Automatica 157 (2023) 111233

Contents lists available at ScienceDirect

Automatica

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

Sparse polynomial optimisation for neural network verification☆

Matthew Newton, Antonis Papachristodoulou *

Department of Engineering Science, University of Oxford, Parks Road, Oxford, OX1 3PJ, UK

ARTICLE INFO

Article history:
Received 27 June 2022
Received in revised form 8 May 2023
Accepted 31 May 2023
Available online 19 August 2023

Keywords:
Neural networks
Sparse polynomial optimisation
Semi-algebraic sets

ABSTRACT

The prevalence of neural networks in applications is expanding at an increasing rate. It is becoming clear that providing robust guarantees on systems that use neural networks is very important, especially in safety-critical applications. A trained neural network's sensitivity to adversarial attacks is one of its greatest shortcomings. To provide robust guarantees, one popular method that has seen success is to describe and bound the activation functions in the neural network using equality and inequality constraints. However, there are numerous ways to form these bounds, providing a trade-off between conservativeness and complexity. We approach the problem from a different perspective, using sparse polynomial optimisation theory and Positivstellensatz, a key result in real algebraic geometry. The former exploits the natural cascading structure of the neural network using ideas from chordal sparsity while the latter tests the emptiness of a semi-algebraic set using algebra, to provide tight bounds. We show that bounds can be tightened significantly, whilst the computational time remains reasonable. We compare the solve times of different solvers and show how accuracy can be improved at the expense of increased computation time. In particular, we show that with the sparse polynomial framework the solve time and accuracy can be improved over other methods for neural network verification, for networks with ReLU, sigmoid and tanh activation functions.

1. Introduction

The field of machine learning has seen a huge resurgence of interest over the past decade. In particular, the development of Alexnet (Krizhevsky, Sutskever, & Hinton, 2012) and Resnet (He, Zhang, Ren, & Sun, 2016) showed a step increase in the capacity of neural networks (NNs) to perform complex tasks that were once thought to be impossible to compute by machine. The increase in computational power available and the abundance of big data, has led to an increase in industrial applications of NNs and their prevalence is expanding. Examples of these include but are not limited to image recognition, weather prediction and natural language processing (Brown, Mann, Ryder, Subbiah, et al., 2020; Zhang & Ma, 2012).

With the success of NNs in general application areas, the transition into safety-critical applications is an important consideration, such as autonomous vehicle technology. However, before this can be achieved, the research community along with industry must overcome one of the biggest shortcomings of this new technology, which is the NN's sensitivity to adversarial inputs: large changes in the output set can be caused by relatively small changes in the input set. There has been a considerable effort to improve our understanding of NNs and to provide certificates on such systems. However, there are still significant hurdles to surmount for their wide-spread use in safety-critical applications. The NN verification problem aims to provide a confidence measure of how uncertainty in the input set maps to the output set. In general, the NN verification problem involves checking whether the output of the NN will classify the same label given some perturbation on the input set. However, the term 'verification' has been used to describe the general class of problems that analyse the input–output properties of NNs.

In the field of control theory, there exists work in the area of NNs dating back to the 1990s (Miller, Werbos, & Sutton, 1995). There has been a sparked interest in this area with the emergence of the parallel field of reinforcement learning: deep reinforcement learning has harnessed the power of NNs to provide a decision making agent to greatly outperform humans in many complex tasks, such as the video game Dota 2 and the board game Go (Silver, Huang, et al., 2016). Bounds to quantify their safety

☆ This work was supported by EPSRC, United Kingdom grants EP/L015897/1 (to M. Newton) and EP/M002454/1 (to A. Papachristodoulou). For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission. The material in this paper was partially presented at the Learning for Decision and Control (L4DC) conference, 2021 and the 60th IEEE Conference on Decision and Control, December 13–15, 2021, Austin, Texas, USA. This paper was recommended for publication in revised form by Associate Editor Luca Schenato under the direction of Editor Christos G. Cassandras.

* Corresponding author.
E-mail addresses: matthew.newton@eng.ox.ac.uk (M. Newton), antonis@eng.ox.ac.uk (A. Papachristodoulou).

https://doi.org/10.1016/j.automatica.2023.111233

have been developed, however are often overly conservative and do not quantify the performance of the algorithm sufficiently (Sutton & Barto, 2018). The success of NNs and reinforcement learning, combined with new advancements in robust control, motivates work at their intersection. There are many methods to compute robust guarantees on an NN to verify its properties. There have been competitions such as VNN-COMP (Müller, Brix, et al., 2022) to provide an objective comparison between state-of-the-art methods for NN verification, this was won by α-β-CROWN (Wang, Zhang, Xu and et al., 2021). One method is to use Satisfiability Modulo Theory to quantify the safety of an NN (Huang, Kwiatkowska, Wang, & Wu, 2017; Katz, Barrett, Dill, Julian, & Kochenderfer, 2017). Another approach focuses on finding the Lipschitz constant for the NN (Fazlyab, Robey, Hassani, Morari, & Pappas, 2019), which can then be used to train the NN to be robust (Pauli, Koch, Berberich, Kohler, & Allgöwer, 2022). Lipschitz bounds can be computed in a scalable way; both Latorre, Rolland, and Cevher (2020) and Chen, Lasserre, Magron, and Pauwels (2020) use sparse polynomial optimisation to achieve this.

A common approach to provide robust guarantees on an NN is to place bounds on the non-linear activation functions (Salman, Yang, Zhang, Hsieh, & Zhang, 2019): this is the idea that is used in this paper. Due to the large number of formulations, and choices of activation functions, there is a sizeable literature showing this method to be successful, with each result finding tighter and more efficient bounds on the properties of the NN. The simplest methods are zero-th order methods such as interval bound propagation (Gowal, Dvijotham, Stanforth, et al., 2018), that computes the worst-case scenarios out of each layer in the network. An abstract interpretation-based approach for NN verification is introduced in Singh, Ganvir, Püschel, and Vechev (2019). Frameworks that provide linear bounds on the activation functions will result in linear programs, which can be solved using convex optimisation (Bunel, Turkaslan, Torr, Kohli, & Mudigonda, 2018; Ehlers, 2017). Furthermore, researchers have found other methods to improve the accuracy and scalability of the problem. A scalable approach that uses this linear programming framework is DeepSplit that uses an operator splitting method to find the bounds (Chen, Wong, Kolter and Fazlyab, 2021).

