The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

STAR SET BASED REACHABILITY ANALYSIS OF NEURAL NETWORKS WITH DIFFERING LAYERS AND ACTIVATION FUNCTIONS

Hana Masara

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Jürgen Giesl

Additional Advisor:
László Antal

Aachen, 13.06.2023

Abstract

In recent years, *neural networks* have gained remarkable importance due to their ability to learn complex patterns, make accurate predictions, and address real-world problems. Ensuring the *safety* and *robustness* of these networks is crucial, making the reachability analysis for neural networks essential. In this work, we focus on *star set* based *reachability analysis* of neural networks with various types of *layers* and *activation functions.* We present a detailed examination of the *exact* and *over-approximate* reachability analysis algorithms of each proposed activation function (ReLU, leaky ReLU, HardTanh, Sigmoid, Unit Step function). Furthermore, we consider the reachability analysis of *unbounded* input star sets. Moreover, we investigate and research different layers types, such as convolutional, pooling, residual, and recurrent layers. To facilitate the integration of various neural network architectures, extend the existing neural network reachability analysis functionality of the Hypro library by implementing an ONNX parser. We demonstrate the effectiveness of our implementation by re-evaluating and verifying different benchmarks and neural networks. Furthermore, we present the results of an experiment aimed at improving the runtime of the reachability analysis algorithms. Overall, our work provides comprehensive and computationally efficient algorithms that enable the reachability analysis of different layers and activation functions.

Keywords— Neural network, safety and robustness verification, piece-wise linear activation functions, reachability analysis, exact computation, over-approximative computation, unbounded computation

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Hana Masara
Aachen, den 13. June 2023

Acknowledgements

I am incredibly grateful to my bachelor thesis supervisor, László Antal, for his countless support and invaluable guidance throughout the entire process of working on this thesis. His deep understanding of the subject and willingness to share his knowledge have been invaluable and helpful. I am also grateful for his constructive feedback, which helped me improve the quality of this work. I would also like to thank my professor, Professor Ábrahám, for giving me the opportunity to write this work and providing me with helpful feedback. Special thanks also go to my partner Lucas Hegerath who gave me valuable mental support during the difficult stages of my bachelor thesis. Lastly, I am very grateful for my parents, who always believed and supported me throughout every significant milestone of my life.

Contents

Chapter 1

Introduction

Artificial intelligence (AI) has experienced remarkable growth in the past two decades, permeating various domains, including healthcare, marketing, banking, gaming, and the automotive industry. Feedforward neural networks have emerged as a leading technique for addressing various problems, including classification [RW17], pattern recognition [AJO+19], natural language processing [HDY+12, LWL+17], and beyond. Their effectiveness is based on their remarkable ability to process and learn from large datasets, enabling them to tackle complex tasks accurately and efficiently. Therefore, verifying neural networks is essential to ensure their reliability, robustness, and safety. It helps to address potential issues, enhance performance, and build trust in AI systems.

In this work, we investigate the reachability analysis of different activation functions using star sets, since it is well suited for the reachability analysis of neural networks. We examine various activation functions: the leaky ReLU, HardTanh, Hard Sigmoid, and Unit Step function. We propose the reachability analysis algorithms for exact, over-approximation applied to bounded/unbounded sets of each of the presented activation functions. The exact and complete analysis of each layer's reachable output set is a union of star subsets. In contrast, the over-approximative analysis tries to reduce the number of stars by over-approximating some of them with fewer stars by performing point-wise over-approximation of the reachable set for all neurons within the layer. We consider the exact and over-approximative analysis of different cases in the unbounded analysis. For each analysis, we formulate lemmas about the worst-case complexity of the number of stars in the reachable set as well as the number of constraints.

In addition to being able to conduct different benchmarks in our used framework Hypro, i.e., hybrid system analysis tool, we implemented an ONNX parser, which unable working with ONNX file format that can represents deep learning models. We re-evaluate different benchmark runtimes using our proposed reachability algorithms besides the safety verification of the benchmark's neural networks and report quantitative results from an experiment to improve the runtime of our implemented algorithms.

1.1 Related Work

Generalized star sets and reachable set computation using the star sets were first introduced in [BD17]. The presented approach is simulation-equivalent, i.e., it can detect if an unsafe state is reachable if and only if a fixed-step simulation exists for the unsafe states.

The reachability analysis of feedforward neural networks with ReLU activation function using the concept of star set was presented in [TMLM+19, Tra20]. This approach deals with the exact and over-approximation analysis. It performs the reachability analysis layer by layer. It evaluates the proposed algorithms, which results in the reachable analysis using the star approach much faster than another approach, such as Reluplex [KBD+17]. Reluplex is a method for verifying the safety and correctness of neural networks. Reluplex is similar to simplex in that it allows variables to temporarily violate their bounds while searching for a viable variable assignment. However, Reluplex goes further by allowing variables with ReLU pairs to temporarily violate ReLU semantics. During the iterative process, Reluplex identifies variables that are either out of bounds or violate a ReLU and applies Pivot and update operations. Furthermore, compared to the zonotope and abstract domain approaches, the over-approximation approach verifies more safety properties due to its minimal over-approximation errors.

Hypro [SÁMK17] is a library to support the implementation of algorithms for the reachability analysis of hybrid systems via flowpipe-construction with mixed discrete-continuous behavior and offers implementations for the most used state-set representations include boxes, convex polytopes, support functions, star sets, or zonotopes. Therefore, we extended hypro by the presented reachability analysis of different activation functions in this work and the implementation of the ONNX parser to facilitate the integration of different neural network architectures represented in the Open Neural Network Exchange (ONNX) format. Furthermore, we researched various layer types that can be implemented in Hypro.

1.2 Thesis Outline

In this thesis, we begin by presenting the fundamental concepts and definitions in Chapter 2. Specifically, in Section 2.1, we introduce the feedforward neural networks, followed by the star set representation and its advantages in Section 2.2.1, and the reachability analysis approach for feedforward neural networks using ReLU activation functions in Section 2.3.

After establishing the foundations for our work, we move forward by introducing our algorithms in Chapter 3. We discuss and investigate the reachability analysis of different activation function layers: the leaky ReLU, HardTanh, Hard Sigmoid, and Unit step function. For each activation function, we present the implemented reachability algorithm for the exact and complete as well as the over-approximation analysis. For both analyses, we investigate the unbounded analysis. Furthermore, we explain the implementation of an ONNX parser.

To assess the effectiveness of the algorithms, we evaluate different benchmarks and conduct an experiment with the intention of improving the runtime of the algorithms in Section 4. Lastly, we comprehensively discuss this work's contributions and limitations. Additionally, we offer insightful suggestions for future directions that can be pursued to expand upon this research.

Chapter 2

Preliminaries

In this chapter, first, we will elucidate the fundamentals of the feedforward neural network (FNN). Then, explain star sets. Lastly, we want to cover the reachability analysis of neural networks with the rectified linear unit (ReLU) activation function. Focusing closely on both the exact and the over-approximation analysis.

2.1 Feedforward Neural Network

Feedforward neural network (FNN) transmits information forward through different layers [Kum19]. The first layer is the input layer, followed by one or multiple hidden layers, and the last layer is the output layer (Figure 2.1).

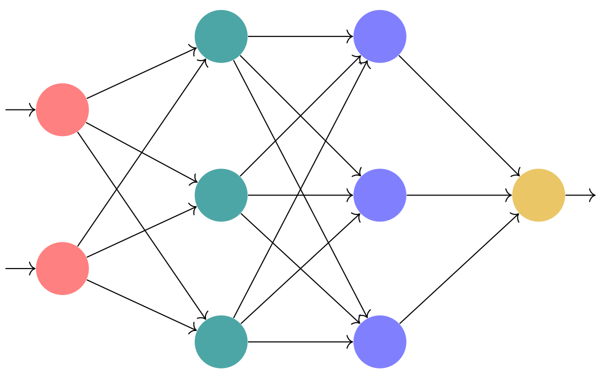


Figure 2.1: Feedforward neural network with four layers. Input with two neurons, two hidden layers, each with three neurons, and an output layer with one neuron.

Each layer consists of neurons connected with all neurons in the next layer and labeled using weights. An input vector x is propagated forward through the FNN. Three components define the output: first, the weight matrix W^k represents the weighted connection between neurons of two layers $k-1$ and k. In addition, the weight coefficient w_{ij}^k characterizes the connection from the j^{th} to the i^{th} neuron (Figure 2.2).

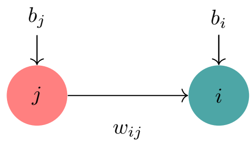


Figure 2.2: Connection between two neurons (Redrawn from [SKP97]).

Secondly, each layer has a bias vector. Therefore b^k represents the bias of the k^{th} layer. However, b_i^k is the bias of the i^{th} neuron in the k^{th} layer. Lastly, the non-linear activation function f applied at each layer [Tra20, SKP97]. Overall y_i give the output of a neuron i by:

$$y_i = f_i(b_i^k + \sum_{j=1}^{n} w_{ij}x_j) \tag{2.1}$$

where x_j is the j^{th} input of the i^{th} neuron [Tra20].

2.2 Star Set Representation

Definition 2.2.1 (Generalized Star Set [BD17])**.** *A generalized n-dimensional star set Θ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, \ldots, v_m\} \subseteq \mathbb{R}^n$ is a set of m vectors called the basis (or generator) vertors, and $P : \mathbb{R}^m \to \{\top, \bot\}$ is a predicate. The set of states represented by the star is given as*

$$[\![\Theta]\!] = \{x \mid x = c + \sum_{i=1}^{m} \alpha_i v_i \text{ such that } P(\alpha_1, \ldots, \alpha_m) = \top\}. \tag{2.2}$$

Sometimes we will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. In this work, we restrict the predicate to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p linear constraints, $C \in \mathbb{R}^{p\times m}$, α is the vector of m-variables i.e. $\alpha = [\alpha_1, \ldots, \alpha_m]^T$, and $d \in \mathbb{R}^{p\times 1}$.

Example 2.2.1. *Let $\Theta = \langle c, V, P\rangle$, where:*

$$\text{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix}, \quad \text{and the center } c = \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

$$\text{also the predicate } P(\alpha) = C\alpha \leq d \text{ where } C = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \text{ and } d = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

In addition, an equivalent definition to the starset would be the following set:

$$[\![\Theta]\!] \equiv \{(x_1, x_2) \mid -1 \leq x_1 \leq 4 \land -3 \leq x_2 \leq 1\}$$

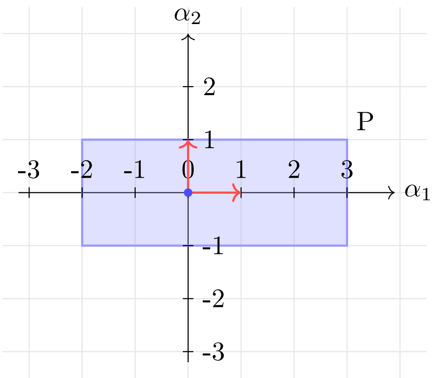


Figure 2.3: The corresponding visualization of the predicate P of the star set from Example 2.2.1.

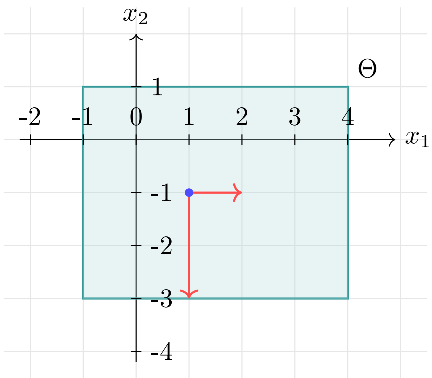


Figure 2.4: The corresponding visualization of the star set from Example 2.2.1.

In the upcoming propositions, we show the universal representation power of star sets. The comprehensive proofs are in Chapter A.

Proposition 2.2.1. *Any bounded convex polyhedron $\mathscr{P} \triangleq \{x \mid Cx \leq d,\ x \in \mathbb{R}^n\}$ can be presented as a star.*

Proposition 2.2.2 (Affine Mapping of Star)**.** *Given a star set $\Theta = \langle c, V, P\rangle$, an affine mapping of the star Θ with the linear mapping matrix W and offset vector b defined by $\bar{\Theta} = \{y \mid y = Wx + b, x \in \Theta\}$ is another star such that*

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}\rangle, \quad \bar{c} = Wc + b, \quad \bar{V} = \{Wv_1, \dots, Wv_m\}, \quad \bar{P} \equiv P$$

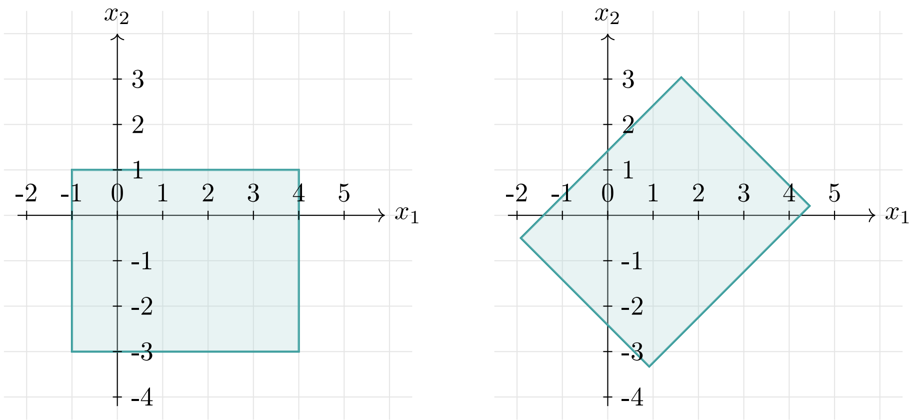
The use of matrix multiplications to the basis and center and one addition of vectors, as well as preserving the predicate in the affine mapping of star sets, means that the complexity of the affine mapping of a star is constant. However, the fact that there is no known polynomial algorithm for the affine mapping of $\mathcal{H}$-polytopes [SFÁ19] indicates that the time complexity of affine mapping of $\mathcal{H}$-polytopes is exponential. In conclusion, star sets are an efficient alternative compared to $\mathcal{H}$-polytopes.

Example 2.2.2. *Let $\Theta = \langle c, V, P\rangle$ be the same as in Example 2.2.1, additionally consider for the affine mapping of the star Θ:*

the affine mapping matrix $W = \begin{bmatrix} cos(45°) & -sin(45°) \\ sin(45°) & cos(45°) \end{bmatrix}$ *and the offset vector* $b = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}$

The resulting star set is defined as $\Theta' = \langle \bar{c}, \bar{V}, P\rangle$ where:

$$\text{the basis } \bar{V} = \begin{bmatrix} 0.707107 & -1.41421 \\ 0.707107 & 1.41421 \end{bmatrix}, \text{ and the center } \bar{c} = \begin{bmatrix} 0.914214 \\ -0.5 \end{bmatrix}$$



Θ before affine mapping. Θ' after affine mapping.

Figure 2.5: The affine mapping results according to Example 2.2.2.

Proposition 2.2.3 (Star and half-space Intersection)**.** *The intersection of a star $\Theta \triangleq \langle c, V, P\rangle$ and a half-space $\mathscr{H} \triangleq \{x \mid Hx \leq g\}$ is another star*

$$\bar{\Theta} = \Theta \cap \mathscr{H} = \langle c, V, \bar{P}\rangle \text{ with } \bar{P} = P \wedge P',$$

$$\text{where } P'(\alpha) \triangleq (H \times V)\alpha \leq g - H \times c,\ V = [v_1, v_2, \ldots, v_m].$$

Example 2.2.3. *Let $\Theta = \langle c, V, P\rangle$ be the same as in Example 2.2.2 where*

$$\text{the basis } V = \begin{bmatrix} 0.707107 & -1.41421 \\ 0.707107 & 1.41421 \end{bmatrix}, \quad \text{and the center } c = \begin{bmatrix} 0.914214 \\ -0.5 \end{bmatrix},$$

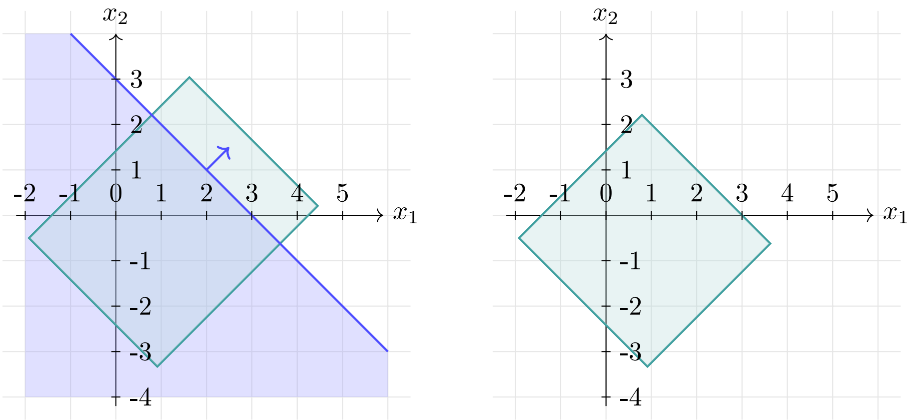
$$\text{also the predicate } P(\alpha) = C\alpha \leq d \text{ where } C = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \text{ and } d = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

Additionally let half-space $\mathscr{H}$ be defined as:

$$\mathscr{H} \triangleq \{x \mid Hx \leq g\} \text{ with } x \in \mathbb{R}^2,\ H = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } g = 3$$

The resulting star set becomes $\Theta' = \langle c, V, \bar{P}\rangle$ where

$$\bar{P}(\bar{\alpha}) = \bar{C}\bar{\alpha} \leq \bar{d} \text{ where } \bar{C} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 1.41421 & 0 \end{bmatrix} \text{ and } d = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \\ 1 \\ 2.58579 \end{bmatrix}$$



Θ before the intersection with the Halfspace $\mathscr{H}$.

Θ' after the intersection.

Figure 2.6: The intersection results according to the Example 2.2.3.

Proposition 2.2.4 (Check for Emptiness)**.** *Given the star set* $\Theta = \langle c, V, P \rangle$ *where* $P(\Theta) \triangleq C\alpha \leq d$ *and* $C \in \mathbb{R}^{p \times m}$*,* α *is the vector of* m*-variables, i.e.,* $\alpha = [\alpha_1 \dots \alpha_m]^T$ *and* $d \in \mathbb{R}^{p \times 1}$*. Check if there exists no alpha that can satify the constraint set* $P = \{C\alpha \leq d \mid \alpha \in \mathbb{R}^m\}$*. This can be accomplished using an LP- or SMT-solver like SMT-RAT [CKJ+15].*

Proposition 2.2.5 (Compute the lower and upper bounds in the star set)**.** *Given an* n*-dimensional set* $\Theta = \langle c, V, P \rangle$ *for each dimension* n*, determine the direction vectors* $[0 \ \dots \ 1 \ \dots \ 0]$, $[0 \ \dots \ -1 \ \dots \ 0]$*, one for the upper bound and one for the lower bound in the inner polytope in the star set. Then, we optimize the obtained direction vectors* $[v_n^1 \ \dots \ v_n^m]$, $[-v_n^1 \ \dots \ -v_n^m]$ *of each dimension by computing the furthest point in the specific direction and shifting it by the center* $c \in \mathbb{R}^n$ *of the star set.*

2.3 Reachability Analysis of Neural Networks with ReLU Activation Function

In this section, we are interested in the reachability analysis of FNN with the ReLU activation function using star set from Definition 2.2.1. But first, to better understand the concept of reachability analysis, it's required to introduce a few other definitions. Generally, reachability analysis involves determining whether a specific state or set of states within a system can be reached [LM17].

Definition 2.3.1 (Reachable Set of FNN)**.** *Given a bounded convex polyhedron input set* $\mathscr{I} \triangleq \{x \mid Ax \leq b,\ x \in \mathbb{R}^n\}$*, and a* k*-layers feed-forward neural network* $F \triangleq \{L_1, \dots, L_k\}$*. Accordingly, the reachable set* $F(\mathscr{I}) = \mathscr{R}_{L_k}$ *of the neural network* F *given the input set* $\mathscr{I}$ *is defined by:*

$$\mathscr{R}_{L_0} \triangleq \mathscr{I},$$
$$\mathscr{R}_{L_i} \triangleq \{y_i \mid y_i = f_i(W_i y_{i-1} + b_i), y_{i-1} \in \mathscr{R}_{L_{i-1}},\}$$

where W_k*,* b_k *and* f_k *are the weight matrix, bias vector and activation function of the* k^{th} *layer* L_k*, respectively. Note that* $\mathscr{R}_{L_i}$ *can be recursively applied to all* $i = 1$ *to* k*.*

Also, the reachable set $\mathscr{R}_{L_k}$ contains the output set of the neural network corresponding to the input set.

Definition 2.3.2 (Safety Verification of FNN)**.** *Given a k-layers feed-forward neural network F, and a safety specification $\mathscr{S}$ defined as a set of linear constraints on the neural network outputs $\mathscr{S} \triangleq \{y_k \mid Cy_k \leq d\}$. The neural network F is safe corresponding to the input set $\mathscr{I}$, .i.e. $F(\mathscr{I}) \models \mathscr{S}$, iff $\mathscr{R}_{L_k} \cap \neg S = \emptyset$.*

Given our current interest in the analysis with the ReLU activation function, let's start with its definition.

Definition 2.3.3 (Rectified Linear Unit (ReLU) Function [YAT+20])**.** *Given the input x,*

$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} = max(0, x) \tag{2.3}$$

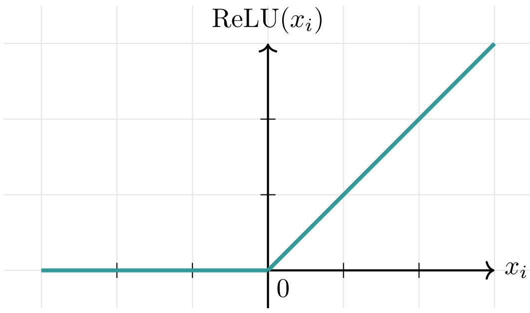


Figure 2.7: Rectified Linear Unit (ReLU) function

2.3.1 Exact and Complete Analysis

Given that any bounded convex polyhedron can be presented as a star by Proposition 2.2.1, we deduce that the input set of a k-layer FNN can be represented as a star set $\Theta = \langle c, V, P \rangle$. According to Proposition 2.2.2, the affine mapping of a star also results in a star, along with Definition 2.3.1 the reachable set on layer k with n neurons can be computed, by applying a sequence of n exactReLU operations $\mathscr{R}_{L_k} = ReLU_n(ReLU_{n-1}(\ldots ReLU_1(\Theta)))$ on the star set input Θ. The ReLU function 2.3.3 handles two cases, first is the input less than zero and second is greater than or equal to zero. Based on that the exactReLU operation on the i^{th} neuron, i.e. $ReLU_i(\cdot)$ works as follows. Given the input star set $\Theta = \langle c, V, P \rangle$, Θ is divided into two subsets $\Theta_1 = \Theta \wedge (x_i \geq 0)$ and $\Theta_2 = \Theta \wedge (x_i < 0)$. Based on Proposition 2.2.3 those subsets are also stars. So let $\Theta_1 = \langle c, V, P_1 \rangle$ and $\Theta_2 = \langle c, V, P_2 \rangle$. The ReLU activation function does not modify the set Θ_1 when applied on x_i in $x = [x_1 \ \ldots \ x_n]^T \in \Theta_1$ becuase ReLU on the positive branch acts as an identity function. On the other side since $x_i < 0$, Θ_2 is projected to zero by the mapping matrix $M = [e_1 \ \ldots \ e_{i-1} \ 0 \ e_{i+1} \ \ldots \ e_n]$. So applying ReLU on the element x_i in $x = [x_1 \ \ldots \ x_n]^T \in \Theta_2$ results a new verctor $x' = [x_1 \ \ldots \ x_{i-1} \ 0 \ x_{i+1} \ \ldots \ x_n]^T$. Therefore, the exactReLU operation for the input star set Θ of the i^{th} neuron results in $ReLU_i(\Theta) = \langle c, V, P_1 \rangle \cup \langle Mc, MV, P_2 \rangle$.

