SDP Verifiers for Deep Neural Networks

I. Preliminaries

A. Notations

$[n]$ stands for the set $\{1, 2, \ldots, n\}$. $\mathbb{S}^n$ is the set of all $n \times n$ real symmetric matrices. For a $n \times n$ matrix A, its trace is defined as $\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}$. For vector v, $v \succeq 0$ means every element is non-negative, i.e., $\forall v_i \geq 0$. For matrix M, $M \succeq 0$ means it is a positive semi-definite matrix, i.e., $M \succeq 0$. The Kronecker product is the elementwise product of two matrix, which is denoted by $A \otimes B = C$ where $C_{ij} = A_{ij}B_{ij}$. The diagonal of matrix is denoted by $\mathrm{diag}(\cdot)$ or $\mathrm{d}(\cdot)$.

B. General Duality Theorem

Suppose we have an optimization problem, which may not be convex:

$$\begin{aligned} & \text{minimize}_x \ f_0(x), \\ \text{s.t.} \quad & f_i(x) \leq 0, i \in [m], \\ & h_i(x) = 0, i \in [p]. \end{aligned} \tag{1}$$

And we represent the legal set $\{x : f_i(x) \leq 0, h_j(x) = 0, i \in [m], j \in [p]\}$ as $\mathcal{D}$.

The optimal solution is p^*.

We introduce two additional variables $\lambda \in \mathbb{R}^m$, and $\nu \in \mathbb{R}^p$. And define the function

$$\Lambda(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x). \tag{2}$$

Define $g(\lambda, \nu)$ and eliminate variable x, we yield the dual function

$$\begin{aligned} & g(\lambda, \nu) = \inf_{x \in \mathcal{D}} \Lambda(x, \lambda, \nu) \\ & \text{s.t.} \qquad \lambda \succeq 0. \end{aligned} \tag{3}$$

The optimization problem defined on g as below is called *Lagrange dual problem*:

$$\begin{aligned} & \text{maximize}_{\lambda, \nu} \ g(\lambda, \nu) \\ & \text{s.t.} \qquad \lambda \succeq 0. \end{aligned} \tag{4}$$

It is a convex optimization problem. The optimal solution is d^*.

Easily seen, for any $\lambda \succeq 0$ and ν,

$$\begin{aligned} g(\lambda, \nu) &= \inf_{x \in \mathcal{D}} \Lambda(x, \lambda, \nu) \\ &= \inf_{x \in \mathcal{D}} f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \\ &\leq \inf_{x \in \mathcal{D}} f_0(x) = p^*. \end{aligned} \tag{5}$$

The last inequity follows $\lambda_i f_i(x) \leq 0$ and $h_i(x) = 0$ for any $\lambda \succeq 0$ and $x \in \mathcal{D}$.

Thus, we always have $d^* \leq p^*$. This property is called *weak duality*. When $d^* = p^*$, we call that *strong duality* holds.

C. SDP Problem & Its Dual Form

1) Trace Lemma:

Lemma I.1. *Given symmetric matrix $A, B \in \mathbb{S}^k$. If A, B are non-negative, i.e., $A, B \succeq 0$. Then $\mathrm{tr}(A^\top B) \geq 0$.*

Proof. A property of trace operator is that, for square matrix A, B, $\mathrm{tr}(AB) = \mathrm{tr}(BA)$.

Apply diagonalization, we get

$$\mathrm{tr}(A^\top B) = \mathrm{tr}((P^{-1}D_1P)^\top (Q^{-1}D_2Q)), \tag{6}$$

where D_1 and D_2 are diagonal matrices, and P and Q are orthogonal matrices. As $A, B \succeq 0$, we know that the diagonal matrix D_1 and D_2 are all non-negative, i.e., each element $d_{1,i} \geq 0, d_{2,i} \geq 0$ for $1 \leq i \leq n$.

