Efficient Neural Network Verification with Exactness Characterization

Krishnamurthy (Dj) Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, Pushmeet Kohli
DeepMind
London, UK

Abstract

Remarkable progress has been made on verification of neural networks, i.e., showing that neural networks are provably consistent with specifications encoding properties like adversarial robustness. Recent methods developed for scalable neural network verification are based on computing an upper bound on the worst-case violation of the specification. Semidefinite programming (SDP) has been proposed as a means to obtain tight upper bounds. However, SDP solvers do not scale to large neural networks. We introduce a Lagrangian relaxation based on the SDP formulation and a novel algorithm to solve the relaxation that scales to networks that are two orders of magnitude larger than the off-the-shelf SDP solvers. Although verification of neural networks is known to be NP-hard in general, we develop the first known sufficient conditions under which a polynomial time verification algorithm (based on the above relaxation) is guaranteed to perform exact verification (i.e., either verify a property or establish it is untrue). The algorithm can be implemented using primitives available readily in common deep learning frameworks. Experiments show that the algorithm is fast, and is able to compute tight upper bounds on the error rates under adversarial attacks of convolutional networks trained on MNIST and CIFAR-10.

1 INTRODUCTION

Neural networks have been used to solve several challenging tasks in computer vision, speech recognition, natural language, robotics, etc. However, across these domains, networks have been shown to be brittle. A well-studied example of this is the phenomenon of adversarial examples, i.e., small imperceptible changes to the input to a classifier that cause it to predict an incorrect label [Szegedy et al., 2013]. A commonly used technique for finding adversarial examples is to use a gradient based method [Madry et al., 2018] to search for small perturbations of the input that induce a misclassification. However, it has been observed by Athalye et al. [2018], Uesato et al. [2018] that computing the optimal adversarial attack is challenging and evaluating a classifier on weak attacks can lead to overestimation of the true robustness of the classifier to adversarial examples. This motivates the need for methods that are attack agnostic, i.e., methods that can guarantee that networks are not susceptible to adversarial attacks regardless of the attack algorithm used. Such methods are known in the literature as verification or certification methods and can be used to prove general input-output properties of the network, including robustness to adversarial examples.

Most verification algorithms that have scaled to modern neural network architectures work by computing bounds on the specification being verified given bounds on the network inputs. For instance, for adversarial robustness, the specification is the difference between the logit corresponding to the target label and the true label and the verification algorithm computes a bound on this difference that is valid for all perturbations of the input within a certain set (for example, norm-bounded perturbations). If the bound is smaller than zero, no attack algorithm can induce a misclassification to the target label. The bound is said to be *tight* if there is a set of inputs for which the bound is attained. In this case, the verification algorithm is complete, since it is guaranteed to either rule out the presence of adversarial examples (if the bound is smaller than 0) or find a valid adversarial perturbation (the inputs that achieve the positive bound). There is an intrinsic trade-off between the tightness of the bound and the computational complexity of the verification procedure and most verification algorithms that scale to modern architectures do not compute tight bounds for general networks. This motivates the work we present in this paper: we develop a verification algorithm that scales to large neural networks and still computes tight bounds across a variety of networks. We do this by exploiting structure in the dual of the semidefinite programming formulation proposed by Raghunathan et al. [2018b].

1.1 Related work

Verification is a computationally demanding process and has been shown to be NP-hard even for simple neural networks with a single hidden layer [Katz et al., 2017]. Complete verification algorithms rely on an exhaustive search of the input space in the worst case. Interesting progress has been made on complete verification algorithms using ideas from mixed-integer programming and satisfiability modulo theory [Katz et al., 2017, Tjeng and Tedrake, 2017, Bunel et al., 2017, Xiao et al., 2018]. These algorithms perform an exhaustive search in the worst case, but prune the search by propagating bounds or solving relaxed versions of the problem to rule out large parts of the search space – this enables efficient computation on many practical instances. Despite this progress, complete verification algorithms have not been applicable to models beyond small convolutional or fully-connected networks.

In contrast, sound but incomplete verification algorithms work by computing upper bounds on the worst case violation of the property being verified. Such algorithms have been derived based on ideas from abstract interpretation [Mirman et al., 2018], propagating bounds through the network [Gowal et al., 2018, Weng et al., 2018, Zhang et al., 2018], analyzing the Lipschitz properties of the network [Weng et al., 2018, Hein and Andriushchenko, 2017] and using convex optimization and duality theory [Raghunathan et al., 2018b, Kolter and Wong, 2017, Dvijotham et al., 2018]. These algorithms are scalable but the bounds computed by them can often be weak for general networks and networks have to be specially trained so that the computed bounds become tight [Wong et al., 2018, Gowal et al., 2018].

Raghunathan et al. [2018b] proposed a Semidefinite programming (SDP) approach to obtain tight bounds for neural network verification. Semidefinite programs (SDPs) were also used to verify nonlinear specifications in Qin et al. [2019]. SDPs produce tight bounds because they can capture correlations between neurons that linear programming (LP) based verification algorithms fail to. However, SDPs are computationally demanding and off-the-shelf solvers are inadequate for verifying even small to medium sized convolutional networks. In this paper, we develop new relaxations and algorithms to solve these relaxations that inherit the advantages of SDPs but can scale to networks two orders of magnitude larger.

1.2 Contributions

We study the problem of neural network verification for feedforward networks with ReLU activations. For such networks, the key contributions of this paper are:

A novel Lagrangian relaxation for neural network verification: We develop a Lagrangian relaxation of the neural network verification problem based on a semidefinite programming formulation from Raghunathan et al. [2018b]. By restricting the space over which the Lagrangian optimization is performed and exploiting the structure of the relaxation, we derive an efficient algorithm to solve a weakened version of the relaxation. The resulting algorithm improves upon previous incomplete verification algorithms based on linear programming or other bound propagation techniques [Dvijotham et al., 2018, Kolter and Wong, 2017, Wong et al., 2018, Weng et al., 2018] in terms of tightness while incurring only modest increases in computation time. Further, our approach can scale to networks two orders of magnitude larger than those considered in Raghunathan et al. [2018b].