Lemma 2.3.1. *The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is $\mathscr{O}(2^N)$.*

Lemma 2.3.2. *The worst-case complexity of the number of constraints of a star in the reachable set of an N-neurons FNN is $\mathscr{O}(N)$.*

Theorem 2.3.3 (Verification Complexity)**.** *Let F be an FNN with N neurons, Θ an input star set with p linear constraints and m-variables in the predicate, S a safety specification with s linear constraints. In the worst case, determining the verification of the safety of neural network $F(\Theta) \overset{?}{\models} S$ is equivalent to solving up to 2^N feasibility problems, each entails $N + p + s$ linear constraints and m variables.*

Theorem 2.3.4 (Safety and complete counter input set)**.** *Let F be an FNN, Θ a star input set, $F(\Theta) = \bigcup_{i=1}^{k} \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i \rangle$ be the reachable set of the neural network, and S be a safety specification. Denote $\bar{\Theta}_i = \Theta_i \cap \neg S = \langle c_i, V_i, \bar{P}_i \rangle$, $i = 1, \ldots, k$. The neural network is safe iff $\bar{P}_i = \emptyset$ for all i. If the neural network violates its safety property, then the complete counter input set containing all possible inputs in the input set that lead the neural network to unsafe states is $\mathscr{C}_\Theta = \bigcup_{i=1}^{k} \langle c, V, \bar{P}_i \rangle$, where $\bar{P}_i \neq \emptyset$.*

2.3.2 Over-approximate Analysis

While it is possible to perform an exact and complete analysis to compute the reachable values of a ReLU FNN, the over-approximative reachability analysis is an alternative approach to address the problem of exponential growth in the number of stars with each neuron, which significantly increases the computational cost and thus limits scalability. This approach uses a different algorithm which involves constructing only a single star at each neuron using the following rule.
For any input x_i the output $y_i = ReLU(x_i)$, let

$$\begin{cases} y_i = x_i & l_i \geq 0, \\ y_i = 0 & u_i \leq 0, \\ y_i \geq 0,\ y_i \leq \frac{u_i(x_i - l_i)}{u_i - l_i},\ y_i \geq x_i & l_i < 0 \text{ and } u_i > 0 \end{cases} \tag{2.4}$$

where l_i and u_i are lower and upper bounds of x_i.

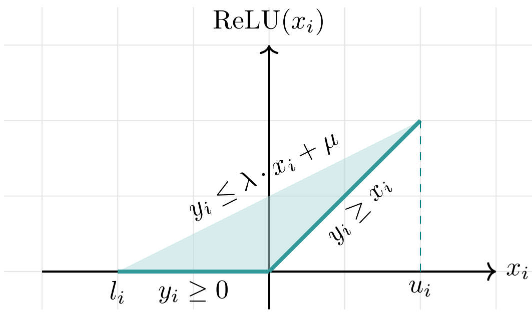


Figure 2.8: Convex relaxation for the ReLU function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $\lambda = \frac{u_i}{u_i - l_i}$ and $\mu = -\frac{l_i u_i}{u_i - l_i}$ (Redrawn from [SGPV19]).

Similar to the exact approach, the over-approximate reachable set of a layer with n neurons can be computed by executing a sequence of n approxReLU operations. The algorithm takes the star set $\Theta = \langle c, V, P \rangle$ as input. After first determining the

lower and upper bounds using Proposition 2.2.5, there are three cases. First, if the lower bound is greater than or equal to zero, which means the set is in the positive range, the star set remains the same. If the upper bound is less than zero, i.e., the set is in the negative range, star set's values in the given dimension are projected to zero, similarly to the exact analysis. But if the upper limit is greater than zero and the lower limit is less than zero, we introduce a new variable α_{m+1} to the predicate P where $P(\alpha) \triangleq C\alpha \leq d$. α_{m+1} encodes the over-approximation of the activation function at the i^{th} neuron according to Equation 2.4.
Consequently, we have three new linear constraints:

$$\alpha_{m+1} \geq x_i, \quad \alpha_{m+1} \geq 0, \quad \alpha_{m+1} \leq \frac{u_i(x_i - l_i)}{u_i - l_i}$$

As the reachable set contains one more variable and three more linear constraints in the predicate, this leads to the following analysis complexity.

Lemma 2.3.5. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neurons FNN is* $N + m_0$ *and* $3N + n_0$*, respectively, where* m_0 *is the number of variables and* n_0 *the number of linear constraints of the predicate of the input set.*

Chapter 3

Implementation

In the previous chapter, we first presented the foundation of neural networks, and furthermore introduced the star sets and their benefits. Moreover, we introduced an algorithm for star based reachability analysis, both exact computation and over-approximation the reachable set of neural networks using ReLU activation function. In this chapter, our interest lies in investigating the reachability analysis using other activation functions. Additionally, we discuss various types of non-fully connected layers. Lastly, we review a common file format to represent, store and share machine learning models, Open Neural Network Exchange (ONNX) [ONN].

3.1 Star Set based Reachability Analysis of Neural Networks

This section presents the extension of our star based reachability analysis algorithm to include and incorporate also leaky rectified linear unit (leaky ReLU), hard hyperbolic tangent (HardTanh), hard sigmoid (HardSigmoid), and unit step activation functions. First, we begin by introducing the definition of each function. Next, we present a concept for the corresponding reachability analysis. More specifically, as with ReLU reachability analysis, we examine the exact and over-approximative analyses, as well as the analysis of an unbounded input. We implemented the described methods using the hybrid system analysis tool HyPro [SÁMK17].

If the neural network does not hold any activation function, the resulting output signal would be a linear function, essentially a polynomial of degree one. However, neural networks are designed to learn relationships between inputs and outputs that are non-linear and complex. Correspondingly, non-linear activation functions are used in neural networks to prevent linearity [SSA20].
Despite the importance of the activation functions in neural networks, they face challenges that can impact their effectiveness. The two main problems are the *vanishing gradient* problem and the *dead neuron* problem.

Vanishing gradient problem [Dat20, Edu23]
The term *vanishing gradient problem* refers to a phenomenon when the gradient approaches a very small value, close to zero. Neural networks train and learn by adjusting the weights of their neurons based on the error it makes in predicting the correct

output. This adjustment happens with the help of backpropagation. Backpropagation computes the network's gradients, which is the measure of how the weights should change, and propagates it back through the network to update the weights. Certain activation functions squish a large input into a small output. Therefore, a large change in the function's input results in a small change in the output, this result in the gradient vanishing. Correspondingly, it makes it difficult to learn and improve the parameters of the earlier layers in the network.

Dead neuron problem [Dat20, Ram21]
The "Dead Neuron" problem in neural networks refers to a scenario where a neuron becomes unresponsive and does not contribute to the network output. The dead neuron problem occurs when an activation function forces a significant part of the input to zero or close to zero, rendering corresponding neurons inactive or "dead" in contributing to the final output. Once this happens, the network struggles to recover, and a large part of the input fails to contribute to the network's functioning.

The ReLU activation function is widely used in neural networks because of its efficiency and substantial contribution to solving the vanishing gradient problem, having its gradient be either 0 or 1. Nevertheless, since the ReLU activation function evaluates the negative input to a zero output, the neural network suffers from the previously seen dead neuron problem.

Moving forward, we will discuss different activation function alternatives and their star set based reachability analysis. We assume that the input is a star set $I = \langle c, V, P \rangle$ for all the upcoming reachability analyses since any bounded convex polyhedron can be presented as a star by Proposition 2.2.1, and the affine mapping of a star also results in a star by Proposition 2.2.2. Since the reachable set is derived laver-by-layer, the main step in computing the reachable set of a layer is applying the activation function on the star input, i.e., calculate $f(\Theta)$ where f is the chosen activation function and $\Theta = \langle c, V, P \rangle$.

3.1.1 Reachability of Leaky ReLU Layer (LReLU)

Leaky ReLU is another type of activation function. It was introduced by Mass et al.[Maa13].

Definition 3.1.1 (Leaky ReLU Function [XLD+20])**.** *Given the input x*

$$LeakyReLU(x) = \begin{cases} x, & x > 0 \\ \gamma \cdot x, & x \leq 0 \end{cases} = max(\gamma \cdot x, x) \tag{3.1}$$

where $\gamma \in (0,1)$.

Due to the dead neuron problem caused by the ReLU activation function, the alternative variant, Leaky ReLU, was introduced. The leaky ReLU function allows a small part of the negative input to be passed on as output, unlike the ReLU function, where the negative part is pushed to zero. Correspondently, this helps to minimize the occurrence of silent or dead neurons. Nevertheless, the gradient of the negative outputs may cause the vanishing gradient problem to occur [Gus22, Dat20, XLD+20].

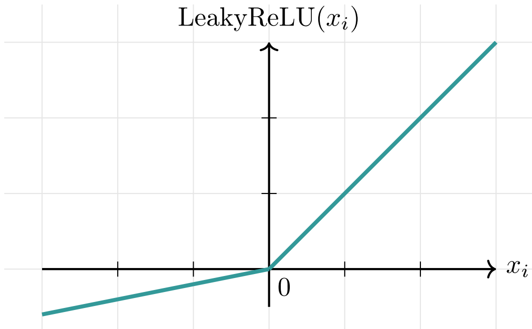


Figure 3.1: Leaky ReLU function (LReLU) where $\gamma = 0.2$

Exact and Complete Analysis

Given the input $\Theta = \langle c, V, P \rangle$ then, we can apply a sequence of n exactLReLU operations so that $\mathscr{R}_k = LeakyReLU_n(LeakyReLU_{n-1}(\ldots LeakyReLU_1(\Theta)))$ computes the reachable set on layer k with n neurons. First, we compute the input's lower bound l_i and upper bound u_i at the i^{th} neuron. After that, the operation as presented in Algorithm 1 distinguishes three cases:

- If the lower bound l_u is in the positive range, i.e., is not negative, no change is applied to the input set, and the exactLReLU returns a set that is the same as the input set.
- If the upper bound u_i is equal or less than zero, i.e., is not positive, the exactLReLU returns a new reachable set where the i^{th} state variable x_i is set to $\gamma \cdot x_i$.
- Otherwise, Θ is decomposed into two subsets $\Theta_1 = \Theta \wedge (x_i > 0)$ and $\Theta_2 = \Theta \wedge (x_i \leq 0)$. Based on Proposition 2.2.3 those subsets are also stars. So let $\Theta_1 = \langle c,V,P_1 \rangle$ and $\Theta_2 = \langle c, V, P_2 \rangle$. By Definition 3.1.1, for x_i in $x = [x_1 \ldots x_n]^T \in \Theta_1$, there is no change after applying the leaky ReLU activation function on Θ_1. Since Θ_2 has $x_i \leq 0$, for $x = [x_1 \ \ldots \ x_n]^T \in \Theta_2$ applying leaky ReLU will yield a new vector $x' = [x_1 \ \ldots \ x_{i-1} \ \gamma x_i \ x_{i+1} \ \ldots \ x_n]^T$. This is equivalent to mapping Θ_2 by the scaling matrix $M = [e_1 \ \ldots \ e_{i-1} \ \gamma \ e_{i+1} \ \ldots \ e_n]$. Accordingly, the exactLReLU operation of the i^{th} neuron for the input star set Θ results in $LeakyReLU_i(\Theta) = \langle c, V, P1 \rangle \cup \langle Mc, MV, P2 \rangle$.

A concrete example of exactLReLU operation is illustrated in Example 3.1.1. To reduce the number of calculations, it is helpful to determine the ranges of all states in the input star set Θ at the i^{th} neuron beforehand, as presented in the first two cases in line 15 and line 17 in Algorithm 1.

Lemma 3.1.1. *The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is $\mathscr{O}(2^N)$.*

Lemma 3.1.2. *The worst-case complexity of the number of constraints of a star in the reachable set of an N-neurons FNN is $\mathscr{O}(N)$.*

Example 3.1.1. *Let* $\Theta = \langle c, V, P \rangle$ *be the input set, where:*

$$\text{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\text{also the predicate } P(\alpha) \triangleq C\alpha \leq d \text{ where } C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ 3.9 & 1.6 \\ 2 & 2.4 \\ -0.9 & -3.8 \end{bmatrix} \text{ and } d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$$

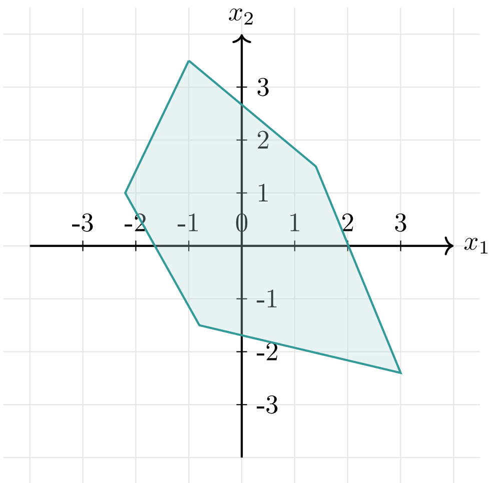


Figure 3.2: The input star set Θ corresponding to Example 3.1.1

We apply the exactLReLU operation on the dimensions of Θ *with* $\gamma = 0.2$*, resulting in:*

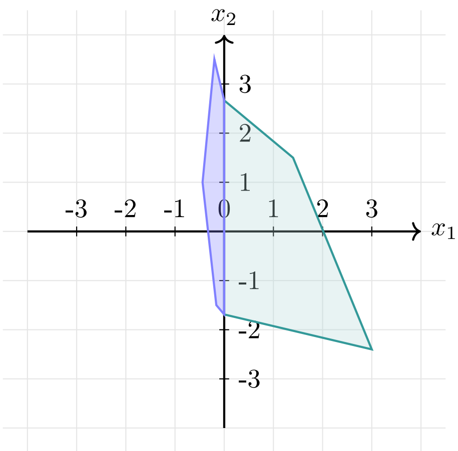


Figure 3.3: Θ' after applying exactLReLU on the first dimension.

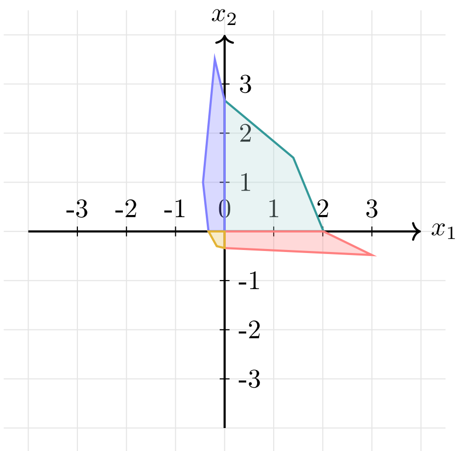


Figure 3.4: Θ'' after applying exactLReLU on the second dimension.

Algorithm 1 Star set based exact reachability analysis for a leaky ReLU layer

Input: Input star set $I = [\Theta_1 \dots \Theta_N]$
Output: Exact reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $\mathcal{R} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    $I_1 \leftarrow W * \Theta_j + b = \langle Wc_j + b, WV_j, P_j \rangle$
    $\mathcal{R}_1 \leftarrow I_1$
    for $i = 1 : n$ **do** ▷ *n exactLReLU operations*
      $\mathcal{R}_1 \leftarrow$ exactLReLU$(\mathcal{R}_1, i, l_i, u_i)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$
  return $\mathcal{R}$
procedure EXACTLRELU$(\tilde{I}, i, l_i, u_i)$
  $\tilde{\mathcal{R}} \leftarrow \emptyset$, $\tilde{I} = [\tilde{\Theta}_1, \dots, \tilde{\Theta}_k]$ ▷ *Intermediate representations*
  for $j = 1 : k$ **do**
    $[l_i, u_i] \leftarrow \tilde{\Theta}_j$.range$(i)$ ▷ $l_i \leq x_i \leq u_i, x_i \in I_1[i]$
    $\mathcal{R}_1 \leftarrow \emptyset$, $M \leftarrow [e_1 \ \dots \ e_{i-1} \ \gamma \ e_{i+1} \ \dots \ e_n]$
    if $l_i > 0$ **then**
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$
    else if $u_i \leq 0$ **then**
      $\mathcal{R}_1 \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j \rangle$
    else ▷ $l_i < 0$ & $u_i > 0$
      $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge x[i] > 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] \leq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j \rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
    $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \mathcal{R}_1$
  return $\tilde{\mathcal{R}}$

Over-approximate Analysis

Although the exact algorithm computes the exact reachable sets of a leaky ReLU FNN, the over-approximative reachability analysis, like in ReLU, is an approach to avoid the exponentially growing number of stars over the number of layers. In this section, we investigate an over-approximative reachability algorithm for leaky ReLU FNNs. Here, we also construct only a single star at each neuron using the following approximation rule.

Lemma 3.1.3. *For the input x_i, the output $y_i = LeakyReLU(x_i)$, let*

$$\begin{cases} y_i = x_i & l_i \geq 0 \\ y_i = x_i \cdot \gamma & u_i \leq 0 \\ y_i \geq x_i \cdot \gamma,\ y_i \leq \frac{u_i - (\gamma \cdot l_i)}{u_i - l_i} \cdot x_i + \frac{(u_i \cdot l_i)(\gamma - 1)}{u_i - l_i},\ y_i \geq x_i & l_i < 0 \ \wedge \ u_i > 0 \end{cases} \tag{3.2}$$

where l_i and u_i are lower and upper bounds of x_i.

The approximate reachable set of a layer with n neurons is also computed by executing a sequence of n approxLReLU operations as shown in Algorithm 2. Given the input $\Theta = \langle c, V, P \rangle$. For the state variable x_i at the i^{th} neuron, the algorithm determines the lower and upper bounds l_i, u_i. The approxLReLU operation differentiates

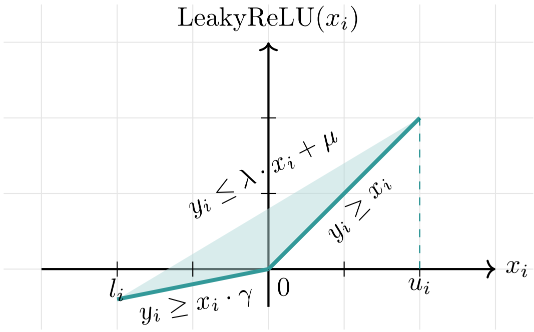


Figure 3.5: Convex relaxation for the leaky ReLU function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $\lambda = \frac{u_i-(\gamma \cdot l_i)}{u_i-l_i}$ and $\mu = \frac{(u_i \cdot l_i)(\gamma-1)}{u_i-l_i}$ and $\gamma = 0.2$.

between three cases.

- The input remains unchanged if the lower bound l_i is non-negative, as in line 12. The approxLReLU returns a set that is the same as the input set.
- If the upper bound u_i is equal or less than zero, the set is mapped by the scaling matrix $M = [e_1 \ \dots \ e_{i-1} \ \gamma \ e_{i+1} \ \dots \ e_n]$ so that the i^{th} state variable is set to γx_i in the new returned reachable set (line 14).
- Though, if the lower bound is negative and the upper bound is positive, as in line 16, we introduce a new variable α_{m+1} to the predicate $P(\alpha) \triangleq C\alpha \leq d$ that represents the over-approximation of the leaky ReLU function at the i^{th} neuron according to Equation 3.2. Resulting in three more linear constraints in the predicate $P'(\alpha') \triangleq C'\alpha' \leq d'$ where $\alpha' = [\alpha_1 \ \dots \ \alpha_m \ \alpha_{m+1}]$:

$$\alpha_{m+1} \geq x_i, \quad \alpha_{m+1} \geq x_i \cdot \gamma, \quad \alpha_{m+1} \leq \frac{u_i - (\gamma \cdot l_i)}{u_i - l_i} \cdot x_i + \frac{(u_i \cdot l_i)(\gamma - 1)}{u_i - l_i}$$

Adding one more variable and three more linear constraints in the predicate of the reachable set leads to the following lemma.

Lemma 3.1.4. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neurons FNN is $N + m_0$ and $3N + n_0$, respectively, where m_0 is the number of variables and n_0 the number of linear constraints of the predicate of the input set.*

Example 3.1.2. *Let $\Theta = \langle c,V,P \rangle$ be the input set where:*

$$\textit{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \textit{ and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\textit{also the predicate } P(\alpha) \triangleq C\alpha \leq d \textit{ where } C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ 3.9 & 1.6 \\ 2 & 2.4 \\ -0.9 & -3.8 \end{bmatrix} \textit{ and } d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$$

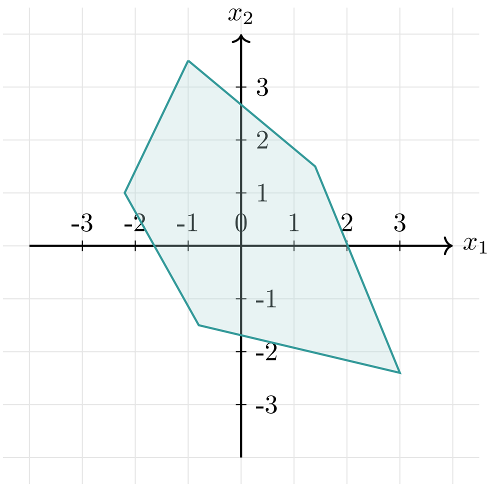


Figure 3.6: The input star set Θ corresponding to Example 3.1.2

We apply the approxLReLU operation on the dimensions of Θ *with* $\gamma = 0.2$*, resulting in:*

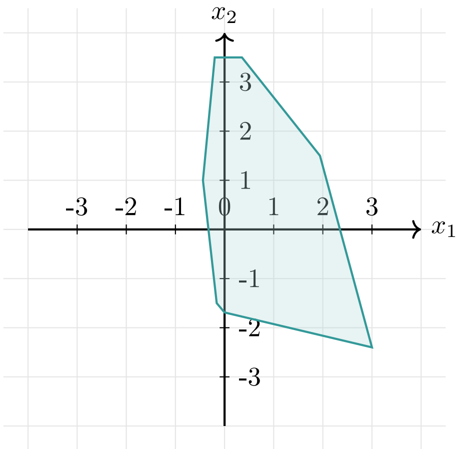


Figure 3.7: Θ' after applying approxLReLU on the first dimension.

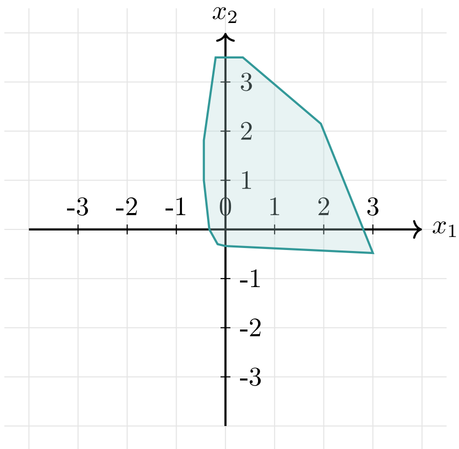


Figure 3.8: Θ'' after applying approxLReLU on the second dimension.

Algorithm 2 Star set based over-approximate reachability analysis for a leaky ReLU layer

Input: Input star set $I = [\Theta]$
Output: Over-approximate reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $I_1 \leftarrow W * I_0 + b = \langle Wc + b, WV, P\rangle$
  $I' \leftarrow I_1$
  for $i = 1 : n$ **do** ▷ *n approxLReLU operations*
    $I' \leftarrow$ approxLReLU(I', i)
  $\mathcal{R}_1 \leftarrow I'$
procedure APPROXLRELU$(\tilde{I}, i)$
  $\tilde{I} \leftarrow \Theta = \langle \tilde{c}, \tilde{V}, \tilde{P}\rangle$
  $[l_i, u_i] \leftarrow \tilde{\Theta}.range(i)$ ▷ $l_i \leq x_i \leq u_i$
  $M \leftarrow [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$
  $M' \leftarrow [e_1 \ \dots \ e_{i-1} \ \gamma \ e_{i+1} \ \dots \ e_n]$
  if $l_i \geq 0$ **then**
    $\tilde{\mathcal{R}} \leftarrow \tilde{\Theta}_j = \langle \tilde{c}, \tilde{V}, \tilde{P}\rangle$
  else if $u_i \leq 0$ **then**
    $\tilde{\mathcal{R}} \leftarrow M' * \tilde{\Theta} = \langle M'\tilde{c}, M'\tilde{V}, \tilde{P}\rangle$
  else ▷ $l_i < 0$ & $u_i > 0$
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \dots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \dots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable* α_{m+1}
    $C_1 \leftarrow [\tilde{V}_i \ \ -1]$, $d_1 \leftarrow -\tilde{c}_i$
    $C_2 \leftarrow [\tilde{V}_i \ \ -\gamma]$, $d_2 \leftarrow -\gamma \cdot \tilde{c}_i$
    $C_3 \leftarrow [-\frac{u_i - (\gamma \cdot l_i)}{u_i - l_i}\tilde{V}_i \ 1]$, $d_3 \leftarrow \frac{(u_i - \gamma \cdot l_i)\tilde{c}_i + l_i \cdot u_i(\gamma - 1)}{u_i - l_i}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m \times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P'\rangle$

Unbounded Analysis

After presenting the exact and over-approximative analysis, in this section, we want to discuss the reachability analysis of an input star set with the condition of being unbounded. For the input star set $\Theta = \langle c, V, P\rangle$, we consider three scenarios: The input Θ has only one lower or only one upper or no bounds at all.

- First, if the input star set Θ has a lower, but no upper bound, i.e., the input (in the given dimension) can take any positive value, without a maximal limit. Computing the exact and complete analysis on this input would be the same process as applying the exactLReLU operation on a bounded input star set in 3.1.1. Since we can detect whether the lower is in the positive range, we get a reachable set that is the same as the input. If the lower bound is in the negative range and we know that the upper bound growth extends infinitely into the positive range, we treat the set the same as in Algorithm 1, line 19. However, the case that the input set is always in the negative range, as in line 17, will never occur as the set continuously grows infinitely in the positive direction.
  Regarding the overapproximative analysis, our presented approximation rule 2.8

will change to the following.

