Analysis of Robust Neural Networks for Control



Matthew Newton
Worcester College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2023

To my grandparents

Before you know it ...

Acknowledgements

First and foremost, my biggest thank you goes to my supervisor Professor Antonis Papachristodoulou for his continued support throughout my PhD. I am incredibly grateful for all of his patience, advice and efforts. Some people say that your PhD supervisor is either too busy to help you or too controlling and restrictive around your project. Antonis has been neither of these, going above and beyond to provide me with support when I have needed it, whilst also allowing me the freedom to explore the research areas that I have wanted to, including a change in topic one year in. I am very fortunate to have had you as a mentor and I will look back fondly on our many discussions.

Often administrative tasks for PhDs students can be time-consuming, frustrating and stressful. However, I have been lucky enough to have had Wendy Poole as the AIMS CDT course administrator, whom I would like to greatly thank. She has been wonderfully helpful throughout my studies, responding to emails faster than I could read them, organising engaging events and fun socials and of course, being a friendly face to talk to. I am also grateful to Professor Maryam Kamgarpour for giving me the opportunity to visit the Sycamore research group at the EPFL. I thoroughly enjoyed the visit and it was greatly beneficial to discuss research directions and learn about the exciting research in the group. Thank you too to Professor Yang Zheng for the many interesting discussions at the start of my PhD; these chats really got my research off the ground. I would like to thank Professor Stephen Duncan and Professor Matthew Turner for examining my thesis and for providing constructive feedback; the quality of my thesis improved following these amendments.

More generally, I am very grateful to EPSRC for funding my PhD, without this funding I would not have been able to study here. Additionally, I would like to thank all of the amazing staff at Worcester College for being helpful, friendly and for providing delicious and affordable food - sometimes it's the little things that make the difference.

I would like to thank all members of the Oxford University Control Group with whom I have spent time and learnt from. We are fortunate to have a supportive group of individuals. Special mentions to Yana for all of the great lunchtime chats and to Martin, Manos and Han for an enjoyable time spent together at CDC. I would also like to thank my fellow AIMS CDT students with whom I have enjoyed many memorable moments.

A miscellaneous thank you to my secondary school maths teacher, who told me I wasn't good enough to take further maths A-level. I never wanted to prove you wrong, I just wanted to prove myself right. That being said, I was extremely fortunate to have many amazing school teachers; a special thank you to Dr Kyle, looking back you were enormously helpful in my academic career.

A shout out to the friends that I have made during my time in Oxford. Most notably to my housemates, Sandy, James and Romain for all the competitive games nights, hilarious moments and for testing my cleaning skills. Thank you also to the friends that I have

made through OUAC - I'll never forget our clean sweep against Cambridge three years in a row and of course the painful lactic sessions.

Most importantly, thank you Jana for all your love and support. You are the loveliest person and a great companion - my time in Oxford would not have been the same without you. Ohne Jana ist alles doof.

Thank you to all my family, for showing love and support through the good and the bad times. Special thank you to Auntie Kate for her amazing cakes and for being a fountain of knowledge on almost any subject. Thank you to my sister Lucy for always one-upping me and keeping the family home lively. Thank you to my parents for always believing in me. Mum and Dad, I have always looked up to you and I am grateful every day for having you there.

Finally, I want to thank my grandparents to whom I dedicate this thesis. Maureen, you opened my eyes to many interesting ideas and taught me to see the world from everyone's perspective. Ray, you were a great role model and it was always nice to have someone who shared my love for numbers. Renate, you helped shape who I am today, du warst die beste Oma.

Abstract

The prevalence of neural networks in many application areas is expanding at an increasing rate, with the potential to provide huge benefits across numerous sectors. However, one of the greatest shortcomings of a trained neural network is its sensitivity to adversarial attacks. It is becoming clear that providing robust guarantees on systems that use neural networks is very important, especially in safety-critical applications. However, quantifying their safety and robustness properties has proven challenging due to the non-linearities of the activation functions inside the neural network.

This thesis addresses this problem from many different perspectives. Firstly, we investigate the sparsity that arises in a recently proposed semidefinite programming framework to verify a fully connected feed-forward neural network. We reformulate and exploit the sparsity in the optimisation problem, showing a significant speed-up in computation. In addition, we approach the problem using polynomial optimisation and show that by using the Positivstellensatz, bounds on the robustness guarantees can be tightened significantly over other popular methods. We then reformulate this approach to simultaneously exploit the sparsity in the problem, whilst improving the accuracy.

Neural networks have also seen a recent increased use in feedback control systems. This is primarily because they have the potential to improve the performance of these systems compared to traditional controllers, due to their ability to act as general function approximators. However, since feedback systems are usually subject to external perturbations and neural networks are sensitive to small changes, providing robustness guarantees has proven challenging.

In this thesis, we analyse non-linear systems that contain neural network controllers. We first address this problem by computing outer-approximations of the reachable sets using sparse polynomial optimisation. We then use a Sum of Squares programming framework to compute the stability of these systems. Both of these approaches provide better robustness guarantees over existing methods. Finally, we extend these approaches to neural network controllers with rational activation functions. We then propose a method to recover a stabilising controller from a Sum of Squares program and apply it to a modified rational neural network controller.

Contents

Abstract vii

Contents xii

List of Figures xiii

List of Tables xvii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Contribution to Literature . . . 5
- 1.3 Thesis Outline . . . 9

2 Background 11
- 2.1 Mathematical Preliminaries . . . 11
  - 2.1.1 Semidefinite Programming . . . 11
  - 2.1.2 Sum of Squares Programming . . . 12
  - 2.1.3 The Positivstellensatz . . . 16
  - 2.1.4 Sum of Squares and The Positivstellensatz . . . 18
  - 2.1.5 Chordal Graphs and Sparse Matrix Decomposition . . . 19
- 2.2 Neural Network Verification . . . 21
  - 2.2.1 Neural Network Verification Problem . . . 22
  - 2.2.2 Neural Network Model . . . 23
  - 2.2.3 Previous Literature on Neural Network Verification . . . 23
- 2.3 Analysing Neural Feedback Loops . . . 26
  - 2.3.1 Neural Feedback Loop Model . . . 27
  - 2.3.2 Previous Literature on Analysing Neural Feedback Loops . . . 28
- 2.4 Abstracting Neural Networks as Constraints . . . 30
  - 2.4.1 Input Constraints . . . 31
  - 2.4.2 Output Constraints . . . 31
  - 2.4.3 Hidden Layer Constraints . . . 31
  - 2.4.4 General Activation Functions in Neural Networks . . . 37

3 Exploiting Sparsity for Neural Network Verification 39
- 3.1 Introduction . . . 39
  - 3.1.1 Neural Network Equations in Matrix Form . . . 40
- 3.2 Neural Network Verification as Quadratic Constraints . . . 40
  - 3.2.1 Input Constraints . . . 41

3.2.2 Output Constraints . . . 42
3.2.3 Hidden Layer Constraints . . . 42
3.3 Neural Network Verification as a Semidefinite Program . . . 44
3.4 Chordal Analysis of Constraints . . . 44
3.4.1 Input Constraint . . . 45
3.4.2 Output Constraint . . . 45
3.4.3 Middle Constraint . . . 45
3.4.4 Neural Network Sparsity Examples . . . 47
3.5 Numerical Results . . . 48
3.5.1 Scalability with Varying Hidden Layers . . . 50
3.5.2 Scalability with Varying Hidden Units . . . 51
3.6 Conclusion . . . 51

4 Neural Network Verification using Polynomial Optimisation 53
4.1 Introduction . . . 53
4.2 Neural Network Verification Emptiness Condition . . . 54
4.3 Improved Sector Bounds . . . 55
4.3.1 Sigmoid Activation Function . . . 55
4.3.2 Tanh Activation Function . . . 57
4.4 Numerical Results . . . 58
4.4.1 ReLU Activation Function Example . . . 59
4.4.2 Sigmoid and Tanh Activation Function Examples . . . 60
4.4.3 Two Dimensional Input and Output Example . . . 61
4.5 Conclusion . . . 61

5 Neural Network Verification using Sparse Polynomial Optimisation 63
5.1 Introduction . . . 63
5.2 Sparse Polynomial Optimisation . . . 64
5.2.1 Sparse Sum of Squares Representations . . . 64
5.2.2 Term Sparsity . . . 66
5.2.3 Correlative Sparsity . . . 67
5.2.4 Similar Hierarchies . . . 69
5.2.5 Extension to Semi-algebraic Sets . . . 70
5.3 Sparse Neural Network Constraints . . . 71
5.3.1 Example: Three Layer, Single Node Neural Network . . . 72
5.3.2 General Neural Network . . . 74
5.4 Numerical Results . . . 76
5.4.1 Scalability Comparison . . . 77
5.4.2 Accuracy Comparison . . . 78
5.5 Conclusion . . . 79

6 Reachability Analysis of Neural Feedback Loops 85
6.1 Introduction . . . 85
6.2 Neural Feedback Loops in Discrete Time . . . 86
6.3 Reachability Problem . . . 87
6.4 Neural Feedback Loop Constraints . . . 88
6.5 Neural Feedback Loop Reachability Emptiness Condition . . . 89
6.6 Sparse Reachability Constraints . . . 90
6.7 Numerical Results . . . 90

6.7.1 Duffing Oscillator . . . 91
6.7.2 Unstable Polynomial System . . . 92
6.7.3 Inverted Pendulum . . . 93
6.8 Conclusion . . . 95

7 Stability Analysis of Neural Feedback Loops 97
7.1 Introduction . . . 97
7.2 Stability Theory . . . 98
7.2.1 Linear Stability . . . 98
7.2.2 Lyapunov Stability . . . 99
7.2.3 Region of Attraction . . . 100
7.2.4 Lyapunov Functions and Sum of Squares . . . 101
7.3 Neural Feedback Loops in Continuous Time . . . 102
7.4 Stability of Neural Feedback Loops . . . 103
7.4.1 Computing the Stability and Region of Attraction . . . 105
7.4.2 Extension to Robustness Analysis . . . 106
7.5 Numerical Examples . . . 106
7.5.1 Duffing Oscillator . . . 107
7.5.2 Three Dimensional Polynomial System . . . 107
7.5.3 Inverted Pendulum . . . 108
7.5.4 Robustness Analysis of Inverted Pendulum . . . 110
7.6 Conclusion . . . 111

8 Rational Neural Network Controllers 113
8.1 Introduction . . . 113
8.2 Rational Neural Networks . . . 115
8.2.1 Rational Approximation of Tanh Activation Function . . . 116
8.2.2 Rational Approximation of Sigmoid Activation Function . . . 118
8.2.3 Irrational Approximation of ReLU Activation Function . . . 119
8.2.4 General Rational Neural Networks . . . 120
8.3 Recovering Stabilising Controllers using Sum of Squares . . . 122
8.3.1 Extension to Rational Neural Network Controllers . . . 125
8.3.2 Refined Rational Neural Network Controller . . . 126
8.4 Numerical Examples . . . 128
8.4.1 One Dimensional System . . . 129
8.4.2 Three Dimensional Non-linear System . . . 129
8.4.3 Non-linear Inverted Pendulum . . . 131
8.5 Conclusion . . . 133

9 Conclusion 135
9.1 Thesis Summary . . . 135
9.2 Future Work . . . 137

Appendices

A CS-TSSOS Hierarchy 143
A.1 Sum of Squares Hierarchies . . . 143
A.2 TSSOS and Chordal-TSSOS . . . 144
A.3 Correlative Sparsity Term Sparse Sum of Squares . . . 147

A.4 Extension to Semi-algebraic Sets 148

References **153**

List of Figures

1.1 The image $\boldsymbol{x}$ being correctly classified as a 'panda' (left). With the addition of a noise component ($\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$) (middle) the image classifier incorrectly labels the (right) image as a 'gibbon' with a high degree of confidence. 4

2.1 Diagram showing how a matrix can be represented by a graph and then how a positive semidefinite constraint can be split into smaller positive semidefinite constraints using the properties of the chordal graph. The edge set of the chordal graph is $\mathcal{E} = \{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{3,4\}\}$ and the corresponding cliques are $\mathcal{J}_1 = \{1,2,3\}$ and $\mathcal{J}_2 = \{2,3,4\}$. 20

2.2 Diagram showing the neural network verification problem used in this thesis. $\mathcal{U}$ represents the input set, $\mathcal{Y}$ represents the output set and $\hat{\mathcal{Y}}$ is the approximation of this set. The safe set of outputs is denoted as $\mathcal{S}_y$. . . 22

2.3 Comparison of the neural feedback loop model used in this thesis and a reinforcement learning framework. 28

2.4 Plot showing a sector and slope constraint for a general non-linearity $\phi(x)$. 34

2.5 Plot showing the inequality constraints in (2.10) that bound the ReLU function (black). The red lines represents the inequality constraints, the blue lines represents the equality constraint and the yellow lines represent the constraints from the IBP values. The green shaded area is the region bounded by the constraints. 34

2.6 Visualisation of the triangular constraints on the ReLU function (black) described by (2.11). The red lines represents the inequality constraints and the green shaded area is the region bounded by the constraints. 35

2.7 Plot showing a comparison of the sector constraints that bound the sigmoid and tanh functions (black). The red lines represents the sector inequality constraints and the yellow lines represent the constraints from the IBP values. The green shaded area is the region bounded by the constraints. . 36

2.8 Visualisation of the triangular constraints described by (2.16). The red lines represents the inequality constraints and the green shaded area is the region bounded by the constraints. 36

2.9 Plots of common activation functions. 37

3.1 Graph showing the chordal structure of the SDP constraint matrix in (3.3). x_i^j represents the term in the constraint matrices corresponding to the jth layer with i^{th} hidden unit. The lines within each layer represent that all of the nodes in that layer are connected to each other. The solid lines between layers represents that all of the nodes in that layer are connected to all of the nodes in the adjacent layer. The dots represent that this pattern continues to the final layer in the neural network. 47
3.2 Diagram showing how the chordal extension of the sparsity graph is created and hence how the cliques in the constraint matrix are formed for a simple example in (3.4). 48
3.3 Diagram comparing the sparsity pattern of the constraint matrix in (3.3) for a ten-layer neural network with ten hidden units in each layer. 49
3.4 Diagram comparing the sparsity pattern of the constraint matrix in (3.3) for a twenty-layer neural network with five hidden units in each layer. . . 49
3.5 Graph comparing the computational time to solve an ℓ-layer network with two hidden units in each layer. 50
3.6 Graph comparing the computational time to solve an 100-layer network with a varying number of hidden units. 51
3.7 Graph comparing the computational time to solve an one-layer network with a varying number of hidden units. 52

4.1 Plot comparing sector constraints that bound the sigmoid function (black). The yellow lines represent the preprocessing bounds and the area in green represents the region bound by the constraints. 57
4.2 Plot comparing sector constraints that bound the tanh function (black). The yellow lines represent the preprocessing bounds and the area in green represents the region bound by the constraints. 58
4.3 Plot of the true output set (black) and the bounds that are created using NNPsatz (red), DeepSDP (blue) and IBP (yellow). 62

5.1 Diagrams showing the correlative sparsity graph and the sparsity graph ($\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$) of the polynomial in (5.5). The blue, red, green and yellow colours represent the cliques $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$ respectively. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$. 69
5.2 Diagram of a neural network showing how the cliques are formed for the example in Section 5.3.1. 73
5.3 Diagram showing the variables that are contained in a generic clique $\mathcal{J}_\tau$. The yellow, green and red nodes are the monomials from layer $\hat{\tau}$, the next layer $\tau = \hat{\tau} + 1$ and overlapping terms respectively. 75

5.4 Diagram of a general neural network showing how the cliques are formed. The yellow nodes are the input nodes, the red nodes are the hidden layers and the green nodes are the output nodes. 76
5.5 Comparing the computational time of different approaches for a neural network with two nodes in each layer and a varying number of layers. . . 81
5.6 Comparing the computational time of different approaches for a neural network with eight nodes in each layer and a varying number of layers. . . 82
5.7 Comparing the computational time of different approaches for a neural network with 100 layers and a varying number of nodes. 83
5.8 Comparing the accuracy of different approaches to the true values of the neural network's output. 84

6.1 Diagram of an NFL governed by (6.1) and (6.2). 87
6.2 Diagram of the reachable sets described by (6.3). 87
6.3 Diagram of the finite-time reach-avoid verification problem described by (6.4). 88
6.4 Diagram of the convex outer-approximations of the reachable sets that are used in (6.5). 88
6.5 Diagram of a neural network controller showing how the cliques $\mathcal{J}_k$ are formed. The yellow nodes are the states z_t, the red nodes are the hidden layers and the green nodes are the control inputs u_t. The clique $\mathcal{J}_{\ell+2}$ represents the connections between the z_t states and u_t, however the connections are not illustrated on the diagram for simplicity. 91
6.6 Reachable sets of the Duffing oscillator (6.9) using ReachSparsePsatz (black) and the true values (blue). 92
6.7 Reachable sets of the unstable polynomial system (6.10) using ReachSparsePsatz (black) and the true values (blue). 93
6.8 Diagram of the sector constraint that bounds the non-linearity in (6.11). . 94
6.9 Reachable sets of the inverted pendulum (6.11) using ReachSparsePsatz (black) and the true values (blue). 94

7.1 The trajectories of the Duffing Oscillator described by (7.10). 108
7.2 The region of attraction prodcued by NNSOSStability in Example 7.5.2 with a second-order Lyapunov function shown as a surface. The blue trajectories tend towards the zero equilibrium, whereas the red ones do not. 109
7.3 Diagram comparing the regions of attraction for different methods in Example 7.5.3. NNSOSStability with a fourth-order Lyapunov function is shown in black, the methods in [179] which uses diagonal and full-block acausal Zames-Falb multipliers are shown in yellow and red respectively. The trajectories of the system are shown in blue. 110

8.1 Plots showing a comparison between the Rtanh(x) and tanh(x) functions. 117
8.2 Reachable sets for the inverted pendulum in Section 6.7.3 where the tanh activation function is replaced with the Rtanh function. The reachable sets are computed using ReachSparsePsatz and are shown in black. The true reachable sets are shown in blue. 118
8.3 Diagram showing the region of attraction for the inverted pendulum in Section 7.5.3 where the tanh activation function is replaced with the Rtanh function. The region of attraction is computed using NNSOSStability (black) with a fourth order Lyapunov function. The system trajectories are shown in blue. 119
8.4 Plots showing a comparison between the Rsig(x) and sig(x) functions. . . 119
8.5 Plots showing a comparison between the IReLU(x) and ReLU(x) functions. 120
8.6 Plot showing the trajectories of the system state in Section 8.4.1 over time. 130
8.7 Plot showing the controller input of the system in Section 8.4.1 over time. 130
8.8 Plot showing the trajectories of the system states for the example in Section 8.4.2. 131
8.9 Plot showing the trajectories of the system states for the example in Section 8.4.3. 133

A.1 Diagrams showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}^\star)$ and the corresponding Gram matrix representation of the polynomial in (A.2). The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$. 146
A.2 Correlative sparsity graph of the polynomial in (A.4). 148
A.3 Diagram showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=1})$ and the corresponding Gram matrix representation of the polynomial in (A.4) with the monomial basis in (A.5) for the first iteration of the CS-TSSOS hierarchy. The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$. 149
A.4 Diagram showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=2})$ and the corresponding Gram matrix representation of the polynomial in (A.4) with the monomial basis in (A.5) for the second iteration of the CS-TSSOS hierarchy. The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$. 150
A.5 Diagram showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}^\star)$ and the corresponding Gram matrix representation of the polynomial in (A.4) with the monomial basis in (A.5) for the third and final iteration of the CS-TSSOS hierarchy. The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$. 151

List of Tables

2.1 Common activation functions with their respective slope and sector bounds. 38

4.1 Comparison of verification methods for the ReLU activation function . . . 60
4.2 Comparison of verification methods for the sigmoid activation function . . 61
4.3 Comparison of verification methods for the tanh activation function . . . 61

Notation

Sets

$\mathbb{N}$	Natural numbers $\{0, 1, 2, \ldots\}$
$\mathbb{R}$	Real numbers
$\mathbb{R}^n$	Real vectors of dimension n ($n \times 1$ matrices)
$\mathbb{R}^{n \times m}$	Real matrices of dimension $n \times m$
$\mathbb{S}^n$	Symmetric matrices of dimension $n \times n$
$\mathbb{S}^n_+$	Symmetric positive semidefinite matrices of dimension $n \times n$
$\mathbb{R}[x]$	Polynomials in x with real coefficients
$\Sigma[x]$	Sum of Squares polynomials in x

Definition and Inequalities

$A = B$	A and B are equal
$A := B$	A is defined by B
$A - B \succeq 0$	$A - B$ is positive semidefinite
$A - B \preceq 0$	$A - B$ is negative semidefinite
$A \circ B$	Hadamard product of A and B
$\lvert A \rvert$	Cardinality of A
$\binom{n}{k}$	Binomial coefficient, n choose k
$\deg(f)$	Degree of polynomial f

Vectors and Matrices

I_n	Identity matrix of dimension $n \times n$
X^T	Transpose of matrix X
$\operatorname{diag}(X_1, \ldots, X_n)$	Block-diagonal matrix with $X_1, \ldots, X_n$ on its diagonal blocks
$\langle x, y \rangle$	Inner product of vectors x and y, ($\langle x, y \rangle = x^T y$)
$\langle A, B \rangle$	Inner product of matrices A and B, ($\langle A, B \rangle = \operatorname{Trace}(A^T B)$)

Acronyms and Abbreviations

LMI	Linear Matrix Inequality
SDP	Semidefinite program and Semidefinite programming
LP	Linear program and Linear programming
SOS	Sum of Squares
NN	Neural network
NFL	Neural feedback loop
Psatz	Positivstellensatz
min	Minimise
s.t.	Subject to
TSSOS	Term sparse sum of squares
CS-TSSOS	Correlative sparsity term sparse sum of squares

1

Introduction

1.1 Motivation

The field of machine learning has seen a huge resurgence of interest over the past decade. The development of neural networks (NNs) and the more general field of machine learning are arguably two of the biggest breakthroughs in the field of Computer Science and Engineering of the current century. In particular, the development of Alexnet [1], Resnet [2] and GPT-3 [3] showed a step increase in the capacity of NNs to perform complex tasks that were once thought to be impossible to compute by machine. The increase in computational power available [4, 5], and the abundance of big data, has led to an increase in industrial applications of NNs and their prevalence is expanding. Over the past decade, there have been several solid use cases in many application areas. Examples of these include but are not limited to: image recognition, weather prediction, natural language processing, autonomous vehicle technology, medical imaging and social media algorithms [3, 6–11].

It is expected that society will be highly dependent on these algorithms and there has been wide speculation about how significantly the modern world will be impacted by this technology. At the time of writing this thesis, modern generative artificial intelligence models have exploded and have shown the potential to overtake humans in many application areas. Examples of such models include DALL-E 2 [12], which is able to generate highly detailed art from a text prompt through a diffusion model [13]. Chat-GPT [3] presents a chat bot that can respond to almost any topic to a high degree of accuracy and can source and compile information for the user in seconds. Finally, MusicML [14]

uses multi-stage autoregressive modeling to generate artificial music, predicted to be one of the most challenging domains of machine learning.

An NN's ability to act as a universal function approximator [15–18], combined with the use and availability of big data has meant that they have been successful in many industries. One of the most exciting applications of NNs is in the field of control theory. There were many works in this area in the 1980s and 1990s [19–21] and some dating back to the 1960s [22, 23]. In-fact NN architectures that have been popularized recently date back to this period, such as the long short-term memory network [24] and the convolutional neural network (CNN) [25]. One key example is [26] and [27] that aims to use an NN as a controller in a feedback control loop. We will refer to dynamical systems that contain an NN as a feedback controller as Neural Feedback Loops (NFLs).

There has been a marked interest in using NNs in control with the emergence of the parallel field of reinforcement learning: deep reinforcement learning has harnessed the power of NNs to create a decision-making agent to greatly outperform humans in many complex tasks, such as the video game Dota 2 [28] and the board game Go [29]. However, performance of these policies that are learnt can perform poorly when the real environment varies from the learnt environment [30–33]. Bounds to quantify their safety have been developed, however are often overly conservative and do not quantify the performance of the algorithm sufficiently [34]. The success of NNs and reinforcement learning, combined with new advancements in robust control, motivates work at their intersection. In particular, the field of reinforcement learning has provided a bridge to develop data-driven control methods, diverging away from traditional model-based approaches [35]. The intersection of these works could be the most useful to society, as big data becomes more widely available.

Data-driven control methods have increased in popularity, most notably due to work from [36], which provides a fundamental framework for the problem. There is the possibility that an NN can be used with recent data-based control methods, to move away from requiring an accurate model of a system. Using NNs as controllers has the potential to provide improved performance over traditional approaches. Early work from [37] introduces ALVINN which uses an NN to control an autonomous land vehicle. By

using a guided policy search, [38] are able to learn an NN policy under unknown dynamics. Work from [39] presents a general procedure to design NN controllers that preserve the desirable properties of a model predictive control scheme.

NNs have also shown to be useful in multiple aspects of control systems, such as in system identification tasks. For example, in [40] a quadrotor's dynamics are learnt through an NN. More generally recurrent NNs and physics-informed NNs [41, 42] can be used to learn an unknown model using data. NNs have also been used as Kalman filters to learn the states and parameters of a system online [43]. Other examples include the work in [44–47], which uses an NN to learn Lyapunov functions to show stability of a feedback system. Further works include [48], which uses an NN to learn and verify a feedback control system; and [49] that can identify false data injection attacks in control systems by implementing an artificial NN. NNs have also shown preliminary uses in solving optimisation problems [50, 51], which have yet to be applied to control systems.

However, despite their overwhelming success at many sophisticated tasks, NN models suffer from the very attribute which has made them so effective. This is the 'black-box' approach, where an extremely large number of parameters are trained without any clear indication of why these values are selected. However, this approach can be justified due to the idea of over-parameterisation, which has shown to be an effective tool [52, 53]. A phenomenon known as double decent has been observed, which is where, as the model size increases, the performance of the model gets worse initially and then improves after a certain size. The merits of these large models that have been shown experimentally can justify their use in some societal applications, despite that lack of interpretability of the algorithms.

One important consideration is the increasing use of NNs in safety-critical applications, such as autonomous vehicle technology. This accentuates the biggest shortcoming of NNs, which is their sensitivity to adversarial inputs [54]: small changes in the input set can lead to large changes in the output. In particular, works from [55–57] show how large fluctuations in the outputs of the NN are caused by relatively small changes to the input. One of the most popular examples of an NN's sensitivity to adversarial attacks is in image classifiers. In Figure 1.1, taken from [58], we can see an NN classifier that

correctly determines 'panda' in the image. However, with the presence of a small amount of noise, it classifies the object as 'gibbon' with a high degree of confidence. This has been observed in countless other image classification examples. It is clear to see how this lack of robustness of these models can lead to disastrous consequences - for example, if an image classification system was used in an autonomous vehicle to detect road signs.

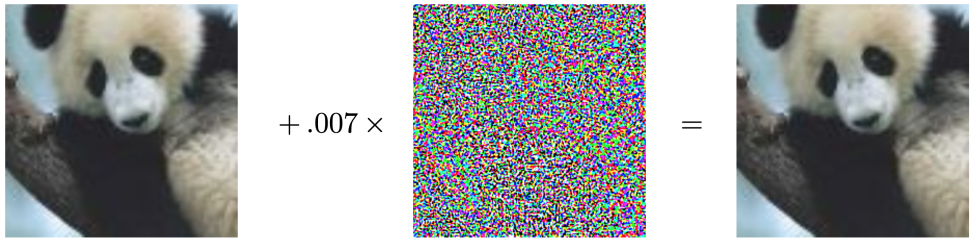


Figure 1.1: The image $\boldsymbol{x}$ being correctly classified as a 'panda' (left). With the addition of a noise component ($\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$) (middle) the image classifier incorrectly labels the (right) image as a 'gibbon' with a high degree of confidence.

Obtaining robustness guarantees on NNs is essential to overcoming the issues surrounding the 'black box' approach. Despite considerable efforts from the research community to improve our understanding and to allow certification of NNs, to date guarantees on these NNs are not sufficient to allow for their widespread use in safety-critical applications. Additionally, given that feedback systems are usually subject to external perturbations, the issue surrounding an NN's sensitivity in an NFL must be overcome. Determining robustness certificates on NFLs such as stability and reachability is important if they are to be used more frequently in practice.

To provide robust guarantees, one popular method that has seen success is to describe and bound the activation functions in the NN using equality and inequality constraints [59]. This technique is what underpins the work in this thesis. However, there are numerous ways to form these bounds, providing a trade-off between conservativeness and complexity. There has been significant progress to improve the efficiency and the accuracy of these methods. This is a rapidly developing field, with a large number of results aiming to provide tighter and more efficient bounds on the NN's performance.

1.2 Contribution to Literature

This thesis is centered around the NN verification problem, which involves computing approximations of the true outputs sets of NNs, (see Section 2.2). One of our main contributions is to improve the scalability of this problem, in particular semidefinite programming (SDP) frameworks are used. We also reformulate the problem and use a sparse polynomial optimisation framework to improve the accuracy of these output sets. We then expand these ideas to non-linear systems with NN controllers, which we refer to as NFLs. We conduct reachability and stability analysis of these systems to achieve better robustness certificates. Finally, we propose a novel approach and NN architecture based on rational functions to recover a stabilising NN controller in a convex way.

Exploiting Sparsity for Neural Network Verification

We reformulate a recently proposed SDP approach [60] that uses quadratic constraints to provide robustness certifications on a general non-linear feed-forward NN. Through examining the formulation of the problem, we show that due to the intrinsic cascading structure of the NN, the constraint matrices in the semidefinite program (SDP) form a block-arrow pattern and satisfy conditions for chordal sparsity or that they can be extended easily to possess such a pattern. We then use an SDP decomposition approach to significantly speed up the computational time that it takes to solve the optimisation problem. We compare the trade-off between conservatism and computational time using SeDuMi [61], which uses interior point methods to solve the optimisation problem. We show how the computational time varies with the structure of the NN. These results have been published in [62].

M. Newton and A. Papachristodoulou, "Exploiting Sparsity for Neural Network Verification," in *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, PMLR, 2021, pp. 715-727.

Neural Network Verification using Polynomial Optimisation

We approach the NN verification problem from a different perspective by considering a general class of constraints representing different activation functions, leading to a semi-algebraic set. The emptiness of this set can then be tested using Positivstellensatz (Psatz) [63] and polynomial optimisation. This method is different, as instead of considering only one type of constraint e.g. linear, quadratic etc., we can use a mixture of the constraints and easily trade off solution accuracy with computational effort. Testing the algebraic condition on the Psatz can be done using Sum of Squares (SOS), or equivalently by solving an SDP. This SOS framework can be parsed in MATLAB using SOSTOOLS [64] and the resulting SDP can be solved using a relevant solver e.g. SDPT3 [65]. We show that the Psatz formulation can drastically tighten the bounds when using the ReLU activation functions against existing methods. The second contribution is that we provide significantly tighter bounds on the sigmoid and tanh activation functions by using a combination of two sector constraints. Using both of these contributions allows more flexibility in the optimisation problem. This is because any set of constraints can be defined and then the accuracy may be improved if needed by increasing the order of the Psatz test. The results of this chapter have been published in [66].

M. Newton and A. Papachristodoulou, "Neural Network Verification using Polynomial Optimisation," 2021 60th IEEE Conference on Decision and Control (CDC), 2021, pp. 5092-5097.

Sparse Polynomial Optimisation for Neural Network Verification

Previous work in the literature has focused on obtaining tighter bounds on the potential outputs of an NN or improving the scalability of the optimisation problem. However, these problems have not been addressed simultaneously. Another important drawback of the previous literature is that each framework is analysed in isolation, with most papers focusing on one type of constraint for an individual activation function. It is of course important to optimise bounds for each activation function to provide the best results. However, having a unified framework to certify the bounds on the NN would be useful,

so that these methods can be combined together to give the best possible bounds in all scenarios. We therefore look for approaches that both generalise the bounds on the NN, whilst considering the scalability of solving the optimisation problem.

We combine the work on exploiting sparsity [62] and polynomial optimisation [66] using a sparse polynomial optimisation approach and the Psatz. Simultaneously exploiting the natural cascading structure of the NN using chordal sparsity, whilst testing the emptiness of a semi-algebraic set using algebra, provides tight bounds without excessive computation. We implement the optimisation problem and show through numerous examples that this method improves both the computational time and accuracy of the bounds. We compare the solve times of different solvers and show how the accuracy can be improved at the expense of increased computation time by modifying the order of the algebraic condition. These results are based on work in [67]. We test our method for networks with ReLU, sigmoid and tanh activation functions, however, this method can be easily extended to any activation function. Previously, NNs of this size have not been verified to this degree of accuracy.

M. Newton, and A. Papachristodoulou, "Sparse Polynomial Optimisation for Neural Network Verification", *arXiv preprint arXiv:2202.02241*, 2022.

Reachability Analysis of Neural Feedback Loops using Sparse Polynomial Optimisation

We investigate the problem of computing the reachable sets of non-linear systems that contain NN controllers. We adapt the sparse polynomial optimisation framework in [67] and incorporate it within the NFL reachability problem. By carefully constructing the semi-algebraic set, the reachable sets can be computed using SOS and SDP. Due to the polynomial optimisation construction, we can include any non-linear polynomial dynamics in the plant model, which has not been possible in previous frameworks that need to use alternative workaround solutions to incorporate the non-linearities. We are also able to easily trade off solution accuracy with computational complexity. We show through examples that our method can provide tighter reachable sets over similar methods, with reasonable computational time. These results have been published in [68].

M. Newton and A. Papachristodoulou, "Reachability Analysis of Neural Feedback Loops using Sparse Polynomial Optimisation," in *Proceedings of the 61st Conference on Decision and Control*, 2022, pp. 2745-2750.

Stability of Non-linear Neural Feedback Loops using Sum of Squares

We use Lyapunov stability theory for constrained systems to analyse non-linear NFLs. Works in previous literature only address linear systems or require one to abstract the plant non-linearities and bound them using slope and sector constraints. We use SOS programming to directly incorporate the non-linearities of the plant, instead of relying on such bounds. Within this framework, we can propose higher order candidate Lyapunov functions with richer structures which are able to better capture the dynamics of the non-linear system and the nonlinearities in the NN. We are also able to analyse these systems in continuous time, whereas other methods rely on discretising the system. These higher order Lyapunov functions are used in conjunction with higher order multipliers on the inequality and equality constraints that bound the NN input-output properties. The volume of the region of attraction computed is increased compared to other methods, allowing for better safety guarantees on the stability of the system. We are also able to easily analyse non-linear polynomial systems and conduct robustness analysis on the parameter uncertainty. The results in this chapter have been published in [69].

M. Newton and A. Papachristodoulou, "Stability of Non-linear Neural Feedback Loops using Sum of Squares," in *Proceedings of the 61st Conference on Decision and Control*, 2022, pp. 6000-6005.

Other publications

The work during my studies which is not covered in this thesis has also led to the publication:

M. Newton and A. Papachristodoulou, "Network Lyapunov Functions for Epidemic Models," in *Proceedings of 59th Conference on Decision and Control*, 2020, pp. 1798-1803.

All code for this thesis can be found here: https://github.com/MNewtonOX.

1.3 Thesis Outline

We now describe the structure of this thesis, outlining the contributions that are presented in each chapter.

Chapter 2 provides the required background information and presents the mathematical concepts used in the thesis. The NN verification problem is defined and a summary of the tasks that it can be used for is outlined. We summarise the ways in which the input-output relationship of an NN can be represented as a series of inequality and equality constraints and therefore abstracted as a constraint set. This is the key concept that underpins the work in the thesis.

Chapter 3 describes how the NN verification problem can be formed as an SDP. An overview of chordal graphs and their relation to matrix decomposition is provided. We then show how this matrix decomposition can be applied to the NN verification problem and demonstrate how it can be used to significantly speed up the computational time required to solve the optimisation problem.

Chapter 4 describes how the Psatz can be used in the NN verification problem. Improved bounds on the sigmoid and tanh activation functions are introduced. The results are presented and discussed, showing how tighter bounds on the NN's output can be computed using this method.

Chapter 5 presents the theory behind sparse polynomial optimisation and the link to Psatz. The theory is then applied to the specific NN verification problem showing how sparsity is preserved in the optimisation problem. The results from various experiments against state-of-the-art methods are presented and analysed, with both scalability and accuracy being compared.

Chapter 6 provides a summary of the preliminary information surrounding the NFL reachability problem. We then show how the reachability problem is formulated in the sparse polynomial optimisation framework and how the sparsity pattern is preserved. We then highlight the benefits of this method with numerical examples of non-linear systems.

Chapter 7 introduces Lyapunov stability and how it can be verified using SOS programming. We describe how Lyapunov stability can be applied to constrained systems and how SOS programming can be used to compute the region of attraction. We then

apply this theory to NFLs by treating them as non-linear constrained dynamical systems. We show how the region of attraction computed using our method can be increased compared to competitive approaches.

Chapter 8 provides a discussion of rational NNs. We propose novel rational activation functions that can be used effectively in robustness problems for NFLs. We propose a method to recover a stabilising controller from an SOS feasibility test and apply this approach to a refined rational NN which is more compatible with SOS programming. We show that this method is able to recover stabilising rational NN controllers for NFLs.

Chapter 9 concludes the thesis and plans for future work are outlined.

2

Background

This chapter provides the required background information used in the subsequent chapters of this thesis. Semidefinite programming, Sum of Squares programming and the Positivstellensatz are introduced and described. To show how sparsity in semidefinite programs can be exploited, details behind chordal graphs and their relation to matrix decomposition are provided. The neural network verification problem is outlined and a summary of previous literature in the area is presented. This chapter then shows the ways in which the input-output relationship of a neural network can be represented as a series of inequality and equality constraints and therefore abstracted as a constraint set.

2.1 Mathematical Preliminaries

2.1.1 Semidefinite Programming

Many problems in control theory can be solved using linear matrix inequalities (LMIs) as constraints [70]. An LMI constraint $A(x)$ is one that can be written as

$$A(x) = A_0 + \sum_{i=1}^{m} A_i x_i \succeq 0,$$

where $x \in \mathbb{R}^m$ and $A_0, A_1, \ldots, A_m \in \mathbb{S}^n$. LMIs are convex in the variable x and the intersection of multiple LMI constraints is also convex.

A semidefinite program (SDP) is one in which the objective function is a linear function of the optimisation variables with respect to a collection of LMI constraints (also referred to as semidefinite constraints) [71]. Consider $C, A_0, A_1, \ldots, A_m \in \mathbb{S}^n$ and $b \in \mathbb{R}^m$ that

are given by the problem data; an SDP is defined as

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & b^T x \\ \text{subject to} \quad & C - \sum_{i=1}^{m} A_i x_i \succeq 0. \end{aligned}$$

One of the most beneficial aspects of SDPs is that it is a convex optimisation problem, since the objective and constraints are convex in X and the LMI constraints are also convex. Therefore, if the size of the problem data matrices is not too large then the problem is computationally tractable. There are many well established SDP solvers such as SeDuMi [61] and SDPT3 [65]. Additionally there are many parsers such as CVX [72] and YALMIP [73], to allow the SDPs to be implemented effectively.

A linear program (LP) is another convex optimisation problem class. Although not specifically utilised in this thesis, LPs will be referred to throughout. An LP can be written in the form

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & b^T x \\ \text{subject to} \quad & c - Ax \geq 0, \\ & x \geq 0, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are given by the problem data. SDPs involve optimising over a cone of positive semidefinite matrices, whereas LPs are optimised over the positive orthant (note that SDPs include LPs). Both LPs and SDPs can be solved using interior-point methods in polynomial time. There have been many recent advancements to improve the scalability of SDPs [74]. However, LPs scale much better when n and m are increased [75]. The literature on convex optimisation is vast; [75, 76] are comprehensive resources on the topic.

2.1.2 Sum of Squares Programming

Many control problems can be solved using an SDP. However, there are numerous instances of control-related problems that require one to check if a polynomial is nonnegative [77], which is usually not possible directly with an LP or SDP. In general checking global nonnegativity of a polynomial is an NP-hard problem [78]. However, it is possible to

use Sum of Squares (SOS) as a computational relaxation to nonnegativity, which results in an SDP and hence solved in polynomial time. For further details on the relationship between SDP and SOS, the readers are referred to [79].

We now discuss how to formulate and solve SOS optimisation problems through SDPs. The core principle of SOS is to replace a positivity condition with a condition that enforces the expression to be an SOS polynomial.

Definition 2.1. A polynomial $p(x)$ is said to be a *Sum of Squares (SOS)* polynomial if it can be expressed as

$$p(x) = \sum_{i=1} r_i^2(x),$$

where $r_i(x) \in \mathbb{R}[x]$. We denote the set of polynomials that admit this decomposition by $\Sigma[x]$ and say '$p(x)$ is SOS'. Note that throughout we define $\mathbb{R}[x_1, \ldots, x_n]$ to be the set of polynomials in $x_1, \ldots, x_n$ with real coefficients. We denote $x = (x_1, \ldots, x_n)$ for simplicity.

Definition 2.2. A *monomial* in $x = (x_1, \ldots, x_n)$ is

$$x^\beta = x_1^{\beta_1} x_2^{\beta_2} \ldots x_n^{\beta_n},$$

where the exponent and degree are denoted as $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{N}^n$ and $|\beta| = \beta_1 + \cdots + \beta_n$ respectively.

We express the column vector of monomials with only certain exponents as $x^{\mathbb{B}} = (x^\beta)_{\beta \in \mathbb{B}}$, where $\mathbb{B} \subset \mathbb{N}^n$ is the set of exponents that are used in the monomials. The summation and multiplication operations on $\mathbb{B}$ are defined as

$$\mathbb{B} + \mathbb{B} := \{\beta + \gamma \ : \ \beta, \gamma \in \mathbb{B}\},$$

$$2\mathbb{B} := \{2\beta \ : \ \beta \in \mathbb{B}\}.$$

Any polynomial $f(x)$ can be written as a linear combination of a set of monomials $x^{\mathbb{B}}$ such that

$$f(x) = \sum_{\beta \in \mathbb{N}_d^n} f_\beta x^\beta,$$

for a set of coefficients $f_\beta \in \mathbb{R}$, where $\mathbb{N}_d^n = \{\beta \in \mathbb{N}^n \ : \ |\beta| \leq d\}$ is the set of all n-variate exponents of degree d or less.

Definition 2.3. The *support* of $f(x)$ is defined as the set of exponents with nonzero coefficient such that

$$\text{supp}(f(x)) = \{\beta \in \mathbb{N}_d^n \ : \ f_\beta \neq 0\}.$$

Theorem 2.4. ([79]) A polynomial $f(x)$ is SOS if and only if it can be written in what is referred to as a *Gram matrix representation* such that

$$f(x) = (x^{\mathbb{B}})^T Q x^{\mathbb{B}},$$

where $Q \in \mathbb{S}_+^{|\mathbb{B}|}$ is a positive semidefinite matrix.

We define the symmetric binary matrix $A_\alpha \in \mathbb{S}^{|\mathbb{B}|}$ for each exponent $\alpha \in \mathbb{B} + \mathbb{B}$ as

$$[A_\alpha]_{\beta,\gamma} := \begin{cases} 1, \ \beta + \gamma = \alpha, \\ 0, \ \text{otherwise.} \end{cases}$$

We can then rewrite the Gram matrix representation as

$$(x^{\mathbb{B}})^T Q x^{\mathbb{B}} = \langle Q, x^{\mathbb{B}}(x^{\mathbb{B}})^T \rangle = \sum_{\alpha = \mathbb{B}+\mathbb{B}} \langle Q, A_\alpha \rangle x^\alpha.$$

Therefore, the following is true

$$f(x) \in \Sigma[x] \Leftrightarrow \exists Q \in \mathbb{S}_+^{|\mathbb{B}|} \text{ where } \langle Q, A_\alpha \rangle = f_\alpha, \ \forall \alpha \in \mathbb{B} + \mathbb{B}.$$

This means that checking the SOS condition is equivalent to solving an SDP. Parsers such as SOSTOOLS [64] in MATLAB or the SumOfSquares.jl package [80] in Julia can be used to formulate this SDP.

Example 2.5. ([81]) Let $x = [x_1, x_2]^T \in \mathbb{R}^2$ and consider the polynomial

$$p(x) = x_1^4 + 2x_1^3 x_2 + 3x_1^2 x_2^2 + 2x_1 x_2^3 + 2x_2^2.$$

By selecting the monomial vector $x^{\mathbb{B}} = [x_1^2, x_1 x_2, x_2^2]^T$, the polynomial can be written in its Gram matrix representation as

$$p(x) = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}.$$

This results in the set of linear equations

$$q_{11} = 1, \ q_{12} = 2, \ q_{22} = 3 - 2q_{13}, \ q_{23} = 1, \ q_{33} = 2,$$

which can be used to express the following LMI condition

$$\begin{aligned} \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} &\succeq 0, \\ \implies \begin{bmatrix} 1 & 1 & \lambda \\ 1 & 3-\lambda & 1 \\ \lambda & 1 & 2 \end{bmatrix} &\succeq 0, \\ \implies \begin{bmatrix} 1 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix} + \lambda \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} &\succeq 0. \end{aligned}$$

The positive semidefinite constraint is valid for $-1 \leq \lambda \leq 1$ and therefore shows that $p(x) \in \Sigma[x]$. For $\lambda = -1$ it follows that

$$\begin{aligned} p(x) &= \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}^T \begin{bmatrix} 1 & 1 & -1 \\ 1 & 4 & 1 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}, \\ &= \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 1 & 2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}, \\ &= \left|\left| \begin{bmatrix} x_1^2 + x_1x_2 - x_2^2 \\ 2x_1x_2 + x_2^2 \end{bmatrix} \right|\right|^2, \\ &= (x_1^2 + x_1x_2 - x_2^2)^2 + (2x_1x_2 + x_2^2)^2, \end{aligned}$$

which demonstrates how $p(x)$ is written as an SOS polynomial.

In the default case the vector $x^{\mathbb{B}}$ contains all monomials of degree up to $\deg(\frac{1}{2}f)$ and the matrix Q is a fully dense matrix of size $\binom{n+d}{d} \times \binom{n+d}{d}$. One of the biggest drawbacks of the SOS technique when solving optimisation problems is that it scales poorly as the number of variables becomes large [82]. However, in a system that possesses a significant level of sparsity then it is expected that not all of the monomial terms may be needed. Hence if only a subset of them are included, the size of the resulting SDP will be reduced; this idea is explored in more detail in Chapter 5.

Note that in general if a polynomial is found to be SOS then it is only a sufficient condition for global nonnegativity. The Motzkin polynomial, which is written as

$$M(x_1, x_2) = x_1^2x_2^4 + x_1^4x_2^2 + 1 - 3x_1^2x_2^2,$$

is a well known example of a nonnegative polynomial that is not representable by an SOS [83]. More recent literature has shown that as the number of variables gets

larger the number of nonnegative polynomials not representable by an SOS becomes increasingly common [84]. There is a very active research community working to better understand this area.

There are many ways in which SOS programming can be taken further; [82] covers the recent advances and limitations in SOS optimization, highlighting the ways efficiency and scalability can be improved. One method that has a trade-off between accuracy and performance is Diagonally Dominant Sum of Squares and Scaled Diagonally Dominant Sum of Squares [85, 86]. Polynomial optimisation problems can be solved using the problem of moments [87], which is equivalent to the dual of the SOS problem.

2.1.3 The Positivstellensatz

One of the central theorems of real algebraic geometry is known as the Positivstellensatz (Psatz) [63], which will now be described briefly. The core idea of the Psatz is to provide an equivalent relation between an algebraic condition and the emptiness of a semi-algebraic set. Therefore, instead of checking whether a constraint set satisfies a feasibility test, we find the limiting values at which the set is infeasible and hence provide bounds on the feasible region.

A basic closed semi-algebraic set is defined by

$$\{x \in \mathbb{R}^n \mid f_i(x) \geq 0 \ \forall\, i = 1, \ldots, m\}.$$

where $f_i(x)$ are polynomials in x, i.e. $f_i(x) \in \mathbb{R}[x]$. This can be extended to define the set

$$S = \{x \in \mathbb{R}^n \mid f_i(x) \star 0 \ \forall\, i = 1, \ldots, m\},$$

where $\star$ denotes $<, \leq, =$, or $\neq$. We express the semi-algebraic set with notation

$$S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0\, \forall\, i = 1, \ldots, p,\ j = 1, \ldots, q\}, \tag{2.1}$$

where g_i and h_j are polynomial functions.

Definition 2.6. Given $f_1, \ldots, f_r \in \mathbb{R}[x]$, the *multiplicative monoid* generated by f_k's is the set of all finite products of f_k's, including 1 (i.e. the empty product). It is denoted as $\mathcal{M}(f_1, \ldots, f_r)$.

Definition 2.7. Given $g_1, \ldots, g_p \in \mathbb{R}[x]$, the *cone* generated by g_i's is

$$\text{cone}\{g_1, \ldots, g_p\} = \left\{ s_0 + \sum_{i=1}^{p} s_i G_i \mid s_i \in \Sigma[x],\ G_i \in \mathcal{M}(g_1, \ldots, g_p) \right\}. \tag{2.2}$$

Definition 2.8. Given $h_1, \ldots, h_q \in \mathbb{R}[x]$, the *ideal* generated by h_k's is

$$\text{ideal}\{h_1, \ldots, h_q\} = \left\{ \sum_{j=1}^{q} t_j h_j \mid t_j \in \mathbb{R}[x] \right\}. \tag{2.3}$$

Theorem 2.9. (Positivstellensatz, [63]) Given the semi-algebraic set S in (2.1), the following are equivalent:

1. The set S is empty.

2. There exist $s_i \in \Sigma[x]$ in (2.2) and $t_j \in \mathbb{R}[x]$ in (2.3) such that

$$\text{cone}\{g_1, \ldots, g_p\} + \text{ideal}\{h_1, \ldots, h_q\} = 0.$$

Theorem 2.9 links the emptiness of a semi-algebraic set with an algebraic test. There are many reformulations of the Psatz that can be used for different purposes, such as Schmüdgen Psatz [88]. To create a convex test based on Theorem 2.9 for computational purposes, one can use a representation of the function $f(x)$.

Proposition 2.10. Consider the set S in (2.1). If

$$f(x) = 1 + \sum_{j}^{q} t_j h_j + s_0 + \sum_{i}^{p} s_i g_i + \sum_{i \neq j}^{p} r_{ij} g_i g_j + \sum_{i \neq j \neq k}^{p} r_{ijk} g_i g_j g_k + \ldots, \tag{2.4}$$

then $f(x) > 0$, $\forall x \in S$, where $s_i, r_{ij}, r_{ijk} \ldots \in \Sigma[x]$ and $t_j \in \mathbb{R}[x]$.

Proof. We substitute (2.2) and (2.3) into the expression in Theorem 2.9 and rearrange the expression for one of the inequalities in the semi-algebraic set (g_0), which we reassign to $f(x)$. ■

However, when there is a large number of constraints in the semi-algebraic set a simplification of Theorem 2.10 is required, which is shown in the following proposition.