To improve the tightness of the linear relaxations, Raghunathan, Steinhardt, and Liang (2018) introduces semidefinite relaxations for the certification of the robustness properties of NNs, this is built upon in Brown, Schmerling, Azizan, and Pavone (2022) through completely positive programming. Another semidefinite approach that provides tight bounds on the NN is built on the formulation of quadratic constraints (Fazlyab, Morari, & Pappas, 2020). However, the semidefinite programming (SDP) approach has the drawback of scaling worse than linear programs. Newton and Papachristodoulou (2021a) improves the scalability of this framework by exploiting the sparsity pattern which is shown to match the intrinsic cascading structure of the NN. This structure has also been utilised in Batten, Kouvaros, Lomuscio, and Zheng (2021), where linear cuts are used to relax the SDP formulation. These SDP approaches have been shown to analyse the robustness of Monotone Deep Equilibrium Models (Chen, Lasserre, Magron and Pauwels, 2021).

1.1. Our contribution

Previous work 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. Our contributions are:

- We formulate the neural network verification problem as a set of equality and inequality constraints, which results in a semi-algebraic set. We then use a theorem from real algebraic geometry called the Positivstellensatz, to certify the emptiness of this set using algebra.
- We use this algebraic test to find bounds on the neural network's output using Sum of Squares and semidefinite programming methods. By modifying the order of the algebraic condition, we demonstrate how we can trade off solution accuracy and computational complexity. Selecting higher order algebraic conditions leads to richer set construction, which generates at least as good – and many times better – tests and tighter bounds on the neural network's output.
- We exploit the 'natural cascading' sparsity structure in a neural network to formulate a structured algebraic condition, thereby reducing the size of the resulting semidefinite program. The key idea that underpins this method is a decomposition theorem that exploits the link between chordal graphs and positive semidefinite matrices, showing that a large positive semidefinite constraint can be split up into smaller positive semidefinite constraints. We analyse the optimisation framework using theory from sparse polynomial optimisation and show how the variables are separated into different constraints.
- We implement the optimisation problem and show through numerous examples that this method improves both the computational time and accuracy. The examples we show have a varying number of nodes and layers for ReLU, sigmoid and tanh activation functions. Previously, neural networks of this size have not been verified to this degree of accuracy.

In Section 2 we provide an overview of the NN verification problem and describe how the input–output relationships can be represented as a series of inequality and equality constraints. The Positivstellensatz is introduced in Section 3 and then how it can be used in the NN verification problem is described. The theory behind sparse polynomial optimisation and the link to Positivstellensatz is delineated in Section 4. The theory is then applied to the specific NN verification problem in Section 5. In Section 6 the results from various experiments are presented and analysed. The paper is then concluded in Section 7, outlining plans for future work.

2. Neural network verification

A multi-layer feed-forward neural network (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. Here $\mathbb{R}^n$ denotes the n-dimensional real vectors and $\mathbb{R}^{m\times n}$ denotes the $m \times n$-dimensional real matrices. 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 $\mathcal{U}$. 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

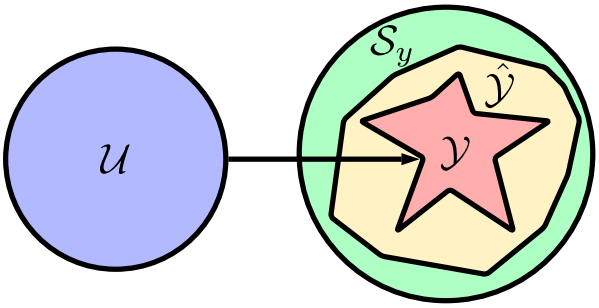


Fig. 1. Diagram showing the neural network verification problem. $\mathcal{U}$ represents the input set, $\mathcal{Y}$ represents the output set and $\hat{\mathcal{Y}}$ is the approximation of this set. The safe set of outputs is denoted as $\mathcal{S}_y$.

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 Fig. 1.

2.1. 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:

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

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 $(k+1)$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 (Nwankpa, Ijomah, Gachagan, & Marshall, 2018).

2.2. Problem overview

We now describe how constraints on the non-linear activation functions are created. How these constraints are used subsequently to form an optimisation problem to verify an NN is presented in Section 3. The constraints can be split into three main categories, by considering the input, output and hidden layer constraints separately.

We assume that the *input constraints* are bounded by a hyper-rectangle defined by $\mathcal{U} = \{u \in \mathbb{R}^{n_u} \mid \underline{u} \le u \le \overline{u}\}$. Therefore, the input constraints can be written as

$$g_{in}^1 = u - \underline{u} \ge 0, \ g_{in}^2 = -u + \overline{u} \ge 0.$$

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

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

where $c_m \in \mathbb{R}^{n_y}$ and $d_m \in \mathbb{R}$ are given parameters of the polytope.

In the optimisation framework the faces of the polytope are considered separately, where we attempt to obtain bounds on the elements d_m to reduce the volume of the polytope. Therefore, the search for bounds on the safe output set can be split into M optimisation programs. The output constraints contain a decision variable that can be minimised in the optimisation program such that

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

where γ_m is the decision variable to be optimised and m denotes the mth optimisation program. A polytope is a common choice for the output constraints since it is composed of linear functions, meaning that the objective of the optimisation problem is also 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.

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. These constraints are formed through many means: popular methods include sector, slope and box constraints, however there is no limitation of how these can be expressed. It is also possible to compute a conservative approximation of the bounds using an efficient pre-processing step known as interval bound propagation (IBP) (Gowal et al., 2018). IBP is a zero-th order method, which uses interval arithmetic to find the minimum and maximum bounds on the activation function, $(\underline{\phi}, \overline{\phi})$ such that

$$\phi - \underline{\phi} \ge 0, \ -\phi + \overline{\phi} \ge 0. \tag{1}$$

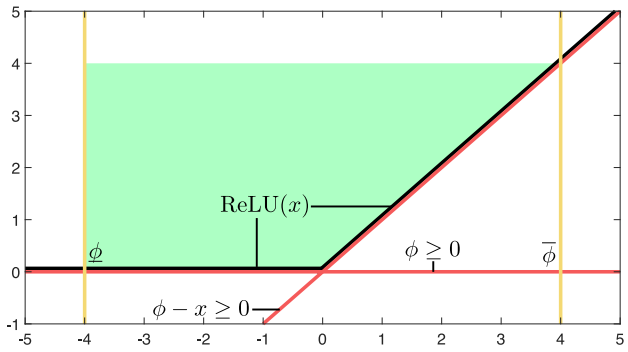
The pre-activation values from IBP are denoted as $(\underline{x}, \overline{x})$. There are other efficient methods that can be used in this pre-processing step such as CROWN (Wang, Zhang et al., 2021) that work for general activation functions and can provide better results, but take longer to compute. 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. A common trait of an activation function is that it is monotonically increasing—a function that has this property can often be bounded by a sector constraint.