Lemma 3.1.5. *For the input x_i, the output $y_i = LeakyReLU(x_i)$, let*

$$\begin{cases} y_i = x_i & l_i \geq 0 \\ y_i \geq x_i \cdot \gamma,\ y_i \leq x_i + l_i \cdot (\gamma - 1),\ y_i \geq x_i & l_i < 0 \end{cases} \tag{3.3}$$

where l_i is the lower bound of x_i.

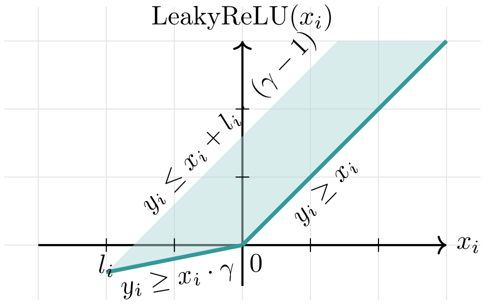


Figure 3.9: Convex relaxation for the leaky ReLU function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $\gamma = 0.2$.

Applying approxLReLU on unbounded input set according to the newly presented rule works as follows. After computing the lower bound, if it is greater than 0, the input remains as it is. But if the lower bound is less than 0, we introduce a new variable α_{m+1} to capture the over-approximation, which will result in three more linear constraints in the predicate:

$$\alpha_{m+1} \geq x_i, \quad \alpha_{m+1} \geq x_i \cdot \gamma, \quad \alpha_{m+1} \leq x_i + l_i \cdot (\gamma - 1)$$

- The second case is when the input star set has an upper but no lower bound, meaning that the input extends infinitely to the negative direction without a minimum limit. Similar to the first case applying exactLReLU on this input will be the same as applying it on a bounded input star set. We will only notice a change with respect to the over-approximative analysis. Therefore, the over-approximation rule changes as follows since we only have an upper bound, and the set does not have a minimum limit.

Lemma 3.1.6. *For the input x_i, the output $y_i = LeakyReLU(x_i)$, let*

$$\begin{cases} y_i = \gamma \cdot x_i & u_i \leq 0 \\ y_i \geq x_i \cdot \gamma,\ y_i \leq \gamma \cdot x_i + u_i \cdot (1 - \gamma),\ y_i \geq x_i & u_i > 0 \end{cases} \tag{3.4}$$

where u_i is the upper bound of x_i.

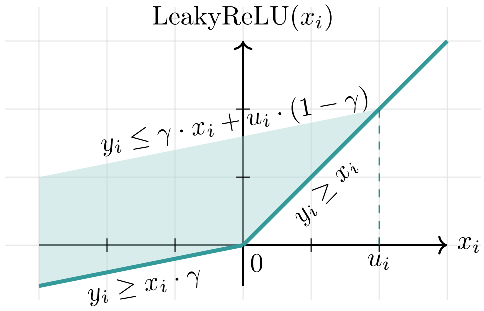


Figure 3.10: Convex relaxation for the leaky ReLU function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $\gamma = 0.2$.

For the approxLReLU, we start by determining the upper bound. If the upper bound is less than zero, we operate the same as over-approximation using a bounded input set. If the upper bound is greater than zero, we introduce a new variable α_{m+1}, representing the over-approximation. Accordingly, we have three new constraints:

$$\alpha_{m+1} \geq x_i, \quad \alpha_{m+1} \geq x_i \cdot \gamma, \quad \alpha_{m+1} \leq \gamma \cdot x_i + u_i \cdot (1-\gamma)$$

- The last case is when the input has no lower or upper bound bounds, i.e., the input has no finite limits in the analyized dimension. According to this case, our exact analysis does not differ from the case of having a bounded input set. Nevertheless, the over-approximative analysis would differ, resulting in the following rule.

Lemma 3.1.7. *For the input x_i, the output $y_i = LeakyReLU(x_i)$, let*

$$\Big\{ y_i \geq x_i \cdot \gamma,\ y_i \geq x_i \quad x_i \in \mathbb{R} \tag{3.5}$$

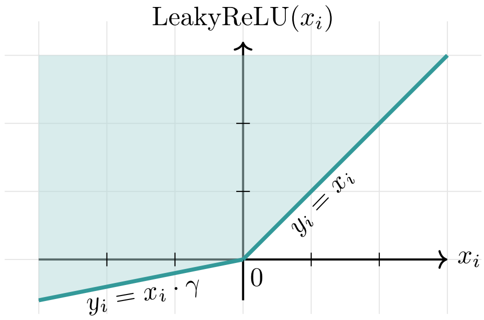


Figure 3.11: Convex relaxation for the leaky ReLU function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $\gamma = 0.2$.

Since the input star set does not have an upper or lower bound, by introducing the new variable α_{m+1}, the over-approximation is only limited by the two new constraints:

$$\alpha_{m+1} \geq x_i, \quad \alpha_{m+1} \geq x_i \cdot \gamma$$

Lemma 3.1.8. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neurons FNN is $N + m_0$ and $3N + n_0$, where respectively m_0 is the number of variables and n_0 the number of linear constraints of the predicate of the input set.*

3.1.2 Reachability of Hard Tanh Layer

The hard hyperbolic tangent function, also referred to as the HardTanh function, is a variation of the hyperbolic tangent activation function. It is defined as follows.

Definition 3.1.2 (Hard hyperbolic tangent Function (HardTanh) [Col04])**.** *For the input x*

$$HardTanh(x) = \begin{cases} -1, & x < -1 \\ 1, & x > 1 \\ x, & -1 \leq x \leq 1 \end{cases} \tag{3.6}$$

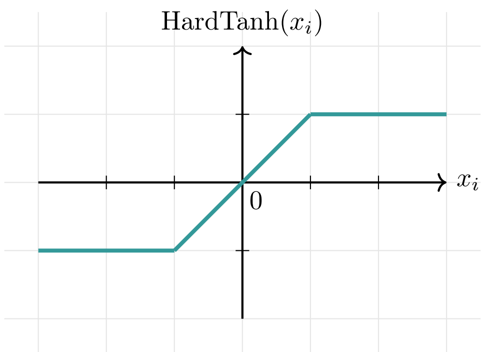


Figure 3.12: Hard hyperbolic function (HardTanh)

The hyperbolic tangent (tanh) is more popular than the sigmoid function since it gives better training performance [Gus22]. The tanh function binds a large input range to a range between -1 and 1. Thus, a large change in the input value results in a minimal change to the output value. This leads to close to zero gradient values. Therefore, the tanh function suffers from the vanishing gradient problem. However, the HardTanh function does not face the dead neuron problem since its range contains positive and negative values. We chose the HardTanh version for our implementation since it is cheaper and more computational than the tanh function [CWB+11, NIGM18] as well as since we can verify it, in contrast to the tanh.

Exact and Complete Analysis

With the input $\Theta = \langle c, V, P \rangle$, we compute the reachable set by executing a sequence of n exactHTangent operations $\mathscr{R}_k = HardTanh_n(HardTanh_{n-1}(\ldots HardTanh_1(\Theta)))$ for a layer k with n neurons. For our implemented exactHTangent operation, we modified the function and exchanged the -1 and 1 with $minVal$ and $maxVal$ to adapt the function to the own use. $minVal$ stands for the minimum value of the linear part and $maxVal$ for the maximum value of the linear part. This yields the following function definition:

$$HardTanh(x) = \begin{cases} minVal, & x < minVal \\ maxVal, & x > maxVal \\ x, & minVal \leq x \leq maxVal \end{cases} \tag{3.7}$$

Like the other reachability analyses, we first compute the lower and upper bounds l_i, u_i on the i^{th} neuron. Afterward, as outlined in the Algorithm 3, the function exactHTangent tackles six different cases.

- If the lower bound l_i is greater than $minVal$ and the upper bound u_i is less than $maxVal$, the input set remains unchanged, and the function returns a new reachable set which is the same as the input set.
- If the upper bound u_i is less than the $minVal$, we project the input onto $minVal$. First, we project the set to zero by the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. Then we set the center of the input set at the i^{th} position to $minVal$, i.e., $c_i = minVal$.
- Similar to the second case, when the lower bound l_i is greater than $maxVal$, we project the input onto the $maxVal$. We initiate the process by mapping the set to zero with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. Afterwards, the center at the i^{th} position is set to $maxVal$, $c_i = maxVal$.
- When the lower bound l_i is between $minVal$ and $maxVal$ and the upper bound u_i is greater than $maxVal$. Then, we split the input set into two subsets $\Theta_1 = \Theta \wedge (minVal \leq x_i \leq maxVal)$ and $\Theta_2 = \Theta \wedge (x_i > maxVal)$. By Definition 3.7, the first subset $\Theta_1 = \langle c, V, P_1 \rangle$ remains unchanged since it is in the range between $minVal$ and $maxVal$. However, since $\Theta_2 = \langle c, V, P_2 \rangle$ is greater than $maxVal$, applying the activation function will leads to the new vector $x' = [x_1 \ \dots \ x_{i-1} \ maxVal \ x_{i+1} \ \dots \ x_n]^T \in \Theta_2$, which is the same as projecting the set to zero by the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$, then the center c_i is set to $maxVal$ so that $c_i = maxVal$. Finally, the exactHTangent operation at the i^{th} neuron for the input star set Θ results in $HardTanh_i(\Theta) = \langle c, V, P1 \rangle \cup \langle Mc + s, MV, P2 \rangle$, where s is the shifting vector $s = [0 \dots maxVal \dots 0]^T$.
- When the lower bound l_i is less than $minVal$, and the upper bound u_i is less than $maxVal$, i.e., in the range between $minVal$ and $maxVal$, we split the input set into two subsets $\Theta_1 = \Theta \wedge (minVal \leq x_i \leq maxVal)$ and $\Theta_2 = \Theta \wedge (x_i < minVal)$. By Definition 3.7, the first subset $\Theta_1 = \langle c, V, P_1 \rangle$ remains unchanged since it is in the range between $minVal$ and $maxVal$. $\Theta_2 = \langle c, V, P_2 \rangle$ is projected to $minVal$, by projecting the set to zero by mapping the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ and afterward changing the center c_i to $minVal$. Finally, the exactHTangent operation at the i^{th} neuron for the input star set Θ results in $HardTanhT_i(\Theta) = \langle c, V, P1 \rangle \cup \langle Mc + s', MV, P2 \rangle$, where s' is the shifting vector, $s' = [0 \ \dots \ minVal \ \dots \ 0]^T$.
- The last case occurs when the lower bound l_i is less than $minVal$, and the upper bound u_i is greater than $maxVal$. We partition the set into three subsets

$\Theta_1 = \Theta \wedge (minVal \leq x_i \leq maxVal)$, $\Theta_2 = \Theta \wedge (x_i > maxVal)$, and $\Theta_3 = \Theta \wedge (x_i < minVal)$. $\Theta_1 = \langle c, V, P_1 \rangle$, by Definition 3.7, remains the same. In contrast, applying the activation function on x_i of $x = [x_1 \ \ldots \ x_n]^T \in \Theta_2$ leads to a new vector $x' = [x_1 \ \ldots \ x_{i-1} \ maxVal \ x_{i+1} \ \ldots \ x_n]^T$,i.e., we map the set with the mapping matrix $M = [e_1 \ \ldots \ e_{i-1} \ 0 \ e_{i+1} \ \ldots \ e_n]$, then set the center to $maxVal$. For Θ_3, we do the same, but instead of setting the center c_i to $maxVal$, we place it to $minVal$. Accordingly, the exactHTangent operation at the i^{th} neuron for the input star set Θ results in a union of three star sets $HardTanh_i(\Theta) = \langle c, V, P1 \rangle \cup \langle Mc+s, MV, P2 \rangle \cup \langle Mc+s', MV, P3 \rangle$, where s, s' are the shifting vectors, $s = [0 \ \ldots \ maxVal \ \ldots \ 0]^T$ and $s' = [0 \ \ldots \ minVal \ \ldots \ 0]^T$.

Lemma 3.1.9. *The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is $\mathcal{O}(3^N)$.*

Lemma 3.1.10. *The worst-case complexity of the number of constraints of a star in the reachable set of an N-neurons FNN is $\mathcal{O}(2N)$.*

Example 3.1.3. *Let $\Theta = \langle c, V, P \rangle$ be the input set where:*

$$\textit{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \textit{ and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\textit{also the predicate } P(\alpha) \triangleq C\alpha \leq d \textit{ where } C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ 3.9 & 1.6 \\ 2 & 2.4 \\ -0.9 & -3.8 \end{bmatrix} \textit{ and } d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$$

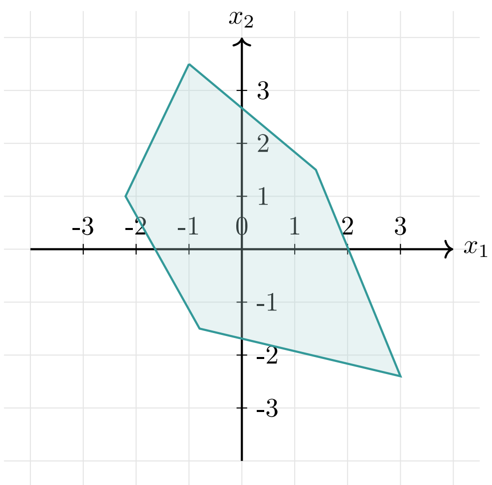


Figure 3.13: The input star set Θ corresponding to Example 3.1.3

We apply the exactHTangent operation on the dimensions of Θ with $minVal = -2$ and $maxVal = 2$, resulting in:

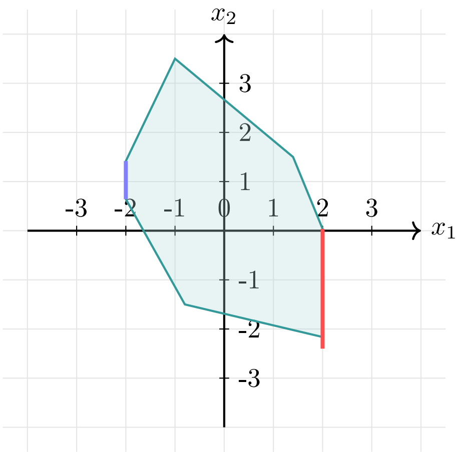


Figure 3.14: Θ' after applying exactHTangent on the first dimension.

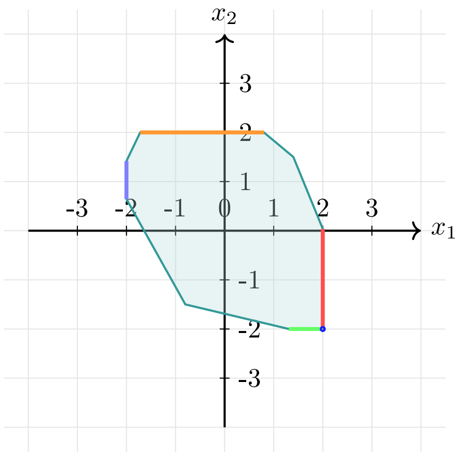


Figure 3.15: Θ'' after applying exactHTangent on the second dimension.

Algorithm 3 Star set based exact reachability analysis for a HardTanh layer

Constants: $minVal, maxVal$
Input: Input star set $I = [\Theta_1 \dots \Theta_N]$
Output: Exact reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
    $\mathcal{R} \leftarrow \emptyset$
    for $j = 1 : N$ **do**
        $I_1 \leftarrow W * \Theta_j + b = \langle Wc_j + b, WV_j, P_j \rangle$
        $\mathcal{R}_1 \leftarrow I_1$
        for $i = 1 : n$ **do** ▷ *n exactHTangent operations*
            $\mathcal{R}_1 \leftarrow \text{exactHTangent}(\mathcal{R}_1, i, l_i, u_i)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$
    return $\mathcal{R}$
procedure EXACTHTANGENT$(\tilde{I}, i, l_i, u_i)$
    $\tilde{\mathcal{R}} \leftarrow \emptyset$, $\tilde{I} = [\tilde{\Theta}_1, \dots, \tilde{\Theta}_k]$ ▷ *Intermediate representations*
    for $j = 1 : k$ **do**
        $[l_i, u_i] \leftarrow \tilde{\Theta}_j.\text{range}(i)$ ▷ $l_i \leq x_i \leq u_i, x_i \in I_1[i]$
        $\mathcal{R}_1 \leftarrow \emptyset$, $M \leftarrow [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$
        $s \leftarrow [0 \ \dots \ maxVal \ \dots \ 0]^T$
        $s' \leftarrow [0 \ \dots \ minVal \ \dots \ 0]^T$
        if $l_i \geq minVal$ **and** $u_i \leq maxVal$ **then**
            $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$
        else if $u_i < minVal$ **then**
            $\tilde{\Theta}_j \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s', M\tilde{V}_j, \tilde{P}_j \rangle$
            $\mathcal{R}_1 \leftarrow \tilde{\Theta}$
        else if $l_i > maxVal$ **then**
            $\tilde{\Theta}_j \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}_j \rangle$
            $\mathcal{R}_1 \leftarrow \tilde{\Theta}$
        else if $minVal \leq l_i \leq maxVal$ **and** $u_i > maxVal$ **then**
            $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge minVal \leq x[i] \leq maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
            $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] > maxVal = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j \rangle$
            $\tilde{\Theta}''_j \leftarrow M * \tilde{\Theta}''_j = \langle M\tilde{c}''_j + s, M\tilde{V}''_j, \tilde{P}''_j \rangle$
            $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
        else if $l_i < minVal$ **and** $u_i \leq maxVal$ **then**
            $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge minVal \leq x[i] \leq maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
            $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] < minVal = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j \rangle$
            $\tilde{\Theta}''_j \leftarrow M * \tilde{\Theta}''_j = \langle M\tilde{c}''_j + s', M\tilde{V}''_j, \tilde{P}''_j \rangle$
            $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
        else ▷ $l_i < minVal$ ***and*** $u_i > maxVal$
            $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge minVal \leq x[i] \leq maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
            $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] > maxVal = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j \rangle$
            $\tilde{\Theta}''_j \leftarrow M * \tilde{\Theta}''_j = \langle M\tilde{c}''_j + s, M\tilde{V}''_j, \tilde{P}''_j \rangle$
            $\tilde{\Theta}'''_j \leftarrow \tilde{\Theta}_j \wedge x[i] < minVal = \langle \tilde{c}'''_j, \tilde{V}'''_j, \tilde{P}'''_j \rangle$
            $\tilde{\Theta}'''_j \leftarrow M * \tilde{\Theta}'''_j = \langle M\tilde{c}'''_j + s', M\tilde{V}'''_j, \tilde{P}'''_j \rangle$
            $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup \tilde{\Theta}''_j \cup \tilde{\Theta}'''_j$
        $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \mathcal{R}_1$
    return $\tilde{\mathcal{R}}$

Over-approximate Analysis

In the previous section, we saw that we mainly deal with three main ranges using the HardTanh function 3.7. Now we want to examine the over-approximation algorithm for HardTanh in FNNs where each layer constructs only one star. In the exact analysis, we discussed six cases. Similar to those cases, our approximation rule is defined as follows.

Lemma 3.1.11. *For any input x_i, the output $y_i = HardTanh(x_i)$, let*

$$\begin{cases} y_i = x_i & l_i \geq minVal \ \wedge \ u_i \leq maxVal \\ y_i = minVal & u_i < minVal \\ y_i = maxVal & l_i > maxVal \end{cases} \tag{3.8}$$

$$\begin{cases} y_i \leq maxVal, \\ y_i \geq -\frac{l_i - maxVal}{u_i - l_i} \cdot x_i - \frac{l_i \cdot (maxVal - u_i)}{u_i - l_i}, & minVal \leq l_i \leq maxVal \ \wedge \ u_i > maxVal \\ y_i \leq x_i \end{cases} \tag{3.9}$$

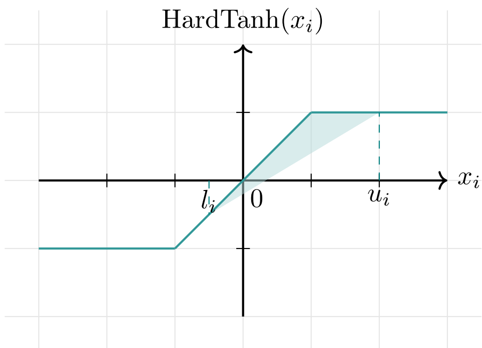


Figure 3.16: Convex relaxation for the HardTanh function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \geq minVal, \\ y_i \leq \frac{u_i - minVal}{u_i - l_i} \cdot x_i - \frac{u_i \cdot (l_i - minVal)}{u_i - l_i}, & l_i < minVal \ \wedge \ u_i \leq maxVal \\ y_i \geq x_i \end{cases} \tag{3.10}$$

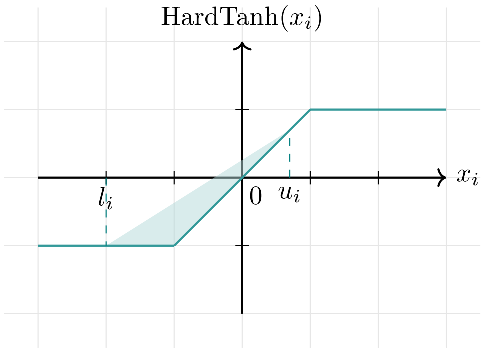


Figure 3.17: Convex relaxation for the HardTanh function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \leq maxVal, \\ y_i \geq minVal, \\ y_i \leq \frac{maxVal - minVal}{maxVal - l_i} \cdot x_i - \frac{maxVal \cdot (l_i - minVal)}{maxVal - l_i}, \\ y_i \geq \frac{minVal - maxVal}{minVal - u_i} \cdot x_i - \frac{minVal \cdot (maxVal - u_i)}{minVal - u_i} \end{cases} \quad l_i < minVal \ \wedge \ u_i > maxVal \tag{3.11}$$

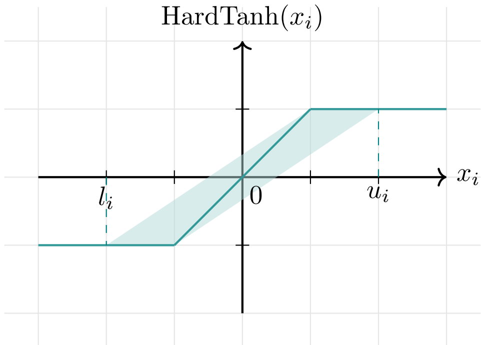


Figure 3.18: Convex relaxation for the HardTanh function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

where l_i *and* u_i *are lower and upper bounds of* x_i

For the over-approximates approach, it is depicted as a triangle when the set is over two ranges. However, when the set covers three ranges, our approach is represented by a parallelogram. Similar to the reachable analyses, we compute the reachable set by the execution of a sequence of n approxHTangent operation of a layer with n neurons.

- If the lower bound l_i is greater than $minVal$, and the upper bound u_i is less than $maxVal$, i.e., the set is between the lower and upper bounds, the function returns a set that is the same as the input set.

- If the upper bound u_i is less than $minVal$, we project the set to $minVal$ by mapping the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ and then changing the i^{th} position in the center c_i to $minVal$.

- In contrast to the second case, we project the set to $maxVal$, since the lower bound l_i is greater than $maxVal$. However, here we also map the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. After that, we set c_i to $maxVal$.

