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


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

1 Introduction ................................................................. 9
   1.1 Related Work ....................................................... 9
   1.2 Thesis Outline ..................................................... 10

2 Preliminaries ............................................................. 11
   2.1 Feedforward Neural Network ...................................... 11
   2.2 Star Set Representation .......................................... 12
   2.3 Reachability Analysis of Neural Networks with ReLU Activation Function ................................................................. 15
      2.3.1 Exact and Complete Analysis ................................ 16
      2.3.2 Over-approximate Analysis .................................. 17

3 Implementation ............................................................ 19
   3.1 Star Set based Reachability Analysis of Neural Networks ................................................................. 19
      3.1.1 Reachability of Leaky ReLU Layer (LReLU) ................. 20
      Exact and Complete Analysis ...................................... 21
      Over-approximate Analysis ....................................... 23
      Unbounded Analysis .............................................. 26
      3.1.2 Reachability of Hard Tanh Layer ............................... 29
      Exact and Complete Analysis ...................................... 29
      Over-approximate Analysis ....................................... 34
      Unbounded Analysis .............................................. 39
      3.1.3 Reachability of Hard Sigmoid Layer ........................... 42
      Exact and Complete Analysis ...................................... 43
      Over-approximate Analysis ....................................... 47
      Unbounded Analysis .............................................. 52
      3.1.4 Reachability of Unit Step Function Layer .................... 55
      Exact and Complete Analysis ...................................... 56
   3.2 Non-Fully Connected Layers ......................................... 58
      3.2.1 Convolutional Layer ........................................... 58
      3.2.2 Pooling Layer ............................................... 59
      3.2.3 Residual Layer .............................................. 60
      3.2.4 Recurrent Layer ............................................. 61
   3.3 ONNX Parser .......................................................... 61

4 Evaluation ................................................................. 63
   4.1 Benchmarks .......................................................... 63
      4.1.1 ACAS Xu ......................................................... 63
      4.1.2 Thermostat ...................................................... 66
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 able 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 [BDI17]. 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 \cite{TMLM19, 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 \cite{KBD17}. 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 \cite{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).

![Feedforward neural network with four layers. Input with two neurons, two hidden layers, each with three neurons, and an output layer with one neuron.](image)

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).
Secondly, each layer has a bias vector. Therefore $b^k$ represents the bias of the $k^{th}$ layer. However, $b^k_i$ 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:

$$y_i = f_i(b^k_i + \sum_{j=1}^{n} w_{ij}x_j) \quad (2.1)$$

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 $\Theta$ 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) vectors, and $P : \mathbb{R}^m \to \{\top, \bot\}$ is a predicate. The set of states represented by the star is given as

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

Sometimes we will refer to both the tuple $\Theta$ and the set of states $[\Theta]$ as $\Theta$. 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, $C \in \mathbb{R}^{p \times m}$, $\alpha$ 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:

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

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

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

$$[\Theta] \equiv \{ (x_1, x_2) \mid -1 \leq x_1 \leq 4 \land -3 \leq x_2 \leq 1 \}$$
Proposition 2.2.2 (Affine Mapping of Star). Given a star set \( \Theta = \langle c, V, P \rangle \), an affine mapping of the star \( \Theta \) 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

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

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 \cite{SFA19} 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 \( \Theta \):

\[
\text{the affine mapping matrix } W = \begin{bmatrix} \cos(45^\circ) & -\sin(45^\circ) \\ \sin(45^\circ) & \cos(45^\circ) \end{bmatrix} \text{ and the offset vector } b = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}
\]

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

\[
\text{the basis } \bar{V} = \begin{bmatrix} 0.707107 & -1.41421 \\ 0.707107 & 1.41421 \end{bmatrix}, \text{ and the center } \bar{c} = \begin{bmatrix} 0.914214 \\ -0.5 \end{bmatrix}
\]
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 \( \mathcal{H} \triangleq \{ x \mid Hx \leq g \} \) is another star

\[
\bar{\Theta} = \Theta \cap \mathcal{H} = \langle c, V, \bar{P} \rangle \quad \text{with} \quad \bar{P} = P \wedge P',
\]

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

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

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

Additionally let half-space \( \mathcal{H} \) be defined as:

\( \mathcal{H} \triangleq \{ x \mid Hx \leq g \} \) with \( x \in \mathbb{R}^2, \ H = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \) 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} \quad \text{where} \quad \bar{C} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 1.41421 & 0 \end{bmatrix} \quad \text{and} \quad \bar{d} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \\ 2.58579 \end{bmatrix} \)
Proposition 2.2.4 (Check for Emptiness). Given the star set \( \Theta = (c, V, P) \) where \( P(\Theta) \triangleq C\alpha \leq d \) and \( C \in \mathbb{R}^{p \times m} \), \( \alpha \) is the vector of \( m \)-variables, i.e., \( \alpha = [\alpha_1 \ldots \alpha_m]^T \) and \( d \in \mathbb{R}^{p \times 1} \). Check if there exists no \( \alpha \) that can satisfy 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 \( \Theta = (c, V, P) \) for each dimension \( n \), determine the direction vectors \([0 \ldots 1 \ldots 0],[0 \ldots -1 \ldots 0]\), 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 \([v_1^n \ldots v_m^n],[v_1^n \ldots v_m^n]\) 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 \( \mathcal{I} \triangleq \{x \mid Ax \leq b, x \in \mathbb{R}^n\} \), and a \( k \)-layers feed-forward neural network \( F \triangleq \{L_1, \ldots, L_k\} \). Accordingly, the reachable set \( F(\mathcal{I}) = \mathcal{R}_{L_k} \) of the neural network \( F \) given the input set \( \mathcal{I} \) is defined by:

\[
\mathcal{R}_{L_0} \triangleq \mathcal{I},
\]

\[
\mathcal{R}_{L_i} \triangleq \{y_i \mid y_i = f_k(W_i y_{i-1} + b_i), y_{i-1} \in \mathcal{R}_{L_{i-1}}\}
\]

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 \( \mathcal{R}_{L_i} \) can be recursively applied to all \( i = 1 \) to \( k \).
Also, the reachable set $R_{L_k}$ 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 $\mathcal{I}$ defined as a set of linear constraints on the neural network outputs $\mathcal{I} \triangleq \{ y_k \mid Cy_k \leq d \}$. The neural network $F$ is safe corresponding to the input set $\mathcal{I}$, i.e. $F(\mathcal{I}) \models \mathcal{I}$, iff $R_{L_k} \cap \neg \mathcal{I} = \emptyset$.

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 $x$,

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

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 $R_{L_k} = ReLU_n(ReLU_{n-1}(\ldots ReLU_1(\Theta)))$ on the star set input $\Theta$. 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 $\Theta = \langle c, V, P \rangle$, $\Theta$ is divided into two subsets $\Theta_1 = \Theta \land (x_i \geq 0)$ and $\Theta_2 = \Theta \land (x_i < 0)$. Based on Proposition 2.2.3 these 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 $\Theta_1$ when applied on $x_i$ in $x = [x_1 \ldots x_n]^T \in \Theta_1$ because ReLU on the positive branch acts as an identity function. On the other side since $x_i < 0$, $\Theta_2$ 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 vector $x' = [x_1 \ldots x_{i-1} 0 x_{i+1} \ldots x_n]^T$. Therefore, the exactReLU operation for the input star set $\Theta$ of the $i$th neuron results in $ReLU_i(\Theta) = \langle c, V, P_1 \rangle \cup \langle M c, M V, P_2 \rangle$.

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


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 $O(N)$.

Theorem 2.3.3 (Verification Complexity). Let $F$ be an FNN with $N$ neurons, $\Theta$ 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 $F(\Theta) \models S$ 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, $\Theta$ a star input set, $F(\Theta) = \cup_{i=1}^{k} \Theta_i$, $\Theta_i = \langle c_i, V_i, P_i \rangle$ be the reachable set of the neural network, and $S$ be a safety specification. Denote $\Theta_i = \Theta_i \cap \neg S = \langle c_i, V_i, P_i \rangle$, $i = 1, \ldots, k$. The neural network is safe iff $P_i = \emptyset$ 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 $\mathcal{C}_\Theta = \cup_{i=1}^{k} \langle c, V, \bar{P}_i \rangle$, 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 $y_i = \text{ReLU}(x_i)$, let

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

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 $\mu = -\frac{l_i}{u_i - l_i}$ (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 $\alpha_{m+1}$ to the predicate $P$ where $P(\alpha) \triangleq Ca \leq d$. $\alpha_{m+1}$ 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 [SAMK17].

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
Implementation

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 \cite{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 layer-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. \cite{Maa13}.

**Definition 3.1.1 (Leaky ReLU Function \cite{XLD_20}).** Given the input $x$

$$LeakyReLU(x) = \begin{cases} x, & x > 0 \\ \gamma \cdot x, & x \leq 0 \end{cases} = max(\gamma \cdot x, x)$$ (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 \cite{Gus22, Dat20, XLD_20}.
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 $R_k = \text{LeakyReLU}_{n}(\text{LeakyReLU}_{n-1}(\ldots \text{LeakyReLU}_{1}(\Theta)))$ 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 $l_i$ 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, $\Theta$ 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 $\Theta_1$. Since $\Theta_2$ 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 $\Theta_2$ 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 $\Theta$ results in $\text{LeakyReLU}_i(\Theta) = \langle c, V, P_1 \rangle \cup \langle Mc, MV, P_2 \rangle$.

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 $\Theta$ 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 $\Theta(2^N)$.

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 $\Theta(N)$. 

Figure 3.1: Leaky ReLU function (LReLU) where $\gamma = 0.2$
Example 3.1.1. Let $\Theta = \{c, V, P\}$ be the input set, where:

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

also the predicate $P(\alpha) \dashv C\alpha \leq d$ 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 $d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$

\[ x_1 \quad x_2 \]
\[ -3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \]

Figure 3.2: The input star set $\Theta$ corresponding to Example 3.1.1

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

\[ x_1 \quad x_2 \]
\[ -3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \]

Figure 3.3: $\Theta'$ after applying exactLReLU on the first dimension.

