TÉCNICO LISBOA

Neural Networks Robustness Verification using Reachability Tools

Lourenço Sequeira Cristóvão e Monteiro da Silva

Thesis to obtain the Master Science Degree in

Electrical and Computer Engineering

Supervisors: Professor Doutor Daniel de Matos Silvestre
Professora Doutoura Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee

Chairperson:	Professora Doutora Teresa Maria Canavarro Menéres Mendes de Almeida
Supervisor:	Professora Doutoura Rita Maria Mendes de Almeida Correia da Cunha
Member of the Committee:	Doutor Paulo André Nobre Rosa

November 2023

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Agradecimentos

Em primeiro lugar agradeço aos meus orientadores, o Prof. Daniel Silvestre e a Prof. Rita Cunha pelo constante apoio e disponibilidade ao longo desta caminhada que culminou no término do meu mestrado. Os seus profundos conhecimentos e visões permitiram o meu crescimento académico, tendo sido vitais para o trabalho desenvolvido ao longo deste último ano. Foi um privilégio trabalhar com eles.
Numa nota mais pessoal, gostaria de agradecer aos meus pais pelo incessante apoio emocional e por nunca terem deixado de acreditar em mim e na minha capacidade de ser melhor e aprender todos os dias. A coragem que demonstram permitiram-me ter a força necessária para manter a perseverança durante os momentos mais difíceis da minha vida (académica e pessoal). Uma palavra de agradecimento aos meus avós que ajudaram e contribuíram para a pessoa que hoje sou. Ainda aos meus irmãos por me permitirem ser mais feliz e que sempre me proporcionaram momentos felizes, em família.
Também uma nota de agradecimento a todos aqueles que conheci durante esta longa caminhada e com os quais partilhei momentos marcantes durante todos os projetos. Foram eles que permitiram relaxar nos momentos essenciais. Não esqueço também todas pessoas envolvidas na Jornadas de Engenharia Eletrotécnica, que para sempre ficarão na minha memória por todas as experiências (melhores e piores) vividas. Aos meus amigos e amigas da Margem Sul, por nunca deixarem-me de apoiar nas minhas decisões e por todos os momentos bem passados.
Como último agradecimento, ao Instituto Superior Técnico, escola à qual cheguei um miúdo ingénuo vindo do secundário e da qual saí um homem completamente diferente: preparado para o futuro, capaz de se manter resiliente nos momentos mais adversos da vida e de enfrentar problemas reais do quotidiano.

This work was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) through project FirePuma (https://doi.org/10.54499/PCIF/MPG/0156/2019), through Institute for Systems and Robotics (ISR), under Laboratory for Robotics and Engineering Systems (LARSyS) project UIDB/50009/2020, and through COPELABS, University Lusófona project UIDB/04111/2020.

Resumo

Garantir a robustez em controladores, tais como, Redes Neuronais (RNs) é crucial em sistemas críticos em áreas como a aeronáutica, missões espaciais ou sistemas ciberfísicos que podem ter consequências para os seres humanos (por exemplo, condução autónoma de automóveis). Atualmente, existem inúmeros métodos na literatura que apresentam algoritmos capazes de calcular conjuntos de forma exata e aproximada. No entanto, estes métodos não são completamente eficientes uma vez que podem-se tornar computacionalmente intratáveis ou então são limitados ao tipo existente de representação de conjuntos (por exemplo, polítopos ou elipsoides). O principal objetivo desta dissertação é desenvolver vários algoritmos usando um novo tipo de representação de conjuntos para verificar robustez de RNs, os quais podem ser usados para, por exemplo, validar a saída no que respeita a estabilidade de sistema em cadeia fechada. Para atingir este objetivo, este trabalho propõe o uso de Constrained Convex Generators (CCGs) permitindo calcular conjuntos sobre-aproximados. Além disso, um tipo de representação de conjuntos mais conservador, Constrained Zonotopes (CZs), também será usado para o mesmo propósito. Este trabalho também contribui para aprimorar métodos na literatura para determinar restrições lineares e não-lineares. As simulações apresentadas, ilustram vantagens e desvantagens deste novo tipo de representação de conjuntos em detrimento da mais conservadora, na qual a principal conclusão é a diferença de volumes considerando cada tipo de representação de conjuntos.

Keywords: Acessibilidade, Aprendizagem de Reforço, Conjuntos Convexos, Redes Neuronais, Robustez

Abstract

Guaranteeing security in controllers such as Neural Networks (NNs) is crucial in critical systems in the domains of aeronautics, space missions or cyber-physical systems that can harm humans like autonomous driving cars. Currently, various state-of-art methods present algorithms allowing to compute exact and approximate sets. However, these methods are not completely efficient since these can become computationally intractable or are limited to a type of set representation (e.g. polytopes and ellipsoids). The main goal of this dissertation is to develop several algorithms using a novel set representation to verify robustness of NNs, where they can be used to, for instance, validate the output with respect to stability of a closed-loop system. To achieve this goal, this work proposes the use of Constrained Convex Generators (CCGs), allowing to determine over-approximated reachable sets. In addition, a more conservative set representation, Constrained Zonotopes (CZs), will be also be used for the same purpose. This work also contributes with improving current methods for determining linear and non-linear constraints. In simulations, it is illustrated current advantages and disadvantages of the novel set representation in detriment of the conservative one, where the main conclusion to be taken is the difference of volumes considering each type of representation.

Keywords: Convex Sets, Neural Networks, Reachability, Reinforcement Learning, Robustness

Contents

List of Tables

List of Figures

List of Symbols

$\lvert a \rvert$	Absolute value of a.
$M[:, i]$	Column i of matrix M.
diag(a)	Diagonal matrix with all the entries in the main diagonal equal to a and implicit dimensions
$v[i]$	Entry i of vector v.
I_k	Identity matrix with dimension k.
$\mathbf{e}_i$	i-th canonical vector with mentioned dimension.
$M[i, :]$	Row i of matrix M.
$\lvert\lvert a \rvert\rvert_\infty$	ℓ_∞ norm of vector a.
$\mathbf{0}_{k \times k}$	Matrix of zeros with dimensions $k \times k$.
$\mathbb{R}$	Set of real numbers.
M^T	Transpose of matrix M.
$\mathcal{B}_2$	Unit ℓ_2 norm ball.
$\mathcal{B}_\infty$	Unit ℓ_∞ norm ball.

Acronyms

CAT Conditional Affine Transformation.

CCG Constrained Convex Generator.

CZ Constrained Zonotope.

DNN Deep Neural Network.

ESA European Space Agency.

LMI Linear Matrix Inequality.

MNIST Modified National Institute of Standards and Technology.

MPC Model Predictive Control.

NASA National Aeronautics and Space Administration.

NN Neural Network.

QC Quadratic Constraint.

ReLU Rectified Linear Unit.

SDP Semidefinite Programming.

SiLU Sigmoid Linear Unit.

Tanh Hyperbolic Tangent.

Chapter 1

Introduction

1.1 Motivation: Robustness of Neural Networks

Although Neural Networks (NNs) first appeared in 1943, only over the end of last century, there was an increasing interest in using NN for many applications, such as, decision-making, image, speech recognition, spacial missions and aeronautics. Also, due to technological evolution, these networks were used to improve performance and reduce computational costs for on-line implementation. Some of the reasons why NNs are used include learning and adapting to new data, handling complex and nonlinear relationships and processing large amounts of data quickly and efficiently.

Additionally, due to the usage of NNs in more important and impactful tasks, precision and low error of outputs is required because of costs and effects related to these tasks. In real-world scenarios, the data provided may be noisy which is something the network must to able to cope with without providing wrong results. A less robust NN is more vulnerable to adversarial attacks, where there is an clear intention to disturb the functioning of the network and provoke wrong outputs. This sturdiness is very important to ensure reliability and security in many applications.

One of the main areas of usage of controllers with NNs are spacial missions. To better showcase how impactful verifying a NN can be, two examples are presented, where two different cases of spacial missions are considered.

In [3], NNs are used both to improve system performance and reducing costs as well as monitor in near real time the environment inside the station. For the first case, it was undertaken a review by on of the divisions of National Aeronautics and Space Administration (NASA) Glenn Research Center in order to assess how some of their systems could be integrated into one unit and cut the time required to perform the daily aircraft analysis. In the second case, a system using NNs was developed to monitor both operating machinery (fans, coils) and crew activities. Also, in 2022, European Space Agency (ESA) funded several projects to explore the use of NNs in making satellites ”more reactive, agile and autonomous”. Some of these projects presented how these satellites can be more effective in managing disasters from space as well as support more sustainable explorations of the Moon.

From these examples it is possible to deduce possible outcomes, in a worst-case scenario, when the networks considered are not robust. For the first example, if the monitoring system is susceptible to input noise, the wrong evaluation of the operating machines can lead, for example, to a later detection of worn out parts of the space station which would have impact in the lives of the crew as well as humans. For the second example, some of these projects have the objective of preventing natural disasters and hence, robustness also is important.

As it will be presented in Chapter 3, current studies reveal the advantages of verifying robustness of

NNs with the computation of reachable sets (exact and approximate).

1.2 Dissertation Overview

The thesis is divided in two distinct parts. First, topics related to this work are presented as well as the problem formulation, followed by a review of the state-of-art. Then, the proposed solution is showcased with all the steps detailed to obtain the algorithms. Finally, simulations results are presented, conclusions are drawn and possible directions that may emerge are proposed.
In more depth, the thesis structure is the following:

- In Chapter 2, the problem is formulated and topics related to this work are presented;
- In Chapter 3, it is presented current methods and algorithms for reachability analysis for Deep Neural Network (DNN), using ReLU and other activation functions;
- In Chapter 4, the reasoning behind current algorithms using ReLU is extended to other known activation functions, such as, logistic/sigmoid, Tanh, SiLU. Also, a novel set presentation, CCG, is used and algorithms are derived;
- In Chapter 5, simulation results are presented using these algorithms on different applications: a more theoretical case, where a Model Predictive Control (MPC) is used and more practical one where real-life examples are used, more specifically, the Modified National Institute of Standards and Technology (MNIST) database;
- In Chapter 6, it is summarized the main conclusions of this work and is proposed possible directions that emerge from this thesis.

1.3 Contributions

It is now highlighted the main contributions in this work:

- Presentation of the reasoning behind the algorithm that allows to over-approximate an output reachable set, when passing through a ReLU activation function;
- Derivation of algorithms to determine over-approximated output reachable sets for different activation functions using CZs and CCGs;
- Development of an alternative methodology to determine constraints to obtain sets;
- Improvement of current algorithms to determine the minimum volume enclosing ellipsoid;

Chapter 2

Problem Statement

In this chapter, the main topics related to the work are presented and ends by formulating the problem.

2.1 Concept of a Neural Network

A NN is inspired by the structure and functioning of the human brain. This is composed by many neurons that are connected between each other, transmitting information. Similarly, a NN is composed by many interconnected processing nodes, called ”neurons”, which are organized into layers.
The concept of a NN corresponds to a function that maps inputs to outputs through a sequence of layers. At each layer, the input goes through a linear transformation, which is parameterized by weights and biases (which are learnt during learning phase), followed by an nonlinear transformation before going to the next layer. This nonlinear transformation is known as an activation function. The most common one is ReLU, which converts all negative values to zero. Other examples of activation functions are sigmoid $(\sigma(x) = \frac{1}{1+e^{-x}})$ and Tanh. The choice behind these functions depends on the purpose of use for the network.
Figure 2.1 illustrates a feedforward neural network. $f_i : \mathbb{R}^{k_{i-1}} \rightarrow \mathbb{R}^{k_i}$ (k_i is the dimension of the hidden variable z_i) is a function that maps the input to an output for layer *i*. The variable $\hat{z}_i$ corresponds to the linear mapping defined by $W_i z_{i-1} + b_i$ and is the node value before activation. Variable $z_i = f_i(z_{i-1}) = \sigma_i(W_i z_{i-1} + b_i)$ is denoted as hidden variable, where σ_i is the activation function for layer i. To note that $z_0 = x$, that is for $i = 1$, which is the first layer of the NN, $z = x$.

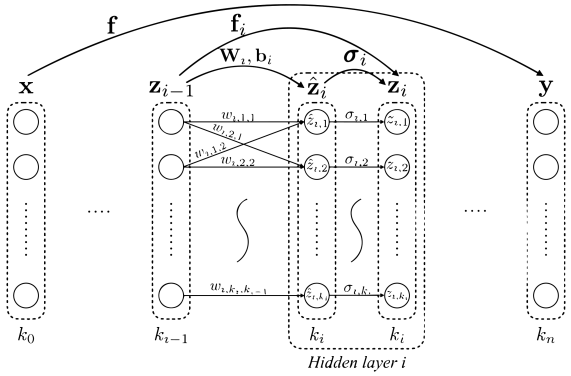


Figure 2.1: Illustration of a feedforward neural network [1].

Also, W_i and b_i are the weights and biases for layer i (respectively) of the NN learnt during learning phase.

2.2 Reachability Analysis

Another topic to be introduced is reachability since a series of methods to verify robustness of a NN with reachability analysis will be presented in the following chapter. These methods intend to verify whether input-output relationship hold, where the input is constrained to a set $\mathcal{X}$ and a set $\mathcal{Y}$ in the output. Solving this verification problem corresponds to checking whether the following implication holds:

$$x \in \mathcal{X} \Rightarrow y = f(x) \in \mathcal{Y}. \tag{2.1}$$

This implication is equivalent to verifying if the NN is able to process input that lies within a certain range. This means that depending on the volume of the input set, if the output of a certain $x \in \mathcal{X}$ lies inside set $\mathcal{Y}$, then the implication is verified. For instance, considering two sets with two different volumes, if (2.1) is verified for the smaller set and not verified for the larger set (in terms of volume), then it is possible to conclude that the NN is more susceptible to adversarial attacks and input uncertainties. The exact rationale is applied when the condition is verified for both sets, resulting in a more robust NN.
As [1] presents, there are many type of analysis to verify robustness of NN, such as, reachability, optimization and search. However, the focus of this work is using reachability tools.

2.3 Background on Constrained Zonotopes (CZs) and Constrained Convex Generators (CCGs)

During Chapter 4, two types of set representations are used: CZs and CCGs and hence, it is important to introduce both the definition of each set representation and the operations available.

Definition 1 *Per [4], a set $\mathcal{Z} \subset \mathbb{R}^n$ is a CZ if there exists $(G, c, A, b) \in \mathbb{R}^{n \times n_g} \times \mathbb{R}^n \times \mathbb{R}^{n_c \times n_g} \times \mathbb{R}^{n_c}$ such that:*

$$\mathcal{Z} = \{G\xi + c : ||\xi||_\infty \leq 1, A\xi = b\}. \tag{2.2}$$

Considering three different CZs:

- $Z = (G_z, c_z, A_z, b_z) \subset \mathbb{R}^n$,
- $W = (G_w, c_w, A_w, b_w) \subset \mathbb{R}^n$,
- $Y = (G_y, c_y, A_y, b_y) \subset \mathbb{R}^m$,

and a matrix $R \in \mathbb{R}^{m \times n}$ and a vector $t \in \mathbb{R}^m$, it is introduced the main required set operations as:

$$RZ + t = (RG_z, Rc_z + t, A_z, b_z), \tag{2.3}$$

$$Z \oplus W = \left(\begin{bmatrix} G_z & G_w \end{bmatrix}, c_z + c_w, \begin{bmatrix} A_z & 0 \\ 0 & A_w \end{bmatrix}, \begin{bmatrix} b_z \\ b_w \end{bmatrix} \right), \tag{2.4}$$

$$Z \cap_R Y = \left(\begin{bmatrix} G_z & 0 \end{bmatrix}, c_z, \begin{bmatrix} A_z & 0 \\ 0 & A_y \\ RG_z & -G_y \end{bmatrix}, \begin{bmatrix} b_z \\ b_y \\ c_y - Rc_z \end{bmatrix} \right). \tag{2.5}$$

Definition 2 ([5]) *A CCG, $\mathcal{Z} \subset \mathbb{R}^n$, is defined by the tuple $(G, c, A, b, \mathfrak{C})$ with $G \in \mathbb{R}^{n \times n_g}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n_c \times n_g}$, $b \in \mathbb{R}^{n_c}$ and $\mathfrak{C} \coloneqq \{C_1, C_2, \cdots, C_{n_p}\}$ such that:*

$$\mathcal{Z} = \{G\xi + c : A\xi = b, \xi \in C_1 \times \cdots \times C_{n_p}\}. \tag{2.6}$$

Considering three different CCGs:

- $Z = (G_z, c_z, A_z, b_z, \mathfrak{C}_z) \subset \mathbb{R}^n$,
- $W = (G_w, c_w, A_w, b_w, \mathfrak{C}_w) \subset \mathbb{R}^n$,
- $Y = (G_y, c_y, A_y, b_y, \mathfrak{C}_y) \subset \mathbb{R}^m$,

and a matrix $R \in \mathbb{R}^{m \times n}$ and a vector $t \in \mathbb{R}^m$, it is introduced the main required set operations as:

$$RZ + t = (RG_z, Rc_z + t, A_z, b_z, \mathfrak{C}_z), \tag{2.7}$$

$$Z \oplus W = \left(\begin{bmatrix} G_z & G_w \end{bmatrix}, c_z + c_w, \begin{bmatrix} A_z & 0 \\ 0 & A_w \end{bmatrix}, \begin{bmatrix} b_z \\ b_w \end{bmatrix}, \{\mathfrak{C}_z, \mathfrak{C}_w\} \right), \tag{2.8}$$

$$Z \cap_R Y = \left(\begin{bmatrix} G_z & 0 \end{bmatrix}, c_z, \begin{bmatrix} A_z & 0 \\ 0 & A_y \\ RG_z & -G_y \end{bmatrix}, \begin{bmatrix} b_z \\ b_y \\ c_y - Rc_z \end{bmatrix}, \{\mathfrak{C}_z, \mathfrak{C}_y\} \right). \tag{2.9}$$

Here, $\oplus$ represents the Minkowski sum of two sets and $\cap_R$ the intersection after applying matrix R to the first set.

2.4 Neural Network Output Reachability Problem

The robustness of NNs and their associated usage in consequential domains are the issue to be addressed throughout this work. As it is discussed in Chapter 3, many studies show that reachability analysis of NN can evaluate how robust a NN.
The main goal of this work is to compute $\mathcal{Y}$ such that, where f represents the feedforward neural network:

$$x \in \mathcal{X} \Rightarrow y = f(x) \in \mathcal{Y}, \tag{2.10}$$

with $\mathcal{X}$ being the input set and $\mathcal{Y}$ the goal set, which should not intersect with an avoiding set $\mathcal{A}$. Previously to that, a NN is trained on a dataset to verify robustness. According to Figure 2.1, for the input layer, $z_0 = x$ i.e. for $i = 1$. For the remaining layers ($i = 1, \cdots, N$, where N is the number of layers of the network), function f is defined by:

$$\hat{z}_i = W_i z_{i-1} + b_i, \tag{2.11}$$

$$z_i = \sigma_i(\hat{z}_i) = \sigma_i(W_i z_{i-1} + b_i), \tag{2.12}$$

where $\hat{z}_i$ and z_i are the values before and after the activation, respectively. Also, σ_i is the activation function for layer i and W_i and b_i are respectively the weights and biases for layer i to be computed at learning phase of the NN.

Chapter 3

Related Work

In this chapter, existing methods for reachability analysis for DNNs will be analysed and discussed. Results will also be presented and conclusions will be drawn. A detailed review of the latest article will frame this work in the state-of-the-art. After that, an approach based on semidefinite programming is presented and described. Experiments and conclusions will also be presented.

A typical method for validating a NN entails training it with a large dataset as input and determining whether the NN output matches the desired outputs. However, this technique cannot verify every potential input because the input space is infinite in cardinality.

The first article to be discussed ([1]) addresses methods that are capable of inferring many properties about DNNs, being reachability the main category to analyse. This article only reviews algorithms that are sound, meaning that if it returns an answer then this answer is certainly true. Some of these algorithms are also complete, which means that they can return wrong answers. Additionally, some algorithms sacrifice completeness in favor of computational effectiveness.

The methods that will be analysed and discussed will be ExactReach [6], Ai2 [7], MaxSens [8], ReluVal [9] and FastLin [10]. These methods compute the reachable set using layer-by-layer analysis, which means that for each layer, the input set is passed through the linear mapping. Then it goes through the nonlinear mapping defined by the activation function (in this case, ReLU is considered). This procedure is repeated until the output layer is reached, which will produce a reachable set $\mathcal{R}$. This process is illustrated in Figure 3.1.

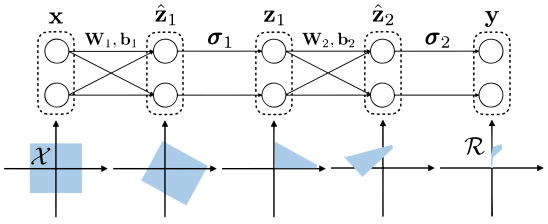


Figure 3.1: Illustration of reachability methods [1].

Throughout this process, there is one step which is non-trivial, the mapping $\hat{z}_i \mapsto \sigma_i(\hat{z}_i)$, where σ_i is the activation function and $\hat{z}_i = W_i z_{i-1} + b_i$ (which is the linear mapping) with *i* being the layer index. Different approaches to overcome this issue will be presented next.