- When the lower bound l_i is between $minVal$ and $maxVal$ and the upper bound u_i is over $maxVal$. To capture the over-approximation at the i^{th} neuron, we introduce a new variable α_{m+1}. As a result, the obtained reachable set has one more variable and three more linear constraints to the predicate $P'(\alpha') \triangleq C'\alpha' \leq d'$ where $\alpha' = [\alpha_1 \ \dots \ \alpha_m \ \alpha_{m+1}]$:

$$\alpha_{m+1} \leq maxVal, \quad \alpha_{m+1} \leq x_i,$$

$$\alpha_{m+1} \geq -\frac{l_i - maxVal}{u_i - l_i} \cdot x_i - \frac{l_i \cdot (maxVal - u_i)}{u_i - l_i}$$

- In the opposite case, when the upper bound u_i is between $minVal$ and $maxVal$ and the lower bound l_i is less than $minVal$, we also introduce a new variable α_{m+1} to the predicate , resulting in three different linear constraints:

$$\alpha_{m+1} \geq minVal, \quad \alpha_{m+1} \geq x_i,$$

$$\alpha_{m+1} \leq \frac{u_i - minVal}{u_i - l_i} \cdot x_i - \frac{u_i \cdot (l_i - minVal)}{u_i - l_i}$$

- If the set is over $minVal$ and $maxVal$, i.e., the lower bound l_i is less than $minVal$, and the upper bound u_i is greater than $maxVal$. We introduce a new variable α_{m+1} to encode the over-approximation at the i^{th} neuron. In addition, the reachable set has one more variable. However, according to Equation 3.1.11, we have four more linear constraints, so the predicate $P'(\alpha') \triangleq C'\alpha' \leq d'$ where $\alpha' = [\alpha_1 \ \dots \ \alpha_m \ \alpha_{m+1}]$ in the reachable set holds:

$$\alpha_{m+1} \geq minVal, \quad \alpha_{m+1} \leq maxVal,$$

$$\alpha_{m+1} \leq \frac{maxVal - minVal}{maxVal - l_i} \cdot x_i - \frac{maxVal \cdot (l_i - minVal)}{maxVal - l_i},$$

$$\alpha_{m+1} \geq \frac{minVal - maxVal}{minVal - u_i} \cdot x_i - \frac{minVal \cdot (maxVal - u_i)}{minVal - u_i}$$

Lemma 3.1.12. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neuron FNN is $N + m_0$ and $4N + n_0$, where respectively m_0 is the number of variables and n_0 the number of linear constraints of the predicate of the input set.*

Example 3.1.4. *Let $\Theta = \langle c, V, P \rangle$ be the input set where:*

$$\text{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\textit{also the predicate } P(\alpha) \triangleq C\alpha \leq d \textit{ where } C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ 3.9 & 1.6 \\ 2 & 2.4 \\ -0.9 & -3.8 \end{bmatrix} \textit{ and } d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$$

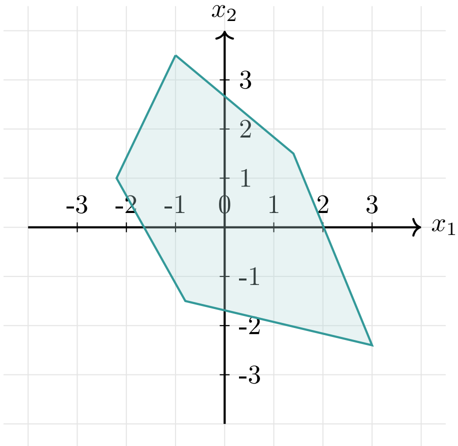


Figure 3.19: The input star set Θ corresponding to Example 3.1.4

We apply the approxHTangent operation on the dimensions of Θ *with* $minVal = -2$ *and* $maxVal = 2$*, resulting in:*

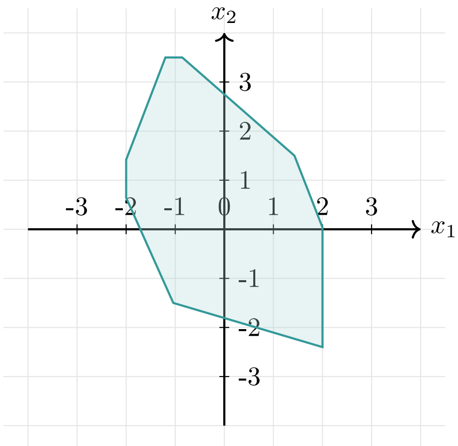


Figure 3.20: Θ' after applying approxHTangent on the first dimension.

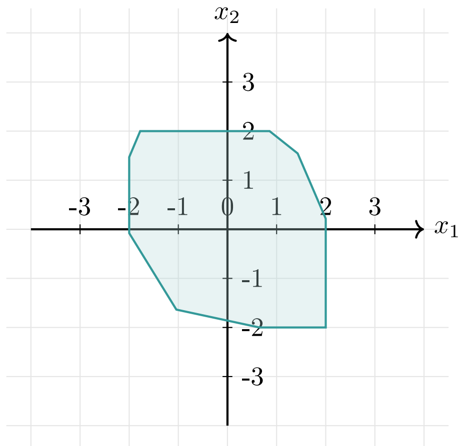


Figure 3.21: Θ'' after applying approxHTangent on the second dimension.

Algorithm 4 Star set based over-approximate reachability analysis for a HardTanh layer

Constants: $minVal, maxVal$
Input: Input star set $I = [\Theta]$
Output: Over-approximate reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $I_1 \leftarrow W * I_0 + b = \langle Wc + b, WV, P \rangle$
  $I' \leftarrow I_1$
  for $i = 1 : n$ **do** ▷ *n approxHTangent operations*
    $I' \leftarrow$ approxHTangent(I', i)
  $\mathcal{R}_1 \leftarrow I'$
procedure APPROXHTANGENT$(\tilde{I}, i)$
  $\tilde{I} \leftarrow \Theta = \langle \tilde{c}, \tilde{V}, \tilde{P} \rangle$
  $[l_i, u_i] \leftarrow \tilde{\Theta}.range(i)$
  $M \leftarrow [e_1 \ \ldots \ e_{i-1} \ 0 \ e_{i+1} \ \ldots \ e_n]$
  $s \leftarrow [0 \ \ldots \ maxVal \ \ldots \ 0]^T$
  $s' \leftarrow [0 \ \ldots \ minVal \ \ldots \ 0]^T$
  if $l_i \geq minVal$ **and** $u_i \leq maxVal$ **then**
    $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j \rangle$
  else if $u_i < minVal$ **then**
    $\tilde{\Theta}_j \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s', M\tilde{V}_j, \tilde{P}_j \rangle$
    $\mathcal{R}_1 \leftarrow \tilde{\Theta}$
  else if $l_i > maxVal$ **then**
    $\tilde{\Theta}_j \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}_j \rangle$
    $\mathcal{R}_1 \leftarrow \tilde{\Theta}$
  else if $minVal \leq l_i \leq maxVal$ **and** $u_i > maxVal$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \ldots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \ldots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable α_{m+1}*
    $C_1 \leftarrow [0 \ldots 0 \ 1]$, $d_1 \leftarrow maxVal$
    $C_2 \leftarrow [\tilde{V}_i \ 1]$, $d_2 \leftarrow \tilde{c}_i$
    $C_3 \leftarrow [-\frac{l_i - maxVal}{u_i - l_i}\tilde{V}_i \ \ -1]$, $d_3 \leftarrow \frac{\tilde{c}_i \cdot (l_i - maxVal) + l_i \cdot (maxVal - u_i)}{u_i - l_i}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m \times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P' \rangle$
  else if $l_i < minVal$ **and** $u_i \leq maxVal$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \ldots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \ldots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable α_{m+1}*
    $C_1 \leftarrow [0 \ldots 0 \ -1]$, $d_1 \leftarrow minVal$
    $C_2 \leftarrow [\tilde{V}_i \ \ -1]$, $d_2 \leftarrow -\tilde{c}_i$
    $C_3 \leftarrow [-\frac{u_i - minVal}{u_i - l_i}\tilde{V}_i \ 1]$, $d_3 \leftarrow \frac{\tilde{c}_i \cdot (u_i - minVal) - u_i \cdot (l_i - minVal)}{u_i - l_i}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m \times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P' \rangle$
  else ▷ *$l_i < minVal$ **and** $u_i > maxVal$*
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \ldots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \ldots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable α_{m+1}*
    $C_1 \leftarrow [0 \ldots 0 \ -1]$, $d_1 \leftarrow minVal$
    $C_2 \leftarrow [0 \ldots 0 \ 1]$, $d_2 \leftarrow maxVal$
    $C_3 \leftarrow [-\frac{maxVal - minVal}{maxVal - l_i}\tilde{V}_i \ 1]$, $d_3 \leftarrow \frac{\tilde{c}_i \cdot (maxVal - minVal) - maxVal \cdot (l_i - minVal)}{maxVal - l_i}$
    $C_4 \leftarrow [\frac{minVal - maxVal}{minVal - u_i}\tilde{V}_i \ -1]$, $d_4 \leftarrow \frac{\tilde{c}_i \cdot (minVal - maxVal) + minVal \cdot (maxVal - u_i)}{u_i - minVal}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m \times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3 \ C_4], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3 \ d_4]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P' \rangle$

Unbounded Analysis

In this section, we investigate the reachability analysis of an unbounded input star set $\Theta = \langle c, V, P \rangle$. For the unboundedness, we consider three cases the same as in 3.1.1.

- In case, the input star set Θ has no upper bound and accordingly increases infinitely without reaching a maximum value. By executing the exaxtHTangent operation to compute an exact reachability set for the input set without an upper bound, in compersion, the exaxtHTangent operation will work the same as with the bounded input star set. However, only three of the six presented cases will occur. First, if the lower bound is greater than $maxVal$, we apply the case of line 22 in Algorithm 3. As we can compute if the lower bound is in the range between $minVal$ and $maxVal$, we can treat the input as presented in Algorithm 3, line 25. Furthermore, if the lower bound is less than $minVal$, we handle the set in a manner equivalent to Algorithm 3, line 35. In contrast, the over-approximative analysis will change, and consequently, the approximation rule as well.

Lemma 3.1.13. *For any input x_i, the output $y_i = HardTanh(x_i)$, let*

$$\begin{cases} y_i \leq maxVal, \\ y_i \geq l_i & minVal \leq l_i \leq maxVal \\ y_i \leq x_i \end{cases} \tag{3.12}$$

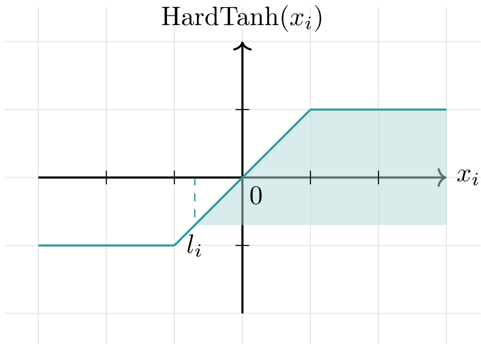


Figure 3.22: Convex relaxation for the HardTanh function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \leq maxVal, \\ y_i \geq minVal, & l_i < minVal \\ y_i \leq \frac{maxVal-minVal}{maxVal-l_i} \cdot x_i - \frac{maxVal\cdot(l_i-minVal)}{maxVal-l_i} \end{cases} \tag{3.13}$$

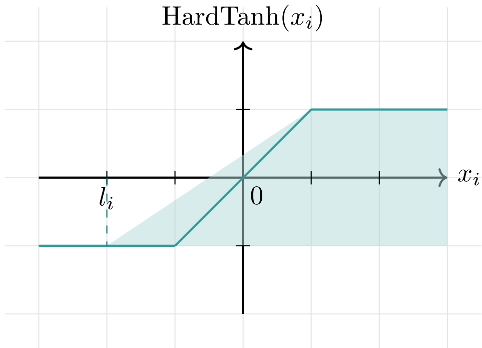


Figure 3.23: Convex relaxation for the HardTanh function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

where l_i is the lower bound of x_i.

We apply approxHTangent on our input set, which will differ between the two cases. First, suppose the lower bound is between $minVal$ and $maxVal$. In that case, we introduce a new variable α_{m+1} to encode the over-approximation of the activation function according to Equation 3.12, which will result in three more linear constraints:

$$\alpha_{m+1} \leq maxVal, \quad \alpha_{m+1} \geq l_i, \quad \alpha_{m+1} \leq x_i$$

The second case is when the lower bound is less than $minVal$. Therefore, the new variable α_{m+1} captures the over-approximation according to Equation 3.13, generating three more linear constraints:

$$\alpha_{m+1} \leq maxVal, \quad \alpha_{m+1} \geq minVal,$$

$$\alpha_{m+1} \leq \frac{maxVal - minVal}{maxVal - l_i} \cdot x_i - \frac{maxVal \cdot (l_i - minVal)}{maxVal - l_i}$$

- The second case is when the input star set Θ has no lower bound, i.e., the input extends infinitely to the negative direction without a minimum limit. Alike the first case, applying the exactHTangent operation would result in the same process as having a bounded input, but only three of the six presented cases will occur. If the upper bound is less than $minVal$, we project the input onto $minVal$, as in Algorithm 4, line 13. If the upper bound is in the range between $minVal$ and $maxVal$, then we handle the set the same as in line 32 in Algorithm 4. Lastly, if our upper bound is greater than $maxVal$, then we apply the case of line 43 Algorithm 4. Regarding the over-approximative analysis, the approximation rule changes as follows.

Lemma 3.1.14. *For any input x_i, the output $y_i = HardTanh(x_i)$, let*

$$\begin{cases} y_i \geq minVal, \\ y_i \leq u_i & minVal \leq u_i \leq maxVal \\ y_i \geq x_i \end{cases} \tag{3.14}$$

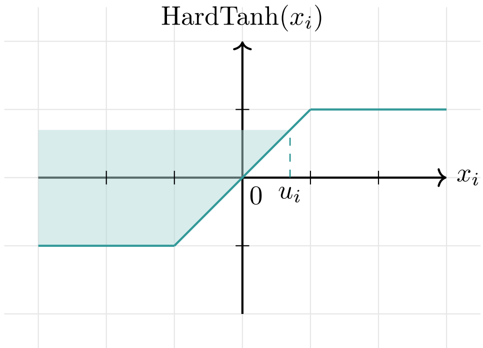


Figure 3.24: Convex relaxation for the HardTanh function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \geq minVal, \\ y_i \leq maxVal, \\ y_i \geq \frac{minVal - maxVal}{minVal - u_i} \cdot x_i - \frac{minVal \cdot (maxVal - u_i)}{minVal - u_i} \end{cases} \quad u_i > maxVal \tag{3.15}$$

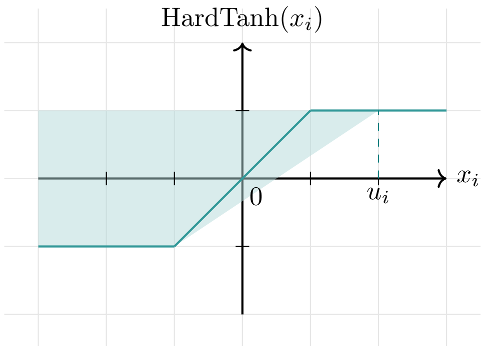


Figure 3.25: Convex relaxation for the HardTanh function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

where u_i is the upper bound of x_i.

If the upper bound is in the range between $minVal$ and $maxVal$, we introduce the new variable α_{m+1} encoding the over-approximation according to Equation 3.14, resulting in three more constraints:

$$\alpha_{m+1} \geq minVal, \quad \alpha_{m+1} \leq u_i, \quad \alpha_{m+1} \geq x_i$$

However, if the upper bound is greater than $maxVal$, the over-approximation is captured by the new variable α_{m+1} according to the Equation 3.15. Hence we have three more constraints:

$$\alpha_{m+1} \geq minVal, \quad \alpha_{m+1} \leq maxVal,$$

$$\alpha_{m+1} \geq \frac{minVal - maxVal}{minVal - u_i} \cdot x_i - \frac{minVal \cdot (maxVal - u_i)}{minVal - u_i}$$

- Lastly, we consider the input with no upper or lower bound, i.e., the input grows infinitely without limits. In this case, the exact analysis also does not differ from the exact analysis with bounded input. Regardless, the over-approximative analysis would change, resulting in the following rule.

  Lemma 3.1.15. *For any input x_i, the output $y_i = HardTanh(x_i)$, let*

  $$\begin{cases} y_i \geq minVal, \; y_i \leq maxVal & x_i \in \mathbb{R} \end{cases} \tag{3.16}$$

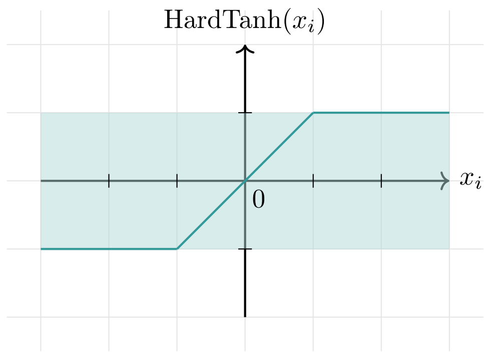


Figure 3.26: Convex relaxation for the HardTanh function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

  Introducing the new variable α_{m+1} to the predicate will result in these two new constraints

  $$\alpha_{m+1} \geq minVal, \quad \alpha_{m+1} \leq maxVal$$

  Regarding the dimension, after applying HardTanh, the obtained star set has any possible value in the codomain.

Lemma 3.1.16. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neurons FNN is $N + m_0$ and $3N + n_0$, where respectively m_0 is the number of variables and n_0 the number of linear constraints of the predicate of the input set.*

3.1.3 Reachability of Hard Sigmoid Layer

The hard sigmoid activation function is a variant of the sigmoid function. It is similar to the Hard Tanh function 3.6.

Definition 3.1.3 (Hard Sigmoid Function (HardSigmoid) [AG21])**.** *Given the input x,*

$$HardSigmoid(x) = \begin{cases} 0 & x \leq -1 \\ 1 & x \geq 1 \\ \frac{1}{2} \cdot x + \frac{1}{2} & -1 < x < 1 \end{cases} = max(0, min(\frac{1}{2} \cdot x + \frac{1}{2})) \tag{3.17}$$

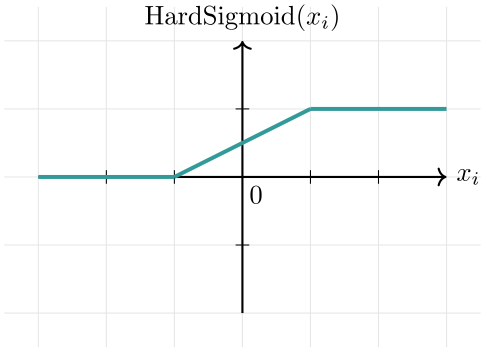


Figure 3.27: Hard sigmoid function (HardSigmoid)

The sigmoid function is known for its nonlinearity and simplicity of computationally inexpensive derivative. Since the function is bound in the range [0,1], it always produces a non-negative value as output. Hence large input changes yield small output changes, thus generating small gradient values. Accordingly, the function suffers from the vanishing gradient problem. However, it does not face the dead neuron problem [Gus22, Dat20]. In our implementation, we decided on the hard sigmoid since it has a lower computation cost (both in software and specialized hardware implementations) and performs excellently in binary classification tasks [CBD15, AG21] as well as since we can verify it, in contrast to the sigmoid.
Considering that the hard sigmoid function has different variants, as in [TF223] and [PT221], we wanted to use a general form of the function definition in our implementation to adjust the function to the own use. Consequently, the function definition is as follows.

$$HardSigmoid(x) = \begin{cases} 0 & x \leq minVal \\ 1 & x \geq maxVal \\ \frac{1}{maxVal - minVal} \cdot x + \frac{minVal}{minVal - maxVal} & minVal < x < maxVal \end{cases} \tag{3.18}$$

We will discuss the analyses in the following sections using this function 3.18.

Exact and Complete Analysis

Given the input $\Theta = \langle c, V, P \rangle$, the core step to compute the reachable set of a layer k is by applying the activation function on the input star set Θ. For a layer with n neurons, we apply a sequence of n exactHSigmoid operation $\mathscr{R}_k = HardSigmoid_n(HardSigmoid_{n-1}(\ldots HardSigmoid_1(\Theta)))$. As described in Algorithm 5, we start by computing the lower and upper bounds l_i, u_i on the i^{th} neuron. The function distinguishes six different cases.

- If the lower and upper bounds between $minVal$ and $maxVal$, we apply the HardSigmoid function on the x_i of the vector $x = [x_1 \ \ldots \ x_n]^T$, leading to a new vector $x' = [x_1 \ \ldots \ x_{i-1} \ \frac{1}{maxVal - minVal} x_i + \frac{minVal}{minVal - maxVal} \ x_{i+1} \ \ldots \ x_n]^T$. This procedure is equivalent to mapping the input set by the scaling matrix $M' = [e_1 \ \ldots \ e_{i-1} \ \frac{1}{maxVal - minVal} \ e_{i+1} \ \ldots \ e_n]$. Afterward, we set the center c_i to $\frac{minVal}{minVal - maxVal}$.

- If the upper bound u_i is less than $minVal$, i.e., $u_i \leq minVal$, we project the input onto zero by mapping the set with $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$.

- If the lower bound l_i exceeds $maxVal$, we project the set onto one. First, we map the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. Then we change the center at the i^{th} position to one, i.e., $c_i = 1$.

- We split the set into two subsets if the lower bound l_i is between $minVal$ and $maxVal$ and the upper bound u_i is greater than $maxVal$.
$\Theta_1 = \Theta \wedge (x_i < maxVal)$, $\Theta_2 = \Theta \wedge (x_i \geq maxVal)$. According to the Definition 3.18, we map $\Theta_1 = \langle c, V, P1 \rangle$ by the scaling matrix
$M' = [e_1 \ \dots \ e_{i-1} \ \frac{1}{maxVal - minVal} \ e_{i+1} \ \dots \ e_n]$ and then change its center c_i to $\frac{minVal}{minVal - maxVal}$. Furthermore, we project $\Theta_2 = \langle c, V, P_2 \rangle$ to 1, as in case 3, first by mapping the Θ_2 by the matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ and then setting the center c_i to 1. Finally, the exactHSigmoid operation at the i^{th} neuron for the input star set Θ results in
$HardSigmoid_i(\Theta) = \langle M'c + s, M'V, P1 \rangle \cup \langle Mc + s', MV, P2 \rangle$, where s, s' are the shifting vectors, $s = [0 \ \dots \ \frac{minVal}{minVal - maxVal} \ \dots \ 0]^T$ and $s' = [0 \ \dots \ 1 \ \dots \ 0]^T$.

- When the lower bound l_i is less than $minVal$, and the upper bound u_i is less than $maxVal$, we split the set into two subsets $\Theta_1 = \Theta \wedge (x_i \leq minVal)$, $\Theta_2 = \Theta \wedge (x_i > minVal)$. By Definition 3.18, the first subset Θ_1 is projected to zero by mapping the set with $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. Θ_2 is in the range between $minVal$ and $maxVal$, so we map it by the scaling matrix
$M' = [e_1 \ \dots \ e_{i-1} \ \frac{1}{maxVal - minVal} \ e_{i+1} \ \dots \ e_n]$ and set c_i to $\frac{minVal}{minVal - maxVal}$.
Consequently, the result of the exactHSigmoid operation of the i^{th} neuron for the input star set Θ is union of two star sets
$HardSigmoid_i(\Theta) = \langle Mc, MV, P1 \rangle \cup \langle M'c + s, M'V, P2 \rangle$, where
$s = [0 \ \dots \ \frac{minVal}{minVal - maxVal} \ \dots \ 0]^T$ is the shifting vector.

- In the last case, we partition the set into three subsets
$\Theta_1 = \Theta \wedge (minVal < x_i < maxVal)$, $\Theta_2 = \Theta \wedge (x_i \leq minVal)$, and
$\Theta_3 = \Theta \wedge (x_i \geq maxVal)$. Similar to the last cases, Θ_1 is scaled according to the Definition 3.18, so first, we map it with the scaling matrix
$M' = [e_1 \ \dots \ e_{i-1} \ \frac{1}{maxVal - minVal} \ e_{i+1} \ \dots \ e_n]$ and adjust c_i to $\frac{minVal}{minVal - maxVal}$.
Θ_2 is projected onto zero, so we map the set with the mapping matrix
$M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$. However, Θ_3 is projected to 1 by mapping the set with $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$, and setting the center c_i to 1. Accordingly, the exactHSigmoid operation at the i^{th} neuron results in a union of three star sets
$HardSigmoid_i(\Theta) = \langle M'c + s, M'V, P1 \rangle \cup \langle Mc, MV, P2 \rangle \cup \langle Mc + s', MV, P3 \rangle$, where $s = [0 \ \dots \ \frac{minVal}{minVal - maxVal} \ \dots \ 0]^T$ and $s' = [0 \ \dots \ 1 \ \dots \ 0]^T$ are the shifting vectors.

Example 3.1.5. *Let $\Theta = \langle c,V,P \rangle$ be the input set, where:*

$$\text{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

also the predicate $P(\alpha) \triangleq C\alpha \leq d$ *where* $C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ 3.9 & 1.6 \\ 2 & 2.4 \\ -0.9 & -3.8 \end{bmatrix}$ *and* $d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$

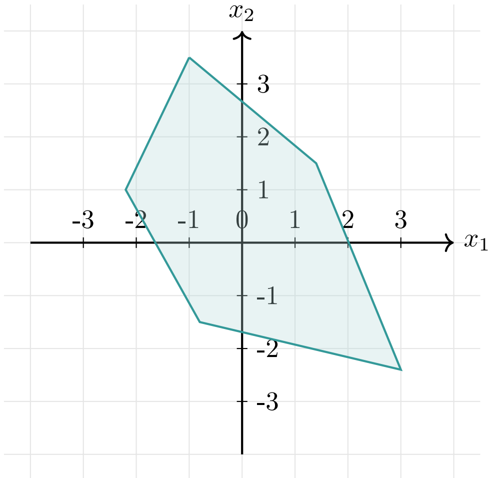


Figure 3.28: The input star set Θ corresponding to Example 3.1.5

We apply the exactHSigmoid operation on the dimensions of Θ *with* $minVal = -2$ *and* $maxVal = 2$*, resulting in:*

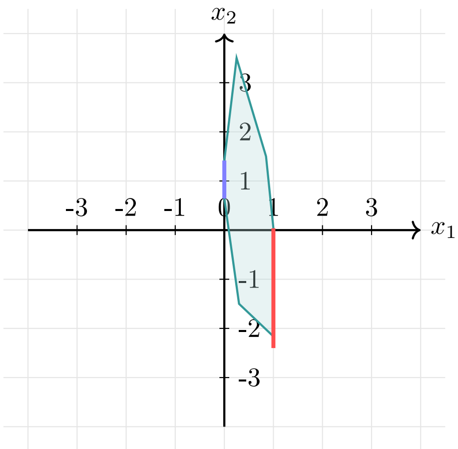


Figure 3.29: Θ' after applying exactHSigmoid on the first dimension.