Figure 3.4: $\Theta''$ after applying exactLReLU on the second dimension.
Algorithm 1 Star set based exact reachability analysis for a leaky ReLU layer

Input: Input star set \( I = \{\Theta_1, \ldots, \Theta_N\} \)
Output: Exact reachable set \( R \)

1: procedure LAYERREACH(\( I, W, b \))
2: \( R \leftarrow \emptyset \)
3: for \( j = 1 : N \) do
4: \( I_1 \leftarrow W \ast \Theta_j + b = (Wc_j + b, WV_j, P_j) \)
5: \( R_1 \leftarrow I_1 \)
6: for \( i = 1 : n \) do \( \triangleright n \) exactLReLU operations
7: \( R_1 \leftarrow \text{exactLReLU}(R_1, i, l, u) \)
8: \( R \leftarrow R \cup R_1 \)
9: return \( R \)
10: procedure EXACTLReLU(\( \bar{I}, i, l, u \)) \( \triangleright \) Intermediate representations
11: \( \bar{R} \leftarrow \emptyset, \bar{I} = [\Theta_1, \ldots, \Theta_N] \)
12: for \( j = 1 : k \) do
13: \( [l, u] \leftarrow \Theta_j.\text{range}(i) \) \( \triangleright l_i \leq x_i \leq u_i, x_i \in I_1[i] \)
14: \( R_1 \leftarrow \emptyset, M \leftarrow [c_1 \ldots c_{i-1} \gamma c_{i+1} \ldots c_n] \)
15: if \( l_i > 0 \) then
16: \( R_1 \leftarrow \Theta_j \leftarrow (\bar{c}_j, \bar{V}_j, \bar{P}_j^\prime) \)
17: else if \( u_i \leq 0 \) then
18: \( R_1 \leftarrow M \ast \Theta_j \leftarrow (M\bar{c}_j, M\bar{V}_j, \bar{P}_j^\prime) \)
19: else \( \triangleright l_i < 0 \& u_i > 0 \)
20: \( \Theta_j^\prime \leftarrow \Theta_j \& x[i] > 0 \leftarrow (\bar{c}_j, \bar{V}_j, \bar{P}_j^\prime) \)
21: \( \Theta_j^\prime \leftarrow \Theta_j \& x[i] \leq 0 \leftarrow (\bar{c}_j, \bar{V}_j, \bar{P}_j^\prime) \)
22: \( R_1 \leftarrow \Theta_j^\prime \cup M \ast \Theta_j^\prime \)
23: \( \bar{R} \leftarrow \bar{R} \cup R_1 \)
24: return \( \bar{R} \)

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 \( y_i = \text{LeakyReLU}(x_i) \), let

\[
\begin{align*}
y_i & = x_i & l_i & \geq 0 \\
y_i & = x_i \gamma, & u_i & \leq 0 \\
y_i & \geq x_i \gamma, & y_i & \leq \frac{u_i(\gamma - 1)}{u_i} \cdot x_i + \frac{(u_i - l_i)(\gamma - 1)}{u_i - l_i} \cdot x_i, & y_i & \geq x_i & l_i & < 0 \& u_i & > 0
\end{align*}
\]

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. Given the input \( \Theta = (c, V, P) \). 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
Implementation

LeakyReLU($x_i$)

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 $M = [e_1 \ldots e_i - 1 \gamma e_{i+1} \ldots e_n]$ so that the $i^{th}$ state variable is set to $\gamma x_i$ 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 $\alpha_{m+1}$ 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 $\alpha' = [\alpha_1 \ldots \alpha_m \alpha_{m+1}]$:

$$
\begin{align*}
\alpha_{m+1} &\geq x_i, \\
\alpha_{m+1} &\geq x_i \cdot \gamma, \\
\alpha_{m+1} &\leq \frac{u_i - (\gamma \cdot l_i)}{u_i - l_i} \cdot x_i + \frac{(u_i \cdot l_i)(\gamma - 1)}{u_i - l_i}
\end{align*}
$$

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:

- the basis $V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, 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 $d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$
We apply the approxLReLU operation on the dimensions of $\Theta$ with $\gamma = 0.2$, resulting in:

Figure 3.6: The input star set $\Theta$ corresponding to Example 3.1.2.

Figure 3.7: $\Theta'$ after applying approxLReLU on the first dimension.

Figure 3.8: $\Theta''$ 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 \( R \)

1: procedure \( \text{-layerReach}(I, W, b) \)

2: \( I_1 \leftarrow W * I_0 + b = \langle Wc + b, WV, P \rangle \)

3: \( I' \leftarrow I_1 \)

4: for \( i = 1 : n \) do \( \triangleright n \text{ approxLReLU operations} \)

5: \( I' \leftarrow \text{approxLReLU}(I', i) \)

6: \( R_1 \leftarrow I' \)

7: procedure \( \text{approxLReLU}(\tilde{I}, i) \)

8: \( I \leftarrow \Theta = \langle \tilde{c}, \tilde{V}, \tilde{P} \rangle \)

9: \( [l_i, u_i] \leftarrow \Theta.\text{range}(i) \) \( \triangleright l_i \leq x_i \leq u_i \)

10: \( M \leftarrow [e_1 \ldots e_{i-1} 0 \ e_{i+1} \ldots \ e_n] \)

11: \( M' \leftarrow [e_1 \ldots e_{i-1} \gamma \ e_{i+1} \ldots \ e_n] \)

12: if \( l_i \geq 0 \) then

13: \( \tilde{R} \leftarrow \Theta_j = \langle \tilde{c}, \tilde{V}, \tilde{P} \rangle \)

14: else if \( u_i \leq 0 \) then

15: \( \tilde{R} \leftarrow M' * \Theta = \langle M'\tilde{c}, M'\tilde{V}, \tilde{P} \rangle \)

16: else \( \triangleright l_i < 0 \& u_i > 0 \)

17: \( \tilde{P} (\alpha) \triangleq \tilde{C} \alpha \leq \tilde{d}, \alpha = [\alpha_1 \ldots \alpha_{m+1}]^T \)

18: \( \alpha' \leftarrow [\alpha_1 \ldots \alpha_{m} \alpha_{m+1}]^T \) \( \triangleright \text{New variable } \alpha_{m+1} \)

19: \( C_1 \leftarrow \tilde{V}_i - 1, d_1 \leftarrow -\tilde{c}_i \)

20: \( C_2 \leftarrow \tilde{V}_i - \gamma, d_2 \leftarrow -\gamma \cdot \tilde{c}_i \)

21: \( C_3 \leftarrow -\frac{u_i - \gamma l_i}{u_i - l_i} \tilde{V}_i 1, d_3 \leftarrow \frac{(u_i - \gamma l_i)\tilde{c}_i + l_i u_i (\gamma - 1)}{u_i - l_i} \)

22: \( C_0 \leftarrow \tilde{C}_{5 \times 1}, d_0 \leftarrow \tilde{d} \)

23: \( C' \leftarrow [C_0 \ C_1 \ C_2 \ C_3], d' \leftarrow [d_0 \ d_1 \ d_2 \ d_3] \)

24: \( P'(\alpha') \triangleq C' \alpha' \leq d' \)

25: \( \tilde{c}' \leftarrow M\tilde{c}, V' \leftarrow M\tilde{V}, V' \leftarrow \tilde{V}' e_i \)

26: \( \tilde{R} \leftarrow \langle \tilde{c}', \tilde{V}', \tilde{P}' \rangle \)

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 \( \Theta \) has only one lower or only one upper or no bounds at all.

- First, if the input star set \( \Theta \) 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 \( y_i = \text{LeakyReLU}(x_i) \), let

\[
\begin{aligned}
y_i &= x_i \\
y_i &\geq x_i \cdot \gamma, y_i \leq x_i + l_i \cdot (\gamma - 1), y_i \geq x_i & l_i \geq 0 \\
y_i &\geq x_i \cdot \gamma, y_i \leq x_i + l_i \cdot (\gamma - 1), y_i \geq x_i & l_i < 0 
\end{aligned}
\]

(3.3)

where \( l_i \) is the lower bound of \( x_i \).

![Convex relaxation for the leaky ReLU function, in the unbounded case.](image)

Applying \text{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 \( \alpha_{m+1} \) 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 \text{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 \( y_i = \text{LeakyReLU}(x_i) \), let

\[
\begin{aligned}
y_i &= \gamma \cdot x_i \quad u_i \leq 0 \\
y_i &\geq x_i \cdot \gamma, y_i \leq \gamma \cdot x_i + u_i \cdot (1 - \gamma), y_i \geq x_i & u_i > 0 
\end{aligned}
\]

(3.4)

where \( u_i \) is the upper bound of \( x_i \).
LeakyReLU\left(x_i\right)
\begin{align*}
y_i &\leq \gamma \cdot x_i + u_i \cdot (1 - \gamma) \\
y_i &\geq x_i \cdot \gamma \\
y_i &\leq \gamma \cdot x_i + u_i \cdot (1 - \gamma)
\end{align*}

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 approxLeReLU, 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 $\alpha_{m+1}$, 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 analyzed 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 $y_i = \text{LeakyReLU}(x_i)$, let

$$
\begin{cases}
y_i \geq x_i \cdot \gamma, & y_i \geq x_i, \\
x_i \in \mathbb{R}
\end{cases}

(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 $\alpha_{m+1}$, 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 $x$

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

![HardTanh Function](image)

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, NIGMS] as well as since we can verify it, in contrast to the tanh.

**Exact and Complete Analysis**

With the input $\Theta = \langle e, V, P \rangle$, we compute the reachable set by executing a sequence of $n$ exactHTangent operations $R_k = \text{HardTanh}_n(\text{HardTanh}_{n-1}(\ldots \text{HardTanh}_1(\Theta)))$ 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:

\[
HardTanh(x) = \begin{cases} 
\minVal, & x < \minVal \\
\maxVal, & x > \maxVal \\
x, & \minVal \leq x \leq \maxVal 
\end{cases}
\] (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 \( M = [e_1 \ldots e_{i-1} \ 0 \ e_{i+1} \ldots e_n] \). 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 \( M = [e_1 \ldots e_{i-1} \ 0 \ e_{i+1} \ldots e_n] \). 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 \( \Theta_1 = \Theta \land (\minVal \leq x_i \leq \maxVal) \) and \( \Theta_2 = \Theta \land (x_i > \maxVal) \). 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 lead to the new vector \( x' = [x_1 \ldots x_i-1 \maxVal \ x_{i+1} \ldots x_n]^T \in \Theta_2 \), which is the same as projecting the set to zero by the mapping matrix \( M = [e_1 \ldots e_{i-1} \ 0 \ e_{i+1} \ldots e_n] \), 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 \( \Theta \) results in \( HardTanh_i(\Theta) = \langle c, V, P_1 \rangle \cup \langle Mc + s, MV, P_2 \rangle \), where \( s \) is the shifting vector \( s = [0 \ldots \maxVal \ldots 0]^T \).

- 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 \( \Theta_1 = \Theta \land (\minVal \leq x_i \leq \maxVal) \) and \( \Theta_2 = \Theta \land (x_i < \minVal) \). 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 \( M = [e_1 \ldots e_{i-1} \ 0 \ e_{i+1} \ldots e_n] \) and afterward changing the center \( c_i \) to \( \minVal \). Finally, the exactHTangent operation at the \( i \)th neuron for the input star set \( \Theta \) results in \( HardTanh_i(\Theta) = \langle c, V, P_1 \rangle \cup \langle Mc + s', MV, P_2 \rangle \), where \( s' \) is the shifting vector, \( s' = [0 \ldots \minVal \ldots 0]^T \).

- 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
\( \Theta_1 = \Theta \land (\text{minVal} \leq x_i \leq \text{maxVal}) \), \( \Theta_2 = \Theta \land (x_i > \text{maxVal}) \), and \( \Theta_3 = \Theta \land (x_i < \text{minVal}) \). \( \Theta_1 = \langle c, V, P_1 \rangle \), 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 \( x' = [x_1 \ldots x_{i-1} \text{maxVal} x_{i+1} \ldots x_n]^T \), i.e., we map the set with the mapping matrix \( M = [e_1 \ldots c_{i-1} 0 e_{i+1} \ldots e_n] \), then set the center to \( \text{maxVal} \). For \( \Theta_3 \), we do the same, but instead of setting the center \( c_i \) to \( \text{maxVal} \), we place it to \( \text{minVal} \). Accordingly, the exactHTangent operation at the \( i \)th neuron for the input star set \( \Theta \) results in a union of three star sets 
\[
\text{HardTanh}_i(\Theta) = \langle c, V, P_1 \rangle \cup \langle Mc+s, MV, P_2 \rangle \cup \langle Mc+s', MV, P_3 \rangle,
\]
where \( s, s' \) are the shifting vectors, \( s = [0 \ldots \text{maxVal} \ldots 0]^T \) and \( s' = [0 \ldots \text{minVal} \ldots 0]^T \).

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

**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 \( \Theta(2^N) \).

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

- the basis \( V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \), 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 \( d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix} \).

We apply the exactHTangent operation on the dimensions of \( \Theta \) with \( \text{minVal} = -2 \) and \( \text{maxVal} = 2 \), resulting in:  

![Figure 3.13: The input star set \( \Theta \) corresponding to Example 3.1.3](image)
Figure 3.14: $\Theta'$ after applying exactHTangent on the first dimension.

