Efficient Neural Network Verification via Layer-based Semidefinite Relaxations and Linear Cuts

Ben Batten , Panagiotis Kouvaros , Alessio Lomuscio , Yang Zheng
Department of Computing, Imperial College London, UK
{b.batten20, p.kouvaros, a.lomuscio, y.zheng}@imperial.ac.uk

Abstract

We introduce an efficient and tight layer-based semidefinite relaxation for verifying local robustness of neural networks. The improved tightness is the result of the combination between semidefinite relaxations and linear cuts. We obtain a computationally efficient method by decomposing the semidefinite formulation into layerwise constraints. By leveraging on chordal graph decompositions, we show that the formulation here presented is provably tighter than current approaches. Experiments on a set of benchmark networks show that the approach here proposed enables the verification of more instances compared to other relaxation methods. The results also demonstrate that the SDP relaxation here proposed is one order of magnitude faster than previous SDP methods.

1 Introduction

Neural networks (NNs) are known to be fragile and susceptible to adversarial attacks [Goodfellow *et al.*, 2014]. In AI-based, safety-critical applications, it is important to formally verify that a network is correct with respect to noteworthy specifications, e.g., *local adversarial robustness*, before they are deployed. Current methods for NN verification can be categorized into *complete* and *incomplete* approaches. Aside from computational considerations, complete approaches are guaranteed to resolve any verification query. Incomplete approaches are normally based on various forms of convex approximations of the network, and only guarantee that whenever they output that the network is safe, then that is indeed the case. While this typically enables faster computation, the looser this approximation is, the more likely it is that the method may not be able to verify the problem instance. As a result, the present objective in incomplete methods is the development of tighter approximations, which can be efficiently computed, thereby strengthening the efficacy of the methods in answering the verification problem [Salman *et al.*, 2019]. In this paper, we advance the state of the art towards this objective by developing a novel relaxation based on semidefinite programs (SDPs). Our SDP relaxation is provably tighter than related SDP approaches, whilst also being more efficient.

Related Work. Complete methods are either based on mixed-integer linear programming (MILP) [Bastani *et al.*, 2016; Lomuscio and Maganti, 2017; Tjeng *et al.*, 2019; Anderson *et al.*, 2020; Botoeva *et al.*, 2020], satisfiability modulo theories [Katz *et al.*, 2017; 2019; Ehlers, 2017], or bound propagation techniques coupled with input refinement [Wang *et al.*, 2018; Henriksen and Lomuscio, 2020]. While these methods offer theoretical termination guarantees, at present they do not scale to the network sizes that incomplete approaches are able to address.

Incomplete methods are typically based on bound propagation [Singh *et al.*, 2018; 2019b; Weng *et al.*, 2018; Tjandraatmadja *et al.*, 2020], duality [Dvijotham *et al.*, 2018; Dathathri *et al.*, 2020; Wong and Kolter, 2018] and SDP relaxations [Raghunathan *et al.*, 2018; Fazlyab *et al.*, 2020]. A common theme in this research is the linear program (LP) relaxation for the univariate ReLU function. A foundational relaxation is the triangle relaxation from [Ehlers, 2017] which gives the tightest possible convex relaxation of the univariate ReLU function and forms the basis of many of the cited methods; see [Salman *et al.*, 2019; Li *et al.*, 2020] for detailed comparisons. It was recently shown that the efficacy of these methods is intrinsically limited by the same *convex relaxation barrier* which is characterised by the tightness of the triangular relaxation [Salman *et al.*, 2019].

Two recent methods to bypass this barrier are the `kPoly` [Singh *et al.*, 2019a] and the `OptC2V` [Tjandraatmadja *et al.*, 2020]; both of these offer tighter LP relaxations by considering interactions of multiple neurons and multivariate inputs. Another way to bypass the barrier is to seek alternative stronger relaxations beyond LPs, such as SDPs [Raghunathan *et al.*, 2018; Fazlyab *et al.*, 2020]. It has been empirically observed that the SDP relaxation in [Raghunathan *et al.*, 2018] is much tighter than LP relaxations. However, SDPs are computationally harder solve. To overcome this, a recent work [Dathathri *et al.*, 2020] develops a customized subgradient algorithm to solve a dual SDP relaxation to the one introduced in [Raghunathan *et al.*, 2018].

Contributions. We develop a novel, efficient, layer-based SDP relaxation for NN verification. Unlike [Dathathri *et al.*, 2020] that centres around first-order algorithm developments, we here focus on tightening the SDP relaxations, and on exploiting the cascading structure of NNs for improved efficiency. Specifically, we first extend the SDP formulation

in [Raghunathan *et al.*, 2018] with linear cuts from the triangle relaxation [Ehlers, 2017]. This leads to a new SDP relaxation that is provably tighter than both [Raghunathan *et al.*, 2018; Dathathri *et al.*, 2020] and the triangle relaxation from [Ehlers, 2017] (Proposition 1). We further show that said linear cuts enable the exploitation of the activation pattern of NNs to simplify our SDP formulation (Proposition 2). By exploiting the cascading structure of NNs, we develop an equivalent layer decomposition to the SDP approach which enjoys a significant computational speed-up (Proposition 3). This layer decomposition strategy is motivated by the advances in sparse SDPs [Vandenberghe and Andersen, 2015].

We evaluated our approach on benchmarks from [Raghunathan *et al.*, 2018; Singh *et al.*, 2019a; Dathathri *et al.*, 2020; Tjandraatmadja *et al.*, 2020]. The experiments reported that the layer-based SDP proposed here offers better accuracy, while being an order of magnitude faster than [Raghunathan *et al.*, 2018]. Also, the method could verify more images than the state-of-the-art LP-based methods such as `kPoly` [Singh *et al.*, 2019a] and `OptC2V` [Tjandraatmadja *et al.*, 2020] for some networks, while the computational cost of the verification step remained in the same order of magnitude.