Proposition 2.11. Consider the set S in (2.1). If

$$f(x) = 1 + \sum_{j}^{q} t_j h_j + s_0 + \sum_{i}^{p} s_i g_i, \tag{2.5}$$

then $f(x) > 0$, $\forall x \in S$, where $s_i \in \Sigma[x]$ and $t_j \in \mathbb{R}[x]$.

Proof. Following from Proposition 2.10, we set $r_{ij}, r_{ijk}, \ldots$ to zero to provide the simplified expression. ■

2.1.4 Sum of Squares and The Positivstellensatz

It is possible to use SOS programming to test the algebraic Psatz condition. We can test if the polynomial $f(x) > 0$, $\forall x \in S$ by rewriting (2.5) as

$$f(x) + \sum_{j}^{q} t_j h_j - \sum_{i}^{p} s_i g_i \in \Sigma[x],$$

where $s_i \in \Sigma[x]$ and $t_j \in \mathbb{R}[x]$. This results in a set of SOS conditions, which can be checked using SOSTOOLS [64]. By choosing higher degree multipliers s_i, t_j etc. one can obtain a series of set emptiness tests of increasing complexity and non-decreasing accuracy. An example of this is how the Psatz generalises and extends the well-known S-procedure [89] in control.

Lemma 2.12. (S-procedure, [89]) Given a set of symmetric matrices $\{A_k \in \mathbb{R}^{n\times n} \mid k = 0, \ldots, p\}$, if $A_0 - \sum_{k=1}^{p} \lambda_k A_k \succeq 0$, where λ_k is a nonnegative scalar $\forall k = 1, \ldots, p$ then

$$\bigcap_{k=1}^{p} \{x \in \mathbb{R}^n \mid x^T A_k x \geq 0\} \subseteq \{x \in \mathbb{R}^n \mid x^T A_0 x \geq 0\}.$$

This can be written as the Psatz by considering if the set

$$\{x \in \mathbb{R}^n \mid x^T A_1 x \geq 0,\ \ldots, x^T A_p x \geq 0,\ -x^T A_0 x > 0\}$$

is empty. Since there are no equality constraints, the Psatz condition becomes $0 \in \text{cone}\{g_1, \ldots, g_p\}$. By setting the inequality constraints to $g_i = x^T A_i x$ and the SOS multipliers to $s_i = \lambda_i$ one can recover the standard S-procedure test, which is known to be conservative. By increasing the order of the multipliers or by multiplying the constraints together (as in the Psatz) one can obtain better tests for the set emptiness, however these tests are more computationally expensive. Therefore, compared to the full Psatz, the S-procedure is limited in many ways: it is restricted to quadratic constraints, no constraints can be multiplied together and the SOS multipliers can only be scalars. For further information about the S-procedure and its variants, the readers are referred to [89].

2.1.5 Chordal Graphs and Sparse Matrix Decomposition

In this section we provide an overview of the theory behind chordal graphs and how they can be used to exploit the sparsity in positive semidefinite matrices, and hence how they can be used to speed up the solve time of an SDP.

A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V} = \{1, 2, \ldots, n\}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A vertex-induced subgraph $G^{'}(\mathcal{V}^{'}, \mathcal{E}^{'})$ is another graph that is formed from a subset of the vertices of the original graph $G(\mathcal{V}, \mathcal{E})$, the edges of the subgraph are all of the edges from the original graph connecting pairs of vertices in that subset. A clique $\mathcal{C} \subseteq \mathcal{V}$ is a subgraph such that all the vertices in the subgraph $\mathcal{C}$ form a complete graph - a complete graph is a graph such that any two nodes are connected by an edge. A maximal clique is a clique that is not a subset of any other clique. A graph can contain a cycle, which is defined by a set of pairwise distinct nodes $\{v_1, v_2, \ldots, v_k\} \subset \mathcal{V}$ such that $(v_k, v_1) \in \mathcal{E}$ and $(v_i, v_{i+1}) \in \mathcal{E}$ for $i = 1, \ldots, k-1$. A chord that lies on the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is an edge that joins two non-adjacent nodes in a cycle [90].

Definition 2.13. A connected undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a chordal graph if every cycle of length four or greater has at least one chord.

Chordal graphs are useful since they can be decomposed into their maximal cliques [91]. The properties of chordal graphs have been exploited in many problems, however there are also different classes of chordal graphs that have additional properties. For this reason it can be useful to extend a non-chordal graph to a chordal graph by adding additional edges to the non-chordal graph.

Definition 2.14. The chordal extension of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is denoted as $\hat{\mathcal{G}}(\mathcal{V}, \hat{\mathcal{E}})$, where $\mathcal{E} \subseteq \hat{\mathcal{E}}$ and $\hat{\mathcal{G}}$ is chordal.

We can relate the sparsity pattern of a symmetric matrix X to that of an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and consider the set

$$\mathbb{S}^n(\mathcal{E}, 0) := \{X \in \mathbb{S}^n \mid X_{ij} = X_{ji} = 0 \text{ if } (i, j) \notin \mathcal{E}\}.$$

This means that the matrix X has a zero in elements that correspond to the nodes that are not connected by edges on the graph. Of interest is whether $X \in \mathbb{S}^n_+(\mathcal{E}, 0)$

where $\mathbb{S}^n_+(\mathcal{E},0) := \mathbb{S}^n(\mathcal{E},0) \cap \mathbb{S}^n_+$. Just as a chordal graph can be decomposed into its maximal cliques, a matrix X with a chordal sparsity pattern can be split up into smaller sub-matrices, with the sub-matrices corresponding to the maximal cliques of the chordal graph. This is shown visually in Figure 2.1. An important result relates matrices X that are positive semidefinite, to such a decomposition.

Theorem 2.15. ([92]) Consider the chordal graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ that is made up of maximal cliques $\{\mathcal{C}_1,\mathcal{C}_2,\ldots,\mathcal{C}_t\}$. Then $Z \in \mathbb{S}^n_+(\mathcal{E},0)$ if and only if there exist $Z_k \in \mathbb{S}^{|\mathcal{C}_k|}_+$ for $k=1,\ldots,t$ such that

$$Z = \sum_{k=1}^{t} E_{\mathcal{C}_k}^T Z_k E_{\mathcal{C}_k},$$

where $\mathbb{S}^n_+(\mathcal{E},0) := \{X \in \mathbb{S}^n \mid X \succeq 0 \mid X_{ij} = X_{ji} = 0, \text{ if } i \neq j \text{ and } (i,j) \notin \mathcal{E}\}$, $|\mathcal{C}_i|$ is the number of vertices in that clique and

$$(E_{\mathcal{C}_k})_{ij} = \begin{cases} 1, & \text{if } \mathcal{C}_k(i) = j, \\ 0, & \text{otherwise.} \end{cases}$$

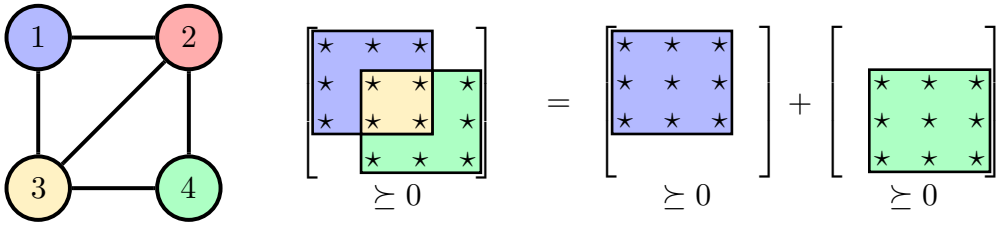


Figure 2.1: Diagram showing how a matrix can be represented by a graph and then how a positive semidefinite constraint can be split into smaller positive semidefinite constraints using the properties of the chordal graph. The edge set of the chordal graph is $\mathcal{E} = \{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{3,4\}\}$ and the corresponding cliques are $\mathcal{J}_1 = \{1,2,3\}$ and $\mathcal{J}_2 = \{2,3,4\}$.

The chordal sparsity property can be used to replace a large positive semidefinite condition of a single large matrix in an SDP with multiple positive semidefinite conditions of smaller size. This is useful as it means that testing positive semidefiniteness of a large matrix with chordal sparsity can be executed in a distributed way. This idea can also be extended to sparse block matrices [93].

The above ideas have been incorporated in many different solvers to exploit this chordal sparsity property. One example is sparseCoLo [94], which uses four conversion methods via positive semidefinite matrix completion. It uses both the primal and dual forms of linear, semidefinite and second-order cone programs with both equality and

inequality constraints. The first conversion method utilizes domain space sparsity using clique trees, the second still in the domain space uses a basis representation. The third also uses clique trees but in the range space, this is combined with the fourth conversion method, which uses matrix decomposition to exploit the range space sparsity in an LMI. CDCS [95] is another solver that uses a first order splitting method called Alternating Direction Method of Multipliers (ADMM). This solver scales better to large systems but often provides less accurate solutions. Another solver that uses an operator splitting method is the Conic Operator Splitting Method (COSMO) [96] for convex optimisation problems with a quadratic objective function and conic constraints. COSMO uses chordal decomposition and a clique merging algorithm to exploit the sparsity of the problem. Other examples of solvers include SMCP [97] and SDPA-C [98].

2.2 Neural Network Verification

The essence of the neural network (NN) verification problem is to provide a confidence measure of how uncertainty in the input set maps to the output set. Most work in the literature has focused on analysing NNs in isolation (i.e. there is no feedback element and the NN is used as an input-output function). This is what is used in most well-known applications of NNs, such as image classification. The term 'verification' has been used to describe the general class of problems that analyse the input-output properties of NNs. However, it is also used to specify a general sub-problem as will be explained now.

The more specific NN verification problem involves checking whether the output of the NN will classify the same label given some perturbation on the input set. The converse of this problem is to determine how large the input perturbation must be for the NN to misclassify a given output; this is commonly known as a minimum adversarial example. A different method to verify an NN is to analyse the reachable sets of the NN's output given an input perturbation. Another metric used to analyse the input-output properties of an NN is to compute its Lipschitz constant, which is a measure of how much the output set changes with respect to a change in the input set. More recently researchers have evaluated the invertibility of NN's as another robustness measure. To determine these properties, most work in this area focuses on propagating the input uncertainties through

the network such that they are bound as tightly as possible by solving some form of optimisation problem. Usually it is the objective of the optimisation problem that is changed to determine the different robustness certificates. Note that these definitions vary between different works and the term 'verification' is commonly used to describe some measure on the input-output properties of the NN.

2.2.1 Neural Network Verification Problem

We now present the NN verification problem that is being addressed in this thesis. A multi-layer feed-forward NN can be described as a non-linear function $\pi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$, where n_u and n_y is the number of inputs and outputs respectively. Consider the set of all possible inputs into the NN $\mathcal{U} \subset \mathbb{R}^{n_u}$; the NN will map these inputs to a set of outputs $\mathcal{Y} \subset \mathbb{R}^{n_y}$

$$\mathcal{Y} = \pi(\mathcal{U}) := \{y \in \mathbb{R}^{n_y} \mid y = \pi(u),\ u \in \mathcal{U}\}.$$

The NN verification problem asks that the outputs of the NN lie within a safe region $\mathcal{S}_y$ given a set of inputs $\mathcal{U}$. Conversely, we could define the safe set of inputs $\mathcal{S}_u := \pi^{-1}(\mathcal{S}_y)$ as the inverse image of $\mathcal{S}_y$ under π. However, determining this inverse may not be possible. Note that there are no guarantees on the convexity of the $\mathcal{S}_y$ set; in fact it is likely that it will be non-convex. It is therefore computationally expensive to check if these outputs lie in a safe set. To overcome this issue a relaxation can be computed as a conservative approximation of the set $\mathcal{Y}$, denoted $\hat{\mathcal{Y}}$, and we instead check the condition $\hat{\mathcal{Y}} \subseteq \mathcal{S}_y$ to verify the NN. A diagram of this setup is shown in Figure 2.2.

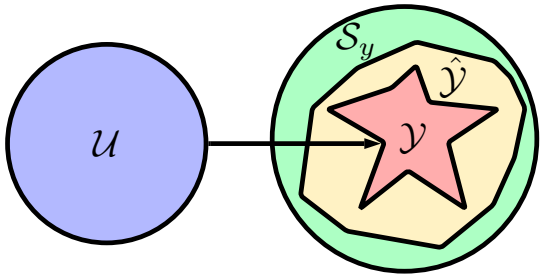


Figure 2.2: Diagram showing the neural network verification problem used in this thesis. $\mathcal{U}$ represents the input set, $\mathcal{Y}$ represents the output set and $\hat{\mathcal{Y}}$ is the approximation of this set. The safe set of outputs is denoted as $\mathcal{S}_y$.

Note that throughout this thesis we refer to analysing the robustness properties of NNs. As stated in the previous subsection, there are many notions of robustness when

considering NNs. In this thesis, robustness refers to the size of the output set, relative to the size of a given input set. Therefore, an NN that is considered robust is one in which the output set remains small given a large perturbation on an input.

2.2.2 Neural Network Model

We consider a feed-forward fully connected NN $\pi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$, with ℓ layers. This can be represented by the set of equations

$$\begin{aligned} x^0 &= u, \\ v^k &= W^k x^k + b^k, \text{ for } k = 0, \dots, \ell - 1, \\ x^{k+1} &= \phi(v^k), \text{ for } k = 0, \dots, \ell - 1, \\ \pi(u) &= W^\ell x^\ell + b^\ell, \end{aligned} \tag{2.6}$$

where $W^k \in \mathbb{R}^{n_{k+1} \times n_k}$, $b^k \in \mathbb{R}^{n_{k+1}}$ are the weights matrix and biases of the $(k+1)^{\text{th}}$ layer respectively and $u = x^0 \in \mathbb{R}^{n_u}$ is the input into the network. The number of neurons in the kth layer is denoted by n_k; the total number of neurons in the NN is therefore $n = \sum_{k=1}^{\ell} n_k$. The non-linear activation function ϕ is applied element-wise to the $v^k = W^k x^k + b^k$ terms such that

$$\phi(v^k) := [\phi(v_1^k), \dots, \phi(v_{n_k}^k)]^T, \; v^k \in \mathbb{R}^{n_k},$$

where ϕ is the activation function and v_j^k is the pre-activation value. There are many different types of activation functions such as ReLU, tanh, sigmoid, GELU, softsign and ELU [99]. In this thesis we will focus on the ReLU, sigmoid and tanh activation functions, however most of this work can be extended to any activation function. For the details of commonly used activation functions see Table 2.1 in Section 2.4.4. Note that in this thesis we refer to feed-forward fully connected NNs as NNs for conciseness.

2.2.3 Previous Literature on Neural Network Verification

The problem of verifying NNs over the past three years has proven to be an extremely competitive research field. There have been multiple competitions such as VNN-COMP [100–102], where state-of-the-art methods have been rigorously tested against each other. The most recent VNN-COMP was won by α-β-CROWN [103]. However, it is clear that all

methods have an inherent trade-off between conservatism and computational time. This is due to the non-linear activation functions in the NN, which make the problem non-convex. Many approaches rely on a convex over-approximation of this set, which unfortunately means that as the input set gets larger, the bounds on the output set become less tight.

One approach to provide these robustness certificates is to bound the non-linear activation functions in the NN [59], (this is the core idea that is used in this thesis). This method has proved successful and is a rapidly developing field, with a large number of results aiming to provide tighter and more efficient bounds on the NN performance.

The simplest methods are zeroth order methods such as interval bound propagation [104], that computes the worst-case scenarios out of each layer in the network. This approach is highly efficient but very provides very inaccurate bounds on the NN's output. Another comparable method is 'DeepZ' [105] that creates zonotope abstractions that are propagated through the network; [106] proposes an extended method which uses polynomial zonotopes. Frameworks that provide linear bounds on the activation functions will result in LPs, which can be solved using convex optimisation [59, 107, 108]. There are many ways in which these linear bounds can be imposed for different activation functions. A scalable approach that uses this LP framework is DeepSplit, which uses an operator splitting method to find the bounds [109]. [110] proposes Fast-lin that exploits the special structure of ReLU networks to solve the LP in closed form. CROWN [111] improves on this work by using a comparable method for general activation functions.

To improve the tightness of the linear relaxations, [112] introduces semidefinite relaxations for the certification of the robustness properties of NNs, and this is built upon in [113] through completely positive programming. The tightness of the semidefinite relaxation has been analysed in [114], to determine if the formulation is exact. One scalable method to the semidefinite relaxation is to use an iterative eigenvector approach to create an efficient algorithm to verify large networks [115]. Another semidefinite approach that provides tight bounds on the NN is built on the formulation of quadratic constraints [60, 116]. However, the SDP approach has the drawback of scaling worse than LPs. This structure has also been utilized in [117], where linear cuts are used to relax the SDP formulation. A Lagrangian relaxation was proposed for the SDP formulation

to allow for faster verification times [118]. These SDP approaches have been shown to analyse the robustness of different NN structures such as Monotone Deep Equilibrium Models [119]. The SDP framework has also been adapted to the probabilistic setting [120], which has been studied less in the literature.

Similar ideas can be used to find the Lipschitz constant for an NN using SDP [121]. This approach is extended using incremental quadratic constraints in [122]. Lipschitz bounds can be computed in a scalable way; both [123] and [124] use sparse polynomial optimisation to achieve this. The idea of Lipschitz constants have been extended to equilibrium NNs, which are a general class of NNs [125].

When ReLU activation functions are used, their piecewise linear natures allows for one to use Satisfiability Modulo Theory to quantify the safety of an NN [126, 127], however this is an NP-complete problem and not appropriate for moderately sized NNs. Mixed integer LP methods have also been a popular approach to verify NNs, although mostly they are only applicable to ReLU activation functions. The most notable examples of these methods are [128–130].

Alternative approaches exist such as works from [131] that use a dual approach to help with the scalability issue. 'DeepPoly' is an abstract interpretation-based approach for NN verification that is introduced in [132]. Another abstraction-based verification algorithm is 'AI^2' [133], which is effective on general NN architectures such as convolutional NNs. Moreover, in [134], the authors use Extreme Value Theory to better quantify the robustness. [135] proposes a new parametric framework called 'k-ReLU' that combines multiple activation functions together. 'PRIMA' is introduced by [136] to obtain convex abstractions for multiple neurons that use algorithms based on convex hull approximations. Similar work from [137] considers the multi-variate input space on the activation function to create tighter bounds for the LP. By treating the NN as a dynamical system, control theory tools have been used to obtain performance metrics on adversarial inputs [138]. Control theory methods have also been used in [139], where the NN training is framed as an optimal control problem, allowing the use of an augmented Lagrangian approach. By treating the NN as a dynamical system, [138] are able to obtain performance metrics on

adversarial inputs. [140] uses notions of stability in dynamical systems theory to diagnose and prevent instability in recurrent NNs for video processing.

In [141], the author introduces the idea of Propagators and Partitioners. The above methods are considered Propagator methods since they take a defined input set and form an optimisation problem around this set, effectively propagating that input set to the outputs. However, it is also possible to partition the input set to form multiple optimisation problems and take the union of all of the output sets. Partitioners separate the input set into smaller regions and solve the optimisation problem over each region. The motivation behind this is that a conservative yet efficient Propagator can be used with a Partitioner, having the potential to provide better bounds on the output set with less computational time than a more expensive Propagator without a Partitioner. There are many partitioning approaches; some examples are [142–146].

Another research question of interest is how to train an NN to some robustness specification. This problem is naturally a lot more challenging, however there are recent works that have addressed the problem. [147] proposes a robust optimization method by minimizing the worst-case loss over an outer approximated region through an LP. Another work uses an SDP relaxation and optimizes over the NN parameters as a form of robustness regularisation [148]. It has been shown that Lipschitz constants can be used to train the NN to be robust [149], using the SDP approach in [121]. These results have been expanded to train NNs under general semidefinite constraints [150] and also to recurrent NNs using dissipativity theory [151].

2.3 Analysing Neural Feedback Loops

Section 2.2 delineates the work that has analysed NNs as an input-output function. To clarify this is where the NN is used to map a set of inputs to a set of outputs, which are then processed in some way to extract information from the input set. This is sometimes referred to as the 'static case' or 'NNs in isolation'. Neural Feedback Loops (NFLs) involve the NN being used in a feedback system, where the outputs of the NN are processed and returned as inputs into the NN in a closed-loop system. NFLs have the potential to be

very useful in many industrial applications, however, there has been comparably less work in the analysis of the robustness of NFLs over the NN input-output case.

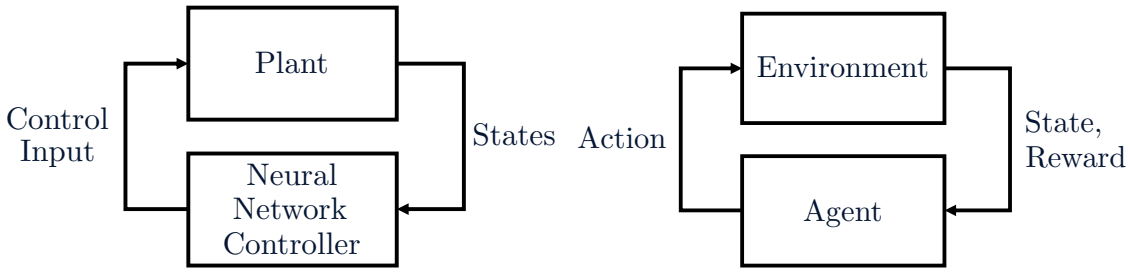
There have been many difficulties in obtaining robustness certificates on the input-output properties of NNs, however NFLs are even more challenging to analyse. This is due to the natural structure of NFLs, as the outputs of the NN components are intertwined with the inputs. This means that any uncertainty and undesirable sensitivity in the NN can be amplified around the loop. This is one of many challenges surrounding NFLs, [152–154] provide comprehensive reviews on the topic. A notable challenge from these reviews is the nature of combining a complex NN component with a complex system and the addition of large uncertainty in the model. The authors further comment on the long run-time of the algorithms that are currently available and the possibility that they may not be fast enough to be used in an online setting. Finally, since this is a relatively new research area there is a lack of benchmarking and standardization between approaches, which is currently being addressed by the research community.

Like the general NN verification problem, there are many possible robustness metrics that can be analysed for NFLs. The most common certificates placed on NFLs in the literature are computing forward and backward reachable sets and determining the stability of the system subject to a region of attraction. These measures are defined in Chapters 6 and 7.

2.3.1 Neural Feedback Loop Model

An NFL is a closed-loop model that contains NNs as components. There is therefore not a singular structure to describe an NFL and they will be defined differently in numerous works in the literature. The NFL model that we consider in this thesis is an NN controller that is connected in feedback to the plant, which may be subject to some uncertainty. NFLs that contain a non-linear plant model are referred to as non-linear NFLs. This is depicted in Figure 2.3a. The equations of the NFL for the discrete-time and continuous-time cases are presented in Chapter 6 and 7 respectively. Note that NFLs are also referred to as neural feedback systems, NN controlled systems and cyber-physical systems. However this is not an exhaustive list and other naming conventions have been used.

Note that there are two general paradigms that have emerged in the literature. There is the control theoretic perspective that refers to NFLs and views the system as a model based controller in a traditional control setting - this is what is used in this thesis. However, there is also the reinforcement learning perspective. This involves an agent/policy performing actions on an environment and determining its decision-making via state and reward inputs, shown in Figure 2.3b. This is usually a data based approach and it is hard to obtain robustness certificates on the system. Despite this, it has shown to be a hugely promising field of machine learning.



(a) Neural Feedback Loop Model **(b)** Reinforcement Learning Model

Figure 2.3: Comparison of the neural feedback loop model used in this thesis and a reinforcement learning framework.

2.3.2 Previous Literature on Analysing Neural Feedback Loops

As NN verification has been increasingly widespread, so has the analysis of NFLs. Competitions in recent years such as ARCH-COMP have been tasked with the testing and verification of multiple forms of hybrid systems [155–158].

Reachability analysis has commonly been applied to NFLs, where forward reachable sets are approximated at future time steps. These approximate reachable sets can be used to determine whether the NFL will reach a given goal set, whilst avoiding certain undesirable regions. Many recent works such as [159–162] address this problem by placing bounds on the non-linear activation functions in the NN controller. These bounds can be too loose, meaning that it is not possible to verify the initial set if it becomes too large. Another issue is that these methods often assume that a linear plant is being controlled, which is restrictive when the plant has non-linear dynamics. By partitioning the input space efficiently and solving multiple optimisation problems, [163] computes tighter bounds on the reachable sets in significantly reduced computational time. This method scales well

to large NNs and has also been extended to compute backward reachable sets. Similar methods have been shown to be effective in [164]. LP methods have been expanded by exploiting the structure of the ReLU function by using constrained zonotopes to obtain an accurate and efficient method [165]. SDP formulations have been used to obtain tighter bounds on the reachable sets [161]; this has been extended to an one-shot reachability method [166], to compute the reachable sets at future time steps without using recursion.

A simulation guided approach is proposed in [167] to adapt the simulations generated from the output of the NN to approximate the reachable sets. Bernstein polynomials are used to verify NNs with activation functions of a general form to effectively compute the reachable sets [168, 169]. A Taylor model is introduced in [170] to compute flowpipe overapproximations for the reachable sets. By focusing on NNs with sigmoid activation functions, [171] exploits the structure of the sigmoid function to transform the NN into a hybrid system, which is solved using state-of-the-art reachability methods for hybrid systems. Furthermore, successful branch and bounds approaches used in the NN verification problem have been applied to NFLs, [172] uses Lipschtiz bounds to achieve this. There have been few works that have focused on stochastic NFLs, however [173] uses a data-driven framework to address this.

Stability is a popular property to analyse control feedback systems and has recently been applied to NFLs. Initial works to quantify the safety of an NFL include [174], which looks at the Lyapunov stability using linear differential inclusions. More recent works have looked at absolute stability by using a transformation into non-linear operator form [175]. By using a reinforcement learning perspective, [176] were able to get robust stability guarantees. Asymptotic stability is computed along with a region of attraction in [177] using Lyapunov stability theory in an SDP-based approach, with local sector constraints on the non-linearities. This is taken further by using integral quadratic constraints to bound perturbations in the plant model. A similar approach is used in [178] for recurrent NNs. Another approach that uses integral quadratic constraints does so in conjunction with the circle criterion and Zames-Falb multipliers [179] to obtain better stability guarantees and larger regions of attraction. Polynomial optimisation techniques have been used in [180] to analyse stability properties of simple NFLs. Another method was proposed in

[181], where the authors use output range analysis to compute the stability certificates. Alternative approaches have shown that linear complementarity systems with ReLU NN controllers can be cast as equivalent stability LMI constraints [182]. There are additional methods that use ideas from robust control theory, for example [183] utilises these tools to certify an NN control policy based on finding a positively invariant set of the system.

By leveraging these robustness measures, recent work has targeted the problem of designing NN controllers with safety guarantees. [184] develops a certifiably robust deep reinforcement learning algorithm online by computing an action under the worst case deviation that is computed. The reinforcement learning framework is combined with robust control guarantees in [185], such that the NN policy is always stable. It is possible to use the SDP framework with integral quadratic constraints and imitation learning [186], by training an NN controller with stability guarantees. These ideas can be applied to recurrent NNs as shown in [187] and also to recurrent equilibrium network controllers [188]. Works from [189–191] propose methods to bound the worst-case error between a model predictive control law and an NN. By using a system level synthesis approach, [192] approximates a rich set of nonlinear controllers by a recurrent equilibrium network, which can be learnt in a convex way. [193] also uses this NN structure along with contraction theory to design models with built in stability guarantees. It is also possible to design NNs for perception in control loops, [194] shows how this is achieved whilst ensuring stability guarantees. [195] synthesises an NN controller simultaneously with a neural Lyapunov function to certify the stability of the feedback system; a similar method is proposed in [196].

2.4 Abstracting Neural Networks as Constraints

We now describe how constraints that approximate the input-output map of the NN can be abstracted. The constraints can be split into three main categories, by considering the input, output and hidden layer constraints separately. We present details on the constraints in this thesis, but note that this is not an exhaustive list of the constraints that can be formed nor which have been used in the literature.

2.4.1 Input Constraints

There are a large number of different sets that can be used to describe the input into the NN. For the majority of this thesis we assume that the input constraints are bounded by a hyper-rectangle defined by $\mathcal{U} = \{u \in \mathbb{R}^{n_u} \mid \underline{u} \leq u \leq \overline{u}\}$. Therefore, the input constraints can be written as

$$g_{\text{in}}^1 = u - \underline{u} \geq 0,\ g_{\text{in}}^2 = -u + \overline{u} \geq 0. \tag{2.7}$$

These can be represented as linear constraints in u or as quadratic constraints as shown in [60]. This is often the simplest input specification set that can be used, however other examples include polytopes, zonotopes, ellipsiods and star sets.

2.4.2 Output Constraints

To formulate the output constraints, we assume that the safe set can be represented by the polytope

$$\mathcal{S}_y = \bigcap_{m=1}^{M} \{y \in \mathbb{R}^{n_y} \mid c_m^T y - d_m \leq 0\}, \tag{2.8}$$

where $c_m \in \mathbb{R}^{n_y}$ and $d_m \in \mathbb{R}$ are given parameters of the polytope. A polytope is a common choice for the output constraints since it is composed of linear functions. Therefore, if our goal is to minimise the output set, then the objective of the optimisation problem will also be a linear function. It is also possible to express the output set as other shapes such as an ellipse using quadratic constraints, but this often requires the making of some convex approximation.

2.4.3 Hidden Layer Constraints

To abstract the hidden layers as constraints, bounds on the non-linear activation functions are created. There are many different ways that this can be achieved, with some designed for LP or SDP frameworks. These constraints are formed through many means: popular methods include box, linear, sector and slope constraints, which we will summarise below. However, there is no limitation of how these can be expressed.

The hidden layer constraints can be represented by relationships between $\phi(v^k)$ and v^k, which can be expressed through properties of the activation function. Since the

activation functions are applied element-wise to the v^k terms, we consider the relationship between $\phi(v_j^k)$ and v_j^k separately. To simplify the notation, we denote the output of the activation function $\phi(v_j^k)$ as ϕ, and the input to the activation function v_j^k as x, and drop the dependence of ϕ on x.

Definition 2.16. Consider the non-linear activation function $\phi(x)$. The minimum and maximum bounds on the activation function $(\underline{\phi}, \overline{\phi})$ can be bounded by a *box constraint* such that

$$\phi(x) - \underline{\phi} \geq 0, \; -\phi(x) + \overline{\phi} \geq 0. \tag{2.9}$$

These are often computed using an efficient preprocessing step known as interval bound propagation (IBP) [104]. IBP is a zeroth order method, which uses interval arithmetic by propagating a simple polytope through the NN. The pre-activation values from IBP are denoted as $(\underline{x}, \overline{x})$. Other approaches exist such as CROWN and DeepPoly [103, 132], that can provide better bounds but can take longer to compute. These box constraints are used in the optimisation framework and there are many different possible constraints that can be used and combined with the preprocessing values. By obtaining approximate maximum and minimum values on the activation functions, the preprocessing is often a vital step to abstract the NN as constraints. This is because the preprocessing values are used to create tighter bounds on the activation function, as shown later in this section.

Linear bounds can be used to define a region that the activation function must reside in. These bounds result in LPs and are popular with larger NNs as they can result in efficient programs.

Definition 2.17. Consider the non-linear activation function $\phi(x)$. A linear constraint states that $\phi(x)$ lies in the region defined by α and λ if the following condition holds

$$\phi(x) - (\alpha x + \lambda) \geq 0, \; \forall x \in \mathbb{R}.$$

There are many ways that these linear constraints can be used for the different activation functions. Often multiple linear bounds are defined to tighten the region.

Within the field of NN verification, sector constraints are commonly used to bound the activation functions in the NN. A sector constraint is shown in Figure 2.4a and defined below.

Definition 2.18. Consider the non-linear activation function $\phi(x)$ with $\phi(0) = \lambda$. A sector constraint states that $\phi(x)$ lies in the sector $[\alpha, \beta]$ $(\alpha \leq \beta < \infty)$ if the following condition holds

$$(\phi(x) - (\alpha x + \lambda))((\beta x + \lambda) - \phi(x)) \geq 0, \, \forall \, x \in \mathbb{R}.$$

Another set of constraints that can be used to bound the activation functions are obtained by considering the slope of the activation function. These are visualised in Figure 2.4b. Since the activation functions have a predefined structure, their slopes are often bounded between two values, which can be used to form a sector constraint. This method was first employed in [60].

Definition 2.19. Consider the non-linear activation function $\phi(x)$. A slope constraint states that $\phi(x_1) - \phi(x_2)$ lies in the sector $[\alpha, \beta]$ $(\alpha \leq \beta < \infty)$ if the following condition holds

$$\alpha \leq \frac{\phi(x_2) - \phi(x_1)}{x_2 - x_1} \leq \beta, \, \forall x_1, x_2 \in \mathbb{R}.$$

By considering the slope of two activation functions from different layers, it can be shown that any two nodes in the NN must satisfy

$$(\phi(x_i) - \phi(x_j) - \alpha(x_i - x_j))(\beta(x_i - x_j) - (\phi(x_i) - \phi(x_j)) \geq 0,$$

$\forall i, j = 1, \ldots, n, \; i \neq j$, where α and β are determined by the activation function that is being used [60]. However, the big issue with constraints of this type is the lack of scalability. As the number of neurons in the network increases, the number of constraints increases with order $\binom{n}{2}$.

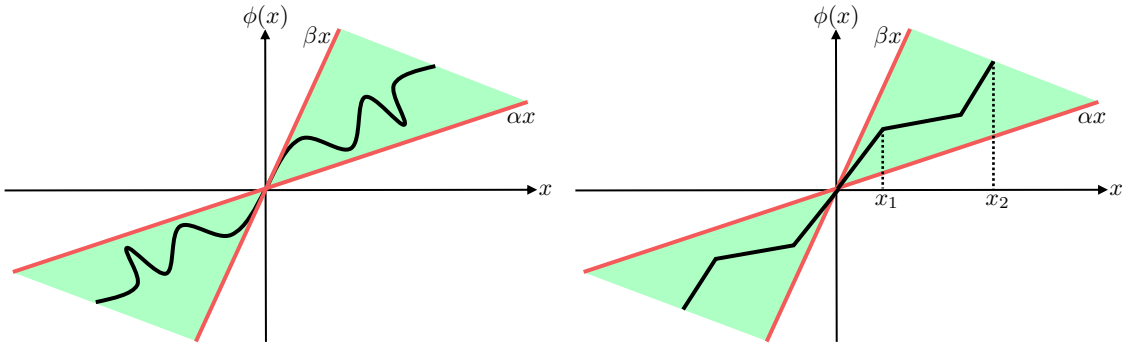
We now explain how these constraints can be used to represent different activation functions.

Example 2.20. [ReLU function] ReLU is a commonly used function and is given by

$$\text{ReLU}(x) = \phi(x) = \begin{cases} 0 & x \leq 0, \\ x & x > 0. \end{cases}$$

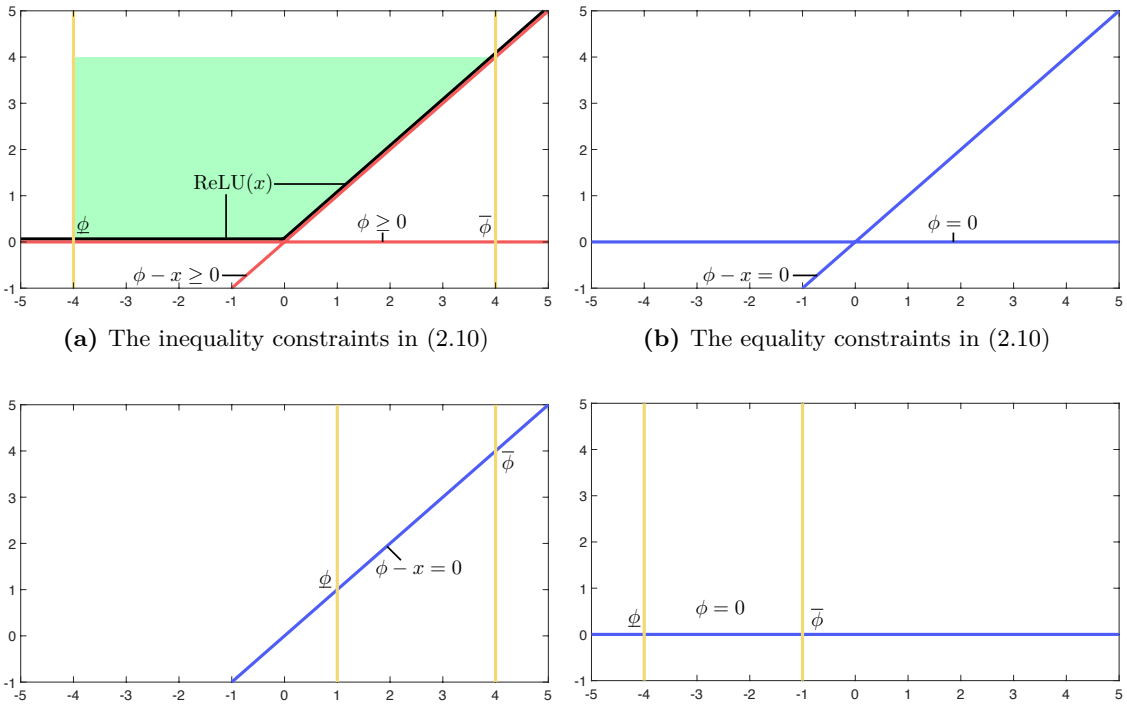
We can gain tight bounds on the ReLU function using two inequalities and one equality constraint [112] such that

$$\phi \geq 0, \; \phi - x \geq 0, \; \phi(\phi - x) = 0. \tag{2.10}$$



(a) Sector constraints in Definition (2.18) **(b)** Slope constraints in Definition (2.19)

Figure 2.4: Plot showing a sector and slope constraint for a general non-linearity $\phi(x)$.



(a) The inequality constraints in (2.10) **(b)** The equality constraints in (2.10)

(c) The equality constraint in (2.10) when $\underline{\phi} > 0$ **(d)** The equality constraint in (2.10) when $\overline{\phi} \leq 0$

Figure 2.5: Plot showing the inequality constraints in (2.10) that bound the ReLU function (black). The red lines represents the inequality constraints, the blue lines represents the equality constraint and the yellow lines represent the constraints from the IBP values. The green shaded area is the region bounded by the constraints.

The values from the IBP can be used to further tighten these constraints: if $\overline{\phi} \leq 0$, then we can replace the first inequality constraint with an equality constraint such that $\phi = 0$. Equivalently if $\underline{\phi} > 0$, we can replace the second inequality constraint with $\phi - x = 0$. These constraints are shown graphically in Figure 2.5a – 2.5d.

There are multiple ways that linear bounds can be used on the ReLU function, one example is depicted in Figure 2.6, where the region is defined by a triangular area such

that

$$\phi \geq 0,\ \phi - x \geq 0,\ \frac{\overline{\phi} - \underline{\phi}}{\overline{x} - \underline{x}}(x - \underline{x}) + \underline{\phi} - \phi \geq 0. \tag{2.11}$$

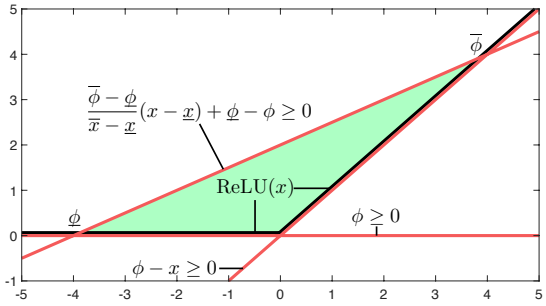


Figure 2.6: Visualisation of the triangular constraints on the ReLU function (black) described by (2.11). The red lines represents the inequality constraints and the green shaded area is the region bounded by the constraints.

Example 2.21 (Sigmoid and Tanh Functions)**.** The sigmoid and tanh functions are given by and represented by a sector constraint such that

$$\text{sig}(x) = \phi(x) = \frac{1}{1 + e^{-x}} \implies (\phi - 0.5)(0.25x + 0.5 - \phi) \geq 0, \tag{2.12}$$

$$\tanh(x) = \phi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \implies \phi(x - \phi) \geq 0, \tag{2.13}$$

respectively. However, this bound is very conservative as there is a large uncertainty in the value of the function. [177] showed that this constraint can be tightened by using the preprocessing values. This is achieved by using the following conditions for the sigmoid function

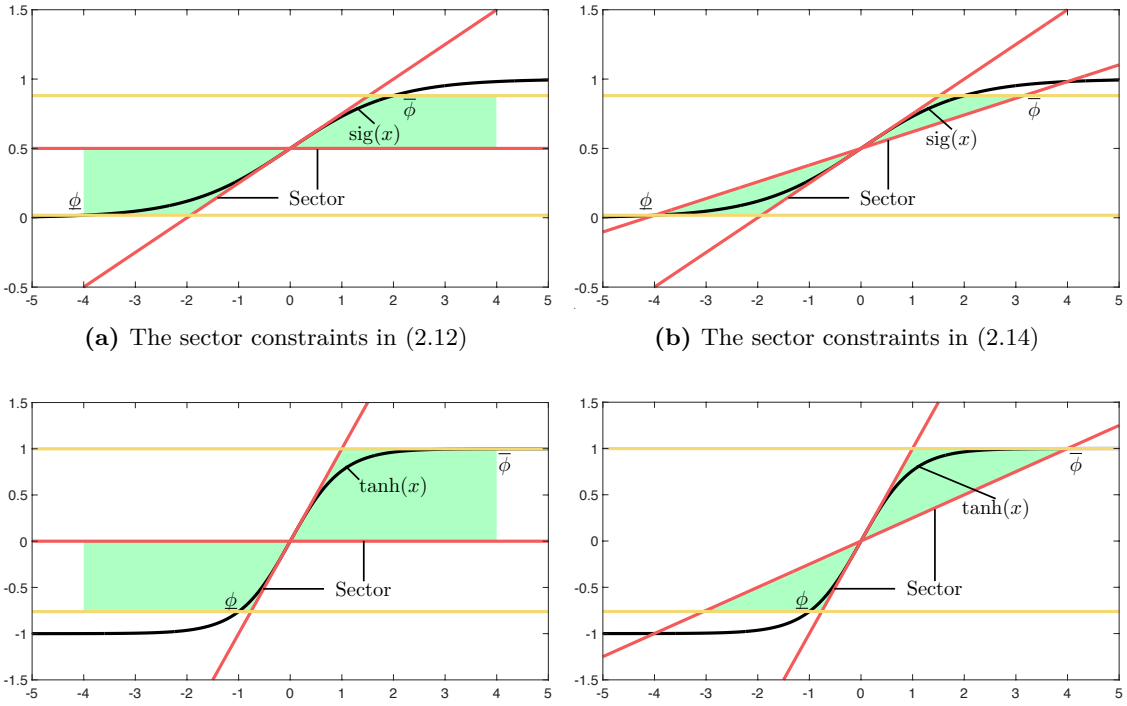
$$\begin{aligned} \left(\phi - \left(\frac{\overline{\phi} - 0.5}{\overline{x}}\right)x - 0.5\right)(0.25x + 0.5 - \phi) \geq 0, \text{ if } |\overline{\phi}| \geq |\underline{\phi}|, \\ \left(\phi - \left(\frac{\underline{\phi} - 0.5}{\underline{x}}\right)x - 0.5\right)(0.25x + 0.5 - \phi) \geq 0, \text{ if } |\overline{\phi}| \leq |\underline{\phi}|, \end{aligned} \tag{2.14}$$

and the following conditions for the tanh function

$$\begin{aligned} \left(\phi - \frac{\overline{\phi}}{\overline{x}}x\right)(x - \phi) \geq 0, \text{ if } |\overline{\phi}| \geq |\underline{\phi}|, \\ \left(\phi - \frac{\underline{\phi}}{\underline{x}}x\right)(x - \phi) \geq 0, \text{ if } |\overline{\phi}| \leq |\underline{\phi}|. \end{aligned} \tag{2.15}$$

A comparison of these constraints is shown in Figure 2.7.

Both sigmoid and tanh functions can instead be described by two linear constraints that provide upper and lower bounds on the activation function. These constraints are



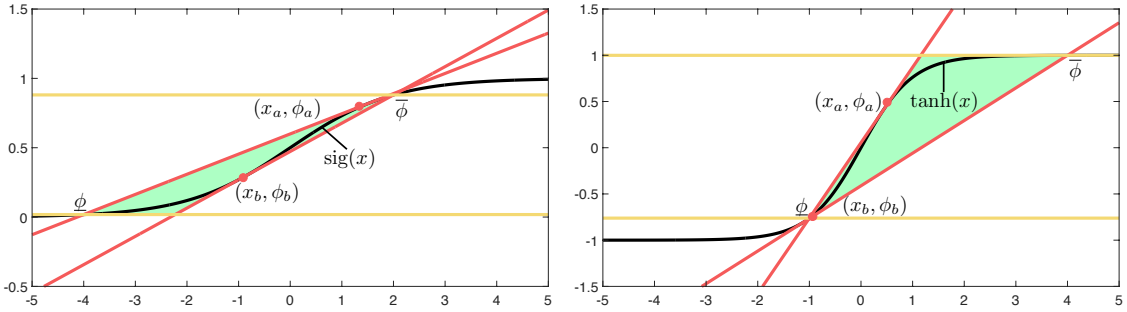
(a) The sector constraints in (2.12)

(b) The sector constraints in (2.14)

(c) The sector constraints in (2.13)

(d) The sector constraints in (2.15)

Figure 2.7: Plot showing a comparison of the sector constraints that bound the sigmoid and tanh functions (black). The red lines represents the sector inequality constraints and the yellow lines represent the constraints from the IBP values. The green shaded area is the region bounded by the constraints.



(a) The constraints in (2.16) for the sigmoid function when $|\overline{\phi}| \leq |\underline{\phi}|$

(b) The constraints in (2.16) for the tanh function when $|\overline{\phi}| \geq |\underline{\phi}|$

Figure 2.8: Visualisation of the triangular constraints described by (2.16). The red lines represents the inequality constraints and the green shaded area is the region bounded by the constraints.

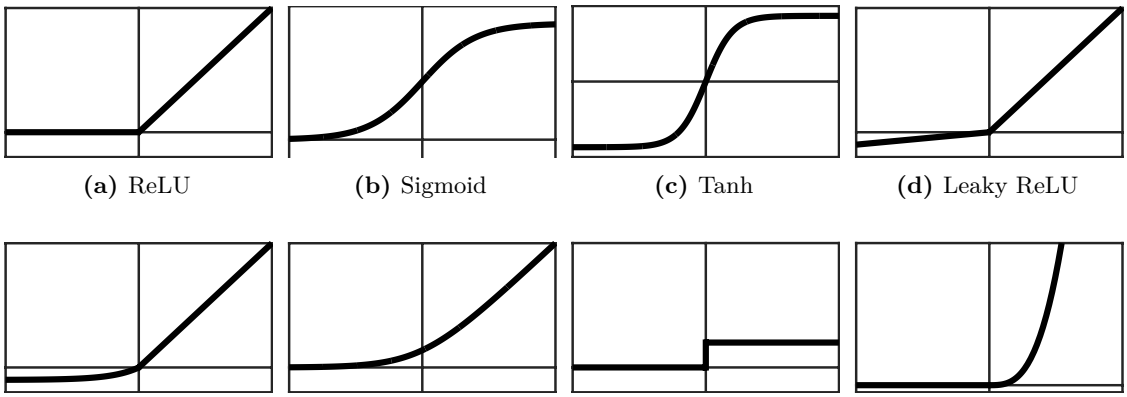
shown in Figure 2.8 and can be written as

$$\left(\frac{\underline{\phi}-\phi_a}{\underline{x}-x_a}\right)(\underline{x}-x)+\underline{\phi}-\phi \geq 0,\ \phi+\left(\frac{\overline{\phi}-\phi_b}{\overline{x}-x_b}\right)(\overline{x}-x)-\overline{\phi} \geq 0, \tag{2.16}$$

where (x_a, ϕ_a) and (x_b, ϕ_b) are points that are tangential to the sigmoid and tanh curves.

2.4.4 General Activation Functions in Neural Networks

There are a huge number of activation functions that have been used in the literature. For a comprehensive survey for activation functions in NNs the reader is referred to [99, 197]. Table 2.1 provides a summary of commonly used activation functions along with their respective slope and sector constraints. Plots of these activation functions are shown in Figure 2.9. Note the limitations of this framework since binary and RePU functions cannot be expressed as slope or sector constraints.



(a) ReLU **(b)** Sigmoid **(c)** Tanh **(d)** Leaky ReLU

(e) ELU **(f)** Softplus **(g)** Binary **(h)** RePU

Figure 2.9: Plots of common activation functions.

Activation Function ($\phi(x)$)	**Sector Bound** $[\alpha, \beta]$	**Slope Bound** $[\alpha, \beta]$
$\text{ReLU}(x) = \begin{cases} 0 & x \leq 0, \\ x & x > 0 \end{cases}$	$[0, 1]$	$[0, 1]$
$\text{sigmoid}(x) = \dfrac{1}{1 + e^{-x}}$	$[0, 0.25]$	$[0, 0.25]$
$\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	$[0, 1]$	$[0, 1]$
$\text{Leaky ReLU}(x) = \begin{cases} ax & x \leq 0, \\ x & x > 0 \end{cases}, \ a > 0$	$[\min(a, 1), \max(a, 1)]$	$[\min(a, 1), \max(a, 1)]$
$\text{ELU}(x) = \begin{cases} a(e^{x} - 1) & x \leq 0, \\ x & x > 0 \end{cases}, \ a > 0$	$[0, 1]$	$[0, 1]$
$\text{Softplus}(x) = \ln(1 + e^{x})$	$[0, 1]$	$[0, 1]$
$\text{Binary}(x) = \begin{cases} 0 & x \leq 0, \\ 1 & x > 0 \end{cases}$	N/A	N/A
$\text{RePU}(x) = \begin{cases} 0 & x \leq 0, \\ x^{s} & x > 0 \end{cases}, \ s \geq 2$	N/A	N/A

Table 2.1: Common activation functions with their respective slope and sector bounds.

3

Exploiting Sparsity for Neural Network Verification

This chapter presents a semidefinite programming framework that has been successful in verifying neural networks. The semidefinite constraints in the neural network verification problem are analysed and then it is shown how a sparse matrix decomposition can be applied to this formulation. Results are presented to demonstrate how this technique can be used to significantly speed up the computational time to solve the optimisation problem.

3.1 Introduction

SDP frameworks have shown to be successful when applied to the NN verification problem [112, 116]. Their ability to encapsulate quadratic constraints can overcome the issue with linear bounds being less tight. However, a big issue with these methods is the limit on their scalability, since SDPs scale poorly when the problem data becomes larger.

There have been previous attempts to speed up the computational time, for example [115] uses an iterative eigenvector approach to allow for better scaling. However, the accuracy of the bounds are sacrificed. The approach in this thesis improves the scalability of the SDP by exploiting the sparsity in the constraint matrices, in particular chordal sparsity. Chordal sparsity has been used in many SDP-related problems such as finding Lyapunov functions [198] and the scalable design of structured controllers [199]. Since NNs have a natural cascading structure it is possible to use ideas from chordal sparsity to dramatically reduce the computational time to solve these NN verification problems.