3.1 ExactReach

The first method presented is ExactReach, which computes the exact reachable set for NN with linear or ReLU activations, where the sets (both input and output) are H-Polytopes. It also keeps track of all reachable sets. For ReLU functions, if the input set is a union of polytopes, then the output set will also be a union of polytopes.
The input set of layer *i* is a list of H-Polytopes, where each H-Polytope is parameterized by $C \in \mathbb{R}^{k \times k_i}$ and $d \in \mathbb{R}^k$, being *k* the number of constraints. Each Polytope defines a set:

$$\Phi = \{z_{i-1} : Cz_{i-1} \leq d\}. \tag{3.1}$$

After the linear mapping, the set before activation is given by:

$$\hat{\Phi} = \{\hat{z}_i : \hat{C}\hat{z} \leq \hat{d}\}, \tag{3.2}$$

where $\hat{C} \in \mathbb{R}^{k \times k_i}$, $\hat{d} \in \mathbb{R}^k$. Here the number of constraints may differ from the ones in (3.1). This set can be separated into several non-intersecting subsets, according to different activations patterns. Therefore, each subset of $\hat{\Phi}$ corresponds to the *h*th activation pattern, which is denoted by:

$$\hat{\Phi}_h = \{\hat{z}_i : P_h\hat{z}_i \geq 0, (I_h - P_h)\hat{z}_i \leq 0, \hat{C}\hat{z} \leq \hat{d}\}, \tag{3.3}$$

where $P_h \in \mathbb{R}^{k_i \times k_i}$ is a diagonal matrix, whose entries are the entries of the binary vector $\delta_i \in \{0,1\}^{k_i}$ (represents the activation status for z_i) and an integer $h \in \{0, 1, \cdots, 2^{k_i} - 1\}$. For each $z_i \in \hat{\Phi}_h$, the after activation nodes satisfy $z_i = P_h\hat{z}_i$ and, therefore, the reachable set $\mathcal{O}_h$ is a linear transformation defined by:

$$\mathcal{O}_h = P_h \circ \hat{\Phi}_h. \tag{3.4}$$

The output reachable set for Φ is the union of each $\mathcal{O}_h$:

$$\mathcal{O} = \bigcup_{h=0}^{2^{k_i}-1} \mathcal{O}_h. \tag{3.5}$$

As stated initially, the reachable sets are exact, meaning that there is not any approximation, i.e., for any point $z_i \in \mathcal{O}$, there is a point $z_{i-1} \in \Phi$ such that $z_i = f_i(z_{i-1})$. This method can be inefficient due to the amount of sets it has to keep track, which amounts to 2^{k_i} per layer and input set. As a result, the number of polytopes will increase exponentially with depth, making a larger NN ineffective.
The remaining methods here described will approximate the reachable sets and the last one approximates the network.

3.2 Ai2

Ai2 estimates the reachable set, denoted by $\tilde{R}(\chi, f)$, such that $R(\chi, f) \subseteq \tilde{R}(\chi, f)$, where χ is the input set of a NN and a function f represents the whole network, as discussed in Chapter 1.
Ai2 uses an abstract domain to approximate the reachable set at each layer, which is represented by a set of logical formulas. In [11], it is stated that these abstract domains combines floating point polyhedra and intervals with custom abstract transformers for affine transforms, ReLU, sigmoid, Tanh and maxpool functions. This domain associates two constraints: an upper polyhedral and lower polyhedral, which reduces the number of constraints correlated to a polyhedra.

Ai2 is valid for piecewise linear activation functions, which can be described as one Conditional Affine Transformation (CAT) that consists of a set of linear conditions and a collection of affine mappings. In order to propagate an abstract domain through a CAT, Ai2 uses two operations: meet and join. The meet operation divides an abstract domain into several subdomains that correspond to various CAT criteria. Due to the limitations of the abstract domain, these subdomains could be over-approximated and overlap each other. One instance of the abstract domain is used in the join operation to cover and approximate all reachable sets. Since it does not maintain track of all sets, this join operation differs from the one in 3.1.
An input set Φ at layer *i* goes through the linear map defined by W_i and b_i. As in 3.1, $\hat{\Phi}$ denotes the set after this linear mapping. The output set is defined by (3.3), where $\mathcal{O}_h$ is given by (3.4) and $\hat{\Phi}_h$ can be written as:

$$\hat{\Phi}_h = \hat{\Phi} \wedge \{\hat{z}_i : (I_h - 2P_h)\hat{z}_i \leq 0\}. \tag{3.6}$$

In this method, given one input polytope, the output reachable set will also be one polytope. This is a big difference relatively to ExactReach, where the number of sets grows exponentially and, hence, Ai2 is more scalable. However, this over-approximation of sets have the effect of incompleteness for this method.

3.3 MaxSens

MaxSens works for networks with monotone activation functions (either non-increasing or non-decreasing) and low-dimensional input and output spaces. It uses interval arithmetic to compute bounds for each node. The reasoning behind MaxSens is to grid the input space and compute the reachable set for each grid.
If the input set is a hyperrectangle, then the output set is any abstract polytope. The input set at layer *i* is:

$$\Phi = \{z_{i-1} : |z_{i-1} - c_{i-1}| \leq r_{i-1}\}, \tag{3.7}$$

where $c_{i-1} \in \mathbb{R}^{k_{i-1}}$ is the center of the hyperrectangle and $r_{i-1} \in \mathbb{R}^{k_{i-1}}$ is the radius of the hyperrectangle. The output reachable set is a hyperrectangle too:

$$\mathcal{O} = \{z_i : |z_i - c_i| \leq r_i\}, \tag{3.8}$$

where $c_i, r_i \in \mathbb{R}^{k_i}$.
In [1], it is defined node-wise values for each layer *i* and node j:

$$\beta_j = \sigma_{i,j}(w_{i,j}c_{i-1} + b_{i,j}), \tag{3.9}$$

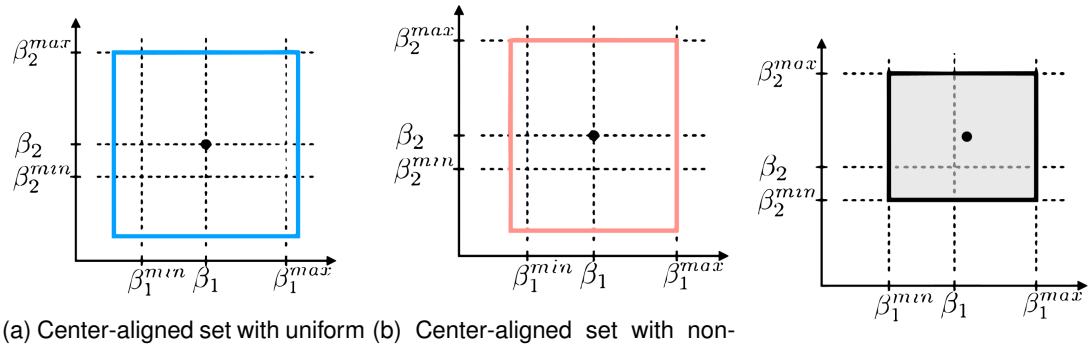
$$\beta_j^{max} = \sigma_{i,j}(w_{i,j}c_{i-1} + |w_{i,j}|r_{i-1}b_{i,j}), \tag{3.10}$$

$$\beta_j^{min} = \sigma_{i,j}(w_{i,j}c_{i-1} - |w_{i,j}|r_{i-1}b_{i,j}). \tag{3.11}$$

Due to the monotonicity of $\sigma_{i,j}$, $z_{i,j}$ can be written as:

$$z_{i,j} = \sigma_{i,j}(w_{i,j}z_{i-1} + b_{i,j}) \in [\beta_j^{min}, \beta_j^{max}], \forall z_{i-1} \in \Phi. \tag{3.12}$$

Figure 3.2 illustrates different approaches to approximate the output reachable set.
MaxSens differs from other methods since the number of geometric objects does not grow with the depth, since this only depends on the partition on the initial set. However, the error of over-approximation will increase with the depth.



(a) Center-aligned set with uniform radius

(b) Center-aligned set with non-uniform radius

(c) Tight set

Figure 3.2: Illustrations of different approaches to approximate output reachable sets [1].

3.4 ReluVal

ReluVal uses interval arithmetic to compute bounds for each node and keeps track of dependencies among nodes using symbolic representations, meaning that it can provide tighter bounds. Also, it uses iterative interval refinement for the search and takes a hyperrectangle as an input set.
Given an input x, [1] creates an extended version of it denoted by $x^e := [x, 1]$. Therefore, a symbolic interval at layer *i* is defined as:

$$z_i \in [L_i x^e, U_i x^e], \text{for} \in [x_0 - r, x_0 + r], \tag{3.13}$$

where $L_i, U_i \in \mathbb{R}^{k_i \times (k_0+1)}$ are coefficients in the symbolic interval.
Let ax_e be a symbolic representation, where $a \in \mathbb{R}^{k_0+1}$ and $\underline{h}$, $\overline{h} : \mathbb{R}^{k_0+1} \to \mathbb{R}$ be a function that maps the symbolic representation to its lower and upper bound, such that:

$$\underline{h}(a) := a[x_0, 1] - |a|[r, 0], \tag{3.14}$$

$$\overline{h}(a) := a[x_0, 1] + |a|[r, 0]. \tag{3.15}$$

As in previous methods, these symbolic intervals have to propagate through layers. Firstly, these are propagated through the linear mapping defined by W_i and b_i. For $i \in \{2, \cdots, n\}$, $\hat{L}_i, \hat{U}_i$ can be written as:

$$\hat{L}_i = [W_i]_+ L_{i-1} + [W_i]_- U_{i-1} + [\mathbf{0}\ b_i], \tag{3.16}$$

$$\hat{U}_i = [W_i]_+ U_{i-1} + [W_i]_- L_{i-1} + [\mathbf{0}\ b_i]. \tag{3.17}$$

For the first layer, $\hat{L}_1, \hat{U}_1$ are defined as $\hat{L}_1 = \hat{U}_1 = [W_1\ b_1]$. After this propagation, these intervals have to go through ReLU activation function, where each node *j* has three possibilities: always active ($j \in \Gamma_i^+$), never active ($j \in \Gamma_i^-$) and undetermined ($j \in \Gamma_i$), for layer *i*. Therefore, these can be expressed as:

$$\Gamma_i^+ = \{j : \underline{h}(\hat{l}_{i,j}) \geq 0\}, \tag{3.18}$$

$$\Gamma_i^- = \{j : \overline{h}(\hat{l}_{i,j}) \leq 0\}, \tag{3.19}$$

$$\Gamma_i = \{j : j \notin \Gamma_i^+ \cup \Gamma_i^-\}. \tag{3.20}$$

The symbolic interval for node *j* can be computed as:

$$j \in \Gamma_i^+ \Rightarrow l_{i,j} = \hat{l}_{i,j}, u_{i,j} = \hat{u}_{i,j}, \tag{3.21}$$

$$j \in \Gamma_i^- \Rightarrow l_{i,j} = u_{i,j} = 0, \tag{3.22}$$

$$j \in \Gamma_i \Rightarrow l_{i,j} = 0, u_{i,j} = \begin{cases} \hat{u}_{i,j} & \underline{h}(\hat{u}_{i,j}) \geq 0 \\ [\mathbf{0}\ \overline{h}(\hat{u}_{i,j})] & \underline{h}(\hat{u}_{i,j}) < 0 \end{cases} \tag{3.23}$$

After propagating these symbolic intervals, it is possible to compute the output reachable set which is a hyperrectangle. The expression for the output set is given by:

$$\tilde{R} = \{y : y_j \in [\underline{h}(l_{n,j}), \overline{h}(u_{n,j})], \forall j = 1, \cdots, k_n\} \tag{3.24}$$

Given the reachable set computed, it is compared to the output set Y to check whether the reachable set is included in the output set, which is the desired outcome. For the case where any conclusion can be drawn from this relationship (i.e. this relationship is neither true or violated), ReluVal performs iterative refinement in order to minimize over-approximation in $\tilde{R}$, though symbolic interval provides tighter bounds when compared to interval arithmetic.
In addition, points in the input interval are returned as counter examples if the outputs of those points do not belong in Y. Figure 3.1 illustrates this iterative procedure in ReluVal.

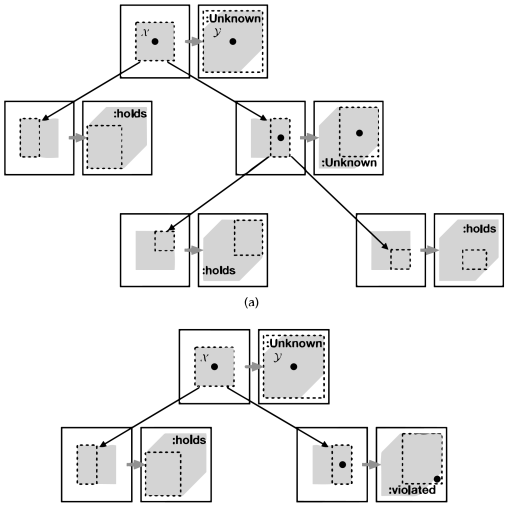


Figure 3.3: Illustration of procedure for ReluVal and Interval refinement [1].

3.5 FastLin

The last method here addressed is FastLin. This method computes the certified lower bound of the maximum allowable disturbance, based on a linear approximation of the network and only works for ReLU activation functions. It takes a hyperrectangle as input set and a polytope or the complement of a polytope as output set and combines reachability analysis with binary search to estimate bounds.
Given the bounds from interval arithmetic, l_i and u_i for $i \leq k$, the bounds $\hat{l}_{k+1}$ and $\hat{u}_{k+1}$ can be computed by optimizing the node values. The bounds are initiated as $l_0 = x_0 - \epsilon\mathbf{1}$ and $u_0 = x_0 + \epsilon\mathbf{1}$ and considering l_i and u_i for $i \leq k$, l_{k+1} and u_{k+1} can be expressed as (3.25). This expression only considers one node per layer, where ϵ is the center of the hyperrectangle.

$$\hat{l}_{k+1} = \min_{||x-x_0|| \leq \epsilon} \hat{z}_{k+1}, \hat{u}_{k+1} = \max_{||x-x_0|| \leq \epsilon} \hat{z}_{k+1} \tag{3.25}$$

In [1], it is introduced the concept of dual variables, where υ_i and $\hat{\upsilon}_i = W_{i+1}\upsilon_{i+1}$ are the dual variables

for $\hat{z}_i$ and z_i (respectively) and γ_i represents the bias in the value function. These dual variables form a dual network, in which the direction is the opposite of the original network. This network satisfies:

$$\upsilon_i = D_i W_{i+1}^T \upsilon, \forall i \leq k, \tag{3.26}$$

where $D_i \in \mathbb{R}^{k_i \times k_i}$ is a diagonal matrix whose diagonal entries $d_{i,j,j}$ for all j satisfy:

$$d_{i,j,j} = \begin{cases} 1 & j \in \Gamma_i^+ \\ 0 & j \in \Gamma_i^- \\ \frac{\hat{u}_{i,j}}{\hat{u}_{i,j} - \hat{l}_{i,j}} & j \in \Gamma_i \end{cases}. \tag{3.27}$$

The upper bound of $\hat{z}_{k+1}$ is:

$$\hat{u}_{k+1} \coloneqq \max_{||x - x_0||_p \leq \epsilon} \hat{\upsilon}_0^T x + \mu^+ = \hat{\upsilon}_0^T x_0 + \epsilon ||\hat{\upsilon}_0||_q + \mu^+, \tag{3.28}$$

where $\mu^+ \coloneqq \sum_{i=1}^n \upsilon_i^T b_i - \sum_{i=1}^{n-1} \sum_{j \in \Gamma_i} \hat{l}_{i,j} [\upsilon_{i,j}]_+$, *q* is a dual variable for *p*, where $p = \infty$ and $q = 1$.

The lower bound of $\hat{z}_{k+1}$ is:

$$\hat{l}_{k+1} \coloneqq \max_{||x - x_0||_p \leq \epsilon} \hat{\upsilon}_0^T x + \mu^- = \hat{\upsilon}_0^T x_0 - \epsilon ||\hat{\upsilon}_0||_q + \mu^-, \tag{3.29}$$

where $\mu^- = \sum_{i=1}^n \upsilon_i^T b_i - \sum_{i=1}^{n-1} \sum_{j \in \Gamma_i} \hat{l}_{i,j} [\upsilon_{i,j}]_-$.

This process should be performed iteratively from $k = 0$ to $k = n - 1$. Next, in [1] it is defined a matrix $V_i^{k+1} \in \mathbb{R}^{k_i \times K_{k+1}}$ to be a horizontal concatenation of $\upsilon_i^{k+1,j}$ for all $j \in \{1, ..., k_{k+1}\}$, that is:

$$V_i^{k+1} = [\upsilon_i^{k+1,1} \ \upsilon_i^{k+1,2} \ ... \ \upsilon_i^{k+1,k+1}]. \tag{3.30}$$

From (3.26) and (3.30), the network structure can be written as:

$$V_i^{k+1} = D_i W_{i+1}^T V_{i+1}^{k+1}, \forall i \leq k. \tag{3.31}$$

Regarding the binary search, [1] keeps track of lower and upper bounds of ϵ, denoted $\underline{\epsilon}$ and $\overline{\epsilon}$. The output reachable set for input set with radius ϵ is denoted $R(\epsilon)$. These bounds need to satisfy:

$$R(\underline{\epsilon}) \subseteq Y, R(\overline{\epsilon}) \nsubseteq Y. \tag{3.32}$$

At each search step, the reachable set $R(\epsilon)$ for $\epsilon \coloneqq \frac{\underline{\epsilon} + \overline{\epsilon}}{2}$ is computed. There are two possibilities: if $R(\epsilon) \subseteq Y$, then $\underline{\epsilon}$ is updated to be ϵ or if $R(\epsilon) \nsubseteq Y$, then $\overline{\epsilon}$ is updated to be ϵ. The search process terminate if either the maximum iteration is reached or the bounds are close enough to each other.

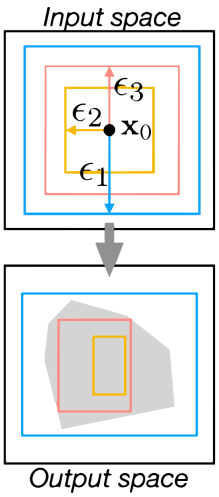


Figure 3.4: Illustration of binary search in FastLin [1].

It is presented a Table that synthesizes the discussion and analysis made for each method. This Table was made from one presented in [1]. In Table 3.1, HP denotes a halfspace-polytope represented by:

$$Cx \leq d, \tag{3.33}$$

where $C \in \mathbb{R}^{k \times k_0}$, $d \in \mathbb{R}^k$ with k being the number of inequality constraints defining the polytope. HR denotes hyperrectangle:

$$|x - c| \leq r, \tag{3.34}$$

where $c \in \mathbb{R}^{k_0}$ is center of the hyperrectangle and $r \in \mathbb{R}^{k_0}$ is the radius of the hyperrectangle. HS denotes a halfspace:

$$c^T c \leq d, \tag{3.35}$$

where $c \in \mathbb{R}^{k_0}$, $d \in \mathbb{R}$.

Method Name	Activation	Approach	Input\Output	Completeness
ExactReach	ReLU	Exact Reachability	HP/HP (bounded)	✓
Ai2	Piecewise Linear	Split and Join	HP/HP (bounded)	×
MaxSens	Any	Interval Arithmetic	HP/HP (bounded)	×
ReluVal	ReLU	Symbolic Interval	HR/HR	✓
FastLin	ReLU	Network Relaxation	HR/HS	×

Table 3.1: Comparison of methods for reachability analysis [1].

3.6 Reachability Analysis via Semidefinite Programming

Now, it is presented another method to verify NN with reachability using Semidefinite Programming (SDP), denoted as Reach-SDP. As [12] states, in SDP, the goal is to minimize a linear objective function constrained to a combination of symmetric matrices have to be positive semidefinite. Such constraints are nonlinear and nonsmooth but convex, therefore semidefinite programs are convex optimizations

problems.

To begin with, it is important to formulate the problem at hand. In [2], it starts by considering a discrete-time linear time-varying system:

$$P : x_{t+1} = A_t x_t + B_t u_t + c_t, \tag{3.36}$$

where $x_t \in \mathbb{R}^{n_x}$, $u_t \in \mathbb{R}^{n_u}$ are the state and control vectors and $c_t \in \mathbb{R}^{n_x}$ is a exogenous input. Also, it is assumed that the input is constrained to an interval, denoted as U_t, as written in expression (3.37).

$$u_t \in [\underline{u}_t, \overline{u}_t], t = 0, 1, \cdots . \tag{3.37}$$

It is also assumed that a state-feedback controller $\pi(x_t) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ is parameterized by a multi-layer feed-forward fully-connected NN. The map $x \mapsto \pi(x)$ is described by the equations:

$$x^0 = x, \tag{3.38}$$
$$x^{k+1} = \phi(W^k x^k + b^k), k = 0, \cdots, l-1, \tag{3.39}$$
$$\pi(x) = W^l x^l + b^l, \tag{3.40}$$

where $W^k \in \mathbb{R}^{n_{k+1} \times n_k}$, $b^k \in \mathbb{R}^{n_{k+1}}$ are the weight matrix and bias vector of the $(k+1)$-th layer, respectively. The nonlinear activation function $\phi(\cdot)$ is applied component-wise to the pre-activation vectors. In [2], it is used ReLU functions in derivations and implementation. To guarantee that the output of NN complies to the input restrictions, it is considered a projection operator in the loop, and therefore the control input is specified as:

$$u_t = \mathsf{Proj}_{U_t}(\pi(x_t)) = \mathsf{min}(\mathsf{max}(\pi(x_t), \underline{u}_t), \overline{u}_t). \tag{3.41}$$

From (3.36) and (3.41), is obtained a expression that denotes the closed-loop system, which is non-smooth nonlinear system:

$$x_{t+1} = f_\pi(x_t) \coloneqq A_t x_t + B_t \mathsf{Proj}_{U_t}(\pi(x_t)) + c_t. \tag{3.42}$$

In Figure 3.5, an illustration of closed-loop reachability with the initial set $\mathcal{X}_0$, *t*-step forward reachable set $R_t(\mathcal{X}_0)$ and its over-approximation $\bar{R}_t(\mathcal{X}_0)$ are presented.

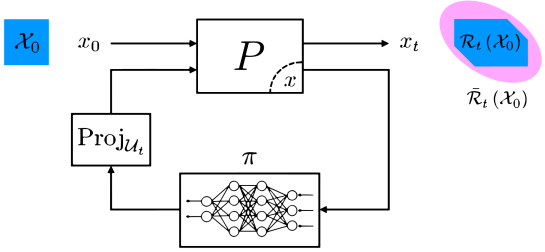


Figure 3.5: Illustration of the closed-Loop reachability [2].