The rest of the paper is organised as follows: we introduce the neural network verification problem and two commonly used relaxations in Section 2. In Section 3, we present new SDP relaxations via linear cuts and layer-based decomposition. Section 4 reports a comprehensive evaluation of the method on various NNs that are commonly used in the literature. We derive some conclusions in Section 5.

2 Preliminaries

We here outline the notation, present the verification problem, and introduce two widely used convex relaxations.

Feed-forward ReLU neural networks (NNs). We consider an L-layer feed-forward NN $f(x_0) : \mathbb{R}^d \to \mathbb{R}^m$. We use $\hat{x}_i \in \mathbb{R}^{n_i}$ and $x_i \in \mathbb{R}^{n_i}$ to denote the pre-activation and activation vectors of the i-th layer, and define the NN output as $f(x_0) := W_L x_L + b_L$, with $x_{i+1} = \text{ReLU}(\hat{x}_{i+1})$ and $\hat{x}_{i+1} = W_i x_i + b_i$, $i = 0, \dots, L-1$, where $W_i \in \mathbb{R}^{n_{i+1} \times n_i}$, $b_i \in \mathbb{R}^{n_{i+1}}$ are the weights and biases, respectively, $n_0 = d, n_{L+1} = m$ are input and output dimensions, and the ReLU function is defined as $\text{ReLU}(z) = \max(z, 0)$ for $z \in \mathbb{R}$ (the ReLU function is applied element-wise). We focus on classification networks whereby an input x_0 is assigned to the class associated with the network output of the highest value: $i^\star = \arg\max_{i=1,\dots,m} f(x_0)_i$.

Verification problem. Given a NN $f : \mathbb{R}^d \to \mathbb{R}^m$, a nominal input $\bar{x} \in \mathbb{R}^d$, a linear function ϕ, also called the specification, on the network's outputs, and a perturbation radius $\epsilon \in \mathbb{R}$, the verification problem we study is to determine whether

$$\phi(y) > 0, \quad \text{subject to } y = f(x_0), \quad \|x_0 - \bar{x}\|_\infty \le \epsilon, \quad (1)$$

where $\|\cdot\|_\infty$ denotes the standard l_∞ norm of a vector. In particular, we hereafter focus on the *local adversarial robustness* problem whereby the specification is $\phi(y) = y(i^\star) - y(i)$ for a target label i. A network is said to be *certifiably robust* on input $\bar{x}$ and perturbation radius ϵ if the answer to the verification problem (1) is true for all $i \neq i^\star$.

Verification via mathematical optimisation. Problem (1) can be answered by solving the optimisation problem

$$\begin{aligned} \gamma^\star := \min_{x_0,\dots,x_L} \quad & c^\mathsf{T} x_L + c_0 \\ \text{subject to} \quad & x_{i+1} = \text{ReLU}(W_i x_i + b_i),\ i \in [L], \quad (2a) \\ & \|x_0 - \bar{x}\|_\infty \le \epsilon, \quad (2b) \end{aligned}$$

where $c^\mathsf{T} = W_L(i^\star, :) - W_L(i, :)$, $c_0 = b_L(i^\star) - b_L(i)$, and $[L]$ denotes $\{0, 1, \dots, L-1\}$. The verification problem (1) is true if and only if the optimal value, $\gamma^\star$, of (2) is positive. The optimisation problem is however non-convex because of (2a) and is therefore generally difficult to solve. To obtain a tractable *convex relaxation* of the problem, we derive an outer-approximation of the feasible region $(x_0, x_1, \dots, x_L)$ in (2) using a convex set, $\mathcal{D}$. This relaxes (2) to a convex problem

$$\gamma_\mathcal{D} := \min_{(x_0,x_1,\dots,x_L) \in \mathcal{D}} \quad c^\mathsf{T} x_L + c_0 \quad (3)$$

which provides a valid lower bound, $\gamma^\star \ge \gamma_\mathcal{D}$. If $\gamma_\mathcal{D} > 0$, then the answer to the verification problem (1) is *true*. If however $\gamma^\star > 0 \ge \gamma_\mathcal{D}$, then the verification problem cannot be decided. In this paper we are concerned with two convex relaxations: the *triangle relaxation* [Ehlers, 2017] and the *semidefinite relaxation* [Raghunathan *et al.*, 2018].

Triangle relaxation. The triangle relaxation approximates a single univariate ReLU function $z = \max\{x, 0\}$ with its convex hull. Specifically, the ReLU constraints (2a) are approximated by a set of linear constraints

$$x_{i+1} \ge 0, \quad x_{i+1} \ge \hat{x}_{i+1}, \quad (4a)$$

$$x_{i+1} \le k_i \odot (\hat{x}_{i+1} - \hat{l}_{i+1}) + \text{ReLU}(\hat{l}_{i+1}), \quad (4b)$$

$$\hat{x}_{i+1} = W_i x_i + b_i, \ \ \hat{l}_{i+1} \le \hat{x}_{i+1} \le \hat{u}_{i+1}, \quad (4c)$$