Characterizing tightness of relaxations: We prove that if the network is locally linear in a sufficiently large region around the optimal adversarial example, the relaxation is tight. It has been observed [Moosavi-Dezfooli et al., 2018] that adversarial training techniques [Madry et al., 2018] indeed promote local linearity around the optimal adversarial example, justifying why the relaxation is likely to be tight for robust networks.

Efficient implementation amenable to deep learning frameworks: We take advantage of the structure of the proposed relaxation to reformulate it as a bound constrained optimization problem and solve it using a projected subgradient method (each iteration of which has a comparable computation cost as a forward/backward pass through the network). Our method only requires operations for which optimized implementations are easily available in deep learning frameworks like TensorFlow [Abadi et al., 2016] (convolutions, deconvolutions, matrix-vector products, etc.). With a straightforward implementation of this approach, we are able to improve the tightness of verification on large convolutional networks that were previously unverifiable.

2 PROBLEM FORMULATION AND SETUP

Notation: $[x]$ denotes the diagonal matrix with diagonal entries given by the vector x. $x \odot y$ denotes the element-wise product of two vectors (or matrices) x, y. We use x/y to denote element-wise division. $\succeq$ denotes the semidefinite ordering, so $X \succeq Y$ means that $X - Y$ is a symmetric positive semidefinite matrix. Throughout we denote $\mathbf{x}$ as the collection of all the activations at each layer in the network, i.e., $\mathbf{x} = (x_0, x_1, \ldots, x_K)$ for a K-layer network. The neural network is denoted by $\phi : \mathcal{R}^n \to \mathcal{R}^m$, where n is the dimension of the input to the network and m the dimension of the output of the network. $|x|$ denotes the element-wise absolute value of the matrix or vector x. $\mathbf{1}$ denotes the vector with every coordinate equal to 1 (with the size of the vector usually inferred from context).

Verification Problem: We consider verification problems of the form:

$$\psi\left(y\right) \leq 0 \quad \forall y : \, y = \phi\left(x\right), \; l \leq x \leq u, \tag{1}$$

where ψ denotes the specification, x the input to the network, y the output of the network and $[l, u]$ is the set of inputs of interest. We will assume that ψ is linear and in particular, we will focus on the adversarial specification: $\psi\left(y\right) = y_j - y_i$ where j denotes the target label and i denotes the true label.

2.1 Verification as optimization

We are interested in finding a counter-example to the specification (1) or proving that none exist. To this end, we study the optimization problem

$$\max_{l \leq x \leq u} \psi\left(\phi\left(x\right)\right). \tag{2}$$

The specification (1) is true if and only if the optimal value of this problem is smaller than 0.

2.2 Neural Network architecture

We consider networks ϕ with each layer being the composition of a linear operation and a ReLU layer. Thus we have $x_{k+1} = \mathrm{ReLU}\left(W_k x_k + b_k\right)$ with $x_0 = x$ and $y = x_K$ for a network with K layers. Here $W_k x_k + b_k$ can denote any linear operation – a fully connected linear layer, a convolutional layer, a batchnorm layer, an average pool layer or compositions of these. Then (2) can be rewritten as

$$\begin{aligned} \max \quad & \psi\left(x_K\right) \\ \text{s.t. } & x_{k+1} = \mathrm{ReLU}\left(W_k x_k + b_k\right), k = 0 \ldots K-1 \\ & l_0 \leq x_0 \leq u_0 \end{aligned} \tag{3}$$

Networks typically have a final linear layer that maps the output of the last ReLU layer to the final network output. This can be accommodated within the above formulation as follows: Let the last linear layer be $W_f x_K + b_f$ and the specification acting on this be $\psi\left(x\right)$. We define a modified specification as $\tilde{\psi}\left(x_K\right) = \psi\left(W_f x_K + b_k\right)$ (not that $\tilde{\psi}$ is also linear since it is a composition of linear functions).

We use bound propagation techniques [Weng et al., 2018, Mirman et al., 2018] to compute bounds on the intermediate activations given bounds on the inputs - we denote these bounds $l_k \leq x_k \leq u_k$ and define

$$\bar{x}_i = \frac{l_i + u_i}{2}, \epsilon_i = \frac{u_i - l_i}{2}$$

2.3 ReLU as quadratic and linear constraints

As observed in [Raghunathan et al., 2018b], we note that the constraint $x = \mathrm{ReLU}\left(y\right)$ can be rewritten as a set of quadratic constraints:

$$x \odot x = x \odot y, \quad x \geq y, \quad x \geq 0. \tag{4}$$

The first constraint ensures that x is either equal to 0 or y and the inequalities ensure that $x = 0$ when $y \leq 0$ and $x = y$ when $y > 0$. Using this, we can rewrite (3) as follows

$$\max_{l_k \leq x_k \leq u_k} \psi\left(x_K\right) \tag{5a}$$

$$\text{s.t. } x_{k+1} \odot x_{k+1} = x_{k+1} \odot \left(W_k x_k + b_k\right) \tag{5b}$$

$$\left(x_k - \bar{x}_k\right) \odot \left(x_k - \bar{x}_k\right) \leq \epsilon_k \odot \epsilon_k \tag{5c}$$

$$x_{k+1} \geq W_k x_k + b_k \tag{5d}$$

$$x_{k+1} \geq 0 \tag{5e}$$

3 LAGRANGIAN RELAXATION OF VERIFICATION PROBLEM

A natural algorithm for solving (2) is to use Projected Gradient Descent (PGD). Such an algorithm can only be expected to converge to a locally optimal solution of (2) which renders the approach unsuitable for verification, since it is unclear how far the local optimum found by PGD is from the true global optimum. In contrast, convex relaxation approaches study a relaxed version of (3) (which is a reformulation of (2)) by allowing the internal activations of the network to deviate from the precise values - this enlarges the set over which the optimization is performed to larger set (hence over-estimating the maximum value), but makes computation of the global optimum feasible using convex optimization techniques. In this section, we develop a convex relaxation of (3) and present conditions under which this relaxation is tight, ie, it has the same optimal value as (3).