Definition 1. Consider the non-linear activation function $\phi(x)$ with $\phi(0) = \lambda$. A sector constraint states that $\phi(x)$ lies in the sector $[\alpha, \beta]$ $(\alpha \le \beta < \infty)$ if the following condition holds

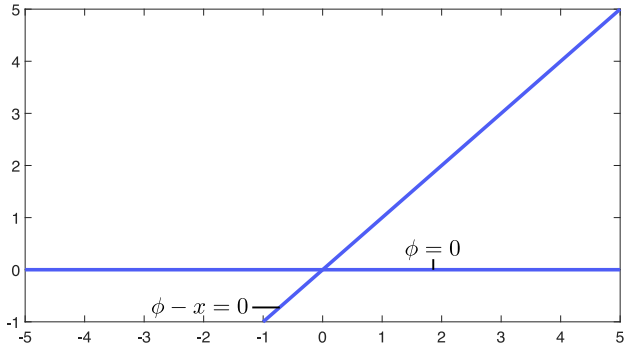
$$(\phi(x) - (\alpha x + \lambda))((\beta x + \lambda) - \phi(x)) \ge 0, \ \forall\, x \in \mathbb{R}.$$

Note that a sector constraint only considers the relationship between the activation function's output and input. We show in Section 5 that this point is important to preserve the sparsity property in the algorithm formulation. Another set of bounds that can be used are known as slope constraints (Fazlyab et al., 2020), which considers the slope of two activation functions from different layers. However, they will not be considered in this paper due to their lack of scalability—as the number of neurons in the network increases, the number of constraints increases as $\binom{n}{2}$. Additionally, 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 Section 5.

We now explain how the sector constraints can be represented for different activation functions.

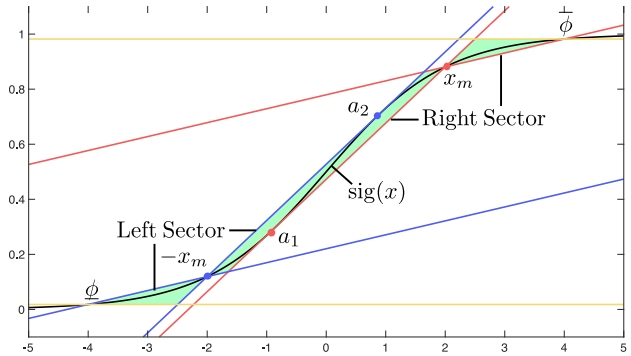


(a) The inequality constraints in (2).

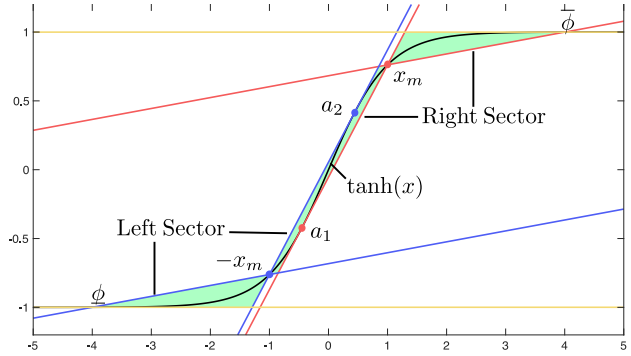


(b) The equality constraints in (2).

Fig. 2. Plot showing the constraints in (2) that bound the ReLU function (black). The yellow lines represent the constraints from the IBP values. The red lines represents the inequality constraints, which bound the green shaded region. The blue lines represents the quadratic equality constraint. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Sigmoid Activation Function



(b) Tanh Activation Function

Fig. 3. Two intersecting sector constraints that bound the activation function (black). The right and left sectors are shown in red and blue respectively. The green shaded area represents the region defined by the intersecting sector constraints. The point x_m is a hyper-parameter to be chosen and defines where the two lines intersect to form the right sector. The same process is repeated with the left sector with mid point of $-x_m$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Example 1 (*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 (Raghunathan et al., 2018) such that

$$\phi \geq 0,\ \phi - x \geq 0,\ \phi(\phi - x) = 0. \tag{2}$$

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 Figs. 2(a)–2(b).

Example 2 (*Sigmoid and Tanh Functions*)**.** The sigmoid function is given by

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

and it can be bounded by a sector constraint such that

$$(\phi - 0.5)(0.25x + 0.5 - \phi) \geq 0. \tag{3}$$

However, this bound is very conservative as there is a large uncertainty in the value of the function. As shown in Newton and Papachristodoulou (2021b) this can be tightened drastically by using two sector constraints that are carefully positioned as in Fig. 3(a). The right sector at an arbitrary point on the sigmoid curve $(x_m, \phi(x_m))$. 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 gradient of the upper line can be found with the formula

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

which can be solved numerically. The lower line of the sector is the line passing through the points $\phi(x_m)$ and $\overline{\phi}$ and is given by the equation

$$y(x) = \frac{\phi(x_m) - \overline{\phi}}{x_m - \overline{x}}(x - \overline{x}) + \overline{\phi}. \tag{4}$$

The tanh function is given by

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

The process of computing the sectors is the same as the sigmoid function, they are shown visually in Fig. 3(b). Having two sectors instead of one allows one to capture the point of inflection of the sigmoid and tanh functions. However, the drawback of this is that there will be twice the number of inequality constraints which will make the optimisation problem more expensive.

3. Problem formulation

As outlined in the introduction, previous literature used a particular type of optimisation framework to verify the NN. 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. To achieve this we use a theorem from real algebraic geometry known as the Positivstellensatz (Psatz) (Stengle, 1974), that uses an algebraic condition to test the emptiness of a semi-algebraic set. We now describe this theorem briefly, and show how it can be applied to the NN verification problem.

3.1. Positivstellensatz

Consider a semi-algebraic set

$$S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0,\ \forall\, i = 1, \ldots, p,\ j = 1, \ldots, q\} \tag{5}$$

with $g_i, h_j \in \mathbb{R}[x]$ polynomials with real coefficients.