where $i \in [L]$, $\odot$ denotes the Hadamard product, $k_i := \frac{\text{ReLU}(\hat{u}_{i+1}) - \text{ReLU}(\hat{l}_{i+1})}{\hat{u}_{i+1} - \hat{l}_{i+1}}$, and $\hat{u}_{i+1}, \hat{l}_{i+1} \in \mathbb{R}^{n_{i+1}}$ are upper and lower bounds of the pre-activation variable $\hat{x}_{i+1}$ for any input satisfying (2b); these bounds can be computed using interval propagation methods, see, e.g., [Wong and Kolter, 2018; Wang *et al.*, 2018]. The optimal value, γ_LP, of the resulting LP relaxation is relatively easy to compute in practice. However, the quality of the LP relaxation (4) is intrinsically limited, *i.e.*, there is always a positive gap $\gamma^\star - \gamma_\text{LP} > 0$ for many practical NNs, referred to as the *convex relaxation barrier* [Salman *et al.*, 2019]. Two recent approaches that improve the tightness of the LP relaxation are the `kPoly` [Singh *et al.*, 2019a], which explicitly considers the interaction of multiple ReLU constraints, and the `OptC2V` [Tjandraatmadja *et al.*, 2020], which explicitly considers multivariate inputs of a single ReLU constraint.

Semidefinite relaxation. This relaxation utilizes a *single* positive semidefinite (PSD) constraint that couples all ReLU constraints in (2a) to obtain a convex SDP [Raghunathan *et al.*, 2018]. The key idea is to equivalently replace the ReLU constraints (2a) with the following quadratic constraints

$$x_{i+1} \ge 0, \ x_{i+1} \ge W_i x_i + b_i, \quad i \in [L], \quad (5a)$$

$$x_{i+1} \odot (x_{i+1} - W_i x_i - b_i) = 0, \ i \in [L]. \quad (5b)$$

Also, the input constraint (2b) as well as the lower and upper bounds $l_i, u_i \in \mathbb{R}^{n_i}$ on the activation vectors $x_i, i = 1, \ldots, L-1$ (which can be obtained using interval prorogation methods) can be reformulated as quadratic constraints

$$x_i \odot x_i - (l_i + u_i) \odot x_i + l_i \odot u_i \leq 0,\ i \in [L], \quad (6)$$

where $i = 0$ corresponds to the l_∞ input constraint (2b).

Polynomial lifting and SDP-based hierarchies can be used to solve the resulting polynomial optimisation problem. Specifically a lifting matrix, P, of monomials

$$P := vv^\mathsf{T},\ \text{where } v := \left[1, x_0^\mathsf{T}, x_1^\mathsf{T}, \ldots, x_L^\mathsf{T}\right]^\mathsf{T} \in \mathbb{R}^{1+\sum_{i=0}^{L} n_i}$$

can be defined [Raghunathan *et al.*, 2018]. Then, all the constraints in (5) and (6) become linear in terms of the elements of P. By relaxing the monomial matrix P to be $P \succeq 0$, we obtain an SDP relaxation of (2) as follows

$$\begin{aligned}
\min_{P}\quad & c^\mathsf{T} P[x_L] + c_0 \\
\text{subject to}\quad & P[x_{i+1}] \geq 0, P[x_{i+1}] \geq W_i P[x_i] + b_i, && \text{(7a)}\\
& \operatorname{diag}\left(P[x_{i+1}x_{i+1}^\mathsf{T}] - W_i P[x_i x_{i+1}^\mathsf{T}]\right) \\
& \qquad - b_i \odot P[x_{i+1}] = 0,\ \ i \in [L] && \text{(7b)}\\
& \operatorname{diag}\left(P[x_i x_i^\mathsf{T}]\right) - (l_i + u_i) \odot P[x_i] \\
& \qquad + l_i \odot u_i \leq 0, i \in [L] && \text{(7c)}\\
& P[1] = 1,\ P \succeq 0, && \text{(7d)}
\end{aligned}$$

where we adopt the same symbolic indexing $P[\cdot]$ as [Raghunathan *et al.*, 2018] to index the elements of P. It is clear that (7a) and (7b) correspond to the ReLU constraints (5), and that (7c) corresponds to the bounds on activation vectors in (6). We denote the optimal value of (7) as $\gamma_{\text{SDP},1}$. We always have $\gamma^\star \geq \gamma_{\text{SDP},1}$, where the equality is achieved if the optimal solution, P, to (7) is of rank one. The exactness of a variant of the SDP relaxation (7) is discussed in [Zhang, 2020] via geometric techniques.

3 Linear Cuts and Layer-based Semidefinite Relaxations

In this section we develop new SDP relaxations for the verification problem (2). The resulting formulations will lead to both tighter and computationally more efficient relaxations.

We begin our analysis by observing two potential drawbacks from the SDP relaxation (7). The first is that, despite the use of the PSD constraint $P \succeq 0$, the relaxation quality of (7) may be looser than the LP relaxation (4), *i.e.*, $\gamma_{\text{LP}} > \gamma_{\text{SDP},1}$. The second is that the matrix variable, P, in the constraint $P \succeq 0$, is of size $N \times N$, where $N = 1 + \sum_{i=0}^{L} n_i$ corresponds to the network size; this may hinder the scalability of the approach for large networks. We resolve the first shortcoming by adding the linear cuts from (4) to the SDP relaxation (7), leading to a provably tighter SDP relaxation. We alleviate the second by introducing a layer-based SDP relaxation consisting of multiple PSD constraints of smaller sizes.

3.1 Tightened SDP Relaxations via Linear Cuts

It is known, see [Raghunathan *et al.*, 2018], that in certain cases the SDP relaxation (7) can be looser than the LP relaxation (4). Figure 1 illustrates this for the case of a single ReLU neuron with a univariate input. As shown in Figure 1, the feasible region in the LP relaxation (4) becomes exact when $u \leq 0$ (*inactive neuron*) or $l \geq 0$ (*strictly active neuron*), whereas the feasible region in the SDP relaxation (7) is not exact unless $l = u$. As can be observed from the feasible regions, the tightness of the LP relaxation w.r.t the SDP relaxation is attributed to the linear cut (4b), which at times tightens the LP relaxation.