3.1 Lagrangian relaxation for single hidden layer

For clarity in presentation, we first start with a single hidden layer neural network with $x_1 = \mathrm{ReLU}(Wx_0 + b)$ and a specification $\psi(x_1)$. Calculating the exact optimal value of (5) (which is a reformulation (3)) is difficult due to the nonlinear and nonconvex constraints. Hence, we construct a Lagrangian relaxation of this problem:

$$\begin{aligned}\max_{\substack{l_0 \le x_0 \le u_0 \\ l_1 \le x_1 \le u_1}} \psi\left(x_1\right) + {\lambda_1}^T \left(x_1 \odot \left(Wx_0 + b\right) - x_1 \odot x_1\right) \\ + \sum_{k=0}^{1} {\nu_k}^T \left(\epsilon_k \odot \epsilon_k - \left(x_k - \bar{x}_k\right) \odot \left(x_k - \bar{x}_k\right)\right) \\ + {\mu^+}^T \left(x_1 - Wx_0 - b\right) + {\mu^-}^T x_1 \qquad (6)\end{aligned}$$

where λ, ν are dual variables and $\nu_0, \nu_1, \mu^+, \mu^- \geq 0$. We can strengthen the relaxation by adding additional linear constraints between x_0, x_1 (details in Appendix C.1). We denote the collection of all dual variables by $\boldsymbol{\alpha} = \{\lambda, \nu_0, \nu_1, \mu^+, \mu^-\}$ and the Lagrangian function by $L\left(\mathbf{x}, \boldsymbol{\alpha}\right)$.

By weak duality [Boyd and Vandenberghe, 2004], the optimal value of (6) is an upper bound on the optimal value of the problem (5) (and hence (3)). However, (6) may be a non-convex optimization problem and hence difficult to solve. Thus, we study when this problem becomes convex. The Hessian of (6) with respect to (x_1, x_0) is equal to

$$\begin{pmatrix} -\left[2(\lambda_1 + \nu_1)\right] & \left[\lambda_1\right] W \\ W^T \left[\lambda_1\right] & -\left[2\nu_0\right] \end{pmatrix}.$$

If this matrix is negative semidefinite, the objective function of (6) is concave and hence the inner maximization is a convex optimization problem and can be solved efficiently. Thus, one can solve the following convex-concave saddle point problem to find an upper bound on (2):

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \max_{\substack{l_0 \le x_0 \le u_0 \\ l_1 \le x_1 \le u_1}} L\left(\mathbf{x}, \boldsymbol{\alpha}\right) \\ &\text{s.t.} \begin{pmatrix} \left[2\left(\lambda_1 + \nu_1\right)\right] & -\left[\lambda_1\right] W \\ -W^T \left[\lambda_1\right] & \left[2\nu_0\right] \end{pmatrix} \succeq 0, \\ &\nu_1, \nu_0 \geq 0, \mu^+, \mu^- \geq 0 \qquad (7)\end{aligned}$$

3.2 Conditions for Exact Verification

We now study conditions under which the relaxation (7) is tight, i.e., has the same optimal value as (2). To this end, we define the following quantities:

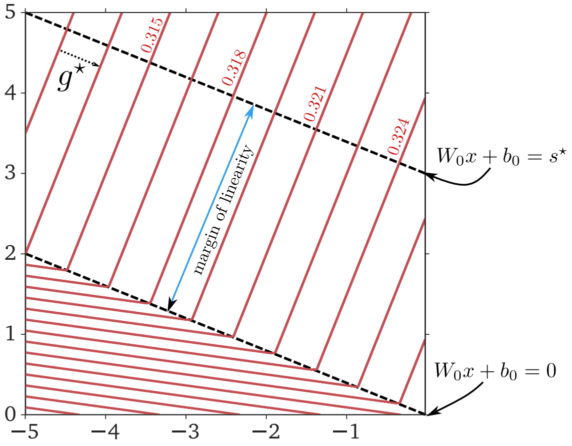

Figure 1: This is a contour plot of the specification with respect to the 2D input space. Here the specification is $\psi(\phi(x)) = c^T\phi(x)$, $c = (1, 0, \cdots, 0)$ and $\phi(x) = \mathrm{ReLU}(W_1\mathrm{ReLU}(W_0x + b_0) + b_1)$. The distance between the contours where $W_0x + b = 0$ and $W_0x + b = s^\star$ is one of the margins of linearity. $g^\star$ is a vector perpendicular to the contours of the specification.

Definition 3.1 (Locally linear point)**.** We say that x' is a *locally linear point* if $\exists\ g \in \mathcal{R}^n$ and $\delta > 0$ such that

$$\psi\left(\phi\left(x\right)\right) = \psi\left(\phi\left(x'\right)\right) + g^T\left(x - x'\right),$$

for all $\{x : \|x - x'\| \leq \delta\}$.

Note that x_0 is a locally linear point if

$$|W_k x_k + b_k| > 0$$

for each k. We note that it has been observed that adversarial training makes the neighbourhood around the nominal point linear [Moosavi-Dezfooli et al., 2018], making it likely that this condition is indeed true for networks trained to be robust.

