consist of three hidden layers, each followed by a ReLU activation function (see Table 7). For further info about the benchmark, we refer to [73]. We compute the reachability set of the networks as well as the safety verification using our algorithm and measure the reachable set computation time and safety checking time in seconds.

The networks are verified both with the exact and relaxed methods. For each neural network, we test two properties. The presented results in Table 8 show, as we would expect, that the relaxed method is much faster compared to the exact algorithm. However, the exact method verifies almost every property while the relaxed approach fails in all cases (though some were inherently unsafe). This confirms that the relaxed analysis is more scalable and has a smaller computational cost; however, it sacrifices the completeness of the method.

4.3. Thermostat controller

This benchmark mentioned in the Master's thesis [68] maintains the room temperature x between $17\,°C$ and $23\,°C$ using a thermostat. It achieves this by activating (mode on) and deactivating (mode off) the heater based on the sensed temperature. The neural network representing the thermostat's controller is a feedforward neural network with four layers.

Table 6
Analysis of our verification algorithms (exact and relaxed) for bounded and unbounded input sets. Each ACAS Xu instance is denoted as $N_{i,j}$. Verified properties include the original ϕ_4 (odd rows) and the unbounded $\overline{\phi}_4$ (even rows). #Set: number of star-sets in exact analysis (=1 in relaxed analysis); #Var: number of variables in the predicates (=5 in exact analysis); RT, CT, VT: reachable set computation, safety checking, and total times (in seconds). Res indicates the verification outcome: ✓ for verified safety, × when safety could not be verified. The '-' symbol indicates that the reachable set computation time exceeded the 30-minute limit.

Net	Exact (8 threads)					Over-approximative (1 thread)				
	#Set	RT (s)	Res	CT (s)	VT (s)	#Var	RT (s)	Res	CT (s)	VT (s)
$N_{1,2}$	13143	161	✓	22	183	81	14	×	3	17
	55061	821	✓	137	958	254	439	×	7	447
$N_{1,3}$	9837	131	✓	11	143	84	17	×	1	18
	95003	1281	×	248	1530	-	-	-	-	-
$N_{1,6}$	4443	48	✓	2	50	60	7	✓	0	7
	23206	296	×	23	319	246	689	×	7	697
$N_{1,7}$	642	4	×	0	5	33	1	×	0	1
	1821	19	×	4	23	229	280	×	4	284
$N_{2,1}$	5066	48	✓	3	52	74	8	×	1	10
	52478	633	✓	71	705	252	795	×	7	803
$N_{2,4}$	913	12	✓	0	12	57	8	✓	0	8
	16687	179	✓	10	190	237	616	×	7	624
$N_{2,6}$	1462	17	✓	0	17	62	11	✓	0	11
	4353	77	✓	2	79	-	-	-	-	-
$N_{2,7}$	555	5	✓	0	5	41	4	✓	0	4
	2282	34	×	1	35	250	1638	×	35	1673
$N_{2,8}$	1805	54	✓	0	55	102	33	×	0	34
	3322	78	×	1	80	-	-	-	-	-
$N_{3,2}$	8708	85	✓	9	95	60	3	✓	0	4
	17637	193	×	21	214	231	855	×	8	863
$N_{3,5}$	3630	45	✓	3	49	82	22	✓	0	23
	6444	96	✓	6	102	238	456	×	17	473
$N_{3,6}$	1495	28	✓	1	29	66	11	✓	0	12
	7266	158	✓	4	163	-	-	-	-	-
$N_{3,8}$	542	8	✓	0	8	54	6	×	0	7
	1795	39	✓	2	41	241	1076	×	6	1083