The objective of [2] is to verify if given a goal set $G \subseteq \mathbb{R}^{n_x}$ and a sequence of sets $A_t \subseteq \mathbb{R}^{n_x}$, test if all initial states can reach the goal set in a finite time horizon, while avoiding A_t, $\forall t = 0, \cdots, N$. However, computing exact reachable sets for the system (3.42) is computationally intractable. Therefore, [2] finds outer approximations of the reachable set, denoted as $\bar{R}_t(\mathcal{X}_0)$, which have be as tight as possible to obtain useful certificates. Therefore, it is derived an abstracted system $\tilde{f}_\pi$ to replace the original closed-loop system, such that, this new system over-approximates the output of f_π (given by (3.42)). Then,

based on this system, it is computed the reachable sets.

To recursively estimate reachable sets, [2] proposes an approach that uses Quadratic Constraint (QC) abstraction, Reach-SDP. It is defined a new NN, where the projection operator is embedded in this network as two additional ReLU layers, defined as:

$$\begin{aligned} x_t^0 &= x_t \\ x_t^{k+1} &= \mathsf{max}(W^k x_t^k + b^k, 0), k = 0, ..., l-1 \\ x_t^{l+1} &= \mathsf{max}(W^l x_t^l + b^l - \underline{u}_t, 0) \\ x_t^{l+1} &= \mathsf{max}(-x_t^{l+1} + \overline{u}_t - \underline{u}_t, 0) \\ u_t &= -x_t^{l+2} + \overline{u}_t, \end{aligned} \tag{3.43}$$

and the closed-loop system:

$$x_{t+1} = A_t x_t + B_t u_t + c_t, x_t \in \bar{R}_t(\chi_0). \tag{3.44}$$

Firstly, it is abstracted the input set and the NN by QC. Then, is represented all QC with the same basis vector:

$$[\mathbf{x}_t^T \ 1] \coloneqq [x_t^{0^T} \ x_t^{1^T} \ \cdots \ x_t^{l+2^T} \ 1]^T \in \mathbb{R}^{n_x+n_n+2n_u+1}. \tag{3.45}$$

Finally, it is propagated the QC through the abstracted NN.

Starting with the input set, it is supposed that the set $\bar{R}_t(\chi_0)$ satisfies the QC defined by $\mathcal{P}$. Depending on the input set, the QCs satisfied are different. If the input set is a polytope $\chi = \{x \in \mathbb{R}^{n_x} | Hx \leq h\}$, with $H \in \mathbb{R}^{m \times n_x}$ and $h \in \mathbb{R}^m$, then $\mathcal{P}$ is defined as:

$$\mathcal{P} = \left\{ P \;\middle|\; P = \begin{bmatrix} H^T \Gamma H & -H^T \Gamma H \\ -h^T \Gamma h & h^T \Gamma h \end{bmatrix}, \Gamma \in \mathbb{S}^m, \Gamma \geq 0 \right\}, \tag{3.46}$$

where $\mathbb{S}^m$ represents the set of symmetric matrices of dimensions $m \times m$. Otherwise, if the input set is a ellipsoid $\chi = \{x \in \mathbb{R}^{n_x} | \ ||Ax + b||_2 \leq 1\}$ with $A \in \mathbb{S}^{n_x}$ and $b \in \mathbb{R}^{n_x}$, then $\mathcal{P}$ is defined as:

$$\mathcal{P} = \left\{ P \;\middle|\; P = \mu \begin{bmatrix} -A^T A & -A^T b \\ -b^T A & 1 - b^T b \end{bmatrix}, \mu \geq 0 \right\}. \tag{3.47}$$

Depending on the input set, for any $P \in \mathcal{P}$, the following expression is valid:

$$\begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix}^T M_{in}(P) \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix} \geq 0, \forall \mathbf{x}_t \in \bar{R}_t(\chi_0), \tag{3.48}$$

where $M_{in}(P) = E_{in}^T P E_{in}$ and the change-of-basis matrix is:

$$E_{in} = \begin{bmatrix} I_{n_x} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{3.49}$$

In the framework of [2], it is assumed that the candidate set $\bar{R}_{t+1}(\chi_0)$ that over-approximates $R_{t+1}(\chi_0)$ in represented by the intersection of finitely quadratic inequalities:

$$\bar{R}_{t+1}(\chi_0) = \bigcap_{i=1}^{m} \left\{ x_{t+1} \in \mathbb{R}^{n_x} \;\middle|\; \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix}^T S_i \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} \leq 0 \right\}, \tag{3.50}$$

where $S_i \in \mathbb{S}^{n_x+1}$ captures the shape and volume of the reachable set. Similarly with the input set,

depending on the set, S_i is different. For a polytopic reachable set, this can be written as:

$$S_i = \begin{bmatrix} 0 & H_i \\ H_i^T & -2h_i \end{bmatrix}, \tag{3.51}$$

where $H_i^T \in \mathbb{R}^{1\times\ n_x}$ is the *i*-th row of H and $h_i \in \mathbb{R}$ is the *i*-th entry of h. For an ellipsoidal reachable set, S_i is expressed by:

$$S_i = S = \begin{bmatrix} A^T A & A^T b \\ b^T A & b^T b - 1 \end{bmatrix}. \tag{3.52}$$

In [2] it is defined $M_{out}(S_i) = E_{out}^T S_i E_{out}$, where E_{out}, the change-of-basis matrix is:

$$E_{out} = \begin{bmatrix} A_t & 0 & \cdots & 0 & -B_t & B_t\overline{u}_t + c_t \\ 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}. \tag{3.53}$$

Also, the ReLU activation function, $\phi(x) = \max(0, x) : \mathbb{R}^n \to \mathbb{R}^n$, satisfies the QC defined by:

$$\mathcal{Q} = \left\{ Q \in \mathbb{S}^{2n+1} \;\middle|\; Q = \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{12}^T & Q_{22} & Q_{23} \\ Q_{13}^T & Q_{23}^T & Q_{33} \end{bmatrix} \right\}, \tag{3.54}$$

where the submatrices are:

$$\begin{aligned} &Q_{11} = 0_{n\times n},\ Q_{12} = \mathrm{diag}(\lambda) + \mathrm{T}, \\ &Q_{13} = -\upsilon,\ Q_{22} = -2(\mathrm{diag}(\lambda) + \mathrm{T}), \\ &Q_{23} = \upsilon + \eta, Q_{33} = 0. \end{aligned} \tag{3.55}$$

Here η,$\upsilon \geq 0$ and $\mathrm{T} \in \mathbb{S}_+^n$ is given by:

$$\mathrm{T} = \sum_{i=1}^{n} \lambda_i e_i e_i^T + \sum_{i=1}^{n-1} \sum_{j>i}^{n} \lambda_{ij}(e_i - e_j)(e_i - e_j)^T, \tag{3.56}$$

where $\lambda_{ij} \geq 0$ and $e_i \in \mathbb{R}^n$ has in the *i*-th entry and 0 everywhere else.

In [2], it is considered the projected neural network given by Equations (3.43) and $\mathcal{Q}$ defined as in (3.54). Then, for any $Q \in \mathcal{Q}$, the following inequality is valid:

$$\begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix}^T M_{mid}(Q) \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix} \geq 0, \forall \mathbf{x}_t \in \mathbb{R}^{n_x}, \tag{3.57}$$

where $M_{mid}(Q) = E_{mid}^T Q E_{mid}$ and the change-of-basis matrix is:

$$E_{mid} = \begin{bmatrix} E_1 & E_1 \\ E_2 & 0 \\ 0 & 1 \end{bmatrix}, \tag{3.58}$$

where

$$E_1 = \begin{bmatrix} W^0 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & W^l & 0 & 0 \\ 0 & \cdots & 0 & -I_{n_u} & 0 \end{bmatrix}, \; e_1 = \begin{bmatrix} b^0 \\ \vdots \\ b^{l-1} \\ b^l - \underline{u}_t \\ \overline{u}_t - \underline{u}_t \end{bmatrix}, \quad E_2 = \begin{bmatrix} 0 & I_{n_1} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & 0 \\ 0 & 0 & \cdots & I_{n_l} & 0 & 0 \\ 0 & 0 & \cdots & 0 & I_{n_u} & 0 \\ 0 & 0 & \cdots & 0 & 0 & I_{n_u} \end{bmatrix} \tag{3.59}$$

Finally, to determine the reachable sets an optimization problem is defined and hence, [2] starts by presenting a theorem. This considers the closed-loop system (3.44) and supposes the set $\bar{R}_t(\chi_0)$ satisfies QC defined by $\mathcal{P}$, $\mathcal{Q}$ defined as in (3.54) and $\bar{R}_{t+1}(\chi_0)$ as in (3.50). Considering the Linear Matrix Inequalitys (LMIs):

$$M_{in}(P) + M_{mid}(Q) + M_{out}(S) \preceq 0, i = 1, \cdots, m. \tag{3.60}$$

If there exists matrices $(P_i, Q_i) \in \mathcal{P} \times \mathcal{Q}$ that satisfy (3.60), then $R_{t+1}(\chi_0) \subseteq \bar{R}_{t+1}(\chi_0)$. This theorem provides a sufficient condition for over-approximating the reachable set and using this result, it is possible to formulate the optimization problem, depending on the input set. If the approximate reachable set is parameterized by a polytope, the problem is:

$$\begin{aligned} &\min_{P \in \mathcal{P}, Q \in \mathcal{Q}, h_i \in \mathbb{R}} h_i, \quad i = 1, \cdots, m \\ &\text{subject to} \quad (3.60). \end{aligned} \tag{3.61}$$

If the approximate reachable set is parameterized by a ellipsoid, the problem is:

$$\begin{aligned} &\min_{P \in \mathcal{P}, Q \in \mathcal{Q}, A \in \mathbb{S}^{n_x}, b \in \mathbb{R}^{n_x}} -\log\det(A) \\ &\text{subject to} \quad (3.60). \end{aligned} \tag{3.62}$$

3.7 Results and Conclusions

For the first methods presented (ExactReach, Ai2, MaxSens, ReluVal, FastLin) results are presented using distinct NN and two datasets: MNIST dataset and the *Aircraft Collision Avoidance System* (ACAS). In [1] it is also created a tiny toy network, denoted as *small nnet*, where it was analytically derived its transfer function. This *small nnet* has two hidden layers of two units each, where the objective is to evaluate simple properties, such as, upper and lower bounds of the input set. Apart from this last network, the remaining networks trained have different characteristics, regarding the size and amount of the hidden layers.

For the MNIST [13], properties that represent areas centered on points that correspond to a hand-written digit were built and the property that was tested corresponded to the correct classification of the image. Regarding the size and number of the hidden layers, different networks have distinct properties:

- *mnist1*, one hidden layer of size 25.
- *mnist2*, one hidden layer of size 100.

- *mnist3*, four hidden layer of size 25.

- *mnist4*, six hidden layer of size 50.

For all MNIST networks, the input size is 748 and the output size is 10.

For the ACAS dataset, the verified property was the situation where the intruder aircraft is far away from the ownship and the desired output is that the advisory is clear-of-conflict. The network has 5 input units, six hidden layers of 50 units and five output units.

The following table is a summarized version from the one presented in [1], where it was recorded the time taken for each algorithm to finish (in seconds), where the time-out threshold was 24 hours. Missing entries corresponds to timed-out experiments, an asterisk corresponds to an unknown result and two asterisk indicates that the algorithm incorrectly returned a violation. Group A supports hyperrectangle input sets, B supports H-polytope input sets and C supports hyperrectangle input sets and networks with one output node.

Group	Group A	Group A	Group A	Group B	Group C	Group B
Algorithm	ExactReach	Ai2	MaxSens	ReluVal		FastLin
small nnet	0.004070968	0.005170172	0.000201878	0.000031296	0.096123516	2.088460281
mnist1	-	-	16.06772067	0.646829741	0.104757137	0.477470278
mnist2	-	-	16.20866383	0.008979679	0.082225486	0.002369427
mnist3	-	-	16.1892307	0.008030365	0.019977288	0.007551539
mnist4	-	-	15.93491465	0.824760803*	0.021589774*	0.01564011
acas	-	-	0.0006839497	0.196084517*	0.000375971*	0.166508046**

Table 3.2: Time in seconds taken for each method to conclude the experiments [1].

Complete algorithms generally take longer to execute and, as a result, are better suited to verifying the properties of smaller networks. On the other hand, faster and better suited for verifying bigger networks, non-complete algorithms typically rely on over-approximations to lower their computing costs. Due to the computationally intensive process of converting sets from H-representation to V-representation, ExactReach and Ai2 timed out for the majority of the properties. For several attributes, ReluVal terminated relatively rapidly, yet it produced inconsistent results. The completeness of an algorithm and its scalability are typically trade-offs.

For the second algorithm presented, Reach-SDP, [2] uses two application examples to demonstrate their approach. It was used MPC controllers to generate training data, which were implemented in YALMIP [14] and the NN were trained with ReLU activation functions. In [2], it was used MATLAB and Mosek [15] to solve the semidefinite programs.

For the first example, it is considered a double integrator system:

$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} u_t, \tag{3.63}$$

which is used to generate training data to train a NN, as previously stated.

The initial set is a polytope $\chi_0 = [3, 4] \times [0.5, 1.5]$ and the goal set $\mathcal{G} = [-0.5, 0.5] \times [-0.5, 0.5]$. The results for this example are shown in Figure 3.6.

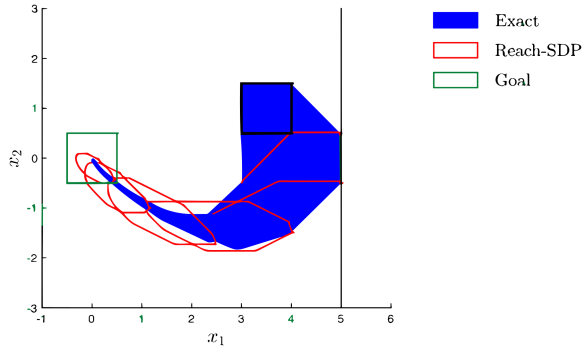


Figure 3.6: Illustration of the initial set (black), exact reachable sets (blue), approximate reachable sets (red) given by Reach-SDP and goal set (green) [2].

For the second example, it is considered a 6D quadrotor model:

$$\dot{x} = \begin{bmatrix} 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} x + \begin{bmatrix} 0_{3\times3} & & \\ g & 0 & 0 \\ 0 & -g & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tan(\theta) \\ \tan\phi \\ \tau \end{bmatrix} + \begin{bmatrix} 0_{5\times1} \\ -g \end{bmatrix}, \tag{3.64}$$

where *g* is the gravitational acceleration, the state vector x include positions and velocities of the quadrotor and the control vector u is a function of θ (pitch), ϕ (roll) and τ (thrust).
The initial set is a ellipsoid $\chi_0 = \mathcal{E}(q_0, Q_0)$, where $q_0 = [4.7\ \ 4.7\ \ 3\ \ 0.95\ \ 0\ \ 0]$ is the center and $Q_0 = \text{diag}(0.05^2, 0.05^2, 0.05^2, 0.01^2, 0.01^2, 0.01^2)$. The goal set $\mathcal{G} = [3.7, 4.1] \times [2.5, 3.5] \times [1.2, 2.6]$. The results for this example are shown in Figure 3.7.

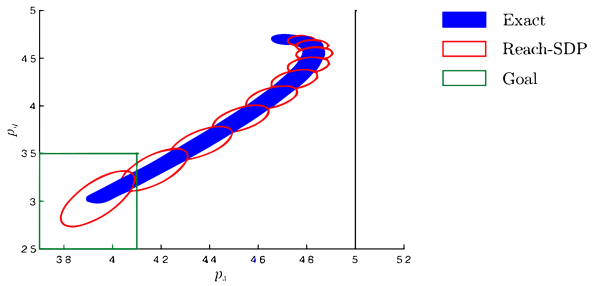


Figure 3.7: Illustration of the initial set (black), exact reachable sets (blue), approximate reachable sets (red) given by Reach-SDP and goal set (green) [2].

In both cases, it is possible to observe that both over-approximated reachable sets successfully verified the safety properties and approximated relatively well the exact reachable sets. Therefore, it is possible to conclude that this novel approach can reach good results. As [2] states, the next challenge is to extend this approach to nonlinear dynamics and approximate backward reachable sets.

Chapter 4

Proposed Solution

Having met the current methods for analysis of a NN, this chapter presents the implementation. This work focuses on developing algorithms, for different activation functions, based on different set representations to validate NNs. As an innovation, the sets will be computed using CCGs, which enables a less conservative representation of these sets where they can act as worst-case boundaries, as [16] states. As an additional contribution, the rationale used to obtain linear inequalities and an altered version of an existing function [17] are thoroughly detailed and explained as well as the approach used for the softmax function.

In the next two subsections, CZs and CCGs will be used and analysed to derive algorithms that allow to determine over-approximated sets. In both cases, each type of representation will be used for functions in which the best approximation is obtained. For instance for the ReLU activation function, using a CCG does not bring any benefit and the same reasoning can be applied to a CZ, when considering the softplus activation function.

Throughout this chapter, $\mathcal{Z}$ and $\hat{\mathcal{R}}$ will be denoted as input and output set at each layer, respectively. Additionally, $\mathcal{I}$ and $\hat{\mathcal{I}}$ represent the initial set considered for each variable and the set resulting from applying the needed operations for each variable (respectively) since the rationale presented is applied to each variable.

For the second pair of sets presented previously, as will be referred later these are parameterized by matrices, which for CZs are c, G, A, b ($\hat{c}, \hat{G}, \hat{A}, \hat{b}$ for the output set for each variable) and for CCGs are $c, G, A, b, \mathfrak{C}$ ($\hat{c}, \hat{G}, \hat{A}, \hat{b}, \hat{\mathfrak{C}}$ for the output set for each variable). To refer to a specific matrix belonging to these or other sets, it will be used $c_{\mathcal{I}}$ where $\mathcal{I}$ is the set considered.

As mentioned, for CZs and CCGs, bounds of each variable for each set need to be computed as shown in Algorithms 1 and 2 (third line). To determine these values, an optimization problem is solved using Mosek [15] or Gurobi [18] (as underlying solvers) and YALMIP [14] (as language to model optimization problems).

4.1 Activation Functions Overbound using Constrained Zonotopes

First, a brief presentation of the approach for computing exact sets will be done and advantages and disadvantages will be discussed, where [19] is analysed. Then, it is presented a set of activation functions for which an over-approximated set can be computed. For each function, an explanation is done and the algorithm that over-approximates the function is presented. Definition 2.2 presents the characterization for a CZ.

In [19], two algorithms are presented: an exact and an over-approximated. In both algorithms, a linear

mapping is first applied using layer weights and biases of a previously trained NN. Then, lower and upper bounds of each variable are computed. Henceforth, i denotes each variable with its lower (lb_i) and upper (up_i) bounds and the number of variables in each layer depend on the dimension of the space. Since the objective is to apply a ReLU, variables for which lower bounds are greater or equal to zero are irrelevant since for non-negative inputs, the output of a ReLU is the input after applied the linear mapping. Therefore, only variables with negative lower bounds will be considered in the next steps.
In Algorithm 1, the approach in each layer is the following:

- The linear mapping is applied to the input set;
- if the upper bound is negative or zero, then that variable is projected on its axis, since the output of the ReLU is zero;
- if the upper bound is positive, the CZ is intersected with two halfspaces separately ($\mathcal{H}^i_-$ and $\mathcal{H}^i_+$), resulting in two CZs ($\mathcal{I}_-$ and $\mathcal{I}_+$);
- Sets $\mathcal{I}_-$ and $\mathcal{I}_+$ are concatenated.

For each variable, the previous procedure is repeated according to the dimension of the space. After each iteration, the resulting set is concatenated with the previous ones and the procedure is repeated up to the last network layer. Also, $\mathcal{H}_-$ and $\mathcal{H}_+$ represent the mentioned halfspaces: $\{\mathbf{x} \in \mathbf{R}^n \mid \mathbf{e}_i^T \cdot x \leq 0\}$ and $\{\mathbf{x} \in \mathbf{R}^n \mid \mathbf{e}_i^T \cdot x \geq 0\}$, respectively. In this definition, $\mathbf{e}_i$ is the i-th canonical vector for $i = 1, \cdots, n$, where n is the dimension of the space of the input set.
As previously described, this algorithm allows to obtain a exact output set but it has problem: in the worst case scenario, the number of constrained zonotopes will grow exponentially with the number of layers and number of neurons of the NN, which would be computationally intractable for deeper neural networks. Therefore, this approach will be abandoned and the focus will be solely on the over-approximated case, the focus of this work.

4.1.1 Rectified Linear Unit function

The second algorithm over-approximates the output set with one constrained zonotope, presented in [19]. The reasoning behind is to find a polygon that covers the output of the ReLU function ($\mathcal{I}_1$ and $\mathcal{I}_2$). When considering $up_i = 4$ and $lb_i = -4$, the mentioned polygon is $\hat{\mathcal{I}}$, as presented in Figure 4.1.

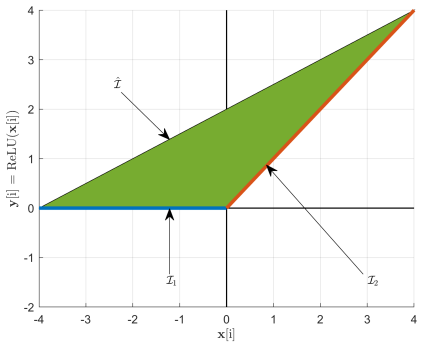
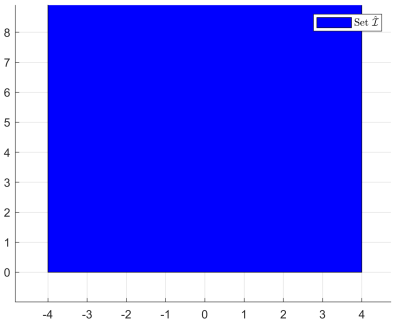


Figure 4.1: Convex relaxation of the ReLU activation function for over-approximation output analysis.