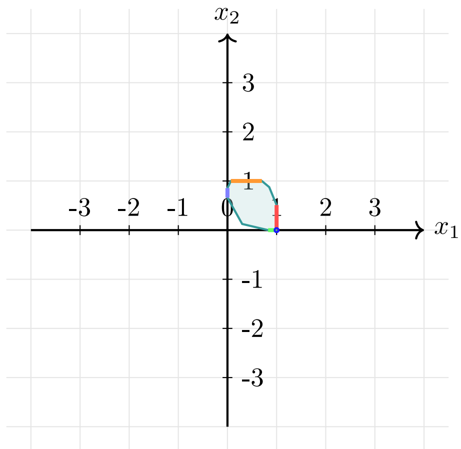


Figure 3.30: Θ'' after applying exactHSigmoid on the second dimension.

Algorithm 5 Star set based exact reachability analysis for a HardSigmoid layer

Constants: $minVal, maxVal$
Input: Input star set $I = [\Theta_1 \dots \Theta_N]$
Output: Exact reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $\mathcal{R} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    $I_1 \leftarrow W * \Theta_j + b = \langle Wc_j + b, WV_j, P_j \rangle$
    $\mathcal{R}_1 \leftarrow I_1$
    for $i = 1 : n$ **do** ▷ *n exactHTangent operations*
      $\mathcal{R}_1 \leftarrow \text{exactHSigmoid}(\mathcal{R}_1, i, l_i, u_i)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$
  return $\mathcal{R}$
procedure EXACTHSIGMOID$(\tilde{I}, i, l_i, u_i)$
  $\tilde{\mathcal{R}} \leftarrow \emptyset$, $\tilde{I} = [\tilde{\Theta}_1, \dots, \tilde{\Theta}_k]$ ▷ *Intermediate representations*
  for $j = 1 : k$ **do**
    $[l_i, u_i] \leftarrow \tilde{\Theta}_j.\text{range}(i)$ ▷ $l_i \leq x_i \leq u_i, x_i \in I_1[i]$
    $\mathcal{R}_1 \leftarrow \emptyset$
    $M \leftarrow [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$
    $M' \leftarrow [e_1 \ \dots \ e_{i-1} \ \frac{1}{maxVal - minVal} \ e_{i+1} \ \dots \ e_n]$
    $s \leftarrow [0 \ \dots \ \frac{minVal}{minVal - maxVal} \ \dots \ 0]^T$
    $s' \leftarrow [0 \ \dots \ 1 \ \dots \ 0]^T$
    if $l_i > minVal$ **and** $u_i < maxVal$ **then**
      $\tilde{\Theta} \leftarrow M' * \tilde{\Theta}_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}_j \rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j$
    else if $u_i \leq minVal$ **then**
      $\mathcal{R}_1 \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j \rangle$
    else if $l_i \geq maxVal$ **then**
      $\tilde{\Theta}_j \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s', M\tilde{V}_j, \tilde{P}_j \rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j$
    else if $minVal < l_i < maxVal$ **and** $u_i \geq maxVal$ **then**
      $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge minVal < x[i] < maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}' \leftarrow M' * \tilde{\Theta}'_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] \geq maxVal = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j \rangle$
      $\tilde{\Theta}''_j \leftarrow M * \tilde{\Theta}''_j = \langle M\tilde{c}''_j + s', M\tilde{V}''_j, \tilde{P}''_j \rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
    else if $l_i \leq minVal$ **and** $u_i < maxVal$ **then**
      $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge minVal < x[i] < maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}' \leftarrow M' * \tilde{\Theta}'_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] < minVal = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j \rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
    else ▷ $l_i \leq minVal$ ***and*** $u_i \geq maxVal$
      $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge minVal < x[i] < maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}' \leftarrow M' * \tilde{\Theta}'_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}'_j \rangle$
      $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] \leq minVal = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j \rangle$
      $\tilde{\Theta}''_j \leftarrow M * \tilde{\Theta}''_j$
      $\tilde{\Theta}'''_j \leftarrow \tilde{\Theta}_j \wedge x[i] \geq maxVal = \langle \tilde{c}'''_j, \tilde{V}'''_j, \tilde{P}'''_j \rangle$
      $\tilde{\Theta}'''_j \leftarrow M * \tilde{\Theta}'''_j = \langle M\tilde{c}'''_j + s', M\tilde{V}'''_j, \tilde{P}'''_j \rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup \tilde{\Theta}''_j \cup \tilde{\Theta}'''_j$
    $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \mathcal{R}_1$
  return $\tilde{\mathcal{R}}$

Lemma 3.1.17. *The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is* $\mathcal{O}(3^N)$*.*

Lemma 3.1.18. *The worst-case complexity of the number of constraints of a star in the reachable set of an N-neurons FNN is* $\mathcal{O}(2N)$*.*

Over-approximate Analysis

In this section, we want to discuss the over-approximation reachability algorithm for HardSigmoid 6. Then, equivalent to the last section, we will distinguish between six cases. Likewise, the other analyses of this over-approximative analysis construct one star at each neuron using the following approximation rule.

Lemma 3.1.19. *Given the input* x_i*, for an output* $y_i = HardSigmoid(x_i)$*, let*

$$\begin{cases} y_i = \frac{1}{maxVal - minVal} \cdot x + \frac{minVal}{minVal - maxVal} & l_i > minVal \ \wedge \ u_i < maxVal \\ y_i = 0 & u_i \leq minVal \\ y_i = 1 & l_i \geq maxVal \end{cases} \tag{3.19}$$

$$\begin{cases} y_i \leq 1, \\ y_i \leq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{minVal - maxVal}, & minVal < l_i < maxVal \ \wedge \\ y_i \geq \frac{l_i - 1}{l_i - u_i} \cdot x_i + \frac{l_i \cdot (1 - u_i)}{l_i - u_i} & u_i \geq maxVal \end{cases} \tag{3.20}$$

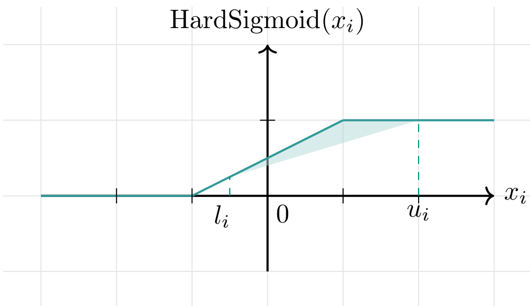


Figure 3.31: Convex relaxation for the HardSigmoid function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \geq 0, \\ y_i \geq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{minVal - maxVal}, & l_i \leq minVal \ \wedge \ u_i < maxVal \\ y_i \leq \frac{u_i \cdot (x_i - l_i)}{u_i - l_i} \end{cases} \tag{3.21}$$

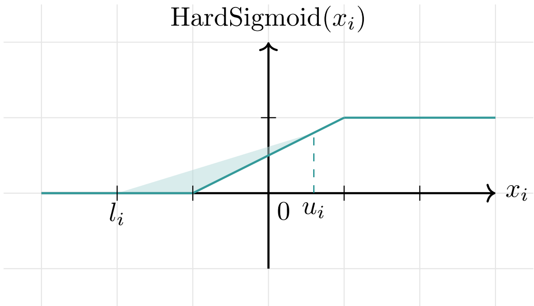


Figure 3.32: Convex relaxation for the HardSigmoid function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \leq 1, & \\ y_i \geq 0, & l_i \leq minVal \ \wedge \ u_i \geq maxVal \\ y_i \leq \frac{1}{maxVal - l_i} \cdot x_i - \frac{l_i}{maxVal - l_i}, & \\ y_i \geq \frac{1}{u_i - minVal} \cdot x_i - \frac{minVal}{u_i - minVal} & \end{cases} \tag{3.22}$$

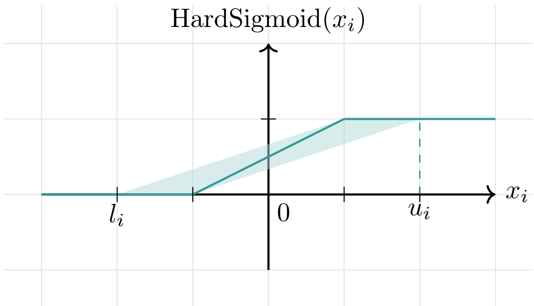


Figure 3.33: Convex relaxation for the HardSigmoid function. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

where l_i and u_i are lower and upper bounds of x_i.

We compute the reachable set by executing a sequence of n approxHSigmoid operations for a layer with n neurons. Given an input star set Θ, we determine the input's lower and upper bounds l_i,u_i at the i^{th} neuron. We differentiate six cases.

- Equivalent to the exact analysis, if the lower bound l_i and upper bound u_i between $minVal$ and $maxVal$, we scale the input set by the scaling matrix $M' = [e_1 \ \ldots \ e_{i-1} \ \frac{1}{maxVal - minVal} \ e_{i+1} \ \ldots \ e_n]$, then adjust the center $c_i = \frac{minVal}{minVal - maxVal}$.
- If the upper bound u_i is less than $minVal$, we project the input set onto zero by mapping the set by the mapping matrix $M = [e_1 \ \ldots \ e_{i-1} \ 0 \ e_{i+1} \ \ldots \ e_n]$.

- If the lower bound l_i is greater than $maxVal$, we project the input set onto one by mapping the set by the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ and afterward adjust c_i to 1.

- If the lower bound l_i is in the range between $minVal$ and $maxVal$ and the upper bound u_i is over $maxVal$, to encode the over-approximation at the i^{th} neuron, we introduce α_{m+1}, which result in three new linear constraints:

$$\alpha_{m+1} \leq 1, \quad \alpha_{m+1} \leq \frac{1}{maxVal - minVal} \cdot x_i - \frac{minVal}{maxVal - minVal}$$

$$\alpha_{m+1} \geq \frac{l_i - 1}{l_i - u_i} \cdot x_i + \frac{l_i \cdot (1 - u_i)}{l_i - u_i}$$

- In contrast to the previous case, in this case, we consider the upper bound u_i is between $minVal$ and $maxVal$, and the lower bound l_i is less than $minVal$. We introduce a new variable α_{m+1} to capture the over-approximation. Correspondingly we have three new constraints:

$$\alpha_{m+1} \geq 0, \quad \alpha_{m+1} \geq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{maxVal - minVal}$$

$$\alpha_{m+1} \leq \frac{u_i \cdot (x_i - l_i)}{u_i - l_i}$$

- Otherwise, since the upper bound u_i is over $maxVal$ and lower bound l_i less than $minVal$, we introduce a new variable α_{m+1}. As a result, the obtained reachable set has four more linear constraints:

$$\alpha_{m+1} \leq 1, \quad \alpha_{m+1} \geq 0,$$

$$\alpha_{m+1} \leq \frac{1}{maxVal - l_i} \cdot x_i - \frac{l_i}{maxVal - l_i},$$

$$\alpha_{m+1} \geq \frac{1}{minVal - u_i} \cdot x_i - \frac{minVal}{u_i - minVal}$$

Lemma 3.1.20. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neurons FNN is $N + m_0$ and $4N + n_0$, where respectively m_0 is the number of variables and n_0 the number of linear constraints of the predicate of the input set.*

Example 3.1.6. *Let $\Theta = \langle c, V, P\rangle$ be the input set where:*

$$\textit{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \textit{ and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\textit{also the predicate } P(\alpha) \triangleq C\alpha \leq d \textit{ where } C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ 3.9 & 1.6 \\ 2 & 2.4 \\ -0.9 & -3.8 \end{bmatrix} \textit{ and } d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$$

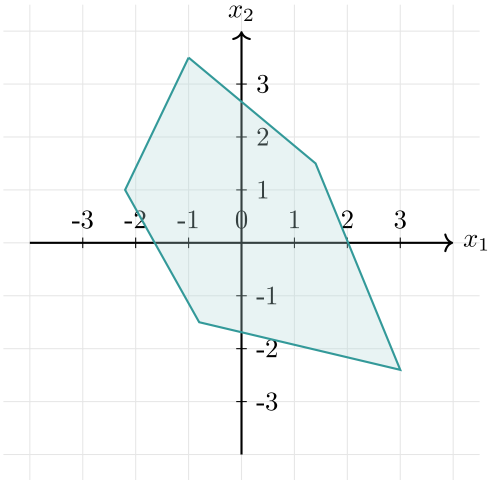


Figure 3.34: The input star set Θ corresponding to Example 3.1.6

We apply the approxHSigmoid operation on the dimensions of Θ *with* $minVal = -2$ *and* $maxVal = 2$*, resulting in:*

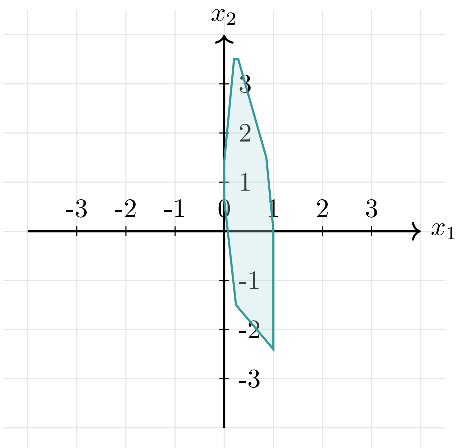


Figure 3.35: Θ' after applying approxHSigmoid on the first dimension.

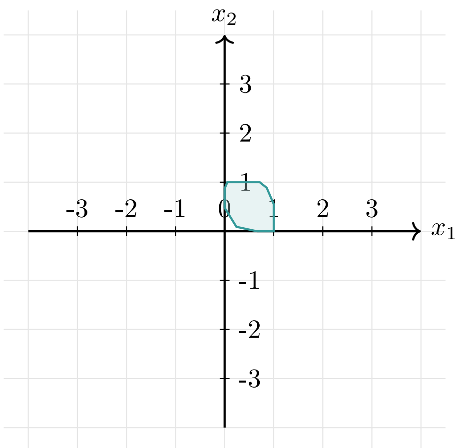


Figure 3.36: Θ'' after applying approxHSigmoid on the second dimension.

Algorithm 6 Star set based over-approximate reachability analysis for a HardSigmoid layer

Constants: $minVal, maxVal$
Input: Input star set $I = [\Theta]$
Output: Over-approximate reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $I_1 \leftarrow W * I_0 + b = \langle Wc + b, WV, P\rangle$
  $I' \leftarrow I_1$
  for $i = 1 : n$ **do** ▷ *n approxHSigmoid operations*
    $I' \leftarrow$ approxHSigmoid(I',i)
  $\mathcal{R}_1 \leftarrow I'$
procedure APPROXHSIGMOID$(\tilde{I}, i)$
  $\tilde{I} \leftarrow \Theta = \langle \tilde{c}, \tilde{V}, \tilde{P}\rangle$
  $[l_i,u_i] \leftarrow \tilde{\Theta}.range(i)$
  $M \leftarrow [e_1 \ \ldots \ e_{i-1} \ 0 \ e_{i+1} \ \ldots \ e_n]$
  $M' \leftarrow [e_1 \ \ldots \ e_{i-1} \ \frac{1}{maxVal-minVal} \ e_{i+1} \ \ldots \ e_n]$
  $s \leftarrow [0 \ \ldots \ \frac{minVal}{minVal-maxVal} \ \ldots \ 0]^T$, $s' \leftarrow [0 \ \ldots \ 1 \ \ldots \ 0]^T$
  if $l_i > minVal$ **and** $u_i < maxVal$ **then**
    $\tilde{\Theta} \leftarrow M' * \tilde{\Theta}_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}_j\rangle$
    $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j$
  else if $u_i \leq minVal$ **then**
    $\mathcal{R}_1 \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j\rangle$
  else if $l_i \geq maxVal$ **then**
    $\tilde{\Theta}_j \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s', M\tilde{V}_j, \tilde{P}_j\rangle$
    $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j$
  else if $minVal < l_i < maxVal$ **and** $u_i \geq maxVal$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \ldots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \ldots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable α_{m+1}*
    $C_1 \leftarrow [0 \ldots 0 \ 1]$, $d_1 \leftarrow 1$
    $C_2 \leftarrow [\frac{-1}{maxVal-minVal}\tilde{V}_i \ 1]$, $d_2 \leftarrow \frac{\tilde{c}_i-minVal}{maxVal-minVal}$
    $C_3 \leftarrow [\frac{l_i-1}{l_i-u_i}\tilde{V}_i \ -1]$, $d_3 \leftarrow -\frac{\tilde{c}_i\cdot(l_i-1)+l_i\cdot(1-u_i)}{l_i-u_i}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m\times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P'\rangle$
  else if $l_i \leq minVal$ **and** $u_i < maxVal$ **then**
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \ldots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \ldots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable α_{m+1}*
    $C_1 \leftarrow [0 \ldots 0 \ -1]$, $d_1 \leftarrow 0$
    $C_2 \leftarrow [\frac{-1}{maxVal-minVal}\tilde{V}_i \ -1]$, $d_2 \leftarrow -\frac{\tilde{c}_i-minVal}{maxVal-minVal}$
    $C_3 \leftarrow [-\frac{u_i}{u_i-l_i}\tilde{V}_i \ 1]$, $d_3 \leftarrow \frac{u_i\cdot(\tilde{c}_i-l_i)}{u_i-l_i}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m\times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P'\rangle$
  else ▷ $l_i \leq minVal$ ***and*** $u_i \geq maxVal$
    $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}, \alpha = [\alpha_1 \ \ldots \ \alpha_m]^T$
    $\alpha' \leftarrow [\alpha_1 \ \ldots \ \alpha_m \ \alpha_{m+1}]^T$ ▷ *New variable α_{m+1}*
    $C_1 \leftarrow [0 \ldots 0 \ -1]$, $d_1 \leftarrow 0$
    $C_2 \leftarrow [0 \ldots 0 \ 1]$, $d_2 \leftarrow 1$
    $C_3 \leftarrow [-\frac{-1}{maxVal-l_i}\tilde{V}_i \ 1]$, $d_3 \leftarrow \frac{\tilde{c}_i-l_i}{maxVal-l_i}$
    $C_4 \leftarrow [\frac{1}{u_i-minVal}\tilde{V}_i \ -1]$, $d_4 \leftarrow -\frac{\tilde{c}_i-minVal}{u_i-minVal}$
    $C_0 \leftarrow [\tilde{C} \ 0_{m\times 1}], d_0 \leftarrow \tilde{d}$
    $C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3 \ C_4], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3 \ d_4]$
    $P'(\alpha') \triangleq C'\alpha' \leq d'$
    $c' \leftarrow M\tilde{c}$, $V' \leftarrow M\tilde{V}$, $V' \leftarrow [V' \ e_i]$
    $\tilde{\mathcal{R}} \leftarrow \langle c', V', P'\rangle$

Unbounded Analysis

In this section, we present the exact and over-approximative analysis for an unbounded input star set $\Theta = \langle c, V, P \rangle$. Therefore we consider three cases of unboundedness.

- The first case is when the input set has no upper bound, only a lower one, i.e., the input grows infinitely into the positive range. Applying the exactHSigmoid operation will not differ from using the operation on a bounded input set. But only three of the six cases from the exact reachability analysis will appear after computing the lower bound. If the lower bound is greater than $maxVal$, we project the set onto one as in line 24, Algorithm 5. If the lower bound is between minVal and maxVal, we handle the set the same in Algorithm 5, line 27. We compute the last case in line 38 when the lower bound is less than $minVal$. For the over-approximative analysis, the approximation rule is changed to as follows.

  Lemma 3.1.21. *For any input x_i, the output $y_i = HardSigmoid(x_i)$, let*

$$\begin{cases} y_i \leq 1, \\ y_i \geq \frac{1}{maxVal - minVal} \cdot l_i + \frac{minVal}{min - maxVal}, & minVal < l_i < maxVal \\ y_i \leq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{min - maxVal} \end{cases} \tag{3.23}$$

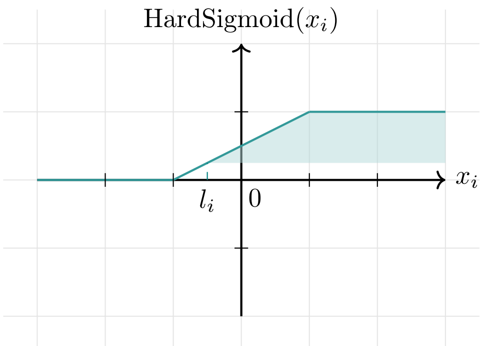


Figure 3.37: Convex relaxation for the HardSigmoid function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \leq 1, \\ y_i \geq 0, & l_i \leq minVal \\ y_i \leq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{min - maxVal} \end{cases} \tag{3.24}$$

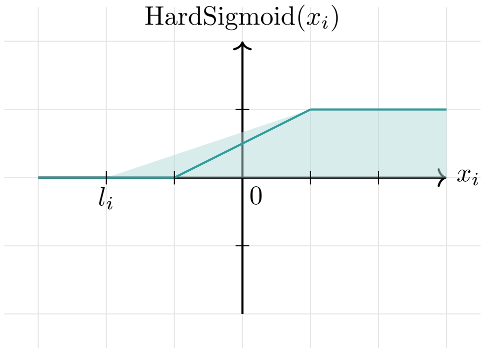


Figure 3.38: Convex relaxation for the HardSigmoid function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$ and $maxVal = 1$.

where l_i is the lower bound of x_i.

If the lower bound is within the range between $minVal$ and $maxVal$, we introduce a new variable denoted as α_{m+1} to represent the over-approximation as specified in Equation 3.23. Correspondingly, the predicate will have three more constraints:

$$\alpha_{m+1} \leq 1, \quad \alpha_{m+1} \geq \frac{1}{maxVal - minVal} \cdot l_i + \frac{minVal}{min - maxVal},$$

$$\alpha_{m+1} \leq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{min - maxVal}$$

In case the lower bound is less than $maxVal$, we introduce the new variable $\alpha_{m+1}$to capture the over-approximation, resulting in three new constraints:

$$\alpha_{m+1} \leq 1, \quad \alpha_{m+1} \geq 0$$

$$\alpha_{m+1} \leq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{min - maxVal}$$

- The second case occurs when the input star set has an upper but no lower bound and extends infinitely into the negative range. Applying the exactHSigmoid operation on the input results in the same process as for a bounded input. However, three of the six cases will occur. If the upper bound is less than $minVal$, we project the set onto zero as in line 22, Algorithm 5. If the upper bound is between $minVal$ and $maxVal$, we treat the set the same as in line 33 in Algorithm 5. In case the upper bound is greater than $maxVal$, we apply the case of line 38. In difference from the exact analysis, the over-approximative analysis will change with an unbounded input. Accordingly, the approximation rule adjusts to the following rule.

  Lemma 3.1.22. *For any input x_i, the output $y_i = HardSigmoid(x_i)$, let*

$$\begin{cases} y_i \geq 0, \\ y_i \leq \frac{1}{maxVal - minVal} \cdot u_i + \frac{minVal}{min - maxVal} \\ y_i \geq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{min - maxVal} \end{cases} \quad minVal < u_i < maxVal \tag{3.25}$$

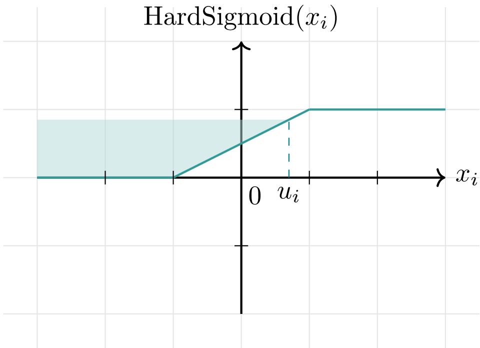


Figure 3.39: Convex relaxation for the HardSigmoid function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

$$\begin{cases} y_i \geq 0, \\ y_i \leq 1, \\ y_i \geq \frac{minVal - maxVal}{u_i - minVal} \cdot x_i - \frac{minVal \cdot (maxVal - u_i)}{u_i - minVal} \end{cases} \qquad u_i \geq maxVal \tag{3.26}$$

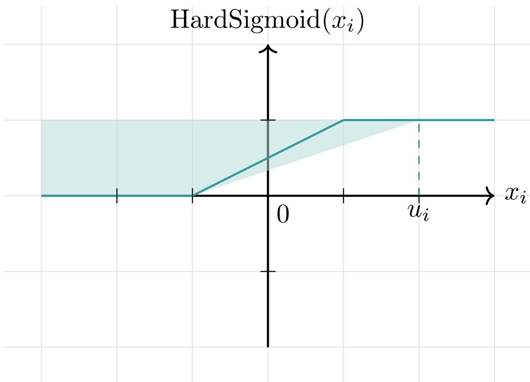


Figure 3.40: Convex relaxation for the HardSigmoid function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

where u_i is the upper bound of x_i.