Figure 3.15: $\Theta''$ after applying exactHTangent on the second dimension.
Algorithm 3 Star set based exact reachability analysis for a HardTanh layer

**Constants:** \( \text{minVal, maxVal} \)

**Input:** Input star set \( I = [\Theta_1, \ldots, \Theta_N] \)

**Output:** Exact reachable set \( R \)

1. procedure \( \text{layerReach}(I, W, b) \)
2. \( R \leftarrow \emptyset \)
3. for \( j = 1 : N \) do
4. \( I_1 \leftarrow W * \Theta_j + b = (W e_j + b, W V_j, P_j) \)
5. \( R_1 \leftarrow I_1 \)
6. for \( i = 1 : n \) do
7. \( R_1 \leftarrow \text{exactHTangent}(R_1, i, l_i, u_i) \)
8. \( R \leftarrow R \cup R_1 \)
9. return \( R \)
10. procedure \( \text{exactHTangent}(I, i, l_i, u_i) \)
11. \( R \leftarrow \emptyset, I = [\Theta_1, \ldots, \Theta_k] \)
12. for \( j = 1 : k \) do
13. \( [l_i, u_i] \leftarrow \Theta_j, \text{range}(i) \)
14. \( R_1 \leftarrow \emptyset, M \leftarrow [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n] \)
15. \( s \leftarrow [0 \ldots \text{maxVal} \ldots 0]^T \)
16. \( s' \leftarrow [0 \ldots \text{minVal} \ldots 0]^T \)
17. if \( l_i \geq \text{minVal} \text{ and } u_i \leq \text{maxVal} \) then
18. \( R_1 \leftarrow \Theta_j = (\tilde{c}_j, \tilde{V}_j, \tilde{P}_j) \)
19. else if \( u_i < \text{minVal} \) then
20. \( \Theta_j \leftarrow M * \Theta_j = (M \tilde{c}_j + s', M \tilde{V}_j, \tilde{P}_j) \)
21. \( R_1 \leftarrow \Theta \)
22. else if \( l_i > \text{maxVal} \) then
23. \( \Theta_j \leftarrow M * \Theta_j = (M \tilde{c}_j + s, M \tilde{V}_j, \tilde{P}_j) \)
24. \( R_1 \leftarrow \Theta \)
25. else if \( \text{minVal} \leq l_i \leq \text{maxVal} \text{ and } u_i > \text{maxVal} \) then
26. \( \Theta'_j \leftarrow \Theta_j \wedge \text{minVal} \leq x[i] \leq \text{maxVal} = (\tilde{c}_j, \tilde{V}_j, \tilde{P}_j) \)
27. \( \Theta''_j \leftarrow \Theta_j \wedge x[i] > \text{maxVal} = (\tilde{c}_j, \tilde{V}_j, \tilde{P}_j) \)
28. \( R_1 \leftarrow \Theta'_j \leftarrow M * \Theta'_j = (M \tilde{c}_j + s, M \tilde{V}_j, \tilde{P}_j) \)
29. else if \( l_i < \text{minVal} \text{ and } u_i \leq \text{maxVal} \) then
30. \( \Theta'_j \leftarrow \Theta_j \wedge \text{minVal} \leq x[i] \leq \text{maxVal} = (\tilde{c}_j, \tilde{V}_j, \tilde{P}_j) \)
31. \( \Theta''_j \leftarrow \Theta_j \wedge x[i] < \text{minVal} = (\tilde{c}_j, \tilde{V}_j, \tilde{P}_j) \)
32. \( R_1 \leftarrow \Theta'_j \leftarrow M * \Theta'_j = (M \tilde{c}_j + s', M \tilde{V}_j, \tilde{P}_j) \)
33. else
34. \( R_1 \leftarrow \emptyset \)
35. \( R \leftarrow R \cup R_1 \)
36. return \( R \)
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 \( y_i = \text{HardTanh}(x_i) \), let

\[
\begin{align*}
    y_i &= x_i & l_i &\geq \text{minVal} \land u_i \leq \text{maxVal} \\
    y_i &= \text{minVal} & u_i &< \text{minVal} \\
    y_i &= \text{maxVal} & l_i &> \text{maxVal} \\
\end{align*}
\]

(3.8)

\[
\begin{align*}
    y_i &\leq \text{maxVal}, \\
    y_i &\geq \frac{l_i - \text{maxVal}}{u_i - l_i} \cdot x_i - \frac{l_i \cdot (\text{maxVal} - u_i)}{u_i - l_i}, & \text{minVal} \leq l_i \leq \text{maxVal} \land u_i > \text{maxVal} \\
    y_i &\leq x_i \\
\end{align*}
\]

(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, \( \text{minVal} = -1, \text{maxVal} = 1 \).
Star Set based Reachability Analysis of Neural Networks

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, \( \text{minVal} = -1, \text{maxVal} = 1 \).

\[
\begin{align*}
    y_i &\leq \text{maxVal}, \\
    y_i &\geq \text{minVal}, \\
    y_i &\leq \frac{\text{maxVal} - \text{minVal}}{\text{maxVal} - l_i} \cdot x_i - \frac{\text{maxVal} - (l_i - \text{minVal})}{\text{maxVal} - u_i} \cdot (\text{maxVal} - u_i) \\
    y_i &\geq \frac{\text{minVal} - \text{maxVal}}{\text{minVal} - u_i} \cdot x_i - \frac{\text{minVal} - (u_i - \text{maxVal})}{\text{minVal} - l_i} \cdot (\text{minVal} - l_i) \\
\end{align*}
\]

(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, \( \text{minVal} = -1, \text{maxVal} = 1 \).

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

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

• When the lower bound $l_i$ is between $\text{minVal}$ and $\text{maxVal}$ and the upper bound $u_i$ is over $\text{maxVal}$. To capture the over-approximation at the $i^{th}$ neuron, we introduce a new variable $\alpha_{m+1}$. 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 $\alpha' = [\alpha_1 \ldots \alpha_m \alpha_{m+1}]$:

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

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

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

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

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

• If the set is over $\text{minVal}$ and $\text{maxVal}$, i.e., the lower bound $l_i$ is less than $\text{minVal}$, and the upper bound $u_i$ is greater than $\text{maxVal}$. We introduce a new variable $\alpha_{m+1}$ 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 $\alpha' = [\alpha_1 \ldots \alpha_m \alpha_{m+1}]$ in the reachable set holds:

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

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

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

**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 $\Theta = \{c, V, P\}$ 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 $d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix}$

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

Figure 3.20: $\Theta'$ after applying approxHTangent on the first dimension. Figure 3.21: $\Theta''$ after applying approxHTangent on the second dimension.
Algorithm 4 Star set based over-approximate reachability analysis for a HardTanh layer

\textbf{Constants:} \textit{minVal}, \textit{maxVal} \\
\textbf{Input:} Input star set \( I = \emptyset \) \\
\textbf{Output:} Over-approximate reachable set \( \mathcal{R} \)

1: \textbf{procedure} \textsc{LayerReach}(\( I, W, b \)) \\
2: \hspace{1em} \( I_1 \leftarrow W \ast I_0 + b = \langle Wc + b, WV, P \rangle \) \\
3: \hspace{1em} \( I' \leftarrow I_1 \) \\
4: \hspace{1em} \textbf{for} \( i = 1 : n \) \hspace{1em} \textbf{do} \hspace{1em} \triangleright\textit{n approxHTangent operations} \\
5: \hspace{2em} \( I' \leftarrow \text{approxHTangent}(I', i) \) \\
6: \hspace{1em} \( \mathcal{R}_1 \leftarrow I' \) \\
7: \textbf{end for} \\
8: \textbf{end procedure} \\

\textbf{procedure} \textsc{approxHTangent}(\( \hat{I}, i \)) \\
9: \hspace{1em} \( \hat{I} \leftarrow \hat{\Theta} = \langle \hat{c}, \hat{V}, \hat{P} \rangle \) \\
10: \hspace{1em} \( [l_i, u_i] \leftarrow \hat{\Theta} \cdot \text{range}(i) \) \\
11: \hspace{1em} \( \hat{M} \leftarrow [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n] \) \\
12: \hspace{1em} \( s \leftarrow [0 \ldots \text{maxVal} \ldots 0]^T \) \\
13: \hspace{1em} \( s' \leftarrow [0 \ldots \text{minVal} \ldots 0]^T \) \\
14: \hspace{1em} \textbf{if} \( l_i \geq \text{minVal} \) \textbf{and} \( u_i \leq \text{maxVal} \) \textbf{then} \\
15: \hspace{2em} \( \mathcal{R}_1 \leftarrow \hat{\Theta}_j = \langle \hat{c}_j, \hat{V}_j, \hat{P}_j \rangle \) \\
16: \hspace{1em} \textbf{else if} \( u_i < \text{minVal} \) \textbf{then} \\
17: \hspace{2em} \( \hat{\Theta}_j \leftarrow M \ast \hat{\Theta}_j = \langle M\hat{c}_j + s', M\hat{V}_j, \hat{P}_j \rangle \) \\
18: \hspace{1em} \textbf{end if} \\
19: \hspace{1em} \textbf{else if} \( l_i > \text{maxVal} \) \textbf{then} \\
20: \hspace{2em} \( \hat{\Theta}_j \leftarrow M \ast \hat{\Theta}_j = \langle M\hat{c}_j + s, M\hat{V}_j, \hat{P}_j \rangle \) \\
21: \hspace{1em} \textbf{end if} \\
22: \hspace{1em} \textbf{else if} \( \text{minVal} \leq l_i \leq \text{maxVal} \) \textbf{and} \( u_i > \text{maxVal} \) \textbf{then} \\
23: \hspace{2em} \( P'(\alpha') \triangleq \hat{C} \alpha \leq d, \alpha = [\alpha_1 \ldots \alpha_m]^T \) \hspace{1em} \triangleright\textit{New variable} \( \alpha_{m+1} \) \\
24: \hspace{2em} \( \alpha' \leftarrow [\alpha_1 \ldots \alpha_m \alpha_{m+1}]^T \) \\
25: \hspace{2em} \( C_1 \leftarrow [0 \ldots 0 1], d_1 \leftarrow \text{maxVal} \) \\
26: \hspace{2em} \( C_2 \leftarrow [\hat{V}_1 1], d_2 \leftarrow \hat{c}_i \) \\
27: \hspace{2em} \( C_3 \leftarrow [-u_{i-1}-\text{maxVal} \hat{V}_i 1], d_3 \leftarrow \hat{c}_i (l_i-\text{maxVal})+1 (\text{maxVal}-u_i) \) \\
28: \hspace{2em} \( C_0 \leftarrow [\hat{C} 0_{\maxVal}], d_0 \leftarrow \hat{d} \) \\
29: \hspace{2em} \( C' \leftarrow C_0 C_1 C_2 C_3, d' \leftarrow [d_0 d_1 d_2 d_3] \) \\
30: \hspace{2em} \( P'(\alpha') \triangleq C'\alpha' \leq d' \) \\
31: \hspace{2em} \( c' \leftarrow M\hat{c}_i, V' \leftarrow M\hat{V}_i, P' \leftarrow [V' e_i] \) \\
32: \hspace{1em} \( \mathcal{R}_1 \leftarrow (c', V', P') \) \\
33: \hspace{1em} \textbf{else if} \( l_i < \text{minVal} \) \textbf{and} \( u_i \leq \text{maxVal} \) \textbf{then} \\
34: \hspace{2em} \( P'(\alpha') \triangleq \hat{C} \alpha \leq d, \alpha = [\alpha_1 \ldots \alpha_m]^T \) \hspace{1em} \triangleright\textit{New variable} \( \alpha_{m+1} \) \\
35: \hspace{2em} \( \alpha' \leftarrow [\alpha_1 \ldots \alpha_m \alpha_{m+1}]^T \) \\
36: \hspace{2em} \( C_1 \leftarrow [0 \ldots 0 1], d_1 \leftarrow \text{minVal} \) \\
37: \hspace{2em} \( C_2 \leftarrow [\hat{V}_1 -1], d_2 \leftarrow \hat{c}_i \) \\
38: \hspace{2em} \( C_3 \leftarrow [-u_{i-1}-\text{minVal} \hat{V}_i 1], d_3 \leftarrow \hat{c}_i (u_i-\text{minVal})+1 (\text{minVal}-u_i) \) \\
39: \hspace{2em} \( C_0 \leftarrow [\hat{C} 0_{\maxVal}], d_0 \leftarrow \hat{d} \) \\
40: \hspace{2em} \( C' \leftarrow C_0 C_1 C_2 C_3, d' \leftarrow [d_0 d_1 d_2 d_3] \) \\
41: \hspace{2em} \( P'(\alpha') \triangleq C'\alpha' \leq d' \) \\
42: \hspace{2em} \( c' \leftarrow M\hat{c}_i, V' \leftarrow M\hat{V}_i, P' \leftarrow [V' e_i] \) \\
43: \hspace{1em} \( \mathcal{R}_1 \leftarrow (c', V', P') \) \\
44: \hspace{1em} \textbf{else} \hspace{1em} \textbf{if} \( l_i < \text{minVal} \) \textbf{and} \( u_i > \text{maxVal} \) \textbf{then} \\
45: \hspace{2em} \( P'(\alpha') \triangleq \hat{C} \alpha \leq d, \alpha = [\alpha_1 \ldots \alpha_m]^T \) \hspace{1em} \triangleright\textit{New variable} \( \alpha_{m+1} \) \\
46: \hspace{2em} \( \alpha' \leftarrow [\alpha_1 \ldots \alpha_m \alpha_{m+1}]^T \) \\
47: \hspace{2em} \( C_1 \leftarrow [0 \ldots 0 1], d_1 \leftarrow \text{minVal} \) \\
48: \hspace{2em} \( C_2 \leftarrow [\text{maxVal}-\text{minVal} \hat{V}_i 1], d_2 \leftarrow \hat{c}_i (\text{maxVal}-\text{minVal})+1 (\text{minVal}-u_i) \) \\
49: \hspace{2em} \( C_3 \leftarrow [\text{minVal}-\text{maxVal} \hat{V}_i -1], d_3 \leftarrow \hat{c}_i (\text{maxVal}-\text{minVal})+1 (\text{maxVal}-u_i) \) \\
50: \hspace{2em} \( C_0 \leftarrow [\hat{C} 0_{\maxVal}], d_0 \leftarrow \hat{d} \) \\
51: \hspace{2em} \( C' \leftarrow C_0 C_1 C_2 C_3, d' \leftarrow [d_0 d_1 d_2 d_3] \) \\
52: \hspace{2em} \( P'(\alpha') \triangleq C'\alpha' \leq d' \) \\
53: \hspace{2em} \( c' \leftarrow M\hat{c}_i, V' \leftarrow M\hat{V}_i, P' \leftarrow [V' e_i] \) \\
54: \hspace{1em} \( \mathcal{R}_1 \leftarrow (c', V', P') \)
Unbounded Analysis