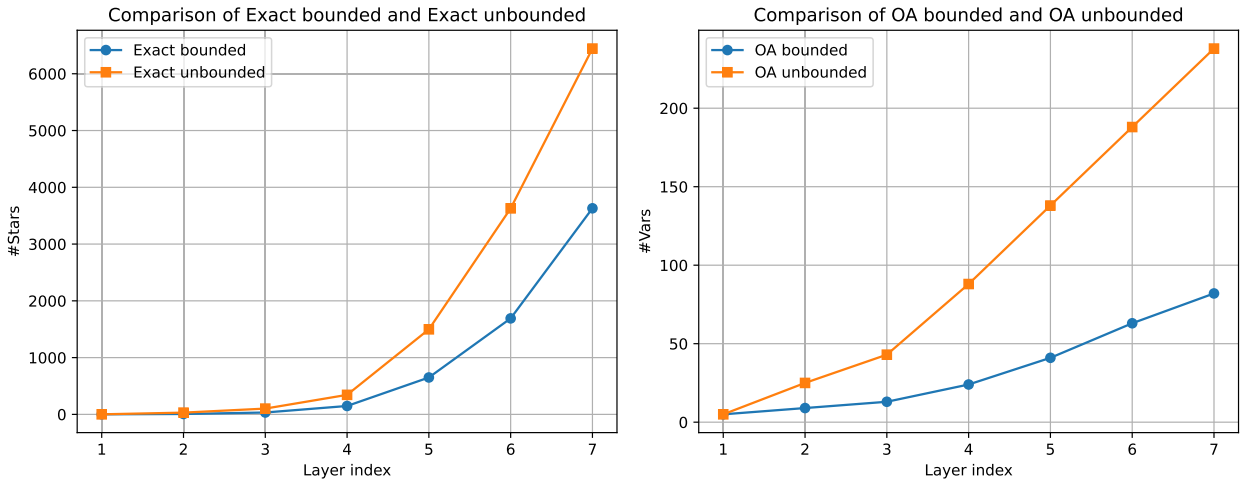
Table 7
Architecture of the networks used in the drone hovering benchmark.

Architecture	Network ID	Neurons
Two layers	AC1	32, 16
	AC2	64, 32
	AC3	128, 64
	AC4	256, 128
Three layers	AC5	32, 16, 8
	AC6	64, 32, 16
	AC7	128, 64, 32
	AC8	256, 128, 64

The input consists of two neurons that express the temperature $x \in \mathbb{R}$ and the current mode (off or on) as $m \in \{0, 1\}$. Furthermore, two hidden layers follow, each with ten neurons. Lastly, using the unit step activation function, the output layer predicts whether the heater should turn on or off, producing the control output $Kh = 15$ or $Kh = 0$, respectively. We compute the exact and over-approximated reachable sets to verify the safety of the described NN properties using our algorithms.

On the thermostat controller benchmark, we formalize and verify two safety properties:

- $\mathcal{P}_1$: the input temperature being between 22° and 23° (i.e., $22 \leq x \leq 23$), and the heater being turned on ($m = 1$), the control output should be the turn off signal ($Kh = 0$).
- $\mathcal{P}_2$: the input temperature being lower than 17° (i.e., $x \leq 17$), and the heater being turned off ($m = 0$), the control output should be the turn on signal ($Kh = 15$). Note that this verification instance is unbounded, since input x has no lower bound.



(a) Growth of the number of stars in the exact analysis.

(b) Growth of the number of variables in relaxed analysis.

Fig. 14. Illustrating the growth of the number of stars and number of variables in the exact respectively relaxed (i.e. over-approximate) methods. For the experiment the ACAS Xu $N_{3,5}$ network was used, combined with properties ϕ_4 and $\overline{\phi}_4$ that provide bounded and unbounded input sets. Observe that in the case of relaxed analysis of $\overline{\phi}_4$, from layer 4 on, the number of variables grows linearly, with 50 new variables after each layer. This means, that all the output sets of each neuron were over-approximated, increasing the number of fresh variables by one with each neuron. This indicates that the bounds of the intermediate stars became very loose, being always $lb < 0$ and $ub > 0$.

Table 8
Evaluation results of the drones benchmark. The network is identified as ACx, the lower-right index y shows the tested property. *RT* is the reachable set computation time, and *CT* is the safety checking time, both in seconds. *RES* is the safety verification result. ✓ indicates that the given neural network was verified to be safe, with respect to the property. Conversely, × means that the computed reachability set intersects with the unsafe set, either due to the network being inherently unsafe or because of over-approximation error (only in the relaxed case). Cells with '-' indicate cases where timeout occurs.