When the upper bound is between the $minVal$ and $maxVal$, we introduce a new variable α_{m+1} encoding the over-approximation according to equation 3, resulting in three new constraints:

$$\alpha_{m+1} \geq 0, \quad \alpha_{m+1} \leq \frac{1}{maxVal - minVal} \cdot u_i + \frac{minVal}{min - maxVal}$$

$$\alpha_{m+1} \geq \frac{1}{maxVal - minVal} \cdot x_i + \frac{minVal}{min - maxVal}$$

If the upper bound exceeds $maxVal$, we introduce a new variable α_{m+1} to represent the over-approximation. Coordinately, we have three new constraints:

$$y_i \geq 0, \quad y_i \leq 1, \quad y_i \geq \frac{minVal - maxVal}{u_i - minVal} \cdot x_i - \frac{minVal \cdot (maxVal - u_i)}{u_i - minVal}$$

- The last case is when the input grows infinitely into the positive and negative regions, i.e., it has no upper or lower bound. The exact analysis is the same as with the bounded input, but the only case that occurs is the same as in Algorithm 5, line 38. However, the over-approximative rule will adjust to this case, resulting in the following rule.

Lemma 3.1.23. *For any input x_i, the output $y_i = HardSigmoid(x_i)$, let*

$$\left\{ y_i \geq 0, \; y_i \leq 1 \quad x_i \in \mathbb{R} \right. \tag{3.27}$$

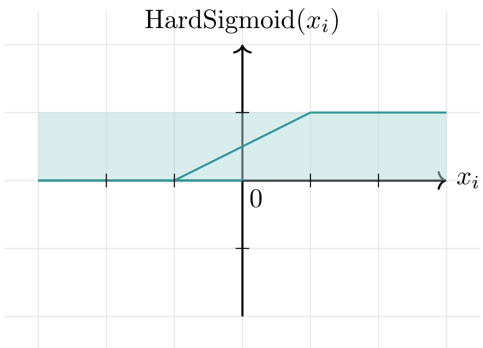


Figure 3.41: Convex relaxation for the HardSigmoid function, in the unbounded case. The dark line represents the exact set (non-convex) and the light area the approximate set (convex and linear). In the figure, $minVal = -1$, $maxVal = 1$.

Since the input star set does not have an upper or lower bound, by introducing a new variable α_{m+1}, two new constraints added to the predicate:

$$\alpha_{m+1} \geq 0, \quad \alpha_{m+1} \leq 1$$

Regarding the dimension, after applying HardSigmoid, the obtained star set has any possible value in the codomain.

Lemma 3.1.24. *The worst-case complexity of the number of variables and constraints in the reachable set of an N-neurons FNN is $N + m_0$ and $3N + n_0$, respectively, where m_0 is the number of variables and n_0 the number of linear constraints of the predicate of the input set.*

3.1.4 Reachability of Unit Step Function Layer

The unit step activation function, also called as the heaviside function is defined as follows

Definition 3.1.4 (Unit step function [GML+08])**.** *For the input x, let*

$$unitStep(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{3.28}$$

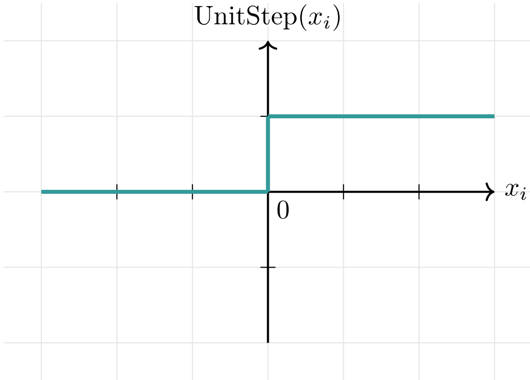


Figure 3.42: Unit step function (UnitStep)

The step function is a straightforward activation function. It takes the input and produces a single-bit output. Accordingly, it is helpful for linear separation between two classes. In our implementation, we needed it to help us round the reachable sets from our previous layers to specific values, enabling a straightforward representation of the reachable set. This makes the verification of neural networks easier. Consequently, we have generalized the function's definition to adapt the function to our use.

$$unitStep(x) = \begin{cases} minRes & x < val \\ maxRes & x \geq val \end{cases} \tag{3.29}$$

where val is the value that serves as the separator for our values, $minRes$ and $maxRes$ are the upper and lower limits for our results.

Exact and Complete Analysis

Given the input $\Theta = \langle c, V, P \rangle$. The exactUStep operation on the i^{th} neuron, i.e., $unitStep_i(\cdot)$, works as follows. We compute the lower and upper bounds l_i, u_i. The function tackles three cases:

- If the upper bound u_i is less than val, we project the set onto $minRes$ by mapping the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ and then setting c_i to $minRes$.
- In the opposite case, when the lower bound l_i is greater than val, we project the set onto $maxRes$ by mapping the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$ and set c_i to $maxRes$.
- Otherwise, we partition the input into two subsets $\Theta_1 = \Theta \wedge (x_i < val)$ and $\Theta_2 = \Theta \wedge (x_i \geq val)$. By definition 3.29, for Θ_1, we project the set onto $minRes$ the same way in the first case, first by mapping the set with the mapping matrix $M = [e_1 \ \dots \ e_{i-1} \ 0 \ e_{i+1} \ \dots \ e_n]$, then adjusting c_i to $minRes$. For

Θ_2, we project it onto $maxRes$ by mapping the set with the mapping matrix $M = [e_1 \; \dots \; e_{i-1} \; 0 \; e_{i+1} \; \dots \; e_n]$ and changing c_i to $maxRes$. Accordingly, the exactUStep operation at the i^{th} neuron for Θ as input yields the union of two star sets $unitStep_i(\Theta) = \langle Mc + s, MV, P1\rangle \cup \langle Mc + s', MV, P2\rangle$, where s, s' are the shifting vectors, $s = [0 \; \dots \; minRes \; \dots \; 0]^T$ and $s' = [0 \; \dots \; maxRes \; \dots \; 0]^T$.

Lemma 3.1.25. *The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is* $\mathcal{O}(2^N)$.

Lemma 3.1.26. *The worst-case complexity of the number of constraints of a star in the reachable set of an N-neurons FNN is* $\mathcal{O}(N)$.

Algorithm 7 Star set based exact reachability analysis for a unit step function layer

Constants: $val, minRes, maxRes$
Input: Input star set $I = [\Theta_1 \dots \Theta_N]$
Output: Exact reachable set $\mathcal{R}$

procedure LAYERREACH(I, W, b)
  $\mathcal{R} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    $I_1 \leftarrow W * \Theta_j + b = \langle Wc_j + b, WV_j, P_j\rangle$
    $\mathcal{R}_1 \leftarrow I_1$
    for $i = 1 : n$ **do** ▷ *n exactUStep operations*
      $\mathcal{R}_1 \leftarrow \text{exactUstep}(\mathcal{R}_1, i, l_i, u_i)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$
  return $\mathcal{R}$
procedure EXACTUSTEP$(\tilde{I}, i, l_i, u_i)$
  $\tilde{\mathcal{R}} \leftarrow \emptyset$, $\tilde{I} = [\tilde{\Theta}_1, \dots, \tilde{\Theta}_k]$ ▷ *Intermediate representations*
  for $j = 1 : k$ **do**
    $[l_i, u_i] \leftarrow \tilde{\Theta}_j.\text{range}(i)$ ▷ $l_i \le x_i \le u_i, x_i \in I_1[i]$
    $\mathcal{R}_1 \leftarrow \emptyset$, $M \leftarrow [e_1 \; \dots \; e_{i-1} \; 0 \; e_{i+1} \; \dots \; e_n]$
    $s \leftarrow [0 \; \dots \; minRes \; \dots \; 0]^T$
    $s' \leftarrow [0 \; \dots \; maxRes \; \dots \; 0]^T$
    if $u_i \le val$ **then**
      $\mathcal{R}_1 \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s, M\tilde{V}_j, \tilde{P}_j\rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j$
    else if $l_i \ge val$ **then**
      $\mathcal{R}_1 \leftarrow M * \tilde{\Theta}_j = \langle M\tilde{c}_j + s', M\tilde{V}_j, \tilde{P}_j\rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j$
    else ▷ $l_i < val$ ***and*** $u_i \ge val$
      $\tilde{\Theta}'_j \leftarrow \tilde{\Theta}_j \wedge x[i] < val = \langle \tilde{c}'_j, \tilde{V}'_j, \tilde{P}'_j\rangle$
      $\tilde{\Theta}'_j \leftarrow M * \tilde{\Theta}'_j = \langle M\tilde{c}'_j + s, M\tilde{V}'_j, \tilde{P}'_j\rangle$
      $\tilde{\Theta}''_j \leftarrow \tilde{\Theta}_j \wedge x[i] \ge val = \langle \tilde{c}''_j, \tilde{V}''_j, \tilde{P}''_j\rangle$
      $\tilde{\Theta}''_j \leftarrow M * \tilde{\Theta}''_j = \langle M\tilde{c}''_j + s', M\tilde{V}''_j, \tilde{P}''_j\rangle$
      $\mathcal{R}_1 \leftarrow \tilde{\Theta}'_j \cup \tilde{\Theta}''_j$
    $\tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \mathcal{R}_1$
  return $\tilde{\mathcal{R}}$

3.2 Non-Fully Connected Layers

In the previous section, we learned about different types of piecewise-linear activation functions and their reachability analyses. In this section, we want to discuss non-fully connected layer types also used frequently in neural networks. However, they were not implemented in this thesis since the implementation was outside the scope of this work.

Definition 3.2.1 (ImageStar [LW20])**.** *An ImageStar Θ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^{h\times w\times nc}$ is the anchor image, $V = \{v_1, \ldots, v_m\} \subseteq \mathbb{R}^{h\times w\times nc}$ is a set of m images called generator images, and $P : \mathbb{R}^m \to \{\top, \bot\}$ is a predicate, and h, w, nc are the height, width and number of channels of the images respectively. The generator images are arranged to form the ImageStar's $h \times w \times nc \times m$ basis array. The set of images represented by the ImageStar is given as:*

$$[\![\Theta]\!] = \{x \mid x = c + \sum_{i=1}^{m} \alpha_i v_i \text{ such that } P(\alpha_1, \ldots, \alpha_m) = \top\}. \tag{3.30}$$

3.2.1 Convolutional Layer

An n-dimensional convolutional layer consists of the weights $W \in \mathbb{R}^{h_f\times w_f\times nc\times nf}$, the bias $b \in \mathbb{R}^{1\times 1\times nf}$, the filter or kernels, the padding size, the stride, and the dilation where h_f, w_f, nc are the height, width, and the number of channels of the filters in the layer. Furthermore, nf is the number of filters [LW20, ON15, AAS20].

- The filter, also known as the learnable kernels, is a small matrix that detects certain features in the input. The filter or kernels size must be specified for each layer. The filter convolves over the image input. Then, as it slides over the input till it parses the complete width, it calculates the scalar product and, in the end, sums with the bias to give us a squashed one-depth convoluted feature output.

- The padding size describes the amount of padding applied to the input. It involves adding extra rows and columns, mostly of zeros, around the input image before performing convolution. By adding the padding to the input, we control the reduction in spatial dimensions during convolution, resulting in better preservation of spatial information.

- The stride sets the distance by which the filter shifts across the input. Therefore, the larger the stride is, the smaller the output. Hence, smaller strides are used for better results.

- The dilation controls the spacing between the kernel points, so we increase the skipped input units by increasing the dilation. It cheaply increases the output units without increasing the kernel size.

A convolutional layer is the core layer of a convolutional neural network. It is mainly used in processing and analyzing structured grid-like data, such as images. By Definition 3.2.1, we saw that ImageStar is an extension of the star set. Moreover, a convolutional layer can process ImageStar. Therefore, the following lemma presents the reachability of a convolutional layer.

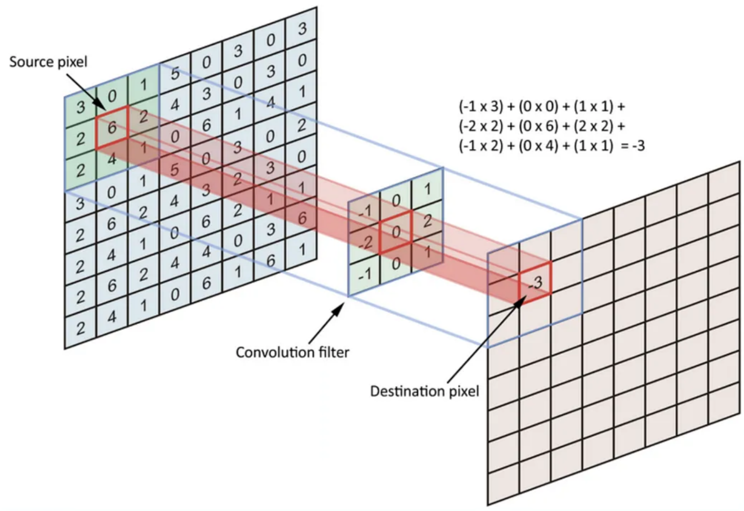


Figure 3.43: The convolution operation [Ste19]

Lemma 3.2.1 (Reachable set of a convolutional layer [Tra20])**.** *Given an ImageStar as input $\mathcal{I} = \langle c, V, P \rangle$, the reachable set of a convolutional layer is another ImageStar $\mathcal{I} = \langle c', V', P' \rangle$ where $c' = Convol(c)$ is the convolution operation applied to the anchor image, $V' = \{v'_1, \ldots, v'_m\}, v'_i = ConvolZeroBias(v_i)$ is the convolution operation with zero bias applied to the generator images, i.e., only using the weights of the layer.*

Convolutional layers have many advantages, by applying filters to small input data regions, it enables localized feature extraction [AAS20]. Using shared weights in convolutional layers reduces the number of learnable parameters compared to fully connected layers. In addition, this parameter sharing allows convolutional neural networks to efficiently learn and generalize from data by reusing learned features across different input regions [YNDT18].

After researching the convolutional layer to be able to implement the described layer to verify different types of neural networks in Hypro, since the convolutional layer takes images, i.e., ImageStar as input, we concluded that implementing both ImageStars and the analysis of the convoltuional layer is outside of the scope of this bachelor's thesis.

3.2.2 Pooling Layer

The pooling layer is another layer type in a convolutional neural network. It aims to reduce the input volume's dimensions (width and height) to reduce the complexity for further layers while maintaining the essential features. There are many pooling layers, but the most common are max and min pooling and average pooling [GK20, ON15, AMAZ17].

- The max pooling also divides the input into smaller regions called pooling regions and returns each region's maximum value. This layer helps preserve the most important features while reducing the dimensionality of the input. Min pooling works similarly to max pooling, but instead of returning the region's maximum value, it returns the minimum value for each region.

- The average pooling layer partitions the input into pooling regions and computes the average values of each region. It provides a smoother downsampling compared to max pooling.

Since pooling layers also process images stores as input and, in our case, ImageStar, implementing this layer was was beyond this project's scope.

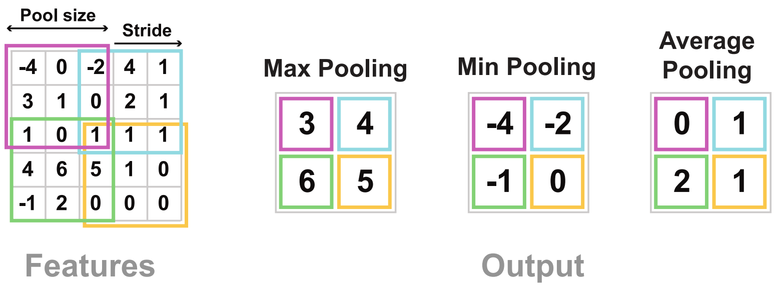


Figure 3.44: Pooling Operation (max, min, and average pooling) [MS21].

3.2.3 Residual Layer

A residual layer, also known as a residual block, is an essential building block in deep neural networks. Residual layers address the degradation problem. The degradation problem refers to the issue that arises when deep neural networks with a large number of layers are trained. As the network's depth increases, the training set's performance begins to saturate and then degrades, even though the network capacity is theoretically increasing [HZRS16, Yad22]. In a residual layer, the input goes through convolutional layers, activation functions, and other operations. The output of these operations is then added to the original input, which creates a residual connection. This connection allows the network to learn the residual or the difference between the input and the desired output rather than learning the entire mapping from scratch. If the residual is close to zero, the layer can effectively pass the input through without adding much distortion [HZRS16, Nan17]. This layer implementation was outside the scope of this work. However, the research took place to simplify the implementation for the future.

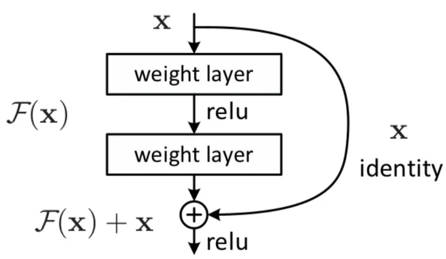


Figure 3.45: Residual learning: a building block [HZRS16]

3.2.4 Recurrent Layer

A recurrent layer is commonly used in recurrent neural networks for processing sequential or time-series data. Within a recurrent layer, a hidden state acts as a memory that stores information about past inputs in the sequence. At each time step, the recurrent layer takes the current input and the previous hidden state as inputs, producing an output and a new hidden state. This operation is repeated for each element in the sequence, allowing the layer to consider the contextual information from prior inputs. The key feature of a recurrent layer is its capacity to share weights across various time steps, allowing it to learn and capture temporal dependencies [PGCB14, Nab19]. Therefore, like the previous section, The implementation of this layer was beyond this project's scope, but still, the research took place.

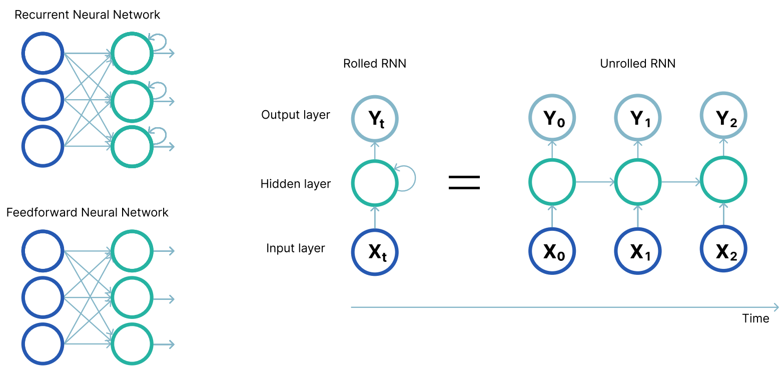


Figure 3.46: Recurrent neural network [IBM]

3.3 ONNX Parser

Open Neural Network Exchange, abbreviated as ONNX, is an open format that allows for the interchange of deep learning models between different frameworks. The ONNX file format is a standard for representing deep learning models [ONN]. An ONNX file contains a serialized representation of a trained neural network model. In addition, this file includes the model's architecture, weights, and computational graph, which makes it usable for inference in different frameworks or deployment scenarios. Until now, Hypro only supports the nnet file format to load models of neural networks [Sta21]. The nnet file format is text-based for feed-forward, fully connected, ReLU-activated neural networks. However, as we implemented activation functions other than ReLU, we needed another format than the nnet file format to build and work with the different model's architecture. Therefore, we implemented the ONNX Parser to be able to read and interpret models in ONNX format in Hypro. ONNX model format uses Protocol Buffers (protobuf) as the underlying serialization format, so the ONNX model files are stored in the Protobuf binary format. Accordingly, we integrated Protobuf library in Hypro. Afterward, we implemented an adapter to read the ONNX model's attributes and save it as an object to work with it in Hypro. So as a user, to be able to use this implementation, it is required to install Protobuf.

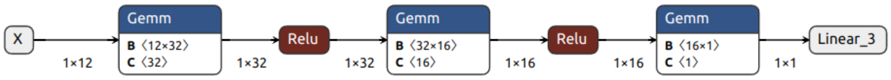


Figure 3.47: A neural network represented with ONNX format

Chapter 4

Evaluation

In this section, we re-evaluate different benchmarks using our implemented star based reachability algorithms, the exact and the over-approximation approach, using Hypro. We start with the run time and safety verification of the ACAS Xu DNNs, move to the thermostat benchmark afterward, and finally, a benchmark based on a case study about autonomous drone control. Lastly, we evaluate the robustness of the binary classification of sonar data.

The evaluations were run on a machine with Intel Xeon Platinum 8160 Processors "SkyLake" (2.1 GHz, 24 cores each) and 16 GB RAM.

4.1 Benchmarks

4.1.1 ACAS Xu

The Airborne Collision Avoidance System Xu (ACAS Xu) is a mid-air collision avoidance system focusing on unmanned aircraft. The ACAS Xu networks (ACAS Xu DNNs) provide advisories for horizontal maneuvers to avoid collisions while minimizing unnecessary alerts. It is a set of 45 feedforward neural networks, each with seven fully connected layers, comprising a combined count of 300 neurons. The networks possess five inputs and five outputs, employing ReLU activation functions. [JKO19, KBD+17] The units for the ACAS Xu DNNs' inputs are:

- ρ : distance from ownship to intruder in feet
- θ : angle to intruder relative to ownship heading direction in radians for the range $[-\tau,\tau]$
- ψ : heading angle of intruder relative to ownship heading direction in radians for the range $[-\tau,\tau]$
- v_{own} : speed of ownship in feet per second
- v_{int} : speed of intruder in feet per second

Additionally, two other variables are used to generate the 45 neural networks mentioned.

- τ : time until loss of vertical separation in seconds, discretized possible values for it [0, 1, 5, 10, 20, 40, 60, 80, 100].

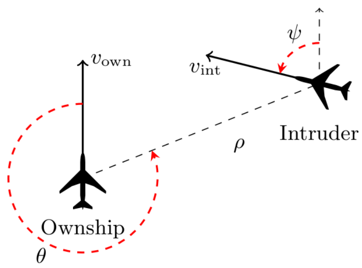


Figure 4.1: Vertical view of a general example of the ACAS Xu. [KBD+17]

- a_{prev} : previous advisory, possible values for it [Clear-of-Conflict, weak left, weak right, strong left, strong right]

The networks are indexed as $N_{x,y}$ where the networks are trained for the x-th value of a_{prev} and y-th value of τ. For example, $N_{3,5}$ is the network trained for the case where a_{prev} = weak right and $\tau = 20$. Further, different 10 properties are defined to test the networks.

- **Property** ϕ_1:
  - If the intruder is distant and is much slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
  - The property is tested on all 45 networks.
  - Three input constraints: $\rho \geq 55947.69$, $v_{own} \geq 1145$, $x_{int} \leq 60$.
  - The desired output property is that the score for COC is at most 1500.
- **Property** ϕ_2:
  - If the intruder is distant and is much slower than the ownship, the score of a COC advisory will never be maximal.
  - The property is tested on $N_{x,y}$ where $x \geq 2$ and for all y.
  - Three input constraints: $\rho \geq 55947.69$, $v_{own} \geq 1145$, $x_{int} \leq 60$.
  - The desired output property is that the score for COC is not the maximal score.
- **Property** ϕ_3:
  - If the intruder is directly ahead towards the ownship, the score for COC will not be minimal.
  - The property is tested on all 45 networks except $N_{1,7}$, $N_{1,8}$ $N_{1,9}$.
  - Five input constraints: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{own} \geq 980$, $v_{int} \geq 960$.
  - The desired output property is that the score for COC is not the minimal score.
- **Property** ϕ_4:
  - If the intruder is directly ahead away from the ownship but at a lower speed than that of the ownship, the score for COC will not be minimal.

- The property is tested on all 45 networks except $N_{1,7}$, $N_{1,8}$ $N_{1,9}$.
- Five input constraints: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi = 0$, $v_{own} \geq 1000$, $700 \leq v_{int} \leq 800$.
- The desired output property is that the score for COC is not the minimal score.

- **Property** ϕ_5:
  - If the intruder is near and approaching from the left, the network advises "strong right".
  - The property is tested on $N_{1,1}$.
  - Five input constraints: $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 400$, $v_{int} \leq 400$.
  - The desired output property is that the score for "strong right" is the minimal score.

- **Property** ϕ_6:
  - If the intruder is sufficiently far away, the network advises COC.
  - The property is tested on $N_{1,1}$.
  - Five input constraints: $12000 \leq \rho \leq 62000$, $(0.7 \leq \theta \leq 3.141592) \vee (-3.141592 \leq \theta \leq 0.7)$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{own} \leq 1200$, $0 \leq v_{int} \leq 1200$.
  - The desired output property is that the score for COC is the minimal score.

- **Property** ϕ_7:
  - For a large vertical separation and a previous "weak left" advisory, the network will either output COC or continue advising "weak left".
  - The property is tested on $N_{1,9}$.
  - Five input constraints: $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq 3.141592$, $100 \leq v_{own} \leq 1200$, $0 \leq v_{int} \leq 1200$.
  - The desired output property is that the scores for "strong right" and "strong left" are never the minimal scores.