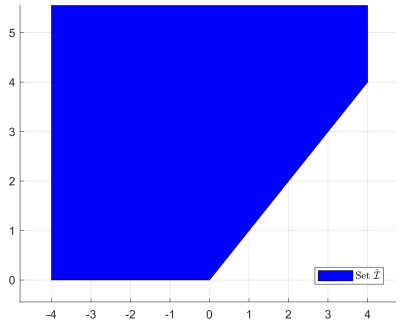
This polygon is a result from the intersection of three halfspaces: $\mathbf{y}[i] \geq 0$, $\mathbf{y}[i] - \mathbf{x}[i] \geq 0$ and $(up_i - lb_i)\mathbf{y}[i] - up_i(\mathbf{x}[i] - lb_i) \leq 0$, where $\mathbf{x}[i]$ has a range of $[lb_i, up_i]$.

Next, it will be detailed each step used to obtain the matrices presented in Algorithm 2 of [19], allowing to compute the over-approximated output reachable set. To note that the following rationale is applied to each variable, which after computing all the needed operations for all variables, permits finding $\mathcal{Z}$.

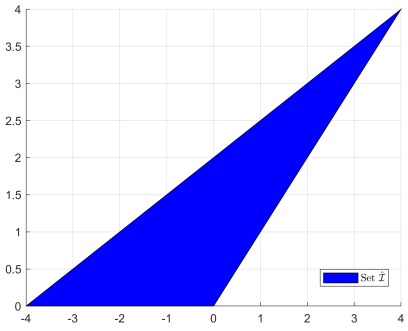
To showcase how constraints are applied for this example, Figure 4.2 is presented depicting how each constraint is applied in a successive order, obtaining the final desired set $\hat{\mathcal{I}}$. To mention that the conditions considered are the same as the ones used for Figure 4.1, that is, $lb_i = -4$ and $up_i = 4$. The final plot represents the same polygon presented in Figure 4.1.



(a) Set obtained from applying the first constraint.



(b) Set obtained from applying the first and second constraint.



(c) Set obtained from applying all constraints.

Figure 4.2: Illustration of applied constraints.

The idea will be to rewrite each inequality in the form of a CZ, i.e, removing the inequality signs and replacing $\mathbf{x}[i]$ and $\mathbf{y}[i]$ by the correct expressions. To begin with, it is important to recall the expression for a ReLU function:

$$\mathbf{y}[i] = \max(0, \mathbf{x}[i]), \tag{4.1}$$

meaning that $\mathbf{y}[i]$ varies between 0 and up_i. A plot of this function is visible in Figure 4.3.

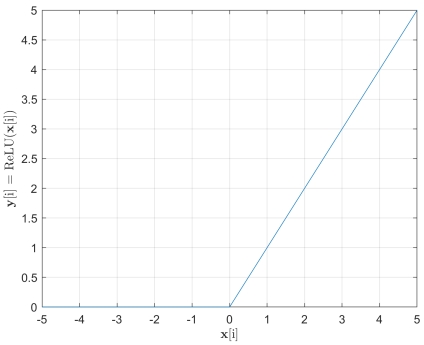


Figure 4.3: ReLU function plot.

Given this information and changing the generator for variable $\mathbf{y}[i]$ (ξ_y), (4.1) can be rewritten as:

$$\mathbf{y}[i] = up_i \xi_y, \tag{4.2}$$

where ξ_y is the generator for variable y (by Definition (2.2) can take 1 and -1 as maximum and minimum values, respectively). For this expression to be valid, ξ_y needs to be scaled to the interval $[0, 1]$, possible through the following expression:

$$\xi_y + \xi_{slack} = 1. \tag{4.3}$$

From (4.3) and according to the Definition (2.2), ξ_y is rescaled to the correct interval and hence, expression (4.2) is valid. Since the resulting set only depends on the y variable, it is possible to compose matrix $\hat{G}$. Also, the $\hat{A}$ and $\hat{b}$ matrices can be written using (4.3) as well as matrix $\hat{c}$, since variable i (input variable) does not affect the output set.

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}}, \tag{4.4}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & up_i \mathbf{e}_i & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.5}$$

$$\hat{A} = \begin{bmatrix} \mathbf{0}_{1 \times n_G} & 1 & 1 \end{bmatrix}, \tag{4.6}$$

$$\hat{b} = 1, \tag{4.7}$$

where $\mathbf{E}_i = [\mathbf{e}_1 \ \cdots \mathbf{e}_{i-1} \ \mathbf{0} \ \mathbf{e}_{i+1} \cdots \ \mathbf{e}_{n_I}]$ and n_I are the number of rows of $G_{\mathcal{I}}$ (i.e. space dimension for the input set). The first entry of (4.5) represent the generators for the input variables, which need to be canceled since (4.2) does not depend on them (in this case, variable i). The second entry represents the generators for the output variables, defined by (4.2) and $\mathbf{e}_i$ is the i-th canonical vector. From the expression (4.2), it is possible to state that the inequality $\mathbf{y}[i] \geq 0$ is met.

For inequality $\mathbf{y}[i] - \mathbf{x}[i] \geq 0$, since it is not possible to write inequalities in a CZ format and therefore, it is necessary to add a slack variable. With that, this inequality is converted to the following equality:

$$\mathbf{y}[i] + \xi_{slack} = \mathbf{x}[i], \text{valid for } \xi_{slack} \in [lb_i - up_i, 0]. \tag{4.8}$$

The final objective is get to an equality where $\xi_{slack} \in [-1, 1]$, which can only be obtained with translations (via addition and subtraction) and scaling (via multiplication and division) to the initial valid interval. Next step would be subtracting to (4.8), the value $\frac{up_i - lb_i}{2}$:

$$\mathbf{y}[i] + \xi_{slack} - \frac{up_i - lb_i}{2} = \mathbf{x}[i], \text{valid for } \xi_{slack} \in \left[\frac{lb_i - up_i}{2}, \frac{up_i - lb_i}{2} \right]. \tag{4.9}$$

Finally, $\frac{up_i - lb_i}{2}$ is multiplied to the slack variable, leading to the following expression:

$$\mathbf{y}[i] + \frac{up_i - lb_i}{2}\xi_{slack} - \frac{up_i - lb_i}{2} = \mathbf{x}[i], \text{valid for } \xi_{slack} \in [-1, 1]. \tag{4.10}$$

Given Definition (2.2) and (4.2), (4.10) is rewritten as:

$$\begin{aligned} & up_i\xi_y + \frac{up_i - lb_i}{2}\xi_{slack} - \frac{up_i - lb_i}{2} = G_{\mathcal{I}}[i,:]\xi_x + c_{\mathcal{I}}[i] \\ \Leftrightarrow\ & up_i\xi_y + \frac{up_i - lb_i}{2}\xi_{slack} - G_{\mathcal{I}}[i,:]\xi_x = c_{\mathcal{I}}[i] + \frac{up_i - lb_i}{2}, \end{aligned} \tag{4.11}$$

where ξ_x is the generator for variable x. To mention that $x[i] = G_{\mathcal{I}}[i,:]\xi_x + c_{\mathcal{I}}[i]$ once G, c characterizes the generators for the input set. Since (4.11) is in the form of a CZ, matrices (4.5), (4.6) and (4.7) can be rewritten as:

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & up_i\mathbf{e}_i & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.12}$$

$$\hat{A} = \begin{bmatrix} \mathbf{0}_{1\times n_G} & 1 & 1 & 0 \\ -G_{\mathcal{I}}[i,:] & up_i & 0 & \frac{up_i - lb_i}{2} \end{bmatrix}, \tag{4.13}$$

$$\hat{b} = \begin{bmatrix} 1 & c_{\mathcal{I}}[i] + \frac{up_i - lb_i}{2} \end{bmatrix}^T. \tag{4.14}$$

Notice that the third and fourth column of $\hat{G}$ are row-vectors of zeros, since expression (4.2) does not depend on these generators, ξ_{slack}.

For inequality $(up_i - lb_i)\mathbf{y}[i] - up_i(\mathbf{x}[i] - lb_i) \leq 0$, the previous reasoning is applied again. First, the inequality in converted to an equality using a slack variable and its limits:

$$(up_i - lb_i)\mathbf{y}[i] + \xi_{slack} = up_i\mathbf{x}[i] - up_i lb_i, \text{valid for } \xi_{slack} \in [0, up_i(up_i - lb_i)]. \tag{4.15}$$

Then, it is added $\frac{up_i(up_i - lb_i)}{2}$ to (4.15) resulting in:

$$\begin{aligned} & (up_i - lb_i)\mathbf{y}[i] + \xi_{slack} + \frac{up_i(up_i - lb_i)}{2} = up_i\mathbf{x}[i] - up_i lb_i, \\ & \text{valid for } \xi_{slack} \in \left[-\frac{up_i(up_i - lb_i)}{2}, \frac{up_i(up_i - lb_i)}{2}\right]. \end{aligned} \tag{4.16}$$

Finally, $\frac{up_i(up_i - lb_i)}{2}$ is multiplied to the slack variable, leading to the expression:

$$(up_i - lb_i)\mathbf{y}[i] + \frac{up_i(up_i - lb_i)}{2}\xi_{slack} + \frac{up_i(up_i - lb_i)}{2} = up_i\mathbf{x}[i] - up_i lb_i, \text{valid for } \xi_{slack} \in [-1, 1]. \tag{4.17}$$

As last step, replace (2.2) and (4.2) in (4.17):

$$\begin{aligned} & (up_i - lb_i)up_i\xi_y + \frac{up_i(up_i - lb_i)}{2}\xi_{slack} + \frac{up_i(up_i - lb_i)}{2} = up_i G_{\mathcal{I}}[i,:]\xi_x + up_i c_{\mathcal{I}}[i] - up_i lb_i \\ \Leftrightarrow\ & (up_i - lb_i)\xi_y + \frac{up_i - lb_i}{2}\xi_{slack} + \frac{up_i - lb_i}{2} = G_{\mathcal{I}}[i,:]\xi_x + c_{\mathcal{I}}[i] - lb_i \\ \Leftrightarrow\ & (up_i - lb_i)\xi_y + \frac{up_i - lb_i}{2}\xi_{slack} - G_{\mathcal{I}}[i,:]\xi_x = c_{\mathcal{I}}[i] - lb_i - \frac{up_i - lb_i}{2}. \end{aligned} \tag{4.18}$$

Again, since (4.18) is in the form of a CZ, matrices (4.12), (4.13) and (4.14) can be rewritten as:

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & up_i \mathbf{e}_i & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.19}$$

$$\hat{A} = \begin{bmatrix} \mathbf{0}_{1 \times n_G} & 1 & 1 & 0 & 0 \\ -G_{\mathcal{I}}[i,:] & up_i & 0 & \frac{up_i - lb_i}{2} & 0 \\ -G_{\mathcal{I}}[i,:] & up_i - lb_i & 0 & 0 & \frac{up_i - lb_i}{2} \end{bmatrix}, \tag{4.20}$$

$$\hat{b} = \begin{bmatrix} 1 & c_{\mathcal{I}}[i] + \frac{up_i - lb_i}{2} & c_{\mathcal{I}}[i] - lb_i - \frac{up_i - lb_i}{2} \end{bmatrix}^T. \tag{4.21}$$

Additionally, the constraints from the original set $\mathcal{I}$ cannot be discarded. Matrices (4.20) and (4.21) are changed to:

$$\hat{A} = \begin{bmatrix} \mathbf{0}_{1 \times n_G} & 1 & 1 & 0 & 0 \\ -G_{\mathcal{I}}[i,:] & up_i & 0 & \frac{up_i - lb_i}{2} & 0 \\ -G_{\mathcal{I}}[i,:] & up_i - lb_i & 0 & 0 & \frac{up_i - lb_i}{2} \\ A & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} \end{bmatrix}, \tag{4.22}$$

$$\hat{b} = \begin{bmatrix} 1 & c_{\mathcal{I}}[i] + \frac{up_i - lb_i}{2} & c_{\mathcal{I}}[i] - lb_i - \frac{up_i - lb_i}{2} & b_{\mathcal{I}} \end{bmatrix}^T. \tag{4.23}$$

Now, it is possible to indicate the matrices that allow to over-approximate output set when passed through a ReLU function, for a certain input $\mathcal{I}$ and for variable i.

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}}, \tag{4.24}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & up_i \mathbf{e}_i & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.25}$$

$$\hat{A} = \begin{bmatrix} \mathbf{0}_{1 \times n_G} & 1 & 1 & 0 & 0 \\ -G_{\mathcal{I}}[i,:] & up_i & 0 & \frac{up_i - lb_i}{2} & 0 \\ -G_{\mathcal{I}}[i,:] & up_i - lb_i & 0 & 0 & \frac{up_i - lb_i}{2} \\ A_{\mathcal{I}} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} \end{bmatrix}, \tag{4.26}$$

$$\hat{b} = \begin{bmatrix} 1 & c_{\mathcal{I}}[i] + \frac{up_i - lb_i}{2} & c_{\mathcal{I}}[i] - lb_i - \frac{up_i - lb_i}{2} & b_{\mathcal{I}} \end{bmatrix}^T. \tag{4.27}$$

When comparing the above matrices to the ones presented in Algorithm 2 of [19], there is some similarity. In [19], factors of $1/2$ do not show up without causing any major difference in the final results.

As a side note, the dimensions of (4.25), (4.26) and (4.27) can be reduced. If (4.3) is not taken into account $\xi_y \in [-1, 1]$, expression (4.2) for y needs to be changed to ensure it is still valid. After some arithmetic manipulations made, expression (4.28) for $\mathbf{y}[i]$ is obtained.

$$\begin{aligned} \mathbf{y}[i] &= k_{y1} \xi_y + k_{y2} \\ &= \frac{up_i}{2} \xi_y + \frac{up_i}{2} \end{aligned} \tag{4.28}$$

With that, the previous steps would have to be repeated given the new expression for y, meaning that only calculations (4.11) and (4.18) have to be redone to obtain matrices $\hat{G}, \hat{c}, \hat{A}, \hat{b}$. Using this new ex-

pression, the obtained matrices are:.

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}} + k_{y2}\mathbf{e}_i, \tag{4.29}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & k_{y1}\mathbf{e}_i & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.30}$$

$$\hat{A} = \begin{bmatrix} A_{\mathcal{I}} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} \\ -G_{\mathcal{I}}[i,:] & k_{y1} & \frac{up_i - lb_i}{2} & 0 \\ -G_{\mathcal{I}}[i,:] & \frac{up_i - lb_i}{2} & 0 & \frac{up_i - lb_i}{2} \end{bmatrix}, \tag{4.31}$$

$$\hat{b} = \begin{bmatrix} b_{\mathcal{I}} & c_{\mathcal{I}}[i] - \frac{lb_i}{2} & c_{\mathcal{I}}[i] - up_i \end{bmatrix}^T. \tag{4.32}$$

In this case, the number of constraints and generators would reduce by one, which has major computational impact for deeper and bigger NN.

4.1.2 Sigmoid function

The second function here analysed will be the logistic, sigmoid or soft step, which can be written as:

$$g(x) = \frac{1}{1 + e^{-x}}. \tag{4.33}$$

A plot of this function is presented in Figure 4.4.

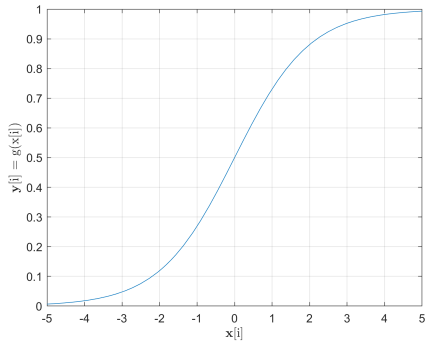


Figure 4.4: Sigmoid function plot.

The reasoning used to obtain the output set $\hat{\mathcal{I}}$ for ReLU is applied again. The first step is to find regions (i.e. inequalities) that enclose the function at hand.

In a first iteration, to obtain part of these inequalities, an optimization problem was solved to get the two lines that give the best fit to the curve, between lb_i and up_i. This problem entails minimizing the area between these lines, whilst the line below the function must pass below $g(lb_i)$ and can only intercept g after the point $(up_i, g(up_i))$ and line above must pass above $g(up_i)$ and can only intercept g after the point $(lb_i, g(lb_i))$. With these constraints, it is assured that both lines do not intercept the function.

The parameters for the line below the function are represented by m_1 and b_1 in the following expressions, while the parameters for the line above the function are represented by m_2 and b_2.

The above description can be translated into the optimization problem (4.34). Set of points x and y are generated according to lower and upper bounds of each variable and $step$ defines the spacing between

each point x.

$$\begin{aligned} \min_{m_1,m_2,b_1,b_2} \quad & |m_1 \cdot x + b_1 - m_2 \cdot x + b_2| \cdot step \\ \text{s.t.} \quad & m_1 \cdot x + b_1 \leq y, \\ & m_2 \cdot x + b_2 \geq y, \\ & m_1 \cdot lb_i + b_1 \leq g(lb_i), \\ & m_2 \cdot up_i + b_2 \geq g(up_i). \end{aligned} \tag{4.34}$$

In a second iteration, to shorten runtime, a corrected version of function [17] implementing *Khachiyan Algorithm* was used. This function computes a minimum-volume covering ellipsoid that encloses N points in a D-dimensional space. More details on how to determine the needed inequalities and the ellipse are presented in Sections 4.3 and 4.4.

Figure 4.5 presents a graphical representation of the lines determined for this function. To mention that this plot was generated using the second approach, while in the first one, lines are more tight to each point.

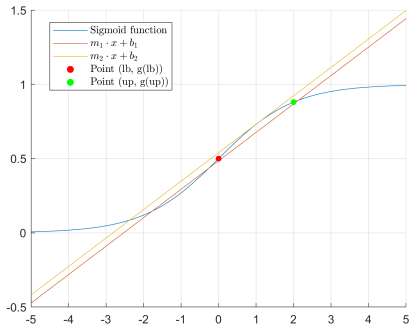


Figure 4.5: Graphical representation of lines determined for the sigmoid function.

From these lines, it is possible to write the desired inequalities since they have the same slope and different values of y-intercept. Therefore, two inequalities are obtained: $\mathbf{y}[i] \geq m_1\mathbf{x}[i] + b_1$ and $\mathbf{y}[i] \leq m_2\mathbf{x}[i] + b_2$. The two missing inequalities are $\mathbf{y}[i] \geq g(lb_i)$ and $\mathbf{y}[i] \leq g(up_i)$, which are true when $\mathbf{y}[i]$ is correctly defined. Having written the expressions, the next step is to define $\mathbf{y}[i]$, similar to (4.2). Since $\mathbf{y}[i] \in [g(lb_i), g(up_i)]$ and $\xi_y \in [-1, 1]$ (by definition), $\mathbf{y}[i]$ is defined by:

$$\begin{aligned} \mathbf{y}[i] &= k_{y1}\xi_y + k_{y2} \\ &= \frac{g(up_i) - g(lb_i)}{2}\xi_y + \frac{g(up_i) + g(lb_i)}{2}. \end{aligned} \tag{4.35}$$

To note that since this expression in generic (since it only depends on the values of bounds for each variable) it will be used throughout this chapter. When that occurs, it will be referred.

The following procedure would be to convert these inequalities in form of a CZ using (4.35) and (2.2), similarly to what was done in the previous section for the ReLU activation function. With that, matrices

that parameterize $\hat{\mathcal{I}}$ for a sigmoid activation function are obtained.

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}} + k_{y2}\mathbf{e}_i, \tag{4.36}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & k_{y1}\mathbf{e}_i & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.37}$$

$$\hat{A} = \begin{bmatrix} A_{\mathcal{I}} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} \\ -m_1 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & k_1 & 0 \\ -m_2 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & 0 & k_2 \end{bmatrix}, \tag{4.38}$$

$$\hat{b} = \begin{bmatrix} b_{\mathcal{I}} & m_1 \cdot c_{\mathcal{I}}[i] + b_1 - k_1 - k_{y2} & m_2 \cdot c_{\mathcal{I}}[i] + b_2 - k_2 - k_{y2} \end{bmatrix}^T. \tag{4.39}$$

The constants used (k_1 and k_2) are defined as:

$$k_1 = \frac{m_1 \times lb_i + b_1 - g(up_i)}{2}, \quad k_2 = \frac{m_2 \times up_i + b_2 - g(lb_i)}{2},$$

where m_1, b_1, m_2 and b_2 are the constants previously referred. Algorithm 1 presents the steps used to compute sets, where [FUNCTION] is the sigmoid. To note that this function is applied to all variables, regardless of their lower and upper bounds values.

4.1.3 Softplus function

The next function here analysed will be the softplus, which can be written as:

$$g(x) = \ln(1 + e^x). \tag{4.40}$$

A plot of this function is presented in Figure 4.6.

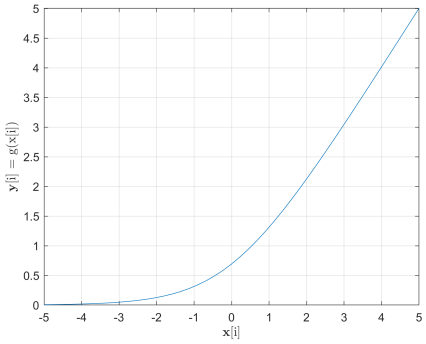


Figure 4.6: Softplus function plot.

Here, there are two possible solutions to obtain the needed inequalities:

- three inequalities, two of them obtained by solving an optimization problem and one delimited above by the points $(lb_i, g(lb_i))$ and $(up_i, g(up_i))$;
- two inequalities obtained by the ellipse.

While the first one allows to obtain a more accurate approximation of the real output set, it requires solving an optimization problem which has a variable execution time and the parameters of the output set $\hat{\mathcal{I}}$ are, computationally speaking, more heavy. The second approach is two to three times faster than the first one, while computing a set with a worst approximation. Although the execution time is not constant for the first approach, it can be more time-consuming when compared to the second one and

does not provide any benefits. For both solutions, matrices that characterize each output set will be presented.

