Proceedings of Machine Learning Research vol 144:1–13, 2021

Exploiting Sparsity for Neural Network Verification

Matthew Newton MATTHEW.NEWTON@ENG.OX.AC.UK
Antonis Papachristodoulou ANTONIS@ENG.OX.AC.UK
Department of Engineering Science, University of Oxford, Parks Road, Oxford, OX1 3PJ, UK

Abstract

The problem of verifying the properties of a neural network has never been more important. This task is often done by bounding the activation functions in the network. Some approaches are more conservative than others and in general there is a trade-off between complexity and conservativeness. There has been significant progress to improve the efficiency and the accuracy of these methods. We investigate the sparsity that arises in a recently proposed semi-definite programming framework to verify a fully connected feed-forward neural network. We show that due to the intrinsic cascading structure of the neural network, the constraint matrices in the semi-definite program form a block-arrow pattern and satisfy conditions for chordal sparsity. We reformulate and implement the optimisation problem, showing a significant speed-up in computation, without sacrificing solution accuracy.

Keywords: Chordal Sparsity, Neural Networks, Verification, Semi-definite Programming

1. Introduction

There has been a huge resurgence in interest in neural networks over the past decade, mainly due to the increased computational power available alongside the creation of Alexnet (Krizhevsky et al. (2012)) and Resnet (He et al. (2015)). The research community, along with industry, have realised the significant potential that neural networks possess to perform complex tasks that were once thought to be impossible to program by machine. The areas that they have been applied to have been expanding and include image recognition, weather prediction and natural language processing (Zhang and Ma (2012)). One of the most exciting applications of neural networks is in the field of control theory. There were many works in this area in the 1990s e.g. Miller et al. (1990). However, with the recent success of neural networks in many machine and reinforcement learning applications, and the parallel developments in advanced robust control methods, there is significant scope for research 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 (Recht (2019)). The intersection of these works could be the most useful to society, as big data becomes more widely available. Neural networks have also shown to be useful in multiple aspects of control systems, such as the work in Chen et al. (2020), which uses a neural network to learn Lyapunov functions to show stability of a feedback system.

One of the biggest issues with neural networks is their sensitivity to adversarial attacks; works from Madry et al. (2019) and Athalye et al. (2018) show how small changes to the input set can lead to large fluctuations in the outputs of the neural network. The 'black box' approach to neural networks is one of the limiting factors for their use in safety-critical applications; robustness guarantees are essential to overcome this issue. One approach to provide these robustness certificates is to bound the non-linear activation functions in the neural network (Salman et al. (2020)). 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 neural network performance.

Works from Krishnamurthy et al. (2018) use a dual approach to help with the scalability issue. The competition ARCH-COMP20 (Lopez et al. (2019)) was created to challenge researchers to verify control systems; the code from each contestant was made available online. Singh et al. (2019) proposes a new parametric framework called 'k-ReLU' that combines multiple activation functions together. Similar work from Tjandraatmadja et al. (2020) considers the multi-variate input space on the activation function to create tighter bounds for the linear program. In Raghunathan et al. (2018), the authors form a semi-definite relaxation to certify the robustness. One scalable method to the semi-definite relaxation is to use an iterative eigenvector approach to create an efficient algorithm to verify large networks (Dathathri et al. (2020)). Quadratic constraints were proposed to bound the activation functions in Fazlyab et al. (2019): it is this framework that will form the basis of this paper. This formulation has also been used to conduct reachability analysis on control feedback systems (Hu et al. (2020)), as the quadratic constraints can be extended to integral quadratic constraints and used to analyse the stability of a neural network controller (Yin et al. (2020)).

However, a big issue with these methods is the limit on their scalability, especially in the semi-definite program (SDP) formulation. One approach to improve the scalability of the SDP is to exploit sparsity in the constraint matrices, in particular *chordal sparsity*. Chordal sparsity has been used in many SDP-related problems such as finding Lypaunov functions (Mason and Papachristodoulou (2014)) and the scalable design of structured controllers (Yang Zheng and Papachristodoulou (2018)). Since neural networks have a natural cascading structure it is possible to use ideas from chordal sparsity to dramatically reduce the computational time to solve these neural network verification problems. Sparsity patterns have previously been used in neural networks. The authors in Latorre et al. (2020) use a polynomial optimisation framework to estimate the Lipchitz constant of a neural network.