The work in this thesis is the first to observe chordal sparsity in the constraint matrices of the SDP framework. However, there have been further works in the literature that have expanded on this idea. For example, [117] uses linear cuts between the layers

of the network to relax the problem, whilst using chordal sparsity to reduce the solve time. This work was extended in [200] by using a reformulated linearisation technique to produce a novel SDP relaxation. Our chordal sparsity idea was expanded in [201] by using a parametric chordal sparsity approach to verify NNs, allowing one to tune the level of sparsity exploitation in the problem.

Note that in this chapter, we will only focus on the ReLU activation function. However, it is possible to expand this sparsity exploiting method to other activation functions.

3.1.1 Neural Network Equations in Matrix Form

The feed-forward fully connected NN in (2.6) can be written in matrix form. We denote the vector of all the neurons as a concatenation of the respective activations such that $\boldsymbol{x} = [(x^0)^T, \dots, (x^\ell)^T]^T \in \mathbb{R}^{n_0+n}$. Each layer can be described by the equation $x^k = \boldsymbol{E}^k \boldsymbol{x}$, $\forall k = 0, \dots, \ell$, where $\boldsymbol{E}^k \in \mathbb{R}^{n_k \times (n_0+n)}$ is a matrix that extracts the relevant layer. The NN can then be written in matrix form as

$$\begin{aligned} u &= \boldsymbol{E}^0 \boldsymbol{x}, \\ \boldsymbol{B}\boldsymbol{x} &= \phi(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}), \\ \pi(u) &= W^\ell \boldsymbol{E}^\ell \boldsymbol{x} + b^\ell, \end{aligned} \tag{3.1}$$

where

$$\boldsymbol{A} = \begin{bmatrix} W^0 & 0 & \dots & 0 & 0 \\ 0 & W^1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & W^{l-1} & 0 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b^0 \\ b^1 \\ \vdots \\ b^{l-1} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 & I_{n_1} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I_{n_{l-1}} & 0 \\ 0 & 0 & \dots & 0 & I_{n_l} \end{bmatrix}.$$

3.2 Neural Network Verification as Quadratic Constraints

The goal in this section is to show how to represent the constraints in the NN verification problem as quadratic constraints. This will allow the problem to be converted into an SDP. This is achieved by using the S-procedure as stated in Lemma 2.12 to concatenate the quadratic equations and represent them in matrix form. To demonstrate this we define a framework from [60].

Definition 3.1. Let $\mathcal{X} \subset \mathbb{R}^{n_x}$ be a nonempty set. Suppose $\mathcal{P}_\mathcal{X}$ is the set of all symmetric indefinite matrices P such that

$$\begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0, \quad \forall x \in \mathcal{X}.$$

We say that $\mathcal{X}$ satisfies the quadratic constraint defined by $\mathcal{P}_\mathcal{X}$.

Definition 3.2. Let $\mathcal{X} \subset \mathbb{R}^{n_x}$ be a nonempty set. Consider the function $\phi(x)$ and suppose $\mathcal{Q}_\phi \subset \mathbb{S}^{2n+1}$ is the set of all symmetric indefinite matrices Q such that

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T Q \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0, \quad \forall x \in \mathcal{X}.$$

Then $\phi(x)$ satisfies the quadratic constraints defined by $\mathcal{Q}_\phi$ on $\mathcal{X}$.

3.2.1 Input Constraints

Proposition 3.3. ([60]) The input set bounded by a hyper-rectangle in (2.7) satisfies the quadratic constraint defined by

$$\mathcal{P}_\mathcal{U} = \left\{ P \;\middle|\; P = \begin{bmatrix} -2\Gamma & \Gamma(\underline{u} + \overline{u}) \\ (\underline{u} + \overline{u})^T\Gamma & -2\underline{u}^T\Gamma\overline{u} \end{bmatrix} \right\},$$

where $\Gamma \in \mathbb{R}^{n_u \times n_u}$ is diagonal and nonnegative.

This quadratic constraint framework can be expanded to other sets such as polytopes and ellipsoids, as shown below.

Proposition 3.4. ([60]) The polytope $\mathcal{U} = \{u \in \mathbb{R}^{n_u} \mid Hu \leq h\}$ where $H \in \mathbb{R}^{m \times n_u}$ and $h \in \mathbb{R}^m$ satisfies the quadratic constraint defined by

$$\mathcal{P}_\mathcal{U} = \left\{ P \;\middle|\; P = \begin{bmatrix} H^T\Gamma H & -H^T\Gamma h \\ -h^T\Gamma H & h^T\Gamma h \end{bmatrix} \right\},$$

where $\Gamma \in \mathbb{S}^m_+$ and $\Gamma_{ii} = 0$, $\forall i = 1, \ldots, m$.

Proposition 3.5. ([60]) The ellipsoid $\mathcal{U} = \{u \in \mathbb{R}^{n_u} \mid ||Au + b||_2 \leq 1\}$ where $A \in \mathbb{S}^{n_u}$, $b \in \mathbb{R}^{n_u}$ and $||\cdot||$ is the matrix two-norm satisfies the quadratic constraint defined by

$$\mathcal{P}_\mathcal{U} = \left\{ P \;\middle|\; P = \mu \begin{bmatrix} -A^TA & -A^Tb \\ -b^TA & 1 - b^Tb \end{bmatrix},\ \mu \geq 0 \right\}.$$

There are limitations to the sets that quadratic constraints can define, meaning that other frameworks are required to incorporate alternative constraints into the problem.

3.2.2 Output Constraints

The output constraints are formulated as the polytope described by (2.8) since its volume can be minimised easily in the SDP framework. However, it is possible to use other output specification sets.

Proposition 3.6. ([60]) The safe set is bounded by a polytope defined by (2.8) which can be written as the intersection of quadratic constraints such that

$$\mathcal{S}_y = \bigcap_{m=1}^{M} \left\{ u \in \mathbb{R}^{n_u} \;\middle|\; \begin{bmatrix} u \\ \pi(u) \\ 1 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & c_m \\ 0 & c_m^T & -2d_m \end{bmatrix} \begin{bmatrix} u \\ \pi(u) \\ 1 \end{bmatrix} \leq 0 \right\}.$$

3.2.3 Hidden Layer Constraints

To verify the NN, we can bound the activation function using a set of quadratic equality and inequality constraints from [60]. In the full formulation there are slope constraints that consider the relationship between nodes. However, if these constraints are incorporated then the chordal structure is broken, so we will not use them in this formulation. In the examples that we tested we noticed that not using slope constraints did not have a significant impact on the accuracy and still provided robustness guarantees on the network, whilst greatly improving the scalability of the problem. We now describe how the SDP in [60] is formulated.

To form the SDP using quadratic constraints on the activation functions, the sector and box constraints in Definition 2.18 and Definition 2.16 respectively must be vectorised. This is achieved through the following propositions.

Proposition 3.7. ([60]) Let $K_1, K_2 \in \mathbb{R}^{n \times n}$ be two matricies such that $K_2 - K_1 \in \mathbb{S}_+^n$. The function $\phi(x)$ is bounded in the sector $[K_1, K_2]$ if

$$(\phi(x) - K_1 x)^T (\phi(x) - K_2 x) \leq 0, \; \forall x \in \mathbb{R}^n,$$

or equivalently in matrix form

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T \begin{bmatrix} -K_1^T K_2 - K_2^T K_1 & K_1^T + K_2^T & 0 \\ K_1 + K_2 & -2I_n & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0.$$

Proposition 3.8. ([60]) Let $\underline{\phi}, \overline{\phi} \in \mathbb{R}^n$ be vectors that upper and lower bound a function $\phi(x)$. This function is bounded by the box constraints in Definition 2.16 if

$$(\phi(x) - \underline{\phi})^T(\overline{\phi} - \phi(x)) \geq 0,$$

or equivalently in matrix form

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2D & D(\underline{\phi} + \overline{\phi}) \\ 0 & (\underline{\phi} + \overline{\phi})^T D & -2\underline{\phi}^T D\overline{\phi}) \end{bmatrix} \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0.$$

where $D \in \mathbb{R}^{n \times n}$ is diagonal and nonnegative.

Using Proposition 3.7 and Proposition 3.8 we can express a collection of the quadratic representation of the ReLU function (2.10) in matrix form. This is shown in the following proposition.

Proposition 3.9. ([60]) Consider the NN in (3.1) with ReLU activation functions in (2.10), then $\phi(x)$ satisfies the quadratic constraint

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{12}^T & Q_{22} & Q_{23} \\ Q_{13}^T & Q_{23}^T & Q_{33} \end{bmatrix} \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0,$$

where

$$Q_{11} = -2\text{diag}(\boldsymbol{\alpha} \circ \boldsymbol{\beta} \circ \lambda), \quad Q_{12} = \text{diag}((\boldsymbol{\alpha} + \boldsymbol{\beta}) \circ \lambda), \quad Q_{13} = -\boldsymbol{\beta} \circ \nu - \boldsymbol{\alpha} \circ \eta,$$
$$Q_{22} = 0, \quad Q_{23} = \nu + \eta, \quad Q_{33} = 0,$$

with

$$\boldsymbol{\alpha} = [\mathbf{1}_{\mathcal{I}^+}(1), \ldots, \mathbf{1}_{\mathcal{I}^+}(n)], \quad \boldsymbol{\beta} = [1 + \mathbf{1}_{\mathcal{I}^-}(1), \ldots, 1 + \mathbf{1}_{\mathcal{I}^-}(n)],$$
$$\nu_i \geq 0, \text{ for } i \notin \mathcal{I}^+, \quad \eta_i \geq 0, \text{ for } i \notin \mathcal{I}^-, \quad \lambda_i \in \mathbb{R},$$

where $\mathbf{1}_{\cdot}(\cdot)$ is the indicator function and

$$\mathcal{I}^+ = \{i \mid x_i \geq 0, \ \forall x \in \mathcal{X}\},$$
$$\mathcal{I}^- = \{i \mid x_i < 0, \ \forall x \in \mathcal{X}\}.$$

3.3 Neural Network Verification as a Semidefinite Program

The quadratic constraints in Section 3.2 can be combined together, allowing the verification problem to be posed as an SDP constraint using the S-procedure.

Theorem 3.10. ([60]) Consider an NN given by (3.1) and the LMI

$$M_{\text{in}}(P) + M_{\text{mid}}(Q) + M_{\text{out}}(S_i) \preceq 0, \tag{3.2}$$

where

$$M_{\text{in}}(P) = \begin{bmatrix} \boldsymbol{E}^0 & 0 \\ 0 & 1 \end{bmatrix}^T P \begin{bmatrix} \boldsymbol{E}^0 & 0 \\ 0 & 1 \end{bmatrix}, \qquad M_{\text{mid}}(Q) = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{B} & 0 \\ 0 & 1 \end{bmatrix}^T Q \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{B} & 0 \\ 0 & 1 \end{bmatrix},$$

$$M_{\text{out}}(S_i) = \begin{bmatrix} \boldsymbol{E}^0 & 0 \\ W^l \boldsymbol{E}^l & b^l \\ 0 & 1 \end{bmatrix}^T S_i \begin{bmatrix} \boldsymbol{E}^0 & 0 \\ W^l \boldsymbol{E}^l & b^l \\ 0 & 1 \end{bmatrix},$$

with

$$P = \begin{bmatrix} -2\Gamma & \Gamma(\underline{u} + \overline{u}) \\ (\underline{u} + \overline{u})^T \Gamma & -2\underline{u}^T \Gamma \overline{u} \end{bmatrix}, \quad Q = \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{12}^T & Q_{22} & Q_{23} \\ Q_{13}^T & Q_{23}^T & Q_{33} \end{bmatrix}, \quad S_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & c_m \\ 0 & c_m^T & -2d_m \end{bmatrix},$$

as parameterised in Proposition 3.3, Proposition 3.9 and Proposition 3.6 respectively. If (3.2) is feasible then

$$\begin{bmatrix} u \\ \pi(u) \\ 1 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & c_m \\ 0 & c_m^T & -2d_m \end{bmatrix} \begin{bmatrix} u \\ \pi(u) \\ 1 \end{bmatrix} \leq 0, \ \forall x \in \mathcal{X}.$$

We can then optimise over the SDP constraint in Theorem 3.10 by minimizing the volume of the polytope. The optimisation problem can be expressed as

$$\begin{aligned} \text{minimize} \quad & d_i \\ \text{subject to} \quad & M_{\text{in}}(P) + M_{\text{mid}}(Q) + M_{\text{out}}(S_i) \preceq 0, \\ & (P, Q, S_i) \in \mathcal{P}_{\mathcal{U}} \times \mathcal{Q}_\phi \times \mathbb{S}^{n_u + n_y + 1}. \end{aligned} \tag{3.3}$$

3.4 Chordal Analysis of Constraints

In this section we investigate the sparsity pattern of the constraints in optimisation problem (3.3) and show that they possess a natural aggregate chordal sparsity structure, with the degree of sparsity depending on the size of the NN. To show this we write the constraints out in full and consider each constraint separately before combining them together.

3.4.1 Input Constraint

The overall input constraint can be written as

$$M_{\text{in}} = \begin{bmatrix} -2\Gamma & 0 & \dots & 0 & \Gamma(\underline{u}+\overline{u}) \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ (\underline{u}+\overline{u})^T\Gamma & 0 & \dots & 0 & (\underline{u}+\overline{u})^T\Gamma(\underline{u}+\overline{u}) \end{bmatrix}.$$

Recall that Γ is a diagonal matrix and $(\underline{u}+\overline{u})$ is a column vector. Hence it can be seen that this matrix has an arrow pattern and is therefore chordal.

3.4.2 Output Constraint

The overall output constraint can be written as

$$M_{\text{out}} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & \dots & 0 & 0 & (W^\ell)^T c \\ 0 & \dots & 0 & c^T W^\ell & 2(c^T b^\ell - d) \end{bmatrix}.$$

Since $(W^\ell)^T c$ is a column vector and $2(c^T b^\ell - d)$ is a scalar, the matrix is entirely zeros apart from the bottom right corner which contains an arrow - this constraint is therefore also chordal.

3.4.3 Middle Constraint

This constraint is more involved, as the matrix has to be expanded out in full to determine the structure. The M_{mid} matrix takes the form

$$M_{\text{mid}} = \begin{bmatrix} \boldsymbol{A}^T Q_{11}\boldsymbol{A} + \boldsymbol{B}^T Q_{12}^T \boldsymbol{A} + \boldsymbol{A}^T Q_{12}\boldsymbol{B} + \boldsymbol{B}^T Q_{22}\boldsymbol{B} & \boldsymbol{A}^T Q_{11}\boldsymbol{b} + \boldsymbol{B}^T Q_{12}^T \boldsymbol{b} + \boldsymbol{A}^T Q_{13} + \boldsymbol{B}^T Q_{23} \\ \boldsymbol{b}^T Q_{11}\boldsymbol{A} + Q_{13}^T \boldsymbol{A} + \boldsymbol{b}^T Q_{12}\boldsymbol{B} + Q_{23}^T \boldsymbol{B} & \boldsymbol{b}^T Q_{11}\boldsymbol{b} + Q_{13}^T \boldsymbol{b} + \boldsymbol{b}^T Q_{13} + Q_{33} \end{bmatrix},$$

which can be split into four sub-matrices

$$M_{\text{mid}} = \begin{bmatrix} M_{\text{mid},11} & M_{\text{mid},12} \\ M_{\text{mid},21} & M_{\text{mid},22} \end{bmatrix}.$$

Note that $M_{\text{mid},22}$ is a scalar and that $M_{\text{mid},12}$ and $M_{\text{mid},21}$ are column and row vectors respectively. Investigating each of the $M_{\text{mid},11}$ matrix terms separately gives

$$A^T Q_{11} A = \begin{bmatrix} (W^0)^T Q_{11}^0 W^0 & 0 & 0 & \dots & 0 \\ 0 & (W^1)^T Q_{11}^1 W^1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & (W^{l-1})^T Q_{11}^{l-1} W^{l-1} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$A^T Q_{12} B + B^T Q_{12} A = \begin{bmatrix} 0 & (W^0)^T Q_{12}^0 & 0 & 0 & \dots & 0 \\ Q_{12}^0 W^0 & 0 & (W^1)^T Q_{12}^1 & 0 & \dots & 0 \\ 0 & Q_{12}^1 W^1 & 0 & (W^2)^T Q_{12}^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & 0 & 0 & (W^{l-1})^T Q_{12}^{l-1} \\ 0 & 0 & \dots & 0 & Q_{12}^{l-1} W^{l-1} & 0 \end{bmatrix},$$

$$B^T Q_{22} B = 0.$$

The Q_{11}^k and Q_{12}^k terms represent the sub-matrices of Q_{11} and Q_{12} respectively, corresponding to the kth layer of the NN. The Q_{11} and Q_{12} matrices are diagonal matrices. As shown in Example 2.20, we can use the preprocessing bounds to determine if a given ReLU function is active ($\underline{\phi} > 0 \Rightarrow \phi = x$), inactive ($\overline{\phi} \leq 0 \Rightarrow \phi = 0$) or unknown ($\underline{\phi} \leq 0$, $\overline{\phi} > 0$). In Q_{11} each term is either 0 or λ_i and in Q_{12} each term is either 0, λ_i or $2\lambda_i$ depending on whether the neuron is active, inactive or unknown respectively. The sum of all of these matrices is therefore

$$M_{\text{mid},11} = \begin{bmatrix} (W^0)^T Q_{11}^0 W^0 & (W^0)^T Q_{12}^0 & 0 & 0 & \dots & 0 \\ Q_{12}^0 W^0 & (W^1)^T Q_{11}^1 W^1 & (W^1)^T Q_{12}^1 & 0 & \dots & 0 \\ 0 & Q_{12}^1 W^1 & (W^2)^T Q_{11}^2 W^2 & (W^2)^T Q_{12}^2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & Q_{12}^{l-2} W^{l-2} & (W^{l-1})^T Q_{11}^{l-1} W^{l-1} & (W^{l-1})^T Q_{12}^{l-1} \\ 0 & 0 & \dots & 0 & Q_{12}^{l-1} W^{l-1} & 0 \end{bmatrix}.$$

The general structure of the M_{mid} matrix is not necessarily chordal, as this structure may be broken if some nodes within the hidden layers are not connected. However to create a chordal extension, the intra-connections in the hidden layers can be assumed to be nonzero. This can be framed by considering the constraint matrix in its densest form. If all of the Q_{11} and Q_{12} terms are full along the diagonal then the matrix forms a block arrow pattern, which is known to be chordal [202]. When all of the constraints are combined to form (3.2) in the SDP, the constraint keeps this block arrow pattern. This is because M_{in} and M_{out} only have terms in the final row and final column of the matrix, which will preserve the sparsity pattern. This result is significant as it shows that

the NN structure is inherited in the algorithm formulation and should be exploited in computation for scalability. This pattern can be shown visually in Figure 3.1.

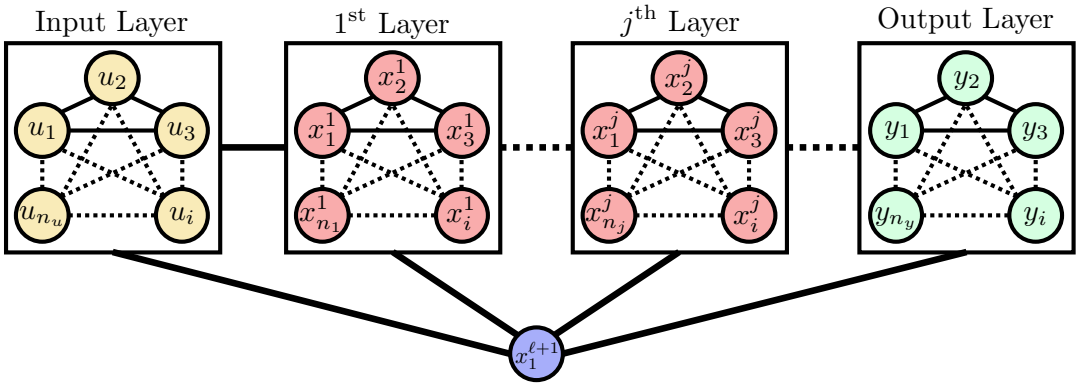


Figure 3.1: Graph showing the chordal structure of the SDP constraint matrix in (3.3). x_i^j represents the term in the constraint matrices corresponding to the jth layer with i^{th} hidden unit. The lines within each layer represent that all of the nodes in that layer are connected to each other. The solid lines between layers represents that all of the nodes in that layer are connected to all of the nodes in the adjacent layer. The dots represent that this pattern continues to the final layer in the neural network.

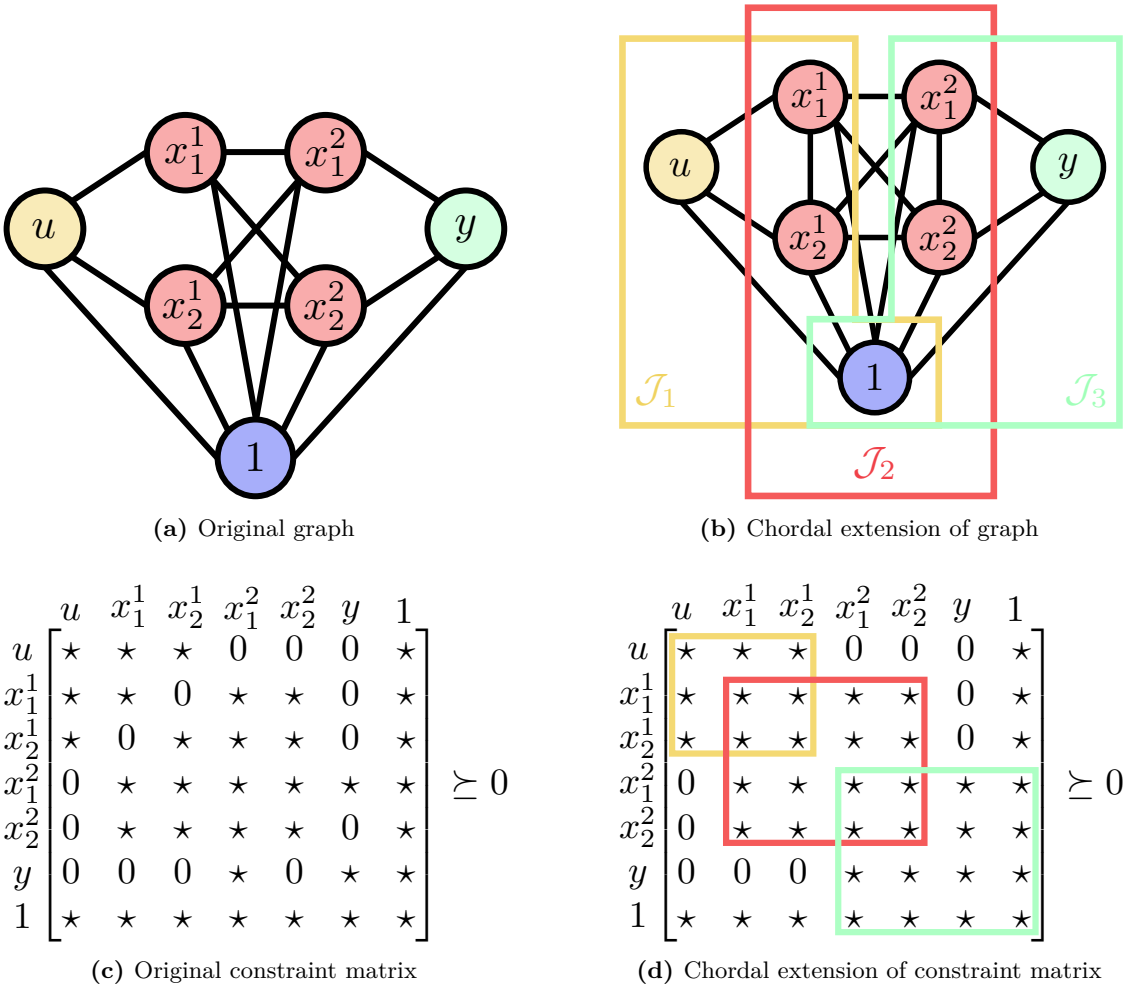
3.4.4 Neural Network Sparsity Examples

Example 3.11. To visualise this block arrow pattern we consider a single input-output NN with two layers and two nodes in each layer. The NN is therefore given by the equations

$$\begin{aligned} x^0 &= u, \\ \begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} &= \begin{bmatrix} W_1^1 \\ W_2^1 \end{bmatrix} x^0 + \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}, \\ \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} &= \begin{bmatrix} W_1^2 & W_2^2 \\ W_3^2 & W_4^2 \end{bmatrix} \begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix}, \\ y &= \begin{bmatrix} W_1^3 & W_2^3 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} + b^3, \end{aligned} \tag{3.4}$$

The chordal structure for this example and how it translates to the constraint matrix is shown in Figure 3.2.

Example 3.12. We now consider a larger NN example with two inputs, two outputs and 100 nodes. Firstly, we consider a ten layer network and ten nodes in each layer. The block arrow pattern for the constraint matrix is shown in Figure 3.3. Next we consider a twenty-layer network with five nodes in each layer, the sparsity pattern for this network is shown in Figure 3.4. We can see that although each network has the same number of



(a) Original graph

(b) Chordal extension of graph

(c) Original constraint matrix

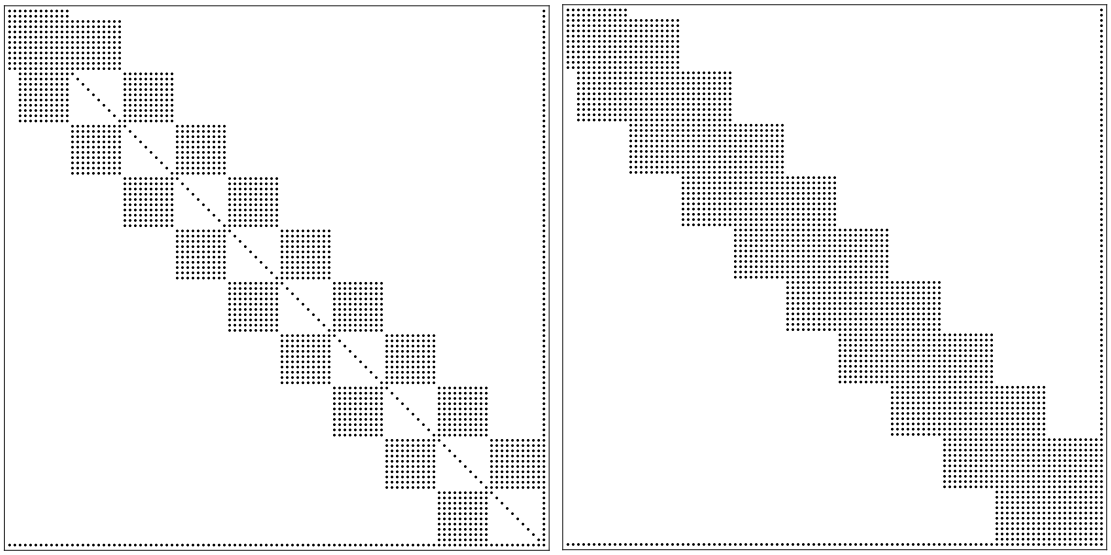
(d) Chordal extension of constraint matrix

Figure 3.2: Diagram showing how the chordal extension of the sparsity graph is created and hence how the cliques in the constraint matrix are formed for a simple example in (3.4).

nodes, there is more sparsity to be exploited in the network with more layers and fewer nodes. This means that using chordal sparsity for SDP-based NN verification will benefit deeper networks more than wider networks. This is useful since depth is more important than width in the expressivity of NNs [203].

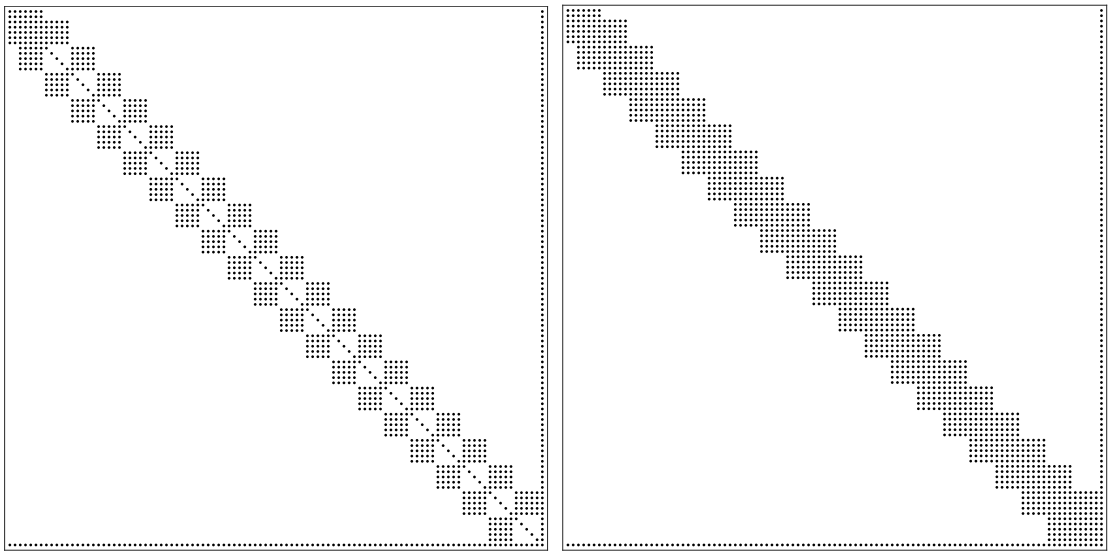
3.5 Numerical Results

In this section we compare the computational aspects of solving optimisation problem (3.3) with and without exploiting chordal sparsity. The solve time and solution accuracy are compared for different SDP solvers using YALMIP [73] as the SDP parser in MATLAB. The solvers we will be comparing are sparseCoLo [94] and SeDuMi [61], which is used as a benchmark SDP solver that uses interior point methods. SparseCoLo will exploit



(a) Original constraint matrices **(b)** Chordal extension of constraint matrices

Figure 3.3: Diagram comparing the sparsity pattern of the constraint matrix in (3.3) for a ten-layer neural network with ten hidden units in each layer.



(a) Original constraint matrices **(b)** Chordal extension of constraint matrices

Figure 3.4: Diagram comparing the sparsity pattern of the constraint matrix in (3.3) for a twenty-layer neural network with five hidden units in each layer.

the sparsity of the problem and then parse the problem to SeDuMi. All experiments were run on a four-core Intel Xeon processor @3.50GHz with 16GB of RAM. We used code provided from the authors of [60] called 'DeepSDP' that creates the matrices that are used in the SDP. The weights and biases of the NN are randomly generated from a Gaussian distribution with the dimensions specified with the number of inputs, number

of hidden layers, size of each hidden layer and number of outputs. For all examples, we set the dimension of the inputs and outputs to two for consistency; it was observed that changing these dimensions modestly had a small effect on the computational time. In all examples we also consider the input set as an ℓ_∞ ball with radius $\epsilon = 0.5$ and a center of $u^\star = (1, 1)$ for comparison purposes to benchmark against previous results. These input constraints can be expressed as a hyper-rectangle such that $0.5 \leq u \leq 1.5$.

3.5.1 Scalability with Varying Hidden Layers

In this example the NN has two nodes in each layer and ℓ layers. Figure 3.5 shows that sparseCoLo becomes faster after 50 layers and scales well. There is no loss in the solution accuracy between the two solvers.

In some instances that we experimented on it was noticed that sparseCoLo was slower than SeDuMi. This happens when there is not much sparsity in the SDP. An example of this would be a two layer network with 100 hidden units in both layers. Analysing the constraint matrix of this case reveals that there is not much sparsity, although the chordal block arrow pattern is still present. If we were to increase the number of layers then the sparsity in the constraint would increase and hence sparseCoLo would become more effective.

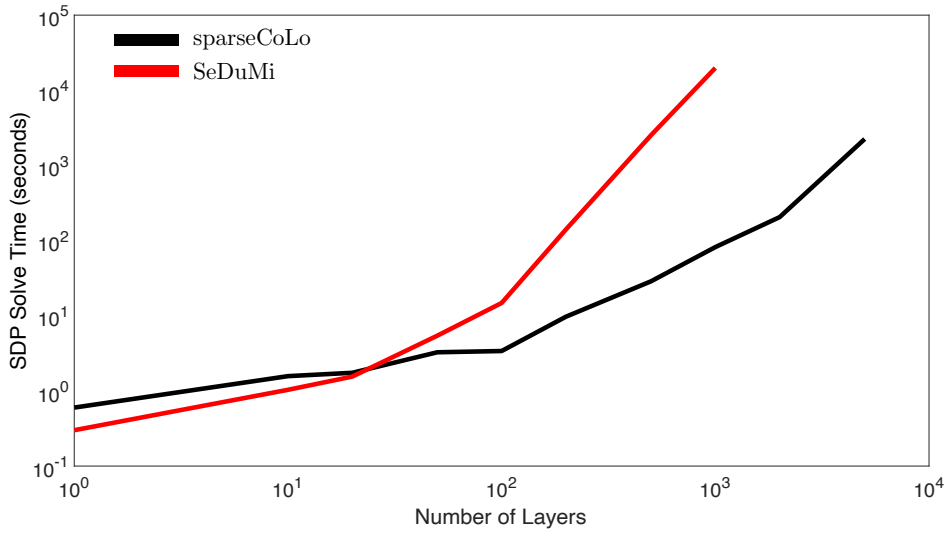


Figure 3.5: Graph comparing the computational time to solve an ℓ-layer network with two hidden units in each layer.

3.5.2 Scalability with Varying Hidden Units

Consider an 100 layer NN with a varying number of neurons in each layer. We can see in Figure 3.6 that by using chordal sparsity, the computational time to solve the SDP decreases when the number of neurons in each layer increases. The bounds generated by each solver were identical, showing no loss of accuracy from the sparse solver.

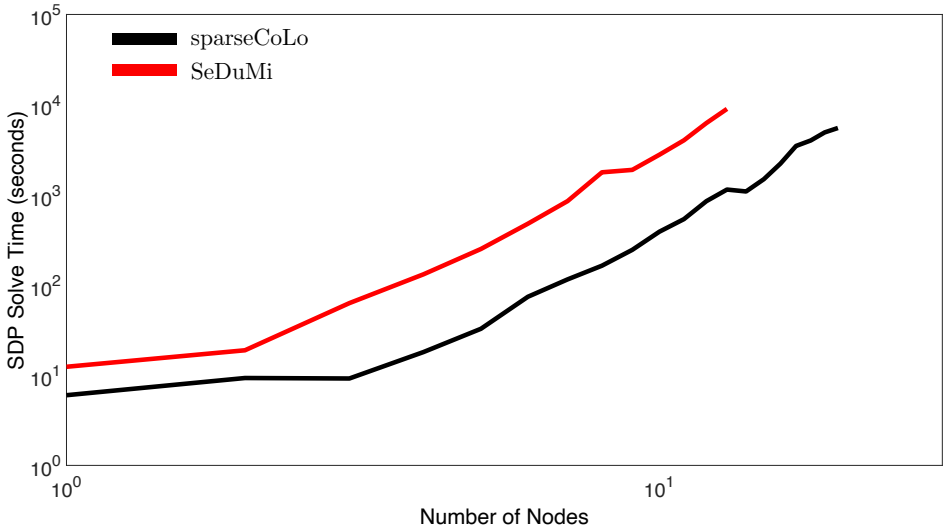


Figure 3.6: Graph comparing the computational time to solve an 100-layer network with a varying number of hidden units.

Now we consider a single layer NN with n_1 hidden neurons in the layer. From Figure 3.7 we can observe a significant speed up in computational time using sparseCoLo over SeDuMi. After 100 neurons we can see the computational time significantly change, with sparseCoLo able to solve the problem with $n_1 = 10,000$ neurons where SeDuMi fails at $n_1 = 5,000$ neurons. This is interesting as since there is only one layer, the chordal sparsity is being exploited from the nodes in the hidden layer, instead of the concatenating structure of the connections of each layer.

3.6 Conclusion

In this chapter we considered the robustness analysis problem for a fully connected feed-forward NN using an SDP framework. Due to the intrinsic cascading structure of the NN there is a naturally arising sparsity pattern in the LMI constraints, which can be taken advantage of in computations. We applied theory from chordal graphs and

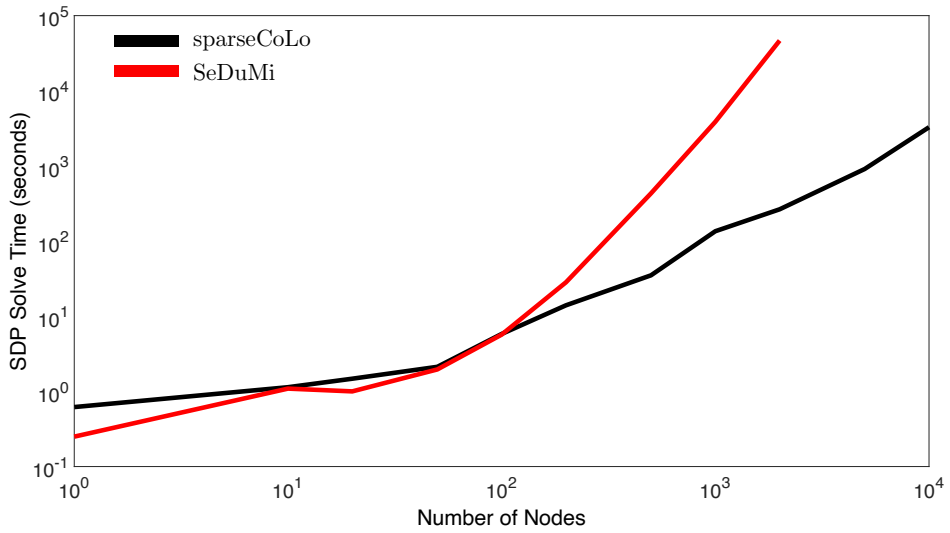


Figure 3.7: Graph comparing the computational time to solve an one-layer network with a varying number of hidden units.

semidefinite matrices to decompose the resulting SDP. Our numerical results show how exploiting the chordal structure can significantly speed up the computational time to solve the optimisation problem and verify the properties of the NN's output. We show that the computational time to solve the problem depends on the NN structure.

4

Neural Network Verification using Polynomial Optimisation

This chapter uses the Positivstellensatz in the neural network verification problem. New sector constraints on the sigmoid and tanh functions are proposed to take advantage of the flexibility of this framework. The results for this method are presented and discussed, demonstrating how tighter bounds on the neural network's output can be computed.

4.1 Introduction

One of the main issues with the approach in Chapter 3 and other similar approaches, is that they only work for ReLU activation functions. Often approaches that are based on LPs or SDPs rely on a particular type of bound for a particular type of activation function. However, when NNs have varying sizes with different activation functions a more general approach may be more useful. This will not only make it easier to express and implement the optimisation problem for an arbitrary NN architecture, but it also allows us to more freely trade off accuracy with scalability.

In this chapter, we approach the problem from a different perspective by considering a general class of constraints representing different activation functions. To proceed with this approach we use the Psatz (outlined in Section 2.1.3), which uses an algebraic condition to test the emptiness of a semi-algebraic set. This method is different, as instead of considering only one type of constraint e.g. linear, quadratic etc., we can use a mixture of the constraints and easily trade-off solution accuracy with computational effort. Testing the algebraic condition on the Psatz can be done using SOS and solved by an equivalent

SDP. This SOS framework can be parsed in MATLAB using SOSTOOLS [64] and the resulting SDP can be solved using a relevant solver e.g. SeDuMi [61], SDPT3 [65] etc.

Using this polynomial optimisation approach, we show that one can drastically tighten the bounds when using the ReLU activation functions against existing methods. Furthermore, we provide significantly tighter bounds on the sigmoid and tanh activation functions by using a combination of two sector constraints. We then incorporate these constraints into the optimisation framework and provide tighter bounds on the robustness problem. Using both of these contributions allows more flexibility in the optimisation problem. This is because any set of constraints can be defined and then the accuracy may be improved if needed by increasing the order of the Psatz test.

4.2 Neural Network Verification Emptiness Condition

To incorporate the Psatz into the NN verification problem, we describe the input-output properties of the NN as a semi-algebraic set. In Section 2.4.3 we show examples of how this can be achieved. We define the semi-algebraic set

$$S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0,\ \forall\, i = 1, \ldots, p,\ j = 1, \ldots, q\}, \tag{4.1}$$

here g_i and h_j are polynomial functions that express the input-output properties of the NN.

As in Chapter 2 we will assume that the safe set is given by the polytope

$$\mathcal{S}_y = \bigcap_{m=1}^{M} \{y \in \mathbb{R}^{n_y} \mid c_m^T y - d_m \leq 0\},$$

where $c_m \in \mathbb{R}^{n_y}$ and $d_m \in \mathbb{R}$ are given. We consider each of the faces of the polytope in turn, and attempt to obtain a bound γ_m on each d_m to approximate the true safe set. As such, the search for the bounds (γ_m) on the safe output set can be split into M SOS programs. Therefore, the output constraints contain a decision variable that can be optimised in the SOS program such that

$$g_{\text{out}}^m = \gamma_m - c_m^T y \geq 0,$$

where γ_m is the decision variable to be optimised and m denotes the mth SOS program.

We adjust the Psatz condition slightly to use it more easily in the SOS optimisation framework. Instead of showing that $g_{\text{out}}^m \geq 0$ is feasible, we can show that $g_{\text{out}}^m < 0$ is

infeasible using the Psatz. By setting γ_m as the decision variable in the SOS program, it can be optimised to find the limiting value to when this emptiness condition is violated. In our case we wish to show that $g^m_{\text{out}} \geq 0$ on S, and therefore by using (2.4) we present the following optimisation problem

$$\begin{aligned} \min \quad & \gamma_m \\ \text{s.t.} \quad & -c_m^T y + \gamma_m - \sum_j^q t_j h_j - \sum_i^p s_i g_i - \sum_{i \neq j}^p r_{ij} g_i g_j - \ldots \in \Sigma[x], \\ & t_j \in \mathbb{R}[x],\ \forall\, j = 1, \ldots, q, \\ & s_i \ \in \Sigma[x],\ \forall\, i = 1, \ldots, p, \\ & r_{ij} \ \in \Sigma[x],\ \forall\, i, j = 1, \ldots, p, \end{aligned} \tag{4.2}$$

where $h_j(x)$ and $g_i(x)$ are the equality and inequality constraints describing S in (4.1).

Using this form of the Psatz is too computationally expensive due to the number of optimisation variables that are introduced from the constraints that are multiplied together. Hence we solve the following optimisation problem

$$\begin{aligned} \min \quad & \gamma_m \\ \text{s.t.} \quad & -c_m^T y + \gamma_m - \sum_j^q t_j h_j - \sum_i^p s_i g_i \in \Sigma[x], \\ & t_j \in \mathbb{R}[x],\ \forall\, j = 1, \ldots, q, \\ & s_i \ \in \Sigma[x],\ \forall\, i = 1, \ldots, p, \end{aligned} \tag{4.3}$$

where $h_j(x)$ and $g_i(x)$ are the equality and inequality constraints describing S in (4.1).

4.3 Improved Sector Bounds

The benefit of the Psatz framework is being able to easily incorporate any type of constraint into the optimisation problem. We note that previous sector constraints on the sigmoid and tanh functions are conservative. Therefore, we propose tighter sector constraints, which would not be possible to use in previous formulations.

4.3.1 Sigmoid Activation Function

Recall that the sigmoid function is given by

$$\text{sig}(x) = \phi(x) = \frac{1}{1 + e^{-x}}.$$

The simplest bound on this activation function is that the output must be in the region $\underline{\phi} \leq \phi \leq \overline{\phi}$. These constraints are of course very conservative, hence we employ sector constraints to provide better results. In this work, we bound the sigmoid and tanh activation functions through two overlapping sectors and optimise the position and slope. Although sector conditions have been used in previous works [60, 177], they have not used multiple overlapping sectors in this way. Previous formulations have only considered a single sector constraint, which is outlined in Section 2.4.3.

Having two sectors will provide much tighter bounds on the activation function as it captures the point of inflection on the sigmoid and tanh functions, which is impossible to achieve with a single sector constraint. The disadvantage of our approach is that it will have twice the number of sector constraints, increasing the number of optimisation variables and therefore increasing the SDP solve time.

We can position the two sectors by using the IBP values to minimise the uncertainty in the constraints. The arrangement for positioning these sectors is shown visually in Figure 4.1c. We start by positioning one sector at an arbitrary point on the sigmoid curve $(x_m, \phi(x_m))$. The lower line of the sector is the line passing through the points $\phi(x_m)$ and $\overline{\phi}$. The upper line of the sector passes through the point $\phi(x_m)$ and a point that is tangent to the sigmoid curve to the left of the x_m point (a_1). The equation of the lower line is

$$\phi(x) = \frac{\phi(x_m) - \overline{\phi}}{x_m - \overline{x}}(x - \overline{x}) + \overline{\phi}. \tag{4.4}$$

The gradient of the sigmoid function can be expressed as

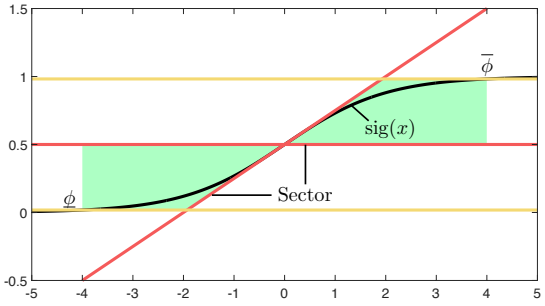
$$\frac{\partial \phi}{\partial x} = \phi(x)(1 - \phi(x)).$$

Therefore, the gradient of the upper line can be found with the formula

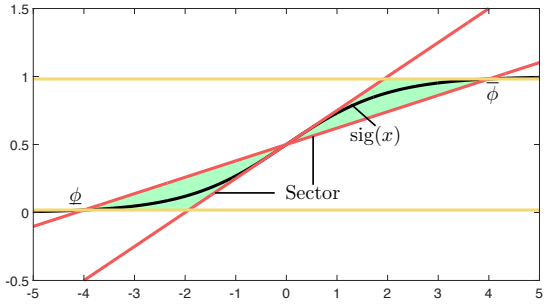
$$\frac{\partial \phi}{\partial x}\Big|_{x=a_1} = \phi(a_1)(1 - \phi(a_1)) = \frac{\phi(x_m) - \phi(a_1)}{x_m - a_1}.$$

Given a value of x_m, this non-linear equation can be solved numerically. The same can be repeated for the sector constraint to the left of the ϕ-axis, with the values of x_m and $\overline{\phi}$ replaced with $-x_m$ and $\underline{\phi}$ respectively. The value of x_m can be treated as a hyperparameter to be chosen; we found that a value of $x_m = 1.5$ provided the best overall results in our simulations. To interpret this we can think of the uncertainty as the area

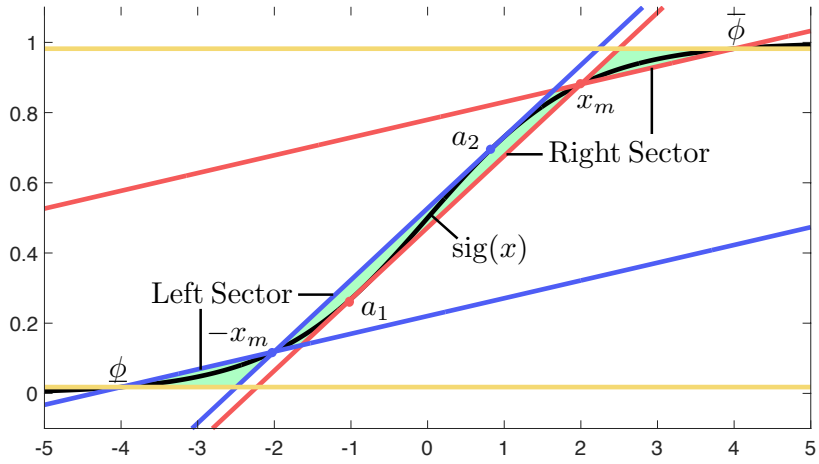
between the sectors; it can be shown numerically that a value of approximately 1.5 will minimise this area on average. If the sectors overlap in the wrong region, then the slope of each sector can be adjusted accordingly. There are certain cases where these sector constraints will need to be adjusted due to the positioning of the preprocessing bounds.



(a) Single sector constraint (red)



(b) Tightened single sector constraint (red)



(c) Two intersecting sector constraints, the right and left sectors are shown in red and blue respectively

Figure 4.1: Plot comparing sector constraints that bound the sigmoid function (black). The yellow lines represent the preprocessing bounds and the area in green represents the region bound by the constraints.

4.3.2 Tanh Activation Function

Recall that the tanh function is given by

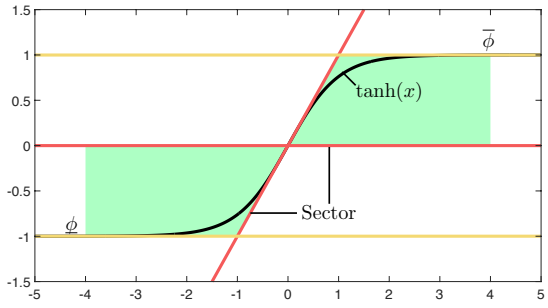
$$\tanh(x) = \phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Similar to the sigmoid case, the function remains within the bounds from the IBP step. The process of computing the sectors is also the same, however they will be positioned differently. The gradient of the lower line can be calculated using the same formula as in (4.4). However,

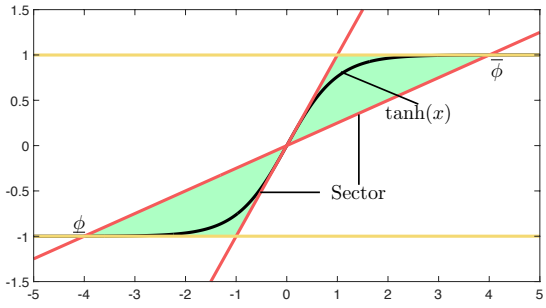
the gradient of the upper line is replaced with a different non-linear equation such that

$$\frac{\partial \phi}{\partial x}\Big|_{x=a_1} = 1 - \phi^2(a_1) = \frac{\phi(x_m) - \phi(a_1)}{x_m - a_1}.$$

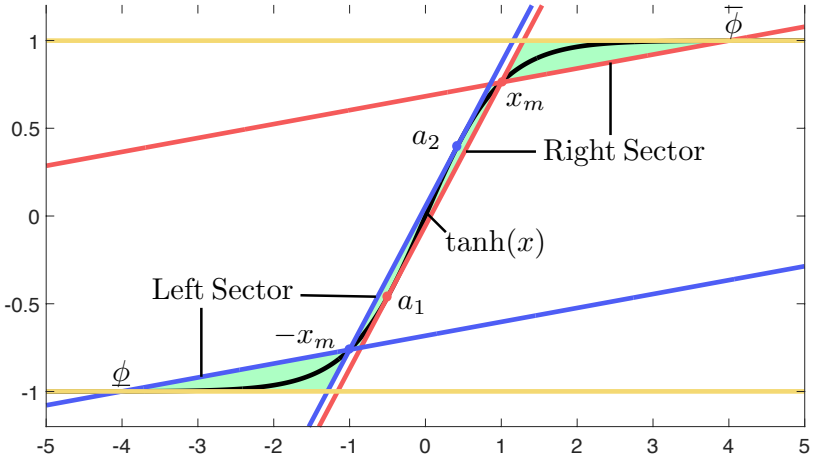
We found that a value of $x_m = 1$ gave the best results. The single sector constraints are shown in Figure 4.2a and Figure 4.2b and the two sector constraint is shown in Figure 4.2c.