Definition 2. 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'.

Definition 3. 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 4. Given $g_1, \ldots, g_p \in \mathbb{R}[x]$, the *cone* generated by g_i's is

$$\text{cone}\{g_1, \ldots, g_p\} = \left\{ s_0 + \sum_{i=1}^{p} s_i G_i \mid s_i \in \Sigma[x],\ G_i \in \mathcal{M}(g_1, \ldots, g_p) \right\}. \tag{6}$$

Definition 5. Given $h_1, \ldots, h_q \in \mathbb{R}[x]$, the *ideal* generated by h_k's is

$$\text{ideal}\{h_1, \ldots, h_q\} = \left\{ \sum_{j=1}^{q} t_j h_j \mid t_j \in \mathbb{R}[x] \right\}. \tag{7}$$

Theorem 1 (*Positivstellensatz*)**.** *Given the semi-algebraic set S in (5), the following are equivalent:*

(1) The set S is empty.
(2) There exist $s_i \in \Sigma[x]$ in (6) and $t_j \in \mathbb{R}[x]$ in (7) such that

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

Theorem 1 links the emptiness of a semi-algebraic set with an algebraic test. To create a convex test based on this theorem for computational purposes, one can use a representation of the function f such that if

$$f = 1 + \sum_{j}^{q} t_j h_j + s_0 + \sum_{i}^{p} s_i g_i \tag{8}$$

then $f(x) > 0,\ \forall x \in S$, where $s_i \in \Sigma[x]$ and $t_j \in \mathbb{R}[x]$. This follows directly by considering each term on the right hand side for $x \in S$. This is the formulation that we use in this paper. In our case we wish to show that $g_{out}^m \geq 0$ on S, and therefore we solve the following optimisation problem:

$$\begin{aligned} \min\quad & \gamma_m, \\ \text{s.t.}\quad & -c_m^T y + \gamma_m - \sum_{j}^{q} t_j h_j - \sum_{i}^{p} s_i g_i \in \Sigma[x], \\ & s_i \in \Sigma[x],\ \forall\, i = 1, \ldots, p, \\ & t_j \in \mathbb{R}[x],\ \forall\, j = 1, \ldots, q, \end{aligned} \tag{9}$$

where $h_j(x)$ and $g_i(x)$ are the equality and inequality constraints describing S in (5).

3.2. Sum of squares and semidefinite programming

We now discuss how to solve optimisation problem (9) using Sum of Squares and semidefinite programming. We require to index the optimisation variables such that sparsity can be exploited as described in Section 4. 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; the set of n-dimensional integers is denoted by $\mathbb{N}^n$. 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. Any polynomial f can be written as $f = \sum_{\beta \in \mathbb{N}_d^n} f_\beta x^\beta$ 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. $\mathbb{S}^n$ and $\mathbb{S}_+^n$ are the sets of $n \times n$ symmetric matrices and positive semidefinite matrices respectively. Define the summation operation on $\mathbb{B}$ as

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

A polynomial f is SOS if and only if it can be written in what is referred to as a Gram matrix representation such that $f = (x^{\mathbb{B}})^T Q x^{\mathbb{B}}$. 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 rewrite the Gram 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

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

This means that checking the SOS condition is equivalent to solving a semidefinite program (SDP). Parsers such as SOSTOOLS (Papachristodoulou et al., 2013) in MATLAB or the SumOfSquares.jl package (Legat, Coey, Deits, Huchette, & Perry, 2017) in Julia can be used to formulate this SDP.

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 $\binom{n+d}{d} \times \binom{n+d}{d}$. In (9) if we choose a higher degree for the multipliers s_i, t_j etc., we can obtain a series of nested set emptiness tests of increasing complexity and non-decreasing accuracy. 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 Section 4.

4. Sparse polynomial optimisation

Looking at (9), one of the issues with using the SOS/SDP framework is that the computational time becomes too large as

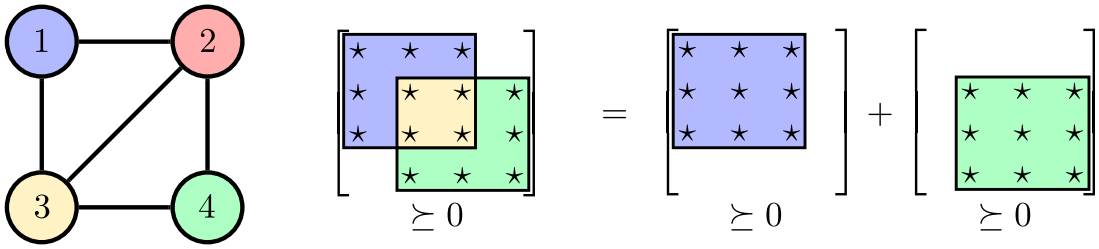


Fig. 4. Shows how a matrix can be represented by a graph and then how a positive semidefinite constraint can be split into smaller positive semidefinite constraints using the properties of the chordal graph. The edge set of the chordal graph is $\mathcal{E} = \{\{1,2\},\{2,3\},\{2,4\},\{3,4\}\}$ and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

the size of the NN gets bigger. 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 Newton and Papachristodoulou (2021a) (for formulations that do not use Psatz) it is possible to use the natural cascading structure of the NN to improve the solve times drastically. This is achieved by using theory from chordal graphs and the ways in which they can be used to exploit the sparsity in positive semidefinite matrices. We now provide an overview of these methods to present how they can be used with the Psatz framework.

4.1. Chordal graphs and sparse matrix decomposition

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 chord that lies on the graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ is an edge that joins two non-adjacent nodes in a cycle.

Definition 6 (*Kakimura, 2010*)**.** 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.

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 maximal clique is a clique that is not a subset of any other clique. Chordal graphs are useful as they can be decomposed into their maximal cliques (Vandenberghe & Andersen, 2015). 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

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

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, so can a matrix $X \in \mathbb{S}^n_+(\mathcal{E},0)$. This is shown visually in Fig. 4.

Theorem 2 (*Agler, Helton, McCullough, & Rodman, 1988*)**.** *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 $|\mathcal{C}_k|$ is the number of vertices in that clique and

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

4.2. Sparsity and sum of squares

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

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

There are many types of sparsity that use the structure of $\mathbb{A}$, such as term and correlative sparsity (Zheng, Fantuzzi, & Papachristodoulou, 2021). Moreover, there are many different hierarchies to define the monomial basis that have been proposed. Most notable are Term Sparse SOS (TSSOS), Chordal-TSSOS (CTSSOS) and Correlative Sparsity-TSSOS (CS-TSSOS) (Wang, Magron and Lasserre, 2021; Wang, Magron, Lasserre, & Mai, 2020).