In Section 2 we outline the definition of the problem and in Section 3 we state how this can be formulated into an SDP. Then in Section 4 we describe the ideas behind chordal sparsity and how they can be used to improve the computational complexity of an SDP. In Section 5 we conduct the analysis on the chordal sparsity of the SDP constraints. In Section 6 we provide the numerical results of our work and provide a discussion on the findings. The paper is concluded in Section 7, outlining plans for future work.

1.1. Our Contribution

In this paper we reformulate a semi-definite programming (SDP) approach that uses quadratic constraints to provide robustness certifications on a general non-linear feed forward neural network. Through examining the formulation of the problem, we are able to show that there is a chordal sparsity pattern in the SDP constraints, 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 it takes to solve the optimisation problem. We compare the trade-off between conservatism and computational time by using SeDuMi (Sturm (1999)), that uses interior point methods to solve the optimisation problem. We show how the computational time varies with the structure of the neural network. Finally, we propose ideas for future work and how this sparsity exploitation method can be expanded to further improve the computational burden of this verification task.

2. Problem Definition

A multi-layer feed-forward neural network is described as a non-linear function $f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, where n_x is the number of inputs and n_y is the number of outputs. If we consider a set of all possible inputs into a neural network $\mathcal{X} \subset \mathbb{R}^{n_x}$, then the neural network will map these inputs to a set of outputs $\mathcal{Y} \subset \mathbb{R}^{n_y}$ such that

$$\mathcal{Y} = f(\mathcal{X}) := \{y \in \mathbb{R}^{n_y} \mid y = f(x),\ x \in \mathcal{X}\}.$$

It is desirable to ensure that all possible inputs $\mathcal{X}$ map to a safe set of outputs, that we define as a safety specification set $\mathcal{S}_y$ and conversely the safe set of inputs are defined as $\mathcal{S}_x := f^{-1}(\mathcal{S}_y)$. The network is said to be safe if the output lies within this safety region. It is computationally expensive to check if the outputs lie in the $\mathcal{S}_y$ set, since it is non-convex. A relaxation can be computed as a conservative approximation of the set $\mathcal{Y}$ denoted by $\hat{\mathcal{Y}}$, through checking the condition $\hat{\mathcal{Y}} \subseteq \mathcal{S}_y$.

2.1. Neural Network Model

Consider a feed-forward fully connected neural network $f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, with ℓ layers. This can be defined by the set of equations:

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

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 $x^0 \in \mathbb{R}^{n_0}$ is the input into the network. The number of neurons in the k^{th} layer is denoted by n_k; the total number of neurons in the neural network 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

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

where ρ is the activation function. There are many different types of activation functions such as ReLU, tanh, sigmoid and ELU. In this paper we will focus on the ReLU activation function.

The complete neural network can be written in a 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 neural network can then be written as:

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

where

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

3. Formulation into Semi-definite Program

To verify the neural network, we can bound the activation function using a set of equality and inequality constraints. In this formulation we consider quadratic constraints as used in the formulation in Fazlyab et al. (2019). 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 the formulation. In the examples that we tested we noticed that this simplification did not have a significant impact on the accuracy and still provides robustness guarantees on the network, whilst greatly improving the scalability of the problem.

Definition 1 *The input set is bounded by a hyper-rectangle with edges $\underline{x}$ and $\overline{x}$ defined by $\mathcal{X} = \{x \in \mathbb{R}^{n_x} | \underline{x} \leq x \leq \overline{x}\}$. This satisfies the quadratic constraint defined by*

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

where $\Gamma \in \mathbb{R}^{n_x \times n_x}$ is diagonal and non-negative.