Definition 3.2 (Margins of linearity)**.** Given a ReLU activated network with weights W_i for $i = 0, \ldots, K-1$, let $s_k = |W_k x_k + b_k|$. The *margins of linearity* of the point x_0 is a set of vectors given by:

$$\mathbf{s} = \{s_k\}_{k=0}^{K-1}.$$

Figure 1 shows the contours/level-sets (lines in red) of $\psi(\phi(x))$ where $x \in \mathcal{R}^2$. In this figure, we show $s^\star$ is the distance which needs to be traveled such that we hit a switching state.

To study when the relaxation (7) is tight, we study conditions for a locally optimal solution to (3) to be globally optimal, and we state these conditions in terms of the concepts defined above.

Informally, our results can be stated as follows:
A locally optimal solution to (3) *is also globally optimal if it is a locally linear point with sufficiently large margins of linearity.*

The following theorem makes this more concrete.

Theorem 1. Let $\psi(y) = c^T y$ and let $x_0^\star$ be a locally optimal solution to (3). Suppose that $x_0^\star$ is also a locally linear point with margins of linearity $\mathbf{s} = \{s^\star\}$ and let $g^\star = \frac{\partial \psi(\phi(x_0))}{\partial x_0}\Big|_{x_0 = x_0^\star}$. Then, if

$$\left[\frac{|g^\star|}{\epsilon_0}\right] - W^T \left[\frac{\max(c, 0)}{2s^\star}\right] W \succeq 0 \tag{8}$$

$x_0^\star$ is globally optimal and the optimal values of (7) and (3) coincide.

Proof. Refer to Appendix B. □

Interpretation of the condition: A simpler sufficient condition that implies (8) is

$$\frac{|g^\star|}{\epsilon_0} \geq (\sigma_{\max}(W))^2 \left\|\frac{\max(c,0)}{2s^\star}\right\|_\infty,$$

where $\sigma_{\max}$ denotes the largest singular value. We can restate this in terms of the "normalized" gradient and margin:

$$\tilde{g} = \frac{|g^\star|}{\sigma_{\max}(W)}, \quad \tilde{s} = \frac{s^\star}{\sigma_{\max}(W)}.$$

In terms of these quantities, the condition is stated as

$$\frac{\tilde{g}}{\epsilon_0} \geq \left\|\frac{\max(c,0)}{2\tilde{s}}\right\|_\infty.$$

If, for all i, $c_i < 0$ then $\psi(\phi(x))$ is a concave function of x. Consequently, this optimization problem (3) is free of local optima and can be solved efficiently to the global optimal solution. However, if there exists $c_j > 0$, those elements contribute to the nonconvexity of the problem. Thus, the condition for exactness can be interpreted as

$$\begin{aligned} &\text{Normalized gradient } \times \text{Normalized margin of linearity} \\ &\geq \text{Degree of nonconvexity } \times \text{Perturbation size} \end{aligned}$$

3.3 Extension to Deep Networks

The Lagrangian relaxation (6) can be extended to deeper networks with multiple hidden layers(for more details see Appendix C.1). Similar to the single hidden layer case, we denote the collection of dual variables by $\boldsymbol{\alpha}$, the Lagrangian by $L(\mathbf{x}, \boldsymbol{\alpha})$ and the Hessian of the Lagrangian with respect to $\mathbf{x}$ by $H(\boldsymbol{\alpha})$. Then, under the constraint $H(\boldsymbol{\alpha}) \preceq 0$, we can pose the following convex optimization problem:

$$\min_{\boldsymbol{\alpha}} \max_{\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}} L(\mathbf{x}, \boldsymbol{\alpha}) \tag{9a}$$

$$\text{s.t} \quad H(\boldsymbol{\alpha}) \preceq 0, \mu, \nu \geq 0. \tag{9b}$$

Similar to Theorem 1, we can establish conditions under which the optimal value of (9) coincides with that of (3), as follows.

Theorem 2. Let $x_0{}^\star$ denote a locally optimal point for (3). Let $x_k^\star$ denote the activations of the layer k corresponding to the input $x_0^\star$. Suppose that $x_0^\star$ is a locally linear point with margins of linearity and gradients

$$\mathbf{s}^\star = \{s_k^\star\}_{k=0}^{K-1}, g_k^\star = \frac{\partial \psi}{\partial x_k}\Big|_{\mathbf{x} = \tilde{\mathbf{x}}^\star}.$$

Suppose further that $\exists \{\theta_k, \kappa_k\}_{k=1}^{K}$ such that $0 < \theta_k < 1, \kappa_k \geq 0$ and the following conditions are satisfied:

$$\begin{aligned} &\left[\frac{\max\left(g_{k-1}^\star, \kappa_{k-1}\right)}{s_{k-1}^\star}\right] \\ &- W_{k-1}^T \left[\frac{\max\left(g_k^\star, \kappa_k\right)}{4\theta_k(1-\theta_{k-1}) s_k^\star}\right] W_{k-1} \succeq 0 \qquad \text{(10a)} \\ &\qquad \text{for } k = 2, \ldots, K \end{aligned}$$

$$\left[\frac{|g_0^\star|}{\epsilon_0}\right] - W_0^T \left[\frac{\max\left(g_1^\star, \kappa_1\right)}{2\theta_1 s_1^\star}\right] W_0 \succeq 0 \tag{10b}$$

Then $x_0^\star$ is globally optimal for (3) and the optimal values of (9) and (3) coincide.