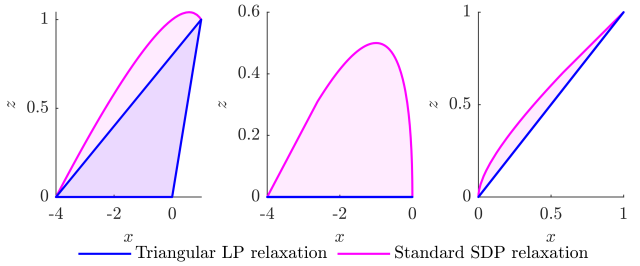

Figure 1: LP and SDP-based outer approximations of $\{(x, z) \in \mathbb{R}^2 \mid z = \text{ReLU}(x), l \leq x \leq u\}$. Left to right: 1) unstable neuron $l = -4, u = 1$; 2) inactive neuron $l = -4, u = 0$; 3) strictly active neuron $l = 0, u = 1$. The standard SDP relaxation (7) is inexact even for inactive/stable neurons, while the triangular relaxation becomes exact. This motivates us to add linear cuts (8) to the SDP (9).

To resolve this, we extend the SDP relaxation (7) to include the linear cut (4b) thereby tightening the relaxation. We express the cut (4b) in terms of the matrix P as follows

$$P[x_{i+1}] \leq k_i \odot \left(W_i P[x_i] + b_i - \hat{l}_i\right) + \text{ReLU}(\hat{l}_i), \quad (8)$$

and add it to (7). This leads to the following SDP relaxation for the verification problem (2):

$$\begin{aligned}
\gamma_{\text{SDP},2} := \min_{P}\quad & c^\mathsf{T} P[x_L] + c_0 \\
\text{subject to}\quad & \text{(7a)}, \text{(7b)}, \text{(7c)}, \text{(7d)} \\
& \text{(8)},\ \ i \in [L].
\end{aligned} \quad (9)$$

Due to the linear cuts (8), the new SDP relaxation (9) is provably tighter than both the original SDP relaxation (7) and the standard triangle LP relaxation (4).

Proposition 1. *Given a non-convex NN verification instance* (2)*, we have that* $\gamma^\star \geq \gamma_{\text{SDP},2} \geq \max\{\gamma_{\text{SDP},1}, \gamma_{\text{LP}}\}$.

The proof of $\gamma^\star \geq \gamma_{\text{SDP},2} \geq \gamma_{\text{SDP},1}$ is straightforward. To establish $\gamma_{\text{SDP},2} \geq \gamma_{\text{LP}}$, it suffices to show that any feasible solution in (9) can be used to construct a feasible solution in (4) with the same cost value.

We now compare SDP to LP relaxations. As already noted in [Raghunathan *et al.*, 2018], the single PSD constraint, $P \succeq 0$ in (7), captures the interaction of ReLU constraints (2a) implicitly, while the triangle relaxation (4) relaxes (2a) individually. However, this constraint $P \succeq 0$ alone does not guarantee an improved relaxation. The linear cuts (4) can still effectively remove some redundant outer-approximation in SDP relaxations (Proposition 1). Recent approaches such as `kPoly` [Singh *et al.*, 2019a] and `OptC2V` [Tjandraatmadja *et al.*, 2020] strengthen the standard triangular LP (4) by adding additional linear cuts via relaxing multiple ReLU neurons together, or reasoning multivariate inputs directly. Those linear

cuts can be potentially combined with the SDP relaxation (9), leading to a tighter relaxation for (2).

Another flexibility in SDP relaxations lies in the fact that they allow the reformulation of different linear constraints into quadratic constraints and then their linearisation in terms of the matrix variable P. This is known as *the reformulation-linearisation technique (RLT)* [Anstreicher, 2009]. We have used this technique to derive (7c) from (6), which often tightens the SDP relaxation.

3.2 Simplified SDP Relaxation via Activation Patterns

It is a common practice in LP-based relaxations and MILP formulations (e.g., [Singh *et al.*, 2019b; Weng *et al.*, 2018; Botoeva *et al.*, 2020]) to simplify the verification problem (2) using the NN's activation pattern. After incorporating the linear cuts (8), the SDP relaxation (9) can also use the activation pattern to reduce the dimension of the PSD constraint $P \succeq 0$; this was not considered in [Raghunathan *et al.*, 2018; Dathathri *et al.*, 2020]. Particularly, given lower and upper bounds on the pre-activation vector $\hat{l}_{i+1} \leq \hat{x}_{i+1} \leq \hat{u}_{i+1}$, it is known that the constraints (4) for stable neurons of the $(i{+}1)$-th layer become exact and can be simplified: 1) if the k-th neuron is strictly active, i.e., $\hat{u}_{i+1}(k) \geq \hat{l}_{i+1}(k) \geq 0$, then $x_{i+1}(k) = W_i(k,:)x_i + b_i(k)$, or 2) if the neuron is inactive, i.e., $0 \geq \hat{u}_{i+1}(k) \geq \hat{l}_{i+1}(k)$, then $x_{i+1}(k) = 0$.

The information regarding inactive neurons can also be removed in (9) since $P[x_{i+1}](k)$ becomes zero due to the linear cuts (8). This effectively reduces the dimension of the PSD constraint $P \succeq 0$ without altering the optimal value.