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

- In case, the input star set \( \Theta \) has no upper bound and accordingly increases infinitely without reaching a maximum value. By executing the exactHTangent operation to compute an exact reachability set for the input set without an upper bound, in comparison, the exactHTangent 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

\[
\begin{align*}
  &\begin{cases}
    y_i \leq maxVal, \\
    y_i \geq l_i & \text{minVal} \leq l_i \leq maxVal \\
    y_i \leq x_i & l_i < \text{minVal}
  \end{cases}
\end{align*}
\]

\( (3.12) \)

![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.](image)

\[
\begin{align*}
  &\begin{cases}
    y_i \leq maxVal, \\
    y_i \geq minVal, \\
    y_i \leq \frac{maxVal-minVal}{maxVal-l_i} \cdot x_i - \frac{maxVal \cdot (l_i-minVal)}{maxVal-l_i} & l_i < \text{minVal}
  \end{cases}
\end{align*}
\]

\( (3.13) \)
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, \( \text{minVal} = -1 \), \( \text{maxVal} = 1 \).

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 \( \text{minVal} \) and \( \text{maxVal} \). In that case, we introduce a new variable \( \alpha_{m+1} \) to encode the over-approximation of the activation function according to Equation 3.12, which will result in three more linear constraints:

\[
\alpha_{m+1} \leq \text{maxVal}, \quad \alpha_{m+1} \geq l_i, \quad \alpha_{m+1} \leq x_i
\]

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

\[
\alpha_{m+1} \leq \text{maxVal}, \quad \alpha_{m+1} \geq \text{minVal}, \\
\alpha_{m+1} \leq \frac{\text{maxVal} - \text{minVal}}{\text{maxVal} - l_i} \cdot x_i - \frac{\text{maxVal} \cdot (l_i - \text{minVal})}{\text{maxVal} - l_i}
\]

- The second case is when the input star set \( \Theta \) has no lower bound, i.e., the input extends infinitely to the negative direction without a minimum limit. Likewise, 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 \( \text{minVal} \), we project the input onto \( \text{minVal} \), as in Algorithm 4 line 13. If the upper bound is in the range between \( \text{minVal} \) and \( \text{maxVal} \), then we handle the set the same as in line 32 in Algorithm 4. Lastly, if our upper bound is greater than \( \text{maxVal} \), 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 \( y_i = \text{HardTanh}(x_i) \), let

\[
\begin{aligned}
 y_i \geq \text{minVal}, \\
y_i \leq u_i & \quad \text{minVal} \leq u_i \leq \text{maxVal} \\
y_i \geq x_i
\end{aligned}
\]
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.

\[
\begin{align*}
\begin{cases}
y_i \geq \text{minVal}, \\
y_i \leq \text{maxVal}, \\
y_i \geq \frac{\text{minVal} - \text{maxVal}}{\text{minVal} - u_i}, x_i = \frac{\text{minVal}(\text{maxVal} - u_i)}{\text{minVal} - u_i}
\end{cases}
\end{align*}
\]  

(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 \( \text{minVal} \) and \( \text{maxVal} \), we introduce the new variable \( \alpha_{m+1} \) encoding the over-approximation according to Equation 3.14 resulting in three more constraints:

\[
\alpha_{m+1} \geq \text{minVal}, \quad \alpha_{m+1} \leq u_i, \quad \alpha_{m+1} \geq x_i
\]

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

\[
\alpha_{m+1} \geq \text{minVal}, \quad \alpha_{m+1} \leq \text{maxVal},
\]
\[
\alpha_{m+1} \geq \frac{\text{minVal} - \text{maxVal}}{\text{minVal} - u_i} \cdot x_i - \frac{\text{minVal} \cdot (\text{maxVal} - u_i)}{\text{minVal} - u_i}
\]

- 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 \( y_i = \text{HardTanh}(x_i) \), let

\[
\begin{align*}
\{ y_i & \geq \text{minVal}, \ y_i \leq \text{maxVal} \quad x_i \in \mathbb{R} \\
\}
\end{align*}
\]  

(3.16)

Introducing the new variable \( \alpha_{m+1} \) to the predicate will result in these two new constraints

\[
\alpha_{m+1} \geq \text{minVal}, \quad \alpha_{m+1} \leq \text{maxVal}
\]

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 \( x \),

\[
\text{HardSigmoid}(x) = \begin{cases} 
0 & x \leq -1 \\
1 & x \geq 1 \\
\frac{1}{2} \cdot x + \frac{1}{2} & -1 < x < 1
\end{cases} = \max(0, \min\left(\frac{1}{2} \cdot x + \frac{1}{2}\right))
\]  

(3.17)
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.

\[
HardSigmoid(x) = \begin{cases} 
0 & x \leq \text{minVal} \\
1 & x \geq \text{maxVal} \\
\frac{1}{\text{maxVal} - \text{minVal}} \cdot x + \frac{\text{minVal}}{\text{maxVal} - \text{minVal}} & \text{minVal} < x < \text{maxVal} 
\end{cases}
\]

We will discuss the analyses in the following sections using this function \ref{eq: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 \( \Theta \). For a layer with \( n \) neurons, we apply a sequence of \( n \) exactHSigmoid operation \( \mathcal{R}_k = \text{HardSigmoid}_n(\text{HardSigmoid}_{n-1}(\ldots \text{HardSigmoid}_1(\Theta))) \). As described in Algorithm \ref{algo:exact}, we start by computing the lower and upper bounds \( l, u \) on the \( i^{th} \) neuron. The function distinguishes six different cases.

- If the lower and upper bounds between \( \text{minVal} \) and \( \text{maxVal} \), we apply the HardSigmoid function on the \( x_i \) of the vector \( x = [x_1 \ldots x_n]^T \), leading to a new vector \( x' = [x_1 \ldots x_{i-1} \frac{1}{\text{maxVal} - \text{minVal}} x_i \frac{\text{minVal}}{\text{maxVal} - \text{minVal}} x_{i+1} \ldots x_n]^T \). This procedure is equivalent to mapping the input set by the scaling matrix \( M' = [c_1 \ldots c_{i-1} \frac{1}{\text{maxVal} - \text{minVal}} c_i \frac{\text{minVal}}{\text{maxVal} - \text{minVal}} c_{i+1} \ldots c_n] \). Afterward, we set the center \( c_i \) to \( \frac{\text{minVal}}{\text{maxVal} - \text{minVal}} \).
Example 3.1.5. Let $\Theta = \{c, V, P\}$ 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 \( d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix} \)

We apply the exactHSigmoid operation on the dimensions of \( \Theta \) with \( \text{minVal} = -2 \) and \( \text{maxVal} = 2 \), resulting in:

Figure 3.28: The input star set \( \Theta \) corresponding to Example 3.1.5

Figure 3.29: \( \Theta' \) after applying exactHSigmoid on the first dimension.