Proof. See appendix section C.1. □

This condition has a similar interpretation as that of theorem 1, since the condition requires the normalized gradient to be large enough relative to the margin of linearity. In this theorem, the choice of κ_k, θ_k for the intermediate layers may be important since it controls the trade-off between how easy it is to satisfy (10) at layer k and $k+1$.

4 EFFICIENT OPTIMIZATION OF THE RELAXATION

The optimization problem (9) can be rephrased as a semidefinite programming problem and solved using an off-the-shelf SDP solver. While this is theoretically efficient (polynomial time), in practice, semidefinite programs are too expensive to solve for deep neural networks. Thus, it is of interest to develop an customized algorithm capable of solving (9) efficiently.

The constrained optimization problem shown in (9) requires us to solve min-max optimization problem. We reduce this to purely a constrained minimization problem in Section 4.1. In Section 4.2, we show that the constraint $H(\boldsymbol{\alpha}) \preceq 0$ can be replaced by an equivalent set of smaller PSD constraints by introducing auxiliary variables and finally in section 4.3, we restrict the space of auxiliary variables to obtain a weaker formulation that can be solved efficinetly using a projected gradient method.

4.1 Reducing the Inner Maximization

Under the constraint (9b), L is a concave function of $\mathbf{x}, \tilde{\mathbf{x}}$. Thus, for any $\tilde{\mathbf{x}}$, we have

$$L(\tilde{\mathbf{x}}, \boldsymbol{\alpha}) \leq L(\mathbf{x}, \boldsymbol{\alpha}) + (\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}))^T (\tilde{\mathbf{x}} - \mathbf{x}).$$

The RHS is a linear function of $\tilde{\mathbf{x}}$ and can be maximized with respect to $\tilde{\mathbf{x}}$ in closed form to obtain:

$$L(\mathbf{x}, \boldsymbol{\alpha}) + \mathbf{1}^T \max(\nabla L \odot (\mathbf{l} - \mathbf{x}), \nabla L \odot (\mathbf{u} - \mathbf{x})).$$

where $\nabla L = \nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha})$. We denote the above expression by $G(\mathbf{x}, \boldsymbol{\alpha})$. Using the concavity of L in $\mathbf{x}$, it can be shown that (Appendix D)

$$\min_{\mathbf{x}} G(\mathbf{x}, \boldsymbol{\alpha}) \equiv \max_{\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}} L(\mathbf{x}, \boldsymbol{\alpha}).$$

Consequently, (9) can be reformulated as:

$$\begin{aligned} &\min_{\mathbf{x}, \boldsymbol{\alpha}} \quad G(\mathbf{x}, \boldsymbol{\alpha}) \\ &\text{s.t. } H(\boldsymbol{\alpha}) \preceq 0 \text{ (PSD constraint)}, \mu, \nu \geq 0 \end{aligned} \tag{11}$$

4.2 Efficiently handling the PSD constraint

We now discuss how one could replace the constraint $H(\boldsymbol{\alpha}) \preceq 0$ with a simpler condition. To begin, we exploit the structure of the matrix $H(\boldsymbol{\alpha})$ to rewrite this large matrix inequality as a set of smaller matrix inequalities. To motivate the construction, consider a network with two hidden layers: in this case, $-H(\boldsymbol{\alpha})$ evaluates to

$$\begin{pmatrix} A_2 & -B_2 & 0 \\ -B_2^T & A_1 & -B_1 \\ 0 & -B_1^T & A_0 \end{pmatrix}$$

where $A_k = 2[\lambda_k + \nu_k]$ (with $\lambda_0 = 0$) and $B_k = [\lambda_k] W_{k-1}$. Rather than enforcing a PSD constraint on this large matrix, we can decompose this matrix as a sum of simpler matrices:

$$\begin{pmatrix} A_2 - \Delta_2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} \Delta_2 & -B_2 & 0 \\ -B_2^T & A_1 - \Delta_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & \Delta_1 & -B_1 \\ 0 & -B_1^T & A_0 \end{pmatrix}.$$

for some values of Δ_2, Δ_1. If each matrix in the above sum is PSD, then so is $H(\boldsymbol{\alpha})$. Remarkably, this is a necessary and sufficient condition as shown in the following theorem. This enables us to rewrite (11) using simpler PSD constraints without affecting the optimal value.

Theorem 3. $H(\boldsymbol{\alpha}) \preceq 0$ if and only if there exist matrices $\Delta_K, \ldots, \Delta_1 \succeq 0$ such that

$$[(\lambda_K + \nu_K)] - \Delta_K \succeq 0 \tag{12a}$$

$$\begin{pmatrix} \Delta_k & -\frac{[\lambda_k] W_{k-1}}{2} \\ -\frac{W_{k-1}^T [\lambda_k]}{2} & [(\lambda_{k-1} + \nu_{k-1})] - \Delta_{k-1} \end{pmatrix} \succeq 0 \quad k = 1, \ldots, K \tag{12b}$$

where $\lambda_0 = 0, \Delta_0 = 0$. Further, the problem

$$\min_{\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \boldsymbol{\alpha}, \Delta} G(\mathbf{x}, \boldsymbol{\alpha}) \tag{13a}$$

$$\text{s.t. } (12), \mu, \nu \geq 0 \tag{13b}$$

has the same optimal value as (9).

Proof. See Appendix D □

4.3 Scalable optimization via diagonal dominance

The transformation from (22) to (13) does not impact the quality of the Lagrangian relaxation (theorem 3 guarantees this). However, the resulting optimization problem still requires maintaining the matrix variables Δ_k – the storage requirements for these matrices grows quadratically in the size of each layer rendering the approach unsuitable for large neural networks. Further, enforcing the semidefinite constraints in (13) requires computing eigenvalues of matrices of size equal to the size of each layer in the neural network. We simplify the optimization (13) using the notion of diagonal dominance [Ahmadi and Majumdar, 2019] as follows:

- We choose $\Delta_k = [\delta_k]$ with $\delta_k \geq 0$.