Proposition 2. *Consider a non-convex NN verification instance* (2)*, with n_i neurons in each layer, $i = 0, 1, ..., L$. Given a set of lower and upper bounds l_i, u_i, $\hat{l}_{i+1}, \hat{u}_{i+1}, i \in [L]$, suppose the number of unstable and strictly active neurons is $\hat{n}_i$. Then, in* (9)*, the PSD constraint $P \succeq 0$ of size $(1 + \sum_{i=0}^{L} n_i) \times (1 + \sum_{i=0}^{L} n_i)$ can be equivalently replaced by a smaller PSD constraint $\hat{P} \succeq 0$ of size $(1 + \sum_{i=0}^{L} \hat{n}_i) \times (1 + \sum_{i=0}^{L} \hat{n}_i)$.*

In many practical cases, a significant portion of the neurons are stable under a given verification query, especially when small perturbation radiuses, ϵ, are considered. Thus, adding the linear cuts (8) not only makes the SDP relaxation (9) theoretically stronger (cf. Proposition 1), but also computationally easier (cf. Proposition 2) than (7) in [Raghunathan *et al.*, 2018]. We note that it is possible to first prune the inactive neurons to form a new NN, and then apply the SDP (7) in [Raghunathan *et al.*, 2018] to this newly pruned NN. This process implicitly uses the power of linear cuts from (4), but was not discussed nor implemented in [Raghunathan *et al.*, 2018; Dathathri *et al.*, 2020].

3.3 Layer-based SDP Relaxations

We here further reduce the dimensionality of the PSD constraint in (9) by exploiting the layer-wise cascading structure of NNs, whereby each activation vector of a layer depends only on the previous layer's activation vector. We begin with the equivalent quadratic formulation of (5). Instead of using a single big matrix, P, as in (7), we introduce multiple matrices of monomials P_i for each $i \in [L]$:

$$P_i := v_i v_i^{\mathsf{T}}, \text{where } v_i := \left[1, x_i^{\mathsf{T}}, x_{i+1}^{\mathsf{T}}\right]^{\mathsf{T}} \in \mathbb{R}^{1+n_i+n_{i+1}}. \quad (10)$$

Then, the constraints (5a)-(5b) become linear in P_i:

$$P_i[x_{i+1}] \geq 0, \quad P_i[x_{i+1}] \geq W_i P_i[x_i] + b_i, \quad (11a)$$

$$\operatorname{diag}\left(P_i[x_{i+1}x_{i+1}^{\mathsf{T}}] - W_i P_i[x_i x_{i+1}^{\mathsf{T}}]\right) = b_i \odot P_i[x_{i+1}]. \quad (11b)$$

Also, (7c) and (8) can be written with respect to P_i as

$$\operatorname{diag}\left(P_i[x_i x_i^{\mathsf{T}}]\right) - (l_i + u_i) \odot P_i[x_i] + l_i \odot u_i \leq 0, \quad (12a)$$

$$P_i[x_{i+1}] \leq k_i \odot \left(W_i P_i[x_i] + b_i - \hat{l}_i\right) + \operatorname{ReLU}(\hat{l}_i). \quad (12b)$$

Upon relaxing the monomial matrices P_i to $P_i \succeq 0$, we need to consider the input-output consistency among the P_i's, *i.e.*,

$$P_i[\hat{x}_{i+1}\hat{x}_{i+1}^{\mathsf{T}}] = P_{i+1}[\hat{x}_{i+1}\hat{x}_{i+1}^{\mathsf{T}}], \quad i = 0, \ldots, L-2, \quad (13)$$

where $\hat{x}_{i+1}^{\mathsf{T}} := \left[1, x_{i+1}^{\mathsf{T}}\right]$.

Now, we can introduce a new layer-based SDP relaxation for the verification problem (2):

$$\begin{aligned} \gamma_{\mathrm{SDP},3} := \min_{P_0,\ldots,P_{L-1}} \quad & c^{\mathsf{T}} P_{L-1}[x_L] + c_0 \\ \text{subject to} \quad & (11a), (11b), , \quad i \in [L], \\ & (12a), (12b), , \quad i \in [L], \\ & P_i[1] = 1, P_i \succeq 0, \quad i \in [L], \\ & (13). \end{aligned} \quad (14)$$

Instead of a single, big PSD constraint, $P \succeq 0$ of network size in (9), the layer-based SDP relaxation (14) employs multiple smaller PSD constraints $P_i \succeq 0$ for each layer. Smaller PSD constraints in an SDP are helpful to improve the efficiency of getting its solution using off-the-self solvers [Vandenberghe and Andersen, 2015; Zheng *et al.*, 2020]. Moreover, the solution quality (14) is equivalent to that from (9). Formally, we have:

Proposition 3. *Given a non-convex NN verification instance* (2)*, we have that $\gamma^\star \geq \gamma_{\mathrm{SDP},3} = \gamma_{\mathrm{SDP},2}$.*

Proof. The proof relies on a celebrated chordal decomposition result for sparse PSD matrices [Vandenberghe and Andersen, 2015, Chapter 10]. In particular, the cascading structure in a network can be abstracted as a chain graph, which is chordal with L maximal cliques $\mathcal{C}_i = \{i, i+1\}, i \in [L]$. The block-chordal completion theorem [Zheng, 2019, Theorem 2.18] allows equivalently replacing the single PSD constraint $P \succeq 0$ in (9) with L smaller PSD constraints $P_i \succeq 0$ in (14), each of which corresponds to maximal clique $\mathcal{C}_i$. Now, (14) becomes a reformulation of (9) via the consistency constraint (13). □