Figure 3.30: \( \Theta'' \) after applying exactHSigmoid on the second dimension.
\textbf{Algorithm 5} Star set based exact reachability analysis for a HardSigmoid layer

\begin{itemize}
  \item \textbf{Constants:} minVal, maxVal
  \item \textbf{Input:} Input star set $I = [\Theta_1 \ldots \Theta_N]$
  \item \textbf{Output:} Exact reachable set $R$
\end{itemize}

1: \textbf{procedure} \texttt{LAYERREACH}(I, W, b) \\
2: \hspace{1em} $R \leftarrow \emptyset$
3: \hspace{1em} \textbf{for} $j = 1 : N$ \textbf{do}
4: \hspace{2em} $I_1 \leftarrow W \ast \Theta_j + b = \langle Wc_j + b, WV_j, P_j \rangle$
5: \hspace{2em} $R_1 \leftarrow I_1$
6: \hspace{2em} \textbf{for} $i = 1 : n$ \textbf{do} \Comment*[r]{n exactHTangent operations}
7: \hspace{3em} $R_1 \leftarrow \texttt{exactHSigmoid}(R_1, i, l_i, u_i)$
8: \hspace{2em} $R \leftarrow R \cup R_1$
9: \textbf{return} $R$

10: \textbf{procedure} \texttt{EXACTSIGMOID}(I, i, l_i, u_i) \Comment*[r]{Intermediate representations}
11: \hspace{1em} $R \leftarrow \emptyset$, $I = [\Theta_1 \ldots \Theta_k]$
12: \hspace{1em} \textbf{for} $j = 1 : k$ \textbf{do}
13: \hspace{2em} $[l_i, u_i] \leftarrow \Theta_j$.range($i$) \Comment*[r]{l_i \leq x_i \leq u_i, x_i \in I_i[i]}$
14: \hspace{2em} $R_1 \leftarrow \emptyset$
15: \hspace{2em} $M \leftarrow [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$
16: \hspace{2em} $M' \leftarrow [e_1 \ldots e_{i-1} \maxVal - \minVal e_{i+1} \ldots e_n]$
17: \hspace{2em} $s \leftarrow [0 \ldots \minVal \maxVal \ldots 0]^T$
18: \hspace{2em} $s' \leftarrow [0 \ldots 1 \ldots 0]^T$
19: \hspace{2em} \textbf{if} $l_i > \minVal$ \textbf{and} $u_i < \maxVal$ \textbf{then}
20: \hspace{3em} $\Theta \leftarrow M' \ast \Theta_j = \langle \tilde{c}_j + s, \tilde{M}_j, \tilde{P}_j \rangle$
21: \hspace{3em} $R_1 \leftarrow \Theta_j$
22: \hspace{2em} \textbf{else if} $u_i \leq \minVal$ \textbf{then}
23: \hspace{3em} $R_1 \leftarrow M \ast \Theta_j = \langle \tilde{c}_j, \tilde{M}_j, \tilde{P}_j \rangle$
24: \hspace{2em} \textbf{else if} $l_i \geq \maxVal$ \textbf{then}
25: \hspace{3em} $\Theta_j \leftarrow M \ast \Theta_j = \langle \tilde{c}_j + s', \tilde{M}_j, \tilde{P}_j \rangle$
26: \hspace{2em} $R_1 \leftarrow \Theta_j$
27: \hspace{2em} \textbf{else if} $\minVal < l_i < \maxVal$ \textbf{and} $u_i \geq \maxVal$ \textbf{then}
28: \hspace{3em} $\Theta'_j \leftarrow \Theta_j \land \minVal < x[i] < \maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j' \rangle$
29: \hspace{3em} $\Theta' \leftarrow M' \ast \Theta'_j = \langle \tilde{M}_j + s, \tilde{M}_j, \tilde{P}_j' \rangle$
30: \hspace{3em} $\Theta'' \leftarrow \Theta_j \land x[i] \geq \maxVal = \langle \tilde{c}_j', \tilde{V}_j', \tilde{P}_j'' \rangle$
31: \hspace{3em} $\Theta_j'' \leftarrow M \ast \Theta'' \leftarrow \langle \tilde{c}_j' + s', \tilde{M}_j', \tilde{P}_j'' \rangle$
32: \hspace{3em} $R_1 \leftarrow \Theta_j'' \cup \Theta_j''$
33: \hspace{2em} \textbf{else if} $l_i \leq \minVal$ \textbf{and} $u_i < \maxVal$ \textbf{then}
34: \hspace{3em} $\Theta'_j \leftarrow \Theta_j \land \minVal < x[i] < \maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j' \rangle$
35: \hspace{3em} $\Theta' \leftarrow M' \ast \Theta'_j = \langle \tilde{M}_j + s, \tilde{M}_j, \tilde{P}_j' \rangle$
36: \hspace{3em} $\Theta'' \leftarrow \Theta_j \land x[i] \leq \minVal = \langle \tilde{c}_j', \tilde{V}_j', \tilde{P}_j'' \rangle$
37: \hspace{3em} $R_1 \leftarrow \Theta_j'' \cup M \ast \Theta_j''$
38: \hspace{2em} \textbf{else} \Comment*[r]{l_i \leq \minVal and u_i \geq \maxVal}
39: \hspace{3em} $\Theta'_j \leftarrow \Theta_j \land \minVal < x[i] < \maxVal = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j' \rangle$
40: \hspace{3em} $\Theta' \leftarrow M' \ast \Theta'_j = \langle \tilde{M}_j + s, \tilde{M}_j, \tilde{P}_j' \rangle$
41: \hspace{3em} $\Theta'' \leftarrow \Theta_j \land x[i] \leq \minVal = \langle \tilde{c}_j', \tilde{V}_j', \tilde{P}_j'' \rangle$
42: \hspace{3em} $\Theta'' \leftarrow M \ast \Theta'' \leftarrow \langle \tilde{c}_j' + s', \tilde{M}_j', \tilde{P}_j''' \rangle$
43: \hspace{3em} $\Theta_j''' \leftarrow M \ast \Theta_j'' \leftarrow \langle \tilde{c}_j'' + s', \tilde{M}_j'', \tilde{P}_j''' \rangle$
44: \hspace{3em} $R_1 \leftarrow \Theta_j''' \cup \Theta_j''' \cup \Theta_j'''$
45: \hspace{1em} $R \leftarrow R \cup R_1$
46: \textbf{return} $R$
Lemma 3.1.17. The worst-case complexity of the number of stars in the reachable set of an N-neurons FNN is $\Theta(3^N)$.

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 $\Theta(2N)$.

Over-approximate Analysis

In this section, we want to discuss the over-approximation reachability algorithm for HardSigmoid. 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 $y_i = \text{HardSigmoid}(x_i)$, let

$$
\begin{aligned}
    y_i &= \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i + \frac{\text{minVal}}{\text{minVal} - \text{maxVal}} \quad l_i > \text{minVal} \land u_i < \text{maxVal} \\
    y_i &= 0 \quad u_i \leq \text{minVal} \\
    y_i &= 1 \quad l_i \geq \text{maxVal}
\end{aligned}
$$

(3.19)

$$
\begin{aligned}
    y_i &\leq 1, \\
    y_i &\geq \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i + \frac{\text{minVal}}{\text{minVal} - \text{maxVal}},
\end{aligned}
$$

(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, $\text{minVal} = -1$, $\text{maxVal} = 1$. 

$$
\begin{aligned}
    y_i &\geq 0, \\
    y_i &\geq \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i + \frac{\text{minVal}}{\text{minVal} - \text{maxVal}},
\end{aligned}
$$

(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.

$$\left\{ \begin{array}{ll}
y_i \leq 1, \\
y_i \geq 0,
\end{array} \right. \quad \begin{aligned}
l_i &\leq \text{minVal} \\
u_i &\geq \text{maxVal}
\end{aligned} \quad (3.22)$$

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 \( \Theta \), 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 \( \text{minVal} \) and \( \text{maxVal} \), we scale the input set by the scaling matrix
  \[
  M' = [e_1 \ldots e_{i-1} \frac{1}{\text{maxVal}-\text{minVal}} e_{i+1} \ldots e_n],
  \]
  then adjust the center
  \[
  c_i = \frac{\text{minVal}}{\text{minVal}-\text{maxVal}}
  \]
- If the upper bound \( u_i \) is less than \( \text{minVal} \), 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 \( \text{maxVal} \), we project the input set onto one by mapping the set by the mapping matrix \( M = [e_1 \ldots e_{i-1} \ 0 \ e_{i+1} \ldots e_n] \) and afterward adjust \( c_i \) to 1.

\[
\begin{align*}
\alpha_{m+1} &\leq 1, \\
\alpha_{m+1} &\leq \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i - \frac{\text{minVal}}{\text{maxVal} - \text{minVal}} \\
\alpha_{m+1} &\geq \frac{l_i - 1}{l_i - u_i} \cdot x_i + \frac{l_i \cdot (1 - u_i)}{l_i - u_i}
\end{align*}
\]

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

\[
\begin{align*}
\alpha_{m+1} &\geq 0, \\
\alpha_{m+1} &\geq \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i + \frac{\text{minVal}}{\text{maxVal} - \text{minVal}} \\
\alpha_{m+1} &\leq \frac{u_i \cdot (x_i - l_i)}{u_i - l_i}
\end{align*}
\]

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

\[
\begin{align*}
\alpha_{m+1} &\leq 1, \\
\alpha_{m+1} &\leq 0, \\
\alpha_{m+1} &\leq \frac{1}{\text{maxVal} - l_i} \cdot x_i - \frac{l_i}{\text{maxVal} - l_i}, \\
\alpha_{m+1} &\geq \frac{1}{\text{minVal} - u_i} \cdot x_i - \frac{\text{minVal}}{\text{u}_i - \text{minVal}}
\end{align*}
\]

**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 \( \Theta = (c, V, P) \) be the input set where:

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

also the predicate \( P(\alpha) \triangleq C\alpha \leq d \) where \( C = \begin{bmatrix} -2.5 & 1.2 \\ -2.5 & 1.4 \\ -0.9 & -3.8 \end{bmatrix} \) and \( d = \begin{bmatrix} 6.7 \\ 4.1 \\ 7.86 \\ 6.4 \\ 6.42 \end{bmatrix} \).
We apply the `approxHSigmoid` operation on the dimensions of $\Theta$ with $\text{minVal} = -2$ and $\text{maxVal} = 2$, resulting in:

**Figure 3.35:** $\Theta'$ after applying `approxHSigmoid` on the first dimension.

**Figure 3.36:** $\Theta''$ 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 $I = \emptyset$

**Output:** Over-approximate reachable set $\mathcal{R}$

1. procedure LAYERREACH($I, W, b$)
2.   $I_0 \leftarrow W \ast I_0 + b = (Wc + b, WV, P)$
3.   $I' \leftarrow I_0$
4.   for $i = 1 : n$
5.     $I' \leftarrow \text{approxHSigmoid}(I', i)$ \hspace{1cm} \(\triangleright n\ \text{approxHSigmoid operations}\)
6. \hspace{1cm} $\mathcal{R}_1 \leftarrow I'$
7. procedure APPROXHSIGMOID($\bar{I}, i$)
8.   $\bar{I} \leftarrow \Theta = (\hat{e}, V, P)$
9.   \[ [l_i, u_i] \leftarrow \Theta \cdot \text{range}(i) \]
10. $M \leftarrow [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$ \hspace{1cm} \(\triangleright \text{New variable } \alpha_{m+1}\)
11. $M' \leftarrow [e_1 \ldots e_{i-1} \frac{\text{minVal} - \text{minVal}(i, (i-1-u_i))}{\text{maxVal} - \text{minVal}} e_{i+1} \ldots e_n]$ \hspace{1cm} \(\triangleright \text{New variable } \alpha_{m+1}\)
12. $s \leftarrow [0 \ldots \frac{\text{maxVal} - \text{minVal}(i, (i-1-u_i))}{\text{maxVal} - \text{minVal}} \ldots 0]^T$, $s' \leftarrow [0 \ldots 1 \ldots 0]^T$
13. if $l_i > \text{minVal} \text{ and } u_i < \text{maxVal}$ then
14.     $\Theta \leftarrow M' \ast \Theta = (\hat{M}\tilde{c} + s, MVj_p, P)$ \hspace{1cm} \(\triangleright \text{New variable } \alpha_{m+1}\)
15.     $\mathcal{R}_1 \leftarrow \Theta$
16. else if $u_i < \text{minVal}$ then
17.     $\mathcal{R}_1 \leftarrow (\hat{M}, MVj_p, P)$
18. else if $l_i \geq \text{maxVal}$ then
19.     $\Theta \leftarrow M \ast \Theta = (\hat{M}\tilde{c} + s', MVj_p, P)$
20.     $\mathcal{R}_1 \leftarrow \Theta$
21. else if $\text{minVal} < l_i < \text{maxVal} \text{ and } u_i \geq \text{maxVal}$ then
22.     $P'(\alpha) \triangleq \tilde{C}\alpha \leq d, \alpha = [\alpha_1 \ldots \alpha_m]^T$
23.     $\alpha' \leftarrow [\alpha_1 \ldots \alpha_{m+1}]^T$ \hspace{1cm} \(\triangleright \text{New variable } \alpha_{m+1}\)
24.     $C_1 \leftarrow [0 \ldots 0 1]$, $d_1 \leftarrow 1$
25.     $C_2 \leftarrow [\frac{1}{\text{maxVal} - \text{minVal}} \tilde{V}_1 - 1]$, $d_2 \leftarrow \frac{\tilde{e}_1 - \text{minVal}}{\text{maxVal} - \text{minVal}}$
26.     $C_i \leftarrow [\frac{1}{\text{maxVal} - \text{minVal}} \tilde{V}_i - 1]$, $d_2 \leftarrow \frac{\tilde{e}_i - \text{minVal}}{\text{maxVal} - \text{minVal}}$
27.     $C_0 \leftarrow [\tilde{C} \hspace{0.2em} 0_{m \times 1}], d_0 \leftarrow \vec{d}$
28.     $C' \leftarrow [C_0 \hspace{0.2em} C_1 \hspace{0.2em} C_2 \hspace{0.2em} C_3], d' \leftarrow [d_0 \hspace{0.2em} d_1 \hspace{0.2em} d_2 \hspace{0.2em} d_3]$
29.     $P'(\alpha') \triangleq C'\alpha' \leq d'$
30.     $c' \leftarrow \hat{M}\tilde{c}, V' \leftarrow MVj_p, V' \leftarrow [V' \hspace{0.2em} e_i]$
31.     $\mathcal{R} \leftarrow \langle c', V', P \rangle$
32. else if $l_i \leq \text{minVal} \text{ and } u_i < \text{maxVal}$ then
33.     $P'(\alpha) \triangleq \tilde{C}\alpha \leq d, \alpha = [\alpha_1 \ldots \alpha_m]^T$
34.     $\alpha' \leftarrow [\alpha_1 \ldots \alpha_{m+1}]^T$ \hspace{1cm} \(\triangleright \text{New variable } \alpha_{m+1}\)
35.     $C_1 \leftarrow [0 \ldots 0 - 1]$, $d_1 \leftarrow 0$
36.     $C_2 \leftarrow [\frac{1}{\text{maxVal} - \text{minVal}} \tilde{V}_1 - 1]$, $d_2 \leftarrow -\frac{\tilde{e}_1 - \text{minVal}}{\text{maxVal} - \text{minVal}}$
37.     $C_i \leftarrow [\frac{1}{\text{maxVal} - \text{minVal}} \tilde{V}_i - 1]$, $d_2 \leftarrow -\frac{\tilde{e}_i - \text{minVal}}{\text{maxVal} - \text{minVal}}$
38.     $C_0 \leftarrow [\tilde{C} \hspace{0.2em} 0_{m \times 1}], d_0 \leftarrow \vec{d}$
39.     $C' \leftarrow [C_0 \hspace{0.2em} C_1 \hspace{0.2em} C_2 \hspace{0.2em} C_3], d' \leftarrow [d_0 \hspace{0.2em} d_1 \hspace{0.2em} d_2 \hspace{0.2em} d_3]$
40.     $P'(\alpha') \triangleq C'\alpha' \leq d'$
41.     $c' \leftarrow \hat{M}\tilde{c}, V' \leftarrow MVj_p, V' \leftarrow [V' \hspace{0.2em} e_i]$
42.     $\mathcal{R} \leftarrow \langle c', V', P \rangle$
43. else \hspace{1cm} \(\triangleright \text{New variable } \alpha_{m+1}\)
44.     $P'(\alpha) \triangleq \tilde{C}\alpha \leq d, \alpha = [\alpha_1 \ldots \alpha_m]^T$
45.     $\alpha' \leftarrow [\alpha_1 \ldots \alpha_{m+1}]^T$
46.     $C_1 \leftarrow [0 \ldots 0 - 1]$, $d_1 \leftarrow 0$
47.     $C_2 \leftarrow [0 \ldots 0 1]$, $d_1 \leftarrow 1$
48.     $C_3 \leftarrow [\frac{1}{\text{maxVal} - \text{minVal}} \tilde{V}_1 - 1]$, $d_3 \leftarrow -\frac{\tilde{e}_1 - \text{minVal}}{\text{maxVal} - \text{minVal}}$
49.     $C_i \leftarrow [\frac{1}{\text{maxVal} - \text{minVal}} \tilde{V}_i - 1]$, $d_3 \leftarrow -\frac{\tilde{e}_i - \text{minVal}}{\text{maxVal} - \text{minVal}}$
50.     $C_0 \leftarrow [\tilde{C} \hspace{0.2em} 0_{m \times 1}], d_0 \leftarrow \vec{d}$
51.     $C' \leftarrow [C_0 \hspace{0.2em} C_1 \hspace{0.2em} C_2 \hspace{0.2em} C_3], d' \leftarrow [d_0 \hspace{0.2em} d_1 \hspace{0.2em} d_2 \hspace{0.2em} d_3]$
52.     $P'(\alpha') \triangleq C'\alpha' \leq d'$
53.     $c' \leftarrow \hat{M}\tilde{c}, V' \leftarrow MVj_p, V' \leftarrow [V' \hspace{0.2em} e_i]$
54.     $\mathcal{R} \leftarrow \langle c', V', P \rangle$
Unbounded Analysis

In this section, we present the exact and over-approximative analysis for an unbounded input star set $\Theta = (c, V, P)$. 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 exact HSigmoid 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 $y_i = \text{HardSigmoid}(x_i)$, let

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

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, $minVal = -1$, $maxVal = 1$. 

$$
\begin{align}
    y_i &\leq 1, \\
    y_i &\geq 0, \\
    y_i &\leq \frac{1}{maxVal-minVal} \cdot x_i + \frac{minVal}{min-maxVal} \\
    l_i &\leq minVal
\end{align}
$$
where $l_i$ is the lower bound of $x_i$.

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

\[
\begin{align*}
\alpha_{m+1} &\leq 1, \quad \alpha_{m+1} \geq \frac{1}{\text{maxVal} - \text{minVal}} \cdot l_i + \frac{\text{minVal}}{\text{min} - \text{maxVal}}, \\
\alpha_{m+1} &\leq \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i + \frac{\text{minVal}}{\text{min} - \text{maxVal}}
\end{align*}
\]

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

\[
\begin{align*}
\alpha_{m+1} &\leq 1, \quad \alpha_{m+1} \geq 0 \\
\alpha_{m+1} &\leq \frac{1}{\text{maxVal} - \text{minVal}} \cdot x_i + \frac{\text{minVal}}{\text{min} - \text{maxVal}}
\end{align*}
\]

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 exact\text{HSigmoid} 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 $\text{minVal}$, we project the set onto zero as in line 22 Algorithm [5]. If the upper bound is between $\text{minVal}$ and $\text{maxVal}$, we treat the set the same as in line 33 in Algorithm [5]. In case the upper bound is greater than $\text{maxVal}$, 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 $y_i = \text{HardSigmoid}(x_i)$, let

\[
\begin{align*}
y_i &\geq 0, \\
y_i &\leq \frac{1}{\text{maxVal} - \text{minVal}} \cdot u_i + \frac{\text{minVal}}{\text{min} - \text{maxVal}} \quad \text{minVal} < u_i < \text{maxVal}
\end{align*}
\]
Implementation

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, \( \minVal = -1, \maxVal = 1 \).

\[
\begin{align*}
\begin{cases}
    y_i &\geq 0, \\
    y_i &\leq 1, \\
    y_i &\geq \frac{\minVal - \maxVal}{u_i - \minVal} \cdot x_i - \frac{\minVal (\maxVal - u_i)}{u_i - \minVal}
\end{cases}
\end{align*}
\]

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 \( \alpha_{m+1} \) encoding the over-approximation according to equation 3, resulting in three new constraints:

\[
\begin{align*}
\alpha_{m+1} &\geq 0, \\
\alpha_{m+1} &\leq \frac{1}{\maxVal - \minVal} \cdot u_i + \frac{\minVal}{\minVal - \maxVal} \\
\alpha_{m+1} &\geq \frac{1}{\maxVal - \minVal} \cdot x_i + \frac{\minVal}{\minVal - \maxVal}
\end{align*}
\]
If the upper bound exceeds \( \text{maxVal} \), we introduce a new variable \( \alpha_{m+1} \) to represent the over-approximation. Coordinately, we have three new constraints:

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

- 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 \( y_i = \text{HardSigmoid}(x_i) \), let

\[
\begin{cases}
y_i \geq 0, \quad y_i \leq 1 \\
x_i \in \mathbb{R}
\end{cases}
\]

Since the input star set does not have an upper or lower bound, by introducing a new variable \( \alpha_{m+1} \), 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

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

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.

$$\text{unitStep}(x) = \begin{cases} 
\text{minRes} & x < \text{val} \\
\text{maxRes} & x \geq \text{val} 
\end{cases}$$ \hspace{1cm} (3.29)

where $\text{val}$ is the value that serves as the separator for our values, $\text{minRes}$ and $\text{maxRes}$ 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., $\text{unitStep}_i(\cdot)$, 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 $\text{val}$, we project the set onto $\text{minRes}$ by mapping the set with the mapping matrix $M = [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$ and then setting $c_i$ to $\text{minRes}$.
- In the opposite case, when the lower bound $l_i$ is greater than $\text{val}$, we project the set onto $\text{maxRes}$ by mapping the set with the mapping matrix $M = [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$ and set $c_i$ to $\text{maxRes}$.
- Otherwise, we partition the input into two subsets $\Theta_1 = \Theta \land (x_i < \text{val})$ and $\Theta_2 = \Theta \land (x_i \geq \text{val})$. By definition (3.29) for $\Theta_1$, we project the set onto $\text{minRes}$ the same way in the first case, first by mapping the set with the mapping matrix $M = [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$, then adjusting $c_i$ to $\text{minRes}$. For
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 $I = [\Theta_1, \ldots, \Theta_N]$