ACx_y	Exact (8 threads)			Overapprox. (1 thread)		
	RT(s)	Res	CT(s)	RT(s)	Res	CT(s)
$AC1_1$	61.4	✓	4.9	0.2	×	0.0
$AC1_2$	0.5	✓	0.0	0.1	×	0.0
$AC2_1$	462.4	✓	17.7	0.5	×	0.0
$AC2_2$	0.1	✓	0.0	0.1	×	0.0
$AC3_2$	5.1	✓	0.1	0.2	×	0.0
$AC4_2$	103.0	✓	5.5	0.7	×	0.0
$AC5_1$	304.8	✓	26.1	0.4	×	0.1
$AC6_1$	2631.7	✓	84.4	0.7	×	0.1
$AC7_2$	4.1	✓	0.1	0.3	×	0.0
$AC5_2$	0.1	×	0.0	0.1	×	0.0
$AC6_2$	0.1	×	0.0	0.1	×	0.0
$AC8_2$	0.8	×	0.0	0.6	×	0.0
$AC3_1$	-	-	-	2.9	×	0.4
$AC4_1$	-	-	-	8.7	×	1.5
$AC7_1$	-	-	-	2.5	×	0.1
$AC8_1$	-	-	-	65.9	×	19.8

We tested both the exact and relaxed algorithms using the two safety properties of the thermostat controller benchmark. The results are summarized in Table 9.

For the first safety property, the exact method correctly verifies that this instance is unsafe, thus we can construct the complete counter input set as explained in Theorem 2 of [44]. From the complete counter input set, we sample a single counter-example candidate, and evaluating the network on this counter-example, we obtain the unsafe output $Kh = 15$. On the contrary, using the relaxed method one cannot prove that this instance is unsafe, since the reachable set might intersect the unsafe zone only due to over-approximation error.

Due to the second safety property being unbounded, the number of final stars, predicate variables, and thus the reachable set computation times are higher than for the first property. However, in this case both the exact and relaxed methods can correctly verify

Table 9
Summary of the experiments conducted on the thermostat controller benchmark. Prop. indicates the analyzed property (the bounded $\mathcal{P}_1$ or the unbounded $\mathcal{P}_2$). Both analyzed with the exact and relaxed methods respectively. #Star and #Var show the number of final stars and the number of variables in their predicates. RT is the reachability time measured in milliseconds, while Result shows the verification result (✓ indicates that safety was verified, × indicates that safety could not be verified). Lastly, in case of unsafe instances, when exact analysis was used, we constructed the complete counter set, we sample a single counter-example, and we evaluated the network, to prove unsafety. The construction of the counter set was not possible for the other instances.

Prop.	Method	#Star	#Var	RT (ms)	Result	Counter Set	Adv. Test
$\mathcal{P}_1$	Exact	4	2	32	×	✓	✓
	Overapprox.	1	5	21	×	×	×
$\mathcal{P}_2$	Exact	14	2	77	✓	×	×
	Overapprox.	1	13	111	✓	×	×

Table 10
Local adversarial robustness tests of the exact approach. *RT* is the reachable set computation time in milliseconds. *RES* is the safety verification result. ✓ indicates that the neural network correctly classifies the input set, while × means that the network was unable to correctly classify the input set.

	$\delta = 0.01$		$\delta = 0.001$		$\delta = 0.0001$		$\delta = 0.00001$	
	RT	RES	RT	RES	RT	RES	RT	RES
Set 1	4359	×	783	✓	263	✓	102	✓
Set 2	206243	×	1284	✓	245	✓	100	✓
Set 3	33945	✓	3768	✓	401	✓	308	✓
Set 4	7974	✓	359	✓	103	✓	102	✓

Table 11
Local adversarial robustness tests of the relaxed approach. *RT* is the reachable set computation time in milliseconds. *RES* is the safety verification result. ✓ indicates that the neural network correctly classifies the input set, while × is assigned when the reachability analysis algorithm cannot provide a conclusive answer due to the over-approximation errors.

	$\delta = 0.01$		$\delta = 0.001$		$\delta = 0.0001$		$\delta = 0.00001$	
	RT	RES	RT	RES	RT	RES	RT	RES
Set 1	234	×	205	✓	163	✓	103	✓
Set 2	396	×	279	✓	157	✓	103	✓
Set 3	407	×	367	✓	177	✓	174	✓
Set 4	339	✓	167	✓	104	✓	101	✓

safety of this instance. Since the instance is truly safe, the construction of counter set is not possible. Lastly, the same phenomenon can be observed here, as at the unbounded ACAS Xu property, that the unbounded relaxed analysis is less efficient than the unbounded exact analysis.