Definition 2 *The safe set is bounded by a polytope defined by $\mathcal{S}_y = \bigcap_{i=1}^m \{y \in \mathbb{R}^{n_y} \,|\, c_i^T y - d_i \leq 0\}$ which can be written as the intersection of quadratic constraints such that*

$$\mathcal{S}_y = \bigcap_{i=1}^{m} \left\{ x \in \mathbb{R}^{n_x} \,\middle|\, \begin{bmatrix} x \\ f(x) \\ 1 \end{bmatrix}^T S_i \begin{bmatrix} x \\ f(x) \\ 1 \end{bmatrix} \right\},$$

where

$$S_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & c_i \\ 0 & c_i^T & -2d_i \end{bmatrix}, \text{ for } i = 1, \ldots, m.$$

Definition 3 *The hidden layers in the neural network are bounded with the quadratic constraints*

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

where

$$\begin{aligned}
Q_{11} &= -2\text{diag}(\boldsymbol{\alpha} \circ \boldsymbol{\beta} \circ \lambda),\ Q_{12} = \text{diag}((\boldsymbol{\alpha}+\boldsymbol{\beta}) \circ \lambda),\ Q_{13} = -\boldsymbol{\beta} \circ \nu - \boldsymbol{\alpha} \circ \eta,\ Q_{22} = 0, \\
Q_{23} &= \nu + \eta,\ Q_{33} = 0,\ \boldsymbol{\alpha} = [\mathbf{1}_{\mathcal{I}^+}(1), \ldots, \mathbf{1}_{\mathcal{I}^+}(n)],\ \boldsymbol{\beta} = [1 + \mathbf{1}_{\mathcal{I}^-}(1), \ldots, 1 + \mathbf{1}_{\mathcal{I}^-}(n)], \\
& \nu_i \geq 0, \text{ for } i \notin \mathcal{I}^+,\ \eta_i \geq 0, \text{ for } i \notin \mathcal{I}^-,\ \lambda_i \in \mathbb{R}, \\
& \mathbf{1}_{\mathcal{I}^+}(i) \text{ and } \mathbf{1}_{\mathcal{I}^-}(i) \text{ are vectors of ones if } i \in \mathcal{I}^+ \text{ or } i \in \mathcal{I}^- \text{ respectively}, \\
\mathcal{I}^+ &= \{i \,|\, x_i \geq 0,\ \forall x \in \mathcal{X} \subseteq \mathbb{R}^n\} \text{ (ReLU active)}, \\
\mathcal{I}^- &= \{i \,|\, x_i \leq 0,\ \forall x \in \mathcal{X} \subseteq \mathbb{R}^n\} \text{ (ReLU inactive)}.
\end{aligned}$$

Using the S-procedure the quadratic constraints can be combined together and the verification problem can be posed as an SDP.

$$\begin{aligned} \text{minimize} \quad & d_i && (2a) \\ \text{subject to} \quad & M_{in}(P) + M_{mid}(Q) + M_{out}(S_i) \preceq 0, && (2b) \\ & (P, Q, S_i) \in \mathcal{P}_{\mathcal{X}} \times \mathcal{Q}_{\phi} \times \mathbb{S}^{n_x+n_y+1}, && (2c) \end{aligned}$$

where

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

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

4. 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 semi-definite matrices, and hence how they can be used to speed up the solve time of an SDP. As mentioned in Section 1, the neural network verification problem has scalability issues, which motivates the use of chordal sparsity in this problem.

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 a subset of the vertices of a graph $G(\mathcal{V}, \mathcal{E})$ together with any edges whose endpoints are both in this 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 (Kakimura (2010)).

Definition 4 *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.*

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 5 *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.*

Consider now a symmetric matrix $X \in \mathbb{S}^n$ with a sparsity pattern represented by an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, such that $X_{ij} = X_{ji} = 0, \forall i \neq j$ 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. 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. An important result relates matrices X that are positive semi-definite, to such a decomposition.

Theorem 6 *(Agler et al. (1988)) Consider the chordal graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ that is made up of maximal cliques $\{\mathcal{C}_1,\mathcal{C}_2,\dots,\mathcal{C}_k\}$. Then $Z \in \mathbb{S}^n_+(\mathcal{E},0)$ if and only if there exist $Z_k \in \mathbb{S}^{|\mathcal{C}_k|}_+$ for $k=1,\dots,n$ such that*

$$Z = \sum_{k=1}^{n} E^T_{\mathcal{C}_k} Z_k E_{\mathcal{C}_k},$$

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

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

This is useful as it means that testing positive semi-definiteness of a large matrix with chordal sparsity can be done in a distributed way. This idea can also be extended to sparse block matrices (Zheng (2019)).