(a) Single sector constraint (red)



(b) Tightened single sector constraint (red)



(c) Two intersecting sector constraints, the right and left sectors are shown in red and blue respectively

Figure 4.2: Plot comparing sector constraints that bound the tanh function (black). The yellow lines represent the preprocessing bounds and the area in green represents the region bound by the constraints.

4.4 Numerical Results

We now compare our method and existing methods for various NN sizes, with different activation functions. All experiments were run on a four-core Intel Xeon processor @3.50GHz with 16GB of RAM. For our 'NNPsatz' method, we use SOSTOOLS [64] and SDPT3 [65] to parse and solve the SDP in (4.3). SDPT3 was chosen as we found it to be the solver that produced the least numerical errors with this SOS program. The

methods that we compare our method to are DeepSDP (without slope constraints) [60], DeepSDP$\star$ (with slope constraints), Interval Bound Propagation (IBP) [104] and the true values which are computed by exhaustive search.

There are a large number of parameters to consider in this problem, due to the NN structure. For example, we can vary the input, hidden layer and output dimensions, as well as the size of the input space. Additionally, there are many ways that the multipliers in the Psatz can be chosen, as the order of the multipliers can change as well as the number of variables that they contain. Furthermore, the order of the polynomial multipliers can be controlled and the number of variables contained in the multipliers can be altered to make the SOS formulation more or less conservative. This is because adding more and higher order constraints into the SOS problem increases the number of variables in the SDP.

4.4.1 ReLU Activation Function Example

To show the effectiveness of our method, we start with a small two layer NN with two nodes in each layer, containing ReLU activation functions, with a single input and output. We consider a large input space $[\underline{u}, \overline{u}] = [-50, 50]$, to rigorously test each method. The results for this example are shown in Table 4.1. NNPsatz uses the ReLU representation in (2.10) and results in a fourth order polynomial optimisation problem. These results show a significant increase in accuracy over the other methods. The maximum and minimum bounds $(y_{\max}, y_{\min})$ obtained by NNPsatz are identical to the true values (to a precision of four decimal places). This is of course at the expense of computational time, however this can be improved with the trade-off of reducing the solution accuracy. DeepSDP was unable to provide tight bounds on the outputs, showing the improved accuracy that results from the higher order polynomial optimisation problem. The added slope constraints from DeepSDP$\star$ can significantly tighten the bounds, however there is a noticeable error in the lower bound. NNPsatz is limited to smaller NNs with sizes of around 25 nodes, as the computational time becomes intractable. The solve time can also be improved with more efficient optimisation methods, that can exploit the sparsity of the problem. This is shown in Chapter 5.

Table 4.1: Comparison of verification methods for the ReLU activation function

	$y_{\min}$	$y_{\max}$	Solve Time (s)	SDP Size
NNPsatz	4.3000	18.0384	1.1147	340×100
DeepSDP	-10.6868	69.6374	0.0767	49×18
DeepSDP$\star$	0.5371	18.0384	0.0975	55×24
IBP	-14.7009	105.3742	0.0011	-
True values	4.3000	18.0384	-	-

4.4.2 Sigmoid and Tanh Activation Function Examples

In this example we show the effectiveness of this method in conjunction with the sigmoid and tanh activation functions. Unlike the ReLU function, the shape of sigmoid and tanh functions reduces the magnitude of the outputs, meaning that a larger NN is required to compare the bounds on the outputs. We consider a single input, single output NN with four hidden layers and four nodes in each layer. We consider the input space $[\underline{u}, \overline{u}] = [0.25, 1.75]$. Since at the time of writing DeepSDP does not have an implementation for sigmoid and tanh functions, we compare the accurate NNPsatz method against similar formulations, to show how solution accuracy and computation can be traded-off. The results for the sigmoid and tanh examples are shown in Table 4.2 and Table 4.3 respectively. NNPsatz (1) uses the two sector constraints as in Figure 4.1c and 4.2c and the IBP constraints (2.9). NNPsatz (2) is similar, however the two sector constraints are replaced by single sectors as in Figure 4.1a and Figure 4.2a. NNPsatz (3) is the same as NNPsatz (2) with added slope restricted constraints. NNPsatz (2) and NNPsatz (3) reflect what is possible with traditional SDP formulations.

For both the sigmoid and tanh activation functions, NNPsatz (1) generates the tightest bounds for the maximum and minimum values on the NN's output. We can see a small increase in accuracy in NNPsatz (3) over NNPsatz (2) when including slope constraints. These numerical examples show that slope constraints do not increase the accuracy of the bounds as much as looking for higher order problem representations using the Psatz. The computational time to solve all NNPsatz methods is similar, as the SDP for each method is of a similar size.

Table 4.2: Comparison of verification methods for the sigmoid activation function

	$y_{\min}$	$y_{\max}$	Solve Time (s)	SDP Size
NNPsatz (1)	0.7417	0.7617	0.7650	360×171
NNPsatz (2)	0.6443	0.8653	0.4397	358×171
NNPsatz (3)	0.6742	0.8565	0.7658	478×171
IBP	0.6440	0.8690	0.004	-
True values	0.7465	0.7550	-	-

Table 4.3: Comparison of verification methods for the tanh activation function

	$y_{\min}$	$y_{\max}$	Solve Time (s)	SDP Size
NNPsatz (1)	-3.8258	0.6841	0.4739	369×171
NNPsatz (2)	-4.2631	4.0443	0.4096	358×171
NNPsatz (3)	-4.2601	2.8704	0.7470	478×171
IBP	-4.2645	4.2780	0.004	-
True values	-2.8080	-2.3531	-	-

4.4.3 Two Dimensional Input and Output Example

The examples in the previous sections only consider a single input, single output NN. We now consider the effectiveness for an NN with two inputs and two outputs. The hidden layer dimensions are $[3, 4, 3]$, containing ReLU activation functions. We compare NNPatz, DeepSDP and IBP for an input space $[\underline{u}, \overline{u}] = [-50, 50]$. In Figure 4.3, we see that NNPsatz gives significantly tighter bounds over the other methods. NNPsatz took 141s to solve the eight optimisation problems, whereas DeepSDP took 3.87s, which shows the additional computation effort that is required to get a more accurate solution.

4.5 Conclusion

In this chapter we investigated the problem of providing bounds on the outputs of NNs, to ensure that they operate safely and reliably. We have approached the problem from a general framework that combines real algebraic geometry with polynomial optimisation, by setting up the problem as the emptiness of an appropriately constructed semi-algebraic set. As the constraints within this framework can be defined generally and hence can be made more or less conservative, it is possible to easily trade off solution accuracy with the amount of computational time required.

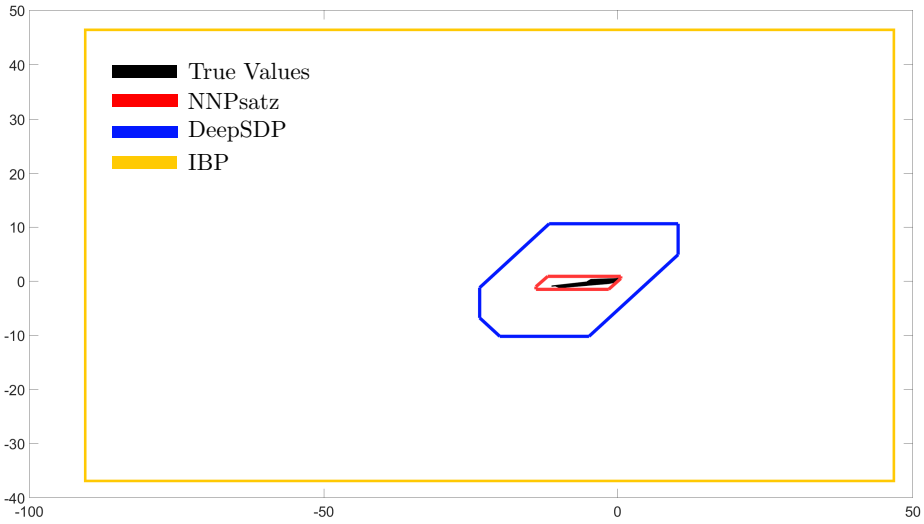


Figure 4.3: Plot of the true output set (black) and the bounds that are created using NNPsatz (red), DeepSDP (blue) and IBP (yellow).

We use existing constraints on the ReLU activation function and propose a bound consisting of two sector constraints for the sigmoid and tanh activation functions. These constraints are then parsed into SOSTOOLS, which converts the problem into an SDP, which is solved using the solver 'SDPT3'. We show with many examples that this formulation can provide the tightest bounds on the NN compared to other SDP formulations. This approach is also very versatile as the constraints can be modified and a different accuracy can be set depending on the size of the network.

5

Neural Network Verification using Sparse Polynomial Optimisation

This chapter provides a more detailed view of Sum of Squares programming and the theory behind sparse polynomial optimisation. It is then shown how this links to the Positivstellensatz. The theory is then applied to the specific neural network verification problem showing how sparsity is persevered in the optimisation problem. The results from various experiments against state-of-the-art methods are presented and analysed, with both scalability and accuracy being compared.

5.1 Introduction

SOS conditions are useful since they can be cast into LMI constraints and then solved using an SDP. Instead of checking the nonnegativity of a polynomial which is known to be an NP-hard problem [204], we can check if a polynomial is SOS, which can be done by solving the SDP in polynomial time. However, the biggest issue with using an SOS framework is the scalability of the optimisation problem. The NN verification problem becomes intractable as the number of nodes in the network reaches about twenty to thirty. Therefore, sparsity exploitation must be used to verify NNs of any larger size.

In Chapter 3 we showed how the sparse structure of an NN can be used to speed up the computational time to solve the NN verification problem. Then in Chapter 4 we showed how the accuracy of the problem can be improved using the Psatz. In this chapter, we combine both of these approaches to simultaneously exploit this sparsity and improve the accuracy of the solution to the problem.

5.2 Sparse Polynomial Optimisation

Looking at (4.3), one of the issues with using the SOS/SDP framework is that the computational time becomes too large as the size of the NN gets bigger: this is what we refer to as the scalability issue. Since NNs in practice have shown great success when the number of nodes in each layer is large and when there are numerous layers, it is important to devise solutions to allow for tractable solve times. However, as shown in Chapter 3 it is possible to use the natural cascading structure of the NN to improve the solve times drastically. In fact, common constraints that are used to bound the activation functions only contain variables in the current layer and the previous layer. This means that the constraint matrices have a very well-defined structure and therefore this cascading is inherited in the algorithm formulation.

5.2.1 Sparse Sum of Squares Representations

We will now show how the ideas of sparse matrix decomposition can be extended to sparse SOS decompositions in a general polynomial optimisation framework. For a detailed account on exploiting sparsity for polynomial optimisation the reader is referred to [205] for a comprehensive overview.

Recall the Gram matrix representation in Theorem 2.4 from Section 2.1.2 and that instead of using the full set of monomials, we use $x^{\mathbb{B}}$ where $\mathbb{B} \subseteq \mathbb{N}^n_d$ is the set of exponents, i.e $f(x) = (x^{\mathbb{B}})^T Q x^{\mathbb{B}}$. If Q is chosen to not be dense and instead possess a chordal sparsity pattern, then the resulting SDP can be decomposed into smaller SDP constraints. If $f \in \mathbb{R}[x]$ has a dense support such that $\text{supp}(f) = \mathbb{N}^n_{2d}$, then the monomial exponents must be $\mathbb{B} = \mathbb{N}^n_d$ and the matrix Q is a fully dense matrix of size $\binom{n+d}{d} \times \binom{n+d}{d}$. However, if the polynomial f only contains a subset of all the monomial terms then we can expect that the matrix Q is not full and that a certain sparsity pattern will arise. There are different types of sparsity that can be analysed, which we will summarise in the following sections.

SOS constraints can also use sparse matrix decomposition due to their relationship with an underlying SDP. To express the relationship between monomials consider a graph $\mathcal{G}(\mathbb{B}, \mathcal{E})$ that has maximal cliques $\mathcal{C}_1, \ldots, \mathcal{C}_t$ and edge set $\mathcal{E} \subseteq \mathbb{B} \times \mathbb{B}$, which conveys how

the monomials are connected together. The exponent set can then be written as

$$\mathbb{A} \subseteq \{\beta + \gamma \ : \ (\beta, \gamma) \in \mathcal{E}\}.$$

Given this exponent set, the subcone of SOS polynomials that are supported on $\mathbb{A}$ is defined as

$$\Sigma[\mathbb{A}] := \{f \in \Sigma \ : \ \text{supp}(f) \subseteq \mathbb{A}\}.$$

We can then use the clique-based positive semidefinite decomposition to express the Gram matrix as

$$Q = \sum_{k=1}^{t} E_{\mathcal{C}_k}^T Z_k E_{\mathcal{C}_k}, \text{ where } Z_k \in \mathbb{S}_+^{|\mathcal{C}_k|}.$$

This $Q \in \mathbb{S}_+(\mathcal{E}, 0)$ belongs to the cone of sparse SOS polynomials expressed as

$$\Sigma[\mathbb{A}; \mathcal{E}] := \{f \in \Sigma[\mathbb{A}] \ : \ f = (x^{\mathbb{B}})^T Q x^{\mathbb{B}}\},$$

and it can be shown that $\Sigma[\mathbb{A}; \mathcal{E}] \subseteq \Sigma[\mathbb{A}]$. The cone can be converted into an SDP condition such that $f \in \Sigma[\mathbb{A}; \mathcal{E}]$ if and only if

$$\exists Z_1 \in \mathbb{S}_+^{|\mathcal{C}_1|}, \ldots, Z_t \in \mathbb{S}_+^{|\mathcal{C}_t|}, \text{ such that } \sum_{k=1}^{t} \langle Z_k, E_{\mathcal{C}_k} A_\alpha E_{\mathcal{C}_k}^T \rangle = f_\alpha \quad \forall \alpha \in \mathbb{B} + \mathbb{B}. \tag{5.1}$$

Consider a given sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E})$, the cone $\Sigma[\mathbb{A}; \mathcal{E}] \subseteq \Sigma[\mathbb{A}]$ contains SOS polynomials that can admit a sparse SOS decomposition. This yields the following representation

$$\begin{aligned} f(x) &= \sum_{k=1}^{t} (x^{\mathbb{B}})^T E_{C_k}^T Z_k E_{C_k} x^{\mathbb{B}}, \\ &= \sum_{k=1}^{t} \sigma_k(x), \end{aligned} \tag{5.2}$$

where $\sigma_k(x) := (E_{C_k} x^{\mathbb{B}})^T Z_k (E_{C_k} x^{\mathbb{B}}) \in \Sigma[x]$ is sparse since it considers terms that are a subset of the monomial vector $x^{\mathbb{B}}$.

This chordal sparsity decomposition can be used to reduce the size of the SDP and hence improve the scalability of the problem. The difficulty now is to select the cliques in the correct way for this decomposition to be valid. For non-chordal graphs this is an NP-hard problem, although chordal extensions can be created to overcome this. However in the NN verification problem we know the structure of the problem and therefore we can directly analyse and select the cliques, so that we know they are chordal.

Note that sparsity in the support set $\mathbb{A}$ does not always imply the existence of a sparse Gram matrix Q [205]. Therefore by using this sparse matrix decomposition the benefit of reduced computational complexity of the problem may be traded-off with the expense of conservatism.

5.2.2 Term Sparsity

For a more comprehensive review of term sparsity, the reader is referred to [205–208]. Term sparsity involves reducing the size of the support set of f. The simplest way to exploit term sparsity is to take the Newton polytope reduction such that

$$\mathbb{B} = \frac{1}{2}\text{New}(f) \cap \mathbb{N}_d^n,$$

where $\text{New}(f)$ of f is the convex hull of $\text{supp}(f)$. This can be simplified further using general facial reduction techniques such as [209] and [210]. Such techniques will remove redundant elements of $\mathbb{B}$ to construct a smaller exponent set by analysing the support of f. However, facial reduction can sometimes only reduce a very small number of terms which might make some problems intractable. This is demonstrated in the following example.

Example 5.1. ([205]) Let $n = 50$ and $d = 2$ and consider the polynomial

$$f(x) = \sum_{i=2}^{49}(x_{i-1} + x_i + x_{i+1})^4. \tag{5.3}$$

There are $\binom{50+4}{4} = 316251$ possible monomials, but the support of $f(x)$ only contains 485 terms. However, taking the Newton polytope reduction only removes 51 of the 1326 exponents of degree 2. This results in an SDP with a Gram matrix of size 1275×1275.

More advanced techniques exist to reduce this term sparsity further, and [205] describes a general approach to exploiting term sparsity. An important point is that although these sparse representations will reduce the computational complexity, there is usually a trade-off with conservatism as term sparsity in the reduced support set does not imply the existence of an SOS polynomial.

5.2.3 Correlative Sparsity

Correlative sparsity takes a different approach to term sparsity. Instead of trying to reduce the size of $|\text{supp}(f)|$, it considers couplings between variables. Two variables x_i and x_j are considered to be coupled if a monomial term in $x^{\mathbb{A}}$ depends on both simultaneously. To describe these couplings we create a correlative sparsity graph of the support set $\mathbb{A}$ with vertices $\{1, \ldots, n\}$. The correlative sparsity graph of the support set $\mathbb{A}$ is defined by

$$\mathcal{S}_{\text{csp}}(\mathbb{A}) := \{(i,j) \,:\, \exists \alpha \in \mathbb{A} \text{ with } \alpha_i \alpha_j > 0\},$$

i.e. the two variables x_i and x_j corresponding to α_i and α_j respectively are considered to be coupled. When applied to the sparse SOS decomposition, the entries in the Q matrix that correspond to couplings that do not belong to the correlative sparsity graph of f are set to zero. The sparsity graph of Q then becomes $\mathcal{G}_{\text{csp}}(\mathbb{B}, \mathcal{E}_{\text{csp}})$ where the edge set is defined by

$$\mathcal{E}_{\text{csp}} := \{(\beta,\gamma) \in \mathbb{B} \times \mathbb{B} \,:\, (\beta_i + \gamma_i)(\beta_j + \gamma_j) > 0 \;\Rightarrow (i,j) \in \mathcal{S}_{\text{csp}}(\mathbb{A})\}. \tag{5.4}$$

From the correlative sparsity graph (with n nodes) we can build $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$ (with $|\mathbb{B}|$ nodes). We must ensure that polynomials $(x^{\mathbb{B}})^T Q x^{\mathbb{B}}$ with $Q \in \mathbb{S}^{|\mathbb{B}(\mathcal{E}_{\text{csp}},0)|}$ inherit the correlative sparsity of the original support set $\mathbb{A}$. It can be shown that the properties of $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$ can be inferred from the correlative sparsity graph [208].

Theorem 5.2. ([205]) Consider a correlative sparsity graph that has a support set $\mathbb{A}$ and maximal cliques $\mathcal{J}_1, \ldots, \mathcal{J}_t$. Then $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$ has maximal cliques $\mathcal{C}_k = \{\beta \in \mathbb{B} \,:\, \text{nnz}(\beta) \subseteq \mathcal{J}_k\}$ for $k = 1, \ldots, t$, where $\text{nnz}(\beta) := \{\beta \in \mathbb{B} : \beta_i \neq 0,\ \forall i = 1, \ldots, |\mathbb{B}|\}$. Moreover, if the correlative sparsity graph of $\mathbb{A}$ is chordal then so is $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$.

This is useful as typically the correlative sparsity graph is much smaller than $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$, which makes finding the maximal cliques easier. It can be shown that $E_{\mathcal{C}_k} x^{\mathbb{B}}$ extracts terms that only depend on the variables indexed by $\mathcal{J}_k$. Using the representation in (5.2), exploiting correlative sparsity is achieved by searching for a sparse SOS decomposition in the form

$$f(x) = \sum_{k=1}^{t} \sigma_k(x_{\mathcal{J}_k}),$$

where $\sigma_k \in \Sigma[x]$ and $x_{\mathcal{J}_k}$ is the subset of variables in x indexed by $\mathcal{J}_k$. The following example shows how this is achieved.

Example 5.3. ([205]) Consider the function (5.3) in Example 5.1. This function possesses correlative sparsity with the correlative sparsity graph consisting of maximal cliques $\mathcal{J}_i = \{i, i+1, i+2\}$ for $i = 1, \ldots, n-2$. The elements in $x^{\mathbb{B}}$ that are used in each clique $\mathcal{J}_i$ are

$$\begin{bmatrix} x_i^2 & x_{i+1}^2 & x_{i+2}^2 & x_i x_{i+1} & x_i x_{i+2} & x_{i+1} x_{i+2} \end{bmatrix},$$

this will not introduce any other couplings in different cliques. Therefore the resulting SDP will have the semidefinite matrix variables $Z_1, \ldots, Z_{n-2} \in \mathbb{S}_+^6$, which is significantly more tractable than the dense formulation in Example 5.1.

Example 5.4. ([205]) Consider the quartic polynomial

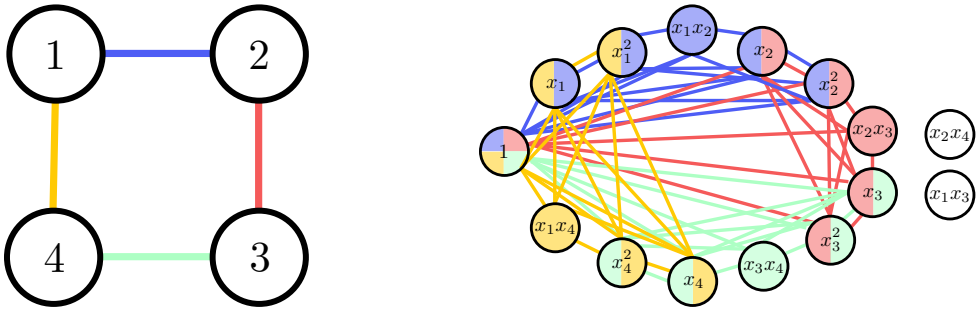
$$\begin{aligned} f(x) = 2 + x_1^4 - x_1^2 + x_2^4 - x_2^2 + x_3^4 - x_3^2 + x_4^4 - x_4^2 + \\ x_1^2 x_2^2 (x_1^2 x_2^2 - 1) + x_2^2 x_3^2 (x_2^2 x_3^2 - 1) + x_3^2 x_4^2 (x_3^2 x_4^2 - 1) + x_1^2 x_4^2 (x_1^2 x_4^2 - 1). \end{aligned} \quad (5.5)$$

One can see a well-defined structure with this polynomial, suggesting that sparsity can be exploited. The correlative sparsity graph can be created, it has maximal cliques $\mathcal{J}_1 = \{1, 2\}$, $\mathcal{J}_2 = \{2, 3\}$, $\mathcal{J}_3 = \{3, 4\}$, $\mathcal{J}_4 = \{4, 1\}$ and is shown in Figure 5.1a. The sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$ is shown in Figure 5.1b along with the cliques. The elements in each clique are given by

$$\begin{aligned} E_{\mathcal{C}_1} x^{\mathbb{B}} &= \begin{bmatrix} x_1^2 & x_2^2 & x_1 x_2 & x_1 & x_2 & 1 \end{bmatrix}, \\ E_{\mathcal{C}_2} x^{\mathbb{B}} &= \begin{bmatrix} x_2^2 & x_3^2 & x_2 x_3 & x_2 & x_3 & 1 \end{bmatrix}, \\ E_{\mathcal{C}_3} x^{\mathbb{B}} &= \begin{bmatrix} x_3^2 & x_4^2 & x_3 x_4 & x_3 & x_4 & 1 \end{bmatrix}, \\ E_{\mathcal{C}_4} x^{\mathbb{B}} &= \begin{bmatrix} x_1^2 & x_4^2 & x_1 x_4 & x_1 & x_4 & 1 \end{bmatrix}. \end{aligned}$$

This decomposition reduces the original 15×15 Gram matrix with four 6×6 smaller semidefinite constraints. Hence searching for these matrices is equivalent to trying to find the sparse SOS decomposition

$$f(x) = \sigma_1(x_1, x_2) + \sigma_2(x_2, x_3) + \sigma_3(x_3, x_4) + \sigma_4(x_4, x_1).$$



(a) Correlative sparsity graph **(b)** Corresponding sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$

Figure 5.1: Diagrams showing the correlative sparsity graph and the sparsity graph ($\mathcal{G}(\mathbb{B}, \mathcal{E}_{\text{csp}})$) of the polynomial in (5.5). The blue, red, green and yellow colours represent the cliques $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$ respectively. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$.

It is not guaranteed that this SOS decomposition will exist, however it does for this example where

$$\begin{aligned}
\sigma_1(x_1, x_2) &= \frac{1}{2}\left(x_1^2 - \frac{1}{2}\right)^2 + \left(x_1x_2 - \frac{1}{2}\right)^2 + \frac{1}{2}\left(x_2^2 - \frac{1}{2}\right)^2, \\
\sigma_2(x_2, x_3) &= \frac{1}{2}\left(x_2^2 - \frac{1}{2}\right)^2 + \left(x_2x_3 - \frac{1}{2}\right)^2 + \frac{1}{2}\left(x_3^2 - \frac{1}{2}\right)^2, \\
\sigma_3(x_3, x_4) &= \frac{1}{2}\left(x_3^2 - \frac{1}{2}\right)^2 + \left(x_3x_4 - \frac{1}{2}\right)^2 + \frac{1}{2}\left(x_4^2 - \frac{1}{2}\right)^2, \\
\sigma_4(x_4, x_1) &= \frac{1}{2}\left(x_4^2 - \frac{1}{2}\right)^2 + \left(x_4x_1 - \frac{1}{2}\right)^2 + \frac{1}{2}\left(x_1^2 - \frac{1}{2}\right)^2.
\end{aligned}$$

5.2.4 Similar Hierarchies

Correlative sparsity can sometimes not account for the full structure of $\mathbb{A}$, especially when a significant amount of term sparsity exists. To overcome this, different hierarchies to define the structure of the Gram matrix and monomial basis have been proposed. Most notably is term sparse SOS (TSSOS), chordal-TSSOS and correlative sparsity-TSSOS (CS-TSSOS), which can exploit term sparsity even when f is not necessarily correlatively sparse [207, 208]. The general technique used in these approaches is to iteratively update the graph to form the hierarchy, starting from the fact that each edge set should contain at least all edges (β, γ) with $\beta + \gamma \in \mathbb{A}$ since it guarantees that $\mathbb{A} \subseteq \text{supp}((x^{\mathbb{B}})^T Q x^{\mathbb{B}})$. The details of these iterative schemes are explained in Appendix A.

5.2.5 Extension to Semi-algebraic Sets

The above methods to construct SOS decompositions by exploiting sparsity only show global nonnegativity, however in the NN verification problem we require this theory to be applied to semi-algebraic sets and establish local nonnegativity. Consider the semi-algebaric set with m polynomial inequalities such that

$$S := \{x \in \mathbb{R}^n \, : \, g_1(x) \geq 0, \ldots, g_m(x) \geq 0\}.$$

Using the Psatz as in (2.5) it is possible to verify that $f \in \mathbb{R}[x]$ is nonnegative on S. Consider a series of exponent sets associated with each inequality constraint $\mathbb{B}_0, \ldots, \mathbb{B}_m \subseteq \mathbb{N}^n_\omega$ where ω is known as the relaxation order. The Psatz states that f is nonnegative on S if

$$f(x) = \sum_{i=0}^{m} g_i(x)(x^{\mathbb{B}_i})^T Q_i x^{\mathbb{B}_i}, \quad Q_i \in \mathbb{S}_+^{|\mathbb{B}_i|}. \tag{5.6}$$

The relaxation order ω is a parameter to be chosen and the higher its value the more accurate the solution will be, however it will increase the size of the SDP in doing so. One must be careful not to make ω too large otherwise the problem may become intractable, but not too small to sacrifice solution accuracy significantly. A common choice for ω is

$$2\omega \geq \max\{\deg(f), \deg(g_1), \ldots, \deg(g_m)\},$$

such that the exponent set becomes

$$\mathbb{B}_i = \mathbb{N}^n_{\omega_i}, \text{ where } \omega_i := \omega - \lceil \frac{1}{2} \deg(g_i) \rceil$$

and $\lceil \cdot \rceil$ is the smallest integer greater than or equal to its value.

For global nonnegativity we only need to consider the sparsity graph of f, however the challenge now is that we must consider how the sparse polynomial multiplier $(x^{\mathbb{B}_i})^T Q_i x^{\mathbb{B}_i}$ interacts with the corresponding inequality constraint g_i. If the $\mathbb{B}_i$ destroys the sparsity in (5.6) then we cannot exploit the sparsity in f or the g_i inequality constraints. This is not a trivial task and requires careful analysis.

If we consider correlative sparsity, to proceed to preserve and analyse the sparsity pattern we replace the correlative sparsity graph of f with a joint correlative sparsity graph of the polynomials $f, g_1, \ldots, g_m$. As before this graph has n vertices, however the edges are constructed differently. An edge between vertices i and j exists if at least one of the following are true:

Condition 1. The variables x_i and x_j are multiplied together in f.

Condition 2. At least one of the $g_1, \ldots, g_m$ depends on both x_i and x_j, even if these variables are not multiplied together, (x_i and x_j simultaneously appear in some $g_1, \ldots, g_m$).

Therefore, the support of $g_i(x)(x^{\mathbb{B}})^T Q_i x^{\mathbb{B}}$ must be consistent with the joint correlative sparsity graph, where each matrix Q_i is the densest possible matrix. The maximal cliques of the joint correlative sparsity graph are defined to be $\mathcal{J}_1, \ldots, \mathcal{J}_t$. Through condition 2, we can be sure that there is at least one clique $\mathcal{J}_k$ such that $\text{var}(g_i) \subseteq \mathcal{J}_k$, where $\text{var}(g_i) \subset \{1, \ldots, n\}$ is the set of indices of the variables on which g_i depends. The set of clique indices k for which this holds is denoted by

$$\mathcal{N}_i := \{k \in \{1, \ldots, t\} \ : \ \text{var}(g_i) \subseteq \mathcal{J}_k\}.$$

The sparsity graph $\mathcal{G}_i(\mathbb{B}_i, \mathcal{E}_i)$ of Q_i is defined to have the edge set

$$\mathcal{E}_i := \bigcup_{k \in \mathcal{N}_i} \{(\beta, \gamma) \in \mathbb{B}_i \times \mathbb{B}_i \ : \ \text{nnz}(\beta + \gamma) \subseteq \mathcal{J}_k\}.$$

The maximal cliques of the sparsity graph are $\mathcal{C}_{i,1}, \ldots, \mathcal{C}_{i,|\mathcal{N}_i|}$, where $\mathcal{C}_{i,k} := \{\beta \in \mathbb{B} \ : \ \text{nnz}(\beta) \subseteq \mathcal{J}_k\}$. The sparse matrix decomposition of Q_i is therefore

$$Q_i = \sum_{k=1}^{|\mathcal{N}_i|} E_{\mathcal{C}_{i,k}}^T Z_k E_{\mathcal{C}_{i,k}}, \quad Z_k \in \mathbb{S}_+^{|\mathcal{C}_{i,k}|}.$$

Consequently this generates the representation

$$f(x) = \sum_{i=0}^{m} \sum_{k \in \mathcal{N}_i} g_i(x) \sigma_{i,k}(x_{\mathcal{J}_k}),$$

where $\sigma_{i,k}(x_{\mathcal{J}_k}) \in \Sigma[x]$ contains only variables that are indexed by a single clique in the joint correlative sparsity graph.

5.3 Sparse Neural Network Constraints

To take advantage of the sparsity in the NN verification problem, we note that common constraints that are used to bound the activation functions only contain variables in the current layer and the previous layer. This means that the constraint matrices have a very well-defined structure. We show how this sparsity is translated into SOS constraints and how the cliques are formed with a simple example.

5.3.1 Example: Three Layer, Single Node Neural Network

Consider a single input-output NN with three layers and a single node in each layer. The equations for this NN are

$$\begin{aligned} x_0 &= u, \\ x_1 &= \phi(W^0 x_0 + b^0), \\ x_2 &= \phi(W^1 x_1 + b^1), \\ x_3 &= \phi(W^2 x_2 + b^2), \\ y &= W^3 x_3 + b^3. \end{aligned}$$

For the constraints we use the notation $g_{i,j}$ and $h_{i,j}$ to represent the jth inequality and equality constraint respectively in the ith layer. We set the input to be bounded by $[-1, 1]$, therefore the input constraints are

$$\begin{aligned} g_{0,1}(x_0) &= x_0 + 1 \geq 0, \\ g_{0,2}(x_0) &= 1 - x_0 \geq 0. \end{aligned}$$

For now we will only consider the ReLU activation function in the hidden layers. We will use the quadratic constraints as outlined in (2.10) such that

$$\begin{aligned} x_i &\geq 0, \\ x_i - (W^{i-1} x_{i-1} + b^{i-1}) &\geq 0, \\ x_i(x_i - (W^{i-1} x_{i-1} + b^{i-1})) &= 0, \text{ for } i = 1, 2, 3. \end{aligned}$$

Note that we are not considering slope constraints in this formulation, as the constraints will no longer be a function of two consecutive layers, since the slope constraints consider any two activation functions in the NN. This will destroy the sparsity in the algorithm formulation that is presented in this section.

If we start with the simplest case and choose the multipliers $t_{i,j}, s_{i,j}$ to be single variables ($\omega = 0$), then the joint correlative sparsity graph will be a line graph as shown in Figure 5.2. This graph is chordal and has four cliques, which are written as

$$\mathcal{J}_1 = \{0, 1\},\ \mathcal{J}_2 = \{1, 2\},\ \mathcal{J}_3 = \{2, 3\},\ \mathcal{J}_4 = \{3, 4\}.$$

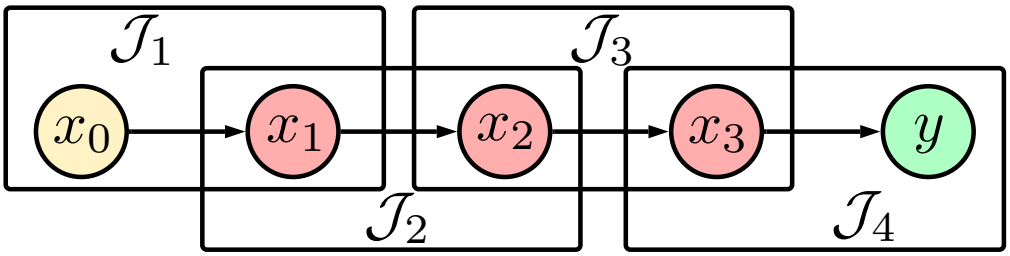


Figure 5.2: Diagram of a neural network showing how the cliques are formed for the example in Section 5.3.1.

We now consider what happens when we increase the relaxation order and how we can select the correct exponents. Condition 1 states that an edge between vertices i' and j' is connected if the variables $x_{i'}$ and $x_{j'}$ are multiplied together in f. Since $f = g_{out}^m = \gamma_m - c^T y$ is a linear function in the output variables, condition 1 will never be true and hence no edges will be constructed from it. Therefore, our attention turns to the second condition that states a connection exists if at least one of $g_{i,j}$, $\forall i, j$ depends on both $x_{i'}$ and $x_{j'}$ even if these variables are not multiplied together. And so for the connection between x_0 and x_1, the only constraints that depend on these two variables are the constraints in the first layer i.e. $g_{1,2}$ and $h_{1,1}$. Recall that through condition 2, we can be sure that there is at least one clique $\mathcal{J}_k$ such that $\text{var}(g_{i,j}) \subseteq \mathcal{J}_k$, where $\text{var}(g_{i,j}) \subset \{1, \ldots, n\}$ is the set of indices of the variables on which $g_{i,j}$ depends. The set of cliques for which this holds is denoted by

$$\mathcal{N}_{i,j} := \{k \in \{1, \ldots, t\} \ : \ \text{var}(g_{i,j}) \subseteq \mathcal{J}_k\}.$$

Let $Q_{i,j}$ be the Gram matrix for the jth constraint in the ith layer. We now impose that the $Q_{i,j}$ matrix is as dense as possible such that the support of $g_{i,j}(x)(x^{\mathbb{B}_{i,j}})^T Q_{i,j} x^{\mathbb{B}_{i,j}}$ is consistent with the joint correlative sparsity graph. The sparsity graph of $Q_{i,j}$ is defined to have the edge set

$$\mathcal{E}_{i,j} := \bigcup_{k \in \mathcal{N}_{i,j}} \{(\beta, \gamma) \in \mathbb{B}_{i,j} \times \mathbb{B}_{i,j} \ : \ \text{nnz}(\beta + \gamma) \subseteq \mathcal{J}_k\}.$$

In this case the only clique where $k \in \mathcal{N}_{i,j}$ is the clique corresponding to the layer that represents that layer and the proceeding layer and hence only overlaps with a

single clique such that

$$\mathcal{E}_{i,j} := \{(\beta, \gamma) \in \mathbb{B}_{i,j} \times \mathbb{B}_{i,j} \ : \ \mathrm{nnz}(\beta + \gamma) \subseteq \mathcal{J}_{\mathcal{N}_{i,j}}\}.$$

Each $Q_{i,j}$ will then only be a function of the variables in the ith and $(i-1)^{\text{th}}$ layer, which matches the cliques. The monomial vectors for multipliers for relaxation order $\omega = 2$ in the cliques $\mathcal{J}_1$, $\mathcal{J}_2$, $\mathcal{J}_3$, $\mathcal{J}_4$ are

$$[x_0, x_1, 1],\ [x_1, x_2, 1],\ [x_2, x_3, 1],\ [x_3, x_4, 1],$$

respectively. This simple example has shown how the chordal structure can be preserved when applying the multipliers, we now generalise this example to any feed-forward NN to show that the sparsity still holds.

5.3.2 General Neural Network

To proceed from the simple example in Section 5.3.1, we introduce new notation. $g_{i,j,k}$ represents the kth inequality constraint of the jth node in the ith layer, $h_{i,j,k}$ has the equivalent meaning for equality constraints. Each node is now represented by x_j^i, where i is the layer index and j is the node index in that layer.

For an NN of any size there can be more than one input, hence there will be $2n_u$ inputs constraints, which can then be written as

$$\begin{aligned} g_{0,j,1}(x_j^0) &= x_j^0 - \underline{u}_j \geq 0, \\ g_{0,j,2}(x_j^0) &= -x_j^0 + \overline{u}_j \geq 0,\ \forall j = 1, \ldots, n_u. \end{aligned}$$

We note that often these constraints are in the form $g_{i,j,k} = g_{i,j,k}(x_j^i, x_1^{i-1}, \ldots, x_{n_{i-1}}^{i-1})$, meaning that each constraint is only a function of the respective node and all of the variables in the previous layer. Therefore, the connection between x_j^i and $x_{j'}^{i-1}$ is established due to the $g_{i,j,k}$ constraint, where j' is any node in the $(i-1)^{\text{th}}$ layer. If we follow this argument for all of the nodes in the network then the joint correlative sparsity graph consists of all the nodes in each layer being joined to the neighbouring layers: this is essentially the structure of the NN equations.

The maximal cliques of the NN verification problem consist of the overlapping layers such that for an NN with ℓ layers, the verification problem can be split into $\ell + 1$ maximal

cliques. This is similar as in the example case in Section 5.3.1 but instead there are multiple nodes in each layer, however this does not greatly impact the sparsity graph of the multiplier $Q_{i,j,k}$. The edge set is given by

$$\mathcal{E}_{i,j,k} := \{(\beta,\gamma) \in \mathbb{B}_{i,j,k} \times \mathbb{B}_{i,j,k} \ : \ \mathrm{nnz}(\beta+\gamma) \subseteq \mathcal{J}_{\mathcal{N}_{i,j,k}}\}.$$

The joint correlative sparsity graph for a general NN is shown in Figure 5.3 and Figure 5.4. Note that as the number of layers in the NN increases then so do the number of maximal cliques and hence the number of positive semidefinite constraints increases. As the number of nodes in each layer becomes larger then the size of these positive semidefinite constraints increases.

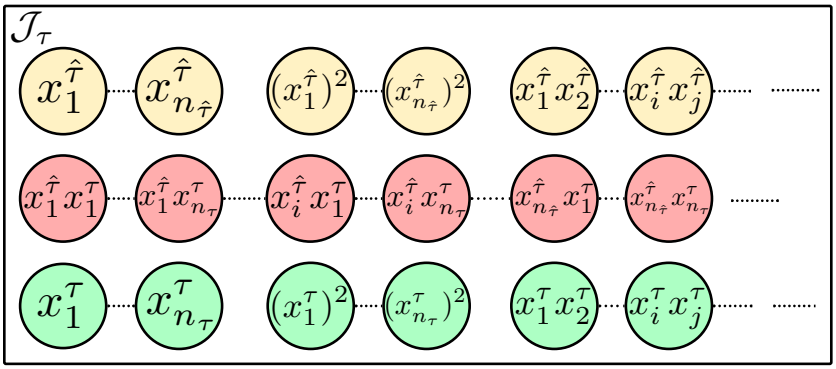


Figure 5.3: Diagram showing the variables that are contained in a generic clique $\mathcal{J}_\tau$. The yellow, green and red nodes are the monomials from layer $\hat\tau$, the next layer $\tau = \hat\tau + 1$ and overlapping terms respectively.

We can now use the joint correlative sparsity graph and the constraints to form the Psatz condition to verify a general NN. The optimisation problem is

$$\begin{aligned}
\min \quad & \gamma_m \\
\text{s.t.} \quad & -c_m^T y + \gamma_m - \sum_{i=0}^{\ell}\sum_{j=1}^{n_i}\sum_{k=1}^{p_{i,j}} t_{i,j,k}h_{i,j,k} - \sum_{i=0}^{\ell}\sum_{j=1}^{n_i}\sum_{k=1}^{q_{i,j}} s_{i,j,k}g_{i,j,k} \text{ is (CS-)TSSOS}, \\
& s_{i,j,k} \text{ is SOS}, \ \forall\, i = 0,\dots,\ell,\ j = 1,\dots,n_i,\ k = 1,\dots,p_{i,j}, \\
& t_{i,j,k} \in \mathbb{R}[x], \ \forall\, i = 0,\dots,\ell,\ j = 1,\dots,n_i,\ k = 1,\dots,q_{i,j},
\end{aligned} \tag{5.7}$$

where $p_{i,j}$ and $q_{i,j}$ are the number of equality and inequality constraints in the ith layer and j^{th} node respectively. The multipliers $s_{i,j,k}$ and $h_{i,j,k}$ are determined by $(x^{\mathbb{B}_{i,j,k}})^T Q_{i,j,k} x^{\mathbb{B}_{i,j,k}}$ using the joint correlative sparsity graph.

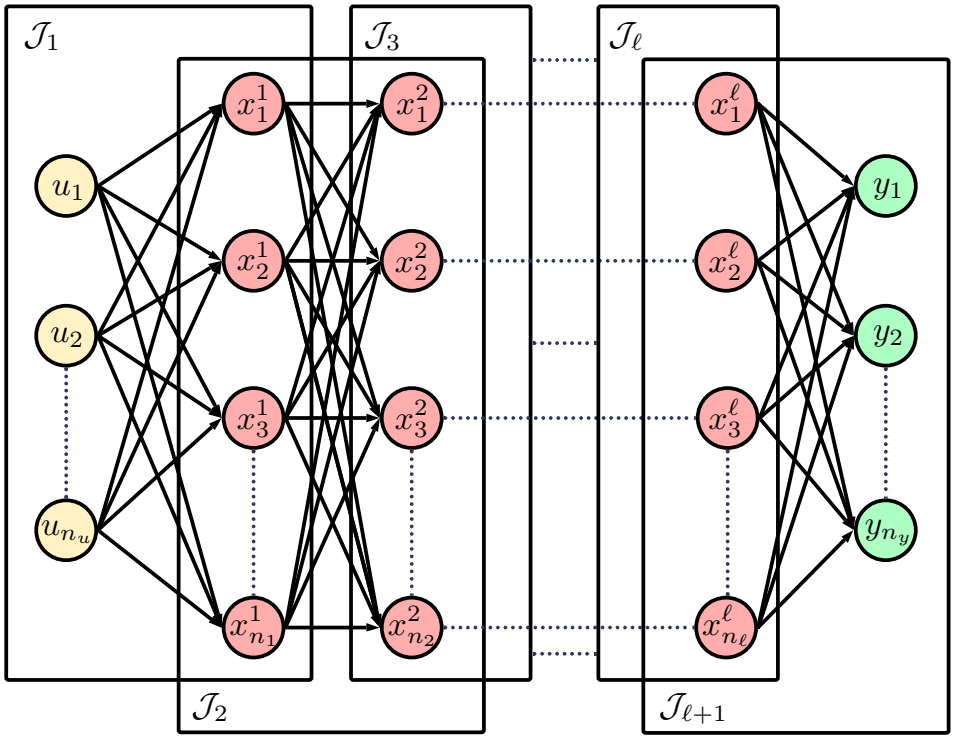


Figure 5.4: Diagram of a general neural network showing how the cliques are formed. The yellow nodes are the input nodes, the red nodes are the hidden layers and the green nodes are the output nodes.

5.4 Numerical Results

Now we examine how our approach performs in experimentation. There are two main aspects to focus on, the first is how this method can improve computational time against an increasing NN size and the second is how the accuracy of the bounds can be tightened.

All experiments were run on a four-core Intel Xeon processor @3.50GHz with 16GB of RAM. We refer to our method as 'NNSparsePsatz', which is built upon the method 'NNPsatz'. We implement our method by using MATLAB to create the NN parameters, which are randomly generated from a Gaussian distribution. These parameters are then parsed into Julia where the semi-algebraic constraint set is constructed. We implement the optimisation problem (5.7) with the CS-TSSOS hierarchy [207] using the TSSOS Julia package [211], which constructs the SDP constraints and parses it to the SDP solver MOSEK [212]. We used MOSEK as the SDP solver since overall it produced the fastest solve times and the most accurate solutions for all methods tested.

To show the trade-off between computational time and accuracy that is possible with this approach, we compare the results when setting the order of the multipliers to second order polynomials against setting them to their minimum level which usually sets their order to zero. We compare the sparse method with NNPsatz [66], the equivalent method which does not exploit sparsity and is implemented with SOSTOOLS in MATLAB and also with MOSEK to solve the SDP. For the experiments with ReLU activation functions, we compare our method to the MATLAB package DeepSDP [60] with MOSEK, which uses an SDP framework without any sparsity. We also compare our approach to another sparsity exploiting method LayerSDP [117], which is the previous best performing SDP formulation. The main contribution of this work is to show how SDP-based NN verification problems can have improved computational time and accuracy. However, we also compare our method to the state-of-the-art solvers α-β-CROWN [103] (the winner of VNN-COMP [101]) and DeepPoly [132]. In these experiments we consider NNs of less width and greater depth, since [203] showed that depth is more important than width in the expressive power of NNs, this will also highlight the benefits of the chordal decomposition.

5.4.1 Scalability Comparison

There are two main ways that we can assess the scalability of the NN verification problem. One approach is to vary the number of layers in the network and the other is to vary the number of nodes in each layer.

Consider a single input, single output NN with two nodes in each layer, we will change the size of the NN by increasing the number of layers. We set the size of the input space of all examples to be [0.5,1.5]. We show the results for ReLU, sigmoid and tanh activation functions in Figures 5.5a, 5.5b and 5.5c respectively. For the ReLU activation function it is shown that the NNSparsePsatz with minimum order provides significantly better results when compared with DeepSDP, LayerSDP and NNPsatz. NNSparsePsatz with second order multipliers also performs similarly well to DeepSDP and LayerSDP. When testing α-β-CROWN we ran into unrecoverable numerical issues when the number of layers was greater than 400. However, from the experiments that we were able to test we can see that α-β-CROWN scales similarly to NNSparsePsatz. For the sigmoid and tanh activation

functions, we see that NNSparsePsatz scales significantly better than α-β-CROWN and NNPsatz. Note that we have not included DeepPoly in the scalability comparisons due to its accuracy being significantly worse than the other methods that are being analysed.

We now increase the size of the layers to eight nodes in each layer, the remaining NN parameters are the same as the two node example. Since there are more nodes in each layer, the size of each clique increases, meaning the problem does not scale as well as the previous example. However, there is still a significant improvement over other methods as we can see in Figures 5.6a, 5.6b and 5.6c. We see that for the ReLU case, NNSparsePsatz with second order multipliers scales poorly, however with minimum order it scales significantly better then NNPsatz, LayerSDP and DeepSDP. α-β-CROWN scales similarly to NNSparsePsatz with minimum order multipliers, however we encountered the same numerical issues when the number of layers was greater than 400 as in the above experiment with two nodes in each layer. For the sigmoid and tanh activation functions, NNSparsePsatz with minimum order multipliers scales significantly better than NNPsatz and α-β-CROWN, however with second order multipliers it scales worse.

We now see what happens when we vary the size of each layer. The number of layers will be fixed to 100 and the number of nodes in each of these layers will be increased. The methods are compared for ReLU, sigmoid and tanh functions and are shown in Figures 5.7a, 5.7b and 5.7c respectively. For the ReLU activation function, NNSparsePsatz scales better than DeepSDP, LayerSDP and α-β-CROWN, whereas NNPsatz exceeds the solve time limit after only two nodes. Similarly in the sigmoid and tanh case, there is a significant improvement using NNSparsePsatz over NNPsatz. α-β-CROWN took over 1000s to solve when there was only one node in the NN, therefore it is not shown on the graphs.

5.4.2 Accuracy Comparison

Having shown that the CS-TSSOS hierarchy can improve the scalability of the problem, we now turn our attention to the accuracy of the methods. We consider a two input-output NN with various sizes for the different activation functions. For the ReLU activation function we use a five layer NN with eight nodes in each layer. It is shown in Figure 5.8a that the NNSparsePsatz method with second order polynomials greatly

improves the tightness of the bounds on the output set such that they are near optimal. If we instead used minimum order multipliers in NNSparsePsatz, then the accuracy would be the same as DeepSDP. Also note that LayerSDP produces the same bounds as DeepSDP. NNSparsePsatz with second order multipliers is more computationally expensive than the other approaches, taking 688s to solve. NNSparsePsatz with minimum order mulitpliers, LayerSDP, DeepSDP, α-β-CROWN and DeepPoly took 3.3s, 15.5s, 3.9s, 4.7s and 0.02s to solve respectively. This shows that using higher order multipliers in our formulations leads to tighter bounds with the trade-off of more computation compared to competitive methods.