4.4. Sonar binary classifier

In this section, we evaluate the robustness of a neural network used for binary classification of a sonar dataset. This dataset describes sonar chirp returns bouncing off from different objects [69]. It contains 60 input variables representing the returned beams' strength at different angles. The verified neural network should be capable of robust binary classification, distinguishing between rocks and metal cylinders. The neural network consists of one hidden layer with 60 neurons, followed by a ReLU activation and an output layer with a single neuron, followed by the composition of a hard sigmoid and a unit step activation function. The property we want to verify is the local robustness of the neural network. A neural network is *δ-locally-robust* at input x, if for every x' such that $\|x - x'\|_\infty \leq \delta$, the network assigns the same output label to x and x'. Our focus lies in determining the robustness threshold that our verification method can provide for the network (i.e., finding the largest δ for which the robustness property still holds).

We examine this problem on four input sets, each one being a δ-cube around a specific element of the dataset. We try four different values of δ. The first two inputs should output 1, which means a rock, and the next two 0, which means a metal cylinder. The ✓ symbol indicates correct classifications within the robustness threshold ($\forall x'$ being correctly classified), × denotes incorrect predictions. A comparison between the exact and the relaxed algorithms reveals that the exact algorithm proves network robustness in more cases. Furthermore, different input sets (meaning a single input and its δ neighborhood) exhibit varying local robustness. For example, in Table 10, for Set 2, the optimal δ value is between 0.01 and 0.001. Tables 10 and 11 are condensed versions of our experiments; the complete results are listed in Appendix A.3.

5. Conclusion

In this paper, we proposed algorithms for star-based reachability analysis of various activation functions used in feed-forward neural networks. To further support generality, we implemented the specific activation functions with flexibility for adaptation to different use cases. Moreover, we presented the formalisms for the analysis of an arbitrary piece-wise linear activation function. We proved that our relaxations are sound and the tightest possible convex over-approximation, that can be achieved considering only a single neuron output.

We implemented an NNET parser in HyPro to simplify the incorporation of additional benchmarks. The presented evaluation results offer valuable insights into network behavior and safety. Our experiments demonstrate that while the relaxed analysis, in contrast to the exact analysis, is incomplete, it is much more scalable in the bounded case. On the contrary, considering unbounded input sets, the exact analysis proved to be more efficient.

In future work, we plan to provide an implementation of the relaxed analysis of general piece-wise linear activation functions, discussing its practical utility via experiments on some benchmarks. We plan to integrate further layer types, and a more widely-used standard such as ONNX [76], for storing and parsing neural network inputs. Moreover, comprehensive experiments and evaluations will offer deeper insights into the performance, accuracy, and limitations of this analysis method when applied to neural networks with other activation functions and layer types, hence, exploring its effectiveness on a more realistic and diverse scale of benchmarks.

Finally, we are planning to adapt abstraction refinement techniques (such as CEGAR), to reduce the over-approximation error during the reachable set computation. This can greatly enhance the practical utility of the relaxed analysis of unbounded input sets, cutting off the over-approximation error. Moreover, CEGAR will leverage both exact and over-approximative computations, offering a good trade-off between runtime efficiency and small over-approximation error, making our verification methods more scalable to larger networks that were previously challenging to verify.

CRediT authorship contribution statement

László Antal: Writing – review & editing, Writing – original draft, Visualization, Methodology, Data curation, Conceptualization. **Erika Ábrahám:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Hana Masara:** Writing – review & editing, Writing – original draft, Visualization, Methodology, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We are grateful to Dario Guidotti, Stefano Demarchi, and Armando Tacchella for generously sharing with us their drone hovering benchmark. This project has received funding from the European Union's Horizon 2020 programme under the Skłodowska-Curie grant agreement No. 956200. For more information, please visit https://remaro.eu.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.scico.2025.103269.