Remark 1 (Merging layers and conversion methods)**.** *Intuitively,* (14) *belongs to a class of chordal conversion methods in sparse SDPs [Fukuda* et al.*, 2001; Zheng* et al.*, 2020]. Besides two consecutive layers in* (10)*, one can merge multiple layers (*e.g.*, the first three layers) as one*

clique and form one corresponding PSD variable. The resulting SDP relaxation is equivalent to (9) *and* (14)*, meaning that they offer the same optimal solution. Similar to general conversion methods in SDPs [Fukuda* et al.*, 2001; Zheng* et al.*, 2020], there exists a tradeoff between the size of* $P_i \succeq 0$ *and the number of additional equality constraints* (13) *for the efficiency of solving* (14)*. Some heuristics exist to balance such a tradeoff [Fukuda* et al.*, 2001].*

Further relaxation via dropping equality constraints. As discussed above, the number of equality constraints (13) is quadratic in the number of neurons in each layer. Here, we consider an SDP relaxation that uses only a subset of the constraints in (13). In particular, we consider a linear number of consistency constraints as

$$P_i[\hat{x}_{i+1}] = P_{i+1}[\hat{x}_{i+1}], \quad i = 0, \dots, L-2, \tag{15}$$

and form another layer-based SDP relaxation:

$$\begin{aligned} \gamma_{\text{SDP},4} := \min_{P_0,\dots,P_{L-1}} \quad & c^{\mathsf{T}} P_{L-1}[x_L] + c_0 \\ \text{subject to} \quad & (11a), (11b), , \ \ i \in [L], \\ & (12a), (12b), , \ \ i \in [L], \\ & P_i[1] = 1, P_i \succeq 0, \ \ i \in [L], \\ & (15). \end{aligned} \tag{16}$$

The solution quality of (16) is worse than (14) but is faster to solve and it is still provably better than the LP relaxation (4), *i.e.*, $\gamma^\star \geq \gamma_{\text{SDP},3} \geq \gamma_{\text{SDP},4} \geq \gamma_{\text{LP}}$. The proof is similar to Proposition 1.

4 Experimental Evaluation

We now evaluate the performance of the proposed SDP formulations on several benchmarks from the literature. We compare the resulting implementation against other state-of-the-art (SoA) convex relaxation methods, including the original SDP formulation in [Raghunathan *et al.*, 2018], the advanced `SDP-FO` algorithm [Dathathri *et al.*, 2020], and two recent SoA LP relaxation methods, i.e., `kPoly` [Singh *et al.*, 2019a] and `OptC2V` [Tjandraatmadja *et al.*, 2020].

4.1 Implementation and Experiment Setup

Verification problem. We consider the standard robustness verification problem for image classifiers: given a correctly classified image, verify that the NN returns the same label for all input within an l_∞ perturbation of ϵ. Formally, given an image $\bar{x} \in [0,1]^d$ with a label $i^\star$ and a radius ϵ, a neural network is verified to be robust on $(\bar{x}, \epsilon)$ if $\gamma^\star$ in (2) is positive for all $i \neq i^\star$. For LP- and SDP-based relaxation methods, we solve (2) multiple times for every potential adversarial target $i \neq i^\star$, and check whether the lower bound is positive.

Implementation of LP/SDP relaxations. We consider the formulation (7), originally proposed in [Raghunathan *et al.*, 2018] and our SDP formulations from (9), (14), and (16) for experiments. Note that (9) and (14) are equivalent. We used a subroutine from SparseCoLO [Fujisawa *et al.*, 2009] for layer-decomposition that balances the size of PSD constraints and equality constraints (see Remark 1). We refer to (9) and (14) as `LayerSDP`, and its relaxed version (16) as `FastSDP`. We also consider the standard LP relaxation (4) for benchmark. We denote (7) as `SDP-IP` [Raghunathan *et al.*, 2018]. The lower and upper bounds $l_i, u_i, \hat{l}_{i+1}, \hat{u}_{i+1}$ were computed using a symbolic interval propagation algorithm in [Botoeva *et al.*, 2020]. To get an upper bound of verified accuracy, we run a standard projected gradient descent (`PGD`) algorithm from [Dathathri *et al.*, 2020].

For numerical computation, we need to convert the convex relaxations into a standard conic optimization before passing them to a numerical solver. The authors in [Raghunathan *et al.*, 2018] used the YALMIP toolbox [Lofberg, 2004] for the modelling process. However, we found that YALMIP introduced too much overhead time consumption, also because very similar instances of (2) need to be converted multiple times. Besides, we found that the YALMIP toolbox neglected the cascading structure in SDP relaxations. Thus, we implemented an automatic transformation from the convex relaxations into standard conic optimization. The resulting LP/SDPs were then solved by MOSEK [Mosek, 2015]. We use the time reported by MOSEK for comparison.

Neural Networks. We considered eight fully connected ReLU networks trained on the MNIST dataset. To facilitate the comparison with existing tools, we divided our experiments into three groups: 1) One self-trained NN with two hidden layers, each having 64 neurons; no adversarial training was used. We varied the perturbation radius, ϵ, from 0.01 to 0.05; 2) Three NNs from [Raghunathan *et al.*, 2018]: MLP-SDP, MLP-LP, and MLP-Adv. We followed closely the setup described in [Raghunathan *et al.*, 2018; Dathathri *et al.*, 2020], and tested a perturbation radius $\epsilon = 0.1$; 3) Four deep NNs from [Singh *et al.*, 2019a]: 6×100 ($\epsilon = 0.026$), 9×100 ($\epsilon = 0.026$), 6×200 ($\epsilon = 0.015$), 9×200 ($\epsilon = 0.015$). These ϵ values were used in [Singh *et al.*, 2019a; Tjandraatmadja *et al.*, 2020] and cited there as challenging.