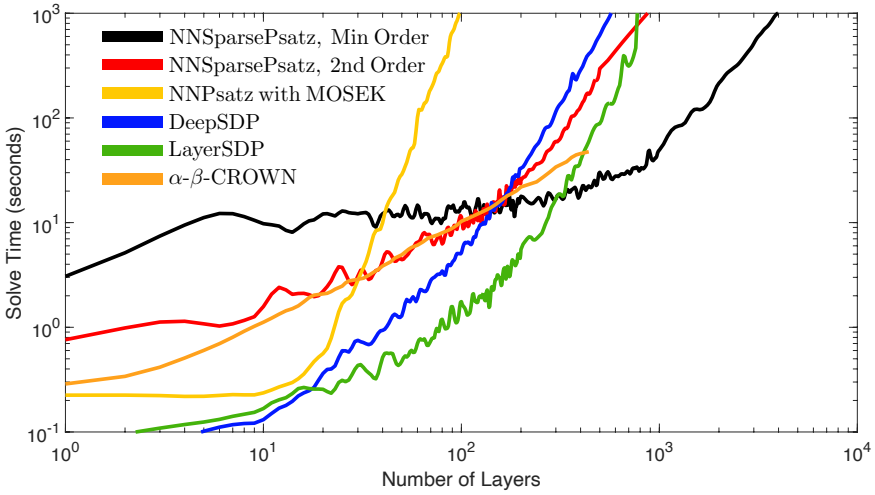
To evaluate what happens when the number of nodes increases, we consider an NN with sigmoid activation functions with 25 nodes in each layer and five layers. Using second order multipliers in NNSparsePsatz becomes too computationally expensive, so we therefore use minimum order multipliers. This approach is still able to outperform the accuracy of the state-of-the-art solvers, as shown in Figure 5.8b. The computational time of NNSparsePsatz was 71.0s, which is in the same order of magnitude as α-β-CROWN (38.9s) but significantly slower than DeepPoly (0.05s), which is known to be conservative but efficient.

Finally we consider a deeper NN with twelve layers and five nodes in each layer with tanh activation functions. The solve times were 462s, 2.0s, 208s, 0.2s for NNSparsePsatz (second order), NNSparsePsatz (minimum order), α-β-CROWN and DeepPoly respectively. This computational time is reflected in the output set as shown in Figure 5.8c, demonstrating how NNSparsePsatz can obtain tighter bounds against existing methods.

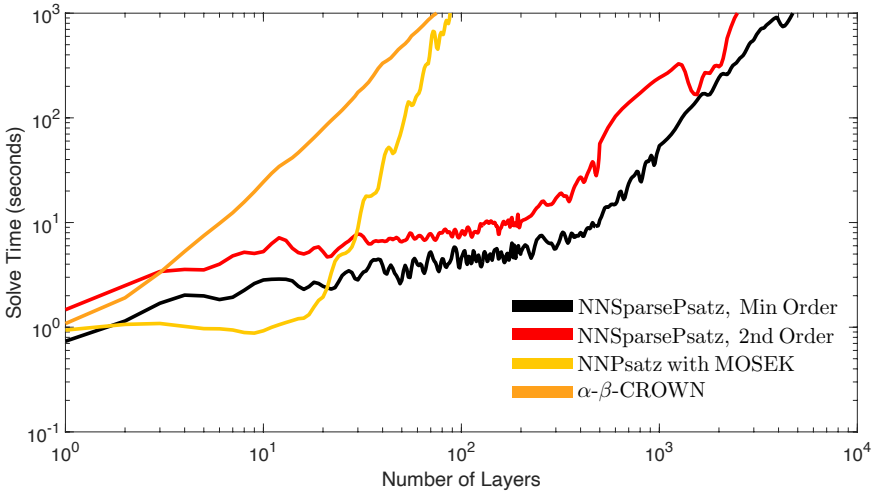
5.5 Conclusion

In this chapter, we propose a framework to solve the NN verification problem, through placing bounds on the non-linear activation functions. Using the Postivstellensatz we are able to trade off solution accuracy with computational time within the optimisation problem. We show that the semi-algebraic set that is constructed possesses a significant amount of sparsity and hence the scalability of this method can be improved by using ideas from sparse polynomial optimisation. Most constraints that are used in problems of this

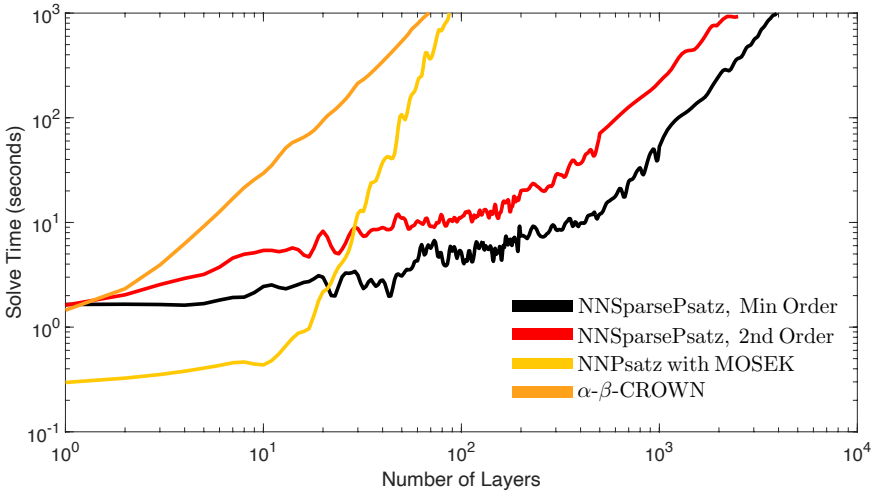
type will exhibit a sparsity pattern, which essentially comes from the natural cascading structure of the NN. We then implement the optimisation framework using the CS-TSSOS hierarchy and show that it improves the computation time to solve the resulting SDP against similar methods and can do so by tightening the bounds on the NN's outputs.



(a) ReLU Activation Function

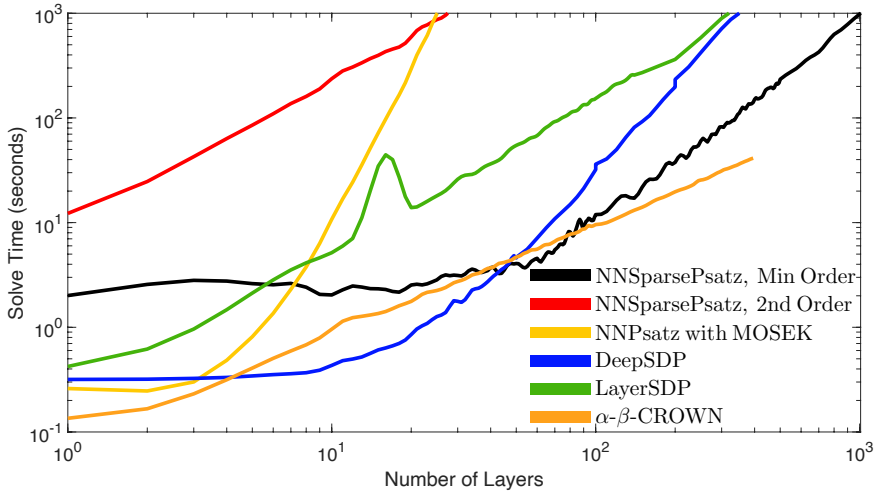


(b) Sigmoid Activation Function

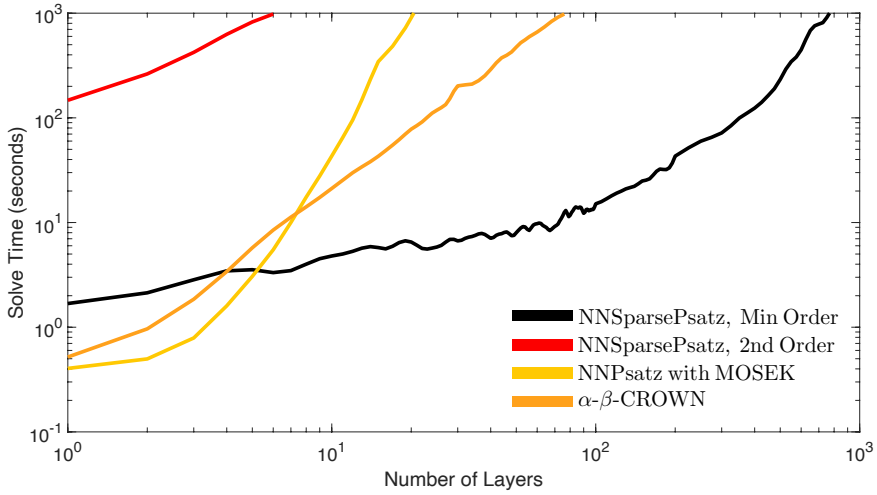


(c) Tanh Activation Function

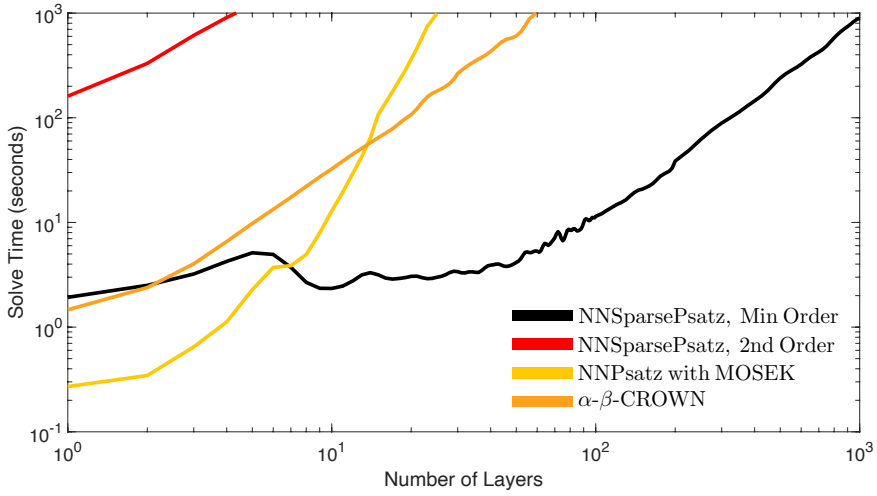
Figure 5.5: Comparing the computational time of different approaches for a neural network with two nodes in each layer and a varying number of layers.



(a) ReLU Activation Function

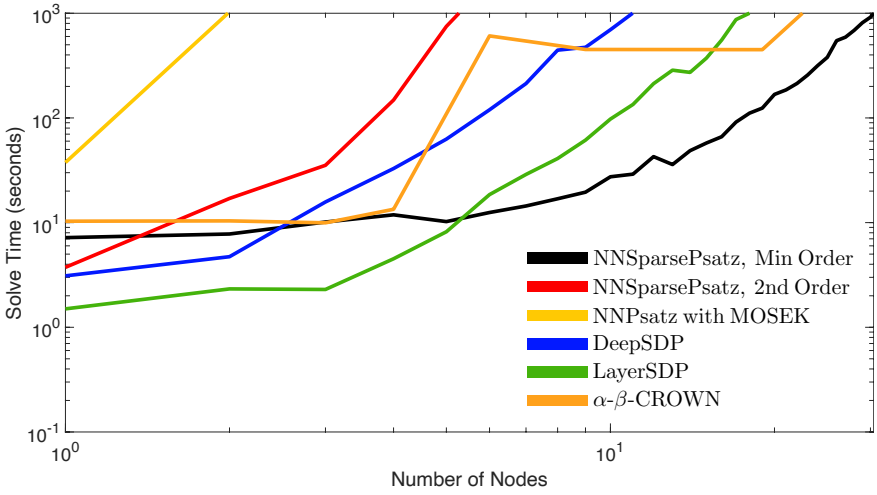


(b) Sigmoid Activation Function

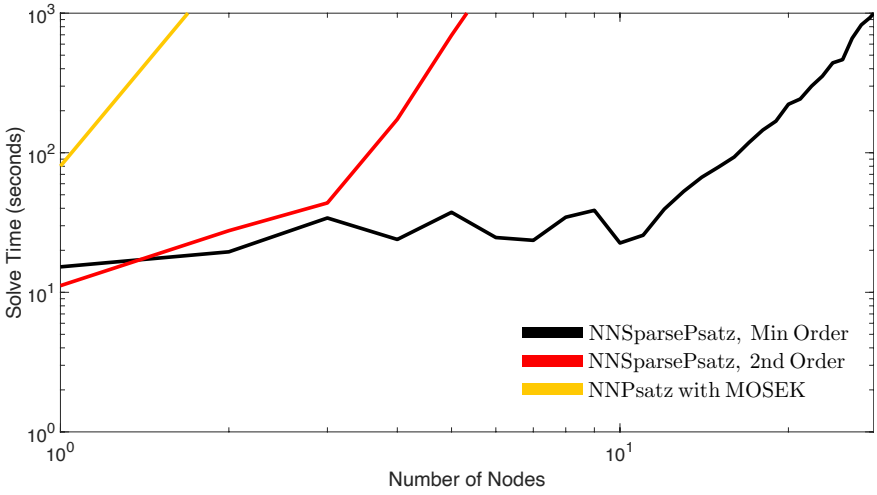


(c) Tanh Activation Function

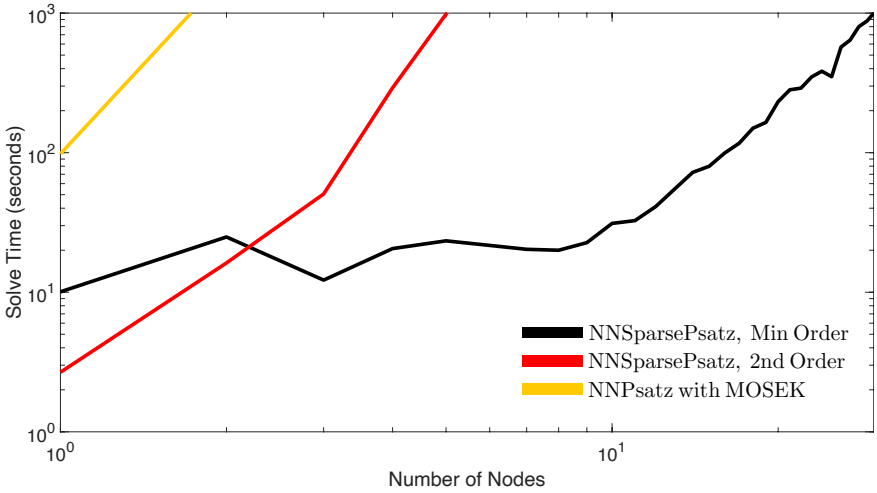
Figure 5.6: Comparing the computational time of different approaches for a neural network with eight nodes in each layer and a varying number of layers.



(a) ReLU Activation Function

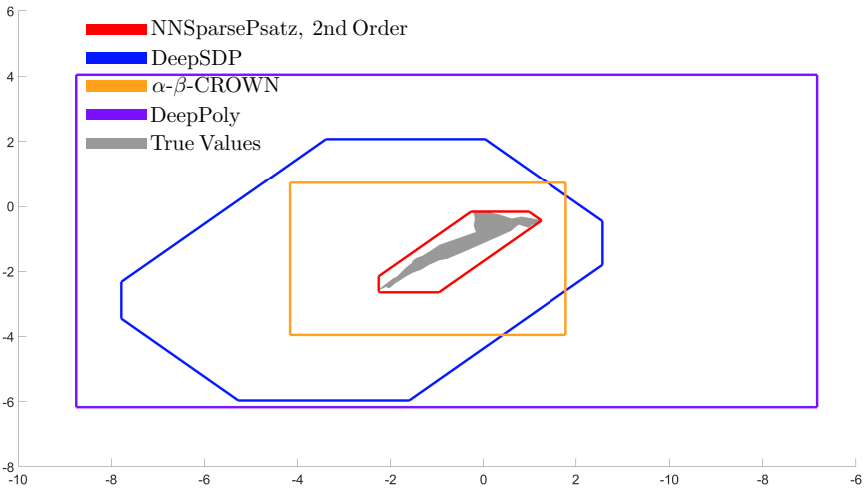


(b) Sigmoid Activation Function

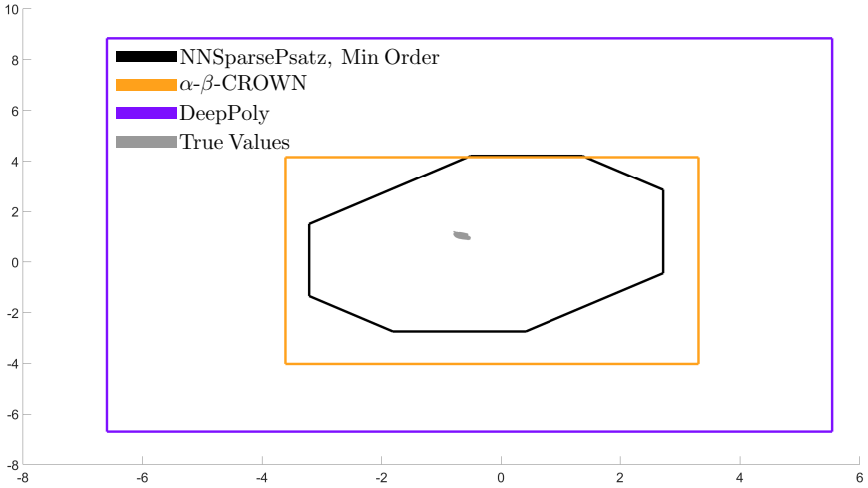


(c) Tanh Activation Function

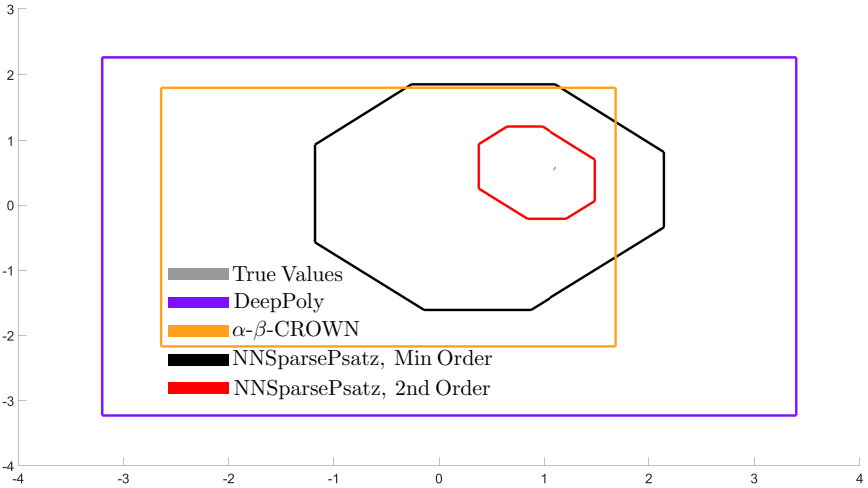
Figure 5.7: Comparing the computational time of different approaches for a neural network with 100 layers and a varying number of nodes.



(a) ReLU Activation Function



(b) Sigmoid Activation Function



(c) Tanh Activation Function

Figure 5.8: Comparing the accuracy of different approaches to the true values of the neural network's output.

6

Reachability Analysis of Neural Feedback Loops

This chapter introduces the neural feedback loop model this is being considered. The neural feedback loop reachability problem is outlined and it is shown how it can be formulated in the sparse polynomial optimisation framework, whilst also demonstrating how the sparsity pattern is preserved. The benefits of this method are then highlighted with numerical examples of the reachability problem applied to non-linear neural feedback loops.

6.1 Introduction

This chapter focuses on the NFL reachability problem in discrete time, where an NN controller is tasked to take the states from an initial to a goal set, whilst avoiding certain regions. The SDP formulation [60] in Chapter 3 was extended to conduct reachability analysis [161]. However, these methods are restricted to the S-procedure and all non-linear dynamics must be over-approximated by sector constraints; this can add significant conservatism into the computed reachable sets. [163] are able to incorporate the non-linear polynomial dynamics by introducing new variables into the optimisation framework, although this does not account for the non-linearities directly.

To overcome these issues we adapt the framework from Chapter 5 and incorporate it within the NFL reachability problem. This approach uses sparse polynomial optimisation methods from [207] along with the Psatz to assert the emptiness of a carefully constructed semi-algebraic set. We then search for the emptiness of this set to find bounds on the reachable set in the same way as in Chapter 4 and Chapter 5 by using SOS and SDP. One advantage of this framework is that we can easily trade off solution accuracy with computational complexity. Due to the polynomial optimisation construction, we can

include any non-linear polynomial dynamics in the plant model, which has not been possible in previous frameworks. This method exploits the natural cascading structure of the NN that we show is inherited in the algorithm formulation. We implement this method on example NFLs that contain NN controllers with ReLU, sigmoid and tanh activation functions and show through numerical experiments the advantages of our method.

6.2 Neural Feedback Loops in Discrete Time

We now propose the NFL model that is used in this chapter, this model is depicted in Figure 6.1. Consider a discrete-time time-invariant system

$$z_{t+1} = f(z_t, u_t), \tag{6.1}$$

where f is a known Lipschitz function and $z_t \in \mathbb{R}^{n_z}$, $u_t \in \mathbb{R}^{n_u}$ are the system states and control inputs respectively. We consider a state feedback controller $u_t = \pi(z_t) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ as a feed-forward fully connected NN, which is governed by the set of equations

$$\begin{aligned} x^0 &= z_t, \\ v^k &= W^k x^k + b^k, \text{ for } k = 0, \dots, \ell - 1, \\ x^{k+1} &= \phi(v^k), \text{ for } k = 0, \dots, \ell - 1, \\ \pi(z_t) &= W^\ell x^\ell + b^\ell, \end{aligned} \tag{6.2}$$

where $W^k \in \mathbb{R}^{n_{k+1} \times n_k}$, $b^k \in \mathbb{R}^{n_{k+1}}$ are the weights matrix and bias of the $(k+1)^{\text{th}}$ layer respectively and $z_t = x^0 \in \mathbb{R}^{n_z}$ is the input into the NN. Note that while each x^k and v^k should be indexed by t, we drop this notation for convenience. The number of neurons in the kth layer is denoted by n_k. The activation function ϕ is applied element-wise to the v^k terms such that

$$\phi(v^k) := [\phi(v_1^k), \dots, \phi(v_{n_k}^k)]^T, \; v^k \in \mathbb{R}^{n_k},$$

where v_j^k is the pre-activation value.

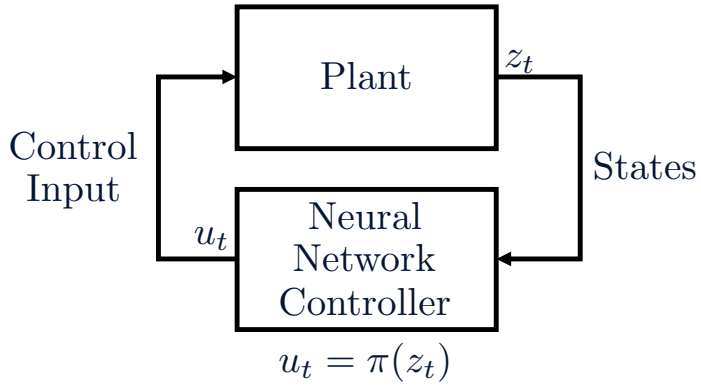


Figure 6.1: Diagram of an NFL governed by (6.1) and (6.2).

6.3 Reachability Problem

For the closed loop system in (6.1), the forward reachable set at time-step t from a set of given initial conditions $\mathcal{Z}_0 \subseteq \mathbb{R}^{n_z}$ is denoted as $\mathcal{R}_t(\mathcal{Z}_0)$. This mapping is shown in Figure 6.2 and is defined by the recursion

$$\begin{aligned} \mathcal{R}_0(\mathcal{Z}_0) &= \mathcal{Z}_0, \\ \mathcal{R}_{t+1}(\mathcal{Z}_0) &= f(\mathcal{R}_t(\mathcal{Z}_0)),\ \forall t. \end{aligned} \tag{6.3}$$

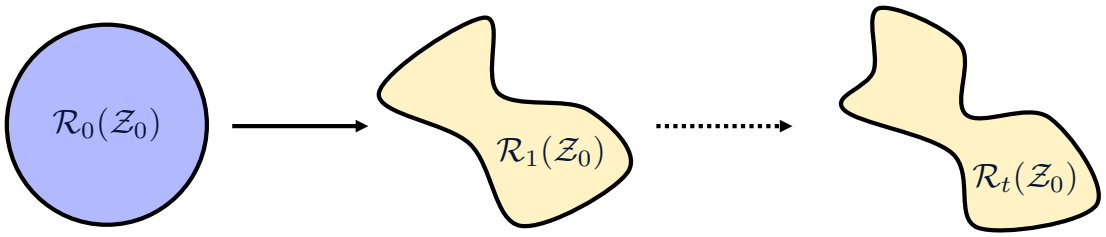


Figure 6.2: Diagram of the reachable sets described by (6.3).

This can be used to formulate the finite-time reach-avoid verification problem. This states that given a goal set $\mathcal{G} \subseteq \mathbb{R}^{n_z}$, an initial state $z_0 \in \mathcal{Z}_0$ and a sequence of avoid sets $\mathcal{A}_t \subseteq \mathbb{R}^{n_z}$, we determine if z_0 can reach $\mathcal{G}$ in finite time while avoiding $\mathcal{A}_t$ for all t in the finite time period. This problem is equivalent to testing if the following are true

$$\begin{aligned} &\mathcal{R}_N(\mathcal{Z}_0) \subseteq \mathcal{G}, \\ &\mathcal{R}_t(\mathcal{Z}_0) \cap \mathcal{A}_t = \emptyset, \forall t = 0, \ldots, N, \end{aligned} \tag{6.4}$$

where N is the finite time horizon. A visual depiction of this problem is shown in Figure 6.3.

Since NFLs contain non-linearities in the activation functions, computing the exact reachable set can be computationally intractable. We can therefore instead compute a

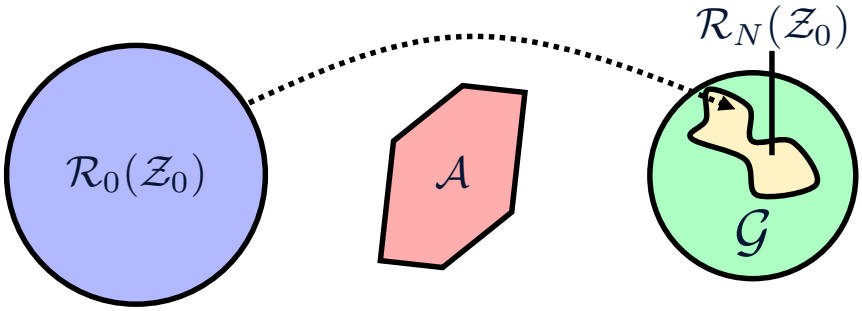


Figure 6.3: Diagram of the finite-time reach-avoid verification problem described by (6.4).

convex outer-approximation of the reachable set denoted by $\overline{\mathcal{R}}_t(\mathcal{Z}_0) \supseteq \mathcal{R}_t(\mathcal{Z}_0)$, shown visually in Figure 6.4. This is useful since if we can verify that $\overline{\mathcal{R}}_t(\mathcal{Z}_0)$ satisfies (6.4) then we know that $\mathcal{R}_t(\mathcal{Z}_0)$ does as well and hence the non-linear NFL is verified. It is desirable that the size of $\overline{\mathcal{R}}_t(\mathcal{Z}_0)$ is as small as possible as it increases the chance of the system satisfying the reachability problem (6.4). Therefore, we can pose this as an optimisation problem such that

$$\begin{aligned} \text{minimize} \quad & \text{Volume}(\overline{\mathcal{R}}_t(\mathcal{Z}_0)), \\ \text{subject to} \quad & \mathcal{R}_t(\mathcal{Z}_0) \subseteq \overline{\mathcal{R}}_t(\mathcal{Z}_0). \end{aligned} \tag{6.5}$$

This can be solved using $N+1$ optimisation problems by first optimising $\overline{\mathcal{R}}_0(\mathcal{Z}_0)$ given $\mathcal{Z}_0$ and then recursively computing the next time step until all N time steps are solved.

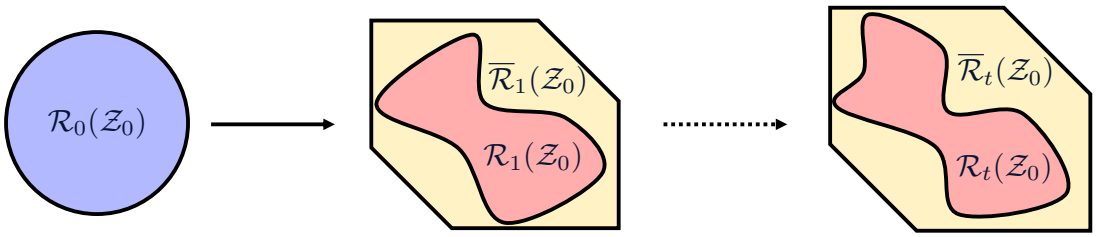


Figure 6.4: Diagram of the convex outer-approximations of the reachable sets that are used in (6.5).

6.4 Neural Feedback Loop Constraints

To solve each optimisation problem, we form a set of constraints based on the input set and the NN structure and then minimize the output set through defining an appropriate objective function. We constrain the input set to be bounded by a polytope such that

$$\mathcal{Z}_t = \bigcap_{m=1}^{M_{\text{in}}} \{z_t \in \mathbb{R}^{n_z} \mid d_m - c_m^T z_t \geq 0\},$$

where $c_m \in \mathbb{R}^{n_z}$, $d_m \in \mathbb{R}$ and M_{in} are the parameters and the number of faces of the polytope respectively. There are other ways of bounding the input such as using ellipsoids or zonotopes, however in this thesis we will only consider polytopes. The reachable set $\overline{\mathcal{R}}_t(\mathcal{Z}_0)$ that represents the region of the states at the next time step z_{t+1} can also be bounded by a polytope such that

$$\mathcal{Z}_{t+1} = \bigcap_{m=1}^{M_{\text{out}}} \{z_{t+1} \in \mathbb{R}^{n_z} \mid b_m - a_m^T z_{t+1} \geq 0\}, \tag{6.6}$$

where $a_m \in \mathbb{R}^{n_z}$ are given parameters of the polytope and b_m can be set as an optimisation variable that is to be optimised. Each reachability problem at time-step t can be split in to a further M_{out} optimisation problems, where each problem aims to push one side of the polytope as close as possible to the exact reachable set.

Recall from (6.1) that $z_{t+1} = f(z_t, \pi(z_t))$. We therefore need to create a set of constraints that represent the NN in (6.2) to bound the controller input $\pi(z_t)$. We use the same ideas from Chapter 2 to abstract the NN into a constraint set that represents its input-output properties. We now show how we can use these constraints and the Psatz as in the previous chapters to create an optimisation problem to minimize the volume of the reachable sets.

6.5 Neural Feedback Loop Reachability Emptiness Condition

To construct the NFL reachability problem we modify the Psatz condition to use it in the optimisation framework. Instead of optimising the output polytope faces in (6.6) such that $b_m - a_m^T z_{t+1} \geq 0$ is feasible, we can show that $-b_m + a_m^T z_{t+1} \geq 0$ is infeasible using the Psatz. We introduce the optimisation decision variable γ_m to approximate b_m and find the limiting values as to when this Psatz condition is violated. The optimisation problem for a given time step t and face m of the output polytope is given by (6.7).

$$\begin{aligned}
\text{minimize} \quad & \gamma_m \\
\text{subject to} \quad & -a_m^T z_{t+1} + \gamma_m - \sum_j^q t_j h_j - \sum_i^p s_i g_i - \sum_{i \neq j}^p r_{ij} g_i g_j - \ldots \text{ is SOS}, \\
& s_i \text{ is SOS}, \ \forall\, i = 1, \ldots, p, \\
& r_{ij} \text{ is SOS}, \ \forall\, i, j = 1, \ldots, p, \\
& t_j \in \mathbb{R}[x], \ \forall\, j = 1, \ldots, q.
\end{aligned} \tag{6.7}$$

Here h_j and g_i are the polynomial equality and inequality constraints that represent the input-output properties of the NN respectively. By choosing higher degree multiplier s_i, t_j etc. we can create a series of nested tests of increasing complexity and non-decreasing accuracy.

6.6 Sparse Reachability Constraints

As discussed in Chapter 3, Chapter 4 and Chapter 5 one issue with using an SDP framework is that the solve time becomes intractable as the size of the NN controller increases. We must therefore use sparsity exploiting methods for the NFL reachability problem. We still have a similar cascading structure of the fully connected feed-forward NN in (6.2). We can use this result for the NFL reachability problem since the sparsity pattern generated by the constraints from the NN is similar. However, instead of bounding the output of the NN as in the NN verification problem, we are bounding the z_{t+1} state given a bound on z_t. As in (6.1) the z_{t+1} state is a function of both the input and output of the NN controller. Therefore, the sparse structure in the graph remains and it is possible to create an extension of the graph that is chordal. This allows the use of sparsity exploiting techniques to decompose the resulting optimisation problem. The sparsity pattern is shown in Figure 6.5 and the sparse optimisation problem is

$$\begin{aligned} \text{minimize} \quad & \gamma_m \\ \text{subject to} \quad & -a_m^T z_{t+1} + \gamma_m - \sum_j^q t_j h_j - \sum_i^p s_i g_i \text{ is CS-TSSOS}, \\ & s_i \text{ is SOS}, \ \forall\, i = 1, \ldots, p, \\ & t_j \in \mathbb{R}[x], \ \forall\, j = 1, \ldots, q, \end{aligned} \tag{6.8}$$

where h_j and g_i are the equality and inequality constraints respectively.

6.7 Numerical Results

We demonstrate our method on various systems to show its advantages and capabilities. Our experiments were run on a four-core Intel Xeon processor @3.50GHz with 16GB of RAM. Our method is referred to as 'ReachSparsePsatz', which solves the optimisation problem (6.8) and is implemented in Julia where the semi-algebraic constraint set is

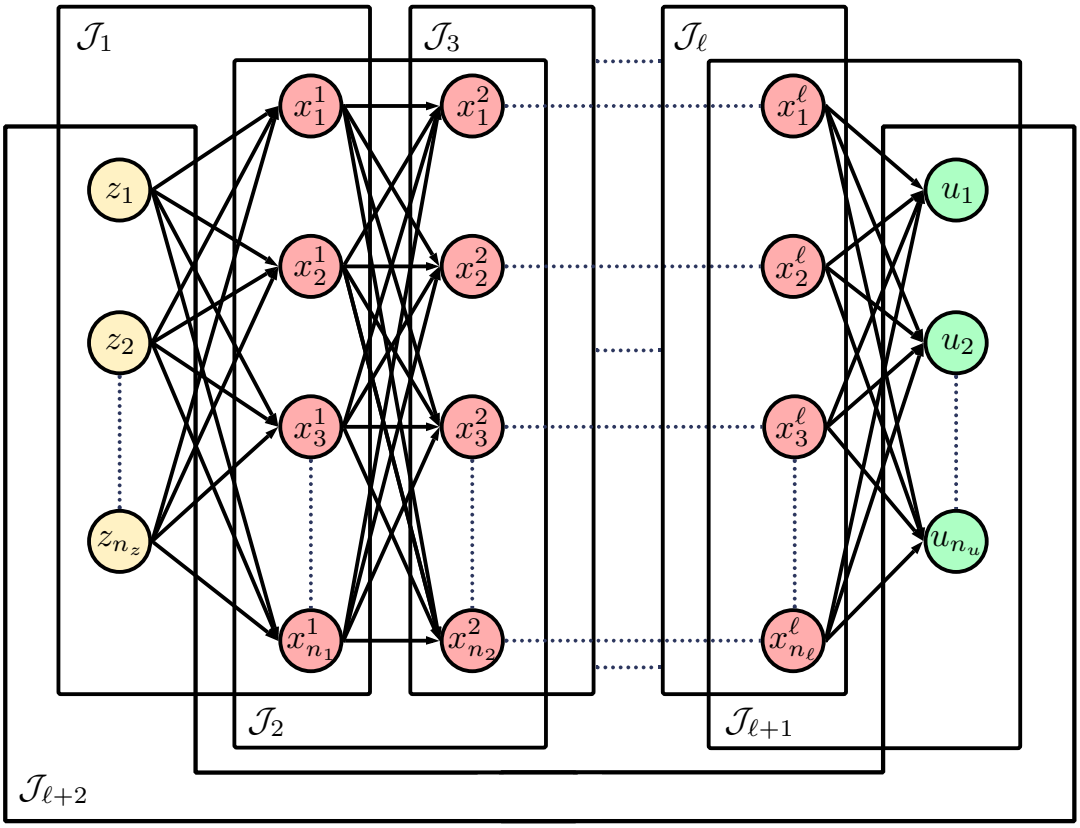


Figure 6.5: Diagram of a neural network controller showing how the cliques $\mathcal{J}_k$ are formed. The yellow nodes are the states z_t, the red nodes are the hidden layers and the green nodes are the control inputs u_t. The clique $\mathcal{J}_{\ell+2}$ represents the connections between the z_t states and u_t, however the connections are not illustrated on the diagram for simplicity.

constructed. The optimisation problem is implemented with the CS-TSSOS hierarchy [207] using the TSSOS Julia package [211], which constructs the SDP constraints and parses them to the SDP solver MOSEK [212]. MATLAB is used to create the NN controller parameters, which are then parsed into Julia. Note that without using sparsity the SDPs for these problems are intractable. The true reachable sets are computed using an exhaustive search on each input set.

6.7.1 Duffing Oscillator

One advantage of this polynomial optimisation framework is that we can use any system (6.1) where f is a polynomial. This is not possible when using similar methods such as [161] and [163], which would require one to either linearise the system or to create an alternative solution to bound the non-linearity. To demonstrate this we take the

Duffing oscillator example proposed in [163] written as

$$\begin{aligned} \dot{z}_1 &= z_2, \\ \dot{z}_2 &= -z_1 - 2\zeta z_2 - z_1^3 + u, \end{aligned} \tag{6.9}$$

where $[z_1, z_2]^T \in \mathbb{R}^2$ and $u \in \mathbb{R}$ are the states and control input respectively. The damping coefficient is set to $\zeta = 0.3$ and we discretized the system with sampling time $\Delta t = 0.3$ using a simple forward Euler scheme. We fit an NN with five layers containing five nodes in each layer with ReLU activation functions as the controller of the system. Each reachable set is a sixteen-sided polytope, taking approximately 11.5s to compute all sixteen optimisation problems. Figure 6.6 shows that our method is able to obtain tight bounds on this non-linear system. This system is stable without a controller, however we have been able to show that we can effectively compute the reachable sets of an NFL with a non-linear plant model.

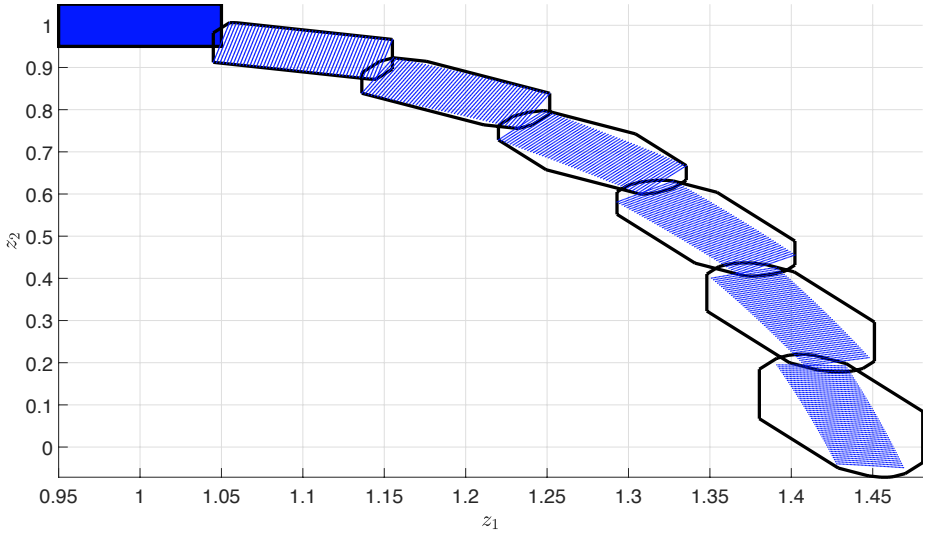


Figure 6.6: Reachable sets of the Duffing oscillator (6.9) using ReachSparsePsatz (black) and the true values (blue).

6.7.2 Unstable Polynomial System

We now consider an unstable system with polynomial dynamics given by

$$\begin{aligned} \dot{z}_1 &= -z_1(0.1 + (z_1 + z_2)^2), \\ \dot{z}_2 &= (u + z_1)(0.1 + (z_1 + z_2)^2). \end{aligned} \tag{6.10}$$

We discretise the system with time step $\Delta t = 0.5$ and fit a large NN controller with 100 layers, three nodes in each layer and ReLU activation functions. Figure 6.7 shows

the reachable sets that are computed at each time step. We can see the approximate reachable sets becoming less accurate at each iteration, however ReachSparsePsatz achieves good results considering the non-linearities that are present in the plant model. Each polytope consists of four sides and took 132s to compute. The solve time is higher in this example due to the large size of the NN.

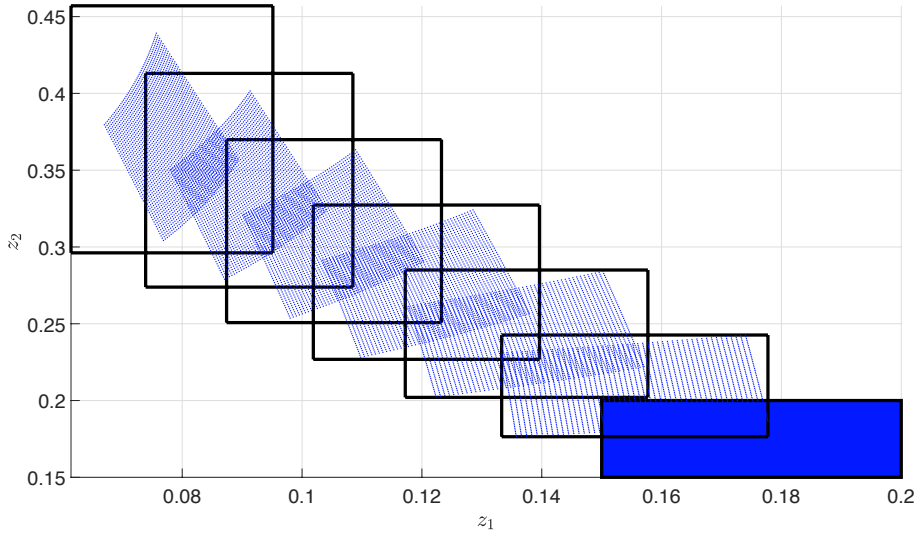


Figure 6.7: Reachable sets of the unstable polynomial system (6.10) using ReachSparsePsatz (black) and the true values (blue).

6.7.3 Inverted Pendulum

We take the inverted pendulum example from [179] and analyse the reachable sets of this system. The inverted pendulum's dynamics can be written in the form

$$\begin{aligned} \ddot{\theta}(t) &= \frac{-mglq(t) + mgl\theta(t) - \mu\dot{\theta}(t) + \text{sat}(u(t))}{ml^2}, \\ q(t) &= \theta(t) - \sin\theta(t), \end{aligned} \tag{6.11}$$

where $\text{sat}(\cdot)$ is the saturation function. We discretise the system with time step $\Delta t = 0.2$. The system is parameterised by $m = 0.15\text{kg}$, $l = 0.5\text{m}$, $\mu = 0.5\text{Nms/rad}$, $g = 10\text{m/s}^2$ and the saturation limit in the saturation function for the controller is set to $u_{\max} = 1$. We bound the non-linearity $q(t)$ by a sector constraint using the maximum and minimum values of θ_t, which is determined by the region of the reachable set at time step t. The sector constraint is shown in Figure 6.8.

We use the NN controller generated in [179], which contains two hidden layers and five nodes in each layer with tanh activation functions. The benefit of our method is

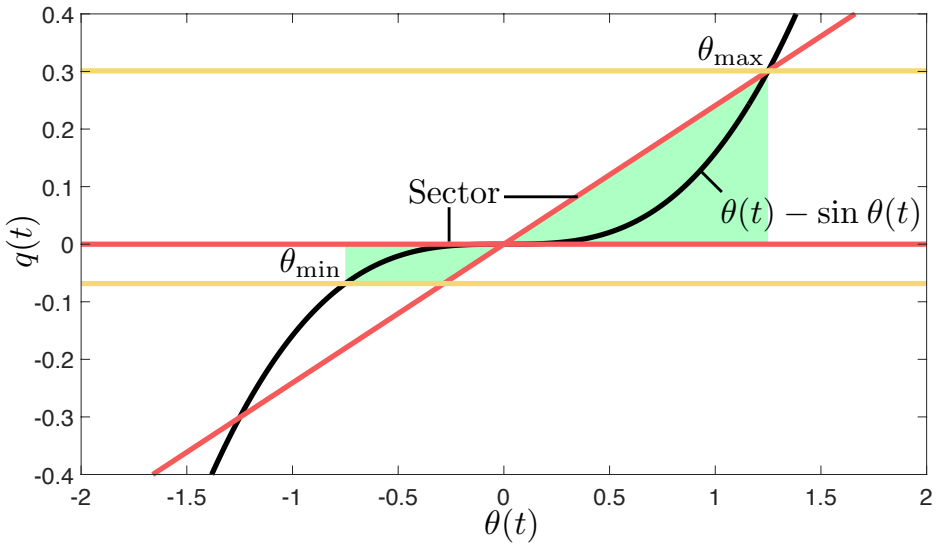


Figure 6.8: Diagram of the sector constraint that bounds the non-linearity in (6.11).

the flexibility of being able to add the sector constraint along with the hidden layer constraints. Figure 6.9 shows the reachable sets when the inverted pendulum is released from the initial condition. We can see that our method is able to obtain tight bounds on the true reachable sets. Our method took approximately 0.4s to compute each reachable set (2.4s for all six reachable sets).

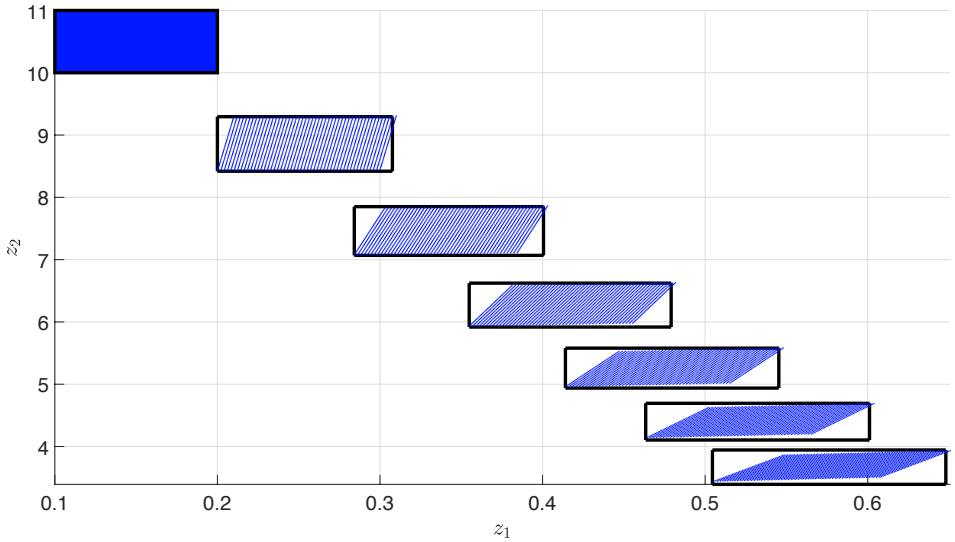


Figure 6.9: Reachable sets of the inverted pendulum (6.11) using ReachSparsePsatz (black) and the true values (blue).

6.8 Conclusion

This chapter proposes a method for computing the reachable sets for non-linear NFLs using sparse polynomial optimisation in conjunction with the Psatz. Our approach uses tight constraints on the ReLU, sigmoid and tanh activation functions in the NN controller to obtain tight polytope over-approximations of the reachable sets. The key advantages of this method are the ability to compute these sets in a scalable and accurate way, whilst also being able to incorporate polynomial dynamics. The benefits of this method are demonstrated through various examples.

7

Stability Analysis of Neural Feedback Loops

This chapter introduces Lyapunov stability and how it can be verified using Sum of Squares programming. It describes how Lyapunov stability can be applied to constrained dynamical systems and how Sum of Squares programming can be used to compute the region of attraction. This theory is applied to neural feedback loops by treating them as non-linear constrained dynamical systems. We show how the region of attraction computed using our method can be increased compared to competitive approaches.

7.1 Introduction

A primary reason why NFLs can be hard to verify as stable is due to the non-linearities that are included in the activation functions in the NN, which are the elements that have made NNs so successful as controllers. In addition approaches that rely on treating the NN nodes as states in the problem run into scalability issues due to the size of NNs often being large. If NNs are to be used in safety-critical control applications, then strong robustness guarantees are required. Therefore, obtaining accurate and reliable robustness measures are important and require the use of expensive computational tests.

Recall that we refer to NFLs that have a non-linear plant as non-linear NFLs. We note that previous methods that verify stability consider systems with linear plant models or bound the plant non-linearities using sector or slope constraints. Two notable examples of these methods are [177, 179], which use sector constraints and integral quadratic constraints to form an SDP-based approach. These methods have shown to be effective, however it is possible to obtain better results by considering the non-linear system directly. In this chapter we use SOS programming to directly incorporate the non-linearities of the plant,

instead of relying on such bounds. Another advantage of this method is that we are able to use higher-order multipliers in the Lyapunov constraints, which usually results in better tests. We consider the system in continuous time, as it does not require us to discretise the system and introduce approximations/conservativeness into the framework. Similar works from [180] have used SOS methods in this context, however they have only looked at the system in discrete time and do not provide analysis on the region of attraction. Section 2.3.2 provides a more detailed account of the previous literature on the stability of NFLs.

7.2 Stability Theory

The stability of a system concerns the long-term behaviour of the states and aims to evaluate the trajectories of the system under small disturbances/perturbations [213].

Definition 7.1. ([214]) The equilibrium point $z^\star = 0$ of $\dot{z} = f(z)$ is:

- Stable if for each $\epsilon > 0$ there is $\delta > 0$ (dependent on ϵ) such that

$$\|z(0)\| < \delta \Rightarrow \|z(t)\| < \epsilon, \forall t \geq 0.$$

- Unstable if it is not stable.

- Asymptotically stable if it is stable and δ can be chosen such that

$$\|z(0)\| < \delta \Rightarrow \lim_{t\to\infty} z(t) = 0.$$

Definition 7.1 states that the trajectories of the states that begin within a close region of the equilibria will remain close to that point for all future times. There are additional definitions of stability that refine this definition, such as exponential stability, absolute stability and global asymptotic stability. We will assume that $z^* = 0$ throughout this chapter without loss of generality.

7.2.1 Linear Stability

Consider a system defined by

$$\dot{z} = f(z),$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is sufficiently smooth such that solutions exist and are unique. Suppose $f(z^\star) = 0$, i.e. the system has an equilibrium at $z = z^\star$. The linearisation of this system around $z^\star$ is given by $\dot{z} = Az$, where $A = \frac{\partial f}{\partial z}|_{z=z^\star}$. The linearised system is asymptotically stable if the eigenvalues of A have negative real parts [215]; in this case, the equilibrium of the non-linear system is also locally asymptotically stable.