- We replace the condition (12) with the condition

  $$\nu_{k-1} \geq \delta_{k-1} - \lambda_{k-1} + \tfrac{1}{4} \left| W_{k-1}^T \left[\tfrac{\lambda_k \odot \lambda_k}{\delta_k} \right] W_{k-1} \right| \mathbf{1} \tag{14}$$

  which ensures that the matrix

  $$[(\lambda_{k-1} + \nu_{k-1})] - \Delta_{k-1} - \frac{1}{4} W_{k-1}^T [\lambda_k] (\Delta_k)^{-1} [\lambda_k] W_{k-1}$$

  is diagonally dominant and hence positive definite. Hence, by Schur complements, the matrices in (12) are also PSD. Since eigenvalues are invariant to similarity transformations, we can apply the diagonal dominance condition (14) after rescaling by a diagonal matrix $[\beta_k]$ with $\beta_k > 0$ and optimize over β_k as well (we defer the details of this re-scaling to the appendix D.3).

With these simplifications, we can set

$$\nu_k = \max\left(\delta_k - \lambda_k + \tfrac{1}{4}\left|W_k^T\left[\tfrac{\lambda_{k+1}\odot\lambda_{k+1}}{\delta_{k+1}}\right]W_k\right|\mathbf{1}, 0\right) \tag{15}$$

for any choice of $\delta_k \geq 0$ and any choice of λ_k so that the constraints of (13) are satisfied. We denote this choice by $\nu(\lambda, \delta)$ and the resulting collection of dual variables by $\boldsymbol{\alpha}(\lambda, \delta, \mu)$. With this choice (13) reduces to

$$\min_{\delta\geq 0, \mu\geq 0, \mathbf{x}, \lambda} G\left(\mathbf{x}, \boldsymbol{\alpha}\left(\lambda, \delta, \mu\right)\right) \tag{16}$$

This gives rise to Algorithm 1, which implements a projected subgradient method to optimize G with respect to $\delta, \mathbf{x}, \lambda, \mu$.

Theorem 4. Let ${x_0}^\star$ denote a locally optimal point for (3) and define all the quantities as in theorem 2. If the matrices in (10) are not just positive definite but also diagonally dominant, i.e.,

$$\begin{aligned}&\frac{\max\left(g^\star_{k-1}, \kappa_{k-1}\right)}{s^\star_{k-1}} \\ &\quad - \left|W_{k-1}^T\left[\frac{\max\left(g^\star_k, \kappa_k\right)}{4\theta_k(1-\theta_{k-1})s^\star_k}\right]W_{k-1}\right|\mathbf{1} \geq 0 \quad \text{(17a)} \\ &\quad\quad \text{for } k = 2, \ldots, K \\ &\frac{|g^\star_0|}{\epsilon_0} - \left|W_0^T\left[\frac{\max\left(g^\star_1, \kappa_1\right)}{2\theta_1 s^\star_1}\right]W_0\right|\mathbf{1} \geq 0 \quad \text{(17b)}\end{aligned}$$

then the optimal value of (16) coincides with that of (3).

Proof. See appendix D □

4.4 Computational complexity

Per-iteration complexity of PGD-SDP. Given the values of o_k, the computation of the objective (line 10) is comparable to the cost of doing a forward pass - we need to generate the activations and multiply simple functions of the activations at adjacent layers to compute the Lagrangian. The computation of o_k is technically $O(n_k^2 n_{k+1})$ where n_k is the size of the input to layer k. However, for convolutional layers (which are typically the largest layers in the network), this computation can be done in time $O(w_k^2 n_{k+1})$ where w_k is the size (height×width×input channels) of the kernel.

Algorithm 1 PGD-SDP

```
Input: ReLU network with weights $\{W_k, b_k\}_{k=0}^{K-1}$
 1: Initialize variables $\mathbf{x}, \delta, \lambda$.
 2: Set learning rate $r$.
 3: Set number of optimization steps $N$.
 4: for all $i \in \{0, 1, \ldots, N\}$ do
 5:     for all $k \in \{0, 1, \ldots, K-1\}$ (loop over the layers) do
 6:         $\zeta_k \leftarrow \left[\frac{\lambda_{k+1}\odot\lambda_{k+1}}{\delta_{k+1}}\right]$
 7:         $o_k \leftarrow \frac{\left|W_k^T \zeta_k W_k\right|\mathbf{1}}{4}$
 8:         $\nu_k \leftarrow \mathrm{ReLU}\left(\delta_k - \lambda_k + o_k\right)$
 9:     end for
10:     Generate objective value $G\left(\mathbf{x}, \boldsymbol{\alpha}\left(\lambda, \delta, \mu\right)\right)$.
11:     for all $v \in \{\mathbf{x}, \delta, \lambda, \mu\}$ (update variables) do
12:         $v \leftarrow v - r\frac{\partial G}{\partial v}$
13:     end for
14:     for all $v \in \{\delta, \mu\}$ (nonnegativity constraints) do
15:         $v \leftarrow \mathrm{ReLU}\left(v\right)$
16:     end for
17:     if $G\left(\mathbf{x}, \boldsymbol{\alpha}\right) \leq 0$ then
18:         Break
19:     end if
20: end for
21: if $G\left(\mathbf{x}, \boldsymbol{\alpha}\right) \leq 0$ then
22:     Return Verified
23: else
24:     Return Failed to Verify
25: end if
```

Finally, the gradients with respect to the optimization variables can be done using a single backpropapagation (to compute the gradients w.r.t. $\mathbf{x}$) and in constant additional time (to compute the gradients w.r.t. $\mathbf{x}, \mu, \delta, \lambda$). Thus, the overall per-iteration cost is equivalent to the cost of doing a fixed number of forward/backward propagations through the network.