From Eq. 6, we yield

$$\mathrm{tr}((P^{-1}D_1P)^\top (Q^{-1}D_2Q)) = \mathrm{tr}(P^{-1}D_1PQ^{-1}D_2Q) = \mathrm{tr}(D_1PQ^{-1}D_2QP^{-1}) = \sum_{i=1}^{n} d_{1,i}(PQ^{-1}D_2QP^{-1})_{ii}. \tag{7}$$

Define $T = QP^{-1}$, then $T^\top = (QP^{-1})^\top = PQ^{-1}$, therefore $PQ^{-1}D_2QP^{-1} = T^\top D_2 T$. Consider $(T^\top D_2 T)_{ii}$, by expanding it we get

$$(T^\top D_2 T)_{ii} = \sum_{j=1}^{n} (T^\top D_2)_{ij} t_{ji} = \sum_{j=1}^{n}\sum_{k=1}^{n} t_{ki} d_{2,kj} \cdot t_{ji} = \sum_{j=1}^{n} t_{ji} d_{2,j} t_{ji} = \sum_{j=1}^{n} t_{ji}^2 d_{2,j}. \tag{8}$$

Obviously, from Eq. 8, $(T^\top D_2 T)_{ii} \geq 0$ for any $1 \leq i \leq n$. Combine this result with Eq. 6 and Eq. 7, we know $\mathrm{tr}(A^\top B) \geq 0$. □

2) SDP & LMI: The SDP (Semi-Definite Programming) problem is defined as follows:

$$\begin{aligned} &\text{minimize}_{X \in \mathbb{S}^n} \ \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}, \\ &\text{s.t.} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} a_{kij}x_{ij} \leq b_k, k \in [m], \\ &\qquad X \succeq 0. \end{aligned} \tag{9}$$

The LMI (Linear Matrix Inequality) problem is:

$$\begin{aligned} &\text{minimize}_x \ c^\top x, \\ &\text{s.t.} \quad x_1F_1 + x_2F_2 + \cdots + x_nF_n + G \preceq 0, \\ &\qquad Ax = b, \end{aligned} \tag{10}$$

where $G, F_1, F_2, \ldots, F_n \in \mathbb{S}^k$.

3) LMI Dual: Following Section I-B, for LMI problem, we define the function

$$\Lambda(x, Z, v) = c^\top x + \mathrm{tr}\left((x_1F_1 + x_2F_2 + \cdots + x_nF_n + G)^\top Z\right) + v^\top(Ax - b). \tag{11}$$

The Lagrange dual function is

$$
\begin{aligned}
g(Z,v) &= \inf_{x\in\mathcal{D}} \Lambda(x,Z,v) \\
&= \inf_{x\in\mathcal{D}} c^\top x + \operatorname{tr}\left(\left(x_1F_1 + x_2F_2 + \cdots + x_nF_n + G\right)^\top Z\right) + v^\top(Ax-b) \\
&= \inf_{x\in\mathcal{D}} x_1\left(c_1 + \operatorname{tr}(F_1^\top Z)\right) + x_2\left(c_2 + \operatorname{tr}(F_2^\top Z)\right) + \cdots + x_n\left(c_n + \operatorname{tr}(F_n^\top Z)\right) + \operatorname{tr}(G^\top Z) + v^\top(Ax-b) \\
&= \inf_{x\in\mathcal{D}} x_1\left(c_1 + \operatorname{tr}(F_1^\top Z) + v^\top A_{:,1}\right) + x_2\left(c_2 + \operatorname{tr}(F_2^\top Z) + v^\top A_{:,2}\right) + \cdots + x_n\left(c_n + \operatorname{tr}(F_n^\top Z) + v^\top A_{:,n}\right) \\
&\quad + \operatorname{tr}(G^\top Z) - v^\top b, \\
\text{s.t.} \quad & Z \succeq 0.
\end{aligned}
\tag{12}
$$

Inject the expanded function to the Lagrange dual problem form:

$$
\begin{aligned}
&\text{maximize}_{Z,v}\ g(Z,v), \\
&\text{s.t.} \qquad Z \succeq 0.
\end{aligned}
\tag{13}
$$