Linear stability is restrictive as it involves analysing the system ignoring nonlinearities, therefore it may only be valid within a small region about the equilibrium. This means that it is not a robust method for determining the long-term behaviour in a global setting, although it is a very powerful tool to quickly analyse some of the properties of the system.

7.2.2 Lyapunov Stability

The theory of Lyapunov functions can be used to determine the stability of non-linear systems further away from the equilibrium [215].

Theorem 7.2. ([214]) Let $z^* = 0$ be an equilibrium point for $\dot{z} = f(z)$ where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a Lipschitz function, and $D \subset \mathbb{R}^n$ be a domain containing $z = 0$. Let $V : D \longrightarrow \mathbb{R}$ be a continous differentiable function, such that

$$V(0) = 0 \text{ and } V(z) > 0 \text{ in } D - \{0\}, \tag{7.1}$$

$$\dot{V}(z) \leq 0 \text{ in } D, \tag{7.2}$$

then 0 is stable. Moreover, if

$$\dot{V}(z) < 0 \text{ in } D - \{0\},$$

then 0 is asymptotically stable. Furthermore, if $V(z) > 0$, $\forall z \neq 0$,

$$\|z\| \to \infty \Rightarrow V(z) \to \infty$$

and $\dot{V}(z) < 0$, $\forall z \neq 0$, then the origin is globally asymptotically stable.

Lyapunov stability can also be applied to dynamical systems with equality and inequality constraints, as the following theorem demonstrates.

Theorem 7.3. ([216]) Consider the nonlinear system

$$\dot{z} = f(z, u), \tag{7.3}$$

where $z \in \mathbb{R}^{n_z}$ is the system state and $u \in \mathbb{R}^{n_u}$ denotes a collection of auxiliary variables (parameters, inputs etc). Let the system have the following constraints

$$\begin{aligned} a_{i_1}(z, u) &\geq 0, \ \forall i_1 = 1, \ldots, N_1, \\ b_{i_2}(z, u) &= 0, \ \forall i_2 = 1, \ldots, N_2, \\ \int_0^T c_{i_3}(z, u) &\geq 0, \ \forall i_3 = 1, \ldots, N_3, \text{ and } \forall T \geq 0, \end{aligned}$$

where a_{i_1}, b_{i_2}, c_{i_3} are polynomial functions in (z, u) and $f(z, u)$ is a polynomial vector field. The region $D \subset \mathbb{R}^{n_z+n_u}$ is defined as

$$D = \{(z, u) \in \mathbb{R}^{n_z+n_u} \mid a_{i_1}(z, u) \geq 0, \ b_{i_2}(z, u) = 0, \ \forall i_1, i_2\}.$$

Suppose that for the system there exist polynomial functions $V(z)$, $p_{i_1}(z, u)$, $q_{i_2}(z, u)$ and constants r_{i_3} such that:

- $V(z)$ is positive definite in a neighborhood of the origin.
- $p_{i_1}(z, u) \geq 0$ in D.

Then

$$-\frac{\partial V}{\partial z} f(z, u) - \sum_{i_1=1}^{N_1} p_{i_1}(z, u) a_{i_1}(z, u) - \sum_{i_2=1}^{N_2} q_{i_2}(z, u) b_{i_2}(z, u) - \sum_{i_3=1}^{N_3} r_{i_3} c_{i_3}(z, u) \geq 0$$

will guarantee that the origin of the state space is a stable equilibrium of the system.

7.2.3 Region of Attraction

Definition 7.4. For $\dot{z} = f(z)$ the region of attraction $\mathcal{R}$ of the system is defined as

$$\mathcal{R} := \{z(t) \in \mathbb{R}^{n_z} \ : \ \lim_{t \to \infty} (z(t)) = 0\}.$$

One can obtain an inner approximation of $\mathcal{R}$, which we will denote by $\hat{\mathcal{R}}$, using the level sets of the Lyapunov function $V(z)$ defined by Theorem 7.2. This is achieved by searching for a $\gamma > 0$ such that $\hat{\mathcal{R}} = \{z \in \mathbb{R}^{n_z} \mid V(z) \leq \gamma\} \subset D$.

Theorem 7.5. ([217]) For $\dot{z} = f(z)$ suppose that a Lyapunov function $V(z)$ satisfying the conditions of asymptotic stability was constructed, as per Theorem 7.2 inside a domain D. Suppose $\{z \mid d(z) \geq 0\} \subset D$. If

$$|z|^{2k}(V(z) - \gamma) + p(z)d(z) \geq 0,$$

where $p(z) \geq 0$ and k is an integer, then the level set $V(z) \leq \gamma$ is contained inside the region of attraction.

In the case of a constrained dynamical system, the Lyapunov function constructed via Theorem 7.3 can also be used to provide estimates of the region of attraction. For more details, see [217].

7.2.4 Lyapunov Functions and Sum of Squares

Since the Lyapunov stability constraints (7.1) and (7.2) are a combination of positivity and nonnegativity constraints, they can be replaced with SOS constraints if they are polynomials. Therefore the Lyapunov stability constraints can be turned into an SDP.

Theorem 7.6. ([218]) Consider a dynamical system $\dot{z} = f(z)$ with $z(t) \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}^n$ that is Lipschitz continuous and has the equilibrium of interest at $f(0) = 0$. Suppose there exists a polynomial $V(z)$ such that $V(0) = 0$ and

$$V(z) - \rho(z) \in \Sigma[z],$$
$$-\frac{\partial V}{\partial z} f(z) \in \Sigma[z],$$

where

$$\rho(z) = \sum_{i=1}^{n} \sum_{j=1}^{d} \epsilon_{ij} z_i^{2j},$$
$$\sum_{j=1}^{m} \epsilon_{ij} > \gamma, \ \forall i = 1, \ldots, n,$$

$\gamma > 0$, $\epsilon_{ij} \geq 0$, $\forall i, j$ and $2d$ is the degree of $V(z)$. Then the zero equilibrium is globally stable.

The SOS conditions in Theorem 7.6 can be extended to constrained dynamical systems as in Theorem 7.3.

Theorem 7.7. ([218]) Consider a dynamical system $\dot{z} = f(z,u)$ in Theorem 7.3. Suppose there exists polynomial functions $V(z)$, $p_{i_1}(z,u)$, $q_{i_2}(z,u)$, a positive definition function $\rho(z)$ defined in Theorem 7.6 and constants $r_{i_3} \geq 0$ such that

$$\begin{gathered} V(z) - \rho(z) \in \Sigma[z], \\ -\frac{\partial V}{\partial z} f(z,u) - \sum_{i_1=1}^{N_1} p_{i_1}(z,u) a_{i_1}(z,u) - \sum_{i_2=1}^{N_2} q_{i_2}(z,u) b_{i_2}(z,u) - \sum_{i_3=1}^{N_3} r_{i_3} c_{i_3}(z,u) \in \Sigma[X], \\ p_{i_1}(z,u) \in \Sigma[X],\ \forall j = 1,\ldots,N_1, \\ q_{i_2}(z,u) \in \mathbb{R}[X],\ \forall j = 1,\ldots,N_2, \end{gathered}$$

where $X = (z,u)$ is a vector of all of the states. Then the origin of the system is a stable equilibrium.

7.3 Neural Feedback Loops in Continuous Time

Consider a continuous-time system

$$\dot{z}(t) = f(z(t), u(t)), \tag{7.4}$$

where f is the plant model (assumed to be a Lipschitz function), $z(t) \in \mathbb{R}^{n_z}$ and $u(t) \in \mathbb{R}^{n_u}$ are the system states and inputs respectively and n_z and n_u are the number of system states and inputs respectively. This system is a continuous time NFL if the controller u is described as an NN. We consider a state feedback controller $u(t) = \pi(z(t)) : \mathbb{R}^{n_z} \to \mathbb{R}^{n_u}$ as a feed-forward fully connected NN such that

$$\begin{aligned} x^0(t) &= z(t), \\ v^k(t) &= W^k x^k(t) + b^k, \text{ for } k = 0,\ldots,\ell-1, \\ x^{k+1}(t) &= \phi(v^k(t)), \text{ for } k = 0,\ldots,\ell-1, \\ \pi(z(t)) &= W^\ell x^\ell(t) + b^\ell, \end{aligned} \tag{7.5}$$

where $W^k \in \mathbb{R}^{n_{k+1} \times n_k}$, $b^k \in \mathbb{R}^{n_{k+1}}$ are the weights matrix and biases of the $(k+1)^{\text{th}}$ layer respectively and $z(t) = x^0(t) \in \mathbb{R}^{n_z}$ is the input into the NN. The activation function ϕ is applied element-wise to the $v^k(t)$ terms. The number of neurons in the kth layer is denoted by n_k.

We denote the equilibrium of the states in the closed-loop system as z^*, u^*, x^*, v^*. For simplicity we will drop the time dependency notation throughout this chapter.

7.4 Stability of Neural Feedback Loops

In this chapter we aim to show that the equilibrium of the closed loop NFL (7.4) is stable and then also use this to compute an inner approximation of the region of attraction.

An NFL can be viewed as a dynamical system with equality and inequality constraints that arise from the input-output description of the NN as in Section 2.4.3. We denote this as the semi-algebraic set

$$S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0,\ \forall\, i = 1, \ldots, p,\ j = 1, \ldots, q\}, \tag{7.6}$$

where g_i and h_j are polynomial functions. In this case, Theorem 7.3 can be used to assess the stability of the NFL.

We note here that the u's in Theorem 7.3 can be the x's describing the NN, extra variables needed to reformulate the system, or parametric uncertainty. Once we have constructed a Lyapunov function showing asymptotic stability of the equilibrium, we can attempt to obtain an estimate of the region of attraction of that equilibrium.

The equality and inequality constraints in the SOS program include the set of constraints from the NN abstraction and form a semi-algebraic set. Further constraints may be needed when only local asymptotic stability is being verified. In this case, we describe a region

$$D^z = \{z \in \mathbb{R}^{n_z} \mid d_k(z) \geq 0,\ k = 1, \ldots, n_d\}, \tag{7.7}$$

where the stability conditions will need to be satisfied. These constraints can be incorporated into the optimisation framework in the same way as in Theorem 7.3.

Proposition 7.8. Consider System (7.4) in feedback with an NN controller given by (7.5). Suppose the input-output properties of the NN are described by (7.6), and consider the region given by (7.7). Suppose there exists a polynomial function $V(z)$ satisfying the

following conditions

$$\begin{gathered} V(z) - \rho(z) \in \Sigma[z], \\ \rho(z) > 0, \\ -\frac{\partial V}{\partial z}(z) f(z, \pi(z)) - \sum_{k=1}^{n_d} p_k(X) d_k(z) - \sum_{j=1}^{q} t_j(X) h_j(x) - \sum_{i=1}^{p} s_i(X) g_i(x) \in \Sigma[X], \\ p_k(X) \in \Sigma[X],\ \forall k = 1, \ldots, n_d, \\ s_i(X) \in \Sigma[X],\ \forall i = 1, \ldots, p, \\ t_j(X) \in \mathbb{R}[X],\ \forall j = 1, \ldots, q, \end{gathered} \tag{7.8}$$

where X is a vector of all the system and NN states, i.e. $X = (x, z)$. Then the equilibrium of the NFL is asymptotically stable in the region D^z.

Proof. This proposition follows from Theorem 7.7 by considering the NFL as a constrained dynamical system. The $a_{i_1}(z, u)$ expressions consist of the $d_k(z)$ and $g_i(x)$ inequality constraints, the $b_{i_2}(z, u)$ expressions consist of the $h_j(x)$ equality constraints and the $c_{i_3}(z, u)$ expressions are all zero. ■

Optimisation problem (7.8) has the benefit of analysing the system in the continuous time domain, whereas methods such as [180] only analyse the discrete time system. By using the continuous time domain we are able to ensure stability without discretising the system, which may introduce conservativeness as it acts as an approximation to the true continuous-time system.

Suppose now that a Lyapunov function is constructed using Proposition 7.8. To determine the region of attraction as stated in Theorem 7.5 we must determine the largest level set of the Lyapunov function $V(z)$ that is contained within the region that the Lyapunov conditions are satisfied, which can also be cast as an SOS program. Suppose $d(z) \geq 0$ is included inside this region. Then

$$\begin{gathered} |z|^k (V(z) - \gamma) + p(z) d(z) \in \Sigma[z], \\ p(z) \in \Sigma[z], \end{gathered} \tag{7.9}$$

where γ is a variable to be maximised and k is a positive integer, ensures that $V(z) \leq \gamma$ is an estimate of the region of attraction. This process can then be repeated by adjusting the $d_k(z)$ constraints to expand the region tested for stability - we provide an algorithm in the next section.

7.4.1 Computing the Stability and Region of Attraction

To compute the stability of System (7.4) we must first ensure that the vector field is polynomial. If the system is not already expressed as a polynomial, this can be achieved by making an appropriate substitution [216]. We assume that we are given an NN controller for the system and want to establish the stability properties of the closed loop system and obtain an estimate of the region of attraction. The SOS constraints that are constructed can be solved with an appropriate parser. The procedure for determining stability is presented in Algorithm 1.

Algorithm 1 Determining the Stability of an NFL

Input: NFL (7.4), NN (7.5), region (7.7)
Initialise $V(z), p_k(X), t_j(X), s_i(X), \rho(z)$, as in (7.8)
feas = 0
while feas = 0 **do**
  Compute IBP values using region (7.7)
  Obtain $g_i(x)$ and $h_j(x)$ from (7.5) and IBP values
  Solve (7.8)
  if (7.8) is feasible **then**
    feas $\leftarrow$ 1
  else
    $d_k \leftarrow \hat{d}_k$ ▷ $\hat{d}_k$ defines a new region $\hat{D}$
  end if
end while
Output: $V(z)$

We now describe these steps in detail. Before the NN constraints can be created, we must first define the region D^z of the state space that we are interested in by setting the constraints d_k. Once these are established, we can then compute the maximum and minimum possible values of each system state $(\underline{z}, \overline{z})$. These values can be used as inputs into the IBP algorithm to find the preprocessing bounds. Once these bounds have been computed the NN constraints can be obtained to form the semi-algebraic set (7.6). These constraints will depend on the problem set-up. Having established the constraints that define our system, we search for a Lyapunov function using (7.8). If this is infeasible, we can reduce the size of the region D^z until a Lyapunov function can be found. Once a Lypaunov function is found the region of attraction can be obtained using (7.9).

7.4.2 Extension to Robustness Analysis

So far, we have assumed that the parameters within our system are fixed. When there is parametric uncertainty, we wish to ensure robust stability. Theorem 7.3 allows for parameters to be included in the description of the system and therefore it is possible to construct parameterised Lyapunov functions. The robustness constraints can be added to the Lyapunov stability condition in optimisation problem (7.8). This condition can then be written as

$$-\frac{\partial V}{\partial z}(z)f(z,\pi(z)) - \sum_{m=1}^{n_\Delta} r_m \Delta_m(z)\ldots \\ - \sum_{k=1}^{n_d} p_k(X)d_k(z) - \sum_{j=1}^{q} t_j(X)h_j(x) - \sum_{i=1}^{p} s_i(X)g_i(x) \in \Sigma[X],$$

where $r_m \in \Sigma[X]$ and $\Delta_m(z) \geq 0$, $m = 1, \ldots, n_\Delta$ are robustness inequality constraints that can be imposed on the systems states.

One potential drawback with adding too many robustness constraints is that it may make the SOS program infeasible, meaning that a smaller region D^z may have to be considered. Adding robustness constraints will also increase the size of the SDP and may lead to intractable solve times.

We demonstrate how robustness constraints can be incorporated into the SOS program in an example in the following section.

7.5 Numerical Examples

We demonstrate our approach on various systems to show improvements over existing methods. All experiments were run on a four-core Intel Xeon processor @3.50GHz with 16GB of RAM. We refer to our method as 'NNSOSStability', which solves optimisation problem (7.8). We implement our method using MATLAB and SOSTOOLS to parse the SOS constraints into an SDP, which is solved using SeDuMi [61], as it produced the least numerical errors. NNs that are commonly used as controllers in NFLs are of a smaller size, hence we consider smaller NNs in the following examples. Additionally, the SOS optimisation problem (7.8) becomes intractable as the number of systems states and NN nodes becomes too large. Note that we do not exploit the sparse structure

of the NNs in this SOS method. It may be possible to exploit this sparsity in future work to allow for more scalable solve times.

7.5.1 Duffing Oscillator

We show the benefits of this SOS method on a simple non-linear system example. Consider the Duffing Oscillator proposed in [163], with system dynamics

$$\begin{aligned} \dot{z}_1 &= z_2, \\ \dot{z}_2 &= -z_1 - 2\zeta z_2 - z_1^3 + u, \end{aligned} \tag{7.10}$$

where $\zeta = 0.5$ is the damping ratio. We fit a small NN controller with two layers and two nodes in each layer with ReLU activation functions. The NN weights and biases were determined by randomising them until a suitable controller was found. We are able to verify global asymptotic stability of this system using a fourth-order Lyapunov function and second-order multipliers. The solve time for the SDP in this example was 10.9s. The benefits of this SOS framework is that we are able to account for the non-linearity directly, instead of bounding it by a sector constraint or by making a substitution to lift the variables. For comparison, if we use the linearised model with sector constraints on $-z_1^3$ we are only able to verify stability in a circle of radius 0.2. If we use the discrete Lyapunov stability conditions then the SOS program becomes too large and we cannot verify global stability. The trajectories of this system are shown in Figure 7.1.

7.5.2 Three Dimensional Polynomial System

The previous example shows that the stability of NFL systems can be verified effectively using SOS and demonstrates the benefits it possesses over other methods. However, the uncontrolled system is globally stable. We now consider a non-linear polynomial plant with three states that is unstable without a controller. The system is given by

$$\begin{aligned} \dot{z}_1 &= -z_1 + z_2 - z_3, \\ \dot{z}_2 &= -z_1(z_3 + 1) - z_2, \\ \dot{z}_3 &= -z_1 + u. \end{aligned}$$

We fit a five layer NN with five nodes in each layer with tanh activation functions to stabilise the system. This was achieved by finding a linear controller to stabilise the

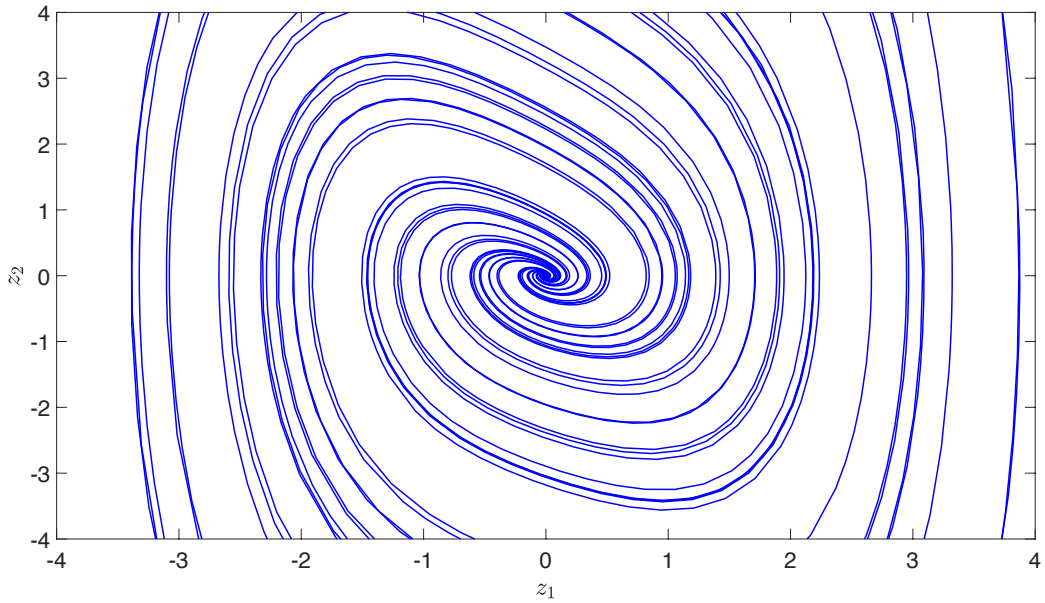


Figure 7.1: The trajectories of the Duffing Oscillator described by (7.10).

system locally and then fitting an NN to the input-output data for this closed-loop system. We specify the region D to be a cube of length six around the origin such that

$$3 - z_1 \geq 0,\ 3 - z_2 \geq 0,\ 3 - z_3 \geq 0,$$

$$3 + z_1 \geq 0,\ 3 + z_2 \geq 0,\ 3 + z_3 \geq 0,$$

to compute the IBP bounds. We propose a quadratic Lyapunov function, which we use to verify stability, taking 89s to solve. The region of attraction shown in Figure 7.2 demonstrates a good approximation of the true region of attraction, which is computed using an exhaustive search. The authors were unable to verify the stability of this system using [177], [179], [180].

7.5.3 Inverted Pendulum

We consider the inverted pendulum example proposed in [177], trained with the NN controller in [179]. The system dynamics are

$$\ddot{\theta}(t) = \frac{mgl \sin\left(\theta(t)\right) - \mu\dot{\theta}(t) + \text{sat}(u(t))}{ml^2},$$

where $m = 0.15$ kg, $l = 0.5$ m, $\mu = 0.5$ Nmsrad^{-1}, $g = 9.81$ ms^{-2}, $u_{\max} = 1$ Nm and $\text{sat}(u(t))$ is the saturation function on the control input. To incorporate this system in

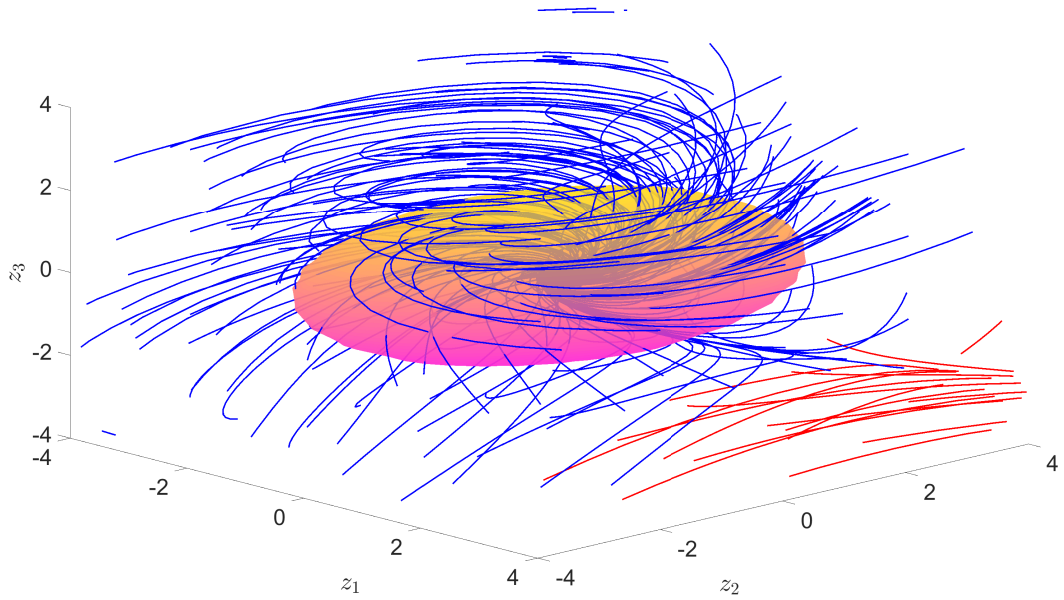


Figure 7.2: The region of attraction prodcued by NNSOSStability in Example 7.5.2 with a second-order Lyapunov function shown as a surface. The blue trajectories tend towards the zero equilibrium, whereas the red ones do not.

the SOS programming framework, we let $z_1 = \theta$, $z_2 = \dot{\theta}$ and $z_3 = z_1 - \sin(z_1)$. This replaces the non-linear system with a linear system, however we must bound the z_3 term, which we can do with a sector constraint depending on how our region D is defined. This system is an approximation of the original non-linear system and can be written as

$$\begin{aligned}
\dot{z_1} &= z_2, \\
\dot{z_2} &= \frac{g}{l}(z_1 - z_3) - \frac{\mu}{ml^2}z_2 + \frac{1}{ml^2}u, \\
u + u_{\max} &\geq 0, \\
u_{\max} - u &\geq 0, \\
z_3(\alpha_{z_3} z_1 - z_3) &\geq 0,
\end{aligned}$$

where

$$\alpha_{z_3} = \max\left(\frac{\overline{z}_1 - \sin(\overline{z}_1)}{\overline{z}_1}, \frac{\underline{z}_1 - \sin(\underline{z}_1)}{\underline{z}_1}\right),$$

is determined by the maximum and minimum bounds of the system states $(\overline{z}_1, \underline{z}_1)$ obtained from the region D. The input saturation constraints and the sector constraints on the non-linearity are added to the derivative condition in the optimisation problem. The NN has five layers and five nodes in each layer with tanh activation functions.

We search for a quartic Lyapunov function $V(z_1, z_2)$ and compare its performance to other methods. We set the system constraints to be a box such that

$$z_1 - \underline{z}_1 \geq 0,\ z_2 - \underline{z}_2 \geq 0,$$

$$\overline{z}_1 - z_1 \geq 0,\ \overline{z}_2 - z_2 \geq 0.$$

We use sector and slope constraints on the tanh activation functions in the NN. Choosing values of $\underline{z}_1 = -0.3$, $\overline{z}_1 = 0.3$, $\underline{z}_2 = -1.4$ and $\overline{z}_2 = 1.4$ we are able to obtain a fourth-order Lyapunov function in this region, which took 135s to compute. We then compute the region of attraction for this Lyapunov function and find that its volume is increased significantly over other methods, as shown in Figure 7.3. We compare this method to the linearised system with different kinds of Zames-Falb multipliers from [179]. We tested the formulations in [177] and [180], however both were infeasible in the regions that we trialled.

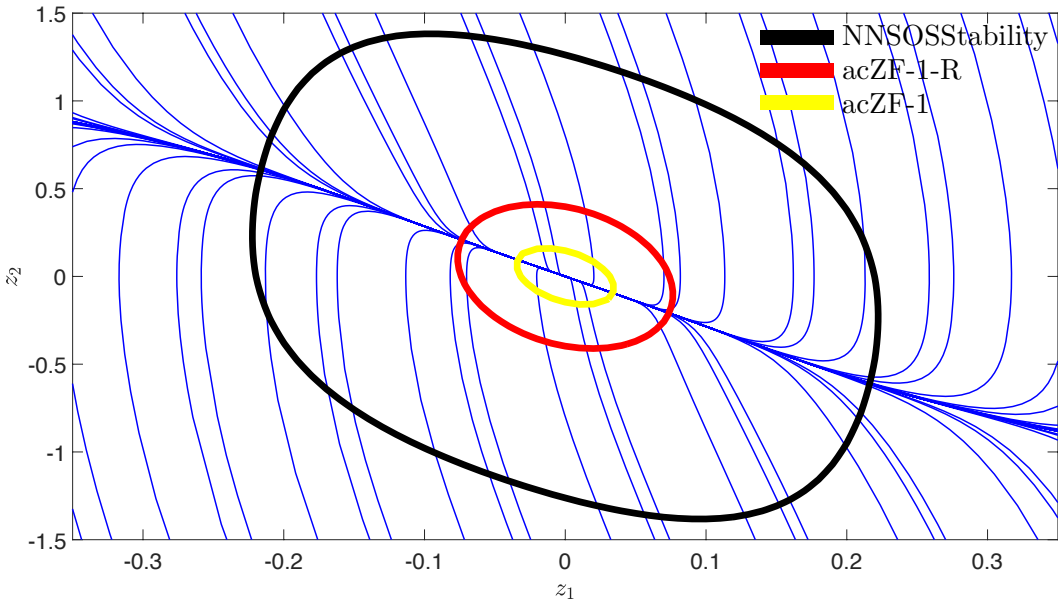


Figure 7.3: Diagram comparing the regions of attraction for different methods in Example 7.5.3. NNSOSStability with a fourth-order Lyapunov function is shown in black, the methods in [179] which uses diagonal and full-block acausal Zames-Falb multipliers are shown in yellow and red respectively. The trajectories of the system are shown in blue.

7.5.4 Robustness Analysis of Inverted Pendulum

We use the above example and conduct robustness analysis on the system to determine how the stability is affected by perturbations on the length of the pendulum arm. We make the

substitution $\delta = 1/l$ and define a robustness certificate on δ. We can rewrite the system as

$$\begin{aligned}
&\dot{z_1} = z_2, \\
&\dot{z_2} = g\delta(z_1 - z_3) - \frac{\mu\delta^2}{m} z_2 + \frac{\delta^2}{m} u, \\
&u + u_{\max} \geq 0, \\
&u_{\max} - u \geq 0, \\
&z_3(\alpha_{z_3} z_1 - z_3) \geq 0, \\
&\Delta(\delta) \geq 0,
\end{aligned}$$

where $\Delta(\delta)$ is a function describing the robustness of the variable δ. We perturb the length by ± 0.3, which generates the robustness constraint

$$(5 - \delta)(\delta - 1.25) \geq 0.$$

We propose a quadratic Lyapunov function with a region defined with values of $\underline{z}_1 = -0.1$, $\overline{z}_1 = 0.1$, $\underline{z}_2 = -0.3$ and $\overline{z}_2 = 0.3$. We are able to find a feasible solution to the SOS optimisation problem, which shows that the system is stable in that region under the perturbation $\Delta(\delta)$. This optimisation problem took 186s to compute. To the best of the author's knowledge, there are no other methods that are able to conduct the robustness analysis on the parameter uncertainty for non-linear NFLs.

7.6 Conclusion

In this chapter, we have proposed an SOS programming framework to determine the stability of NFLs in the continuous time domain. We use a well-established approach to abstract the NN as a set of equality and inequality constraints to construct a semi-algebraic set. We then use Lyapunov stability theory for constrained systems and SOS programming to construct the relevant Lyapunov certificates. Once a Lyapunov function is constructed within a specified region, a region of attraction of the equilibrium can be computed by solving another SOS program. This framework allows us to conduct robustness analysis if parametric or other uncertainties are present. We test our method on various numerical examples and compare our results to existing methods. The region of attractions that are computed using our method are increased compared to competitive approaches.

Additionally, we showed that we can effectively certify stability of a non-linear NFL that contains parametric uncertainty, which would not be possible with previous methods.

8

Rational Neural Network Controllers

This chapter considers rational neural networks and presents novel rational activation functions, which can be used effectively in robustness problems for neural feedback loops. Rational activation functions are replaced by a general rational neural network structure, which is convex in the neural network's parameters. A method is proposed to recover a stabilising controller from a Sum of Squares feasibility test. This approach is then applied to a refined rational neural network which is more compatible with Sum of Squares programming. Numerical examples show that this method can successfully recover stabilising rational neural network controllers for neural feedback loops with non-linear plants with noise and parametric uncertainty.

8.1 Introduction

The work in this thesis has addressed the analysis of NFLs. However, the more challenging task is to obtain the NN controller's parameters, whilst enforcing robustness guarantees. As outlined in Section 1.1, NNs are useful since they can be used as general function approximators, but contain a large number of parameters. This means that optimising over all of the parameters is often computationally expensive.

One method to design the NN controller is by learning an expert control law through input-output data and then checking the robustness guarantees using analysis methods such as in Chapters 6 and 7. However, this may lead to poor robustness certificates because the NN has no objective to optimise the robustness metrics when it is trained. It is also possible to use reinforcement learning methods. This often involves training the controller by simulating the system trajectories and then updating the controller's

parameters subject to maximising a reward function. However, there are significant challenges with this process; it is very computationally intensive, requiring a large amount of hyperparameter tuning and sometimes leading to undesirable behaviour [141]. The parameterised NN controller can then be analysed to obtain robustness certificates in a defined operating region, however these guarantees can be poor. It can be difficult for these approaches to outperform traditional control laws. Furthermore, adding additional non-linearities into the model through the NN's activation functions can increase the complexity of the closed-loop system.

Despite these drawbacks, recent methods have focused on addressing these issues by designing NN controllers whilst ensuring robustness guarantees in the process. Methods that focus on developing reinforcement learning algorithms such as [184, 185] are able to create an NN policy which can be combined with robust control guarantees. However, these approaches still suffer from other reinforcement learning issues such as requiring significant computational time and hyperparameter tuning. These drawbacks can be mitigated by instead trying to obtain an NN controller by learning from an optimal model predictive control law. This allows the NN controller to be trained offline and when implemented it can be significantly less computationally expensive than the full model predictive control law. To achieve this an SDP framework that incorporates integral quadratic constraints is presented in [186]. An iterative algorithm is used to alternate between optimising the NN parameters to fit the control law and maximising the region of attraction. However, these approaches require a known model predictive control law to optimise over the controller's input-output data, which may not be easy to obtain. This approach relies on a loop transformation and Schur complement to ensure the optimisation problem is convex. Similar approaches have been used for different NN architectures such as recurrent NNs [187] and recurrent equilibrium network controllers [188]. These methods also include a projected policy gradient algorithm and reinforcement learning to synthesise the controller, instead of requiring an expert control law.

One shortfall of many recent approaches such as [186–188] is that large convex approximations must be made, as the original problem is non-convex. For example, the ReLU and tanh activation functions must be sector bounded in the region $[0, 1]$ with no

constraints obtained from preprocessing bounds. This can lead to the acquired robustness certificates being conservative. Another issue with these methods is that they rely on iterative approaches, where the algorithm alternates between optimising the performance of the controller and the stability guarantees. This can make the problem computationally expensive and intractable for larger scale systems. They also require that the plant model is linear subject to sector bounded non-linearities.

One root cause of the problems with recent approaches is that they use traditional NN architectures with ReLU, sigmoid and tanh activation functions. These structures have shown to be very effective in many machine learning tasks, however they may not be preferable for use in control systems. By considering a class of NNs that are better aligned with control system techniques, these large convex approximates could be mitigated. Since SOS programming is effective for systems with polynomial and rational functions, we investigate what happens when the NN is built upon them.

8.2 Rational Neural Networks

There has been prior work investigating NNs with rational expressions, although this research has been limited. [16] showed the expressive power of using rational activation functions and how they can be used to approximate commonly used activation functions such as ReLU. The effectiveness of rational NNs in machine learning tasks was shown in [219]. Polynomial activation functions have also been used in previous work [220–224], however they have not seen much investigation within the machine learning research community due to vanishing and exploding gradients that can be exhibited when used with the backpropagation algorithm [197]. Another class of NNs that has seen recent interest are quadratic NNs. These can take many different forms, which are summarised in [225]. Quadratic NNs have been shown to be beneficial in many aspects [18, 53, 226], such as being general universal function approximators.

One of the benefits of using polynomial or rational activation functions in the NN is that they can contain trainable parameters, which can improve the performance of the NN and reduce the number of neurons in the network. For example, the rational

activation function used in [219] is expressed as

$$\phi(x) = \frac{P(x)}{Q(x)} = \frac{\sum_{i=0}^{r_P} a_i x^i}{\sum_{j=0}^{r_Q} b_j x^j}, \tag{8.1}$$

where $r_P = \deg(P(x))$, $r_Q = \deg(Q(x))$ and a_i and b_j are the trainable parameters within the activation function. Another proposed activation function based on rational functions is the Padé activation unit, which has shown to be useful when applied to image classification [227]. This activation function is expressed as

$$\phi(x) = \frac{P(x)}{Q(x)} = \frac{\sum_{i=0}^{r_P} a_i x^i}{1 + |\sum_{j=0}^{r_Q} b_j x^j|}$$

and can be used to learn and approximate commonly used activation functions, whilst training the NN. The resulting NNs can have compact representations and can perform similarly to state-of-the-art NNs.

8.2.1 Rational Approximation of Tanh Activation Function

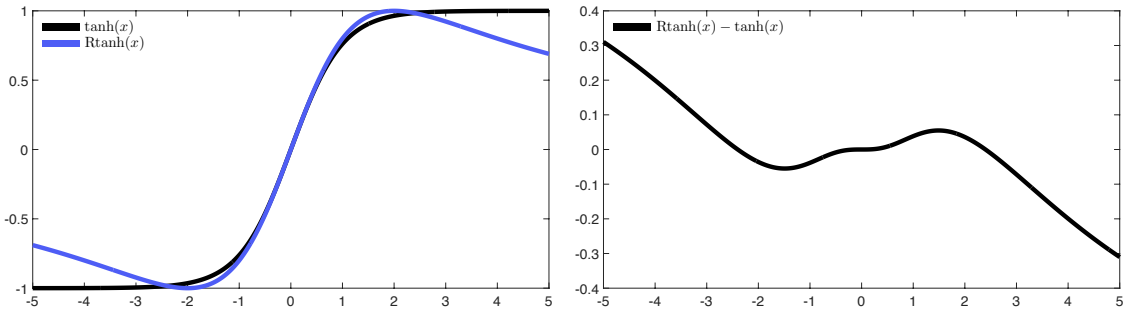
Despite their success in some machine learning applications, rational activation functions have yet to be applied to NN controllers. Most methods to train NNs in control systems use traditional activation functions such as ReLU, sigmoid and tanh. In addition, NNs that are used in control systems are often significantly smaller than those used for machine learning tasks such as image classification. The reason behind this is that methods to train NNs for control often require the use of an SDP, meaning that having a large number of parameters in the NN makes the problem become intractable. However, the requirement for the NN to only contain a small number of neurons to learn a function that can sufficiently control a system has not been well explored. One justification for this is the low number of input and output dimensions that are required for small NFLs. However, if the number of system states increases, then larger networks may be required, which may be intractable to obtain with current methods.

For NNs to effectively be used as controllers, it would be of interest to investigate the use of alternative NN structures and activation functions that may give a sufficient level of expressivity in the network, whilst being easier to compute the robustness guarantees. Motivated by this we propose a novel activation function defined by a

simple rational expression. We name this function 'Rtanh' as it is an approximation of the tanh activation function and defined as

$$\text{Rtanh}(x) = \phi(x) = \frac{4x}{x^2 + 4}.$$

This function is shown in Figure 8.1a against the tanh function and the error between these functions is shown in Figure 8.1b.



(a) Comparison between Rtanh(x) and tanh(x) **(b)** Error between Rtanh(x) and tanh(x)

Figure 8.1: Plots showing a comparison between the Rtanh(x) and tanh(x) functions.

As shown in Section 2.4.3 and demonstrated throughout this thesis, the tanh function can be over-approximated by sector constraints, which are conservative. Whereas the 'Rtanh' function can be represented by a singular equality constraint such that

$$\phi(x)(x^2 + 4) - 4x = 0. \tag{8.2}$$

If this activation function were to be used in an NN, then to test its robustness properties using the Psatz, the equality constraint (8.2) can be used directly. This is beneficial as no conservatism is introduced when abstracting the input-output properties of the NN as a semi-algebraic set.

To demonstrate that the Rtanh activation function is useful when analysing NFLs, we take an existing NFL that uses tanh activation functions. We consider the system (6.11) in Section 6.7.3, which uses a five layer NN with five nodes in each layer and tanh activation functions. We then replace the tanh function with Rtanh and observe the system behaviour. We conduct the same reachability analysis as in Section 6.6 by computing the reachable sets for six time steps. In Figure 8.2 we can see that this activation function behaves similarly to that of the tanh function and that using the Psatz with sparse polynomial

optimisation (ReachSparsePsatz) gives very tight approximations of the reachable sets. We also compute the region of attraction using the same approach as in Section 7.4.1. We find that the region of attraction is increased significantly when using Rtanh over the tanh activation function. The areas of the region of attraction on the phase plane are 58 and 0.94 per square unit for the Rtanh and tanh activation functions respectively.

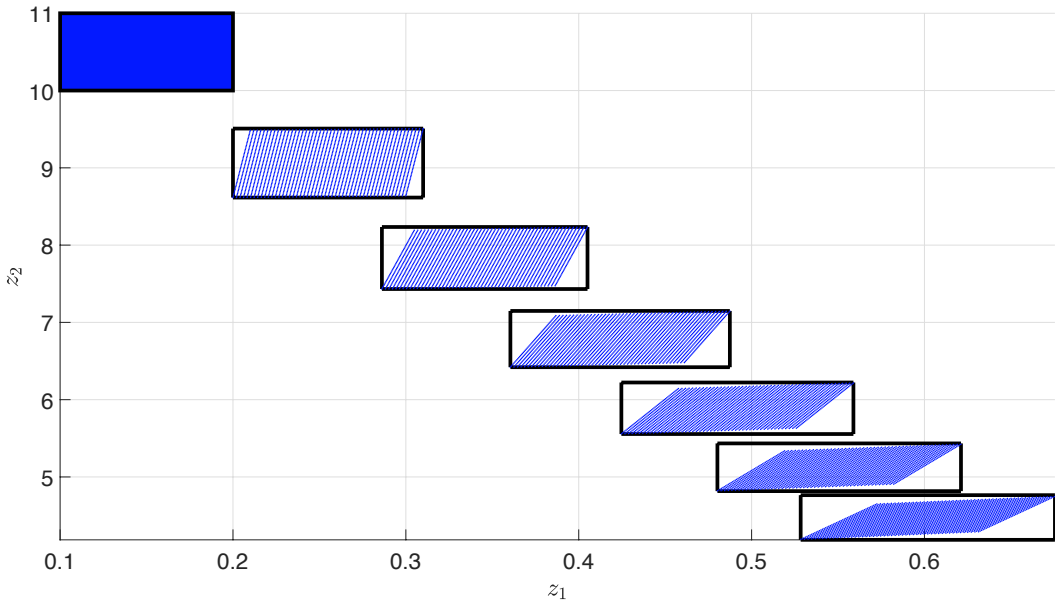


Figure 8.2: Reachable sets for the inverted pendulum in Section 6.7.3 where the tanh activation function is replaced with the Rtanh function. The reachable sets are computed using ReachSparsePsatz and are shown in black. The true reachable sets are shown in blue.

These results are beneficial for the use of the Rtanh function in NNs and NFLs. However, it is not compatible with most recent methods to learn NN controllers due to it being represented by an equality that is a polynomial of degree three. Most methods require sector inequality constraints and the Schur complement, which is used to make the problem convex. Therefore, this function cannot be used in those formulations. This requires the use of an alternative method to obtain an NN controller with these activation functions, which we will investigate later in this chapter.

8.2.2 Rational Approximation of Sigmoid Activation Function

It is also possible to approximate the sigmoid function as a rational function. The function

$$\text{Rsig}(x) = \phi(x) = \frac{(x+4)^2}{2(x^2+16)},$$

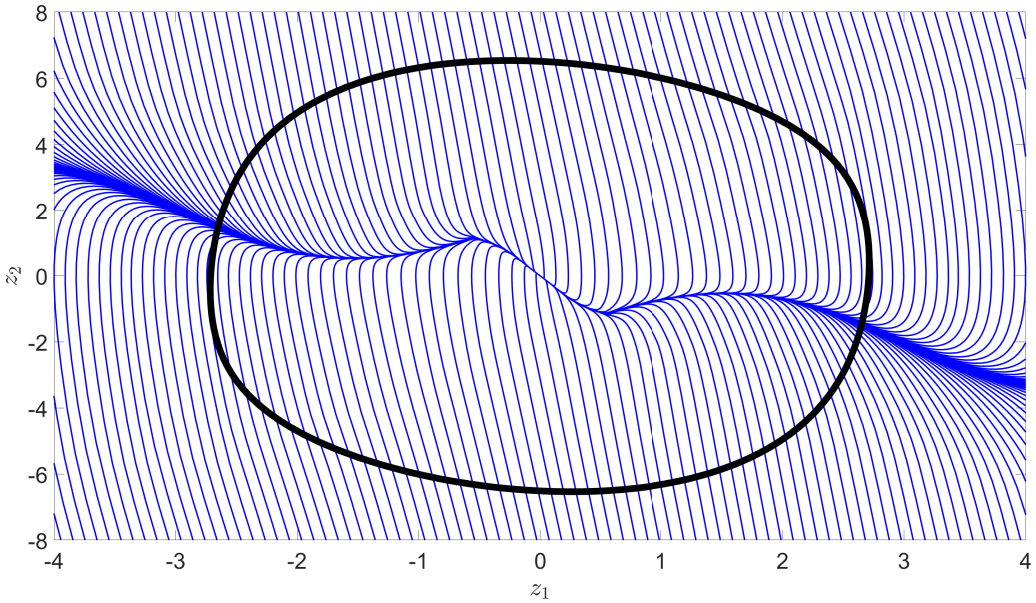
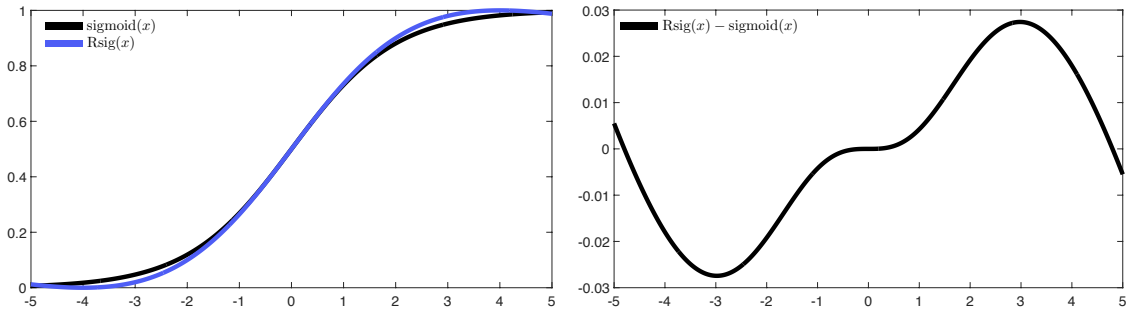


Figure 8.3: Diagram showing the region of attraction for the inverted pendulum in Section 7.5.3 where the tanh activation function is replaced with the Rtanh function. The region of attraction is computed using NNSOSStability (black) with a fourth order Lyapunov function. The system trajectories are shown in blue.

is a good approximation of the sigmoid function. Rsig is shown in Figure 8.4a along with the error with the sigmoid function in Figure 8.4b. Rsig can be expressed as the equality constraint

$$2(x^2 + 16)\phi(x) - (x + 4)^2 = 0.$$



(a) Comparison between Rsig(x) and sig(x)

(b) Error between Rsig(x) and sig(x)

Figure 8.4: Plots showing a comparison between the Rsig(x) and sig(x) functions.

8.2.3 Irrational Approximation of ReLU Activation Function

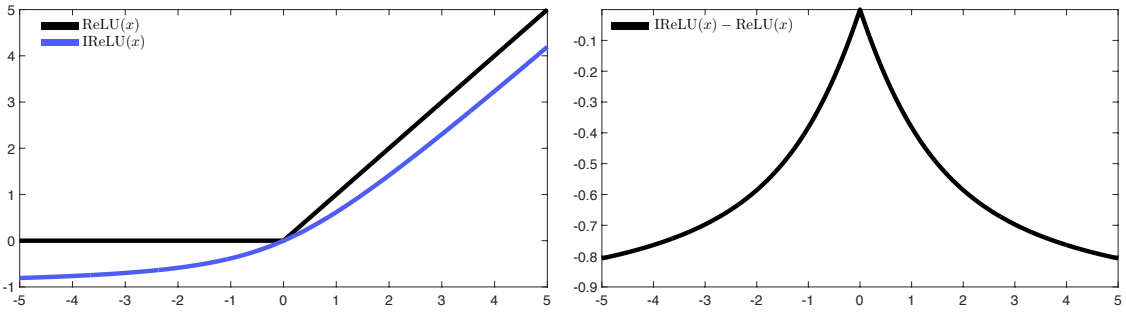
Approximating the ReLU function with a rational function is more difficult. However, the class of irrational activation functions could be considered. This class of functions

will not be considered in this thesis, however to demonstrate how they could be used we propose the following function

$$\text{IReLU}(x) = \phi(x) = \sqrt{x^2 + 1} + x - 1.$$

This function and the error with the ReLU function are shown in Figure 8.5a and Figure 8.5b respectively. By making a substitution the IReLU function can be expressed as a semi-algebraic set with two equality constraints and one inequality constraint

$$\begin{aligned} \phi(x) - y - x + 1 &= 0, \\ y^2 - x^2 - 1 &= 0, \\ y &\geq 0. \end{aligned}$$



(a) Comparison between IReLU(x) and ReLU(x) **(b)** Error between IReLU(x) and ReLU(x)

Figure 8.5: Plots showing a comparison between the IReLU(x) and ReLU(x) functions.

8.2.4 General Rational Neural Networks

The rational approximations of the tanh and sigmoid functions proposed in Section 8.2 can be used as activation functions in an NN. However, if we were to use these structures we could be missing the potential expressivity that can be obtained from a general class of rational functions. As in (8.1) rational activation functions can contain training parameters. Therefore, instead of considering a rigid structure for the activation function and fixing the parameters, we can consider a general rational expression that contains the NN parameters.

If we were to consider a feed-forward fully connected NN as defined in (2.6) with predefined rational activation functions, then the NN can be written as

$$\begin{aligned} x^0 &= u, \\ v^k &= W^k x^k + b^k, \text{ for } k = 0, \dots, \ell - 1, \\ x^{k+1} &= \frac{p(v^k)}{q(v^k)}, \text{ for } k = 0, \dots, \ell - 1, \\ \pi(u) &= W^\ell x^\ell + b^\ell, \end{aligned} \tag{8.3}$$

where $p(v^k)$ and $q(v^k)$ are polynomial functions with specified coefficients. However, if we substitute the preactivation value v_j^k into the polynomial expressions, we obtain a rational expression in x_j^k, where the coefficients are parameterised by the values of the weight matrices W^k and biases vector b^k. Any coefficients in the rational activation function will be multiplied by the weights and bias terms. Therefore, we can instead consider an NN with no affine transformation and parameters only in the rational activation function. We can then write the rational NN as

$$\begin{aligned} x^0 &= u, \\ x_i^{k+1} &= \frac{p_i(x^k)}{q_i(x^k)}, \text{ for } i = 0, \dots, n_k, \text{ for } k = 0, \dots, \ell - 1, \\ \pi_i(u) &= \frac{p_i(x^\ell)}{q_i(x^\ell)}, \text{ for } i = 0, \dots, n_\ell, \end{aligned} \tag{8.4}$$

where $p_i(x^k)$ and $q_i(x^k)$ are general polynomial functions which can be written as

$$\begin{aligned} p_i(x) &= \sum_{\alpha \in \mathbb{N}^n_{d_{p_i}}} \lambda_\alpha x^\alpha, \\ q_i(x) &= \sum_{\beta \in \mathbb{N}^n_{d_{q_i}}} \gamma_\beta x^\beta, \end{aligned}$$

where $d_{q_i} = \deg(q_i)$, $d_{p_i} = \deg(p_i)$, α and β are the exponents that are defined in Section 2.1.2 and λ_α and γ_β are the coefficients of $p_i(x)$ and $q_i(x)$ respectively. To show the similarity between (8.3) and (8.4) we present the following simple example.