For the first solution, an optimization problem was solved to get the two lines that give the best fit to the curve, between lb_i and up_i. A set of points x and y are generated according to bounds of each variable. Since g has a single concavity facing upwards, the optimization problem is different from the previous.

This problem entails minimizing the area between each line and points y, whilst each line must pass on points $(lb_i, g(lb_i))$ and $(up_i, g(up_i))$ and both lines can only intercept g after the points referred.

The parameters for the first line are represented by m_1 and b_1, while the parameters for the second line are represented by m_2 and b_2. The above description can be traduced in the following optimization problem:

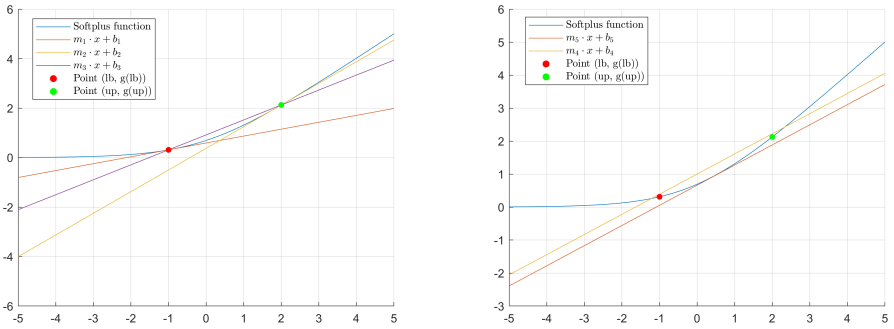
$$\begin{aligned} \min_{m_1,m_2,b_1,b_2} \quad & (|m_1 \cdot x + b_1 - y| + |m_2 \cdot x + b_2 - y|) \cdot step \\ \text{s.t.} \quad & m_1 \cdot x + b_1 \leq y, \\ & m_2 \cdot x + b_2 \leq y, \\ & m_1 \cdot lb_i + b_1 = g(lb_i), \\ & m_2 \cdot up_i + b_2 = g(up_i). \end{aligned} \tag{4.41}$$

$Step$ defines the spacing between each point x. Solving the problem above results in two inequalities: $\mathbf{y}[i] \geq m_1\mathbf{x}[i]+b_1$ and $\mathbf{y}[i] \geq m_2\mathbf{x}[i]+b_2$. The third inequality is defined by the bounds: $\mathbf{y}[i] \leq m_3\mathbf{x}[i]+b_3$, where:

$$m_3 = \frac{g(up_i) - g(lb_i)}{up_i - lb_i}, \quad b_3 = g(up_i) - m_3 \cdot up_i.$$

Figure 4.7 presents a graphical representation of the lines determined for the softplus using different approaches. For the first approach, the behaviour of the function at hand is better captured however, it uses more constraints and has an higher execution time.

As in the sigmoid function, these lines are converted into inequalities depending on the values of y-intercept, obtaining the above expressions.



(a) First approach.

(b) Second approach.

Figure 4.7: Graphical representation of lines determined for the softplus function using different approaches.

Using the valid expression for $\mathbf{y}[i]$ ((4.35)) and, after converting these inequalities in form of a CZ, the

output set can be parameterized by the following matrices:

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}} + k_{y2}\mathbf{e}_i, \tag{4.42}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & k_{y1}\mathbf{e}_i & \mathbf{0}_{n_I\times 1} & \mathbf{0}_{n_I\times 1} & \mathbf{0}_{n_I\times 1} \end{bmatrix}, \tag{4.43}$$

$$\hat{A} = \begin{bmatrix} A_{\mathcal{I}} & \mathbf{0}_{n_A\times 1} & \mathbf{0}_{n_A\times 1} & \mathbf{0}_{n_A\times 1} & \mathbf{0}_{n_A\times 1} \\ -m_1 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & k_1 & 0 & 0 \\ -m_2 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & 0 & k_2 & 0 \\ -m_3 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & 0 & 0 & k_3 \end{bmatrix}, \tag{4.44}$$

$$\hat{b} = \begin{bmatrix} b_{\mathcal{I}} & m_1 \cdot c_{\mathcal{I}}[i] + b_1 - k_1 - k_{y2} & m_2 \cdot c_{\mathcal{I}}[i] + b_2 - k_2 - k_{y2} & m_3 \cdot c_{\mathcal{I}}[i] + b_3 - k_3 - k_{y2} \end{bmatrix}, \tag{4.45}$$

where k_1, k_2 and k_3 are defined as:

$$k_1 = \frac{m_2 \times lb_i + b_2 - g(up_i)}{2}, \quad k_2 = \frac{m_3 \times lb_i + b_3 - g(up_i)}{2}, \quad k_1 = \frac{m_1 \times up_i + b_1 - g(lb_i)}{2}.$$

For the second solution, as in the sigmoid function, two inequalities are obtained: $\mathbf{y}[i] \leq m_4\mathbf{x}[i] + b_4$ and $\mathbf{y}[i] \geq m_5\mathbf{x}[i] + b_5$, where $\mathbf{y}[i]$ being equal to (4.35). Using the appropriate Definitions ((2.2) and (4.35)) and after converting these inequalities in form of a CZ, the matrices below are obtained.

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}} + k_{y2}\mathbf{e}_i, \tag{4.46}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & k_{y1}\mathbf{e}_i & \mathbf{0}_{n_I\times 1} & \mathbf{0}_{n_I\times 1} \end{bmatrix}, \tag{4.47}$$

$$\hat{A} = \begin{bmatrix} A_{\mathcal{I}} & \mathbf{0}_{n_A\times 1} & \mathbf{0}_{n_A\times 1} & \mathbf{0}_{n_A\times 1} \\ -m_4 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & k_4 & 0 \\ -m_5 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & 0 & k_5 \end{bmatrix}, \tag{4.48}$$

$$\hat{b} = \begin{bmatrix} b_{\mathcal{I}} & m_4 \cdot c_{\mathcal{I}}[i] + b_4 - k_4 - k_{y2} & m_5 \cdot c_{\mathcal{I}}[i] + b_5 - k_5 - k_{y2} \end{bmatrix}^T. \tag{4.49}$$

Constants k_4 and k_5 is defined as:

$$k_4 = \frac{m_4 \times lb_i + b_4 - g(lb_i)}{2}, \quad k_5 = \frac{m_5 \times up_i + b_5 - g(up_i)}{2},$$

As previously mentioned, the first approach is more time-consuming and therefore, during simulation, the second approach is the preferred one to be used.

4.1.4 Leaky Rectified Linear Unit function

The following function here analysed will be the Leaky ReLU, which can be written as:

$$g(x) = \begin{cases} 0.01x & x < 0 \\ x & x \geq 0 \end{cases}. \tag{4.50}$$

A plot of this function is visible in Figure 4.8.

The main difference from this function to the ReLU occurs for $x < 0$, where instead of being zero, it is described by the linear relationship $y = 0.01x$. Additionally since this function does not contain any non-linear section, there is no need to use an optimization problem to discover the inequalities that enclose function g.

For this function, three points are considered in order to determine the inequalities: $(0, 0)$, $(lb_i, g(lb_i))$ and $(up_i, g(up_i))$. This allows to define the three following inequalities: $\mathbf{y}[i] \geq x$, $lb_i\mathbf{y}[i] \geq g(lb_i)\mathbf{x}[i]$ and

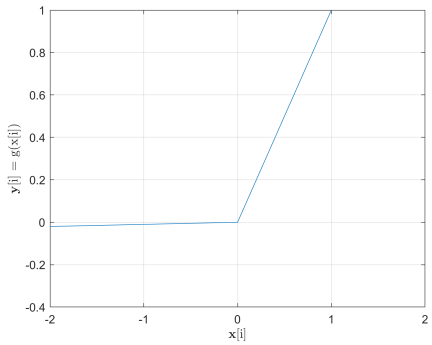


Figure 4.8: Leaky ReLU function plot.

$(up_i - lb_i)\mathbf{y}[i] \leq (g(up_i) - g(lb_i))\mathbf{x}[i] + up_i g(lb_i)) - lb_i g(up_i)$.

The output for each variable i ($\mathbf{y}[i]$) is (4.35) and, after some calculations, matrices that characterize the output set are obtained. In the matrices written below, m_1 and b_1, m_2 and b_2 and m_3 and b_3 represent the slope and y-intercept for the first, second and third inequality, respectively.

$$\hat{c} = \mathbf{E}_i c_{\mathcal{I}} + k_{y2}\mathbf{e}_i, \tag{4.51}$$

$$\hat{G} = \begin{bmatrix} \mathbf{E}_i G_{\mathcal{I}} & k_{y1}\mathbf{e}_i & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} & \mathbf{0}_{n_I \times 1} \end{bmatrix}, \tag{4.52}$$

$$\hat{A} = \begin{bmatrix} A_{\mathcal{I}} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} & \mathbf{0}_{n_A \times 1} \\ -m_1 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & k_1 & 0 & 0 \\ -m_2 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & 0 & k_2 & 0 \\ -m_3 \cdot G_{\mathcal{I}}[i,:] & k_{y1} & 0 & 0 & k_3 \end{bmatrix}, \tag{4.53}$$

$$\hat{b} = \begin{bmatrix} b_{\mathcal{I}} & m_1 \cdot c_{\mathcal{I}}[i] + b_1 - k_1 - k_{y2} & m_2 \cdot c_{\mathcal{I}}[i] + b_2 - k_2 - k_{y2} & m_3 \cdot c_{\mathcal{I}}[i] + b_3 - k_3 - k_{y2} \end{bmatrix}, \tag{4.54}$$

where constants k_1, k_2 and k_3 can be written as:

$$k_1 = \frac{m_1 \times lb_i + b_1 - g(up_i)}{2}, \quad k_2 = \frac{m_2 \times lb_i + b_2 - g(up_i)}{2}, \quad k_3 = \frac{m_3 \times up_i + b_3 - g(lb_i)}{2}.$$

4.1.5 Summary

Throughout this section, matrices that allow to over-approximate output sets were presented. These matrices are to be used in Algorithm 1, where is presented a scheme of other necessary steps to compute these sets, in which [FUNCTION] refers to the function being analysed. To remember that $\mathcal{I}$ is the input set for each variable and $\hat{\mathcal{I}}$ is the respective output set, which in turn is the next input set.

It is also important to point out that there is currently a wide variety of activation functions, each with its own main purpose. The reasoning behind the choice of activation functions was to select the most widely used and known in the literature, and also to demonstrate different ways of obtaining constraints. For instance, for the Tanh and sigmoid functions, the method that would be used to obtain constraints would be the same and, therefore, not relevant to mention both. The main idea is present and explain the reasoning used to obtain these matrices, making possible to extend it to any other activation function.

It is important to mention that resorting to optimization problems or any other methodologies only occurs due to the non-linearities in some activations functions. When that is not the case, it is only necessary to determine the needed points and, with that, get the inequalities. This last activation function presented only contained linear parts and hence, three points were used: origin, lower and upper bounds.

Algorithm 1 Over-approximated output analysis for one layer of FNN for a certain activation function

Input: weight matrix W, bias vector v, constrained zonotope input set $\mathcal{Z}$
Output: over-approximated output set $\hat{\mathcal{R}}$

function OVERREACHNN($\mathcal{Z}$, W, v)
  $\mathcal{I} = W\mathcal{Z} + v$
  $[lb, up] \leftarrow$ range of x in $\mathcal{I}$
  for *i in length(*lb*)* **do**
    $\mathcal{I} =$ OVERSTEP[FUNCTION]$(\mathcal{I}, i, lb_i, up_i)$
  return $\hat{\mathcal{R}} = \mathcal{I}$

function OVERSTEP[FUNCTION]$(\mathcal{I}, i, lb_i, up_i)$
  $\mathcal{I} = CZ\{c, G, A, b\} \subset \mathbb{R}^{n_I}$
  $E_i = [e_1 \ \cdots \ e_{i-1} \ \mathbf{0} \ e_{i+1} \ \cdots \ e_{n_I}]$
  $\hat{\mathcal{I}} = CZ\{\hat{c}, \hat{G}, \hat{A}, \hat{b}\}$
  return $\hat{\mathcal{I}}$

4.2 Activation Functions Overbound using Constrained Convex Generators

As initially stated, the exact approach was abandoned. Definition 2.6 presents the characterization for a CCG.

The following subsections present algorithms to compute over-approximated sets for a set of activation functions, with each one being thoroughly described. Some operations on CCGs are performed during the execution of these algorithms, such as intersections and linear mappings. For that purpose, the functions listed in [16] were utilized, which are implemented in toolbox available in `https://github.com/danielmsilvestre/ReachTool`.

Again, to note that the rationale presented for each activation function is applied to each variable, which after computing all the needed operations for all variables, permits finding $\mathcal{Z}$.

4.2.1 Hyperbolic tangent function

The first function here analysed will be the Tanh, which can be written as (4.55). The advantages of CCGs are highlighted since both concavities of g are better approximated with non-linear functions (which can be produced by a CCGs) rather than using linear functions.

$$\begin{aligned} g(x) &= tanh(x) \\ &= \frac{e^x - e^{-x}}{e^x + e^{-x}}. \end{aligned} \tag{4.55}$$

A plot of this function is visible in Figure 4.9.

It will be used two different polygons to over-approximate the input set: an ellipse and two straight lines, both defining regions containing points comprised between lb_i and up_i. The first one is calculated using function [17], while the second type of polygon is obtained by the ellipse.

Similarly to CZs, both lines are used to define a CCG as well as the ellipse, with these lines being computed in a different way. After defining both sets, the next step is to create another set containing the input set and a variable limiting the output. As last step, these sets are intercepted and one of the variables is projected, using a matrix E_i. A more detailed description to obtain each CCG will be presented below.

Although it is possible to define CCGs with inequalities, it can be computationally intractable since it

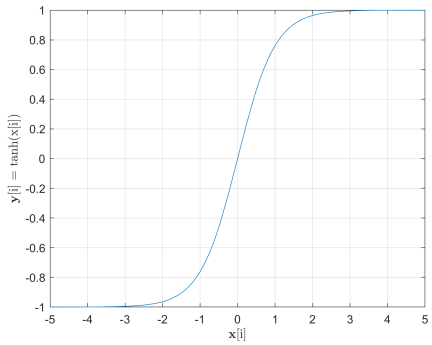


Figure 4.9: Tanh function plot.

requires defining more variables. The approach to find the correct CCG is the same: convert inequalities in equalities using slack variables.

To compute the minimum volume enclosing ellipsoid, a set of points (x, y), where $x \in [lb_i, up_i]$ and $y = g(x)$ is considered. It is used a corrected version function [17] (more detailed in Section 4.3), which outputs a matrix A and vector c respecting the following equality:

$$(x - c)^T A(x - c) = 1, \tag{4.56}$$

which is not in form of a CCG. Therefore, eigenvalues and eigenvectors of matrix A are computed, where the highest and lowest eigenvalues allow to compute a and b, respectively. The angle of rotation of the ellipse, θ, is given by the arc-tangent of the y-component and x-component of the eigenvector with the highest eigenvalue associated. Vector c already defines the center of this polygon. Given these values, it is defined the following CCG:

$$G = R \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} R^T, \tag{4.57}$$

$$A = \mathbf{0}_{0\times 2}, \tag{4.58}$$

$$b = \mathbf{0}_{0\times 1}, \tag{4.59}$$

$$\mathfrak{C} = \mathcal{B}_2, \tag{4.60}$$

where $\mathcal{B}_2$ is the unit ℓ_2 ball and R is a rotation matrix defined by the matrix below. This set will be denoted as Y and represents the ellipse.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{4.61}$$

For the lines, the set (denoted as Z_{lines}) is constructed using inequalities, which amount depends on the value of the bounds. The goal of this set is to reflect the constraints associated with inequalities, which will eventually intersect with set Y. As a result, it is required to define $\mathbf{x}[i]$, since the input will not be cancelled. In practice, this is accomplished by first generating a box whose limits are defined by the value of the bounds and then intersecting with these inequalities. The last of step of this procedure is to project each variable onto their respective axis.

Similarly to what was done for $\mathbf{y}[i]$, $\mathbf{x}[i]$ can also be expressed using ξ_x since, by definition, $\xi_x \in [-1, 1]$:

$$\begin{aligned} \mathbf{x}[i] &= k_{x1}\xi_x + k_{x2} \\ &= \frac{up_i - lb_i}{2}\xi_x + \frac{up_i + lb_i}{2}. \end{aligned} \tag{4.62}$$

Figure 4.10 present a plot of lines determined for the Tanh function when using the ellipse.

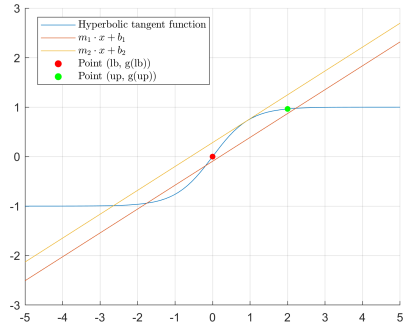


Figure 4.10: Graphical representation of lines determined for the Tanh function.

The inequalities are derived from the equalities (calculated from the ellipse): $\mathbf{y}[i] \geq m_1\mathbf{x}[i] + b_1$ and $\mathbf{y}[i] \leq m_2\mathbf{x}[i] + b_2$. Using these inequalities and expressions (4.62) and (4.35), the following matrices characterizing Z_{lines} are yielded:

$$c = \begin{bmatrix} k_{x2} \\ k_{y2} \end{bmatrix}, \tag{4.63}$$

$$G = \begin{bmatrix} k_{x1} & 0 & 0 & 0 \\ 0 & k_{y1} & 0 & 0 \end{bmatrix}, \tag{4.64}$$

$$A = \begin{bmatrix} -m_1 \cdot k_{x1} & k_{y1} & k_1 & 0 \\ -m_2 \cdot k_{x1} & k_{y1} & 0 & k_2 \end{bmatrix}, \tag{4.65}$$

$$b = \begin{bmatrix} m_1 \cdot k_{x2} + b_1 - k_1 - k_{y2} & m_2 \cdot k_{x2} + b_2 - k_2 - k_{y2} \end{bmatrix}^T, \tag{4.66}$$

$$\mathfrak{C} = \mathcal{B}_\infty, \tag{4.67}$$

where k_1 and k_2 are defined as:

$$k_1 = \frac{m_1 \times lb_i + b_1 - g(up_i)}{2}, \quad k_2 = \frac{m_2 \times up_i + b_2 - g(lb_i)}{2},$$

For the third set, this results from concatenating the input set with a variable that limits the output set for one single variable. For instance, for a two-dimensional set, a variable would be added to limit the output since $\mathbf{y}[i] \in [g(lb_i), g(up_i)]$, resulting in a three-dimensional set. This set, denoted as $\mathcal{I}_{new}$, can be parameterized by the following matrices:

$$c = \begin{bmatrix} c_\mathcal{I} \\ k_{y2} \end{bmatrix}, \tag{4.68}$$

$$G = \begin{bmatrix} G_\mathcal{I} & k_{y1} \end{bmatrix}, \tag{4.69}$$

$$A = \begin{bmatrix} A_\mathcal{I} & \mathbf{0}_{n_{c_\mathcal{I}} \times 1} \end{bmatrix}, \tag{4.70}$$

$$b = b_\mathcal{I}, \tag{4.71}$$

$$\mathfrak{C} = \{\mathfrak{C}_\mathcal{I}, \mathcal{B}_\infty\}. \tag{4.72}$$

To obtain the over-approximated set for each variable, the first step is to intersect Z_{lines} and Y, resulting on a CCG (denoted as $constraint\_set$) used to impose constraints to a set of variables of input and output variables. Then, $\mathcal{I}_{new}$ and $constraint\_set$ are intersected using matrix P, enabling to apply the

constraints to the correct pair of input and output, resulting in set denoted as $inter\_set$. For example considering a two-dimensional input set, $\mathcal{I}_{new}$ is defined by variables x_1, x_2, y_1 (where y_1 is the added variable). When applying this algorithm to the first input variable, the set of variables to be constrained are x_1, y_1 and, therefore, $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

As done for CZs, the final step is projecting the variable being analysed (since the output does not depend on it), done through applying a linear map to $inter\_set$ using a matrix E_i. Taking the same example, x_1 would be the variable to be projected and, therefore, $E_i = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. The result of this final operation leads to the output set. If there are any variables to analyse, then the over-approximated reachable set $\hat{\mathcal{I}}$ is determined.

Since this process is iterative and applied to all variables, the next step would be to apply the same process to following variable (if the set is not one-dimensional). Algorithm 2 presents a summarized version of all the steps necessary to compute the over-approximated set, where [FUNCTION] is the Tanh.

4.2.2 Softplus function

The second function where this new set representation will be used is the softplus. The expression was introduced previously and can be written as (4.40) and a plot was presented in Figure 4.6. When comparing the approach made for CZs, it is possible to conclude that using this new set representation is more advantageous regarding time-performance ratio. This happens since the two of the lines used as constraints are now replaced by an ellipse, which are obtained by arithmetic operations and not by an optimization problem. Apart from the ellipse, the missing constraint is an inequality defined by points $(lb_i, g(lb_i))$ and $(up_i, g(up_i))$.