- **Property** ϕ_8:
  - For a large vertical separation and a previous "weak left" advisory, the network will either output COC or continue advising "weak left".
  - The property is tested on $N_{2,9}$.
  - Five input constraints: $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq -0.75 \cdot 3.141592$, $-0.1 \leq \psi \leq 0.1$, $600 \leq v_{own} \leq 1200$, $600 \leq v_{int} \leq 1200$.
  - The desired output property is that the score for "weak left" is minimal or the score for COC is minimal.

- **Property** ϕ_9:
  - Even if the previous advisory was "weak right", the presence of a nearby intruder will cause the network to output a "strong left" advisory instead.

  - The property is tested on $N_{3,3}$.
  - Five input constraints: $2000 \leq \rho \leq 7000$, $-0.4 \leq \theta \leq -0.14$, $-3.141592 \leq \psi \leq -3.141592 + 0.01$, $100 \leq v_{own} \leq 150$, $0 \leq v_{int} \leq 150$.
  - The desired output property is that the score for "strong left" is minimal.

- **Property** ϕ_{10}:
  - For a far away intruder, the network advises COC.
  - The property is tested on $N_{4,5}$.
  - Five input constraints: $36000 \leq \rho \leq 60760$, $0.7 \leq \theta \leq 3.14159$, $-3.141592 \leq \psi \leq -3.141592{+}0.01$, $900 \leq v_{own} \leq 1200$, $600 \leq v_{int} \leq 1200$.
  - The desired output property is that the score for COC is minimal.

For our evaluation, we first compute the reachable set of the networks. Afterward, we check the safety verification of the networks, i.e., if the reachable set lies fully in the safe zone. If the result is not empty, we know that the star set contains elements that are not in the safe set and, therefore, not safe. We check both the *reachable set computation time* (RT) and the *safety verification time* (VT) in seconds. From the results, which are also shown in Table 4.1, we can conclude that the star set approach is, on average, faster than without the Reluplex [KBD+17]. Furthermore, the exact method is 7× faster, and the over-approximation method is 134× faster. Additionally, the over-approximative method is 19× faster than the exact method. As the affine mapping and halfspace intersection operations are cheap in computation, the efficiency of star sets in the reachability analysis and verification of piecewise linear DNNs are shown in the results. It is also noticeable that the over-approximation reachability approach verifies fewer networks than the exact reachability approach since the Reluplex benchmarks only consider the exact computations [KBD+17]. We refer to Appendix B for the detailed computation results.

properties	Exact	Overapproximation
	AVG RT(s)	AVG RT(s)
ϕ_1	29402.5067	2049.4807
ϕ_2	44631.003	1775.056
ϕ_3	254.60	10.38
ϕ_4	140.7655	9.4855
Sum	74428.8752	3844.4022

Table 4.1: Average Verification results for properties $\phi_1, \phi_2, \phi_3, \phi_4$ in seconds.

4.1.2 Thermostat

In this section, we consider another benchmark mentioned in this master thesis [Jia23]. The presented thermostat maintains the room temperature x between 17°C and 23°C. It achieves this by activating (mode on) and deactivating (mode off) the heater based on the measured temperature. The neural network representing the thermostat is a feedforward neural network with three layers. The input consists of two neurons that express the temperature $x \in \mathbb{R}$ and mode (on or off) as $m \in \{0{,}1\}$. Furthermore, two hidden layers, each with ten neurons. Lastly, using the unit step function, the output layer predicts whether the heater will turn on $Kh \in \{15\}$ or off $Kh \in \{0\}$. We compute the reachable sets to verify the safety of the described neural network using our reachability algorithm and input as a star set consisting of two variables, one that presents the temperature x and the other the mode m, defined as $\Theta = \langle c,V,P \rangle$ where:

$$\text{the basis } V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{and the center } c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\text{and the predicate } P(\alpha) = C\alpha \leq d \text{ where } C = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \text{ and } d = \begin{bmatrix} 23 \\ -22 \\ 1 \\ -1 \end{bmatrix}$$

As we have the input as a star set and the output reachability set contains multiple stars, using a unit step function layer presented in Algorithm 7 round the reachable star sets to 0 or 15. Thus, we choose $val = 7.5$, $minRes = 0$, and $maxRes = 15$. With this configuration, the input temperature being between 22° and 23°, and the thermostat being turned on, i.e., $m = 1$, the expected control output should be turn off signal. However, the results show two star sets vertices with the value 15, meaning that the neural network violates its safety specification. The two unsafe star set properties are:

$$\Theta_1 = \langle c,V,P \rangle \text{ where: the basis } V = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad \text{the center } c = \begin{bmatrix} 15 \end{bmatrix},$$

$$\text{and } P(\alpha) = C\alpha \leq d \text{ where } C = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0.16383 & -0.68016 \\ 0.16627 & -0.67315 \\ -4.30272 & 17.4634 \\ 66.2391 & -268.762 \end{bmatrix} \text{ and } d = \begin{bmatrix} 23 \\ -22 \\ 1 \\ -1 \\ 2.9718 \\ 3.0211 \\ -77.214 \\ 1195.96 \end{bmatrix}$$

$$\Theta_2 = \langle c,V,P \rangle \text{ where: the basis } V = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad \text{the center } c = \begin{bmatrix} 15 \end{bmatrix},$$

$$\text{and } P(\alpha) = C\alpha \leq d \text{ where } C = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0.16383 & -0.68016 \\ 0.16627 & -0.67315 \\ 4.30272 & -17.4634 \end{bmatrix} \text{ and } d = \begin{bmatrix} 23 \\ -22 \\ 1 \\ -1 \\ 2.9718 \\ 3.0211 \\ 77.214 \end{bmatrix}$$

Therefore, we take those star sets and construct a complete counter input set as in Theorem 2.3.4 to falsify the neural networks, i.e., prove that it is unsafe. Accordingly, we change the basis and center for each star set to the same value of basis and center in the input star set. Afterward, we give those star sets as input in the presented thermostat neural network with an extra unit step function layer. Since the resulted reachability sets have the vertices 15, we know the neural network is unsafe.

4.1.3 Drones

The functionality of autonomous drone control revolves around launching a drone into the air and enabling it to hover at a desired altitude. [GDPT22] This benchmark consists of eight neural networks. The first four consist of two, and the other four networks of three hidden layers, each followed by a ReLU activation function layer. The exact number of neurons, i.e., the size of the layers, is shown in Table 4.2.

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

Table 4.2: The used network architectures. The layer size, i.e., the number of neurons in each layer in column **Neurons**. [GDPT22]

We compute the reachability set of the networks as well as the safety verification using our algorithm and measure the computation time verification time in milliseconds. The computation is tested with the exact and the over-approximation method. For each neural network we test two properties.

$N_{x,y}$	Exact			Overapproximation		
	RT(ms)	RES	VT(ms)	RT(ms)	RES	VT(ms)
$AC1_1$	61385	True	4925	199	False	19
$AC1_2$	526	True	13	60	False	2
$AC2_1$	462405	True	17707	530	False	6
$AC2_2$	112	True	2	66	False	1
$AC3_1$	-	-	-	2951	False	441
$AC3_2$	5119	True	123	242	False	7
$AC4_1$	-	-	-	8711	False	1510
$AC4_2$	103014	True	5528	685	False	5
$AC5_1$	304844	True	26078	366	False	68
$AC5_2$	68	False	2	69	False	0
$AC6_1$	2631726	True	84416	651	False	91
$AC6_2$	98	False	1	73	False	0
$AC7_1$	-	-	-	2545	False	42
$AC7_2$	4050	True	96	288	False	2
$AC8_1$	-	-	-	65930	False	19805
$AC8_2$	812	False	48	579	False	4

Table 4.3: Verification results for the used networks. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in milliseconds. *RES* is the safety verification result. The cells with *(-)* correspond to networks in which our algorithm was not able to complete the verification successfully in less than 48 hours.

The presented results in Table 4.3 show, as we would expect, that the over-approximative algorithm is much faster compared to the exact algorithm. However, the exact algorithm verify almost every network compared to the over-approximation approach.

4.1.4 Sonar Binary Classifier

In this section, we evaluate the robustness of the binary classification of the sonar dataset. This dataset describes sonar chirp returns bouncing off different services. It contains 60 input variables representing the returns' strength at different angles. The verified neural network should be capable of binary classification, distinguishing between rocks and metal cylinders. The neural network consists of two layers, the first followed by a ReLU activation function and the second by a HardSigmoid activation function. The verification we use is an analysis of the local robustness of the neural network. A neural network is δ-*locally-robust* at input x off for every x' such that $|x - x'|_\infty \leq \delta$, the network assigns the same label to x and x'. Our focus lies in determining the robustness value that a verification method can provide a robustness guarantee for the network.

	$\delta = 0.1$		$\delta = 0.01$		$\delta = 0.001$		$\delta = 0.0001$		$\delta = 0.00001$	
	RT	RES	RT	RES	RT	RES	RT	RES	RT	RES
Set 1	100	False	107	False	103	False	100	False	106	False
Set 2	103	False	101	False	103	False	109	False	109	False
Set 3	103	False	102	False	104	False	109	False	112	False
Set 4	105	False	116	False	107	False	110	False	111	False
Set 5	107	False	109	False	105	True	108	True	122	True
Set 6	103	True	107	True	104	True	108	True	114	True
Set 7	104	False	106	False	106	True	109	True	103	True
Set 8	103	True	101	True	103	True	104	True	107	False
Set 9	104	True	106	True	105	True	111	True	103	False
Set 10	104	True	103	True	105	True	107	True	105	False

Table 4.4: Local adversarial robustness tests of the exact approach. *RT* is the reachable set computation time in milliseconds. *RES* is the safety verification result.

We examine this problem on ten input sets of the dataset and five δ values. The first five input sets should be 1, which means a rock, and the next five 0, which means a metal cylinder. True results show that the networks indicate the output right, and False means the network indicates the wrong output. Tables 4.4 and 4.5 show the results of our tests. By comparing the exact algorithm with the over-approximative algorithm, we can observe that the exact algorithm proves the robustness of the network in more cases than the over-approximative algorithm. Furthermore, different input sets have different local robustness depending on the algorithm. For example, in Table 4.4 for Set 5, the optimal δ value is between 0.01 and 0.001, but in Table 4.5, it is between 0.001 and 0.0001.

	$\delta = 0.1$		$\delta = 0.01$		$\delta = 0.001$		$\delta = 0.0001$		$\delta = 0.00001$	
	RT	RES	RT	RES	RT	RES	RT	RES	RT	RES
Set 1	105	False	104	False	104	False	112	False	108	False
Set 2	102	False	103	False	108	False	106	False	107	False
Set 3	101	False	107	False	107	False	109	False	107	False
Set 4	102	False	104	False	108	False	110	False	109	False
Set 5	109	False	104	False	108	False	131	True	109	True
Set 6	104	True	102	True	109	True	107	True	107	True
Set 7	102	False	110	False	109	False	107	False	107	True
Set 8	104	True	105	True	109	True	107	True	109	False
Set 9	104	True	102	True	108	True	108	True	128	True
Set 10	106	True	102	True	109	True	108	True	119	False

Table 4.5: Local adversarial robustness tests of the over-approximate approach. *RT* is the reachable set computation time in milliseconds. *RES* is the safety verification result.

4.2 Experimental Results

4.2.1 Run Time

In this section, we will present performed experiments for the purpose of improving the running time of our proposed exact reachability algorithms. Since in our reachability algorithms, in most cases, we compute matrix multiplications, the approach is to take a closer look at the run time when we perform only single column multiplications instead of matrix multiplications. The matrix multiplication is done when we scale or project the star set by the mapping or scaling matrix. In Hypro, matrix multiplication uses the Eigen library [GJ+10], making the calculation very efficient and cheap. In the experimental approach, column multiplication, at each position in the result matrix, we only multiply the row of the mapping or the scaling matrix with the corresponding column of the basis respectively center matrix. We tested the run time of the column and matrix multiplication on different matrix dimensions and different matrices numbers with OpenMP and without the OpenMp. OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory multiprocessing programming. It provides a portable and scalable solution for parallel programming allowing to write code that can be executed on systems with multiple processors or cores. [OMP] Since using OpenMP leads to performance improvement and faster run time, we test with and without it to better understand both methods. From the corresponding results illustrated in Figure 4.2 and 4.3, we can conclude that with increasing matrices numbers, the run time of column multiplication is faster. In addition, with or without the OpenMP, the results are very similar, but still, using the OpenMP improves the run time. However, with growing matrix dimension, the matrix multiplication is much faster than the column multiplication. Moreover, using the openMP in the matrix multiplication with different dimensions makes a noticeable difference in the run time. In conclusion, depending on the application, it should be decided which method to use, as each method has its advantages.

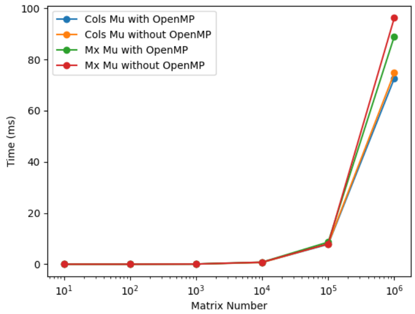


Figure 4.2: Multiplications running time with increasing matrix numbers.

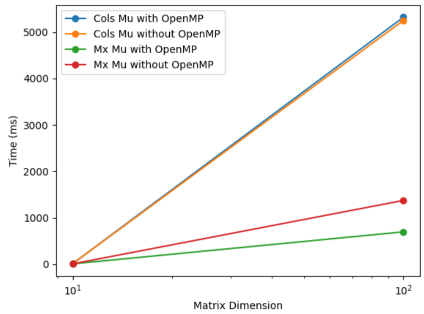


Figure 4.3: Multiplications running time with increasing matrix dimensions.

Chapter 5

Conclusion

5.1 Discussion

In this work, we developed and discussed various algorithms for the star based reachability analysis of different activation functions. The reachability analysis comprises both the exact and complete analysis, in addition to the unbounded analysis, for each of which we examined the different possible cases. We have implemented the activation functions as generally as possible to adapt them to their own use. The computed reachable analyses are useful for observing the complex behavior of networks and determining the networks' safety.

We evaluate the effectiveness of the algorithms on different benchmarks derived from related literature. Through this evaluation, we present quantitative results that provide valuable insights into the runtime of the methods as well as safety verification of the used benchmark networks. Furthermore, we presented an experimental approach to improve the runtime of the methods. Additionally, implementing the ONNX parser in Hypro makes integrating further benchmarks with different networks architectures easier.

However, the current methods are limited to fully connected feedforward neural networks with piecewise linear activation functions. As we saw in the results of our evaluation of the benchmarks, the star based reachability analysis is very efficient. Therefore it would be useful to perform backpropagation to train neural networks, enabling them to learn from adversarial examples, adjust their parameters, and make accurate predictions. Additionally, it is important to expand the scope of evaluation beyond the current restricted set of architectures, i.e., different types of layers, in Hypro. All these restrictions give rise to the following improvements in future work.

5.2 Future work

Currently, the functionality of Hypro is limited to supporting star set representation only. However, as discussed in Section 3.2, it lacks support for other representations like ImageStar. Additionally, implementing different layer types, such as convolutional, pooling, residual, and recurrent layers, is not possible in Hypro. To address these limitations, it would be highly beneficial to expand Hypro's capabilities to include support for additional representations like ImageStar and enable the implementation of various layer types.

Furthermore, it is essential to thoroughly investigate and evaluate the proposed reachability analysis with these different layer types using appropriate benchmarks. By conducting comprehensive experiments and evaluations, we can gain deeper insights into the performance, accuracy, and limitations of the reachability analysis method when applied to neural networks utilizing these layer types.

In addition, given that the star set based reachability analysis focuses on fully connected feedforward neural networks with different activation functions, it would be highly valuable to extend the investigation to integrate the backpropagation methods with star sets. Therefore it would be valuable to investigate the backpropagation methods using the star sets since it is a widely used algorithm for training artificial neural networks and offers numerous advantages in efficient training, scalability, flexibility, and generalization capabilities [WOS+22]. This investigation can provide insights into the feasibility and benefits of incorporating backpropagation with star sets, ultimately contributing to the advancement of safe and reliable learning-based neural networks.

Bibliography

[AAS20] Arohan Ajit, Koustav Acharya, and Abhishek Samanta. A review of convolutional neural networks. In *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*, pages 1–5. IEEE, 2020.

[AG21] Sushma Priya Anthadupula and Manasi Gyanchandani. A review and performance analysis of non-linear activation functions in deep neural networks. *Int. Res. J. Mod. Eng. Technol. Sci*, 2021.

[AJO+19] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Abubakar Malah Umar, Okafor Uchenwa Linus, Humaira Arshad, Abdullahi Aminu Kazaure, Usman Gana, and Muhammad Ubale Kiru. Comprehensive review of artificial neural network applications to pattern recognition. *IEEE Access*, 7:158820–158846, 2019.

[AMAZ17] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[BD17] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 401–420, 2017.

[CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. Smt-rat: An open source c++ toolbox for strategic and parallel smt solving. In *International Conference on Theory and Applications of Satisfiability Testing*, 2015.

[Col04] Ronan Collobert. Large scale machine learning. 2004.

[CWB+11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch, 2011.

[Dat20] Leonid Datta. A survey on activation functions and their relation with xavier and he normal initialization. *CoRR*, abs/2004.06632, 2020.

[Edu23] Educative. What is the vanishing gradient problem? `https://www.educative.io/answers/what-is-the-vanishing-gradient-problem`, 2023. [Accessed: May 12, 2023].

[GDPT22] Dario Guidotti, Stefano Demarchi, Luca Pulina, and Armando Tacchella. Evaluating reachability algorithms for neural networks on never2, 09 2022.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. [Accessed : May 29, 2023].

[GK20] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review, 2020.

[GML+08] Osvaldo Gervasi, Beniamino Murgante, Antonio Laganà, David Taniar, Youngsong Mun, and Marina L. Gavrilova, editors. *Computational Science and Its Applications - ICCSA 2008, International Conference, Perugia, Italy, June 30 - July 3, 2008, Proceedings, Part I*, volume 5072 of *Lecture Notes in Computer Science*. Springer, 2008.

[Gus22] Murilo Gustineli. A survey on recently proposed activation functions for deep learning, 2022.

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

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[IBM] IBM. Was sind rekurrente neuronale netze? `https://www.ibm.com/de-de/topics/recurrent-neural-networks`. [Accessed : June 9, 2023].

[Jia23] Ruoran Gabriela Jiang. Verifying ai-controlled hybrid systems. Master's thesis, RWTH Aachen University, 2023.

[JKO19] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3):598–608, mar 2019.

[KBD+17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.

[Kum19] Kumar, Niranjan. Deep learning: Feedforward neural networks explained. `https://medium.com/hackernoon/deep-learning-feedforward-neural-networks-explained-\c34ae3f084f1`, 2019. [Accessed: May 03, 2023].

[LM17] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351, 2017.

[LW20] Shuvendu K Lahiri and Chao Wang. *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part II.* Springer Nature, 2020.

[LWL+17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[Maa13] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[MS21] Florian Malard and Stéphanie Olivier-Van Stichelen. Pooling (cnn). `https://epynn.net/Pooling.html`, 2021. [Accessed : June 9, 2023].

[Nab19] Javaid Nabi. Recurrent neural networks (rnns). `https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85`, 2019. Accessed : May 20, 2023.

[Nan17] Raghuram Nandepu. Understanding and implementation of residual networks(resnets). `https://medium.com/analytics-vidhya/understanding-and-implementation-of-residual-networks\-resnets-b80f9a507b9c`, 2017. Accessed : May 20, 2023.

[NIGM18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[OMP] Openmp. `https://www.openmp.org/`. [Accessed : May 29, 2023].

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[ONN] Onnx. `https://onnx.ai/`. [Accessed: May 30, 2023].

[PGCB14] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks, 2014.

[PT221] PyTorch: torch.nn.hardsigmoid. `https://pytorch.org/docs/stable/generated/torch.nn.Hardsigmoid.html`, 2021. Accessed : May 16, 2023.

[Ram21] Luthfi Ramadhan. Neural network: The dead neuron. `https://towardsdatascience.com/neural-network-the-dead-neuron-eaa92e575748`, 2021. [Accessed: May 14, 2023].

[RW17] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[SÁMK17] Stefan Schupp, Erika Ábrahám, Ibtissem Makhlouf, and Stefan Kowalewski. Hypro: A c++ library of state set representations for hybrid systems reachability analysis. In Clark Barrett, Misty Davies, and Temesghen Kahsai, editors, *NASA Formal Methods*, pages 288–294, 2017.

[SFÁ19] Stefan Schupp, Goran Frehse, and Erika Ábrahám. *State set representations and their usage in the reachability analysis of hybrid systems.* PhD thesis, RWTH Aachen University, 2019.

[SGPV19] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3:1–30, 01 2019.

[SKP97] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, 39(1):43–62, 1997.

[SSA20] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 2020.

[Sta21] Stanford Intelligent Systems Laboratory. NNet: Neural Network Control Toolbox. `https://github.com/sisl/NNet`, 2021. Accessed : May 24 2023.

[Ste19] Matthew Stewart. Simple introduction to convolutional neural networks. `https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac`, 2019. [Accessed : June 9, 2023].

[TF223] Tensorflow documentation. `https://www.tensorflow.org/api_docs/python/tf/keras/activations/hard_sigmoid`, 2023. Accessed : May 16, 2023.

[TMLM+19] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In *Formal Methods–The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings 3*, pages 670–686. Springer, 2019.

[Tra20] Dung Tran. *Verification of Learning-enabled Cyber-Physical Systems.* PhD thesis, Vanderbilt University Graduate School, aug 2020.

[WOS+22] Logan G. Wright, Tatsuhiro Onodera, Martin M. Stein, Tianyu Wang, Darren T. Schachter, Zoey Hu, and Peter L. McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022.

[XLD+20] Jin Xu, Zishan Li, Bowen Du, Miaomiao Zhang, and Jing Liu. Reluplex made more practical: Leaky relu. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, 2020.

[Yad22] Harsh Yadav. Residual blocks in deep learning. `https://towardsdatascience.com/residual-blocks-in-deep-learning-11d95ca12b00`, 2022. [Accessed : May 30, 2023].

[YAT+20] Yongbin Yu, Kwabena Adu, Nyima Tashi, Patrick Anokye, Xiangxiang Wang, and Mighty Abra Ayidzoe. Rmaf: Relu-memristor-like activation function for deep learning. *IEEE Access*, 8:72727–72741, 2020.

[YNDT18] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9:611–629, 2018.

Appendix A

Supplementary Proofs

This section contains additional proofs that were omitted from the thesis. These proofs further support the main theorems and lemmas discussed throughout the thesis. Therefore, the purpose of the supplementary proofs section is to be read alongside the thesis, enhancing the reader's understanding.

Proposition A.0.1. *2.2.1 Any bounded convex polyhedron* $\mathscr{P} \triangleq \{x \mid Cx \leq d, x \in \mathbb{R}^n\}$ *can be presented as a star.*

Proof. The star set Θ represents the polyhedron $\mathscr{P}$ with the center $c = [0\ 0\ \ldots\ 0]^T$, the basis vectors $V = \{e_1, \ldots, e_n\}$ in which e_i is the $i-th$ basic vector of $\mathbb{R}^n$, and the predicate $P(\alpha) \triangleq C\alpha \leq d$. $\square$

Proposition A.0.2. *2.2.2 Given a star set* $\Theta = \langle c, V, P\rangle$*, an affine mapping of the star* Θ *with the linear mapping matrix* W *and offset vector* b *defined by* $\bar{\Theta} = \{y \mid y = Wx + b, x \in \Theta\}$ *is another star such that*

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}\rangle, \quad \bar{c} = Wc + b, \quad \bar{V} = \{Wv_1, \ldots, Wv_m\}, \quad \bar{P} \equiv P$$

Proof. By the definition of a star, we have $\bar{\Theta} = \{y \mid y = Wc + b + \sum_{i=1}^{m}(\alpha_i W v_i)\}$, so that $P(\alpha_1, \ldots, \alpha_m) = \top$ yields that $\bar{\Theta}$ is another star with the center $\bar{c} = Wc + b$, basis vectors $\bar{V} = \{Wv_1, \ldots, Wv_m\}$ and the same predicate P as the original star Θ. $\square$

Lemma A.0.3. *2.3.1 The worst-case complexity of the number of stars in the reachable set of an* N*-neurons FNN is* $\mathscr{O}(2^N)$*.*

Proof. The exactReLU operation produces one or two more stars at most. Accordingly, in the worst-case scenario, the number of stars of one layer is 2^{n_L} where n_L is the number of neurons in the layer. Due to the output reachable sets of one layer being the inputs for the next layer, the total number of stars in the reachable set of an FNN with k layers and N neurons in the worst case is $2^{n_{L_1}} \cdot \cdots \cdot 2^{n_{L_k}} = 2^{n_{L_1} + \cdots + n_{L_k}} = 2^N$ $\square$

Lemma A.0.4. *2.3.2 The worst-case complexity of the number of constraints of a star in the reachable set of an* N*-neurons FNN is* $\mathscr{O}(N)$*.*

Proof. The exactReLU sub-procedure produces for the given input set Θ one or two more stars that have at most one more constraint. $\square$

The exactReLU sub-procedure produces for the given input set Θ one or two more stars with at most one more constraint. Consequently, for a layer of n neurons, n exactReLU operations are performed most, yielding to star reachable sets with each having at most n constraints more than the input star set. As a result, the number of constraints in a star input set increases linearly over layers leading to the worst-case complexity $\mathscr{O}(N)$.