4.3. 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-algebraic 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 (8) 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

$$f(x) = \sum_{i=0}^{m} g_i(x)(x^{\mathbb{B}_i})^T Q_i x^{\mathbb{B}_i}, \quad Q_i \in \mathbb{S}_+^{|\mathbb{B}_i|}. \tag{10}$$

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. 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

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

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 (10) then we cannot exploit the sparsity in f or g_i. To proceed to preserve the sparsity pattern, we create what is referred to as a joint correlative sparsity graph of the polynomials $f, g_1, \ldots, g_m$. This graph has n vertices where 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 $\mathcal{J}_1, \ldots, \mathcal{J}_t$.

5. 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.

5.1. Example: Three layer, single node neural network

Consider a single input/output NN with three layers and a single node in each layer with a ReLU activation function. The equations for this NN are:

$$x_0 = u,\ x_1 = \phi(W^0 x_0 + b^0),\ x_2 = \phi(W^1 x_1 + b^1),$$
$$x_3 = \phi(W^2 x_2 + b^2),\ y = W^3 x_3 + b^3.$$

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

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

For now we will only consider the ReLU activation function in the hidden layers. We will use the quadratic constraints as outlined in (2) such that

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

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 Fig. 5. This graph is chordal and has four cliques, which are written as:

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

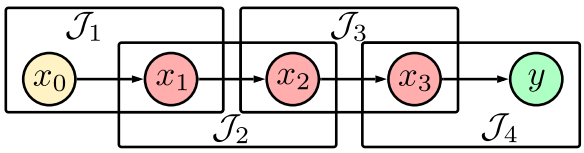


Fig. 5. Diagram of a neural network showing how the cliques are formed for the example in Section 5.1.

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 $f = g^m_{out} = \gamma_m - c^T y$ 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. 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 $h_{1,1}$. 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

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

where $\text{nnz}(\beta) := \{\beta \in \mathbb{B} : \beta_i \neq 0,\ \forall i = 1, \ldots, |\mathbb{B}|\}$ (Zheng et al., 2021). 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

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

Each $Q_{i,j}$ will then only be a function of the variables in the ith and $(i-1)$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.

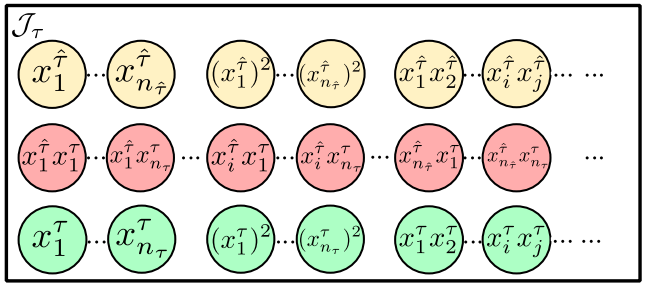


Fig. 6. Diagram showing the variables that are contained in a generic clique $\mathcal{J}_\tau$. The yellow, green and red nodes are the monomials from layer $\hat{\tau}$, the next layer $\tau = \hat{\tau} + 1$ and overlapping terms respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.2. General neural network

To proceed from the simple example in Section 5.1, 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:

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

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 $(i-1)$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 5.1 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 Figs. 6 and 7. 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.

For the verification of a general NN, the Psatz SOS condition within the optimisation problem is:

$$-c_m^T y + \gamma_m - \sum_{i=0}^{\ell} \sum_{j=1}^{n_i} \sum_{k=1}^{p_{i,j}} t_{i,j,k} h_{i,j,k} \ldots$$
$$- \sum_{i=0}^{\ell} \sum_{j=1}^{n_i} \sum_{k=1}^{q_{i,j}} s_{i,j,k} g_{i,j,k} \text{ is (CS−)TSSOS}, \tag{11}$$

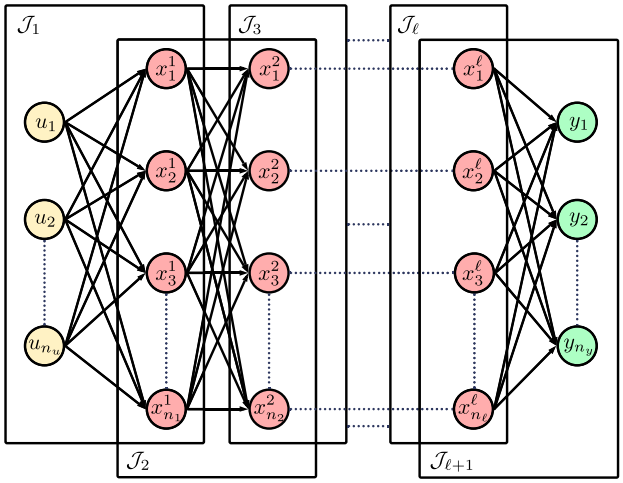


Fig. 7. Diagram of a general neural network showing how the cliques are formed. The yellow nodes are the input nodes, the red nodes are the hidden layers and the green nodes are the output nodes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$s_{i,j,k} \text{ is SOS}, \ \forall\, i = 0, \ldots, \ell, \ j = 1, \ldots, n_i, \ k = 1, \ldots, p_{i,j}$$
$$t_{i,j,k} \in \mathbb{R}[x], \ \forall\, i = 0, \ldots, \ell, \ j = 1, \ldots, n_i, \ k = 1, \ldots, q_{i,j}$$

where $p_{i,j}$ and $q_{i,j}$ are the number of equality and inequality constraints in the ith layer and jth 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.

6. Experimental 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 4-core Intel Xeon processor @3.50 GHz with 16 GB 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 with constraint (11) with the CS-TSSOS hierarchy (Wang et al., 2020) using the TSSOS Julia package (Magron & Wang, 2021), which constructs the SDP constraints and parses it to the SDP solver MOSEK (MOSEK, 2021). The code for all numerical experiments is available at https://github.com/MNewtonOX/NNSparsePsatz.