As was done for the Tanh, using a set of points (x, y) and modified version of function [17] an ellipse is determined. After that and repeating the process described for the previous function, set Y is obtained. Since the softplus function has a single concavity facing upwards, the missing constraint is $\mathbf{y}[i] \leq m\mathbf{x}[i] + b$, where $m = \frac{g(up_i) - g(lb_i)}{up_i - lb_i}$ and $b = g(up_i) - m_1 \cdot up_i$. Using (4.35) and (4.62) and converting this inequality to an equality, Z_{lines} is obtained and defined by the following matrices:

$$c = \begin{bmatrix} k_{x2} \\ k_{y2} \end{bmatrix}, \tag{4.73}$$

$$G = \begin{bmatrix} k_{x1} & 0 & 0 \\ 0 & k_{y1} & 0 \end{bmatrix}, \tag{4.74}$$

$$A = \begin{bmatrix} -m \cdot k_{x1} & k_{y1} & k \end{bmatrix}, \tag{4.75}$$

$$b = \begin{bmatrix} m \cdot k_{x2} + b - k - k_{y2} \end{bmatrix}^T, \tag{4.76}$$

$$\mathfrak{C} = \mathcal{B}_\infty, \tag{4.77}$$

where $k = \frac{m \times up_i + b - g(lb_i)}{2}$. The process to obtain set L_{new} is repeated and is parameterized by matrices (4.68) to (4.72). The previous reasoning to compute the over-approximated output set $\hat{\mathcal{I}}$ is also repeated.

4.2.3 Sigmoid linear unit function

The last function here analysed is the SiLU function, which expression is (4.78).

$$g(x) = \frac{x}{1 + e^{-x}} \tag{4.78}$$

A plot of this function is presented in Figure 4.11.

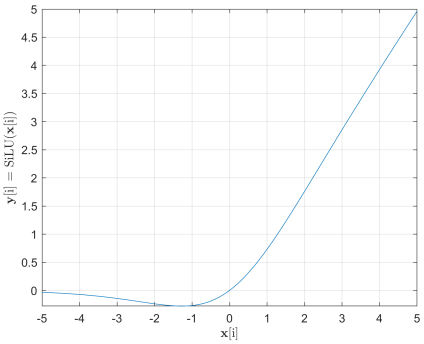


Figure 4.11: SiLU function plot.

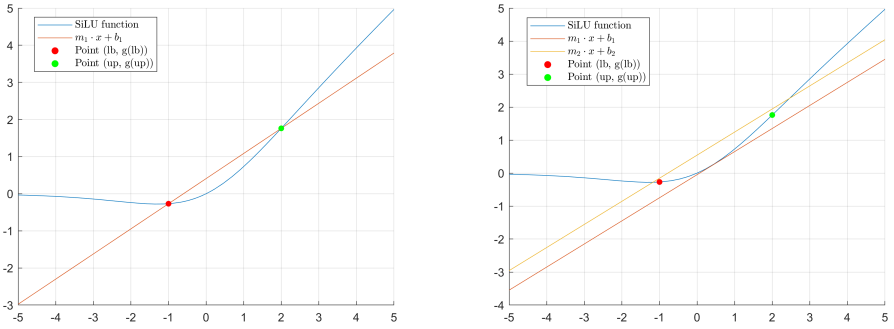
To obtain a set that represents a ellipse, denoted as Y, the same process mentioned for the Tanh function is repeated. To determine the inequalities, which are used to define set Z_{lines}, there are two approaches: solving an optimization problem or using the ellipse. For the optimization problem, the objective is to find a line with slope m and y-intercept b which is above the function for the interval $[lb_i, up_i]$ and minimizing the area between this line and the set of points. This set of points is generated according to values of the bounds.

The above description is traduced in the following optimization problem:

$$\begin{aligned} \min_{m,b} \quad & |m \cdot x + b - y| \cdot step \\ \text{s.t.} \quad & m \cdot x + b \geq y. \end{aligned} \tag{4.79}$$

Given m and b, the region that enclose function g is defined by inequality $\mathbf{y}[i] \leq m\mathbf{x}[i] + b$.

Figure 4.5 presents a graphical representation of the lines determined for the SiLU using different approaches. For the first approach, the only missing constraints are the ones delimiting x and y according to the boundaries value, which are included when defining the CCG. The same occurs for the second approach.



(a) First approach.

(b) Second approach.

Figure 4.12: Graphical representation of applied constraints for the SiLU function using different approaches.

After converting this inequality in form of a CCG and using Equation (4.35), matrices that parameterize

set Z_{lines} are:

$$c = \begin{bmatrix} k_{x2} \\ k_{y2} \end{bmatrix}, \tag{4.80}$$

$$G = \begin{bmatrix} k_{x1} & 0 & 0 \\ 0 & k_{y1} & 0 \end{bmatrix}, \tag{4.81}$$

$$A = \begin{bmatrix} -m \cdot k_{x1} & k_{y1} & k \end{bmatrix}, \tag{4.82}$$

$$b = \begin{bmatrix} m \cdot k_{x2} + b - k - k_{y2} \end{bmatrix}^T, \tag{4.83}$$

$$\mathfrak{C} = \mathcal{B}_\infty, \tag{4.84}$$

where $k = \frac{m \times up_i + b - g(lb_i)}{2}$. For the second solution, two inequalities are obtained: $\mathbf{y}[i] \leq m_1 \mathbf{x}[i] + b_1$ and $\mathbf{y}[i] \geq m_2 \mathbf{x}[i] + b_2$, with $\mathbf{y}[i]$ being equal to (4.35).

$$c = \begin{bmatrix} k_{x2} \\ k_{y2} \end{bmatrix}, \tag{4.85}$$

$$G = \begin{bmatrix} k_{x1} & 0 & 0 \\ 0 & k_{y1} & 0 \end{bmatrix}, \tag{4.86}$$

$$A = \begin{bmatrix} -m_1 \cdot k_{x1} & k_{y1} & k_1 & 0 \\ -m_2 \cdot k_{x1} & k_{y1} & 0 & k_2 \end{bmatrix}, \tag{4.87}$$

$$b = \begin{bmatrix} m_1 \cdot k_{x2} + b_1 - k_1 - k_{y2} & m_2 \cdot k_{x2} + b_2 - k_2 - k_{y2} \end{bmatrix}^T, \tag{4.88}$$

$$\mathfrak{C} = \mathcal{B}_\infty, \tag{4.89}$$

where k_1 and k_2 can be written as:

$$k_1 = \frac{m_1 \times lb_i + b_1 - g(lb_i)}{2}, \quad k_2 = \frac{m_2 \times up_i + b_2 - g(up_i)}{2}.$$

The process to determine $\mathcal{I}_{new}$ is repeated and the set that over-approximates the output is obtained and defined by $\hat{\mathcal{I}}$.

4.2.4 Summary

Algorithm 2 presents a scheme of steps to compute the over-approximated set, in which [FUNCTION] is the function at hand. Matrices presented throughout this section are to be used in this algorithm. Also, it is possible to observe that, in most functions presented for CZs, some of the constraints are replaced by an ellipse, which is one of the advantages of this new set representation. Throughout this section, ellipses were used to define constraints (directly and indirectly) because they fit the problem at hand. However, there is a wide range of polygons that can be used.

As mentioned previously, not all the more used activation functions were documented since the methods used to determine constraints do not vary. For example, for the Leaky ReLU and Softplus functions, the needed constraints would be an inequality delimited above by the bounds and an ellipse. Therefore, documenting both functions is not relevant.

It is important to refer that the set representing an ellipse would be sufficient to approximate the output and the inequalities provide a more precise over-approximated set i.e., closer to the real output set.

The next chapter presents simulation results on various networks, each one detailed, using the aforementioned algorithms.

Algorithm 2 Over-approximated output analysis for one layer of FNN

Input: weight matrix W, bias vector v, constrained zonotope input set $\mathcal{Z}$
Output: over-approximated output set $\hat{\mathcal{R}}$

function OVERREACHNN($\mathcal{Z}$, W, v)
  $\mathcal{I} = W\mathcal{Z} + v$
  $[lb, up] \leftarrow$ range of x in $\mathcal{I}$
  for *i in length(lb)* **do**
    $\mathcal{I} = \text{OVERSTEP[FUNCTION]}(\mathcal{I}, i, lb_i, up_i)$
  return $\hat{\mathcal{R}} = \mathcal{I}$

function OVERSTEP[FUNCTION]($\mathcal{I}, i, lb_i, up_i$)
  $\mathcal{I} = CZ\{c, G, A, b\} \subset \mathbb{R}^{n_I}$
  $E_i = [e_1 \ \cdots \ e_{i-1} \ \mathbf{0} \ e_{i+1} \ \cdots \ e_{n_I}]$

  $Y = CCG\{c, G, A, b, \mathfrak{C}\}$
  $Z_{lines} = CCG\{c, G, A, b, \mathfrak{C}\}$
  $\mathcal{I}_{new} = CCG\{c, G, A, b, \mathfrak{C}\}$

  $constraint\_set = CCGIntersect(I_2, Z_{lines}, Y)$
  $inter\_set = CCGIntersect(P, L_{new}, constraint\_set)$

  $\hat{\mathcal{I}} = CCGLinMap(E_i, inter\_set, \mathbf{0}_{2\times 1})$
  return $\hat{\mathcal{I}}$

4.3 Application of Khachiyan Algorithm to Compute Quadratic Constraints

Throughout this chapter, algorithms to determine the over-approximated output set were presented and explained. As previously mentioned, function [17] computes a minimum-volume covering ellipsoid that encloses N points in a D-dimensional space. In this work, space is two-dimensional and points (x, y) are generated according to the values of lower and upper bounds.

For all algorithms presented in this chapter, a methodology to both determine an ellipse and derive inequalities from it was presented.

Although function [17] generally outputs correct parameters for the ellipse, there are some cases where parameters a, b of the ellipse are miscalculated. This results in incorrect inequalities and, consequently producing incorrect output sets. To solve this problem, a algorithm was developed to modify the ellipse, allowing to determine the correct one. With this corrected version of the ellipse is then possible to obtain the required constraints, being inequalities when considering CZs and adding an ellipse when using CCGs.

Initially, after a thorough examination, it was first able to identify that these parameters were incorrect when, locally, points generated according to the values of the bounds could be characterized as a linear function. To confirm this fact, a linear regression is done to this points and the sum of the distances, indicated as *residualsSum*, from these points to this line is computed. It was possible to deduce that the value of a and/or b would be wrong if *residualsSum* was below a certain value. In these cases, a and/or b were substituted by 0.001 and 0.01, respectively and the threshold value considered for *residualsSum* was 0.02. The parameters and thresholds used were chosen as a result of numerous tests, allowing to determine values that would lead to a correct ellipse.

After these initial modifications, the next step is to find whether the four tangent lines to the ellipse (passing on the vertices and co-vertices) define a region containing points (x, y). Initially, lines tangent to the co-vertices of the ellipse are found and checked whether they are above or below the points

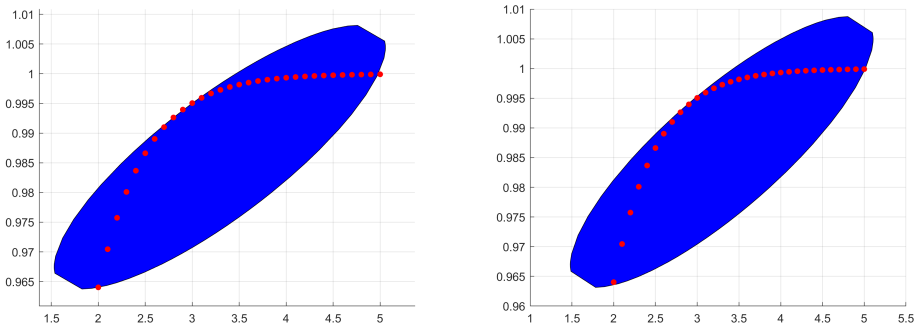
(x, y). If this condition is not fulfilled, a is increased 0.3 (depending on the order of magnitude of a), i.e. $a = a + 0.3 \cdot 10^{order_a - 1}$, where $order_a$ represents the order of magnitude of a. During this stage, b is also adjusted since the region between these lines might contain the points that are outside the ellipse. Therefore, if any point lie outside b is increased 0.5 (depending on the order of magnitude of b), i.e., $b = b + 0.5 \cdot 10^{order_b - 1}$, where $order_b$ is the order of magnitude of b. This process was repeated until this region contained points (x, y).

In addition to these adjustments, the function itself needed some changes. In some cases, the data used to determine the ellipse creates a matrix that is not singular and hence, it is not possible to compute its inverse. To overcome this problem, the *Moore–Penrose* inverse of the ellipse was computed.

Another issue was that for some activation functions this function would run indefinitely. What would happen was that in the first iteration, the error was minimum and, in the following iterations, would increase and remain constant in a higher value. Therefore, the execution of the while cycle (line 10 of algorithm 3) would stop when the runtime was greater than 20 seconds and the variable err was lower than 20. As will be discussed in Chapter 5, this problems occurs for the exponential function for higher values of bounds.

Another requirement defined for the exponential function was that if $lb > 18$, the while cycle had to terminate in its first iteration. This was imposed since for these values of bounds, variable err is minimum in the first iteration and afterwards increases and remains constant in a higher value. This would result in an infinite execution, and terminating it (in the above conditions) results in a correct ellipse.

Figure 4.13 presents the ellipse obtained before and after correcting function [17] with the proposed aforementioned changes. These figures prove the effectiveness of these corrections, since the set of points (x, y) lie inside the set for Figure 4.13b.



(a) Before the corrections.

(b) After the corrections.

Figure 4.13: Graphical representation of the ellipse obtained and set of points (x, y) before and after the correction.

It is also important to mention that, with these changes, the corrected ellipse is larger than the original one, presented in Figure 4.14. As it will be discussed in Section 6.2, using other algorithms to obtain inequalities or improving the ones here referred would have improvements in these algorithms.

Algorithm 3 presents a modified version of function [17], comprising only the changes related to time execution. Additional corrections, such as, changing values of a and b are made separately from this function.

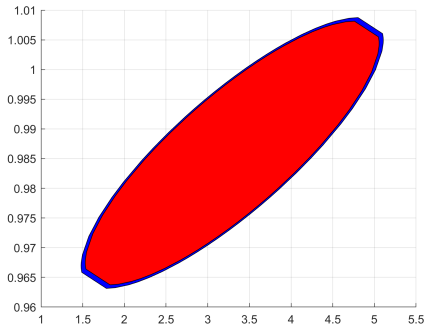


Figure 4.14: Intersection of ellipses obtained before and after the corrections.

4.4 Method to Translate a Quadratic Constraint to CCG Format

After modifying the original function [17], it is possible to determine the relevant tangent lines to the ellipse, allowing the desired inequalities to be defined.
According to [20], the general equation for a conic section is given by:

$$Ax^2 + Bxy + Cy^2 + Dx + EY + F = 0. \tag{4.90}$$

This equation is valid for the ellipse when $B^2 - 4AC < 0$, which applies for the points considered. Also, (x, y) is a set of points of Cartesian plane that satisfy the above expression in non-degenerate cases, which are the conditions. Coefficients presented in this expression are determined from parameters a, b, c and θ (output from the corrected function [17]), since the ellipse is not centered on the origin and has a rotation associated:

$$\begin{aligned}
&A = a^2 \times \sin^2\theta + b^2 \times \cos^2\theta, \quad B = 2 \times (b^2 - a^2) \times \sin\theta \times \cos\theta, \\
&C = a^2 \times \cos^2\theta + b^2 \times \sin^2\theta, \quad D = -2 \times A \times c_x - B \times c_y, \\
&E = -B \times c_x - 2 \times C \times c_y, \quad F = A \times c_x^2 + B \times c_x \times c_y + C \times c_y^2 - a^2 \times b^2,
\end{aligned}$$

where c is a vector (c_x, c_y) defining the center of the ellipse.
To determine the lines, the slopes and y-intercepts need to be calculated. Therefore, expression (4.90) was differentiated using implicit differentiation:

$$\begin{aligned}
&2Ax + Bx\frac{dy}{dx} + By + 2Cy\frac{dy}{dx} + D + E\frac{dy}{dx} = 0 \\
&\Leftrightarrow Bx\frac{dy}{dx} + 2Cy\frac{dy}{dx} = -2Ax - By - D - E\frac{dy}{dx} \\
&\Leftrightarrow \frac{dy}{dx}(Bx + 2Cy) = -2Ax - By - D - E\frac{dy}{dx} \\
&\Leftrightarrow \frac{dy}{dx} = \frac{-2Ax - By - D}{Bx + 2Cy + E} = m
\end{aligned} \tag{4.91}$$

Then, the vertices and co-vertices are determined: $(c_x \pm a \times cos(\theta), c_y \pm a \times sin(\theta))$ and $(c_x \pm b \times sin(\theta), c_y \mp a \times cos(\theta))$, yielding 4 points. Given these points, 4 lines with parameters m_i and b_i with $i = 1, \cdots, 4$ are computed:

$$\begin{aligned}
m_i &= \frac{-2Ax - By - D}{Bx + 2Cy + E}, \\
b_i &= y - m_i \times x_i,
\end{aligned} \tag{4.92}$$

Algorithm 3 Corrected version of function [17]

function MINVOLELLIPSE($P, tolerance$)
  $[d, N] \leftarrow$ size of P
  $Q = \mathbf{0}_{(d+1)\times N}$
  $Q(1:d,:) = P(1:d, 1:N)$
  $Q(d+1,:) = \mathbf{1}_{1\times N}$

  $count = 1$
  $err = 1$
  $u = (1/N) \times \mathbf{1}_{N\times 1}$
  Start counter
  while $err > tolerance$ **do**
    $X = Q \times diag(u) \times Q^T$
    $M = diag(Q^T \times pinv(X) \times Q)$
    $[m, j] \leftarrow$ max of M
    $step_{size} = \frac{m-d-1}{(d+1)\times(m-1)}$
    $new_u = (1 - step_{size}) \times u$
    $new_u(j) = new_u(j) + step_{size}$
    $count = count + 1$
    $err = ||new_u - u||$
    $u = new_u$
    $Time \leftarrow$ **Time elapsed**
    if $Time > 20$ and $err < 20$ or ($exp(P(1,:) == P(2,:))$ and $P(1,1) > 18$) **then**
      $break$
  $U = diag(u)$
  $A = (1/d) \times pinv(P \times U \times P^T - (P \times u) \times (P \times u)^T)$
  $c = P \times u$
  return A, c

where point (x, y) designates a pair of vertices, which are the ones mentioned above. After all the lines being determined, the ones tangent to the co-vertices are converted into inequalities.

Figure 4.15 showcases an example of the relevant lines when $lb_i = 1$ and $up_i = 3$ and the sigmoid as the activation function. Then, the only missing computation is finding which line is above and below points (x, y). For the example at hand, the inequalities would be $\mathbf{y}[i] \geq m_1\mathbf{x}[i] + b_1$ and $\mathbf{y}[i] \leq m_2\mathbf{x}[i] + b_2$, where the parameters of the lines are indicated in the graph.

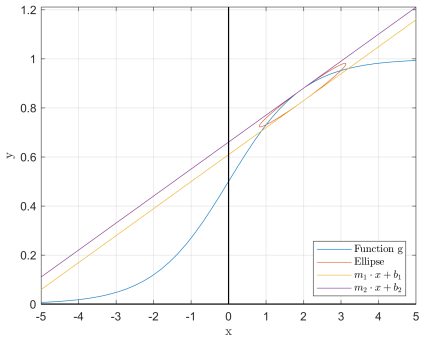


Figure 4.15: Plot of the sigmoid function as well as lines and ellipse used to define constraints.

4.5 Multiple Input Activation Functions: Softmax Example

In this last section, the rationale to pass a set through an softmax function is presented since it is different from the ones presented previously and is needed in Chapter 5 when obtaining simulation results with [13]. This procedure is transversal to CZs as well as CCGs. The expression for the softmax function is:

$$softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}},\ for\ i = 1, \cdots, K\ and\ \mathbf{x} = (x_1, \cdots, z_K) \in \mathbb{R}^K, \tag{4.93}$$

where K is the number of variables on the output layer.
Here the rationale is the following: applying the softmax function is equivalent to applying the exponential function to each variable and with lower and upper bounds, probabilities associated with each label are computed (which are the values that the softmax functions takes for each variable). If the set representation of choice is a CZ, then the constraints are linear inequalities which need to be converted in the form of a CZ. Else, if it is a CCG, the available constraints are linear and non-linear inequalities which also need to be converted. The necessary steps to apply these constraints have been presented and thoroughly explained throughout this chapter.

Chapter 5

Simulation Results

In this chapter, simulation results for several of the previously mentioned activation functions are shown to validate the proposed algorithms. Two examples will be provided: one more theoretical, employing a double integrator and the closed-loop system 3.5, and one more practical, utilizing the MNIST handwritten digit database. In the first scenario, generated synthetic data is used whereas for the second one, real-world examples are used. As an additional contribution, the approach used to sample a CZ is presented and explained.

For the first example, MPC controllers are used to generate data that are then implemented using YALMIP [14], resorted as language to model optimization problems. MATLAB was used to implement and train all neural networks as well as implement MPC controllers. For the first example, Mosek [15] was used as underlying solver for some optimization problems, previously indicated. In the second case, the solver used was Gurobi [18] since Mosek presented some limitations when computing bounds.

Simulations were run in Matlab 2019a running on a HP machine with a Intel Core i7-8550U CPU @ 1.80GHz and 8 GB of memory.

5.1 Neural Controller using Model Predictive Control Data

In this first case, it is recovered the first example presented in [2]. A double integrator system is considered, being defined by (3.63). This system was discretized with a sample time of $t_s = 1s$ and subject to state and control constraints namely $x_{k+1} \in [-5, 5]$ and $u_k \in [-1, 1]$, respectively. A standard linear MPC was used with a prediction horizon of $N_{MPC} = 10$ and weighting matrices $Q = I_2$, $R = 1$. This MPC is a stabilizing controller that returns the system back to the origin while meeting the limitations. The controller is then used to generate N samples of state and control pairs in order to train the NN. Here, the neural network contains four layers, each with ten neurons, with three different activation functions in the following order: ReLU, Sigmoid, Tanh and ReLU.