The above ideas have been extended to SDP, and different solvers have been developed to exploit this chordal sparsity property. One example is sparseCoLo (Kim et al. (2011)), which uses four conversion methods via positive semi-definite matrix completion. It uses both the primal and dual forms of linear, semi-definite 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 a linear matrix inequality. Another is CDCS (Zheng et al. (2017)) which uses a first order splitting method called alternating direction method of multipliers (ADMM).

5. Chordal Analysis of Constraints

In this section we investigate the sparsity pattern of the constraints in our problem (2) and show that they possess a natural aggregate chordal sparsity structure, with the degree of sparsity depending on the size of the neural network. To show this we write the constraints out in full and consider each constraint separately before combining them together.

5.1. Input Constraint

The overall input constraint can be written as

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

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

5.2. Output Constraint

The overall output constraint can be written as

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

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.

5.3. 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

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

which can be split into four sub-matrices

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

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

$$A^T Q_{11} A = \begin{bmatrix} (W^0)^T Q_{11}^0 W^0 & 0 & 0 & \dots & 0 \\ 0 & (W^1)^T Q_{11}^1 W^1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & (W^{l-1})^T Q_{11}^{l-1} W^{l-1} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B^T Q_{22} B = 0,$$

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

The Q_{11}^k and Q_{12}^k terms represent the sub-matrices of Q_{11} and Q_{12} respectively, corresponding to the k^{th} layer of the neural network. The sum of all these terms gives

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

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 (Sun et al. (2013)). This pattern can be shown visually in Figure 1.

When all of the constraints are combined to form (2b) in the SDP, the constraint keeps this block arrow pattern. This is because M_{in} and M_{out} only have terms which will overlap with the terms in M_{mid}. Since the sum of the constraints has the same pattern as the M_{mid} matrix, the constraint will be chordal or have a simple chordal extension that can be created. This block arrow pattern is shown in Figure 2. This result is significant as it shows that the neural network structure is inherited in the algorithm formulation and should be exploited in computation for scalability.

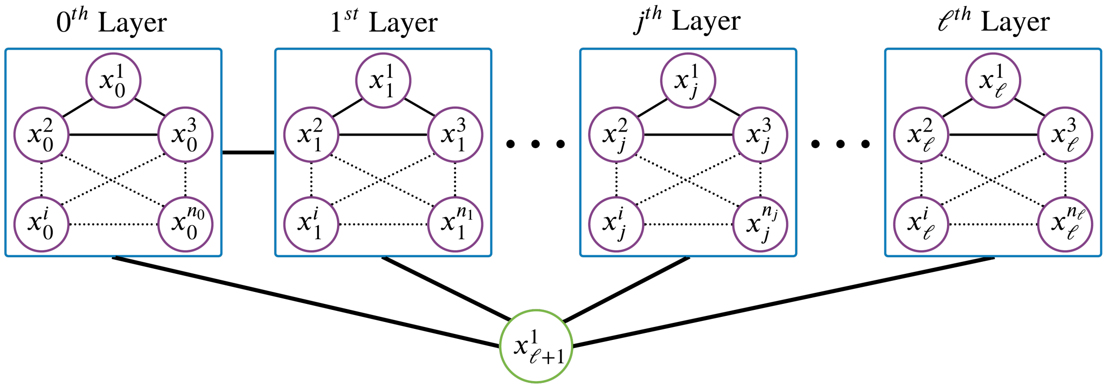
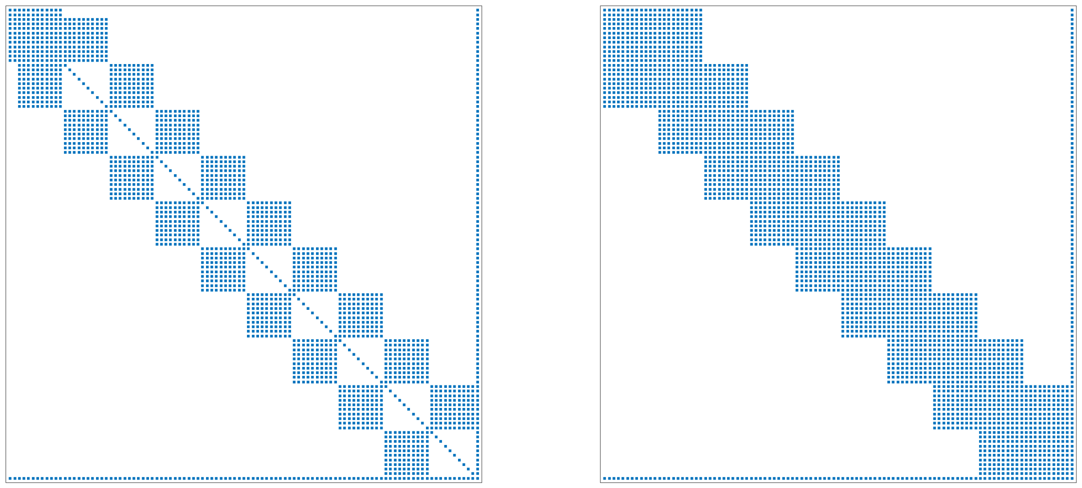