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 (Newton & Papachristodoulou, 2021b), 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 DeepSDP (Fazlyab et al., 2020) with MOSEK, which uses an SDP framework without any sparsity. We also compare our approach to another sparsity exploiting method LayerSDP (Batten et al., 2021), 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 (Wang, Zhang et al., 2021) and DeepPoly (Singh et al., 2019). In these experiments we consider NNs of less width and greater depth, since Vardi, Yehudai, and Shamir (2022) showed that depth is more important than width in the expressive power of NNs. This will also highlight the benefits of chordal decomposition.

6.1. Scalability comparison

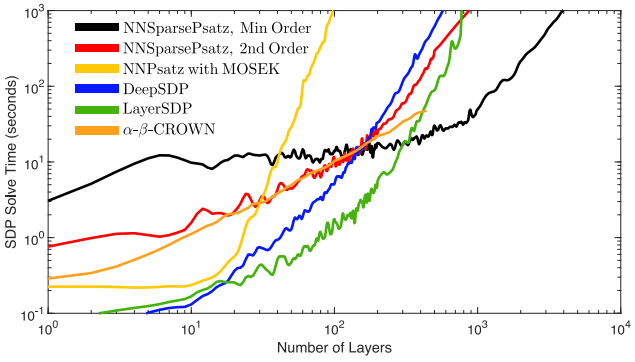
There are two main ways that we can assess the scalability of the NN verification problem. One approach is to vary the number of layers in the network and the other is to vary the number of nodes in each layer.

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 Figs. 8(a), 8(b) and 8(c) 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.

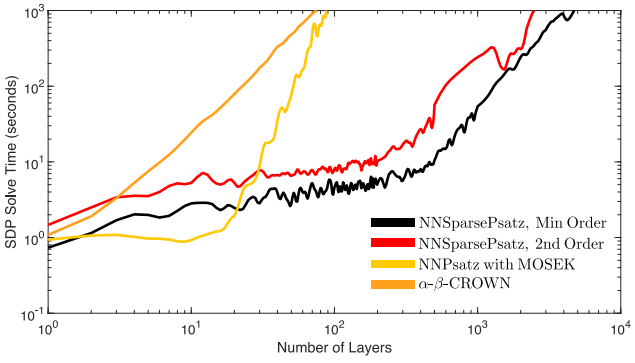
We now see what happens when we vary the size of each layer. The number of layers will be fixed to 100 and the number of nodes in each of these layers will be increased. The methods are compared for ReLU, sigmoid and tanh functions and are shown in Figs. 9(a), 9(b) and 9(c) respectively. For the ReLU 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.

6.2. Accuracy comparison

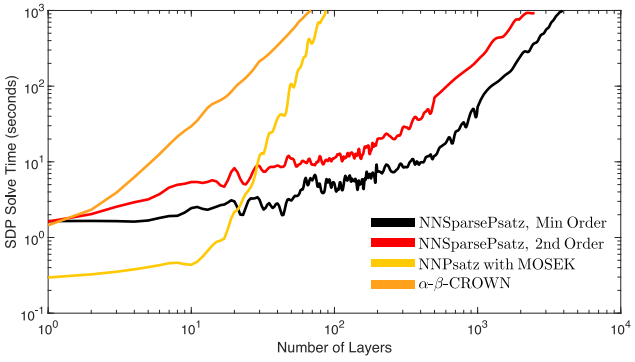
Having showed that the CS-TSSOS hierarchy can improve the scalability of the problem, we now turn our attention to the accuracy. We consider a two input/output NN with various sizes for the different activation functions. For the ReLU activation function we use a five layer NN with eight nodes in each layer. It is shown in Fig. 10(a) that the NNSparsePsatz method with second order polynomial multipliers greatly improves the tightness of the bounds on the output set such that they are near optimal. If we instead used zeroth order multipliers in NNSparsePsatz, then the accuracy is 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 688 s to solve. NNSparsePsatz with zeroth order multipliers, LayerSDP DeepSDP, α-β-CROWN and DeepPoly took 3.3 s, 15.5 s, 3.9 s, 4.7 s and 0.02 s 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.



(a) ReLU Activation Function



(b) Sigmoid Activation Function

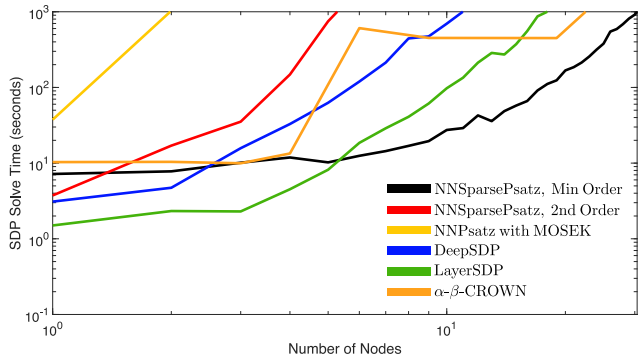


(c) Tanh Activation Function

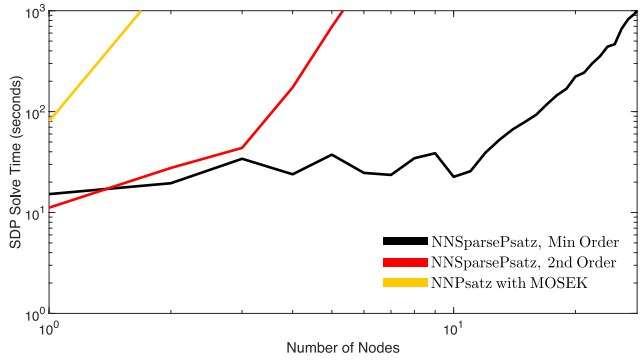
Fig. 8. Comparing the computational time of different approaches for a neural network with two nodes in each layer and a varying number of layers.

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 (i.e. constant). This approach is still able to outperform the accuracy of the state-of-the-art solvers, as shown in Fig. 10(b). The computational time of NNSparsePsatz was 71.0 s, which is in the same order of magnitude as α-β-CROWN (38.9 s) but significantly slower than DeepPoly (0.05 s), which is known to be conservative but efficient.