We find the coefficients of $x_1,\ldots,x_n$ are all fixed for given Z,v, which means that if $\mathcal{D}$ is unbounded, $g(Z,v) = -\infty$. Therefore, the maximum value of $g(Z,v)$ is achieved when all these coefficients equal to 0, i.e., $c_i + \operatorname{tr}(F_i^\top Z) + v^\top A_{:,i} = 0$ for $1 \le i \le n$. As the result, the problem Eq. 13 is equivalent to

$$
\begin{aligned}
&\text{maximize}_{Z,v}\ \operatorname{tr}(G^\top Z) - v^\top b, \\
&\text{s.t.} \quad \begin{aligned} &Z \succeq 0, \\ &c_i + \operatorname{tr}(F_i^\top Z) + v^\top A_{:,i} = 0, 1 \le i \le n. \end{aligned}
\end{aligned}
\tag{14}
$$

This is a SDP problem (see standard form in Eq. 9). And it provides a lower bound of the original LMI problem (Eq. 10).

II. SDP Formulation for NN Verification

A. Neural Network Model

We formulate the deep neural network model as follows. Here, we consider the neural network models for image classification, while it can be easily generalized to many other applications. The input is the image, and the output is usually the predict class, while more concretely, we deem the output as the confidence score of each class here. So far we only consider ReLU activations.

Definition 1 (Deep Neural Network Model)**.** *The neural network model $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_y}$ with L hidden layers is defined as such:*

$$
\begin{cases}
x_1 = \phi(W_0 x_0 + b_0), \\
x_2 = \phi(W_1 x_1 + b_1), \\
\cdots, \\
x_L = \phi(W_{L-1} x_{L-1} + b_{L-1}), \\
f(x_0) = W_L x_L + b_L.
\end{cases}
\tag{15}
$$

$\phi(\cdot)$ means the elementwise application of ReLU function (ReLU$(x) = \max\{x, 0\}$). $n_0, n_1, \ldots, n_L$ denote the dimension of $x_0, x_1, \ldots, x_L$ respectively. $n_{L+1} = n_y$. $W_i \in \mathbb{R}^{n_{i+1}\times n_i}, b_i \in \mathbb{R}^{n_{i+1}}$ for $0 \le i \le L$. The model predicts the label with highest confidence score: $\arg\max_i f(x_0)_i$.

Some models may use softmax activation in the last layer, but it does not change the model decision so it is equivalent to our free-of-softmax model. The fully-connected hidden layers with ReLU activations can express (multidimension) convolutional layer, max-pooling layer, and batch normalizaton layer, which covers common neural network structures, and the approaches below can easily be extended to other types of layers.

B. Neural Network Verification

In this write-up, we consider the robustness verification of neural network models. Specifically, we would like to know that for a given sample (x, y), whether the given neural network model always predicts correctly in the whole neighborhood region of x, where the neighborhood region is characterized by ℓ_∞ bounded perturbations and the bound radius is ϵ.

The formal definition is as below.

Definition 2 (Neural Network Robustness Verification)**.** *Given a neural network model $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_y}$, which classifies the input to n_y classes. For a given sample (x, y), i.e., the input is x and the correct class label is $y \in [n_y]$, and a fixed radius ϵ.*

The model is considered safe if and only if $\forall x' \in \left\{x + \epsilon : \|\epsilon\|_\infty \leq \epsilon\right\}, \forall [n_y] \ni y' \neq y, f(x')_y \geq f(x')'_y$.

The following approaches verifies whether the model is safe or not given the sample and radius. It is known that *exact* verification is an NPC problem, so we focus on *inexact* verification, where unsafe samples can be detected, but safe samples may be falsely deem as unsafe. The verification tightness is empirically measured by how likely the safe samples be falsely deem as unsafe.

C. Activation Bounds

In the following approaches, a prerequisite is knowing the output interval of each neuron, given the model, the input, and the perturbation radius.

Still, tight interval computation is a NPC problem. In trade of speed, here we apply interval bound propagation (6) and inequality propagation (Fast-Lin) (8) to get interval. We use the overlap of the intervals from the two approaches for better tightness.

D. Direct Approach

Raghunathan et al (7), the robustness verification problem is formalized as a SDP problem.

The ReLU function is formalized as the constraints:

$$\begin{aligned} z \geq 0, z \geq Wx, z \otimes z = z \otimes (Wx + b), \quad &\text{(ReLU constraints)} \\ x \otimes x \leq (l + u) \otimes x - l \otimes u. \quad &\text{(Input constraints)} \end{aligned} \tag{16}$$

The constraints induce no relaxations.