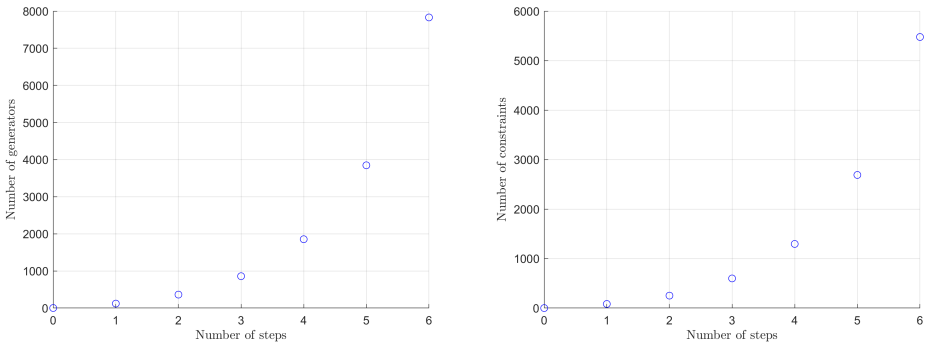
Figure 3.5 illustrates the closed-loop system used in this simulations, where P denotes the double integrator system, π represents the neural network described above and the remaining block is a projection block, guaranteeing that the output of this network complies with the input restrictions. The initial set is $X(0) = [3, 4] \times [0.5, 1.5]$.

Since the exact technique to compute sets has been abandoned, algorithm 2 is used at each layer of the neural network to provide an over-approximation of the output set, where these sets are $X(1), \cdots, X(K)$, where K are the number of steps considered. For both types of set representation, this value is equal to 6. $X(0)$ is also sampled having been considered 1000 points. These sampled points are then propagated through the closed-loop system to validate output reachable sets, allowing to validate the

proposed algorithms. If each set of points lie inside the set for each step K, the set is correct.
Below results using the two different types of set representation is presented, with the three most important considered factors to evaluate: size of matrices and vectors used in each set, the computational time to run the closed-loop system for K steps and volumes of each set for each set representation.

5.1.1 System State Reachability using Constrained Zonotopes

Figure 5.3 displays simulations results, where the closed-loop system described above is applied. For each subfigure, the red set represents each state, that is, $X(0), ..., X(6)$, while the blue dotted points indicate the sampled input set (for $K = 0$) and the outcome of propagating these points through the system. Although it may visually appear that some points lie outside each set, it does not happen.
Regarding these results, there are some observations which are not related to the correctness of the sets. To begin, it is possible to observe that the sets increase step by step. In each step, the new state is computed using a control state u_t (also represented by a CZ) and applied to the entire set, causing the sets to gradually increase in size. Also related to this observation, is the loss of precision with the steady rise of steps concluded through the dispersion of points which tends to increase. This occurs because raising the prediction horizon raises the uncertainty in each computed state.
Relatively to the size of matrices and vectors, Figure 5.1 depicts the evolution of the number of generators and constraints as function of the number of steps. It is plausible to deduce that this growth is exponential, which can be detrimental in terms of execution time when considering higher values for K.



(a) Number of generators. (b) Number of constraints.

Figure 5.1: Evolution of factors when using CZs to represent sets.

As initially stated, the execution time is not constant since the bounds of each set are computed by solving an optimization problem. For that, 30 tests were carried out to acquire a median value and a variance, yielding approximately 122.7338 ± 16.0768 seconds. The extreme values of this interval correspond to the minimum and maximum values of runtime. Figure 5.2 shows the evolution of the volume of each set $X(K)$ for each state k. To mention that for $K = 6$, the volume is maximum and equal to 171.4277.

5.1.2 System State Reachability using Constrained Convex Generators

Figure 5.6 depicts simulations results when the novel set representation, Constrained Convex Generators, is used for in closed-loop system 3.5. The details about each plot was previously detailed for CZs. Additionally, all the points propagated through the system are contained within the sets.
As with Constrained Zonotopes, increasing set size and corresponding loss of precision occur with Constrained Convex Generators for the same reasons. Figure 5.4 shows the growth of the number

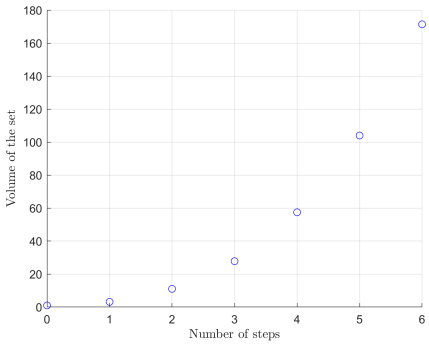
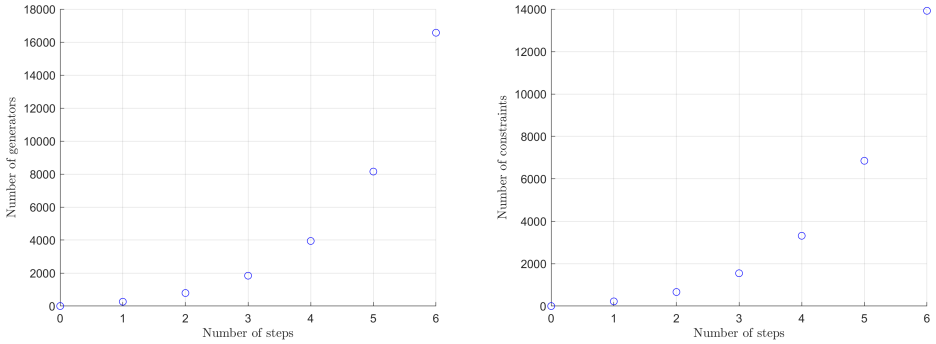


Figure 5.2: Evolution of volume when using CZs to represent sets.

of generators and constraints as a function of the number of steps in relation to the size of matrices and vectors. Since there are more sets involved in the projection of each variable, this growth is also exponential and faster than that observed for CZs, that is, there are more generators and constraints for each set.



(a) Number of generators. (b) Number of constraints.

Figure 5.4: Evolution of factors when using CCGs to represent sets.

To determine the computing time required to run the closed-loop system, 50 tests were done yielding approximately 554.5343 ± 40.2917 seconds. The extreme values of this interval correspond to the minimum and maximum values of runtime. Figure 5.5 depicts the evolution of the volume of each set $X(K)$, with $K = 6$ being the maximum volume and equal to 169.0546.

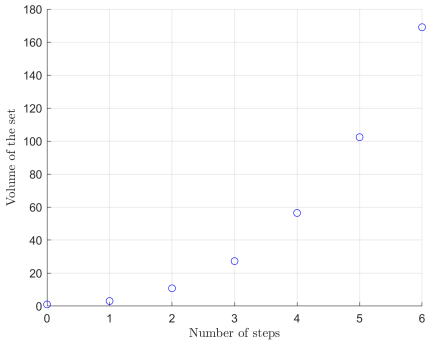
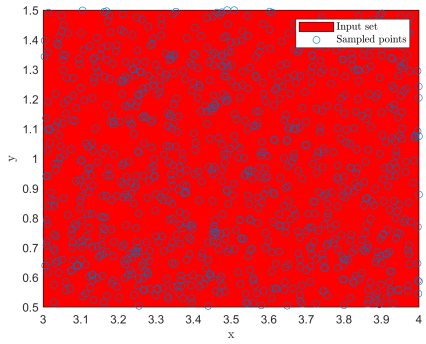
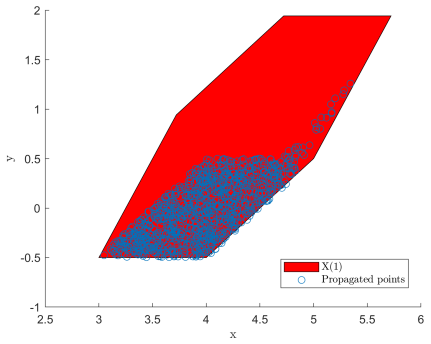


Figure 5.5: Evolution of volume when using CCGs to represent sets.

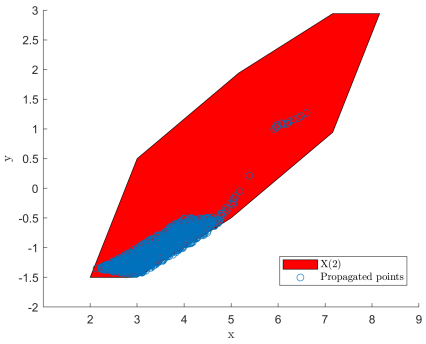
Table 5.1 compares the aforementioned factors for certain relevant values. Volume, number of generators and number of constraints presented belong to the state $X(K)$ where they are maximum, i.e., for



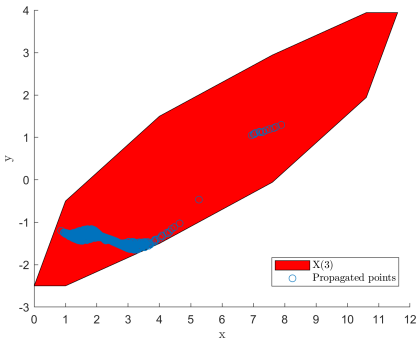
(a) Initial set and sampled points.



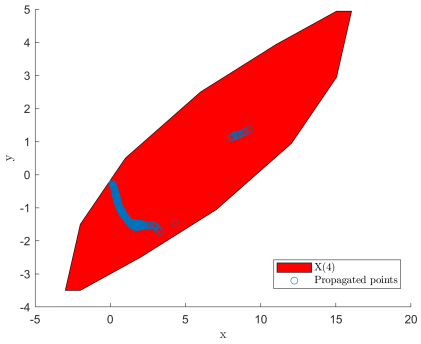
(b) Set for $K = 1$ and propagated points.



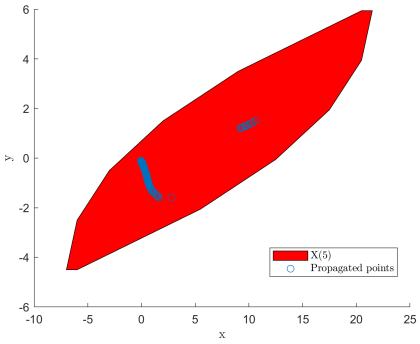
(c) Set for $K = 2$ and propagated points.



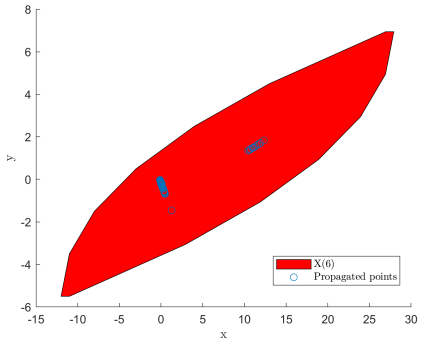
(d) Set for $K = 3$ and propagated points.



(e) Set for $K = 4$ and propagated points.



(f) Set for $K = 5$ and propagated points.



(g) Set for $K = 6$ and propagated points.

Figure 5.3: Sets obtained for each state with points sampled and propagated through the closed-loop system, using CZs for the set representation.

$K = 6$.

From Table 5.1, it is possible to observe that the runtime, number of generators and constraints are all

Set representation	Runtime (s)	Volume	No. generators	No. constraints
Constrained Zonotopes	122.7338 ± 16.0768	171.4277	7831	5478
Constrained Convex Generators	554.5343 ± 40.2917	169.0546	16571	13923

Table 5.1: Comparison of some factors using different types of set representation.

higher when using CCGs than when using CZs. This arises because the projection of each variable involves 3 sets and the operations performed ((2.7) and (2.9)) between these sets increase the size of matrices and vectors, that is, the number of generators and constraints. The runtime is also longer as a result of these factors. To mention that these higher values of runtime occurs for this example due to the amount of layers contained in the considered network. From Figures 5.3 and 5.6, it is plausible to state for the considered network and initial conditions, the closed-loop system is unstable.

As initially described, the employment of CCGs allows to resort to any unit ball following a p-norm instead of only a ℓ_∞ ball. As a result, the key advantage of utilizing CCGs is a lower volume in these type of sets, resulting in a better over-approximation. The following example will highlight other aspects of this set representation mainly, the ability of obtaining sets with lower volume within a approximate runtime.

It is also important to mention that the decrease of, approximately, 1.39% of the set represented by a CCG over a CZ can be differential when considering marginally stable or stable systems since it can impact the final conclusions to be made about the system regarding the stability.

One could say that the Monte-Carlo method would be enough to evaluate the stability of system. However, recurring to a CZ or a CCG allows stating certain facts regarding the evolution of the states with the increase of K, such as, when considering a CCG the state $x_t = [10, -5]^T$ is not reached for $K = 6$.

5.2 Noise Tolerance for a Classifier using the MNIST Dataset

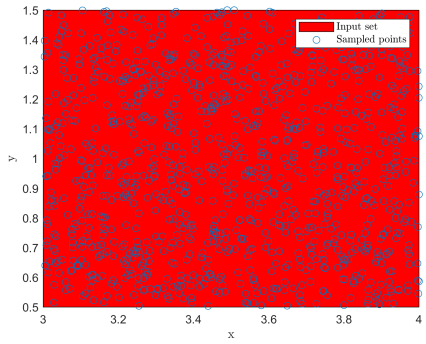
For the second example, the MNIST handwritten digit database [13] is used, where digits ranging from 0 to 9 are considered. Figure 5.7 presents 25 examples of these digits with their respective label.

In this subsection, two type of experiments will be carried out: comparing the performance between the function *net* (available in MATLAB's Deep Learning Toolbox) and the algorithms using each set representations (presented in the previous chapter) and comparing set's volume when using different set representations. The first one demonstrates the capability of obtaining correct predictions while utilizing both set representations and also having a wider range of images being tested as input. The latter one validates the conjecture made, that is, with CCGs sets have a more precise over-approximation. With this final test, the main goal of this work is presented: the usage of CCGs.

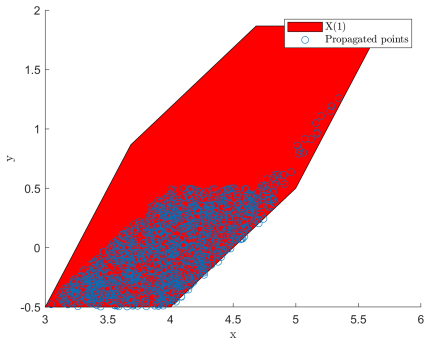
The first step is to train a NN. The network here used has two layers: the first layer has 100 neurons and the activation function is the Tanh while the second layer has 10 neurons and the activation function is the softmax. Other variations of this neural networks were tested including with different activation functions and number of neurons, performing worse than the above network.

The network was trained using a dataset of 28000 examples, with each input sample being a vector of dimension 784, where each values varies from 0 to 255. This is due to the fact that each digit is represented by a 28×28 image. An output example is represented by a vector of dimension 10, being equal to 1 in the position matching the number and the rest is equal to 0. The reasoning used to represent each input sample with a CZ or CCG is not trivial and, therefore, it will be detailed next. Furthermore, the Tanh and softmax activation functions are applied to the input. Also, the rationale used to apply the latter was explained in Section 4.5.

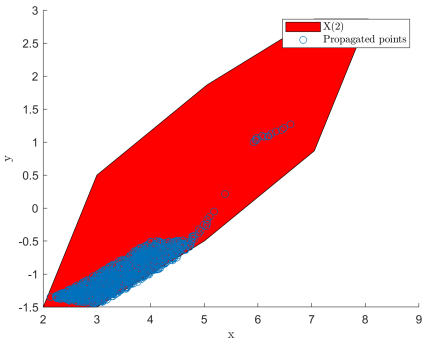
For both types of representation, vector c will be the vector representing the input and matrix G is the noise associated with each pixel in the image. To note that the notion of noise in each pixel is perceived



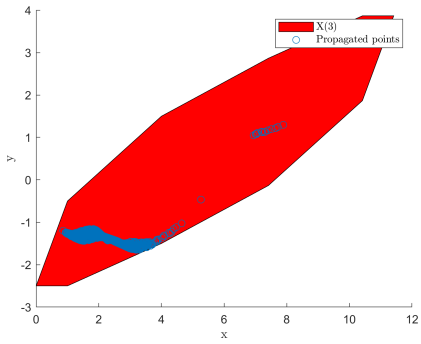
(a) Initial set and sampled points.



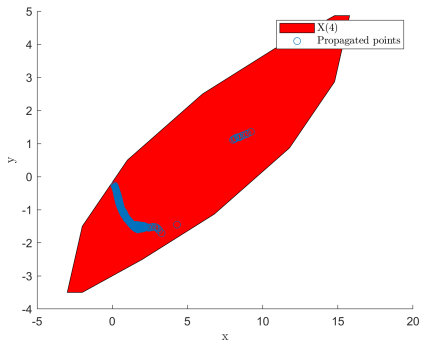
(b) Set for $K = 1$ and propagated points.



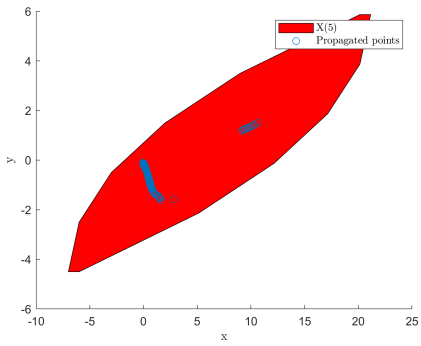
(c) Set for $K = 2$ and propagated points.



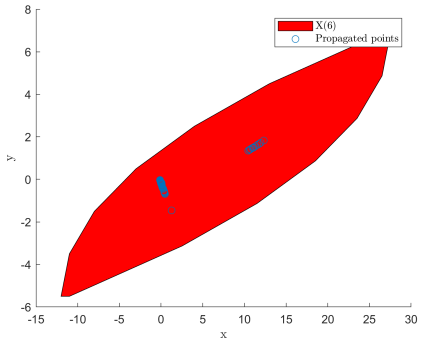
(d) Set for $K = 3$ and propagated points.



(e) Set for $K = 4$ and propagated points.



(f) Set for $K = 5$ and propagated points.



(g) Set for $K = 6$ and propagated points.

Figure 5.6: Sets obtained for each state with points sampled and propagated through the closed-loop system, using CCGs for the set representation.

as an uncertainty, quantifying how distant it is from its original value of the pixel. Therefore, each variable has a interval associated.

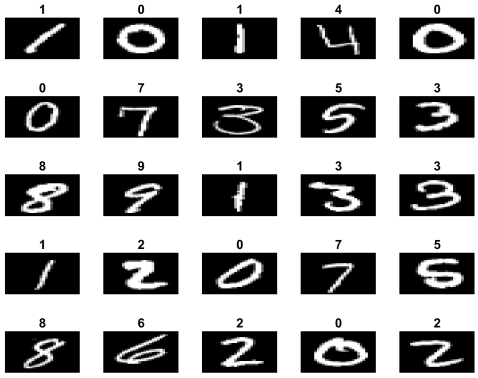


Figure 5.7: Some digits available on the MNIST database.

In addition, matrix A and b are empty since there is not any constraint on the pixels. As it is not possible to represent points, the sets must be defined in such a way. Otherwise, vector c would be equal to an output example (vector of dimension 10) and matrix G equal to a matrix of zeros. Considering the one-dimensional case, c would define a point and the value G influences the length of an interval, where c is the central point.

To clarify how to convert an input sample to one of the set representation, the first step is to consider X as an input sample and $input$ as the neural network's input set, X can be written in either as an CZ and an CCG. The first set of equations below defines an CZ for the $input$, whereas the second one defines an CCG. In both matrices G, value n represents the noise considered, where these matrices are square with a dimension of 784.

$$G_{input} = \begin{bmatrix} n & 0 & 0 & \cdots & 0 \\ 0 & n & 0 & \cdots & 0 \\ 0 & 0 & n & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & n \end{bmatrix} \tag{5.1}$$

$$c_{input} = X \tag{5.2}$$

$$A_{input} = \mathbf{0}_{0\times 784} \tag{5.3}$$

$$b_{input} = \mathbf{0}_{0\times 1} \tag{5.4}$$

$$G_{input} = \begin{bmatrix} n & 0 & 0 & \cdots & 0 \\ 0 & n & 0 & \cdots & 0 \\ 0 & 0 & n & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & n \end{bmatrix} \tag{5.5}$$

$$c_{input} = X \tag{5.6}$$

$$A_{input} = \mathbf{0}_{0\times 784} \tag{5.7}$$

$$b_{input} = \mathbf{0}_{0\times 1} \tag{5.8}$$

$$\mathfrak{C}_{input} = \mathcal{B}_2 \tag{5.9}$$

To pass an input set through layers whose activation functions are a Tanh, a sigmoid or any other function detailed here is trivial since it only requires applying the algorithms previously described. As a result, in this example, since the first layer's activation function is a Tanh, algorithm 1 or 2 is applied, depending on the set representation. The approach to pass a set through the second activation function is different and explained in section 4.5.

The following subsections will present simulation findings, using different approaches, comparing the different set representations using a test dataset.

5.2.1 Monte-Carlo Sampling versus CZs and CCGs

As initially stated, the first approach entails proving the ability of making correct predictions when using these representations, compared to the *net* function.

In an optimal situation, where the focus is to evaluate a noiseless case, the noise has to be close to 0. Since the objective is to assess the performance of the algorithms, various levels of noise are considered. The values considered for n were: 0.001, 0.01, 0.1, 1 and 10, where the optimum scenario (i.e. noiseless case) occurs when $n = 0.001$.

To showcase the effect of each value of noise in a digit, some figures are presented, exhibiting one digit with and without noise to demonstrate the influence of each value of noise. Figure 5.8 displays digit 3 without noise and a colorbar showing the color scale (0 to 255).

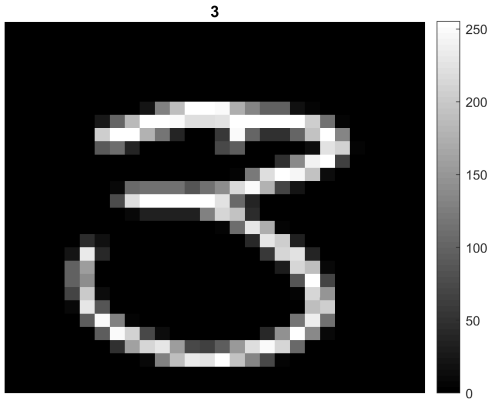


Figure 5.8: Digit 3 when $n = 0$.