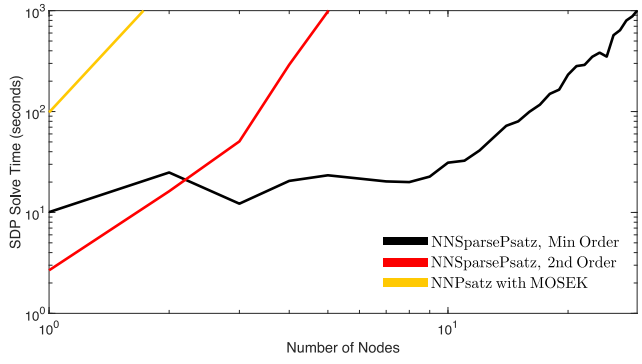
Finally we consider a deeper NN with twelve layers and five nodes in each layer with tanh activation functions. The solve times were 462 s, 2.0 s, 208 s, 0.2s for NNSparsePsatz (second



(a) ReLU Activation Function

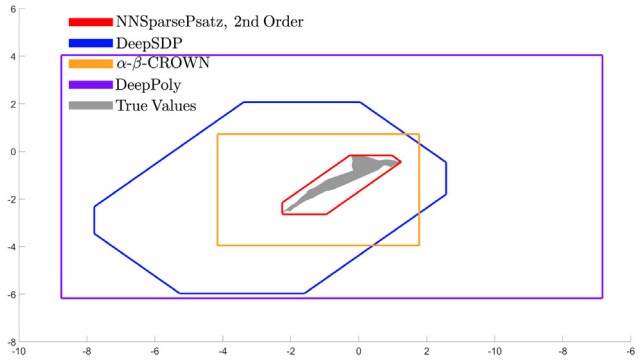


(b) Sigmoid Activation Function

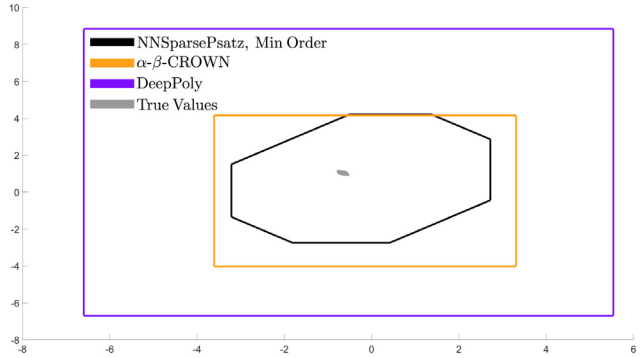


(c) Tanh Activation Function

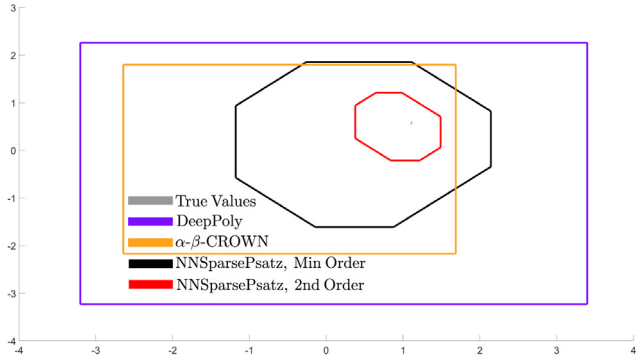
Fig. 9. Comparing the computational time of different approaches for a neural network with 100 layers and a varying number of nodes.



(a) ReLU Activation Function



(b) Sigmoid Activation Function



(c) Tanh Activation Function

Fig. 10. Comparing the accuracy of different approaches to the true values of the neural network's output.

order), NNSparsePsatz (minimum order), α-β-CROWN and DeepPoly respectively. This computational time is reflected in the output set as shown in Fig. 10(c), demonstrating how NNSparsePsatz can obtain tighter bounds against existing methods.

7. Conclusion

In this paper, we propose a framework to solve the NN verification problem, through placing bounds on the non-linear activation functions. Using a theory called 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 both improves the computation time to solve the resulting SDP against similar methods and can do so by tightening the bounds on the NN outputs.

This paper leaves scope for future work, such as using these ideas in control feedback systems and combining them with notions of stability and other robustness measures. Another way of improving the scalability of the NN verification problem is to use NN pruning (Blalock, Ortiz, Frankle, & Guttag, 2020) to reduce the size of the NN and hence make it more efficient to verify. Related works in Drummond, Turner, and Duncan (2021) synthesise a reduced order NN with robustness guarantees. Finally, it would be interesting to see the performance of the ideas from this paper when applied to other activation functions, since this is easy

to do. With a combination of these improvements, NNs can be verified more effectively in the future.

References

Agler, Jim, Helton, William, McCullough, Scott, & Rodman, Leiba (1988). Positive semidefinite matrices with a given sparsity pattern. *Linear Algebra and its Applications*, *107*, 101–149.

Batten, Ben, Kouvaros, Panagiotis, Lomuscio, Alessio, & Zheng, Yang (2021). Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *IJCAI* (pp. 2184–2190).

Blalock, Davis, Ortiz, Jose Javier Gonzalez, Frankle, Jonathan, & Guttag, John (2020). What is the state of neural network pruning? arXiv preprint arXiv:2003.03033.

Brown, Tom B, Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, et al. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

Brown, Robin, Schmerling, Edward, Azizan, Navid, & Pavone, Marco (2022). A unified view of SDP-based neural network verification through completely positive programming. arXiv preprint arXiv:2203.03034.

Bunel, Rudy R, Turkaslan, Ilker, Torr, Philip HS, Kohli, Pushmeet, & Mudigonda, Pawan Kumar (2018). A unified view of piecewise linear neural network verification. In *NeurIPS*.

Chen, Tong, Lasserre, Jean B, Magron, Victor, & Pauwels, Edouard (2020). Semialgebraic optimization for Lipschitz constants of ReLU networks. *Advances in Neural Information Processing Systems*, *33*, 19189–19200.

Chen, Tong, Lasserre, Jean B, Magron, Victor, & Pauwels, Edouard (2021). Semialgebraic representation of monotone deep equilibrium models and applications to certification. *Advances in Neural Information Processing Systems*, *34*.

Chen, Shaoru, Wong, Eric, Kolter, J. Zico, & Fazlyab, Mahyar (2021). DeepSplit: Scalable verification of deep neural networks via operator splitting. arXiv preprint arXiv:2106.09117.

Drummond, Ross, Turner, Mathew C., & Duncan, Stephen R. (2021). Reduced-order neural network synthesis with robustness guarantees. arXiv preprint arXiv:2102.09284.

Ehlers, Ruediger (2017). Formal verification of piece-wise linear feed-forward neural networks. In *International symposium on automated technology for verification and analysis* (pp. 269–286). Springer.

Fazlyab, Mahyar, Morari, Manfred, & Pappas, George J. (2020). Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*.