Theorem A.0.5. *2.3.4 Let F be an FNN, Θ a star input set, $F(\Theta) = \bigcup_{i=1}^{k} \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i \rangle$ be the reachable set of the neural network, and S be a safety specification. Denot $\bar{\Theta}_i = \Theta_i \cap \neg S = \langle c_i, V_i, \bar{P}_i \rangle$, $i = 1, \ldots, k$. The neural network is safe iff $\bar{P}_i = \emptyset$.*

Proof. The exact reachable set is a union of stars. The neural network is considered safe if and only if none of the stars intersect with the unsafe region, which is trivial. In other words, $\bar{\Theta}_i$ is an empty set for all i, equivalently the predicate $\bar{P}_i$ is empty for all i. □

Lemma A.0.6. *3.1.9 The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is $\mathscr{O}(3^N)$.*

Proof. The exactHTangent operation produces three more stars at most. Accordingly, in the worst-case scenario, the number of stars of one layer is 3^{n_L} where n_L is the number of neurons in the layer. Due to the output reachable sets of one layer being the inputs for the next layer, the total number of stars in the reachable set of an FNN with k layers and N neurons in the worst case is $3^{n_{L_1}} \cdot \cdots \cdot 3^{n_{L_k}} = 3^{n_{L_1} + \cdots + n_{L_k}} = 3^N$ □

Lemma A.0.7. *3.1.17 The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is $\mathscr{O}(3^n)$.*

Proof. The exactHSigmoid operation produces three more stars at most. Accordingly, in the worst-case scenario, the number of stars of one layer is 3^{n_L} where n_L is the number of neurons in the layer. Due to the output reachable sets of one layer being the inputs for the next layer, the total number of stars in the reachable set of an FNN with k layers and N neurons in the worst case is $3^{n_{L_1}} \times \cdots \times 3^{n_{L_k}} = 3^{n_{L_1} + \cdots + n_{L_k}} = 3^N$ □

Theorem A.0.8. *2.3.4 Let F be an FNN, Θ a star input set, $F(\Theta) = \bigcup_{i=1}^{k} \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i \rangle$ be the reachable set of the neural network, and S be a safety specification. Denot $\bar{\Theta}_i = \Theta_i \cap \neg S = \langle c_i, V_i, \bar{P}_i \rangle$, $i = 1, \ldots, k$. The neural network is safe iff $\bar{P}_i = \emptyset$. neural network violates its safety property, then the complete counter input set containing all possible inputs in the input set that lead the neural network to unsafe states is $\mathscr{C}_\Theta = \bigcup_{i=1}^{k} \langle c, V, \bar{P}_i \rangle, \bar{P}_i \neq \emptyset$.*

Proof. **Safety**: The exact reachable set is a union of stars. It is trivial that the neural network is safe iff all stars in the reachable set do not intersect with the unsafe region, i.e., $\bar{\Theta}_i$ is an empty set for all i, or the predicate $\bar{P}_i$ is empty for all i. **Complete counter input set**: All star sets in the computation process are defined on the same predicate variable $\alpha = [\alpha_1 \ldots \alpha_m]^T$, which remains unchanged in the computation. However, the number of constraints on α changes. Therefore, when $\bar{P}_i \neq \emptyset$, it contains values of α that make the neural network unsafe. New conditions are added from the basic predicate P via exactReLU operations so that $\bar{P}_i$ contains all the conditions of the basic predicate P. Accordingly, the complete counter input set containing all possible inputs that make the neural network unsafe is defined by $\mathscr{C}_\Theta = \bigcup_{i=1}^{k} \langle c, V, \bar{P}_i \rangle, \bar{P}_i \neq \emptyset$. □

Appendix B

Detailed Tables and Figures

This section contains detailed tables and figures.

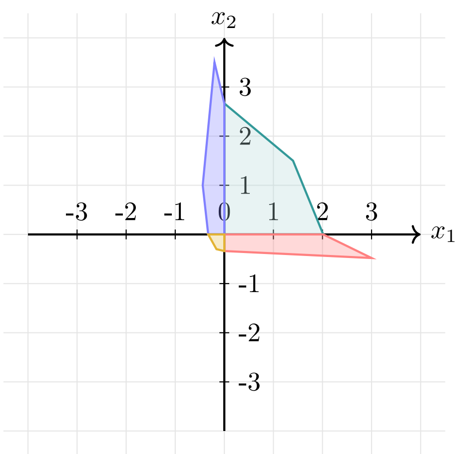


Figure B.1: exactLReLU results from Example 3.1.1.

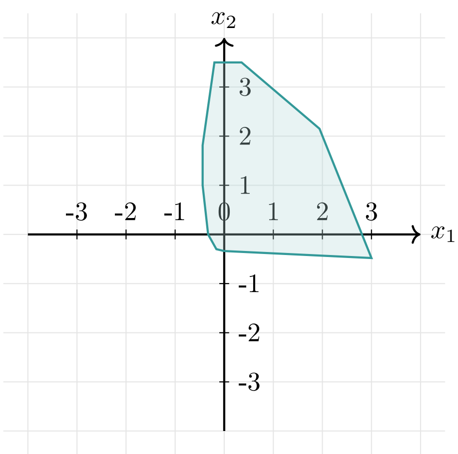


Figure B.2: approxLReLU results from Example 3.1.2.

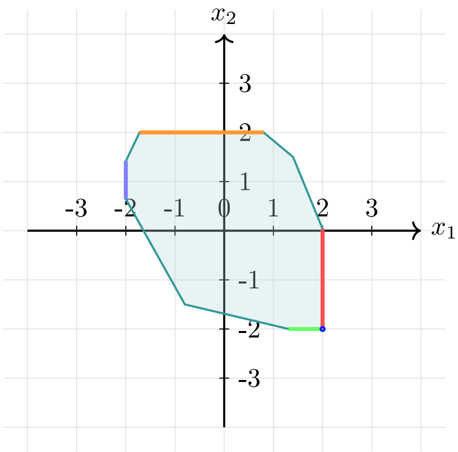


Figure B.3: exactHTangent results from Example 3.1.3.

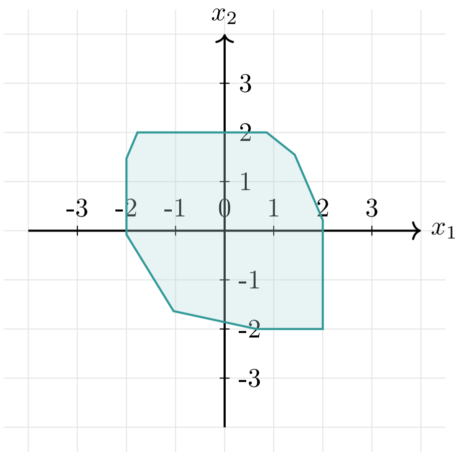


Figure B.4: approxHTangent results from Example 3.1.4.

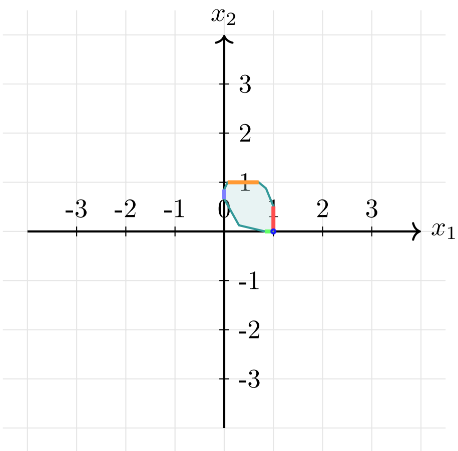


Figure B.5: exactHSigmoid results from Example 3.1.5.

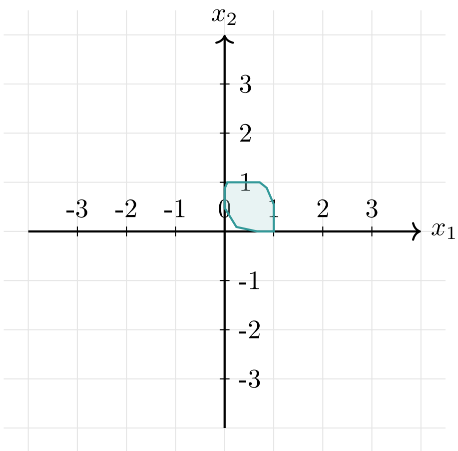


Figure B.6: approxHSigmoid results from Example 3.1.6.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{1,1}$	3013.64	True	45.71	39835	823.72	False	7.97	1
$N_{1,2}$	3575.72	True	53.74	45648	2214.74	False	15.48	1
$N_{1,3}$	11037.81	True	200.90	114287	3211.44	False	16.37	1
$N_{1,4}$	13111.44	True	267.78	154529	2915.33	False	21.64	1
$N_{1,5}$	9756.54	True	196.13	122297	1618.80	False	9.97	1
$N_{1,6}$	35718.94	True	823.34	376647	1385.90	False	13.06	1
$N_{1,7}$	4712.34	True	85.86	66416	1228.45	False	13.30	1
$N_{1,8}$	8279.50	True	174.76	110139	2226.28	False	38.84	1
$N_{1,9}$	9136.22	True	189.03	135645	2999.83	False	24.42	1
$N_{2,1}$	15355.00	True	325.53	193197	1538.98	False	11.57	1
$N_{2,2}$	34071.21	True	617.56	472257	1147.34	False	19.50	1
$N_{2,3}$	13319.28	True	253.42	194275	956.66	False	18.24	1
$N_{2,4}$	8124.85	True	167.98	114155	1141.23	False	21.41	1
$N_{2,5}$	53191.97	True	1103.15	677510	1489.67	False	22.42	1
$N_{2,6}$	23772.93	True	417.89	309631	1972.41	False	10.89	1
$N_{2,7}$	53504.01	True	1016.92	679523	3330.97	False	39.63	1
$N_{2,8}$	48084.68	True	777.85	585647	4132.09	False	38.59	1
$N_{2,9}$	86837.06	True	1395.51	910575	2225.92	False	19.55	1
$N_{3,1}$	14553.80	True	497.52	252793	1130.78	False	22.86	1
$N_{3,2}$	16570.78	True	359.74	181433	2678.57	False	19.91	1
$N_{3,3}$	28386.63	True	649.15	341669	994.11	False	8.74	1
$N_{3,4}$	8765.59	True	154.49	133782	1662.63	False	23.46	1
$N_{3,5}$	28583.49	True	641.96	365066	1884.56	False	15.86	1
$N_{3,6}$	65843.71	True	1595.55	1003886	2494.56	False	28.20	1
$N_{3,7}$	47664.54	True	1094.01	475299	4453.61	False	36.35	1
$N_{3,8}$	38414.57	True	652.14	472562	1501.02	False	11.65	1
$N_{3,9}$	33949.16	True	746.01	379221	3221.20	False	17.70	1
$N_{4,1}$	35166.18	True	603.51	402853	1007.71	False	19.63	1
$N_{4,2}$	41913.56	True	748.73	484555	1322.22	False	21.65	1
$N_{4,3}$	9966.29	True	164.86	138170	1256.42	False	10.84	1
$N_{4,4}$	12343.79	True	213.54	143424	1295.82	False	20.12	1
$N_{4,5}$	39853.04	True	861.19	457447	3795.18	False	28.86	1
$N_{4,6}$	134265.70	True	2703.07	1296311	1816.04	False	18.24	1
$N_{4,7}$	61325.43	True	942.40	652417	999.71	False	10.53	1
$N_{4,8}$	32988.07	True	775.16	515113	2781.59	False	49.10	1
$N_{4,9}$	87048.69	True	1456.81	984701	14379.08	False	7.58	1
$N_{5,1}$	13215.58	True	262.90	201773	1284.01	False	14.30	1
$N_{5,2}$	17025.91	True	347.06	261011	1025.24	False	7.94	1
$N_{5,3}$	10237.07	True	219.97	125860	1089.35	False	13.62	1
$N_{5,4}$	5989.44	True	106.26	81364	1228.81	False	22.77	1
$N_{5,5}$	14346.32	True	239.07	213059	1361.83	False	17.98	1
$N_{5,6}$	102872.82	True	1120.75	622084	2093.55	False	22.16	1
$N_{5,7}$	31414.81	True	595.66	355919	2211.67	False	21.59	1
$N_{5,8}$	95265.60	True	886.15	544813	3226.16	False	24.57	1
$N_{5,9}$	95694.38	True	1002.30	619284	3546.82	False	24.30	1

Table B.1: Verification results for property P_1 on 45 ACAS Xu networks. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{2,1}$	15837.45	False	325.53	193197	1338.12	False	43.21	1
$N_{2,2}$	36112.55	False	1174.41	472257	1005.57	False	68.81	1
$N_{2,3}$	18532.01	False	303.91	194275	962.63	False	74.02	1
$N_{2,4}$	8254.17	False	299.08	114155	1391.11	False	73.80	1
$N_{2,5}$	56241.64	False	2275.48	677510	1790.96	False	111.03	1
$N_{2,6}$	25991.47	False	650.43	309631	2044.11	False	45.39	1
$N_{2,7}$	65508.57	False	1544.92	679523	3366.64	False	161.01	1
$N_{2,8}$	53195.81	False	1260.82	585647	3677.42	False	135.44	1
$N_{2,9}$	-	-	-	-	1855.61	False	64.66	1
$N_{3,1}$	15559.97	False	727.35	252793	939.06	False	54.55	1
$N_{3,2}$	12911.84	False	295.19	181433	2362.05	False	72.16	1
$N_{3,3}$	24675.76	True	508.77	341669	900.05	False	33.53	1
$N_{3,4}$	7393.05	False	205.80	133782	1500.14	False	95.91	1
$N_{3,5}$	23852.32	False	795.67	365066	1723.58	False	67.01	1
$N_{3,6}$	70608.24	False	2823.21	1003886	2694.87	False	120.82	1
$N_{3,7}$	41256.50	False	1233.86	475299	4753.87	False	150.12	1
$N_{3,8}$	37253.31	False	754.01	472562	1304.76	False	38.45	1
$N_{3,9}$	48309.81	False	1152.44	379221	2498.83	False	54.83	1
$N_{4,1}$	66720.59	False	854.57	402853	973.59	False	74.23	1
$N_{4,2}$	85507.29	True	1130.23	484555	1139.11	False	74.50	1
$N_{4,3}$	10627.68	False	303.07	138170	1096.26	False	38.32	1
$N_{4,4}$	12923.14	False	357.63	143424	1257.89	False	77.31	1
$N_{4,5}$	75982.74	False	1223.96	457447	3312.96	False	93.14	1
$N_{4,6}$	-	-	-	-	1802.32	False	68.50	1
$N_{4,7}$	107331.18	False	2132.42	652417	781.43	False	32.85	1
$N_{4,8}$	67596.46	False	1893.68	515113	2661.60	False	186.54	1
$N_{4,9}$	-	-	-	-	13969.59	False	29.64	1
$N_{5,1}$	30332.56	False	407.09	201773	1291.09	False	60.81	1
$N_{5,2}$	43636.56	False	486.83	261011	1173.13	False	36.83	1
$N_{5,3}$	10740.98	False	191.73	125860	1072.01	False	56.17	1
$N_{5,4}$	6221.00	False	138.55	81364	1415.43	False	109.63	1
$N_{5,5}$	31217.25	False	386.70	213059	1275.20	False	61.52	1
$N_{5,6}$	97829.27	False	1149.06	622084	2196.38	False	94.40	1
$N_{5,7}$	57210.41	False	918.01	355919	2347.15	False	92.50	1
$N_{5,8}$	101123.47	False	1521.47	544813	3443.00	False	106.33	1
$N_{5,9}$	78557.70	False	1164.09	619284	3216.64	False	92.62	1

Table B.2: Verification results for property P_2 on 36 ACAS Xu networks. *(RT)* is the reachable set computation time, and *(VT)* is the safety verification time, both in seconds. *(RES)* is the safety verification result. *(OSS)* describes the number of the output star sets. The cells with *(-)* correspond to networks in which our algorithm was not able to compute the reachability set successfully in less than 48 hours.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{1,1}$	1768.51	True	206.94	71930	33.88	False	1.45	1
$N_{1,2}$	1647.32	True	115.58	40273	34.65	False	4.28	1
$N_{1,3}$	442.65	True	29.13	12444	28.54	False	1.87	1
$N_{1,4}$	224.24	True	3.64	4346	12.95	True	0.09	1
$N_{1,5}$	246.37	True	4.09	4820	12.65	True	0.11	1
$N_{1,6}$	65.95	True	0.91	1281	3.79	True	0.03	1
$N_{2,1}$	485.40	True	17.82	16382	23.79	False	1.84	1
$N_{2,2}$	178.63	True	6.48	6924	10.60	False	1.04	1
$N_{2,3}$	316.67	True	10.55	10694	22.85	False	1.21	1
$N_{2,4}$	20.16	True	0.21	351	1.84	True	0.07	1
$N_{2,5}$	111.27	True	2.09	2466	8.17	True	0.11	1
$N_{2,6}$	13.62	True	0.12	255	3.79	True	0.03	1
$N_{2,7}$	60.04	True	0.91	1229	5.07	True	0.15	1
$N_{2,8}$	17.78	True	0.17	329	3.84	True	0.01	1
$N_{2,9}$	9.46	True	0.09	189	0.81	True	0.00	1
$N_{3,1}$	153.28	True	8.77	5999	5.19	True	0.73	1
$N_{3,2}$	2018.12	True	88.51	37541	27.15	False	1.97	1
$N_{3,3}$	390.10	True	12.46	7935	23.19	True	1.89	1
$N_{3,4}$	76.08	True	1.53	2100	29.34	False	2.27	1
$N_{3,5}$	44.62	True	1.19	1042	6.18	True	0.30	1
$N_{3,6}$	99.32	True	1.46	1868	15.38	False	1.73	1
$N_{3,7}$	4.20	True	0.04	107	1.39	True	0.01	1
$N_{3,8}$	33.03	True	0.55	669	5.84	True	0.28	1
$N_{3,9}$	42.36	True	0.82	1223	3.04	True	0.12	1
$N_{4,1}$	50.28	True	1.66	2298	4.80	False	0.90	1
$N_{4,2}$	627.65	True	19.89	18088	16.52	False	1.07	1
$N_{4,3}$	976.11	True	25.53	21237	19.26	False	1.15	1
$N_{4,4}$	34.30	True	0.39	560	2.86	True	0.06	1
$N_{4,5}$	9.23	True	0.23	361	2.41	True	0.03	1
$N_{4,6}$	107.72	True	1.86	2533	39.84	True	0.64	1
$N_{4,7}$	51.25	True	0.61	948	4.47	True	0.08	1
$N_{4,8}$	35.98	True	0.38	576	3.01	True	0.02	1
$N_{4,9}$	36.69	True	0.38	616	7.89	True	0.14	1
$N_{5,1}$	328.52	True	11.35	9556	12.10	False	0.66	1
$N_{5,2}$	61.58	True	2.10	2126	5.45	True	0.88	1
$N_{5,3}$	72.33	True	4.63	2906	10.64	True	3.13	1
$N_{5,4}$	33.32	True	0.86	765	4.98	True	0.09	1
$N_{5,5}$	44.61	True	1.02	1310	7.65	True	0.54	1
$N_{5,6}$	63.87	True	0.71	1166	10.97	True	0.56	1
$N_{5,7}$	4.93	True	0.04	88	0.81	True	0.01	1
$N_{5,8}$	134.31	True	2.11	2406	9.51	True	0.15	1
$N_{5,9}$	4.04	True	0.05	111	1.86	True	0.01	1

Table B.3: Verification results for property P_3 on 45 ACAS Xu networks. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{1,1}$	424.83	True	29.77	19142	12.57	False	4.78	1
$N_{1,2}$	366.72	True	23.35	13143	18.92	False	4.37	1
$N_{1,3}$	277.95	True	13.40	9837	22.69	False	1.40	1
$N_{1,4}$	24.99	True	1.42	1184	5.78	False	0.67	1
$N_{1,5}$	208.17	True	6.32	6608	6.67	False	0.39	1
$N_{1,6}$	117.16	True	3.32	4443	9.87	True	0.92	1
$N_{2,1}$	123.79	True	5.38	5066	12.60	False	2.10	1
$N_{2,2}$	149.75	True	6.18	4500	14.89	False	2.42	1
$N_{2,3}$	27.12	True	0.99	1087	3.62	True	0.72	1
$N_{2,4}$	22.95	True	0.51	913	8.44	True	0.06	1
$N_{2,5}$	88.98	True	2.45	3419	11.61	True	0.45	1
$N_{2,6}$	46.37	True	0.97	1462	15.22	True	0.46	1
$N_{2,7}$	18.88	True	0.29	555	5.68	True	0.08	1
$N_{2,8}$	126.04	True	1.03	1805	51.33	False	1.45	1
$N_{2,9}$	8.05	True	0.06	157	1.85	True	0.01	1
$N_{3,1}$	160.56	True	5.17	4281	9.46	True	1.15	1
$N_{3,2}$	231.23	True	14.38	8708	4.15	True	1.19	1
$N_{3,3}$	25.16	True	1.25	1201	2.44	True	0.16	1
$N_{3,4}$	31.60	True	1.22	1214	4.10	True	0.27	1
$N_{3,5}$	122.59	True	5.01	3630	26.23	True	1.13	1
$N_{3,6}$	62.39	True	1.21	1495	14.23	True	0.85	1
$N_{3,7}$	55.24	True	0.48	862	5.02	True	0.12	1
$N_{3,8}$	20.56	True	0.56	542	8.46	False	0.35	1
$N_{3,9}$	148.09	True	1.57	2684	15.36	True	0.95	1
$N_{4,1}$	19.95	True	0.87	848	1.34	True	0.35	1
$N_{4,2}$	38.94	True	1.41	1348	9.42	True	2.55	1
$N_{4,3}$	78.86	True	3.72	3725	12.85	True	4.57	1
$N_{4,4}$	58.67	True	0.82	1253	19.74	False	1.14	1
$N_{4,5}$	45.51	True	0.71	1255	8.60	True	0.23	1
$N_{4,6}$	87.67	True	1.65	2366	11.26	True	0.56	1
$N_{4,7}$	6.78	True	0.10	216	3.65	True	0.08	1
$N_{4,8}$	79.35	True	1.01	1591	9.29	True	0.09	1
$N_{4,9}$	139.12	True	1.75	2566	9.37	True	0.20	1
$N_{5,1}$	166.59	True	9.26	6932	9.45	True	0.63	1
$N_{5,2}$	117.59	True	6.30	4361	4.22	True	0.56	1
$N_{5,3}$	42.58	True	2.18	1662	7.46	True	0.71	1
$N_{5,4}$	40.08	True	1.24	1088	5.68	True	0.16	1
$N_{5,5}$	52.80	True	0.89	1549	7.65	True	0.24	1
$N_{5,6}$	27.96	True	0.50	677	6.74	True	0.47	1
$N_{5,7}$	6.08	True	0.06	157	2.63	True	0.04	1
$N_{5,8}$	24.29	True	0.28	502	12.76	True	0.73	1
$N_{5,9}$	34.99	True	0.55	992	5.26	True	0.15	1

Table B.4: Verification results for property P_4 on 42 ACAS Xu networks. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{1,1}$	2933.99	True	352.76	59734	461.70	False	8.63	1

Table B.5: Verification results for property P_5 on 1 ACAS Xu network. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{1,1}$	44026.93	True	1159.88	187775	1083.38	False	17.91	1

Table B.6: Verification results for property P_6 on 1 ACAS Xu network. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
N_{11}	-	-	-	-	1520.91	False	147.38	1

Table B.7: Verification results for property P_7 on 1 ACAS Xu network. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets. The cells with *(-)* correspond to networks in which our algorithm was not able to compute the reachability set successfully in less than 48 hours

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{2,9}$	-	-	-	-	1560.52	False	46.37	1

Table B.8: Verification results for property P_8 on 1 ACAS Xu network. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets. The cells with *(-)* correspond to networks in which our algorithm was not able to compute the reachability set successfully in less than 48 hours.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{3,3}$	33727.84	False	458.09	338600	541.62	False	7.83	1

Table B.9: Verification results for property P_9 on 1 ACAS Xu network. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.

$N_{x,y}$	Exact				Overapproximation			
	RT(s)	RES	VT(s)	OSS	RT(s)	RES	VT(s)	OSS
$N_{4,5}$	3281.44	True	181.59	41088	1087.41	False	17.68	1

Table B.10: Verification results for property P_{10} on 1 ACAS Xu network. *RT* is the reachable set computation time, and *VT* is the safety verification time, both in seconds. *RES* is the safety verification result. *OSS* describes the number of the output star sets.