Example 8.1. Consider an NN with structure defined by (8.3) with two layers and two nodes in each layer. The rational activation functions with $\deg(p) = \deg(q) = 2$ can be written as

$$\begin{aligned} p(v) &= c_1 v^2 + c_2 v + c_3, \\ q(v) &= d_1 v^2 + d_2 v + d_3. \end{aligned}$$

The first node in the second layer can be expressed as

$$v_1^1 = W_1^1 x_1^1 + W_2^1 x_2^1 + b_1^1,$$
$$x_1^2 = \frac{p(v_1^1)}{q(v_1^1)}.$$

We can substitute the preactivation terms into the activation functions to obtain the polynomials

$$p(v_1^1) = c_1(W_1^1)^2(x_1^1)^2 + 2c_2W_1^1W_2^1(x_1^1x_2^1) + c_3(W_2^1)^2(x_2^1)^2 + (c_1W_1^1 + 2c_2W_1^1b_1^1)(x_1^1) + (c_1W_2^1 + 2c_2W_2^1b_1^1)(x_2^1) + (c_1(b_1^1)^2 + c_2b_1^1 + c_3),$$

$$q(v_1^1) = d_1(W_1^1)^2(x_1^1)^2 + 2d_2W_1^1W_2^1(x_1^1x_2^1) + d_3(W_2^1)^2(x_2^1)^2 + (d_1W_1^1 + 2d_2W_1^1b_1^1)(x_1^1) + (d_1W_2^1 + 2d_2W_2^1b_1^1)(x_2^1) + (d_1(b_1^1)^2 + d_2b_1^1 + d_3),$$

which form the rational expression for x_1^2. However, we can instead write the rational expression as

$$x_1^2 = \frac{\lambda_1(x_1^1)^2 + \lambda_2x_1^1x_2^1 + \lambda_3(x_2^1)^2 + \lambda_4x_1^1 + \lambda_5x_2^1 + \lambda_6}{\gamma_1(x_1^1)^2 + \gamma_2x_1^1x_2^1 + \gamma_3(x_2^1)^2 + \gamma_4x_1^1 + \gamma_5x_2^1 + \gamma_6},$$

which is in the form of (8.4). This can be generalised to larger NNs with higher degree polynomials in the rational functions. This approach can reduce the number of parameters in the NN and each polynomial is convex in the decision variables. This allows the parameters in the rational expression to be tuned directly instead of simultaneously tuning the weights, biases and rational activation parameters.

8.3 Recovering Stabilising Controllers using Sum of Squares

In this section, we propose a novel procedure to obtain a stabilising controller for a non-linear polynomial system using SOS programming. To do this we leverage the Psatz and exploit its structure to satisfy a feasibility test that is equivalent to satisfying a feasibility test for a stabilising controller.

Proposition 8.2. Consider the polynomial $f(x) \in \mathbb{R}[x]$ and the set

$$S = \{y \in \mathbb{R}^n,\ z \in \mathbb{R} \mid p_i(y,z) - q_i(y)z = 0\ \forall\, i = 1, \ldots, m\},$$

where $p_i(y,z) \in \mathbb{R}[x]$, $q_i(z) \in \mathbb{R}[z]$ and $x = [y,z]$. If

$$\begin{aligned} f(x) - \sum_{i=1}^{m}(p_i(y,z) - q_i(y)z) \in \Sigma[x], \\ q_i(y) \neq 0,\ \forall\, i = 1,\ldots,m, \end{aligned} \tag{8.5}$$

then $f(x) \geq 0$ on the set

$$T = \left\{ y \in \mathbb{R}^n,\ z \in \mathbb{R} \;\middle|\; \frac{p_i(y,z)}{q_i(y)} - z = 0,\ q_i(y) \neq 0\ \forall\, i = 1,\ldots,m \right\}.$$

Proof. Consider the set S, if

$$f(x) - \sum_{i=1}^{m} t_i(p_i(y,z) - q_i(y)z) \in \Sigma[x],$$

where $t_i = 1$, since $t_i \in \mathbb{R}[x]$, then $f(x) \geq 0$ on S. If we include the condition $q_i(y) \neq 0$, $\forall\, i = 1,\ldots,m$, then we can rewrite (8.5) as

$$\begin{aligned} f(x) - \sum_{i=1}^{m} q_i(y)\left(\frac{p_i(y,z)}{q_i(y)} - z\right) \in \Sigma[x], \\ q_i(y) \neq 0,\ \forall\, i = 1,\ldots,m, \end{aligned}$$

since $q_i(y) \in \mathbb{R}[x]$, then $f(x) \geq 0$ on T. ■

Proposition 8.2 considers the Psatz in a particular form to obtain a positivity certificate of a function over a set of rational functions. In essence, this formulation fixes the multipliers in the Psatz to unity and then searches over the polynomials to find a constraint set that satisfies the feasibility test. This is useful if we want to find a constraint set for the nonnegativity of a function, instead of testing that function over a known constraint set. This can be leveraged to find a controller that satisfies the Lyapunov stability conditions.

Proposition 8.3. Consider the non-linear system

$$\begin{aligned} \dot{z} &= f(z) + g(z)u, \\ u &= \frac{p(z)}{q(z)},\ q(z) \neq 0, \end{aligned} \tag{8.6}$$

where $z \in \mathbb{R}^{n_z}$ are the system states, $u \in \mathbb{R}^{n_u}$ is the controller input and $f(z)$ and $g(z)$ are polynomials. Suppose that there exists a Lyapunov function $V(z)$ such that $V(z)$ is positive definite in a neighbourhood of the origin and polynomials $p(z)$, $q(z)$ satisfying

$$-\frac{\partial V}{\partial z}(f(z) + g(z)u) - (p(z) - q(z)u) \geq 0\, \forall z, u.$$

Then the origin of the state space is a stable equilibrium of the system.

Proof. We consider the stability of constrained dynamical systems as in Theorem 7.3. Using the same argument as in Proposition 8.2, if

$$-\frac{\partial V}{\partial z}(f(z)+g(z)u),$$

is nonnegative on the set

$$\{z \in \mathbb{R}^{n_z},\ u \in \mathbb{R}^{n_u} \mid p(z) - q(z)u = 0,\ q(z) \neq 0\},$$

then it is also nonnegative on the set

$$\left\{z \in \mathbb{R}^{n_z},\ u \in \mathbb{R}^{n_u} \;\middle|\; \frac{p(z)}{q(z)} - u = 0,\ q(z) \neq 0\right\},$$

which defines the controller in the closed-loop system. ■

To compute the Lyapunov function and polynomials that define the rational controller in Proposition 8.3 we can formulate an SOS program.

Proposition 8.4. Consider the dynamical system (8.6) in Proposition 8.3. Suppose there exists polynomial functions $V(z)$, $p(z)$, $q(z)$, a positive definite function $\rho(z)$ defined in Theorem 7.6 such that

$$\begin{aligned} V(z) - \rho(z) &\in \Sigma[z], \\ -\frac{\partial V}{\partial z}(f(z)+g(z)u) - (p(z) - q(z)u) &\in \Sigma[X], \\ p(z) &\in \mathbb{R}[X], \\ q(z) &\in \mathbb{R}[X], \\ q(z) &\neq 0, \end{aligned}$$

where $X = (z, u)$ is a vector of all of the states. Then the origin of the system is a stable equilibrium.

Proof. Using the techniques outlined in Chapter 2, the Lyapunov stability condition in Proposition 8.3 is replaced with SOS conditions to verify stability. ■

Proposition 8.4 allows us to reconstruct a stabilising controller from a feasibility test using SOS programming. However, the structure of the controller is limited as it is a simple rational function. We therefore extend this approach to a more expressive class of functions through rational NNs.

8.3.1 Extension to Rational Neural Network Controllers

The technique outlined in the previous section can be used to recover a stabilising controller that is a rational function of the system states. However, we can expand this approach to consider an NN architecture that contains rational functions similar to that of the one proposed in (8.4). We consider a state feedback controller $u(t) = \pi(z(t)) : \mathbb{R}^{n_z} \to \mathbb{R}^{n_u}$ as a rational NN such that

$$\begin{aligned} x^0(t) &= z(t), \\ x_i^{k+1}(t) &= \frac{p_i^k(x^k(t))}{q_i^k(x^k(t))}, \text{ for } i = 1, \dots, n_k, \ k = 0, \dots, \ell - 1, \\ u_i(t) = \pi_i(z(t)) &= \frac{p_i^\ell(x^\ell(t))}{q_i^\ell(x^\ell(t))}, \text{ for } i = 1, \dots, n_z, \end{aligned} \tag{8.7}$$

where $p_i^k(x^k(t))$, $q_i^k(x^k(t))$ are the polynomials that form the rational expression associated with the ith node in the $(k+1)^{\text{th}}$ layer. The number of neurons in the kth layer is denoted by n_k. We will drop the time dependence notation throughout the rest of this chapter for simplicity.

We can apply Proposition 8.4 and Theorem 7.7 in Section 7.2.4 due to the well-defined structure of this controller. The following proposition shows how Lyapunov stability over constrained dynamical systems can be used to recover a controller of this form.

Proposition 8.5. Consider the non-linear system

$$\dot{z} = f(z) + g(z)u,$$

$$u = \pi(z),$$

where $z \in \mathbb{R}^{n_z}$ are the system states, $u \in \mathbb{R}^{n_u}$ is the controller input and $f(z)$ and $g(z)$ are polynomials. Consider the controller structure $\pi(z)$ defined in (8.7) and the region given by (7.7). Suppose there exist polynomial functions $V(z)$, $p_i^k(x^k) \forall i = 1, \dots, n_{k+1}, k = 0, \dots, \ell$,

and $q_i^k(x^k)\,\forall i = 1,\dots,n_{k+1},\ k = 0,\dots,\ell$ satisfying the following conditions

$$\begin{aligned} V(z) - \rho(z) &\in \Sigma[z], \\ \rho(z) &> 0, \\ -\frac{\partial V}{\partial z}(z)(f(z)+g(z)u) - \sum_{k=1}^{n_d} s_k(X) d_k(z) \dots & \\ - \sum_{k=1}^{\ell}\sum_{i=1}^{n_k}\Big(p_i^k(x^k) - q_i^k(x^k)x_i^{k+1}\Big)\dots & \\ - \sum_{i=1}^{n_u}\Big(p_i^\ell(x^\ell) - q_i^\ell(x^\ell)u_i\Big) &\in \Sigma[X], \\ s_k(X) &\in \Sigma[X],\ \forall k = 1,\dots,n_d, \\ q_i^k(x^k) &\neq 0,\ \forall\, i = 1,\dots,n_k,\ k = 0,\dots,\ell, \end{aligned}$$

where X is a vector of all the system and NN states, i.e. $X = (x,u,z)$. Then the equilibrium of the system is stable.

Proof. This proposition follows from Proposition 8.4 by using the steps from Proposition 7.8. All of the equality expressions that define the rational NN in (8.7) are added to the Lyapunov stability condition. ■

The above proposition can generate a feasible SOS program, however the rational NN controller will be unrecoverable. This is because the coefficients in the $q_i^k(x^k)$ terms will be set to very small values by the SOS program, due to each term cancelling with the adjacent layers. We therefore propose an alternative rational NN controller structure in the following section to mitigate this issue.

8.3.2 Refined Rational Neural Network Controller

We consider a state feedback controller $u = \pi(z) : \mathbb{R}^{n_z} \to \mathbb{R}^{n_u}$ as a rational NN such that

$$\begin{aligned} y^0 &= z, \\ x_i^{k+1} &= \frac{\sum_{j=1}^{n_k} p_{i,j}^k(y^k) y_j^k}{q_i^k(z)},\ \text{for } i = 1,\dots,n_{k+1},\ k = 0,\dots,\ell-1, \\ y_i^{k+1} &= x_i^{k+1} + 1,\ \text{for } i = 1,\dots,n_k,\ k = 0,\dots,\ell-1, \\ u_i = \pi_i(z) &= \frac{\sum_{j=1}^{n_\ell} p_{i,j}^\ell(y^\ell) y_j^\ell}{q_i^\ell(z)}\Big(\sum_{m=1}^{n_z} z_m^2\Big),\ \text{for } i = 1,\dots,n_z, \end{aligned} \tag{8.8}$$

where $p_{i,j}^k(x^k)$, $q_i^k(z)$ are the jth polynomials that form the rational expression associated with the ith node in the $(k+1)^{\text{th}}$ layer.

Each x_j^k node in the NN contains a rational activation function and each y_j^k node is equal to the x_j^k term with the addition of a bias term which we set to unity. The y_j^k term that appears in the numerator of the rational activation function is to ensure that all terms in the x_i^{k+1} node are a function of all of the nodes in the kth layer and to impose more structure on the NN. The denominator $q_i^k(z)$ is a function of the system states to ensure that all nodes in the network are tied to the system states and not just the nodes in the previous layer. This will ensure that the SOS program does not set the coefficients in the $q_i^k(z)$ terms to very small values. The final layer contains a multiplier term which is a sum of all of the system states $\sum_{m=1}^{n_z} z_m^2$ to ensure that the controller input goes to zero at the origin.

We can then adapt Proposition 8.5 for this modified rational NN structure, to obtain a controller that can be recoverable from the feasibility test.

Proposition 8.6. Consider the non-linear system

$$\dot{z} = f(z) + g(z)u,$$

$$u = \pi(z),$$

where $z \in \mathbb{R}^{n_z}$ are the system states, $u \in \mathbb{R}^{n_u}$ is the controller input and $f(z)$ and $g(z)$ are polynomials. Consider the controller structure $\pi(z)$ defined in (8.8) and the region given by (7.7). Suppose there exist polynomial functions $V(z)$, $p_{i,j}^k(x^k)\ \forall i = 1, \ldots, n_{k+1},\ j = 1, \ldots, n_k,\ k = 0, \ldots, \ell$ and $q_i^k(z)\ \forall i = 1, \ldots, n_{k+1},\ k = 0, \ldots, \ell$ satisfying the following

conditions

$$
\begin{aligned}
V(z) - \rho(z) &\in \Sigma[z], \\
\rho(z) &> 0, \\
-\frac{\partial V}{\partial z}(z)(f(z)+g(z)u) - \sum_{k=1}^{n_d} s_k(X) d_k(z) \dots & \\
- \sum_{k=1}^{\ell} \sum_{i=1}^{n_k} \Big(\Big(\sum_{j=1}^{n_{k-1}} p_{i,j}^k(y^k) y_j^k \Big) - q_i^k(z) x_i^{k+1} \Big) \dots & \\
- \sum_{k=1}^{\ell} \sum_{i=1}^{n_k} t_{i,k}(X)(y_i^k - x_i^k - 1) \dots & \\
- \sum_{i=1}^{n_u} \Big(\sum_{j=1}^{n_\ell} \Big(p_{i,j}^\ell(y^\ell) y_j^\ell \Big(\sum_{m=1}^{n_z} z_m^2 \Big) \Big) - q_i^\ell(z) u_i \Big) &\in \Sigma[X], \\
s_k(X) &\in \Sigma[X],\ \forall k = 1, \dots, n_d, \\
t_{i,k}(X) &\in \mathbb{R}[X],\ \forall k = 1, \dots, \ell,\ i = 1, \dots, n_k, \\
q_i^k(x^k) &\neq 0,\ \forall\, i = 1, \dots, n_k,\ k = 0, \dots, \ell,
\end{aligned}
$$

where X is a vector of all the system and NN states, i.e. $X = (u, x, y, z)$. Then the equilibrium of the system is stable.

Proof. This proposition follows from Proposition 8.5, where the equations for (8.7) are replaced with the equations for the refined rational NN controller (8.8). ■

The SOS program in the above proposition is convex in the rational NN parameters and can hence be solved using SOS programming. Note that saturation and any uncertainty and robustness conditions can be incorporated in the same way that is demonstrated in Section 7.4.1. Proposition 8.6 presents a method to obtain a stabilising NN controller for a non-linear polynomial system in a convex way by solving one SOS optimisation problem. As described in Section 8.1, other recent approaches such as [184–188] rely on iterative algorithms or reinforcement learning formulations that are significantly more expensive to compute.

8.4 Numerical Examples

We demonstrate how the approach in Proposition 8.6 can be used to recover stabilising rational NN controllers through numerical examples. These examples were run on a four-core Intel Xeon processor @3.50GHz with 16GB of RAM. The SOS programs were

implemented using MATLAB and SOSTOOLS to parse the SOS constraints into an SDP, which is solved using MOSEK [212].

8.4.1 One Dimensional System

To show that this method is able to obtain a stabilising rational NN controller, we consider a very simple one dimensional linear system of the form

$$\dot{z} = z + u.$$

This system is unstable without a feedback controller. To recover a stabilising controller we set the size of the rational NN to be a small two layer network with a single node in each layer. The equations of the controller can be written as

$$\begin{aligned} x_1 &= \frac{\lambda_{1,1} z^4 + \lambda_{1,2} z^2}{\gamma_{1,1} z^2 + \gamma_{1,2}}, \\ y_1 &= x_1 + 1, \\ u &= \frac{(\lambda_{2,1} y_1^2 + \lambda_{2,2} y_1) z}{\gamma_{2,1} z^2 + \gamma_{2,2}}, \end{aligned}$$

where $\gamma_{1,1} \geq 0$, $\gamma_{1,2} > 0$, $\gamma_{2,1} \geq 0$, $\gamma_{2,2} > 0$. We also add saturation to the controller such that $-10 \leq u \leq 10$ and we enforce the local region of the state space to be $-10 \leq z \leq 10$. This is achieved by adding the constraints

$$\begin{aligned} u + 10 \geq 0,\ 10 - u \geq 0, \\ z + 10 \geq 0,\ 10 - z \geq 0, \end{aligned}$$

into the Lyapunov stability condition. The SOS program is able to recover a feasible controller which can stabilise the system to the zero equilibrium. The state trajectory and controller input for this NFL over time are shown in Figure 8.6 and 8.7 respectively.

8.4.2 Three Dimensional Non-linear System

We now consider the three dimensional non-linear system from Section 7.5.2 and attempt to find a stabilising rational NN controller for the system. We set the NN to have two layers and three nodes in each layer, with saturation $-10 \leq u \leq 10$. The system states are defined to operate in the region

$$1^2 - z_1^2 - z_2^2 - z_3^2 \geq 0.$$

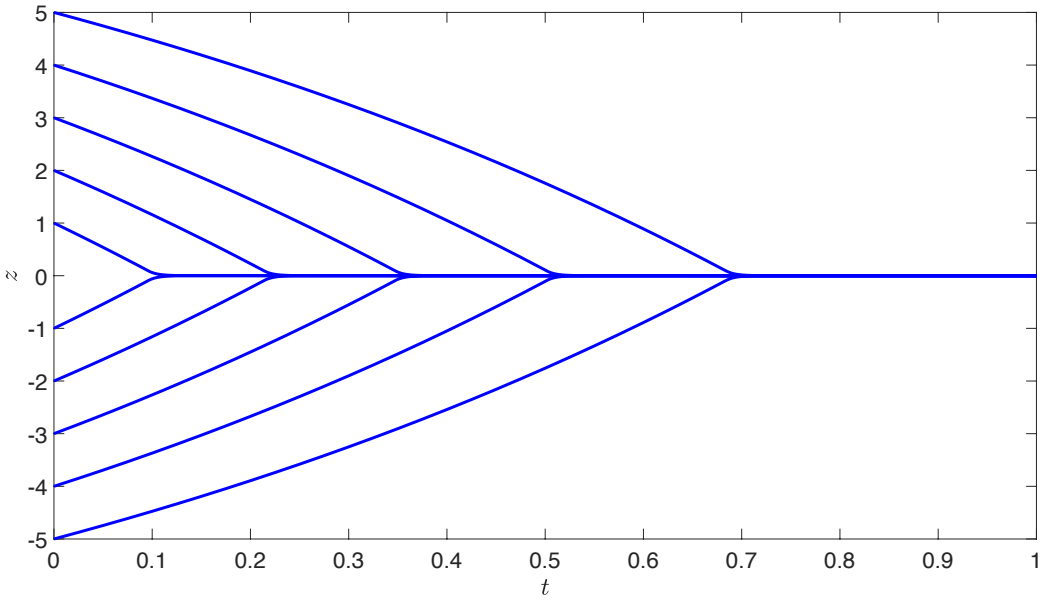


Figure 8.6: Plot showing the trajectories of the system state in Section 8.4.1 over time.

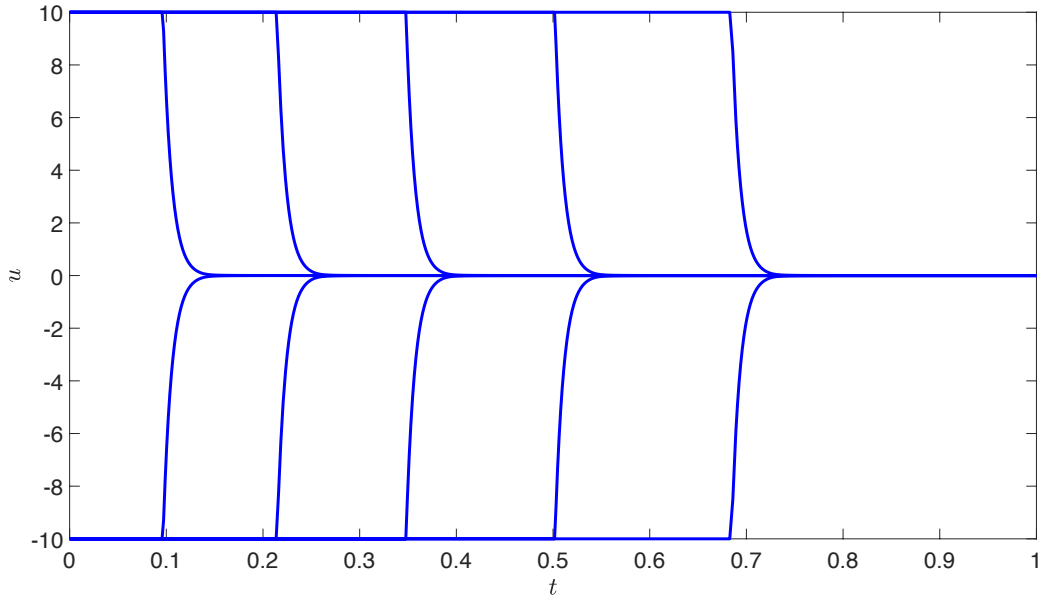


Figure 8.7: Plot showing the controller input of the system in Section 8.4.1 over time.

Each polynomial in the rational NN is assigned to contain zeroth to fourth order terms. We define a quartic Lyapunov function and second order polynomials for the s_k and $t_{i,k}$ terms. The trajectories for this system are shown in Figure 8.8, which shows that the controller can successfully stabilise the system.

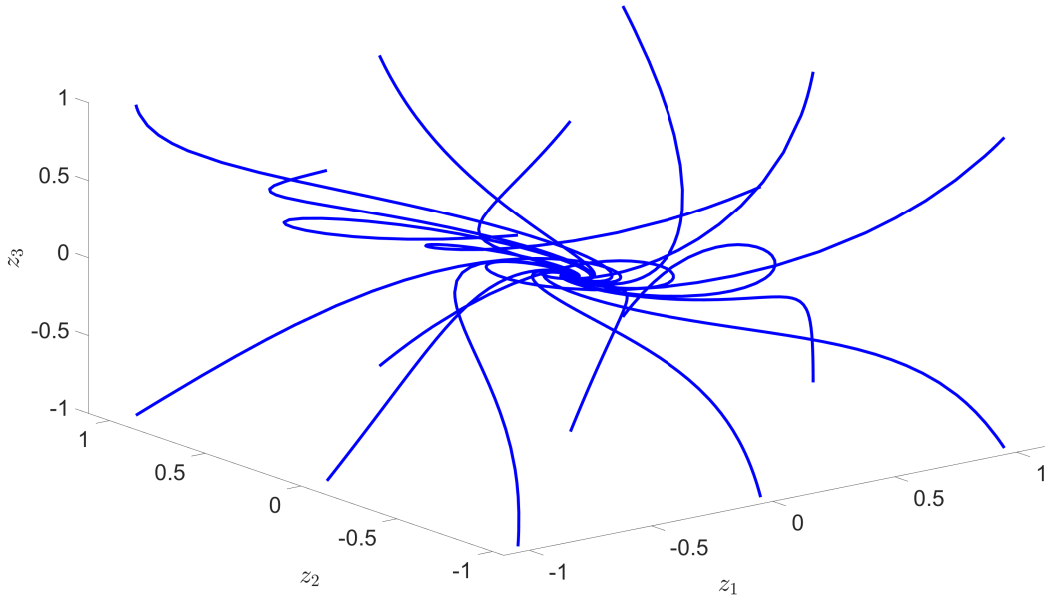


Figure 8.8: Plot showing the trajectories of the system states for the example in Section 8.4.2.

8.4.3 Non-linear Inverted Pendulum

We now consider the inverted pendulum from Section 7.5.3. As shown in [216], we can rewrite the dynamics as a four dimensional polynomial system. By making the substitution $z_3 = \sin(z_1)$, $z_4 = \cos(z_1)$ the system can be written as

$$\begin{aligned}
\dot{z}_1 &= z_2, \\
\dot{z}_2 &= \frac{g}{l} z_3 - \frac{\mu}{ml^2} z_2 + \frac{1}{ml^2} u, \\
\dot{z}_3 &= z_2 z_4, \\
\dot{z}_4 &= -z_2 z_3,
\end{aligned}$$

where $m = 0.15\,\text{kg}$, $l = 0.5\,\text{m}$, $\mu = 0.5\,\text{Nmsrad}^{-1}$, $g = 9.81\,\text{ms}^{-2}$ and the controller input is saturated such that $-1 \leq u \leq 1$. The system also requires the equality constraint

$$z_3^2 + z_4^2 - 1 = 0,$$

to be enforced. We do not define any region of the state space and instead consider global stability. We include the additional robustness constraints on the length of the pendulum to be ± 0.1 its original length and additive white noise w on the angular velocity such that $||w||_\infty \leq 0.1$. By making the substitution $\delta = 1/l$ the full dynamical

system can be written as

$$\begin{aligned}
\dot{z}_1 &= z_2, \\
\dot{z}_2 &= g\delta z_3 - \frac{\mu\delta^2}{m} z_2 + \frac{\delta^2}{m} u + w, \\
\dot{z}_3 &= z_2 z_4, \\
\dot{z}_4 &= -z_2 z_3, \\
0 &= z_3^2 + z_4^2 - 1, \\
0 &\leq 1^2 - u^2, \\
0 &\leq 0.1^2 - w^2, \\
0 &\leq (1/0.4 - \delta)(\delta - 1/0.6).
\end{aligned}$$

The Lyapunov function must be carefully constructed due to the z_4 state being equal to one at the origin. We therefore define the Lyapunov function to be the sum of two Lyapunov functions, the first of which is defined as

$$V_1(z_3, z_4) = a_1 z_3^2 + a_2 z_4^2 + a_3 z_4 + a_4$$

and the second is quadratic in z_1 and z_2. To ensure that the Lyapunov function is zero at the origin we must enforce

$$a_2 + a_3 + a_4 = 0.$$

To ensure that the Lyapunov function is positive definite, we define

$$\rho(z) = \epsilon_1 z_1^2 + \epsilon_2 z_2^2 + \epsilon_3 (1 - z_4),$$

where $\epsilon_1 \geq 0.1$, $\epsilon_2 \geq 0.1$, $\epsilon_3 \geq 0.1$.

The size of the rational NN controller is a two layer network with four nodes in each layer. By setting the size of the polynomials in the network to be between zeroth and fourth order, we are able to recover a stabilising controller. The trajectories for this NFL are shown in Figure 8.9. We can see that the controller initially drives the system states towards a manifold and then moves it towards the equilibrium. Since the controller is discontinuous at the manifold, further analysis is required to show stability of the system.

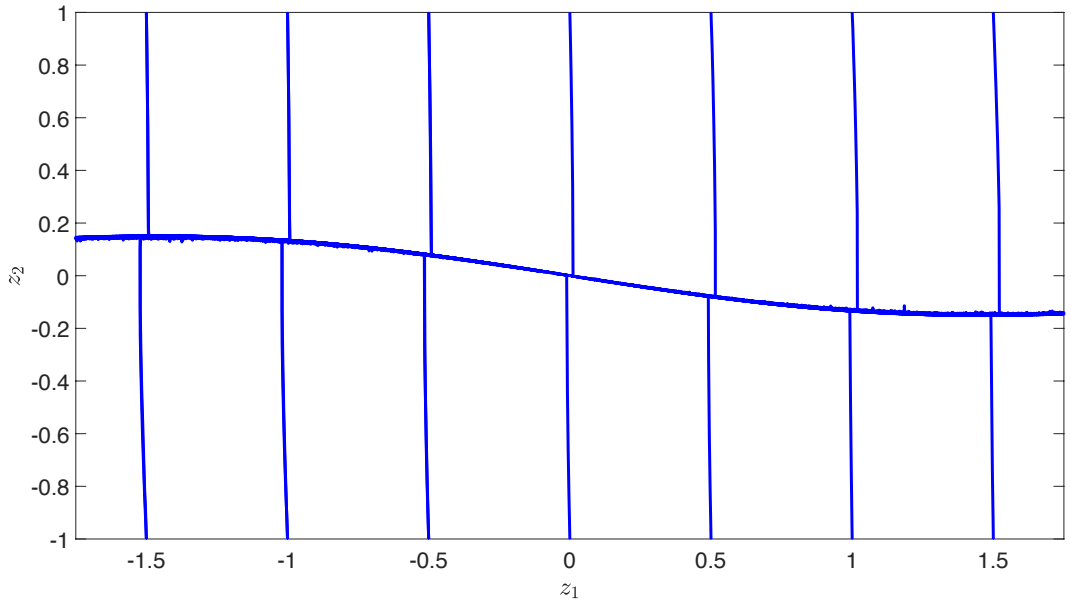


Figure 8.9: Plot showing the trajectories of the system states for the example in Section 8.4.3.

8.5 Conclusion

In this chapter, we analyse the use of rational NNs in previous application areas. We present novel rational activation functions to approximate the traditional sigmoid and tanh functions and show how they can be used in robustness problems for NFLs. We argue that rational activation functions can be replaced with a general rational NN structure where each layer is convex in the NN's parameters. We then propose a method to recover a stabilising controller from a feasibility test and then extend this approach to rational NNs. This structure is refined to make it more compatible when used in conjunction with SOS programming. Through numerous numerical examples we show how this approach can be used to recover stabilising rational NN controllers for NFLs with non-linear plants with noise and parametric uncertainty.

9

Conclusion

9.1 Thesis Summary

The key outcome of this thesis is the improvement of both the scalability and accuracy of tests that obtain robustness certificates for NNs and NFLs. The thesis is underpinned by two main observations. The first is the natural cascading structure which is possessed in a feed-forward fully connected NN. The second is that the problem of verifying NNs can be framed as an emptiness condition using Psatz.

In Chapter 3 we evaluated a recently proposed SDP framework for verifying fully connected feed-forward NNs. We observed a sparsity pattern in the LMI constraints and exploited it to improve the solve time of the SDP. This approach utilised theories from chordal graphs and semidefinite matrices to decompose the resulting SDP into a smaller problem. The numerical results that we obtained showed a significant speed-up in the computational time to solve the optimisation problem. Hence we showed that the robustness properties of the NN's output can be verified more efficiently.

In Chapter 4 we investigated the NN robustness problem from a general framework. The Psatz is used to combine real algebraic geometry with polynomial optimisation. This allowed the problem to be set as the emptiness of an appropriately constructed semi-algebraic set, which described the input-output properties of the NN; this problem was then solved using an equivalent SDP. To construct the semi-algebraic set we used existing constraints on the ReLU activation function and proposed new bounds on the sigmoid and tanh activation functions, that consisted of two overlapping sector constraints. Through numerous examples we demonstrated that this formulation can

provide the tightest bounds on the NN's output compared to other SDP formulations. This approach is versatile as we were able to easily trade off the amount of computational time required and the solution accuracy.

The ideas from Chapter 3 and Chapter 4 were combined in Chapter 5, which proposed a sparse polynomial optimisation framework. We showed that constructing the semi-algebraic set in an appropriate way leads the optimisation problem to possess a chordal sparsity pattern. We highlighted that common constraints used in NN verification will exhibit this type of sparsity. The optimisation framework was implemented with the CS-TSSOS hierarchy. Through numerous numerical examples we showed that this method can outperform state-of-the-art solvers in terms of both the computational time required and the accuracy of the output set.

The above ideas were applied to non-linear NFLs in Chapter 6 and Chapter 7. In the former, reachability analysis was addressed. This was achieved by abstracting the NN controller's input-output properties as a semi-algebraic set, which was combined with the dynamics of the plant model. The sparse polynomial optimisation framework developed in Chapter 5 was then applied to this problem. Through numerical experiments we showed that this method can compute tight approximations of the true reachable sets of an NFL. We emphasized that the main advantage of this approach is that non-linear polynomial dynamics can be easily incorporated, whilst being computed scalably and accurately. Stability of continuous-time NFLs was analysed in Chapter 7. We treated the NFL as a non-linear plant model acting over a constrained dynamical system, which was defined by a semi-algebraic set. We used Lyapunov stability theory on constrained dynamical systems and SOS programming to propose a method to compute Lyapunov certificates on non-linear NFLs. We also proposed another SOS program to compute the region of attraction of the NFL's equilibrium. In addition, we showed that this approach can allow for robustness analysis to be performed if the system possesses parametric or other uncertainties. Our method was tested on various numerical examples and was compared to existing methods, showing improved robustness certificates.

Rational NNs were introduced in Chapter 8 and their use in existing literature was discussed. We argued that recent approaches to synthesise NN controllers rely on

conservative approximations of the non-linear activation functions. Motivated by this, we proposed novel rational activation functions, which approximate the sigmoid and tanh functions. We then showed that these rational activation functions can be used to obtain very good robustness metrics when analysing robustness properties of an NFL. We reasoned that in order to harness the benefits of rational functions, a general rational NN structure can be considered, where the polynomials in each layer in the NN are convex in the NN's parameters. Additionally, we presented a novel method to find a stabilising controller from an SOS feasibility test. We incorporated this approach with a refined rational NN structure to make it more compatible with SOS programming. We tested our method through numerical examples and we showed that the approach was able to recover stabilising rational NN controllers for systems that have non-linear plants, additive noise and parametric uncertainty.

9.2 Future Work

There is significant scope for future research related to the work in this thesis. This section describes some of these research directions.

Further Neural Network Architectures

The analysis in this thesis has mostly focused on ReLU, sigmoid and tanh activation functions, as they are some of the most commonly used activation functions in practical implementations of NNs. As mentioned throughout the thesis, the ideas from the thesis can easily be applied to other activation functions, such as the ones in Section 2.4.4 and [99, 197]. Further investigation into the rational and irrational activation functions proposed in Chapter 8 would be of merit. It may also be possible to use further activation functions that align better with the SOS programming framework to achieve improved robustness guarantees.

The examples in Chapters 3, 4 and 5, which address NNs in isolation, have only focused on randomly generated NNs to show scalability and accuracy improvements. It is likely that these methods will perform similarly well when applied to real-life examples, however a comparison of their performance would be insightful. Furthermore, it would

be interesting to conduct a more in-depth investigation into how to trade off solution accuracy against computational time for the benefit of practical implementation.

This thesis has assumed a fully connected NN, which is intrinsically dense; if the network could be shaped in a way during training then the optimisation problem could become even sparser. In this case there are methods such as NN pruning that can be used to simplify the network [228] and decrease the number of constraints in the optimisation problem. Related works in [229] synthesise a reduced order NN with robustness guarantees. There also exist more complex NN architectures such as long short-term memory networks [24], recurrent NNs [154], convolutional NNs [25] and transformer models [230], which also suffer from robustness issues. It may be possible to extend the methods in this thesis to those structures. Initial works to analyse robustness of these models include [231] for long short-term memory networks and recurrent NNs, [133, 232] for convolutional NNs and [233] for transformer models.

Semidefinite Programming Algorithms

Although the examples in this thesis have been applied to moderately-sized NNs, these SDP-based approaches still run into scalability issues. Despite the polynomial time computation of second order SDP solvers such as interior-point methods, solving large SDPs is still a huge open-ended research question. It may be possible to use recent advances in solving algorithms for SDPs or decomposition methods to help with these scalability issues [74]. For example, first-order methods such as CDCS [95] can be used to improve scalability further with the trade-off of a less accurate solution. One example of a first order method applied to the NN verification problem is DeepSplit [109], which uses alternating direction method of multipliers (ADMM). However, it relies on an LP relaxation, which makes the solution to the problem conservative. There may be other first order SDP algorithms that could be utilised effectively for this problem. There are also variations of SOS relaxations, such as Diagonally Dominant Sum of Squares and Scaled Diagonally Dominant Sum of Squares [85], which can improve the speed of computation. In addition, there are many different hierarchies that can be used for

sparsity exploitation. CS-TSSOS [207] was proposed very recently and there may be other decomposition hierarchies that would work well for NNs.

Additional Robustness Properties of Neural Feedback Loops

There are many future research directions for work surrounding NFLs. It may be possible to use the methods in this thesis to analyse other robustness properties of NFLs, such as input-output properties and backward reachable sets [141]. It would also be beneficial to combine this method with other recent approaches to improve both the scalability and accuracy of the reachable sets. Lipschitz constants have been a popular metric to analyse NFLs [149, 172], however their utility is limited since a single value is used to evaluate the system. Often the perturbation is only considered in a single direction. To overcome this, advanced ideas from robust control theory [234] can be used to consider more complex perturbations to obtain a matrix of values to quantify the system's behaviour. Another direct extension of the work in this thesis is using sparsity exploiting methods proposed in [67] (Chapter 5) to reduce the computational time required to solve the SOS program for the stability problem. This would also allow us to extend these results to larger-scale plant models with larger NN controllers.

There has been a proportionally larger amount of research that has been produced for verification of NNs as input-output functions compared to NFLs. It may be possible to combine the recent approaches from NNs in isolation and the work in this thesis to improve performance. For example, finding a way to incorporate state-of-the-art NN verification solvers such as α-β-CROWN [103] might be beneficial.

Neural Network Controller Synthesis

We would like to extend the SOS method to focus on the control synthesis problem for traditional NN architectures, by optimising the performance of the NN whilst providing robustness guarantees in the process. This problem has been investigated in [151, 186–188], which require the use of very conservative sector constraints on the activation functions, usually without any prepossessing bounds. A loop transformation along with the S-procedure and Schur complement [235] are used to obtain a convex SDP constraint,

which can be used in the training process. However, it may be possible to get better results using SOS methods by using higher order multipliers and tighter bounds on the activation functions. The equality constraints for the rational activation functions proposed in Section 8.2 will not work with this framework as Schur complement is only valid for inequality constraints. However, the approach in Section 8.3 could be extended to consider the design of rational NNs. A cost function could potentially be augmented into the SOS constraints or incorporated into some iterative algorithm to optimise the performance of the controller as well as ensuring stability.

More generally, there are many frameworks based on reinforcement learning that can be used to learn an NN controller from data. An optimisation problem of this type could involve optimising a reward function subject to an LMI constraint. However, synthesising a controller subject to safety constraints in this way is challenging. One recent approach [150] proposes a training procedure for an NN subject to an SDP constraint based on Lipschitz constants. It may be possible to generalise this to other robustness specifications and to align it in a reinforcement learning framework. A popular data-based approach that was proposed recently in [36] uses a Hankel matrix to represent the system with data and therefore synthesise a controller. It may be possible to combine this synthesis approach with NN controllers to get improved performance.

Outlook

There are many remaining difficulties surrounding the verification of NNs and NFLs. As the size of some recent NN architectures have increased massively, the biggest challenge is the scalability of NN verification algorithms. Even the most conservative and efficient methods are insufficient to analyse models of this magnitude. Without significant advances in solving algorithms, SDP-based approaches may not be appropriate for larger scale NNs. However, they have the potential to significantly improve the accuracy when the size of the NNs are smaller. This is a rapidly developing field and most control-related applications do not require such large networks. With continued development, a combination of these methods will lead to more accurate and efficient algorithms to verify NNs and NFLs and allow for their widespread use in safety-critical applications.

Appendices

A

CS-TSSOS Hierarchy

This appendix outlines the correlative sparsity term sparse sum of squares hierarchy that is used in the neural network verification problem in Chapters 5 and 6. In doing so, the term sparse sum of squares and chordal term sparse sum of squares hierarchies are introduced.

A.1 Sum of Squares Hierarchies

Consider a polynomial f with $\text{supp}(f) \subseteq \mathbb{A}$ and a fixed exponent set $\mathbb{A} \subset \mathbb{N}^n_{2d}$. Using correlative sparsity will only exploit the sparsity from couplings between variables from the correlative sparsity graph of $\mathbb{A}$. This is limiting as there may be some further structure in $\mathbb{A}$ such as term sparsity that can be exploited. Furthermore, Example 5.1 shows that only using term sparsity is very limiting. Motivated by this, works from [207, 208] have introduced correlative sparsity term sparse sum of squares (CS-TSSOS), term sparse sum of squares (TSSOS) and chordal term sparse sum of squares (chordal-TSSOS) as more advanced decomposition hierarchies. However, there are many other hierarchies that can be used for polynomial optimisation problems.

The hierarchies being considered here work by considering a sequence of graphs $\{\mathcal{G}(\mathbb{B}, \mathcal{E}_k)\}_{k\geq 0}$ for clique-based Gram matrix decomposition, where increasing k gives increasingly connected sparsity graphs ($\mathcal{E}_k \subseteq \mathcal{E}_{k+1}$).

For the SDP in (5.1) to be feasible, one should require that $\mathbb{A} \subseteq \text{supp}((x^{\mathbb{B}})^T Q x^{\mathbb{B}})$. Therefore, the construction of the sparsity graphs starts with the objective to ensure that each edge set $\mathcal{E}_k$ contains at least all edges (β, γ) with $\beta + \gamma \in \mathbb{A}$. Additionally, the diagonal entries of the Gram matrix $Q_{\beta,\beta}$ should not be forced to be zero. This is because if the monomial x^β is made redundant, then β could be removed from the

exponent set $\mathbb{B}$. Therefore, for the hierarchies considered here the initial exponent set $\mathbb{B}_0$ and initial edge set $\mathcal{E}_0$ are defined as

$$\begin{aligned}\mathbb{B}_0 &:= 2\mathbb{B} \cup \mathbb{A},\\ \mathcal{E}_0 &= \{(\beta,\gamma) \in \mathbb{B}\times\mathbb{B} \,:\, \beta+\gamma \in \mathbb{B}_0\}.\end{aligned}$$

To extend an edge set $\mathcal{E} \subset \mathbb{B}\times\mathbb{B}$ subject to a given rule, an extension operator is defined such that $\mathbb{E} : \mathbb{B}\times\mathbb{B} \to \mathbb{B}\times\mathbb{B}$. Consequently the support sets $\mathbb{B}_k$ and the edge sets $\mathcal{E}_1 \subseteq \mathcal{E}_2 \subseteq \ldots \subseteq \mathcal{E}_k \subseteq \ldots$ are defined by the iterative rule

$$\begin{aligned}\mathbb{B}_k &:= \{\beta+\gamma \,:\, (\beta,\gamma) \in \mathcal{E}_k\},\\ \mathcal{E}_k &:= \mathbb{E}(\{(\beta,\gamma) \in \mathbb{B}\times\mathbb{B} \,:\, \beta+\gamma \in \mathbb{B}_{k-1}\}).\end{aligned} \tag{A.1}$$

Two important properties of this operator are that it guarantees that $\mathcal{E}_k \subseteq \mathcal{E}_{k+1}$ and the sequence converges to an edge set $\mathcal{E}^\star$ in a finite number of operations. Therefore, the sequence of sparsity graphs $\{\mathcal{G}(\mathbb{B},\mathcal{E}_k)\}_{k\geq 1}$ that is obtained using this iterative rule is given by

$$\Sigma[\mathbb{A};\mathcal{E}_1] \subseteq \Sigma[\mathbb{A};\mathcal{E}_2] \subseteq \ldots \subseteq \Sigma[\mathbb{A};\mathcal{E}^\star] \subseteq \Sigma[\mathbb{A}].$$

The choice of operator will generate different types of sparse SOS decomposition hierarchies. The following sections provide some example hierarchies.

A.2 TSSOS and Chordal-TSSOS

We will now show how particular extension operators can generate the TSSOS and chordal-TSSOS hierarchies.

Proposition A.1. ([205]) Consider the iterative rule in (A.1). The TSSOS hierarchy is generated if $\mathbb{E}$ is a block-completion operator such that it completes all connected components in the edge set

$$\{(\beta,\gamma) \in \mathbb{B}\times\mathbb{B} \,:\, \beta+\gamma \in \mathbb{B}_{k-1}\},$$

For each edge set $\mathcal{E}_k$ in the hierarchy, the corresponding Gram matrix Q has a block-diagonal structure and hence possesses chordal sparsity.

Proposition A.2. ([205]) Consider the iterative rule in (A.1). The chordal-TSSOS hierarchy is generated if $\mathbb{E}$ is an approximately minimal chordal extension operator such that it extends the edge set

$$\{(\beta,\gamma) \in \mathbb{B} \times \mathbb{B} \,:\, \beta+\gamma \in \mathbb{B}_{k-1}\}.$$

where $\mathcal{G}(\mathbb{B},\mathcal{E}_k)$ is chordal. For each edge set $\mathcal{E}_k$ in the hierarchy, the corresponding Gram matrix Q possesses chordal sparsity.

The choice between these hierarchies depends on the application/polynomial being analysed. In most cases chordal-TSSOS is less computationally complex than TSSOS, as its sparsity graph has fewer edges. However, it can result in less accurate solutions, due to the inherent trade-off of these approaches. Infact a result from [236] shows that if $\mathcal{E}^{\star}_{\text{TSSOS}}$ is the stabilised edge set of the TSSOS hierarchy, then f is SOS if and only if $f \in \Sigma[\mathbb{A},\mathcal{E}^{\star}_{\text{TSSOS}}]$.

Example A.3. ([205]) Consider the trivariate quartic polynomial

$$f(x) = 1 + x_1^4 + x_2^4 + x_3^4 + x_1^2x_2^2 + x_1^2x_3^2 + x_2^2x_3^2 + x_2x_3. \tag{A.2}$$

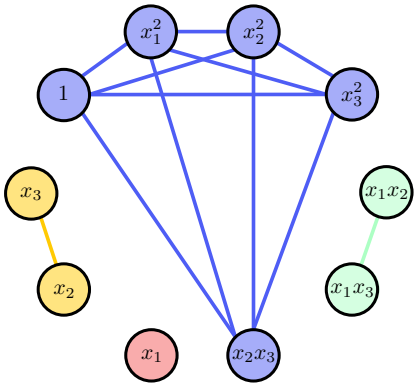
One can see that $f(x)$ possesses term sparsity but not correlative sparsity since the correlative sparsity graph is a complete graph. $f(x)$ admits the Gram matrix representation

$$f(x) = \frac{1}{8}(x^{\mathbb{B}})^T \begin{bmatrix} 8 & 3 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 8 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x^{\mathbb{B}}, \tag{A.3}$$

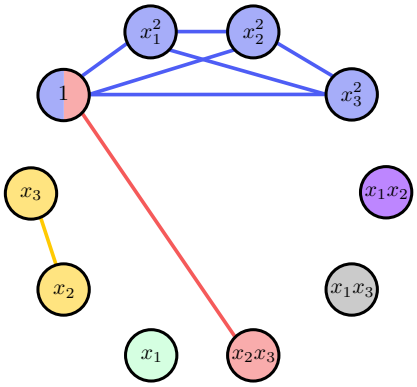
where

$$x^{\mathbb{B}} = \begin{bmatrix} x_3^2 & x_2^2 & x_1^2 & x_2x_3 & 1 & x_1 & x_1x_3 & x_1x_2 & x_3 & x_2 \end{bmatrix}^T.$$

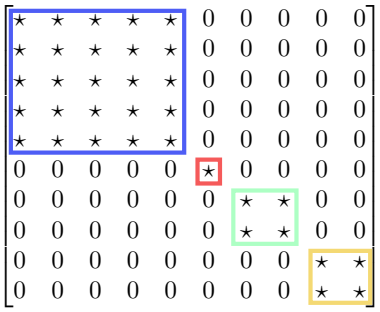
To evaluate the performance of TSSOS and chordal-TSSOS we show their sparsity graphs and corresponding Gram matrix sparsity patterns in Figure A.1. The TSSOS hierarchy stabilised after $k = 2$ iterations, whereas chordal-TSSOS stabilises at the first iteration ($k = 1$). We can see that both hierarchies are consistent with the Gram matrix



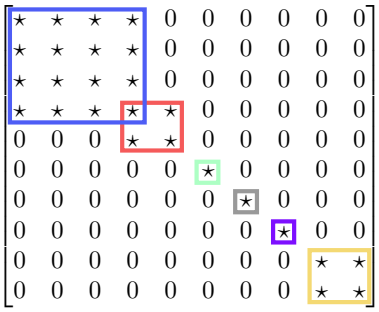
(a) $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=2})$ for TSSOS hierarchy



(b) $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=1})$ for chordal-TSSOS hierarchy



(c) Gram matrix for TSSOS hierarchy



(d) Gram matrix for chordal-TSSOS hierarchy

Figure A.1: Diagrams showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}^\star)$ and the corresponding Gram matrix representation of the polynomial in (A.2). The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$.

representation in (A.3), however the Gram matrix from the chordal-TSSOS hierarchy offers more sparsity than TSSOS.

We can also see that a smaller monomial basis given by

$$x^{\mathbb{B}} = \begin{bmatrix} x_3^2 & x_2^2 & x_1^2 & x_2x_3 & 1 \end{bmatrix}^T$$

would be sufficient to find the SOS decomposition, however finding a reduced basis has shown to be a challenging problem.

Note that both of these hierarchies are performed on the graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_k)$ that has $|\mathbb{B}|$ nodes. Recall that correlative sparsity in Section 5.2.3 is performed on the correlative sparsity graph, which is smaller with n nodes. However, it is possible to use the correlative sparsity graph in a similar hierarchy, as we will see in the next section.

A.3 Correlative Sparsity Term Sparse Sum of Squares

The CS-TSSOS hierarchy effectively combines the approaches of TSSOS and chordal-TSSOS. This hierarchy simultaneously exploits term and correlative sparsity and is described by the following proposition.

Proposition A.4. ([205]) Consider the iterative rule in (A.1) and the edge set generated by the correlative sparsity graph $\mathcal{E}_{\text{csp}}$ in (5.4). The CS-TSSOS hierarchy is constructed by the sequence of graphs

$$\mathcal{G}(\mathbb{B}, \mathcal{E}_k \cap \mathcal{E}_{\text{csp}}), \quad k \geq 1,$$

where $\mathcal{E}_k$ is the edge set generated from the TSSOS operator in Proposition A.1.

Similarly to chordal-TSSOS and TSSOS, the CS-TSSOS hierarchy stabilises to the edge set $\mathcal{E}^\star$ in a finite number of steps, this is because $\mathcal{E}_k \subseteq \mathcal{E}_{k+1}$. Consequently, the sequence of sparsity graphs $\{\mathcal{G}(\mathbb{B}, \mathcal{E}_k)\}_{k\geq 1}$ satisfies

$$\Sigma[\mathbb{A}; \mathcal{E}_1 \cap \mathcal{E}_{\text{csp}}] \subseteq \Sigma[\mathbb{A}; \mathcal{E}_2 \cap \mathcal{E}_{\text{csp}}] \subseteq \ldots \subseteq \Sigma[\mathbb{A}; \mathcal{E}^\star \cap \mathcal{E}_{\text{csp}}] \subseteq \begin{cases} \Sigma[\mathbb{A}; \mathcal{E}^\star] \\ \Sigma[\mathbb{A}; \mathcal{E}_{\text{csp}}]. \end{cases}$$

Each step of the hierarchy will produce a more expensive computational test, but may increase the chance of obtaining a feasible solution. Note that even in the final stabilised iteration there is no guarantee that the resulting SDP will be feasible. The following example from [236] shows how the Gram matrix becomes increasingly dense at each stage in the hierarchy.