Fazlyab, Mahyar, Robey, Alexander, Hassani, Hamed, Morari, Manfred, & Pappas, George (2019). Efficient and accurate estimation of Lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, *32*, 11427–11438.

Gowal, Sven, Dvijotham, Krishnamurthy, Stanforth, Robert, et al. (2018). On the effectiveness of interval bound propagation for training verifiably robust models. arXiv preprint arXiv:1810.12715.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, & Sun, Jian (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Huang, Xiaowei, Kwiatkowska, Marta, Wang, Sen, & Wu, Min (2017). Safety verification of deep neural networks. In *International conference on computer aided verification* (pp. 3–29). Springer.

Kakimura, Naonori (2010). A direct proof for the matrix decomposition of chordal-structured positive semidefinite matrices. *Linear Algebra and its Applications*, *433*(4), 819–823.

Katz, Guy, Barrett, Clark, Dill, David L, Julian, Kyle, & Kochenderfer, Mykel J (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification* (pp. 97–117). Springer.

Krizhevsky, Alex, Sutskever, Ilya, & Hinton, Geoffrey E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems, vol. 25* (pp. 1097–1105).

Latorre, Fabian, Rolland, Paul, & Cevher, Volkan (2020). Lipschitz constant estimation of neural networks via sparse polynomial optimization. arXiv preprint arXiv:2004.08688.

Legat, Benoît, Coey, Chris, Deits, Robin, Huchette, Joey, & Perry, Amelia (2017). Sum-of-squares optimization in Julia. In *The first annual JuMP-dev workshop*.

Magron, Victor, & Wang, Jie (2021). TSSOS: a Julia library to exploit sparsity for large-scale polynomial optimization. arXiv preprint arXiv:2103.00915.

Miller, W. Thomas, Werbos, Paul J., & Sutton, Richard S. (1995). *Neural networks for control*. MIT Press.

MOSEK, ApS (2021). MOSEK optimization toolbox for MATLAB. Release 9.2. 40.

Müller, Mark Niklas, Brix, Christopher, et al. (2022). The third international verification of neural networks competition (VNN-COMP 2022): Summary and results. arXiv preprint arXiv:2212.10376.

Newton, Matthew, & Papachristodoulou, Antonis (2021a). Exploiting sparsity for neural network verification. In *Learning for dynamics and control* (pp. 715–727). PMLR.

Newton, Matthew, & Papachristodoulou, Antonis (2021b). Neural network verification using polynomial optimisation. In *Proceedings of the 60th conference on decision and control*.

Nwankpa, Chigozie, Ijomah, Winifred, Gachagan, Anthony, & Marshall, Stephen (2018). Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378.

Papachristodoulou, A., Anderson, J., Valmorbida, G., Prajna, S., Seiler, P., & Parrilo, P. A. (2013). SOSTOOLS: Sum of squares optimization toolbox for MATLAB. http://arxiv.org/abs/1310.4716.

Pauli, Patricia, Koch, Anne, Berberich, Julian, Kohler, Paul, & Allgöwer, Frank (2022). Training robust neural networks using Lipschitz bounds. *IEEE Control Systems Letters*, *6*, 121–126.

Raghunathan, Aditi, Steinhardt, Jacob, & Liang, Percy (2018). Semidefinite relaxations for certifying robustness to adversarial examples. arXiv preprint arXiv:1811.01057.

Salman, Hadi, Yang, Greg, Zhang, Huan, Hsieh, Cho-Jui, & Zhang, Pengchuan (2019). A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems*, *32*, 9835–9846.

Silver, David, Huang, Aja, et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–503.

Singh, Gagandeep, Ganvir, Rupanshu, Püschel, Markus, & Vechev, Martin (2019). Beyond the single neuron convex barrier for neural network certification. In *Advances in neural information processing systems, vol. 32*.

Stengle, Gilbert (1974). A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, *207*(2), 87–97.

Sutton, Richard S., & Barto, Andrew G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Vandenberghe, Lieven, & Andersen, Martin S. (2015). Chordal graphs and semidefinite optimization.

Vardi, Gal, Yehudai, Gilad, & Shamir, Ohad (2022). Width is less important than depth in ReLU neural networks. In *Conference on learning theory* (pp. 1249–1281). PMLR.

Wang, Jie, Magron, Victor, & Lasserre, Jean-Bernard (2021). Chordal-TSSOS: a moment-SOS hierarchy that exploits term sparsity with chordal extension. *SIAM Journal on Optimization*, *31*(1), 114–141.

Wang, Jie, Magron, Victor, Lasserre, Jean B, & Mai, Ngoc Hoang Anh (2020). CS-TSSOS: Correlative and term sparsity for large-scale polynomial optimization. arXiv preprint arXiv:2005.02828.

Wang, Shiqi, Zhang, Huan, Xu, Kaidi, et al. (2021). Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. arXiv preprint arXiv:2103.06624.

Zhang, Cha, & Ma, Yunqian (2012). *Ensemble machine learning: Methods and applications*. Springer.

Zheng, Yang, Fantuzzi, Giovanni, & Papachristodoulou, Antonis (2021). Chordal and factor-width decompositions for scalable semidefinite and polynomial optimization. *Annual Reviews in Control*.

Matthew Newton received an M.Eng. (M.A.) degree in Engineering Science from the University of Oxford, UK. During his degree he specialised in Control Theory, Machine Learning, Optimisation and Dynamical Systems. He is currently in the final stages of a D.Phil. (Ph.D.) in the Department of Engineering Science at the University of Oxford, UK, where he is supervised by Professor Antonis Papachristodoulou in the Control Group. His research focuses on analysing the robustness properties of neural networks for Control applications. His research interests include sum-of-squares programming, large-scale optimisation and controller synthesis through learning.

Antonis Papachristodoulou FIEEE received the M.A./M.Eng. degree in Electrical and Information Sciences from the University of Cambridge, Cambridge, U.K., and the Ph.D. degree in Control and Dynamical Systems (with a minor in Aeronautics) from the California Institute of Technology, Pasadena, CA, USA. He is currently Professor of Engineering Science at the University of Oxford, Oxford, U.K., and a Tutorial Fellow at Worcester College, Oxford. He was previously an EPSRC Fellow. His research interests include large-scale nonlinear systems analysis, sum of squares programming, synthetic and systems biology, networked systems, and flow control. Professor Papachristodoulou received the 2015 European Control Award for his contributions to robustness analysis and applications to networked control systems and systems biology. In the same year, he received the O. Hugo Schuck Best Paper Award.