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, https://doi.org/10.1038/nature14539.
[2] W. Samek, G. Montavon, S. Lapuschkin, C.J. Anders, K.-R. Müller, Explaining deep neural networks and beyond: a review of methods and applications, Proc. IEEE 109 (3) (2021) 247–278, https://doi.org/10.1109/JPROC.2021.3060483.
[3] S. Kuutti, R. Bowden, Y. Jin, P. Barber, S. Fallah, A survey of deep learning applications to autonomous vehicle control, IEEE Trans. Intell. Transp. Syst. 22 (2) (2021) 712–733, https://doi.org/10.1109/TITS.2019.2962338.
[4] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97, https://doi.org/10.1109/MSP.2012.2205597.
[5] D. Erhan, C. Szegedy, A. Toshev, D. Anguelov, Scalable object detection using deep neural networks, in: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), IEEE, 2014, pp. 2155–2162.
[6] A. Lee, Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics, Swarthmore University, 2015, https://api.semanticscholar.org/CorpusID:10011895.
[7] E.D. Cubuk, B. Zoph, S.S. Schoenholz, Q.V. Le, Intriguing properties of adversarial examples, in: Proceedings of the 6th International Conference on Learning Representations (ICLR 2018), 2018, pp. 1–17, https://openreview.net/, https://openreview.net/forum?id=Skz1zaRLz.
[8] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, CoRR, arXiv:1412.6572 [abs], 2014, https://api.semanticscholar.org/CorpusID:6706414.
[9] X. Huang, M. Kwiatkowska, S. Wang, M. Wu, Safety verification of deep neural networks, in: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017), Springer, 2017, pp. 3–29.