For each network, we verified the first 100 images from the MNIST test set and excluded those incorrectly classified. We performed our experiments on an Intel(R) i9-10850K CPU 3.60GHz machine with 32 GB of RAM, except for `SDP-FO` which was carried out on an Intel i7-1065G7 with 15GB RAM, due to a different implementation from [Dathathri *et al.*, 2020].

4.2 Verification Results

Figure 2 reports the verified accuracy for the 64×2 network with different perturbation radius, ϵ, using different verifiers: `LayerSDP` and `FastSDP` (from this paper), `SDP-IP` [Raghunathan *et al.*, 2018], `SDP-FO` [Dathathri *et al.*, 2020], and standard LP. As expected (cf. Proposition 1), our formulation, `LayerSDP`, offered improved robust accuracies than `SDP-IP` and LP across different ϵ. Interestingly, we observe that `SDP-IP` verified fewer images than the standard LP relaxation when $\epsilon = 0.03, 0.035, 0.04$, and required longer time. This indicates that the behavior in Figure 1 persists in practical NN verification, confirming the tightness of `LayerSDP`. Furthermore, a combination of inactive neuron pruning (Proposition 2) and layer decomposition (Section 3.3) made `LayerSDP` and `FastSDP` two orders of magnitude faster to solve than `SDP-IP`. We observe that `SDP-FO`

	Accuracy		LayerSDP		FastSDP		SDP-IP		SDP-FO‡		LP	kPoly	OptC2V
Models	Correct	PGD	verified	time	verified	time	verified†	time*	verified	time	verified	verified†	verified†
MLP-Adv	98%	94%	**91%**	159.8	72%	81.1	82%	3 974	84%	811.2	65%	–	–
MLP-LP	89%	**80%**	**80%**	12.1	◁	◁	80%	2 453	78%	43.3	79%	–	–
MLP-SDP	98%	**84%**	**84%**	3 392	◁	◁	80%	17 726	64%	153.3	35%	–	–
6 × 100	99%	91%	**75%**	545.3	24%	31.3	–	2 456	◇	◇	21%	44.1%	42.9%
9 × 100	97%	86%	35%	470.3	18 %	38.8	–	2 386	◇	◇	18%	36.9%	**38.4%**
6 × 200	99%	96%	**92%**	2 133	33 %	89.2	–	16 030	◇	◇	30%	57.4%	60.1%
9 × 200	97%	91%	42%	1 874	27 %	130.8	–	19 698	◇	◇	27%	50.6%	**52.8%**

†: These results are taken from previously reported values (SDP-IP from [Raghunathan *et al.*, 2018], kPoly from [Singh *et al.*, 2019a], and OptC2V from [Tjandraatmadja *et al.*, 2020]); Dashes (–) indicate previously reported numbers are unavailable.

‡: We re-run the implementation of SDP-FO from [Dathathri *et al.*, 2020], and the verified accuracies are slightly lower than reported numbers due to different hyper-parameters. ◇ indicates SDP-FO failed to verify any instance within maximum iterations.

*: To facilitate time consumption comparison, we re-run SDP-IP [Raghunathan *et al.*, 2018] over three images for these networks on our machine, and took an average per image.

Table 1: Verified accuracy and runtime per image (in seconds) for a set of NNs used in [Raghunathan *et al.*, 2018; Singh *et al.*, 2019a; Tjandraatmadja *et al.*, 2020; Dathathri *et al.*, 2020]. LayerSDP and FastSDP are identical for NNs with one hidden layer (denoted as ◁).

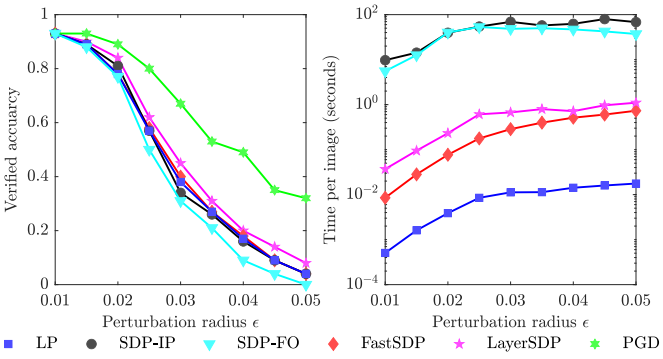

Figure 2: Verified accuracy and runtime per image across various perturbation radius by different methods. These results were run for a 64×2 neural network. As expected, our method LayerSDP consistently offered improved accuracy and was two orders of magnitude faster than SDP-IP [Raghunathan *et al.*, 2018].

verified fewer images than SDP-IP using similar time.

The results in Table 1 demonstrate that LayerSDP is also much faster than SDP-IP, while being more precise than the LP baseline across the networks in [Raghunathan *et al.*, 2018; Dathathri *et al.*, 2020; Singh *et al.*, 2019a; Tjandraatmadja *et al.*, 2020]. Furthermore, for the robustly trained NNs (MLP-Adv, MLP-SDP, MLP-LP), our LayerSDP achieved a very good verified accuracy compared to PGD, with MLP-SDP and MLP-LP matched; this is consistent with the experiments in [Dathathri *et al.*, 2020]. However, we found that SDP-FO [Dathathri *et al.*, 2020] is very sensitive to some hyper-parameters due to the nature of subgradient algorithms.

Compared to the SoA LP-based methods, kPoly [Singh *et al.*, 2019a] and OptC2V [Tjandraatmadja *et al.*, 2020], our LayerSDP significantly improved the verified accuracy for 6×100 and 6×200 networks, while remaining competitive for the other two networks. We observe that the linear cuts in kPoly and OptC2V can be potentially combined in LayerSDP to obtain a stronger relaxation. Finally, while the timings in [Singh *et al.*, 2019a; Tjandraatmadja *et al.*, 2020] may not be comparable, the authors reported average times for kPoly within a range of 2 minutes to 8 minutes per image, and for OptC2V within a range of 2 minutes to 58 minutes per image.