Example A.5. ([236]) Consider the function

$$f(x) = 1+x_1x_2x_3+x_3x_4x_5+x_3x_4x_6+x_3x_5x_6+x_4x_5x_6+x_1^4+x_2^4+x_3^4+x_4^4+x_5^4+x_6^4, \quad \text{(A.4)}$$

which possesses both correlative and term sparsity. The correlative sparsity graph is shown in Figure A.2 and has maximal cliques $\mathcal{J}_1 = \{1, 2, 3\}$ and $\mathcal{J}_2 = \{3, 4, 5, 6\}$. To obtain the monomial vector, the full basis vector $(x^{\mathbb{N}_2^6})$ is used and the terms that violate the correlative sparsity graph are removed. This produces the monomial vector

$$x^{\mathbb{B}} = (x_2, x_1x_3, x_2x_3, x_1, x_1x_2, x_3, x_5x_6, x_4x_6, x_3x_6, x_5, x_4x_5, \\ x_3x_5, x_4, x_3x_4, x_6, x_1^2, x_2^2, x_3^2, 1, x_4^2, x_5^2, x_6^2)^T, \quad \text{(A.5)}$$

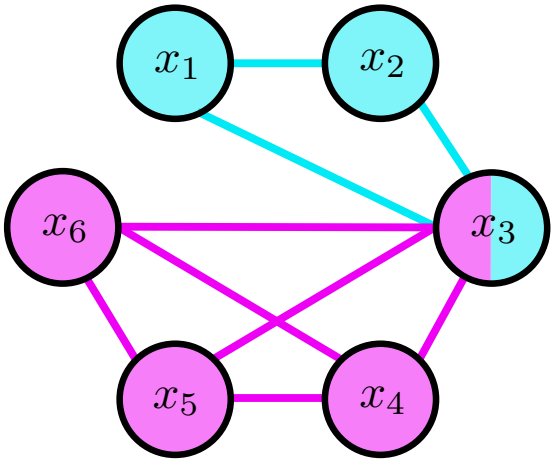


Figure A.2: Correlative sparsity graph of the polynomial in (A.4).

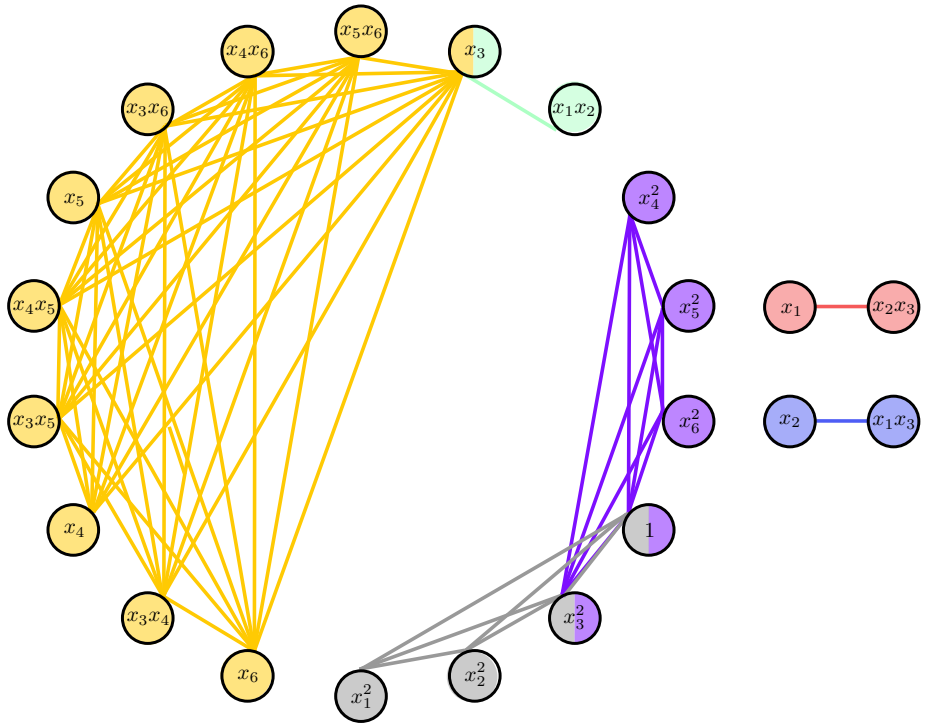
the ordering here is to ease with visualising the cliques.

Applying the CS-TSSOS hierarchy results in three iterations before it stabilises. The original fully dense Gram matrix is a 22×22 semidefinte constraint, whereas the first step of the hierarchy results in six semidefinte constraint with sizes 2, 2, 2, 10, 4 and 5. As comparison the TSSOS hierarchy results in five semidefinte constraints with sizes 2, 2, 2, 10 and 7. However, the second and third steps of the CS-TSSOS hierarchy result in a much more dense Gram matrix. There are four semidefinte constraints of sizes 2, 2, 6 and 15 for the second iteration and three semidefinte constraints of sizes 4, 6 and 15 for the third iteration. The sparsity graphs and corresponding Gram matrices for each step in the CS-TSSOS hierarchy are shown in Figure A.3, Figure A.4 and Figure A.5.

It is not guaranteed at any step that the resulting SDP will be feasible due to the conservative nature of correlative sparsity. In this example each iteration of the hierarchy results in a feasible SDP.

A.4 Extension to Semi-algebraic Sets

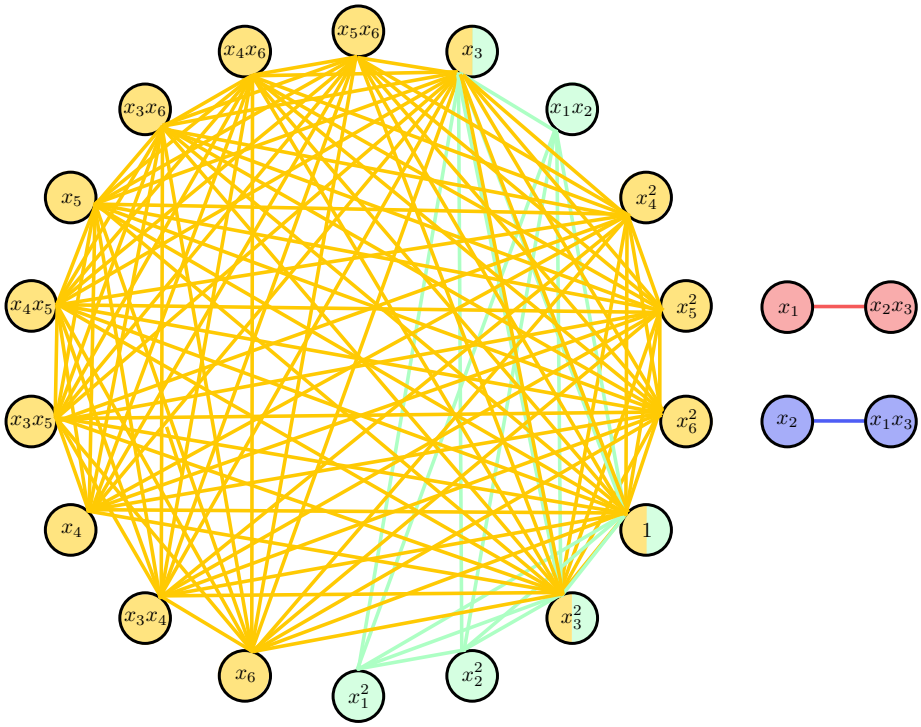
The TSSOS, chordal-TSSOS and CS-TSSOS hierarchies described above can be extended to semi-algebraic sets. The techniques used are similar to that of correlative sparsity outlined in Section 5.2.5, however the technicalities of this process are much more involved. For full details about how this is achieved the readers should refer themselves to the original works that introduce these hierarchies [207, 208, 236].



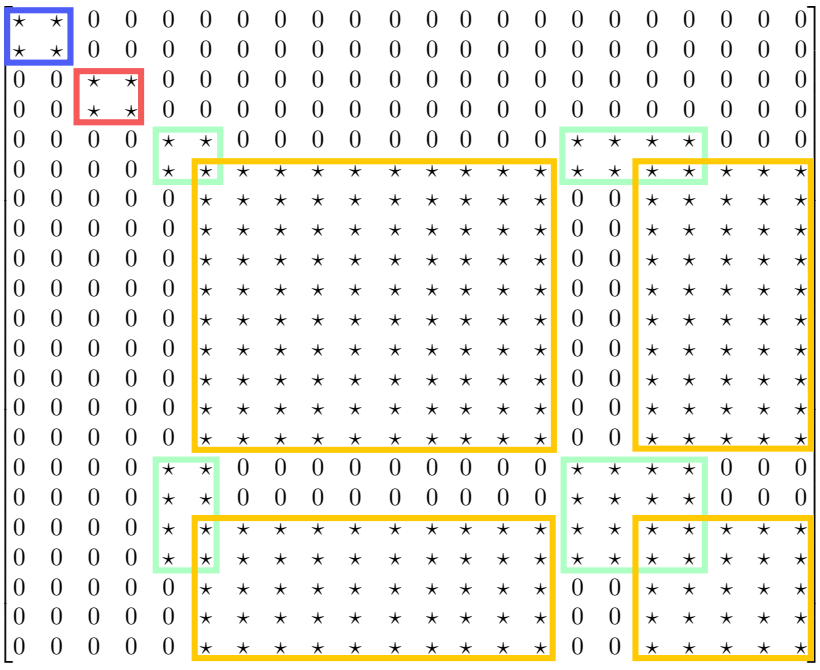
(a) $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=1})$

(b) Gram matrix

Figure A.3: Diagram showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=1})$ and the corresponding Gram matrix representation of the polynomial in (A.4) with the monomial basis in (A.5) for the first iteration of the CS-TSSOS hierarchy. The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$.

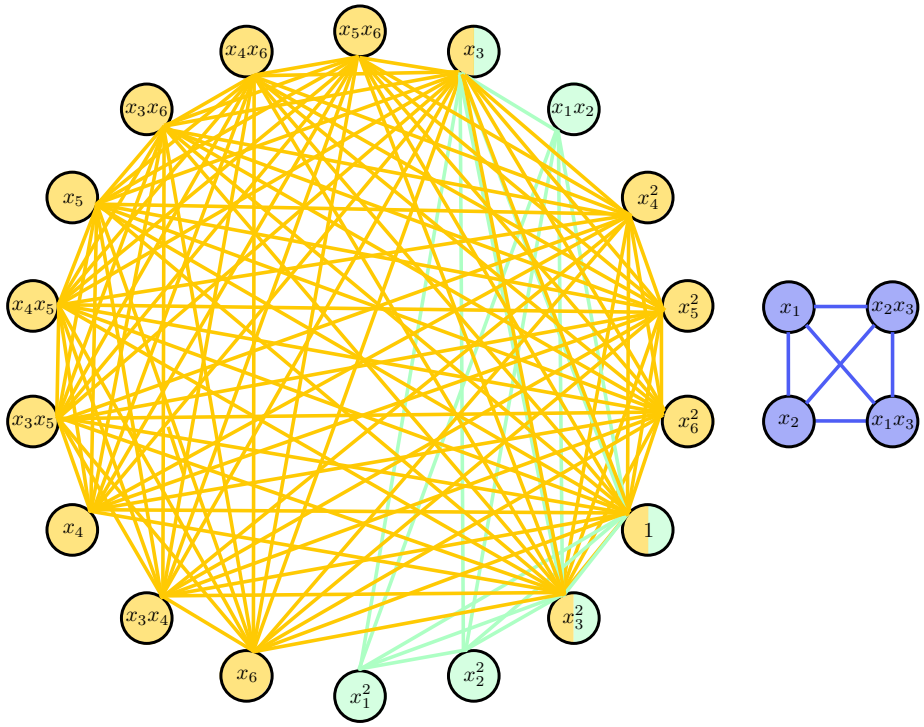


(a) $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=2})$

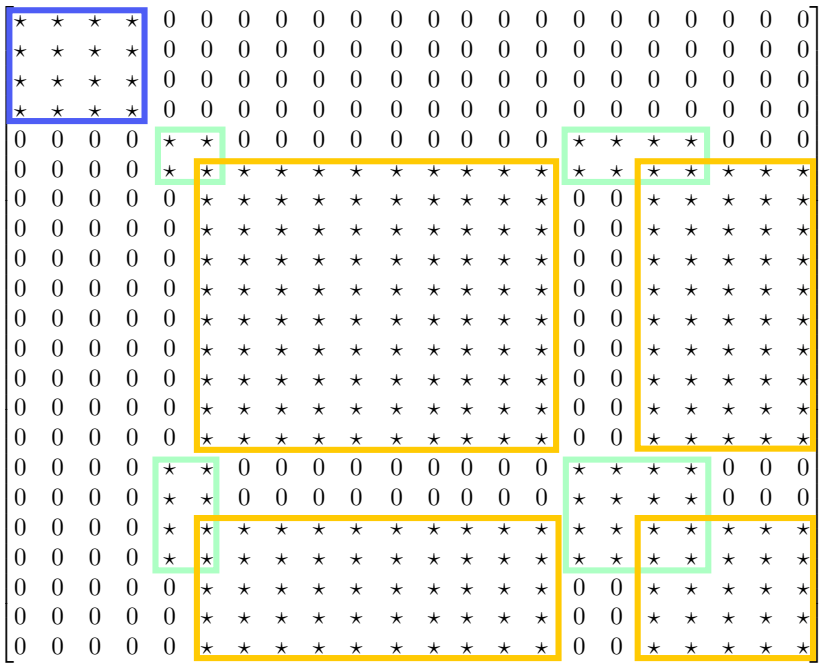


(b) Gram matrix

Figure A.4: Diagram showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}_{k=2})$ and the corresponding Gram matrix representation of the polynomial in (A.4) with the monomial basis in (A.5) for the second iteration of the CS-TSSOS hierarchy. The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$.



(a) $\mathcal{G}(\mathbb{B}, \mathcal{E}^\star)$



(b) Gram matrix

Figure A.5: Diagram showing the sparsity graph $\mathcal{G}(\mathbb{B}, \mathcal{E}^\star)$ and the corresponding Gram matrix representation of the polynomial in (A.4) with the monomial basis in (A.5) for the third and final iteration of the CS-TSSOS hierarchy. The colours represent the different cliques. The graph vertices in the sparsity graph are labeled by the monomials in $x^{\mathbb{B}}$.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[4] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. "In-datacenter performance analysis of a tensor processing unit". In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017, pp. 1–12.

[6] C. Zhang and Y. Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

[7] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[8] B. Bochenek and Z. Ustrnul. "Machine learning in weather prediction and climate analyses-applications and perspectives". In: *Atmosphere* 13.2 (2022), p. 180.

[9] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah. "A survey of deep learning applications to autonomous vehicle control". In: *IEEE Transactions on Intelligent Transportation Systems* 22.2 (2020), pp. 712–733.

[10] M. I. Razzak, S. Naz, and A. Zaib. "Deep learning for medical image processing: Overview, challenges and the future". In: *Classification in BioApps: Automation of Decision Making* (2018), pp. 323–350.

[11] T. Balaji, C. S. R. Annavarapu, and A. Bablani. "Machine learning algorithms for social media analysis: A survey". In: *Computer Science Review* 40 (2021), p. 100395.

[12] G. Marcus, E. Davis, and S. Aaronson. "A very preliminary analysis of dall-e 2". In: *arXiv preprint arXiv:2204.13807* (2022).

[13] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, Y. Shao, W. Zhang, B. Cui, and M.-H. Yang. "Diffusion models: A comprehensive survey of methods and applications". In: *arXiv preprint arXiv:2209.00796* (2022).

[14] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, et al. "MusicLM: Generating Music From Text". In: *arXiv preprint arXiv:2301.11325* (2023).

[15] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[16] M. Telgarsky. “Neural networks and rational functions”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3387–3393.

[17] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.

[18] F. Fan, J. Xiong, and G. Wang. “Universal approximation with quadratic deep networks”. In: *Neural Networks* 124 (2020), pp. 383–392.

[19] D. Psaltis, A. Sideris, and A. A. Yamamura. “A multilayered neural network controller”. In: *IEEE control systems magazine* 8.2 (1988), pp. 17–21.

[20] W. Li and J.-J. E. Slotine. “Neural network control of unknown nonlinear systems”. In: *1989 American Control Conference*. IEEE. 1989, pp. 1136–1141.

[21] W. T. Miller, R. S. Sutton, and P. J. Werbos. “Adaptive control using neural networks”. In: (1995).

[22] B. Widrow. “Pattern-recognizing control systems”. In: *Compurter and Information Sciences* (1964).

[23] K.-S. Fu. “Learning control systems–Review and outlook”. In: *IEEE transactions on Automatic Control* 15.2 (1970), pp. 210–221.

[24] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[25] K. Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[26] W. T. Miller, P. J. Werbos, and R. S. Sutton. *Neural networks for control*. MIT press, 1995.

[27] A. U. Levin and K. S. Narendra. “Control of nonlinear dynamical systems using neural networks: Controllability and stabilization”. In: *IEEE Transactions on neural networks* 4.2 (1993), pp. 192–206.

[28] C. Berner et al. “Dota 2 with large scale deep reinforcement learning”. In: *arXiv preprint arXiv:1912.06680* (2019).

[29] D. Silver, A. Huang, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (2016), pp. 484–503.

[30] V. Behzadan and A. Munir. “Vulnerability of deep reinforcement learning to policy induction attacks”. In: *Machine Learning and Data Mining in Pattern Recognition: 13th International Conference, MLDM 2017, New York, NY, USA, July 15-20, 2017, Proceedings 13*. Springer. 2017, pp. 262–275.

[31] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese. “Adversarially robust policy learning: Active construction of physically-plausible perturbations”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3932–3939.

[32] J. Morimoto and K. Doya. “Robust reinforcement learning”. In: *Neural computation* 17.2 (2005), pp. 335–359.

[33] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell. “Adversarial policies: Attacking deep reinforcement learning”. In: *arXiv preprint arXiv:1905.10615* (2019).

[34] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[35] B. Recht. “A Tour of Reinforcement Learning: The View from Continuous Control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 2.1 (2019), pp. 253–279.

[36] C. De Persis and P. Tesi. “Formulas for data-driven control: Stabilization, optimality, and robustness”. In: *IEEE Transactions on Automatic Control* 65.3 (2019), pp. 909–924.

[37] D. A. Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems* 1 (1988).

[38] S. Levine and P. Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in neural information processing systems* 27 (2014).

[39] F. Fabiani and P. J. Goulart. "Reliably-stabilizing piecewise-affine neural network controllers". In: *arXiv preprint arXiv:2111.07183* (2021).

[40] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin. "Learning quadrotor dynamics using neural network for flight control". In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 4653–4660.

[41] Y. Chen, Y. Shi, and B. Zhang. "Optimal control via neural networks: A convex approach". In: *arXiv preprint arXiv:1805.11835* (2018).

[42] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[43] R. Zhan and J. Wan. "Neural network-aided adaptive unscented Kalman filter for nonlinear state estimation". In: *IEEE Signal Processing Letters* 13.7 (2006), pp. 445–448.

[44] N. Noroozi, P. Karimaghaee, F. Safaei, and H. Javadi. "Generation of Lyapunov functions by neural networks". In: *Proceedings of the World Congress on Engineering*. Vol. 2008. 2008.

[45] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado. "Learning Lyapunov functions for piecewise affine systems with neural network controllers". In: *arXiv preprint arXiv:2008.06546* (2020).

[46] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo. "Formal synthesis of Lyapunov neural networks". In: *IEEE Control Systems Letters* 5.3 (2020), pp. 773–778.

[47] S. M. Richards, F. Berkenkamp, and A. Krause. "The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems". In: *Conference on Robot Learning*. PMLR. 2018, pp. 466–476.

[48] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. "Learning and verification of feedback control systems using feedforward neural networks". In: *IFAC-PapersOnLine* 51.16 (2018), pp. 151–156.

[49] S. Potluri, C. Diedrich, and G. K. R. Sangala. "Identifying false data injection attacks in industrial control systems using artificial neural networks". In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2017, pp. 1–8.

[50] T. Kriváchy, Y. Cai, J. Bowles, D. Cavalcanti, and N. Brunner. "Fast semidefinite programming with feedforward neural networks". In: *arXiv preprint arXiv:2011.05785* (2020).

[51] A. Nikseresht and A. Nazemi. "A novel neural network for solving semidefinite programming problems with some applications". In: *Journal of Computational and Applied Mathematics* 350 (2019), pp. 309–323.

[52] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. "Deep double descent: Where bigger models and more data hurt". In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (2021), p. 124003.

[53] S. Du and J. Lee. "On the power of over-parametrization in neural networks with quadratic activation". In: *International conference on machine learning*. PMLR. 2018, pp. 1329–1338.

[54] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. "Adversarial attacks on neural network policies". In: *arXiv preprint arXiv:1702.02284* (2017).

[55] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[56] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[57] A. Athalye, N. Carlini, and D. Wagner. "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples". In: *International conference on machine learning*. PMLR. 2018, pp. 274–283.

[58] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[59] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang. "A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 9835–9846.

[60] M. Fazlyab, M. Morari, and G. J. Pappas. "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming". In: *IEEE Transactions on Automatic Control* (2020).

[61] J. F. Sturm. "Using SeDuMi 1.02, A Matlab toolbox for optimization over symmetric cones". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 625–653.

[62] M. Newton and A. Papachristodoulou. "Exploiting Sparsity for Neural Network Verification". In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 715–727.

[63] G. Stengle. "A nullstellensatz and a positivstellensatz in semialgebraic geometry". In: *Mathematische Annalen* 207.2 (1974), pp. 87–97.

[64] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. Parrilo. "SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB". In: *arXiv preprint arXiv:1310.4716* (2013).

[65] K. C. Toh, M. J. Todd, and R. H. Tütüncü. "SDPT3 — A Matlab software package for semidefinite programming, Version 1.3". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 545–581.

[66] M. Newton and A. Papachristodoulou. "Neural Network Verification using Polynomial Optimisation". In: *Proceedings of the 60th Conference on Decision and Control*. 2021.

[67] M. Newton and A. Papachristodoulou. "Sparse Polynomial Optimisation for Neural Network Verification". In: *arXiv preprint arXiv:2202.02241* (2022).

[68] M. Newton and A. Papachristodoulou. "Reachability Analysis of Neural Feedback Loops using Sparse Polynomial Optimisation". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE. 2022, pp. 2745–2750.

[69] M. Newton and A. Papachristodoulou. "Stability of Non-linear Neural Feedback Loops using Sum of Squares". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE. 2022, pp. 6000–6005.

[70] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[71] L. Vandenberghe and S. Boyd. "Semidefinite programming". In: *SIAM review* 38.1 (1996), pp. 49–95.

[72] M. Grant and S. Boyd. *CVX: Matlab software for disciplined convex programming, version 2.1*. 2014.

[73] J. Löfberg. "YALMIP : A Toolbox for Modeling and Optimization in MATLAB". In: *In Proceedings of the CACSD Conference*. Taipei, Taiwan, 2004.

[74] A. Majumdar, G. Hall, and A. A. Ahmadi. "Recent scalability improvements for semidefinite programming with applications in machine learning, control, and robotics". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2020), pp. 331–360.

[75] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[76] D. Bertsekas. *Convex optimization theory*. Vol. 1. Athena Scientific, 2009.

[77] Z. Jarvis-Wloszek, R. Feeley, W. Tan, K. Sun, and A. Packard. "Some controls applications of sum of squares programming". In: *42nd IEEE international conference on decision and control (IEEE Cat. No. 03CH37475)*. Vol. 5. IEEE. 2003, pp. 4676–4681.

[78] P. A. Parrilo. "Semidefinite programming relaxations for semialgebraic problems". In: *Mathematical programming* 96 (2003), pp. 293–320.

[79] P. A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

[80] B. Legat, C. Coey, R. Deits, J. Huchette, and A. Perry. "Sum-of-squares optimization in Julia". In: *The First Annual JuMP-dev Workshop*. 2017.

[81] M. Laurent. "Sums of squares, moment matrices and optimization over polynomials". In: *Emerging applications of algebraic geometry* (2009), pp. 157–270.

[82] A. A. Ahmadi, G. Hall, A. Papachristodoulou, J. Saunderson, and Y. Zheng. "Improving efficiency and scalability of sum of squares optimization: Recent advances and limitations". In: *2017 IEEE 56th annual conference on decision and control (CDC)*. IEEE. 2017, pp. 453–462.

[83] T. S. Motzkin. "The arithmetic-geometric inequality". In: *Inequalities (Proc. Sympos. Wright-Patterson Air Force Base, Ohio, 1965)*. Academic Press, New York, 1967, pp. 205–224.

[84] A. A. Ahmadi and C. Dibek. "A sum of squares characterization of perfect graphs". In: *arXiv preprint arXiv:2110.08950* (2021).

[85] A. A. Ahmadi and A. Majumdar. "DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization". In: *SIAM Journal on Applied Algebra and Geometry* 3.2 (2019), pp. 193–230.

[86] Y. Zheng, G. Fantuzzi, and A. Papachristodoulou. "Sparse sum-of-squares (SOS) optimization: A bridge between DSOS/SDSOS and SOS optimization for sparse polynomials". In: *2019 American Control Conference (ACC)*. IEEE. 2019, pp. 5513–5518.

[87] J. B. Lasserre. "Global optimization with polynomials and the problem of moments". In: *SIAM Journal on optimization* 11.3 (2001), pp. 796–817.

[88] G. Stengle. "Complexity estimates for the Schmüdgen Positivstellensatz". In: *Journal of Complexity* 12.2 (1996), pp. 167–174.

[89] V Yakubovich. "S-procedure in nonlinear control theory". In: *Vestnick Leningrad Univ. Math.* 4 (1997), 73–93.

[90] N. Kakimura. "A direct proof for the matrix decomposition of chordal-structured positive semidefinite matrices". In: *Linear Algebra and its Applications* 433.4 (2010), pp. 819 –823.

[91] L. Vandenberghe and M. S. Andersen. *Chordal Graphs and Semidefinite Optimization*. 2015.

[92] J. Agler, W. Helton, S. McCullough, and L. Rodman. "Positive semidefinite matrices with a given sparsity pattern". In: *Linear Algebra and its Applications* 107 (1988), pp. 101 –149.

[93] Y. Zheng. *Chordal sparsity in control and optimization of large-scale systems (PhD thesis). University of Oxford*. 2019.

[94] S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. "Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion". In: *Mathematical programming* 129.1 (2011), pp. 33–68.

[95] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. "Chordal decomposition in operator-splitting methods for sparse semidefinite programs". In: *Mathematical Programming* 180.1 (2020), pp. 489–532.

[96] M. Garstka, M. Cannon, and P. Goulart. "COSMO: A conic operator splitting method for large convex problems". In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 1951–1956.

[97] M. S. Andersen, J. Dahl, and L. Vandenberghe. "Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones". In: *Mathematical Programming Computation* 2.3 (2010), pp. 167–201.

[98] K. Fujisawa, M. Fukuda, M. Kojima, K. Nakata, and M. Yamashita. "SDPA-C (SemiDefinite Programming Algorithm - Completion method) User's Manual — Version 6.2.0". In: 2004.

[99] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018).

[100] S. Bak, C. Liu, and T. Johnson. "The second international verification of neural networks competition (VNN-COMP 2021): Summary and results". In: *arXiv preprint arXiv:2109.00498* (2021).

[101] M. N. Müller, C. Brix, et al. "The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results". In: *arXiv preprint arXiv:2212.10376* (2022).

[102] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu. "First Three Years of the International Verification of Neural Networks Competition (VNN-COMP)". In: *arXiv preprint arXiv:2301.05815* (2023).

[103] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29909–29921.

[104] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. "On the effectiveness of interval bound propagation for training verifiably robust models". In: *arXiv preprint arXiv:1810.12715* (2018).

[105] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. "Fast and effective robustness certification". In: *Advances in neural information processing systems* 31 (2018).

[106] N. Kochdumper, C. Schilling, M. Althoff, and S. Bak. "Open-and closed-loop neural network verification using polynomial zonotopes". In: *arXiv preprint arXiv:2207.02715* (2022).

[107] R. Ehlers. "Formal verification of piece-wise linear feed-forward neural networks". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2017, pp. 269–286.

[108] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. "Automatic perturbation analysis for scalable certified robustness and beyond". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1129–1141.

[109] S. Chen, E. Wong, J. Z. Kolter, and M. Fazlyab. "DeepSplit: Scalable Verification of Deep Neural Networks via Operator Splitting". In: *arXiv preprint arXiv:2106.09117* (2021).

[110] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. "Towards fast computation of certified robustness for ReLU networks". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5276–5285.

[111] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. "Efficient neural network robustness certification with general activation functions". In: *Advances in neural information processing systems* 31 (2018).

[112] A. Raghunathan, J. Steinhardt, and P. Liang. "Semidefinite relaxations for certifying robustness to adversarial examples". In: *arXiv preprint arXiv:1811.01057* (2018).

[113] R. Brown, E. Schmerling, N. Azizan, and M. Pavone. "A Unified View of SDP-based Neural Network Verification through Completely Positive Programming". In: *arXiv preprint arXiv:2203.03034* (2022).

[114] R. Zhang. "On the tightness of semidefinite relaxations for certifying robustness to adversarial examples". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 3808–3820.

[115] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang, and P. Kohli. "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 5318–5331.

[116] M. Fazlyab, M. Morari, and G. J. Pappas. "An Introduction to Neural Network Analysis via Semidefinite Programming". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 6341–6350.

[117] B. Batten, P. Kouvaros, A. Lomuscio, and Y. Zheng. "Efficient Neural Network Verification via Layer-based Semidefinite Relaxations and Linear Cuts". In: *IJCAI*. 2021, pp. 2184–2190.

[118] K. D. Dvijotham, R. Stanforth, S. Gowal, C. Qin, S. De, and P. Kohli. "Efficient neural network verification with exactness characterization". In: *Uncertainty in artificial intelligence*. PMLR. 2020, pp. 497–507.

[119] T. Chen, J. B. Lasserre, V. Magron, and E. Pauwels. "Semialgebraic representation of monotone deep equilibrium models and applications to certification". In: *Advances in Neural Information Processing Systems* 34 (2021).

[120] M. Fazlyab, M. Morari, and G. J. Pappas. "Probabilistic verification and reachability analysis of neural networks via semidefinite programming". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE. 2019, pp. 2726–2731.

[121] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas. "Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 11427–11438.

[122] N. Hashemi, J. Ruths, and M. Fazlyab. "Certifying Incremental Quadratic Constraints for Neural Networks via Convex Optimization". In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 842–853.

[123] F. Latorre, P. Rolland, and V. Cevher. "Lipschitz constant estimation of neural networks via sparse polynomial optimization". In: *arXiv preprint arXiv:2004.08688* (2020).

[124] T. Chen, J. B. Lasserre, V. Magron, and E. Pauwels. "Semialgebraic optimization for Lipschitz constants of ReLU networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19189–19200.

[125] M. Revay, R. Wang, and I. R. Manchester. "Lipschitz Bounded Equilibrium Networks". In: *arXiv preprint arXiv:2010.01732* (2020).

[126] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. "Safety verification of deep neural networks". In: *International conference on computer aided verification*. Springer. 2017, pp. 3–29.

[127] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. "Reluplex: An efficient SMT solver for verifying deep neural networks". In: *International conference on computer aided verification*. Springer. 2017, pp. 97–117.

[128] V. Tjeng, K. Xiao, and R. Tedrake. "Evaluating robustness of neural networks with mixed integer programming". In: *arXiv preprint arXiv:1711.07356* (2017).

[129] R. R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and P. K. Mudigonda. "A Unified View of Piecewise Linear Neural Network Verification". In: *NeurIPS*. 2018.

[130] C.-H. Cheng, G. Nührenberg, and H. Ruess. "Maximum resilience of artificial neural networks". In: *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer. 2017, pp. 251–268.

[131] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli. "A Dual Approach to Scalable Verification of Deep Networks." In: *UAI*. Vol. 1. 2. 2018, p. 3.

[132] G. Singh, T. Gehr, M. Püschel, and M. Vechev. "An abstract domain for certifying neural networks". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–30.

[133] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. "AI2: Safety and robustness certification of neural networks with abstract interpretation". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 3–18.

[134] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. "Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach". In: *International Conference on Learning Representations*. 2018.

[135] G. Singh, R. Ganvir, M. Püschel, and M. Vechev. "Beyond the Single Neuron Convex Barrier for Neural Network Certification". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[136] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev. "PRIMA: General and precise neural network certification via scalable convex hull approximations". In: *arXiv preprint arXiv:2103.03638* (2021).

[137] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. Patel, and J. P. Vielma. "The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification". In: *arXiv preprint arXiv:2006.14076* (2020).

[138] S. A. Deka, D. M. Stipanović, and C. J. Tomlin. "Feedback-Control Based Adversarial Attacks on Recurrent Neural Networks". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 4677–4682.

[139] B. Evens, P. Latafat, A. Themelis, J. Suykens, and P. Patrinos. "Neural network training as an optimal control problem: An augmented Lagrangian approach". In: *arXiv preprint arXiv:2103.14343* (2021).

[140] T. Tanay, A. Sootla, M. Maggioni, P. K. Dokania, P. Torr, A. Leonardis, and G. Slabaugh. "Diagnosing and Preventing Instabilities in Recurrent Video Processing". In: *arXiv preprint arXiv:2010.05099* (2020).

[141] M. Everett. "Neural network verification in control". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 6326–6340.

[142] M. Everett, G. Habibi, and J. P. How. "Robustness analysis of neural networks via efficient partitioning with applications in control systems". In: *IEEE Control Systems Letters* 5.6 (2020), pp. 2114–2119.

[143] B. G. Anderson, Z. Ma, J. Li, and S. Sojoudi. "Tightened convex relaxations for neural network robustness certification". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 2190–2197.

[144] W. Xiang, H.-D. Tran, and T. T. Johnson. "Output reachable set estimation and verification for multilayer neural networks". In: *IEEE transactions on neural networks and learning systems* 29.11 (2018), pp. 5777–5783.

[145] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. "Formal security analysis of neural networks using symbolic intervals". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 1599–1614.

[146] V. Rubies-Royo, R. Calandra, D. M. Stipanovic, and C. Tomlin. "Fast neural network verification via shadow prices". In: *arXiv preprint arXiv:1902.07247* (2019).

[147] E. Wong and Z. Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *International conference on machine learning*. PMLR. 2018, pp. 5286–5295.

[148] A. Raghunathan, J. Steinhardt, and P. Liang. "Certified defenses against adversarial examples". In: *arXiv preprint arXiv:1801.09344* (2018).

[149] P. Pauli, A. Koch, J. Berberich, P. Kohler, and F. Allgöwer. "Training Robust Neural Networks Using Lipschitz Bounds". In: *IEEE Control Systems Letters* 6 (2022), pp. 121–126.

[150] P. Pauli, N. Funcke, D. Gramlich, M. A. Msalmi, and F. Allgöwer. "Neural network training under semidefinite constraints". In: *arXiv preprint arXiv:2201.00632* (2022).

[151] P. Pauli, J. Berberich, and F. Allgöwer. "Robustness analysis and training of recurrent neural networks using dissipativity theory". In: *at-Automatisierungstechnik* 70.8 (2022), pp. 730–739.

[152] H.-D. Tran, W. Xiang, and T. T. Johnson. "Verification approaches for learning-enabled autonomous cyber–physical systems". In: *IEEE Design & Test* 39.1 (2020), pp. 24–34.

[153] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T. Johnson. "Verification for machine learning, autonomy, and neural networks survey". In: *arXiv preprint arXiv:1810.01989* (2018).

[154] F. Bonassi, M. Farina, J. Xie, and R. Scattolini. "On Recurrent Neural Networks for learning-based control: Recent results and ideas for future developments". In: *Journal of Process Control* 114 (2022), pp. 92–104.

[155] G. Ernst, P. Arcaini, A. Donze, G. Fainekos, L. Mathesen, G. Pedrielli, S. Yaghoubi, Y. Yamagata, and Z. Zhang. "ARCH-COMP 2019 Category Report: Falsification". In: *EPiC Series in Computing* 61 (2019), pp. 129–140.

[156] T. T. Johnson, D. Manzanas Lopez, P. Musau, H.-D. Tran, E. Botoeva, F. Leofante, A. Maleki, C. Sidrane, J. Fan, and C. Huang. "ARCH-COMP20 category report: artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants". In: *EPiC Series in Computing* 74 (2020).

[157] G. Ernst, P. Arcaini, I. Bennani, A. Chandratre, A. Donze, G. Fainekos, G. Frehse, K. Gaaloul, J. Inoue, T. Khandait, et al. "ARCH-COMP 2021 Category Report: Falsification with Validation of Results". In: *EPiC Series in Computing* 80 (2021), pp. 133–152.

[158] G. Ernst, P. Arcaini, G. Fainekos, F. Formica, J. Inoue, T. Khandait, M. M. Mahboob, C. Menghi, G. Pedrielli, M. Waga, et al. "ARCH-COMP 2022 Category Report: Falsification with Ubounded Resources". In: *Proceedings of 9th International Workshop on Applied*. Vol. 90. 2022, pp. 204–221.

[159] M. Everett, G. Habibi, and J. P. How. "Efficient reachability analysis of closed-loop systems with neural network controllers". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 4384–4390.

[160] A. Lomuscio and L. Maganti. "An approach to reachability analysis for feed-forward ReLU neural networks". In: *arXiv preprint arXiv:1706.07351* (2017).

[161] 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)*. IEEE. 2020, pp. 5929–5934.

[162] W. Xiang and T. T. Johnson. "Reachability analysis and safety verification for neural network control systems". In: *arXiv preprint arXiv:1805.09944* (2018).

[163] M. Everett, G. Habibi, C. Sun, and J. P. How. "Reachability Analysis of Neural Feedback Loops". In: *IEEE Access* 9 (2021), pp. 163938–163953.

[164] G. Yang, G. Qian, P. Lv, and H. Li. "Efficient verification of control systems with neural network controllers". In: *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*. 2019, pp. 1–7.

[165] 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)*. IEEE. 2022, pp. 2737–2744.

[166] S. Chen, V. M. Preciado, and M. Fazlyab. "One-shot reachability analysis of neural network dynamical systems". In: *arXiv preprint arXiv:2209.11827* (2022).

[167] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson. "Reachable set estimation for neural network control systems: A simulation-guided approach". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.5 (2020), pp. 1821–1830.

[168] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. "ReachNN: Reachability analysis of neural-network controlled systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 18 (2019), pp. 1–22.

[169] J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu. "ReachNN*: A Tool for Reachability Analysis of Neural-Network Controlled Systems". In: *18th International Symposium on Automated Technology for Verification and Analysis, ATVA 2020*. Springer. 2020, pp. 537–542.

[170] S. Dutta, X. Chen, and S. Sankaranarayanan. "Reachability analysis for neural feedback systems using regressive polynomial rule inference". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 157–168.

[171] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. "Verisig: verifying safety properties of hybrid systems with neural network controllers". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 169–178.

[172] T. Entesari, S. Sharifi, and M. Fazlyab. "ReachLipBnB: A branch-and-bound method for reachability analysis of neural autonomous systems using Lipschitz bounds". In: *arXiv preprint arXiv:2211.00608* (2022).

[173] M. Cleaveland, L. Lindemann, R. Ivanov, and G. J. Pappas. "Risk verification of stochastic systems with neural network controllers". In: *Artificial Intelligence* 313 (2022), p. 103782.

[174] K. Tanaka. "An approach to stability criteria of neural-network control systems". In: *IEEE Transactions on Neural Networks* 7.3 (1996), pp. 629–642.

[175] K.-K. K. Kim, E. R. Patrón, and R. D. Braatz. "Standard representation and unified stability analysis for dynamic artificial neural network models". In: *Neural Networks* 98 (2018), pp. 251–262.

[176] M. Jin and J. Lavaei. "Stability-certified reinforcement learning: A control-theoretic perspective". In: *IEEE Access* 8 (2020), pp. 229086–229100.

[177] H. Yin, P. Seiler, and M. Arcak. "Stability analysis using quadratic constraints for systems with neural network controllers". In: *IEEE Transactions on Automatic Control* (2021).

[178] V. Magron, Y. Ebihara, H. Waki, N. H. A. Mai, D. Peaucelle, and S. Tarbouriech. "Stability Analysis of Recurrent Neural Networks by IQC with Copositive Mutipliers". In: *Control and Decision Conference (CDC) 2021*. 2021.

[179] P. Pauli, D. Gramlich, J. Berberich, and F. Allgöwer. "Linear systems with neural network nonlinearities: Improved stability analysis via acausal Zames-Falb multipliers". In: *arXiv preprint arXiv:2103.17106* (2021).

[180] M. Korda. "Stability and performance verification of dynamical systems controlled by neural networks: algorithms and complexity". In: *arXiv preprint arXiv:2102.02273* (2021).

[181] B. Karg and S. Lucia. "Stability and feasibility of neural network-based controllers via output range analysis". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 4947–4954.

[182] A. Aydinoglu, M. Fazlyab, M. Morari, and M. Posa. "Stability analysis of complementarity systems with neural network controllers". In: *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*. 2021, pp. 1–10.

[183] Y.-S. Wang, L. Weng, and L. Daniel. "Neural Network Control Policy Verification With Persistent Adversarial Perturbation". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10050–10059.

[184] M. Everett, B. Lütjens, and J. P. How. "Certifiable robustness to adversarial state uncertainty in deep reinforcement learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.9 (2021), pp. 4184–4198.

[185] P. L. Donti, M. Roderick, M. Fazlyab, and J. Z. Kolter. "Enforcing robust control guarantees within neural network policies". In: *arXiv preprint arXiv:2011.08105* (2020).

[186] H. Yin, P. Seiler, M. Jin, and M. Arcak. "Imitation learning with stability and safety guarantees". In: *IEEE Control Systems Letters* 6 (2021), pp. 409–414.

[187] F. Gu, H. Yin, L. El Ghaoui, M. Arcak, P. Seiler, and M. Jin. "Recurrent neural network controllers synthesis with stability guarantees for partially observed systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 5. 2022, pp. 5385–5394.

[188] N. Junnarkar, H. Yin, F. Gu, M. Arcak, and P. Seiler. "Synthesis of Stabilizing Recurrent Equilibrium Network Controllers". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE. 2022, pp. 7449–7454.

[189] R. Drummond, S. Duncan, M. Turner, P. Pauli, and F. Allgower. "Bounding the difference between model predictive control and neural networks". In: *Learning for Dynamics and Control Conference*. PMLR. 2022, pp. 817–829.

[190] F. Fabiani and P. J. Goulart. "Neural network controllers for uncertain linear systems". In: *arXiv preprint arXiv:2204.13209* (2022).

[191] S. Adhau, V. V. Naik, and S. Skogestad. "Constrained Neural Networks for Approximate Nonlinear Model Predictive Control". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 295–300.

[192] L. Furieri, C. L. Galimberti, and G. Ferrari-Trecate. "Neural system level synthesis: Learning over all stabilizing policies for nonlinear systems". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE. 2022, pp. 2765–2770.

[193] M. Revay, R. Wang, and I. R. Manchester. "Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness". In: *arXiv preprint arXiv:2104.05942* (2021).

[194] M. Marchi, J. Bunton, B. Gharesifard, and P. Tabuada. "Safety and stability guarantees for control loops with deep learning perception". In: *IEEE Control Systems Letters* 6 (2021), pp. 1286–1291.

[195] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake. "Lyapunov-stable neural-network control". In: *arXiv preprint arXiv:2109.14152* (2021).

[196] Y.-C. Chang, N. Roohi, and S. Gao. "Neural Lyapunov Control". In: *Advances in neural information processing systems* 32 (2019).

[197] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* (2022).

[198] R. P. Mason and A. Papachristodoulou. "Chordal sparsity, decomposing SDPs and the Lyapunov equation". In: *Proceedings of the American Control Conference* (2014), pp. 531–537.

[199] R. P. M. Yang Zheng and A. Papachristodoulou. "Scalable Design of Structured Controllers Using Chordal Decomposition". In: *IEEE Transactions on Automatic Control* 63.3 (2018), pp. 752–767.

[200] J. Lan, Y. Zheng, and A. Lomuscio. "Tight neural network verification via semidefinite relaxations and linear reformulations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 7. 2022, pp. 7272–7280.

[201] A. Xue, L. Lindemann, and R. Alur. "Parametric chordal sparsity for SDP-based neural network verification". In: *arXiv preprint arXiv:2206.03482* (2022).

[202] Y. Sun, M. S. Andersen, and L. Vandenberghe. "Decomposition in conic optimization with partially separable structure". In: *SIAM Journal on Optimization* 24.2 (2014), pp. 873–897.

[203] G. Vardi, G. Yehudai, and O. Shamir. "Width is less important than depth in ReLU neural networks". In: *Conference on Learning Theory*. PMLR. 2022, pp. 1249–1281.

[204] K. G. Murty and S. N. Kabadi. *Some NP-complete problems in quadratic and nonlinear programming*. Tech. rep. 1985.

[205] Y. Zheng, G. Fantuzzi, and A. Papachristodoulou. "Chordal and factor-width decompositions for scalable semidefinite and polynomial optimization". In: *Annual Reviews in Control* (2021).

[206] V. Magron and J. Wang. "Sparse Polynomial Optimization: Theory and Practice". In: *arXiv preprint arXiv:2208.11158* (2022).

[207] J. Wang, V. Magron, J. B. Lasserre, and N. H. A. Mai. "CS-TSSOS: Correlative and term sparsity for large-scale polynomial optimization". In: *arXiv preprint arXiv:2005.02828* (2020).

[208] J. Wang, V. Magron, and J.-B. Lasserre. "Chordal-TSSOS: a moment-SOS hierarchy that exploits term sparsity with chordal extension". In: *SIAM Journal on Optimization* 31.1 (2021), pp. 114–141.

[209] J. Lofberg. "Pre- and Post-Processing Sum-of-Squares Programs in Practice". In: *IEEE Transactions on Automatic Control* 54.5 (2009), pp. 1007–1011.

[210] F. Permenter and P. A. Parrilo. "Basis selection for SOS programs via facial reduction and polyhedral approximations". In: *53rd IEEE Conference on Decision and Control*. 2014, pp. 6615–6620.

[211] V. Magron and J. Wang. "TSSOS: a Julia library to exploit sparsity for large-scale polynomial optimization". In: *arXiv preprint arXiv:2103.00915* (2021).

[212] A. MOSEK. *MOSEK Optimization Toolbox for MATLAB. Release 9.2. 40*. 2021.

[213] S. H. Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[214] H. K. Khalil. *Nonlinear control*. Vol. 406. Pearson New York, 2015.

[215] J.-J. E. Slotine, W. Li, et al. *Applied nonlinear control*. Vol. 199. 1. Prentice hall Englewood Cliffs, NJ, 1991.

[216] A. Papachristodoulou and S. Prajna. "On the construction of Lyapunov functions using the sum of squares decomposition". In: *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.* Vol. 3. IEEE. 2002, pp. 3482–3487.

[217] J. Anderson and A. Papachristodoulou. "Robust nonlinear stability and performance analysis of an F/A-18 aircraft model using sum of squares programming". In: *International Journal of Robust and Nonlinear Control* 23.10 (2013), pp. 1099–1114.

[218] A. Papachristodoulou and S. Prajna. "A tutorial on sum of squares techniques for systems analysis". In: *Proceedings of the 2005, American Control Conference, 2005.* IEEE. 2005, pp. 2686–2700.

[219] N. Boullé, Y. Nakatsukasa, and A. Townsend. "Rational neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 14243–14253.

[220] L. Hou, D. Samaras, T. Kurc, Y. Gao, and J. Saltz. "Convnets with smooth adaptive activation functions for regression". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 430–439.

[221] Y. Berradi. "Symmetric power activation functions for deep neural networks". In: *Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications*. 2018, pp. 1–6.

[222] E. López-Rubio, F. Ortega-Zamorano, E. Domínguez, and J. Muñoz-Pérez. "Piecewise polynomial activation functions for feedforward neural networks". In: *Neural Processing Letters* 50 (2019), pp. 121–147.

[223] F. Farhadi, V. P. Nia, and A. Lodi. "Activation adaptation in neural networks". In: *arXiv preprint arXiv:1901.09849* (2019).

[224] B. Li, S. Tang, and H. Yu. "PowerNet: Efficient representations of polynomials and smooth functions by deep neural networks with rectified power units". In: *arXiv preprint arXiv:1909.05136* (2019).

[225] Z. Xu, F. Yu, J. Xiong, and X. Chen. "QuadraLib: A Performant Quadratic Neural Network Library for Architecture Optimization and Design Exploration". In: *Proceedings of Machine Learning and Systems* 4 (2022), pp. 503–514.

[226] I. Bukovsky, N. Homma, L. Smetana, R. Rodriguez, M. Mironovova, and S. Vrana. "Quadratic neural unit is a good compromise between linear models and neural networks for industrial applications". In: *9th IEEE International Conference on Cognitive Informatics (ICCI'10)*. IEEE. 2010, pp. 556–560.

[227] A. Molina, P. Schramowski, and K. Kersting. "Padé Activation Units: End-to-end Learning of Flexible Activation Functions in Deep Networks". In: *arXiv preprint arXiv:1907.06732* (2019).

[228] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag. "What is the state of neural network pruning?" In: *arXiv preprint arXiv:2003.03033* (2020).

[229] R. Drummond, M. C. Turner, and S. R. Duncan. "Reduced-Order Neural Network Synthesis with Robustness Guarantees". In: *arXiv preprint arXiv:2102.09284* (2021).

[230] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[231] C.-Y. Ko, Z. Lyu, L. Weng, L. Daniel, N. Wong, and D. Lin. "POPQORN: Quantifying robustness of recurrent neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3468–3477.

[232] D. Gramlich, P. Pauli, C. W. Scherer, F. Allgöwer, and C. Ebenbauer. "Convolutional Neural Networks as 2-D systems". In: *arXiv preprint arXiv:2303.03042* (2023).

[233] Z. Shi, H. Zhang, K.-W. Chang, M. Huang, and C.-J. Hsieh. "Robustness verification for transformers". In: *arXiv preprint arXiv:2002.06622* (2020).

[234] G. E. Dullerud and F. Paganini. *A course in robust control theory: a convex approach*. Vol. 36. Springer Science & Business Media, 2013.

[235] F. Zhang. *The Schur complement and its applications*. Vol. 4. Springer Science & Business Media, 2006.

[236] J. Wang, V. Magron, and J.-B. Lasserre. "TSSOS: A moment-SOS hierarchy that exploits term sparsity". In: *SIAM Journal on Optimization* 31.1 (2021), pp. 30–58.

... it will all be over.