Consider the matrix

$$P = \begin{bmatrix} 1 & P[x_1^\top] & \cdots & P[x_L^\top] \\ P[x_1] & P[x_1 x_1^\top] & \cdots & P[x_1 x_L^\top] \\ \vdots & \vdots & \ddots & \vdots \\ P[x_n] & P[x_L x_1^\top] & \cdots & P[x_L x_L^\top] \end{bmatrix} \tag{17}$$

for L layer neural network, by doing optimization over the whole matrix, for an sample (x, y) and another false label $y' \neq y$, we have the following SDP to be solved:

$$\begin{aligned} f_y^{\text{SDP}}(x, y') = \max_P (W_{L,y':} - W_{L,y:})^\top P[x_L] + b_{L,y'} - b_{L,y} \\ P[x_i] \geq 0, P[x_i] \geq W_{i-1} P[x_{i-1}] + b_{i-1}, \\ s.t. \quad \text{diag}(P[x_i x_i^\top]) = \text{diag}(W P[x_{i-1} x_i^\top]) + b_{i-1} \otimes P[x_{i-1}], \quad &\text{(ReLU constraints for layer } i) \\ \text{diag}(P[x_{i-1} x_{i-1}^\top]) \leq (l_{i-1} + u_{i_1}) \otimes P[x_{i-1}] - l_{i-1} \otimes u_{i-1}, \quad &\text{(Input constraints for layer } i) \\ P \succeq 0, \qquad 1 \leq i \leq L. \end{aligned} \tag{18}$$

In the formulation, l_i and u_i is the lower bound and upper bound of the post-activation neuron values for each layer, and as aforementioned, it does not need to be tight bounds.

The relaxation lies in optimizing the matrix P as an entire body ignoring that P is the tensor product. So P is in fact a rank-1 matrix but the relaxation permits arbitrary ranks. We directly optimize P instead of concrete x_i's. This SDP problem can be solved by off-the-shelf solvers. In the following text, we denote **Direct-SDP** to this approach.

Compare to the original paper, we inject the bias terms.

E. Dual Approach

Fazlyab et al (5) formalizes the robustness verification problem as an LMI problem.

Theorem II.1. *Consider a neural network described in Definition 1.*

- *When the pre-activation value set is characterized by*

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix}^\top P \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0, \tag{19}$$

where $x_i \in \mathbb{R}^{n_i}$, $1 \leq i \leq L$.
Given the interval bound l_i and u_i for x_i, P could be

$$\begin{bmatrix} -(\Gamma + \Gamma^\top) & \Gamma l_i + \Gamma^\top u_i \\ l_i^\top \Gamma^\top + u_i^\top \Gamma & -l_i^\top \Gamma^\top u_i - u_i^\top \Gamma l_i \end{bmatrix} \tag{20}$$

where $\Gamma \in \mathbb{R}^{n_0 \times n_0}$ and $\Gamma_{i,j} \geq 0$ for all i, j. We can reduce number of variables by constraining Γ to be a diagonal matrix.

- *When the activation function $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is characterized by*

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^\top Q \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0, \tag{21}$$

where $x \in \mathbb{R}^d$ and $Q \in \mathbb{R}^{(2d+1)\times(2d+1)}$.
For ReLU activations, Q could be from

$$\left\{ Q \in \mathbb{S}^{(2d+1)\times(2d+1)} : \begin{bmatrix} 0 & \text{diag}(\lambda) & -\nu \\ \text{diag}(\lambda) & -2\text{diag}(\lambda) & \nu + \eta \\ -\nu^\top & \nu^\top + \eta^\top & 0 \end{bmatrix}, \eta \geq 0, \nu \geq 0; \eta, \nu, \lambda \in \mathbb{R}^d \right\}, \tag{22}$$

where

$$T = \sum_{i=1}^{d} \lambda_i e_i e_i^\top \tag{23}$$

with $\lambda_i \in \mathbb{R}$. Note that comparing with (5) we discard repeated nonlinearities for better scalability.
If every layer uses ReLU as the activation function, then the model-level $Q \in \mathbb{R}^{\left(2(n_1+\cdots+n_L)+1\right)\times\left(2(n_1+\cdots+n_L)+1\right)}$ can be decomposed as