5 Conclusions

We have presented a new layer-based semidefinite relaxation, LayerSDP, that is provably tighter than the SDP relaxations in [Raghunathan *et al.*, 2018; Dathathri *et al.*, 2020]. The additional linear cuts and layer decomposition also make LayerSDP an order of magnitude faster to solve than [Raghunathan *et al.*, 2018] using off-the-shelf solvers. Experiments on a set of fully connected NNs demonstrated the tightness and computational efficiency of LayerSDP. Note that convolutional neural networks have inherent sparsity and structure in convolutional layers, for which chordal graph decomposition is directly applicable and beneficial [Vandenberghe and Andersen, 2015; Zheng *et al.*, 2020]. We leave for further work how to unroll efficiently the convolution layers and incorporate decompositions. Similarly, we would also like to explore customized algorithms similar to [Dathathri *et al.*, 2020] for solving SDPs by exploiting the cascading network structures. Finally, we note that SDP-based approaches can easily deal with other robustness specifications involving linear and quadratic constraints on network inputs. This is a further direction that appears worth exploring.

Acknowledgements

This work is partly funded by DARPA under the Assured Autonomy programme (FA8750-18-C-0095), and the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence. Alessio Lomuscio is supported by a Royal Academy of Engineering Chair in Emerging Technologies.

References

[Anderson *et al.*, 2020] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. Vielma. Strong mixed-integer programming formulations for trained neural networks. *Math. Program.*, pages 1–37, 2020.

[Anstreicher, 2009] K. Anstreicher. Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. *J. Global Optim.*, 43:471–484, 2009.

[Bastani *et al.*, 2016] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. Measur-

ing neural net robustness with constraints. In *NeurIPS16*, pages 2613–2621, 2016.

[Botoeva *et al.*, 2020] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of neural networks via dependency analysis. In *AAAI20*, pages 3291–3299. AAAI Press, 2020.

[Dathathri *et al.*, 2020] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. Liang, and K. Pushmeet. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *NeurIPS20*, pages 5318–5331, 2020.

[Dvijotham *et al.*, 2018] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018.

[Ehlers, 2017] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *ATVA17*, volume 10482, pages 269–286. Springer, 2017.

[Fazlyab *et al.*, 2020] M. Fazlyab, M. Morari, and G. J. Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Trans. Automat. Contr.*, pages 1–1, 2020.

[Fujisawa *et al.*, 2009] K. Fujisawa, S. Kim, M. Kojima, Y. Okamoto, and M. Yamashita. User's manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems. *Dept. of Math. and Comp. Sci. Japan, Tech. Rep.*, pages 152–8552, 2009.

[Fukuda *et al.*, 2001] M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: General framework. *SIAM J. Optim.*, 11(3):647–674, 2001.

[Goodfellow *et al.*, 2014] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[Henriksen and Lomuscio, 2020] P. Henriksen and A. Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[Katz *et al.*, 2017] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV17*, volume 10426, pages 97–117. Springer, 2017.

[Katz *et al.*, 2019] G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou framework for verification and analysis of deep neural networks. In *CAV19*, pages 443–452, 2019.

[Li *et al.*, 2020] L. Li, X. Qi, T. Xie, and B. Li. Sok: Certified robustness for deep neural networks. *arXiv preprint arXiv:2009.04131*, 2020.

[Lofberg, 2004] J. Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *ICRA04*. IEEE, 2004.

[Lomuscio and Maganti, 2017] A. Lomuscio and L. Maganti. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351, 2017.

[Mosek, 2015] ApS Mosek. The mosek optimization toolbox for matlab manual, 2015.

[Raghunathan *et al.*, 2018] A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS18*, pages 10877–10887, 2018.

[Salman *et al.*, 2019] H. Salman, G. Yang, H. Zhang, C. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *NeurIPS19*, pages 9835–9846, 2019.

[Singh *et al.*, 2018] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. In *NeurIPS18*, pages 10802–10813, 2018.

[Singh *et al.*, 2019a] G. Singh, R. Ganvir, M. Püschel, and M. Vechev. Beyond the single neuron convex barrier for neural network certification. In *NeurIPS19*, pages 15098–15109, 2019.

[Singh *et al.*, 2019b] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.

[Tjandraatmadja *et al.*, 2020] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. PATEL, and J. Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. In *NeurIPS20*, pages 21675–21686, 2020.

[Tjeng *et al.*, 2019] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *ICLR19*, 2019.

[Vandenberghe and Andersen, 2015] L. Vandenberghe and M. Andersen. Chordal graphs and semidefinite optimization. *Foundations and Trends in Optimization*, 1(4), 2015.

[Wang *et al.*, 2018] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *NeurIPS18*, pages 6367–6377, 2018.

[Weng *et al.*, 2018] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, D. Boning, I. Dhillon, and L. Daniel. Towards fast computation of certified robustness for ReLU networks. In *ICML18*, pages 5276–5285, 2018.

[Wong and Kolter, 2018] E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML18*, pages 5286–5295. PMLR, 2018.

[Zhang, 2020] R. Zhang. On the tightness of semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS20*, volume 33, pages 3808–3820, 2020.

[Zheng *et al.*, 2020] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. Chordal decomposition in operator-splitting methods for sparse semidefinite programs. *Math. Program.*, 180(1):489–532, 2020.

[Zheng, 2019] Y. Zheng. *Chordal sparsity in control and optimization of large-scale systems*. PhD thesis, University of Oxford, 2019.