Output: Exact reachable set $\mathcal{R}$

1: procedure $\text{LAYERREACH}(I, W, b)$
2: \hspace{1em} $\mathcal{R} \leftarrow \emptyset$
3: for $j = 1 : N$ do
4: \hspace{1em} $I_j \leftarrow W \ast \Theta_j + b = (Wc_j + b, Wv_j, P_j)$
5: \hspace{1em} $\mathcal{R}_1 \leftarrow I_1$
6: \hspace{1em} for $i = 1 : n$ do \hfill $\triangleright$ exactUStep operations
7: \hspace{1em} \hspace{1em} $\mathcal{R}_1 \leftarrow \text{exactUStep}(\mathcal{R}_1, i, l_i, u_i)$
8: \hspace{1em} $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_1$
9: return $\mathcal{R}$
10: procedure $\text{EXACTUStep}(\bar{I}, i, l_i, u_i)$
11: \hspace{1em} $\bar{\mathcal{R}} \leftarrow \emptyset, \bar{I} = [\Theta_1, \ldots, \Theta_h]$ \hfill $\triangleright$ Intermediate representations
12: for $j = 1 : k$ do
13: \hspace{1em} $[l_i, u_i] \leftarrow \Theta_j, \text{range}(i)$ \hfill $\triangleright$ $l_i \leq x_i \leq u_i, x_i \in I_1[i]$
14: \hspace{1em} $\mathcal{R}_1 \leftarrow \emptyset, M \leftarrow [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$
15: \hspace{1em} $s \leftarrow [0 \ldots minRes \ldots 0]^T$
16: \hspace{1em} $s' \leftarrow [0 \ldots maxRes \ldots 0]^T$
17: \hspace{1em} if $u_i \leq val$ then
18: \hspace{1em} \hspace{1em} $\mathcal{R}_1 \leftarrow M \ast \Theta_j = (M\tilde{e}_j + s, M\tilde{v}_j, \tilde{P}_j)$
19: \hspace{1em} \hspace{1em} $\mathcal{R}_1 \leftarrow \Theta_j$
20: \hspace{1em} else if $l_i \geq val$ then
21: \hspace{1em} \hspace{1em} $\mathcal{R}_1 \leftarrow M \ast \Theta_j = (M\tilde{e}_j + s, M\tilde{v}_j, \tilde{P}_j)$
22: \hspace{1em} \hspace{1em} $\mathcal{R}_1 \leftarrow \Theta_j$
23: \hspace{1em} else \hfill $\triangleright$ $l_i < val$ and $u_i \geq val$
24: \hspace{1em} \hspace{1em} $\tilde{\Theta}_j \leftarrow \Theta_j \wedge x[i] < val = (\tilde{c}'_j, \tilde{v}'_j, \tilde{P}'_j)$
25: \hspace{1em} \hspace{1em} $\tilde{\Theta}_j' \leftarrow M \ast \tilde{\Theta}_j' = (M\tilde{c}'_j + s, M\tilde{v}'_j, \tilde{P}'_j)$
26: \hspace{1em} \hspace{1em} $\tilde{\Theta}_j'' \leftarrow \tilde{\Theta}_j \wedge x[i] \geq val = (\tilde{c}''_j, \tilde{v}''_j, \tilde{P}''_j)$
27: \hspace{1em} \hspace{1em} $\tilde{\Theta}_j'' \leftarrow M \ast \tilde{\Theta}_j'' = (M\tilde{c}''_j + s, M\tilde{v}''_j, \tilde{P}''_j)$
28: \hspace{1em} \hspace{1em} $\mathcal{R}_1 \leftarrow \tilde{\Theta}_j' \cup \tilde{\Theta}_j''$
29: \hspace{1em} $\bar{\mathcal{R}} \leftarrow \bar{\mathcal{R}} \cup \mathcal{R}_1$
30: return $\bar{\mathcal{R}}$

$\Theta_2$, we project it onto $maxRes$ by mapping the set with the mapping matrix $M = [e_1 \ldots e_{i-1} 0 e_{i+1} \ldots e_n]$ and changing $e_i$ to $maxRes$. Accordingly, the exactUStep operation at the $i^{th}$ neuron for $\Theta$ as input yields the union of two star sets $\text{unitStep}(\Theta) = (Mc + s, MV, P1) \cup (Mc + s', MV, P2)$, where $s, s'$ are the shifting vectors, $s = [0 \ldots minRes \ldots 0]^T$ and $s' = [0 \ldots maxRes \ldots 0]^T$. 
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 $\Theta$ 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 $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ 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:

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

(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.
Non-Fully Connected Layers

Figure 3.43: The convolution operation [Ste19]

Lemma 3.2.1 (Reachable set of a convolutional layer [Tra20]). Given an ImageStar as input $\mathcal{I} = (c,V,P)$, the reachable set of a convolutional layer is another ImageStar $\mathcal{I}' = (c',V',P')$ where $c' = \text{Convol}(c)$ is the convolution operation applied to the anchor image, $V' = \{v'_1, \ldots, v'_m\}$, $v'_i = \text{ConvolZeroBias}(v_i)$ 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 convolutional 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 stored as input and, in our case, ImageStar, implementing this layer was beyond the 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 \cite{PGCB14, Nab19}. Therefore, like the previous section, the implementation of this layer was beyond this project’s scope, but still, the research took place.

![Recurrent Neural Network](https://via.placeholder.com/150)

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 \cite{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 \cite{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. The units for the ACAS Xu DNNs’ inputs are:

- \( \rho \): distance from ownship to intruder in feet
- \( \theta \): angle to intruder relative to ownship heading direction in radians for the range \([-\tau, \tau]\)
- \( \psi \): heading angle of intruder relative to ownship heading direction in radians for the range \([-\tau, \tau]\)
- \( v_{\text{own}} \): speed of ownship in feet per second
- \( v_{\text{int}} \): speed of intruder in feet per second

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

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

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 $\tau$. For example, $N_{3,5}$ is the network trained for the case where $a_{prev} = \text{weak right}$ and $\tau = 20$. Further, different 10 properties are defined to test the networks.

- **Property $\phi_1$:**
  - 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 $\phi_2$:**
  - 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 $\phi_3$:**
  - 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 $\phi_4$:**
  - 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: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi = 0$, $v_{\text{own}} \geq 1000$, $700 \leq v_{\text{int}} \leq 800$.

The desired output property is that the score for COC is not the minimal score.

**Property $\phi_5$:**

- 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: $250 \leq \rho \leq 400$, $-0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{\text{own}} \leq 400$, $v_{\text{int}} \leq 400$.
- The desired output property is that the score for “strong right” is the minimal score.

**Property $\phi_6$:**

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

**Property $\phi_7$:**

- 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: $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq 3.141592$, $-3.141592 \leq \psi \leq 3.141592$, $100 \leq v_{\text{own}} \leq 1200$, $0 \leq v_{\text{int}} \leq 1200$.
- The desired output property is that the scores for “strong right” and “strong left” are never the minimal scores.

**Property $\phi_8$:**

- 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: $0 \leq \rho \leq 60760$, $-3.141592 \leq \theta \leq -0.75 \cdot 3.141592$, $-0.1 \leq \psi \leq 0.1$, $600 \leq v_{\text{own}} \leq 1200$, $600 \leq v_{\text{int}} \leq 1200$.
- The desired output property is that the score for “weak left” is minimal or the score for COC is minimal.

**Property $\phi_9$:**

- 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 $\phi_{10}$:**
  – 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 $7 \times$ faster, and the over-approximation method is $134 \times$ faster. Additionally, the over-approximative method is $19 \times$ 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.

<table>
<thead>
<tr>
<th>properties</th>
<th>Exact AVG RT(s)</th>
<th>Overapproximation AVG RT(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>29402.5067</td>
<td>2049.4807</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>44631.003</td>
<td>1775.056</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>254.60</td>
<td>10.38</td>
</tr>
<tr>
<td>$\phi_4$</td>
<td>140.7655</td>
<td>9.4855</td>
</tr>
<tr>
<td>Sum</td>
<td>74428.8752</td>
<td>3844.4022</td>
</tr>
</tbody>
</table>

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 $K_h \in \{15\}$ or off $K_h \in \{0\}$. 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:

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

and the predicate $P(\alpha) = C\alpha \leq d$ where $C = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}$ 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 $\text{val} = 7.5$, $\text{minRes} = 0$, and $\text{maxRes} = 15$. 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:

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

and $P(\alpha) = C\alpha \leq d$ where $C = \begin{bmatrix} 1.06383 & -0.68016 \\ 0.16627 & -0.67315 \\ -4.30272 & 17.4634 \\ 66.2391 & -268.762 \end{bmatrix}$ and $d = \begin{bmatrix} 23 \\ -22 \\ 1 \\ -1 \end{bmatrix}$

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

and $P(\alpha) = C\alpha \leq d$ where $C = \begin{bmatrix} 1.06383 & -0.68016 \\ 0.16627 & -0.67315 \\ 4.30272 & -17.4634 \end{bmatrix}$ and $d = \begin{bmatrix} 23 \\ -22 \\ 1 \\ -1 \end{bmatrix}$

Therefore, we take those star sets and construct a complete counter input set as in Theorem 2.3.3 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. 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.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Network ID</th>
<th>Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two layers</td>
<td>AC1</td>
<td>32, 16</td>
</tr>
<tr>
<td></td>
<td>AC2</td>
<td>64, 32</td>
</tr>
<tr>
<td></td>
<td>AC3</td>
<td>128, 64</td>
</tr>
<tr>
<td></td>
<td>AC4</td>
<td>256, 128</td>
</tr>
<tr>
<td>Three layers</td>
<td>AC5</td>
<td>32, 16, 8</td>
</tr>
<tr>
<td></td>
<td>AC6</td>
<td>64, 32, 16</td>
</tr>
<tr>
<td></td>
<td>AC7</td>
<td>128, 64, 32</td>
</tr>
<tr>
<td></td>
<td>AC8</td>
<td>256, 128, 64</td>
</tr>
</tbody>
</table>

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

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.

<table>
<thead>
<tr>
<th>Nx,y</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT(ms) RES VT(ms)</td>
<td>RT(ms) RES VT(ms)</td>
</tr>
<tr>
<td>AC1</td>
<td>61385 True 4925</td>
<td>199 False 19</td>
</tr>
<tr>
<td>AC1</td>
<td>526 True 13</td>
<td>60 False 2</td>
</tr>
<tr>
<td>AC2</td>
<td>462405 True 17707</td>
<td>530 False 6</td>
</tr>
<tr>
<td>AC2</td>
<td>112 True 2</td>
<td>66 False 1</td>
</tr>
<tr>
<td>AC3</td>
<td>5119 True 123</td>
<td>242 False 7</td>
</tr>
<tr>
<td>AC4</td>
<td>- - -</td>
<td>8711 False 1510</td>
</tr>
<tr>
<td>AC4</td>
<td>103014 True 5528</td>
<td>685 False 5</td>
</tr>
<tr>
<td>AC5</td>
<td>304844 True 26078</td>
<td>366 False 68</td>
</tr>
<tr>
<td>AC5</td>
<td>68 False 2</td>
<td>69 False 0</td>
</tr>
<tr>
<td>AC6</td>
<td>2651726 True 84416</td>
<td>651 False 91</td>
</tr>
<tr>
<td>AC6</td>
<td>98 False 1</td>
<td>73 False 0</td>
</tr>
<tr>
<td>AC7</td>
<td>- - -</td>
<td>2545 False 42</td>
</tr>
<tr>
<td>AC7</td>
<td>4050 True 96</td>
<td>288 False 2</td>
</tr>
<tr>
<td>AC8</td>
<td>- - -</td>
<td>65930 False 19805</td>
</tr>
<tr>
<td>AC8</td>
<td>812 False 48</td>
<td>579 False 4</td>
</tr>
</tbody>
</table>

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 \( \delta \)-locally-robust at input \( x \) if 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.

<table>
<thead>
<tr>
<th>( \delta = 0.1 )</th>
<th>( \delta = 0.01 )</th>
<th>( \delta = 0.001 )</th>
<th>( \delta = 0.0001 )</th>
<th>( \delta = 0.00001 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT</td>
<td>RES</td>
<td>RT</td>
<td>RES</td>
<td>RT</td>
</tr>
<tr>
<td>100</td>
<td>False</td>
<td>107</td>
<td>False</td>
<td>103</td>
</tr>
<tr>
<td>103</td>
<td>False</td>
<td>101</td>
<td>False</td>
<td>103</td>
</tr>
<tr>
<td>103</td>
<td>False</td>
<td>102</td>
<td>False</td>
<td>104</td>
</tr>
<tr>
<td>105</td>
<td>False</td>
<td>116</td>
<td>False</td>
<td>107</td>
</tr>
<tr>
<td>107</td>
<td>False</td>
<td>109</td>
<td>False</td>
<td>105</td>
</tr>
<tr>
<td>103</td>
<td>True</td>
<td>107</td>
<td>True</td>
<td>104</td>
</tr>
<tr>
<td>104</td>
<td>False</td>
<td>106</td>
<td>False</td>
<td>106</td>
</tr>
<tr>
<td>103</td>
<td>True</td>
<td>101</td>
<td>True</td>
<td>103</td>
</tr>
<tr>
<td>104</td>
<td>True</td>
<td>106</td>
<td>True</td>
<td>105</td>
</tr>
<tr>
<td>104</td>
<td>True</td>
<td>103</td>
<td>True</td>
<td>105</td>
</tr>
</tbody>
</table>

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 \( \delta \) 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 \( \delta \) value is between 0.01 and 0.001, but in Table 4.5, it is between 0.001 and 0.0001.
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.