Number of iterations. The standard convergence rate for the subgradient methods applies to Algorithm 1, i.e., the algorithm achieves an ϵ-suboptimal solution to (16) in $O(\frac{1}{\epsilon^2})$ iterations [Nesterov, 2018]. In practice, the number of iterations depends on the nature of the network and the specification - we find a large variance in the number of iterations in our experiments – this can range from a few hundred steps to tens of thousands of steps and we do not find a strong correlation with the size of the network.

5 EXPERIMENTS

We evaluate our verification algorithm on several image classification networks on MNIST and CIFAR-10. For the sake of easy comparison with prior work, we focus on the specification of robustness to perturbations of the input in the ℓ_∞ norm. We report three metrics for each network: the *nominal error rate* (the fraction of test examples classified incorrectly), the *adversarial error rate* (the fraction of test examples classified incorrectly under a projected gradient attack) and the *verified error rate* (the fraction of test examples for which a verification algorithm was unable to rule out the existence of adversarial examples). We report the verified error rate computed using different verification algorithms.

For PGD-SDP, we used the Adam optimizer [Kingma and Ba, 2015] with an initial learning rate of 10^{-3} and a decay schedule where the learning rate was reduced by .8 every 1000 iterations. We ran Adam for 100000 iterations to obtain the results, stopping early whenever the example is verified. We use the FAST-LIN bound propagation algorithm [Weng et al., 2018] - note that our algorithm can be used with any bound propagation method and methods that compute tighter bounds (like [Zhang et al., 2018])

5.1 Networks trained

It has been previously shown that networks trained with losses related to the verification problem (typically an efficiently computable upper bound on specification) can be verified more easily using a cheap verification algorithm [Raghunathan et al., 2018a, Kolter and Wong, 2017, Wong et al., 2018, Gowal et al., 2018, Mirman et al., 2018]. However, in the process of promoting verifiability, these additional losses tend to over-regularize networks leading to significant increase in the nominal error rate. Indeed, as reported in [Kolter and Wong, 2017, Gowal et al., 2018], the convolutional kernels learned for models trained with these losses are significantly sparser. Thus, these models under-utilize available network capacity.

In [Raghunathan et al., 2018b], it was shown that the SDP relaxation is capable of verifying so-called *foreign networks*, i.e., networks that are not trained with a loss related to the SDP bound on the specification. Since our approach is closely related to the SDP approach, we indeed observe the same phenomenon. Note that, since we solve a more restricted formulation (Algorithm 1), our bounds are not as tight on some networks. However, our scalable approach applies to convolutional models that are two order of magnitude larger than those considered in [Raghunathan et al., 2018b]. For these larger models, we indeed find that we can obtain significant improvements in verified accuracy compared to the other scalable relaxations [Dvijotham et al., 2018, Kolter and Wong, 2017, Ehlers, 2017]. In order to demonstrate this, we train a range of models with a mixture of the regular cross-entropy loss, the PGD adversarial loss from [Madry et al., 2018] and the Interval Bound Propagation (IBP) loss from [Gowal et al., 2018]. Unless stated otherwise, we always set the weight of the cross-entropy loss to 0.5, the weight of the adversarial loss to 0.5 and vary the weight of the IBP loss. As we increase the emphasis on verifiability of these networks (i.e., by increasing the weight of the IBP loss), we obverse that gap in verified error rates between the LP and SDP formulation shrinks.

5.2 Benchmarking experiments

We study the three networks used in [Raghunathan et al., 2018b], and compute a verified bound on the adversarial error rate under ℓ_∞ perturbations of size $\epsilon = 0.1$, averaged over the MNIST test set of 10,000 examples. The three networks used are called LP-NN, Grad-NN and PGD-NN, after the techniques used to train them (the LP based technique from [Wong et al., 2018], the gradient bounding technique [Raghunathan et al., 2018a] and adversarial training [Madry et al., 2018]).

The results in Table 1 show that our approach outperforms both the SDP approach from [Raghunathan et al., 2018b] and the LP approach originally described in [Ehlers, 2017] (and closely related to [Kolter and Wong, 2017, Dvijotham et al., 2018]) for the first two networks. For the final network (which is deeper), the technique from [Raghunathan et al., 2018a] produces a better bound. In terms of solution time, our approach is much faster and only takes about $130s$ second per example (on average) while the approach from [Raghunathan et al., 2018a] takes nearly 8100 seconds. We plot the change in the verified error rate as a function of computation time in figure 2. Figure 3 shows a cumulative histogram (over the test examples) of the relative improvement (in %)from the LP to PGD-SDP in the maximum radius that can be verified to be adversarially robust - so for about 40% of the examples, PGD-SDP improves the maximum radius and for 10% the improvement is over 10%.

Model	**PGD (lower bound)**	**PGD-SDP**	**LP**	**SDP-IP**
Grad-NN	14.43%	**16.32%**	97%	18%[1]
LP-NN	18.73%	**18.97%**	22%	20%[1]
PGD-NN	9%	67%	78%	**20%**[1]

Table 1: Verified error rate as computed by our method (PGD-SDP) compared to the LP approach from [Ehlers, 2017] and the SDP approach from [Raghunathan et al., 2018b]

[1] Evaluated on a random subset of a 1000 test examples

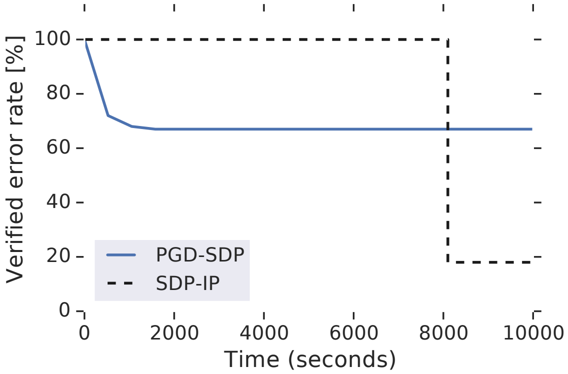

Figure 2: Comparison of SDP-IP and PGD-SDP verified error rate as a function of runtime on PGD-NN. The average time of verification using PGD-SDP is 130s while the average under SDP-IP is 8100s.