[10] J. Woodcock, P.G. Larsen, J. Bicarregui, J. Fitzgerald, Formal methods: practice and experience, ACM Comput. Surv. 41 (4) (2009) 1–36, https://doi.org/10.1145/1592434.1592436.
[11] E.M. Clarke, J.M. Wing, Formal methods: state of the art and future directions, ACM Comput. Surv. 28 (4) (1996) 626–643, https://doi.org/10.1145/242223.242257.
[12] M. Hinchey, J. Bowen, C. Rouff, Introduction to Formal Methods, Springer, 2006.
[13] J.M. Wing, A specifier's introduction to formal methods, Computer 23 (9) (1990) 8–22, https://doi.org/10.1109/2.58215.
[14] L. Pulina, A. Tacchella, An abstraction-refinement approach to verification of artificial neural networks, in: Proceedings of the 22nd International Conference on Computer Aided Verification (CAV 2010), in: LNCS, vol. 6174, Springer, 2010, pp. 243–257.
[15] H.-D. Tran, P. Musau, D.M. Lopez, X. Yang, L.V. Nguyen, W. Xiang, T.T. Johnson, Parallelizable reachability analysis algorithms for feed-forward neural networks, in: Proceedings of the 7th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE 2019), IEEE, 2019, pp. 51–60.
[16] W. Xiang, H.-D. Tran, T.T. Johnson, Output reachable set estimation and verification for multilayer neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (11) (2018) 5777–5783, https://doi.org/10.1109/TNNLS.2018.2808470.
[17] R. Ehlers, Formal verification of piece-wise linear feed-forward neural networks, in: Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA 2017), Springer, 2017, pp. 269–286.
[18] C.-H. Cheng, G. Nührenberg, H. Ruess, Maximum resilience of artificial neural networks, in: Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA 2017), Springer, 2017, pp. 251–268.
[19] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana, Efficient formal safety analysis of neural networks, in: Proceedings of the 31st Annual Conference on Advances in Neural Information Processing Systems (NeurIPS 2018), Curran Associates Inc., 2018, pp. 6369–6379, https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html.
[20] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M.J. Kochenderfer, Algorithms for verifying deep neural networks, Found. Trends Optim. 4 (3–4) (2021) 244–404, https://doi.org/10.1561/2400000035.
[21] A. Fromherz, K. Leino, M. Fredrikson, B. Parno, C.S. Pasareanu, Fast geometric projections for local robustness certification, in: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021), 2021, pp. 1–15, https://openreview.net/, https://openreview.net/forum?id=zWy1uxjDdZJ.
[22] H.-D. Tran, N. Pal, D.M. Lopez, P. Musau, X. Yang, L.V. Nguyen, W. Xiang, S. Bak, T.T. Johnson, Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter, Form. Asp. Comput. 33 (4) (2021) 519–545, https://doi.org/10.1007/s00165-021-00553-4.
[23] A. Boopathy, T. Weng, P. Chen, S. Liu, L. Daniel, CNN-Cert: an efficient framework for certifying robustness of convolutional neural networks, in: Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019), AAAI Press, 2019, pp. 3240–3247.
[24] P. Henriksen, A. Lomuscio, DEEPSPLIT: an efficient splitting method for neural network verification via indirect effect analysis, in: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021), 2021, pp. 2549–2555, https://www.ijcai.org/.
[25] G. Katz, D.A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D.L. Dill, M.J. Kochenderfer, C.W. Barrett, The Marabou framework for verification and analysis of deep neural networks, in: Proceedings of the 31st International Conference on Computer Aided Verification (CAV 2019), in: LNCS, vol. 11561, Springer, 2019, pp. 443–452.
[26] B. Sun, J. Sun, T. Dai, L. Zhang, Probabilistic verification of neural networks against group fairness, in: Proceedings of the 24th International Symposium on Formal Methods (FM 2021), in: LNCS, vol. 13047, Springer, 2021, pp. 83–102.
[27] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. Patel, J.P. Vielma, The convex relaxation barrier, revisited: tightened single-neuron relaxations for neural network verification, CoRR, arXiv:2006.14076 [abs], 2020, arXiv:2006.14076, https://arxiv.org/abs/2006.14076.