<table>
<thead>
<tr>
<th></th>
<th>( \delta = 0.1 )</th>
<th>( \delta = 0.01 )</th>
<th>( \delta = 0.001 )</th>
<th>( \delta = 0.0001 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT</td>
<td>RES</td>
<td>RT</td>
<td>RES</td>
</tr>
<tr>
<td>Set 1</td>
<td>105 True</td>
<td>104 True</td>
<td>104 True</td>
<td>112 True</td>
</tr>
<tr>
<td>Set 2</td>
<td>102 True</td>
<td>107 True</td>
<td>107 True</td>
<td>109 True</td>
</tr>
<tr>
<td>Set 3</td>
<td>101 True</td>
<td>104 True</td>
<td>108 True</td>
<td>110 True</td>
</tr>
<tr>
<td>Set 4</td>
<td>109 True</td>
<td>104 True</td>
<td>108 True</td>
<td>131 True</td>
</tr>
<tr>
<td>Set 5</td>
<td>102 True</td>
<td>108 True</td>
<td>107 True</td>
<td>107 True</td>
</tr>
<tr>
<td>Set 6</td>
<td>104 True</td>
<td>102 True</td>
<td>109 True</td>
<td>107 True</td>
</tr>
<tr>
<td>Set 7</td>
<td>102 True</td>
<td>109 True</td>
<td>107 True</td>
<td>107 True</td>
</tr>
<tr>
<td>Set 8</td>
<td>104 True</td>
<td>105 True</td>
<td>109 True</td>
<td>107 True</td>
</tr>
<tr>
<td>Set 10</td>
<td>106 True</td>
<td>105 True</td>
<td>108 True</td>
<td>108 True</td>
</tr>
</tbody>
</table>

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 [GJ10], 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 OpenM. 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.
Experimental Results

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 3.3 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


Bibliography


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

*Proof.* The star set \( \Theta \) represents the polyhedron \( \mathcal{P} \) 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.** Given a star set \( \Theta = \langle c, V, P \rangle \), an affine mapping of the star \( \Theta \) with the linear mapping matrix \( W \) and offset vector \( b \) defined by \( \tilde{\Theta} = \{ y \mid y = Wx + b, x \in \Theta \} \) is another star such that
\[
\tilde{\Theta} = \langle \tilde{c}, \tilde{V}, P \rangle, \quad \tilde{c} = Wc + b, \quad \tilde{V} = \{ Wv_1, \ldots, Wv_m \}, \quad \tilde{P} \equiv P
\]

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

**Lemma A.0.3.** The worst-case complexity of the number of stars in the reachable set of an \( N \)-neurons FNN is \( \Theta(2^N) \).

*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 \( 2^{n_{L_1}} \cdot \ldots \cdot 2^{n_{L_k}} = 2^{n_{L_1} + \ldots + n_{L_k}} = 2^N \). \( \square \)

**Lemma A.0.4.** The worst-case complexity of the number of constraints of a star in the reachable set of an \( N \)-neurons FNN is \( \Theta(N) \).

*Proof.* The exactReLU sub-procedure produces for the given input set \( \Theta \) one or two more stars that have at most one more constraint. \( \square \)
The exactReLU sub-procedure produces for the given input set \( \Theta \) 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 \( \Theta(N) \).

**Theorem A.0.5.** Let \( F \) be an FNN, \( \Theta \) 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.** The worst-case complexity of the number of stars in the reachable set of an \( N \)-neurons FNN is \( \Theta(3^N) \).

**Proof.** The exactHTangent operation produces three more stars at most. Accordingly, in the worst-case scenario, the number of stars of one layer is \( 3^n \) where \( n \) 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 \( 3^n \times \cdots \times 3^n = 3^n \times \cdots \times 3^n = 3^N \) \( \square \)

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

**Proof.** The exactHSigmoid operation produces three more stars at most. Accordingly, in the worst-case scenario, the number of stars of one layer is \( 3^n \) where \( n \) 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 \( 3^n \times \cdots \times 3^n = 3^n \times \cdots \times 3^n = 3^N \) \( \square \)

**Theorem A.0.8.** Let \( F \) be an FNN, \( \Theta \) 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 \( C_\Theta = \bigcup_{i=1}^{k} \langle c, V, \bar{P}_i \rangle, \bar{P}_i \neq \emptyset \).

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

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.
<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT(s)</td>
<td>RES</td>
</tr>
<tr>
<td>$N_{2,1}$</td>
<td>15837.45</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,2}$</td>
<td>36112.55</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,3}$</td>
<td>18532.01</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,4}$</td>
<td>8254.17</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,5}$</td>
<td>56241.47</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,6}$</td>
<td>65508.57</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,7}$</td>
<td>53195.81</td>
<td>False</td>
</tr>
<tr>
<td>$N_{2,8}$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$N_{3,1}$</td>
<td>15559.97</td>
<td>False</td>
</tr>
<tr>
<td>$N_{3,2}$</td>
<td>12911.84</td>
<td>False</td>
</tr>
<tr>
<td>$N_{3,3}$</td>
<td>24675.76</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,4}$</td>
<td>7393.05</td>
<td>False</td>
</tr>
<tr>
<td>$N_{3,5}$</td>
<td>23852.32</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,6}$</td>
<td>70608.24</td>
<td>False</td>
</tr>
<tr>
<td>$N_{3,7}$</td>
<td>41256.50</td>
<td>False</td>
</tr>
<tr>
<td>$N_{3,8}$</td>
<td>37253.31</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,1}$</td>
<td>66720.59</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,2}$</td>
<td>85507.29</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,3}$</td>
<td>10627.68</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,4}$</td>
<td>12923.14</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,5}$</td>
<td>75982.74</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,6}$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$N_{4,7}$</td>
<td>107331.18</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,8}$</td>
<td>67596.46</td>
<td>False</td>
</tr>
<tr>
<td>$N_{4,9}$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$N_{5,1}$</td>
<td>30332.56</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,2}$</td>
<td>43636.56</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,3}$</td>
<td>10740.98</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,4}$</td>
<td>6221.00</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,5}$</td>
<td>31217.25</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,6}$</td>
<td>97829.27</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,7}$</td>
<td>57210.41</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,8}$</td>
<td>10123.47</td>
<td>False</td>
</tr>
<tr>
<td>$N_{5,9}$</td>
<td>78557.70</td>
<td>False</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT(s)</td>
<td>RES</td>
</tr>
<tr>
<td>$N_{1,1}$</td>
<td>1768.51</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,2}$</td>
<td>1647.32</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,3}$</td>
<td>442.65</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,4}$</td>
<td>224.24</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,5}$</td>
<td>246.37</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,6}$</td>
<td>65.95</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,1}$</td>
<td>485.40</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,2}$</td>
<td>178.63</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,3}$</td>
<td>316.67</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,4}$</td>
<td>20.16</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,5}$</td>
<td>111.27</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,6}$</td>
<td>1647.32</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,7}$</td>
<td>60.04</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,8}$</td>
<td>17.78</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,9}$</td>
<td>9.46</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,1}$</td>
<td>153.28</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,2}$</td>
<td>2018.12</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,3}$</td>
<td>390.10</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,4}$</td>
<td>76.08</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,5}$</td>
<td>44.62</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,6}$</td>
<td>99.32</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,7}$</td>
<td>4.20</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,8}$</td>
<td>33.03</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,9}$</td>
<td>42.36</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,1}$</td>
<td>50.28</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,2}$</td>
<td>627.65</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,3}$</td>
<td>976.11</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,4}$</td>
<td>34.30</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,5}$</td>
<td>9.23</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,6}$</td>
<td>107.72</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,7}$</td>
<td>51.25</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,8}$</td>
<td>35.98</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,9}$</td>
<td>36.69</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,1}$</td>
<td>328.52</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,2}$</td>
<td>61.58</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,3}$</td>
<td>72.33</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,4}$</td>
<td>33.32</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,5}$</td>
<td>44.61</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,6}$</td>
<td>63.87</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,7}$</td>
<td>4.93</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,8}$</td>
<td>134.31</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,9}$</td>
<td>4.04</td>
<td>True</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT(s)</td>
<td>RES</td>
</tr>
<tr>
<td>$N_{1,1}$</td>
<td>424.83</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,2}$</td>
<td>366.72</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,3}$</td>
<td>277.95</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,4}$</td>
<td>24.99</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,5}$</td>
<td>208.17</td>
<td>True</td>
</tr>
<tr>
<td>$N_{1,6}$</td>
<td>117.16</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,1}$</td>
<td>123.79</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,2}$</td>
<td>149.75</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,3}$</td>
<td>123.79</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,4}$</td>
<td>231.23</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,5}$</td>
<td>25.16</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,6}$</td>
<td>31.60</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,7}$</td>
<td>122.59</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,8}$</td>
<td>62.39</td>
<td>True</td>
</tr>
<tr>
<td>$N_{2,9}$</td>
<td>88.98</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,1}$</td>
<td>27.72</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,2}$</td>
<td>22.95</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,3}$</td>
<td>88.98</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,4}$</td>
<td>46.37</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,5}$</td>
<td>18.88</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,6}$</td>
<td>126.04</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,7}$</td>
<td>8.05</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,8}$</td>
<td>160.56</td>
<td>True</td>
</tr>
<tr>
<td>$N_{3,9}$</td>
<td>231.23</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,1}$</td>
<td>78.86</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,2}$</td>
<td>58.67</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,3}$</td>
<td>45.51</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,4}$</td>
<td>87.67</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,5}$</td>
<td>6.78</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,6}$</td>
<td>79.35</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,7}$</td>
<td>139.12</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,8}$</td>
<td>166.59</td>
<td>True</td>
</tr>
<tr>
<td>$N_{4,9}$</td>
<td>117.59</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,1}$</td>
<td>42.58</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,2}$</td>
<td>40.08</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,3}$</td>
<td>52.80</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,4}$</td>
<td>27.96</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,5}$</td>
<td>6.08</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,6}$</td>
<td>24.29</td>
<td>True</td>
</tr>
<tr>
<td>$N_{5,7}$</td>
<td>34.99</td>
<td>True</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{RT}(s)$</td>
<td>$\text{RES}$</td>
</tr>
<tr>
<td>$N_{1,1}$</td>
<td>2933.99</td>
<td>True</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{RT}(s)$</td>
<td>$\text{RES}$</td>
</tr>
<tr>
<td>$N_{1,1}$</td>
<td>44026.93</td>
<td>True</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{RT}(s)$</td>
<td>$\text{RES}$</td>
</tr>
<tr>
<td>$N_{11}$</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table B.7: Verification results for property $P_7$ on 1 ACAS Xu network. $\text{RT}$ is the reachable set computation time, and $\text{VT}$ is the safety verification time, both in seconds. $\text{RES}$ is the safety verification result. $\text{OSS}$ 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.

<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{RT}(s)$</td>
<td>$\text{RES}$</td>
</tr>
<tr>
<td>$N_{2,9}$</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table B.8: Verification results for property $P_8$ on 1 ACAS Xu network. $\text{RT}$ is the reachable set computation time, and $\text{VT}$ is the safety verification time, both in seconds. $\text{RES}$ is the safety verification result. $\text{OSS}$ 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.
<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT(s)</td>
<td>RES</td>
</tr>
<tr>
<td>$N_{3,3}$</td>
<td>33727.84</td>
<td>False</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>$N_{x,y}$</th>
<th>Exact</th>
<th>Overapproximation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RT(s)</td>
<td>RES</td>
</tr>
<tr>
<td>$N_{4,5}$</td>
<td>3281.44</td>
<td>True</td>
</tr>
</tbody>
</table>

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.