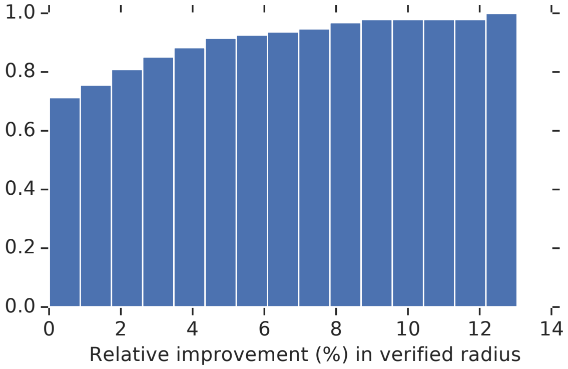

Figure 3: Cumulative histogram of relative percentage improvement in verified radius using PGD-SDP vs LP on the PGD-NN model. For 10% of the test examples, the certified radius improves by at least 10%.

5.3 Large-scale experiments with deep convolutional networks

In order to demonstrate the scalability of PGD-SDP, we conduct experiments on larger convolutional models trained on CIFAR-10. Remember that we train with a mixture of regular cross-entropy loss, adversarial PGD loss and IBP loss and vary the relative weighting to obtain a spectrum of networks. We experiment with both small and medium sized convolutional neural networks. The small network has 8.3K hidden units, while the medium network has 47K hidden units (see appendix E).

The results (Table 2) show that as the networks get larger, the gains PGD-SDP obtains over the LP grow. For the small convnet, PGD-SDP verification takes about 100s per example, while for the larger networks (which are nearly ten times larger), it takes roughly a 1000s, indicating a linear scaling with the size of the network.

Nominal error	**PGD (lower bound)**	**PGD-SDP**	**LP**
	small models		
26.1%	45.3%	95.5%	97%
24.5%	49.1%	57.3%	57.9%
	medium models		
29.27%	46.74%	65.4%	71.3%
26.3%	43.1%	81.7%	87.2%
24.5%	43.2%	85.4%	91.1%

Table 2: Comparison of LP and PGD-SDP on small and medium-sized CIFAR-10 convolutional networks (perturbation size $2/255$). For both small and medium models, as one goes down the table, the weight on the IBP loss while training decreases thereby making the networks harder to verify.

6 CONCLUSIONS

The twin challenges of developing both scalable and tight verification algorithms for neural networks has remained elusive despite significant progress in recent years. In this paper, we formulated a convex relaxation for neural network verification that is provably tight under natural assumptions. We developed efficient algorithms to solve the relaxation with per-iteration complexity comparable to a small number of forward/backward passes through the network. The resulting algorithm is able to compute rigorous bounds on the worst case adversarial loss, and in practice is often quite tight even for networks that have not been explicitly trained to be verifiable. This enables us to significantly improve the verifiability vs nominal accuracy tradeoff on several models. We hope that our work will inspire further research into tighter relaxations on larger models and richer specifications.

Acknowledgements

We thank Brendan O'Donoghue for reviewing this paper and providing valuable feedback. We thank Aditi Raghunathan for helpful comments on this paper and for sharing the networks used for the experiments in section 5.2.

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operat-*

ing Systems Design and Implementation ({OSDI} 16), pages 265–283, 2016.

A. A. Ahmadi and A. Majumdar. Dsos and sdsos optimization: more tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebra and Geometry*, 3(2):193–230, 2019.

A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, 2018.

S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar. Piecewise linear neural network verification: A comparative study. *arXiv preprint arXiv:1711.00455*, 2017.

K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. Towards scalable verification of neural networks: A dual approach. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Automated Technology for Verification and Analysis, International Symposium on*, 2017.

S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

M. Hein and M. Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*, pages 2263–2273, 2017.

G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

J. Z. Kolter and E. Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

M. Mirman, T. Gehr, and M. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.

S.-M. Moosavi-Dezfooli, A. Fawzi, J. Uesato, and P. Frossard. Robustness via curvature regularization, and vice versa. *arXiv preprint arXiv:1811.09716*, 2018.

Y. Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2018.

C. Qin, K. D. Dvijotham, B. O'Donoghue, R. Bunel, R. Stanforth, S. Gowal, J. Uesato, G. Swirszcz, and P. Kohli. Verification of non-linear specifications for neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyeFAsRctQ`.

A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018a. URL `https://openreview.net/forum?id=Bys4ob-Rb`.

A. Raghunathan, J. Steinhardt, and P. S. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10900–10910, 2018b.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

V. Tjeng and R. Tedrake. Verifying neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

J. Uesato, B. O'Donoghue, A. van den Oord, and P. Kohli. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, 2018.

L. Vandenberghe, M. S. Andersen, et al. Chordal graphs and semidefinite optimization. *Foundations and Trends® in Optimization*, 1(4):241–433, 2015.

T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, pages 8410–8419, 2018.

K. Y. Xiao, V. Tjeng, N. M. Shafiullah, and A. Madry. Training for faster adversarial robustness verification via inducing relu stability. *arXiv preprint arXiv:1809.03008*, 2018.

H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pages 4939–4948, 2018.