Figure 1: Graph showing the chordal structure of the constraint matrices in the SDP. x_j^i represents the term in the constraint matrices corresponding to the j^{th} layer with i^{th} hidden unit. The solid lines between layers represents that all of the nodes in that layer are connected to all of the nodes in the adjacent layer. The dots represent that this pattern continues to the ℓ^{th} layer in the network.

6. Numerical Results

In this section we compare the computational aspects of solving this optimisation problem with and without exploiting chordal sparsity. The solve time and solution accuracy are compared for different SDP solvers using YALMIP (Löfberg (2004)) as the SDP parser. The solvers we will be comparing are sparseCoLo (Kim et al. (2011)), which will exploit the sparsity of the problem and SeDuMi (Sturm (1999)), which is an SDP solver that uses interior point methods. All experiments were run on a 4-core processor with 16GB of RAM. We used code provided from the authors of Fazlyab et al. (2019) called 'Deep-SDP' that creates the matrices that are used in the SDP. The weights and biases of the neural network are randomly generated with the dimensions specified with the number of inputs, number of hidden layers, size of each hidden layer and number of outputs. Our main focus is on the classification problem, to verify a static feed-forward neural network. 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 $x^\star = (1, 1)$ for comparison purposes to benchmark against previous results.



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

Figure 2: Diagram comparing the sparsity pattern of the $M_{in} + M_{mid} + M_{out}$ matrix constraints of a 10-layer neural network with 10 hidden units in each layer.

6.1. One layer with varying hidden units

Consider a single layer neural network with n_1 hidden neurons in the layer. From Figure 3 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. The bounds in each solver were identical, showing no loss of accuracy from the sparse solver.

6.2. Two hidden units with varying layers

In this case the neural network has two nodes in each layer and ℓ layers. Figure 4 shows that sparseCoLo becomes faster after 50 layers and scales well. Similarly to the experiment in Section 6.1, 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.

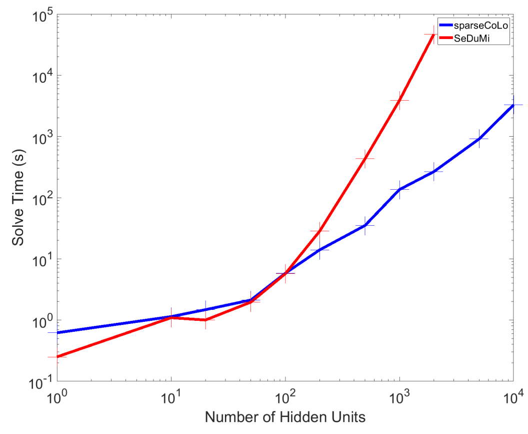


Figure 3: Graph comparing the computational time to solve a one-layer network with a varying number of hidden units.

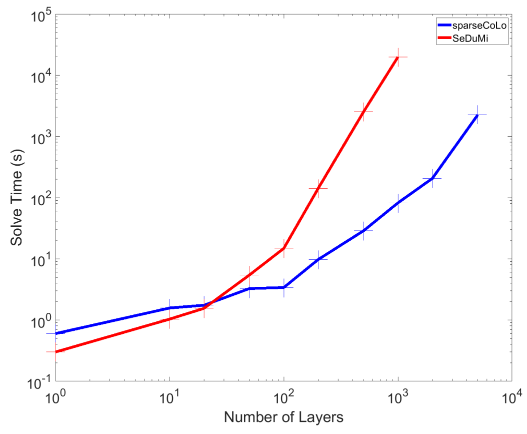


Figure 4: Graph comparing the computational time to solve an ℓ-layer network with two hidden units in each layer.