Figure 5.9 displays the original digit 3 when noise is added but with two differences: the first one is obtained when is subtracted the value of n for to each pixel, whereas the second corresponds to adding the value. For both examples, $n = 100$.

It is important to remark that both these figures do not exactly correspond to what is being inputted but rather reflect maximum and minimum values that each pixel can take. This means that each pixel can take a certain color within a certain interval: the greater the value of n, greater the interval of colors and therefore, more uncertainties associated with each pixel.

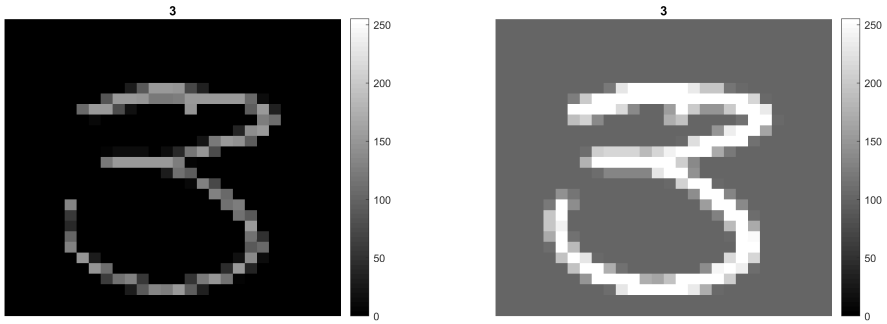
As a result, when adopting these set representations, an infinite amount of images are being passed through the network. Due to the large number amount of images being tested, this can also be seen as an advantage when compared to the function *net*.

Table 5.2 presents the accuracy of both algorithms (1 or 2) when using CZs or CCGs, for different levels of noise n.

	Level of noise introduced in each pixel				
Set representation	0.001	0.01	0.1	1	10
Constrained Zonotopes	95.80%	95.80%	95.60%	95.80%	71.27%
Constrained Convex Generators	95.07%	95.87%	95.53%	95.73%	71.77%

Table 5.2: Accuracy of the algorithms for different levels of noise n.

Another simulation result to be added to the above table is the accuracy value, when using the function available in MATLAB to make predictions with NN. The **value obtained** was, approximately, $\mathbf{95.83}\%$.



(a) Result of subtracting n.

(b) Result of adding n.

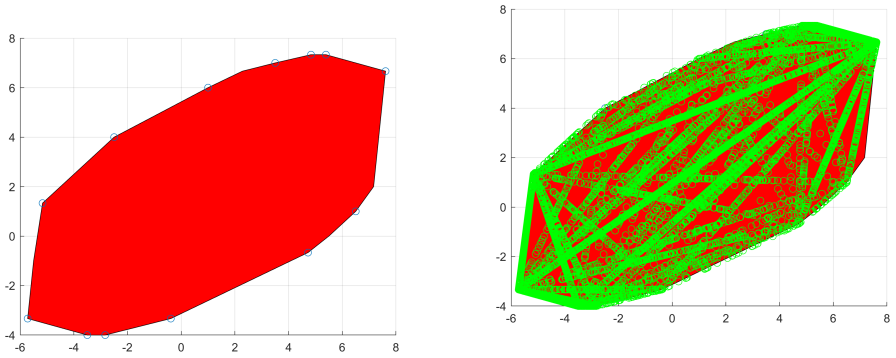
Figure 5.9: Result of subtracting and adding $n = 100$ from each pixel value for digit 3.

The first observation to be made is that, when equivalent conditions (i.e. for smaller values of n) are considered, the network gives predictions with a success rate that do not differ from the ones obtained using set representations. Also, to mention that on average, function *net* takes 7.32 seconds to run whereas using a CZ representation takes 12.77 seconds and 10.37 for a CCG representation.
Another point to mention is that for lower noise values, both algorithms allow to obtain similar accuracies and thus, increasing the value of n results in a decrease of accuracy and vice-versa when decreasing noise. This occurs since adding noise has the effect of increasing the intervals for each variable, that is, it increases the difference between bounds, resulting in higher volume sets. When compared to smaller sets, larger sets (in terms of volume) have a higher degree of uncertainty and consequently have more inaccurate predictions associated to them.
In Section 5.1, it was possible to check that with this novel set representation, sets had a lower volume when compared to those represented using CZs, which entails one of the benefits. Facing these results, it is not possible to immediately state that fact which is one of the main objectives of this work.
The following subsection presents some simulation findings supporting the preceding assertion.

5.2.2 Volume Comparison for Reachable Sets using CZs and CCGs

This subsection begins by providing a brief description of how to determine, indirectly, the volume of sets when using both set representations. Then, the results are presented and discussed.
Firstly, due to the dimensions of the matrices and vectors characterizing the sets, the toolbox function to compute the volume is not useful. Therefore, an alternative technique is developed and used: the set represented by a CZ, denoted as X, is sampled and the intersection between this set and the set represented by a CCG, denoted as Y, is computed using the sampled points. The amount of points sampled which are inside set Y allows to indirectly compare the volumes between each set.
The initial step is to determine the points lying on the boundary of set X, using an optimization problem. To mention that the sets to be sampled correspond to the output of the network's first layer. Since the final operation being performed on the network, which is passing from 100 to 10 neurons, entails tightening the intervals for each variable. As a result, sampling sets in the final layer would produce misleading results, since they do not capture the true value for the boundaries.
For this optimization problem, the constraint is defined by the LMI F_X (Yalmip constraint set), the variable to be minimized p_X (*sdpvar* representing a point in set X) and the objective to be maximized is the product of an arbitrary direction v and p_X once the goal is to determine points lying on the limit of the set. The number of times this problem is solved is determined by number of points desired in the set's border. The previous described procedure is known as *ray-shooting*.

The second step is to apply the convex combination between pairs of vertices, determined in the previous step, whose result are points inside set X.
Following that, an optimization problem is solved to determine whether each point is within the set Y, where the constraints are F_Y and an equality between each point and p_Y. Also, the objective function is zero since the goal is to verify whether the point is inside set Y. As a final note, it is also important to mention that all the sets being dealt with are convex and therefore, the points resulting from the convex combination will be within set X.
Figure 5.10a depicts the points (in blue) lying on the border of the set represented by a CZ and Figure 5.10b the points (in green) resulting from the convex combination.



(a) Points obtained on the border of the set.

(b) Points resulting from the convex combination.

Figure 5.10: Illustration of points obtained when sampling the set represented by a CZ.

Algorithm 4 presents the function used with all the steps described above used to sample a CZ.

Algorithm 4 Function used to sample a CZ

function SAMPLECZ($F, p, n_{samples}, n_{combinations}$)
    initialize vector $vertices$
    for *i in* $n_{samples}$ **do**
        optimize($F, (-1 + 2 \times rand(2,1))^T \cdot p$)
        $vertices = [vertices, value(p)]$

    $n \leftarrow$ number of vertices
    $idx_{pairs} \leftarrow$ unique pairs of indices considering n
    Initialize vector $points$
    for *i in number of unique pairs* **do**
        $weights = rand(n_{combinations}, 2)$
        Normalize $weights$
        for *j in* $n_{combinations}$ **do**
            $point = vertices(:, idx_{pairs}(i,1)) \cdot weights(j,1) + vertices(:, idx_{pairs}(i,2)) \cdot weights(j,2)$
            $points = [points, point]$
    return $points$

The number of points inside set X that are also within set Y are used to compute the ratio. Table 5.3 presents values obtained for the ratio, considering different levels of noise.

	Level of noise introduced in each pixel				
	0.001	0.01	0.1	1	10
Ratio	0.0083%	37.6977%	99.7764%	98.9234%	98.2609%

Table 5.3: Ratio of points inside set X contained in set Y.

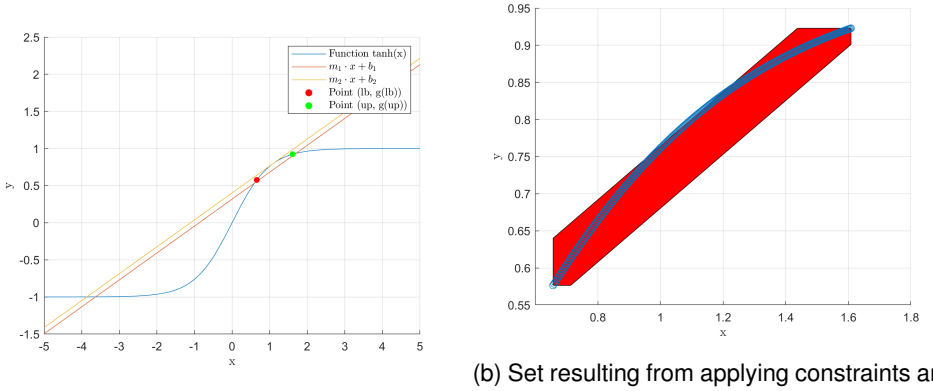
The above table emphasises one the advantages of a CCG, the ability to represent over-approximated sets with a lower volume. Considering the lowest value for n, set Y is 0.0083% of set X meaning that the set represented by a CCG is very small when compared to the set represented by a CZ and, for higher values of n, the sets match almost completely.

The previous subsection could induce that, at low noise levels, sets X and Y would have approximate volumes. The results in this section demonstrate the exactly opposite, and it is also possible to check that, at high noise levels, both sets present similar volumes and predictions, which was the expected result.

This behaviour occurs for the following reasoning: for lower values of noise, the difference between the lower and upper bound is small (when compared to higher values of noise) which entails smaller sets. Here, CCG have an advantage since they can resort to any unit ball following a p-norm instead of only a ℓ_∞ ball and therefore, sets are better over-approximated. To display this reasoning, plots are presented below.

In order to show clearly the plots, these were generated considering $n = 1$. Figures 5.11a and 5.12a display the lines determined to a single variable (of the output set of the first layer) when using the different set representations as well as the points $(lb, g(lb))$ and $(up, g(up))$, where g is the Tanh function. In these figures, lines $m_1 \cdot x + b_1$ and $m_2 \cdot x + b_2$ are used to find the needed inequalities as mentioned in Section 4.

Figures 5.11b and 5.12b present the set resulting from intersecting the constraints (after converting each equality into a inequality), as well as a scatter of points generated based on the value of the bounds.



(a) Lines determined for the Tanh function.

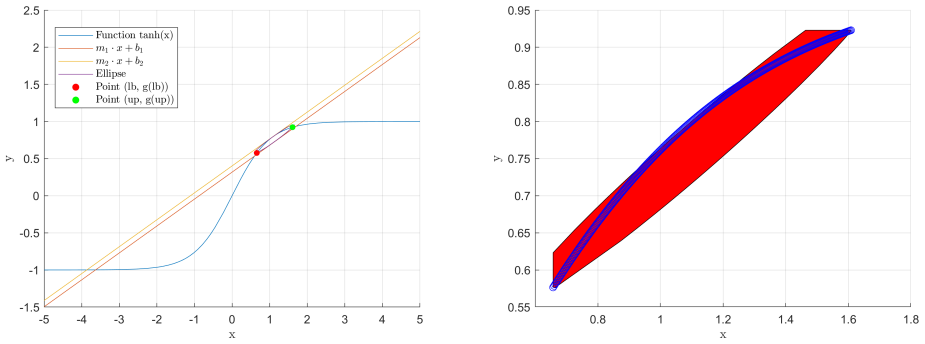
(b) Set resulting from applying constraints and scatter of points.

Figure 5.11: Lines determined and resulting set, when using a CZ set representation.

These first two pairs of plots demonstrate how the set resulting from each intersection is obtained. Figure 5.13 gathers both sets and presents a plot of the overlapping between them as well as a scatter of points. In this figure, the red set is a result from using a CZ and the blue one from using a CCG. Furthermore, it is possible to observe the difference between using a ℓ_∞ ball and a ℓ_2 ball since the blue set is smaller than the red one. To display these differences, another example is considered using two different values for n: 0.01, 1 and 10. Figure 5.14 display three plots of overlapping sets obtained using the two different set representations.

Again, the red and blue set are obtained using a CZ and a CCG, respectively. While in the first one only resorts to a ℓ_∞ ball, the second one can resort to a ℓ_∞ ball and (at the same time) a ℓ_2 ball. Also, the ratio is equal to 38.3023%, 55.1222% and 70.4596% in the first, second and third plot, respectively.

This final series of graphs demonstrates the benefits of using a CCG since it allows to resort to unit balls following different norms and hence, permits more tight approximations when compared to CZs, a more conservative representation. For higher values of noise, if both sets are large (in terms of volume), a



(a) Lines and ellipse determined for the Tanh function.

(b) Set resulting from applying constraints and scatter of points.

Figure 5.12: Lines and ellipse determined and resulting set, when using a CCG set representation.

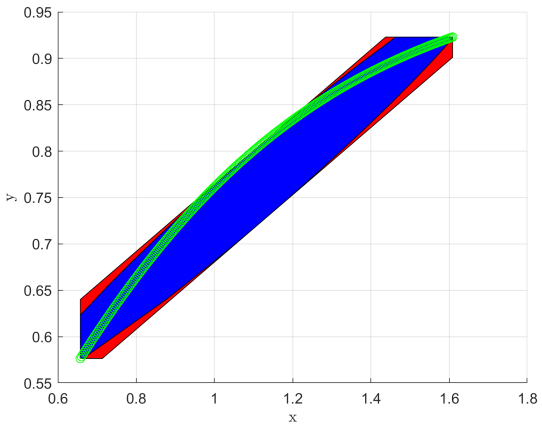
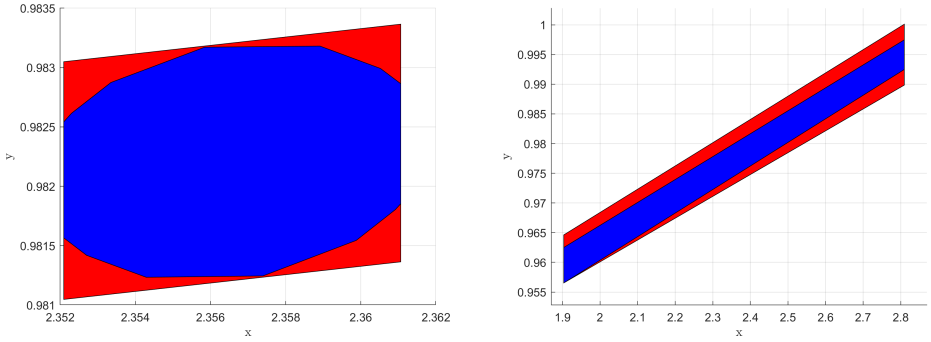


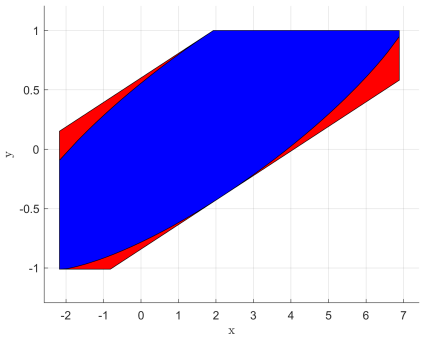
Figure 5.13: Overlap of sets when using different types of set representation: CZ (red) and CCG (blue).

minor decreasing of volume of the set being represented by a CCG is not noticeable, thus resulting in a high ratio. For lower values of noise, the opposite occurs: a slight difference between the size of each set is perceptible, resulting in a low ratio.



(a) $n = 0.01$.

(b) $n = 1$.



(c) $n = 10$.

Figure 5.14: Overlapping of sets for different values of n, when using different set representations: CZ (red) and CCG (blue).

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Throughout this work, the focus of was on the problem of verifying robustness of a NN, through the computation of reachable sets. To better understand all concepts, an brief introduction to this problem is done, where the main topics related to this are addressed and the problem is formulated. After that, an extensive analysis and discussion of the current state-of-art reachability methods is done, followed by results presenting both advantages and disadvantages of these.

The implementation starts by considering an existing algorithm for the ReLU activation function, that permits to obtain exact reachable sets, presented in [19]. However, as it was proven the number of sets would increase exponentially with number of layers and neurons of a certain NN, leading to computationally intractable problem. Therefore, the sole focus of this work was through over-approximated sets.

After presenting this initial algorithm, a second one which over-approximates sets is explained how it was discovered. For this function, inequalities are derived from a graphical representation of the function at hand and all the needed steps to obtain the matrices and vectors that parameterize a set represented by a CZ are presented. Furthermore, improvements to this algorithm are highlighted allowing for lower-dimensional matrices and vectors, which has influence when considering deeper and wider NNs.

This initial step is key to the development of the remaining work since understanding this is the starting point to derive algorithms for any other activation function. Although the focus of this work is the use of the novel set representation to evaluate the robustness of a network, algorithms using CZs are derived, explained and presented to showcase the advantages that the new set representation evokes. Additionally, some methodologies to determine constraints are presented: an efficient and approximated (using an ellipse) as well as a time-consuming and exact (solving an optimization problem), allowing a wider range of choices.

Then, since there are not any current state-of-art algorithms explaining on how to propagate sets through activation functions using CCGs, an extensive description and explanation for the hyperbolic tangent function is done. The approach to obtain the matrices and vectors for each CCG is different from CZs since it used three sets to obtain the over-approximated one. This reasoning is replicated for a series of other functions.

In this novel set representation, generator variables are constrained to some convex sets, meaning that sets with abnormal shapes can be better represented. As a consequence, this method maintains efficiency and performance while keeping sets tight.

The last part of this work has as main objective of proving the usage of CCGs over CZs. The theoretical

case is initially used to prove that with the first set representation grants lower volume sets due to the possibility of resorting to any unit ball following a p-norm. However, due to the number of generators and constraints involved in linear mapping and intersection operations, it was demonstrated that it is more time demanding. To mention that in this example, the network is deeper when compared to the second example.
The second example compares both types of set representation through a real-life example (MNIST dataset) with the aim of proving that the novel set representation provides smaller sets with a reduced runtime, permitting to derive better over-approximated reachable sets. Furthermore, it provides a detailed method that allows to sample any CZ.
Since the majority of the constraints used are determined using an ellipse, Chapter 4 explains some limitations of function [17] and suggests a correction, where the rationale behind each suggested correction is presented. Additionally, the logic behind the computation of the inequalities using the ellipse is showcased. As seen in Chapter 5, these rectifications allow to obtain correct results as well as determining correct constraints.
All the approaches to over-approximate output reachable sets have the same intent: verifying the robustness of a NN. Allowing to determine better over-approximations has as consequence better evaluations of robustness of a certain network and that can be accomplished with the use of a CCG. As mentioned in the introduction, nowadays NNs are being used in many important and impactful tasks and assuring that these are precise is key.

6.2 Future Work

There are several directions that may emerge from this work with the following importance (higher to lower):

- When using CCGs to represent sets, the approach used where three sets are considered to obtain a single one can be changed. As it was stated, it was time-consuming and hence, reducing from three sets to two or one would make this approach the preferable one due to the advantages associated. The solution is to manipulate directly matrices and vectors;
- As seen in the first set of results (closed-loop system 3.5), the sets increase with each step K. Developing an algorithm that could reduce the volume of each set would reduce computational time and obtain better over-approximated sets. This could be achievable by analysing constraints and generators for each set.
- Testing the presented methodologies and algorithms in real-world networks. This would validate them in real problems where real-life obstacles occur;
- Developing other algorithms to obtain inequalities. The methods here presented are approximated and thus, using more exact and fast methods can exponentiate even more the results obtained. A solution can be to use efficient optimization problems or looking to use state-of-art methods.
- Resorting to other norms to represent sets, since the algorithms here presented for the new set representation used only ℓ_∞ and ℓ_2 balls. The necessity of using of other norms would be related to the function at hand;

Bibliography

[1] C. Liu, T. Arnon, C. Lazarus, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *CoRR*, vol. abs/1903.06758, 2019.

[2] H. Hu, M. Fazlyab, M. Morari, and G. J. Pappas, "Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming," in *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 5929–5934, 2020.

[3] K. Jules and P. P. Lin, "Artificial neural networks applications: from aircraft design optimization to orbiting spacecraft on-board environment monitoring," *2001 Advanced Study Institute on Neural Networks for Instrumentation, Measurement and Related Industrial Applications*, no. NASA/TM-2002-211811, 2002.

[4] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz, "Constrained zonotopes: A new tool for set-based estimation and fault detection," *Automatica*, vol. 69, pp. 126–136, 2016.

[5] D. Silvestre, "Accurate guaranteed state estimation for uncertain lpvs using constrained convex generators," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 4957–4962, 2022.

[6] W. Xiang, H. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with relu activations," *CoRR*, vol. abs/1712.08163, 2017.

[7] Gehr, Timon and Mirman, Matthew and Drachsler-Cohen, Dana and Tsankov, Petar and Chaudhuri, Swarat and Vechev, Martin, "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, 2018.

[8] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[9] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," *CoRR*, vol. abs/1804.10829, 2018.

[10] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon, "Towards fast computation of certified robustness for relu networks," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 5273–5282, PMLR, 2018.

[11] Singh, Gagandeep and Gehr, Timon and Püschel, Markus and Vechev, Martin, "An Abstract Domain for Certifying Neural Networks," *Proc. ACM Program. Lang.*, vol. 3, jan 2019.

[12] Vandenberghe, Lieven and Boyd, Stephen, "Semidefinite Programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[13] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[14] J. Lofberg, "Yalmip : a toolbox for modeling and optimization in matlab," *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pp. 284–289, 2004.

[15] M. ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 10.0.*, 2022.

[16] D. Silvestre, "Constrained convex generators: A tool suitable for set-based estimation with range and bearing measurements," *IEEE Control Systems Letters*, vol. 6, pp. 1610–1615, 2022.

[17] Moshtagh, "Minimum volume enclosing ellipsoid." MATLAB Central File Exchange, 2009.

[18] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023.

[19] Y. Zhang and X. Xu, "Safety verification of neural feedback systems based on constrained zonotopes," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 2737–2744, 2022.

[20] E. W. Weisstein, "Conic section," *https://mathworld. wolfram. com/*, 2003.