[28] M. Fischetti, J. Jo, Deep neural networks as 0-1 mixed integer linear programs: a feasibility study, CoRR, arXiv:1712.06174 [abs], 2017, arXiv:1712.06174, http://arxiv.org/abs/1712.06174.
[29] R. Bunel, J. Lu, I. Turkaslan, P.H.S. Torr, P. Kohli, M.P. Kumar, Branch and bound for piecewise linear neural network verification, CoRR, arXiv:1909.06588 [abs], 2019, arXiv:1909.06588, http://arxiv.org/abs/1909.06588.
[30] V. Tjeng, R. Tedrake, Verifying neural networks with mixed integer programming, CoRR, arXiv:1711.07356 [abs], 2017, arXiv:1711.07356, http://arxiv.org/abs/1711.07356.
[31] G. Katz, C.W. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer, Reluplex: an efficient SMT solver for verifying deep neural networks, in: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017), in: LNCS, vol. 10426, Springer, 2017, pp. 97–117.
[32] H. Wu, A. Zeljić, G. Katz, C. Barrett, Efficient neural network analysis with sum-of-infeasibilities, in: Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022), in: LNCS, vol. 13243, Springer, 2022, pp. 143–163.
[33] C. Huang, J. Fan, W. Li, X. Chen, Q. Zhu, ReachNN: reachability analysis of neural-network controlled systems, ACM Trans. Embed. Comput. Syst. 18 (5s) (2019) 1–22, https://doi.org/10.1145/3358228.
[34] W. Ruan, X. Huang, M. Kwiatkowska, Reachability analysis of deep neural networks with provable guarantees, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18), International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 2651–2659.
[35] A. Lomuscio, L. Maganti, An approach to reachability analysis for feed-forward ReLU neural networks, CoRR, arXiv:1706.07351 [abs], 2017, http://arxiv.org/abs/1706.07351.
[36] P. Kern, M.K. Büning, C. Sinz, Optimized symbolic interval propagation for neural network verification, CoRR, arXiv:2212.08567 [abs], 2022, https://doi.org/10.48550/ARXIV.2212.08567.
[37] H. Salman, G. Yang, H. Zhang, C. Hsieh, P. Zhang, A convex relaxation barrier to tight robustness verification of neural networks, CoRR, arXiv:1902.08722 [abs], 2019, arXiv:1902.08722, http://arxiv.org/abs/1902.08722.
[38] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, M. Vechev, Ai2: safety and robustness certification of neural networks with abstract interpretation, in: 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 3–18.
[39] G. Singh, T. Gehr, M. Mirman, M. Püschel, M. Vechev, Fast and effective robustness certification, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018, https://proceedings.neurips.cc/paper_files/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf.
[40] G. Singh, T. Gehr, M. Püschel, M. Vechev, An abstract domain for certifying neural networks, Proc. ACM Program. Lang. 3 (POPL) (Jan. 2019), https://doi.org/10.1145/3290354.
[41] C. Ferrari, M.N. Mueller, N. Jovanović, M. Vechev, Complete verification via multi-neuron relaxation guided branch-and-bound, in: International Conference on Learning Representations, 2022, https://openreview.net/forum?id=l_amHf1oaK.
[42] G. Singh, R. Ganvir, M. Püschel, M. Vechev, Beyond the single neuron convex barrier for neural network certification, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019, https://proceedings.neurips.cc/paper_files/paper/2019/file/0a9fdbb17feb6ccb7ec405cfb85222c4-Paper.pdf.
[43] S. Bak, P.S. Duggirala, Simulation-equivalent reachability of large linear systems with inputs, in: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017), in: LNCS, vol. 10426, Springer, 2017, pp. 401–420.
[44] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L.V. Nguyen, W. Xiang, T.T. Johnson, Star-based reachability analysis of deep neural networks, in: Formal Methods – The Next 30 Years, Springer, 2019, pp. 670–686.
[45] L. Antal, H. Masara, E. Ábrahám, Extending neural network verification to a larger family of piece-wise linear activation functions, in: Proceedings of the 5th International Workshop on Formal Methods for Autonomous Systems (FMAS@iFM 2023), in: EPTCS, vol. 395, 2023, pp. 30–68.