7. Conclusion

In this paper we considered the robustness analysis problem for a fully-connected feed forward neural network using a semi-definite programming (SDP) framework. Due to the intrinsic cascading structure of the neural network there is a naturally arising sparsity pattern in the linear matrix inequality constraints, which can be taken advantage of in computations. We applied theory from chordal graphs and semi-definite 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 neural network classifiers. The computational time to solve the problem depends on the neural network structure.

This paper opens up many areas for future work. The first is expanding these ideas to the stability of feedback systems with neural network controllers (Yin et al. (2020)). It would be interesting to do a more in-depth investigation into how to trade accuracy to computation. This analysis has only focused on ReLU activation functions, however this can be expanded to many others. First order methods such as CDCS (Zheng et al. (2017)) can be used to improve scalability further with the trade off of a less accurate solution. Finally, this paper has assumed a fully-connected network, 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 neural network pruning that can be used to simplify the network (Blalock et al. (2020)). A combination of these methods will lead to more accurate and efficient algorithms to verify neural networks.

Acknowledgments

The authors would like to thank Dr Yang Zheng for his valued discussions on this topic and Prof Mahyar Fazlyab for making their code available on GitHub. This work was supported by EPSRC grants EP/L015897/1 (to M. Newton) and EP/M002454/1 (to A. Papachristodoulou).

References

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

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, 2018.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv*, 2020.

Shaoru Chen, Mahyar Fazlyab, Manfred Morari, George J. Pappas, and Victor M. Preciado. Learning Lyapunov Functions for Piecewise Affine Systems with Neural Network Controllers. 2020.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy Liang, and Pushmeet Kohli. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. (NeurIPS):1–22, 2020.

Mahyar Fazlyab, Manfred Morari, and George J. Pappas. Safety Verification and Robustness Analysis of Neural Networks via Quadratic Constraints and Semidefinite Programming. 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Haimin Hu, Mahyar Fazlyab, Manfred Morari, and George J. Pappas. Reach-SDP: Reachability Analysis of Closed-Loop Systems with Neural Network Controllers via Semidefinite Programming. pages 1–15, 2020.

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

Sunyoung Kim, Masakazu Kojima, Martin Mevissen, and Makoto Yamashita. Series b: Operations research exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion. *Math. Program.*, 129:33–68, 09 2011.

Krishnamurthy, Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

Fabian Latorre, Paul Rolland, and Volkan Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization, 2020.

J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, and Taylor T. Johnson. Arch-comp19 category report: Artificial intelligence and neural network control systems for continuous and hybrid systems plants. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61, pages 103–119, 2019.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

Richard P. Mason and Antonis Papachristodoulou. Chordal sparsity, decomposing SDPs and the Lyapunov equation. *Proceedings of the American Control Conference*, pages 531–537, 2014.

W. Thomas Miller, Richard S. Sutton, and Paul J. Werbos, editors. *Neural Networks for Control*. MIT Press, Cambridge, MA, USA, 1990.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples, 2018.

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):253–279, 2019.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks, 2020.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32(NeurIPS):14–16, 2019.

Jos F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

Yifan Sun, Martin S. Andersen, and Lieven Vandenberghe. Decomposition in conic optimization with partially separable structure, 2013.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. The Convex Relaxation Barrier, Revisited: Tightened Single-Neuron Relaxations for Neural Network Verification. 2020.

Richard P. Mason Yang Zheng and Antonis Papachristodoulou. Scalable Design of Structured Controllers Using Chordal Decomposition. *IEEE Transactions on Automatic Control*, 63(3):752–767, 2018.

He Yin, Peter Seiler, and Murat Arcak. Stability Analysis using Quadratic Constraints for Systems with Neural Network Controllers. 2020.

Cha Zhang and Yunqian Ma. *Ensemble Machine Learning: Methods and Applications*. Springer Publishing Company, Incorporated, 2012.

Yang Zheng. Chordal sparsity in control and optimization of large-scale systems (phd thesis). university of oxford., 2019.

Yang Zheng, Giovanni Fantuzzi, Antonis Papachristodoulou, Paul Goulart, and Andrew Wynn. Chordal decomposition in operator-splitting methods for sparse semidefinite programs. 2017.