[46] S. Schupp, E. Ábrahám, I. Makhlouf, S. Kowalewski, HyPro: a C++ library of state set representations for hybrid systems reachability analysis, in: Proceedings of the 9th International Symposium NASA Formal Methods (NFM 2017), in: LNCS, vol. 10227, Springer, 2017, pp. 288–294.
[47] N. Kumar, Deep learning: feedforward neural networks explained, https://medium.com/hackernoon/deep-learning-feedforward-neural-networks-explained-c34ae3f084f1, 2019. (Accessed 3 May 2023).
[48] D. Svozil, V. Kvasnicka, J. Pospichal, Introduction to multi-layer feed-forward neural networks, Chemom. Intell. Lab. Syst. 39 (1) (1997) 43–62, https://doi.org/10.1016/S0169-7439(97)00061-0.
[49] S. Schupp, State set representations and their usage in the reachability analysis of hybrid systems, Ph.D. thesis, RWTH Aachen University, 2019.
[50] G. Singh, T. Gehr, M. Püschel, M.T. Vechev, An abstract domain for certifying neural networks, Proc. ACM Program. Lang. 3 (POPL) (2019) 41:1–41:30, https://doi.org/10.1145/3290354.
[51] L. Datta, A survey on activation functions and their relation with Xavier and He normal initialization, CoRR, arXiv:2004.06632 [abs], 2020, https://arxiv.org/abs/2004.06632.
[52] L. Ramadhan, Neural network: the dead neuron, https://towardsdatascience.com/neural-network-the-dead-neuron-eaa92e575748, 2021. (Accessed 14 May 2023).
[53] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proceedings of the 30th International Conference on Machine Learning, 2013, pp. 1–6, https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.
[54] J. Xu, Z. Li, B. Du, M. Zhang, J. Liu, Reluplex made more practical: leaky ReLU, in: 2020 IEEE Symposium on Computers and Communications (ISCC 2020), IEEE, 2020, pp. 1–7.
[55] R. Collobert, Large scale machine learning, Ph.D. thesis, Université de Paris VI, 2004, https://ronan.collobert.com/pub/2004_phdthesis_lip6.pdf.
[56] M. Courbariaux, Y. Bengio, Binarynet: training deep neural networks with weights and activations constrained to +1 or -1, CoRR, arXiv:1602.02830 [abs], 2016, arXiv:1602.02830, http://arxiv.org/abs/1602.02830.
[57] Tensorflow documentation, https://www.tensorflow.org/api_docs/python/tf/keras/activations/hard_sigmoid, 2023. (Accessed 16 May 2023).
[58] S.P. Anthadupula, M. Gyanchandani, A review and performance analysis of non-linear activation functions in deep neural networks, in: International Research Journal of Modernization in Engineering Technology and Science (IRJMETS), 2021.
[59] PyTorch: torch.nn.hardsigmoid, https://pytorch.org/docs/stable/generated/torch.nn.Hardsigmoid.html, 2021. (Accessed 16 May 2023).
[60] Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2008), LNCS, vol. 5072, Springer, 2008.
[61] R.L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Inf. Process. Lett. 1 (4) (1972) 132–133, https://doi.org/10.1016/0020-0190(72)90045-2.
[62] G.M. Ziegler, Lectures on Polytopes, Graduate Texts in Mathematics, vol. 152, Springer, 2012.
[63] A. Basu, Introduction to convexity, https://www.ams.jhu.edu/~abasu9/AMS_550-465/notes-without-frills.pdf, 2019.
[64] G.B. Dantzig, B.C. Eaves, Fourier-Motzkin elimination and its dual, J. Comb. Theory, Ser. A 14 (3) (1973) 288–297, https://doi.org/10.1016/0097-3165(73)90004-6.
[65] J. Nalbach, V. Promies, E. Ábrahám, P. Kobialka, FMplex: a novel method for solving linear real arithmetic problems 390 (2023) 16–32, https://doi.org/10.4204/EPTCS.390.2.
[66] G. Katz, C.W. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer, Reluplex: an efficient SMT solver for verifying deep neural networks, CoRR, arXiv:1702.01135 [abs], 2017, arXiv:1702.01135, http://arxiv.org/abs/1702.01135.
[67] D. Guidotti, S. Demarchi, L. Pulina, A. Tacchella, Evaluating reachability algorithms for neural networks on NeVer2, https://www.researchgate.net/publication/363845518_Evaluating_Reachability_Algorithms_for_Neural_Networks_on_NeVer2, 2022.
[68] R.G. Jiang, Verifying AI-controlled hybrid systems, Master's thesis, RWTH Aachen University, Aachen, Germany, 2023, available at https://ths.rwth-aachen.de/wp-content/uploads/sites/4/master_thesis_jiang.pdf.
[69] J. Brownlee, Binary classification tutorial with the keras deep learning library, https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library, 2022. (Accessed 1 June 2023).
[70] R. A. University, RWTH high performance computing (Linux), https://help.itc.rwth-aachen.de/service/rhr4fjjutttf/, 2023. (Accessed 24 July 2023).
[71] K.D. Julian, M.J. Kochenderfer, M.P. Owen, Deep neural network compression for aircraft collision avoidance systems, J. Guid. Control Dyn. 42 (3) (2019) 598–608, https://doi.org/10.2514/1.g003724.
[72] D. Guidotti, Verification of neural networks for safety and security-critical domains, in: R.D. Benedictis, N. Gatti, M. Maratea, A. Micheli, A. Murano, E. Scala, L. Serafini, I. Serina, A. Umbrico, M. Vallati (Eds.), Proceedings of the 10th Italian Workshop on Planning and Scheduling (IPS 2022), RCRA Incontri e Confronti (RiCeRcA 2022), and the Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (SPIRIT 2022), in: CEUR Workshop Proceedings, vol. 3345, 2022, pp. 1–10, https://ceur-ws.org/, https://ceur-ws.org/Vol-3345/paper10_RiCeRCa3.pdf.
[73] H. Masara, Star set-based reachability analysis of neural networks with differing layers and activation functions, Bachelor's thesis, RWTH Aachen University, 52074 Aachen, Germany, 2023, available at https://ths.rwth-aachen.de/wp-content/uploads/sites/4/Thesis-Hana-Masara.pdf.
[74] L.G. Wright, T. Onodera, M.M. Stein, T. Wang, D.T. Schachter, Z. Hu, P.L. McMahon, Deep physical neural networks trained with backpropagation, Nature 601 (7894) (2022) 549–555, https://doi.org/10.1038/s41586-021-04223-6.
[75] D. Tran, Verification of learning-enabled cyber-physical systems, Ph.D. thesis, Vanderbilt University Graduate School, 2020, http://hdl.handle.net/1803/15957.
[76] ONNX, https://onnx.ai/, 2017. (Accessed 30 May 2023).
[77] J. Alama, Euler's polyhedron formula, Formaliz. Math. 16 (1) (2008) 7–17.