$$Q := \begin{bmatrix} & & & \text{diag}(\lambda_1) & & & -\nu_1 \\ & 0 & & & \ddots & & \vdots \\ & & & & & \text{diag}(\lambda_L) & -\nu_L \\ \text{diag}(\lambda_1) & & & -2\text{diag}(\lambda_1) & & & \nu_1 + \eta_1 \\ & \ddots & & & \ddots & & \vdots \\ & & \text{diag}(\lambda_L) & & & -2\text{diag}(\lambda_L) & \nu_L + \eta_L \\ -\nu_1^\top & \cdots & -\nu_L^\top & \nu_1^\top + \eta_1^\top & \cdots & \nu_L^\top + \eta_L^\top & 0 \end{bmatrix}, \tag{24}$$

where $\lambda_i, \nu_i, \eta_i \in \mathbb{R}^{n_i}$, $\eta_i, \nu_i \geq 0$. In the following text, we shorten $\text{diag}(\cdot)$ *to* $\text{d}(\cdot)$.

- *When safe output y is characterized by*

$$\begin{bmatrix} y \\ 1 \end{bmatrix}^\top S \begin{bmatrix} y \\ 1 \end{bmatrix} \leq 0, \tag{25}$$

where $S \in \mathbb{R}^{(n_y+1)\times(n_y+1)}$.

When the safety set has the form $e_y - e_{y'} \geq 0$, S is given by $\begin{bmatrix} 0 & -e_y + e_{y'} \\ -e_y^\top + e_{y'}^\top & 0 \end{bmatrix}$.

Then, the neural network is robust at the output x, where the input perturbation and output safety is specialized by P and S, and the activation function ϕ is specified by Q, if

$$M_{in} + M_{mid} + M_{out} \preceq 0. \tag{26}$$

Here $M_{in}, M_{mid}, M_{out} \in \mathbb{R}^{(n_0+\cdots+n_L+1)\times(n_0+\cdots+n_L+1)}$, their detail definitions are as follows:

$$M_{in} = \begin{bmatrix} -(\Gamma_0 + \Gamma_0^\top) & 0 & \cdots & \Gamma_0 l_0 + \Gamma_0^\top u_0 \\ 0 & -(\Gamma_1 + \Gamma_1^\top) & \cdots & \Gamma_1 l_1 + \Gamma_1^\top u_1 \\ \vdots & \vdots & \ddots & \vdots \\ l_0^\top \Gamma_0^\top + u_0^\top \Gamma_0 & l_1^\top \Gamma_1^\top + u_1^\top \Gamma_1 & \cdots & -\sum_{i=0}^L l_i^\top \Gamma_i^\top u_i - \sum_{i=0}^L u_i^\top \Gamma_i l_i \end{bmatrix}, \tag{27a}$$

$$M_{mid} = \begin{bmatrix} 0 & W_0^\top \mathrm{d}(\lambda_1) & 0 & 0 & \cdots & 0 & 0 & 0 & -W_0^\top \nu_1 \\ \mathrm{d}(\lambda_1) W_0 & -2\mathrm{d}(\lambda_1) & W_1^\top \mathrm{d}(\lambda_2) & 0 & \cdots & 0 & 0 & 0 & \mathrm{d}(\lambda_1) b_0 - W_1^\top \nu_2 + \nu_1 + \eta_1 \\ 0 & \mathrm{d}(\lambda_2) W_1 & -2\mathrm{d}(\lambda_2) & W_2^\top \mathrm{d}(\lambda_3) & \cdots & 0 & 0 & 0 & \mathrm{d}(\lambda_2) b_1 - W_2^\top \nu_3 + \nu_2 + \eta_2 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \mathrm{d}(\lambda_{L-1}) W_{L-2} & -2\mathrm{d}(\lambda_{L-1}) & W_{L-1}^\top \mathrm{d}(\lambda_L) & \mathrm{d}(\lambda_{L-1}) b_{L-2} - W_{L-1}^\top \nu_L + \nu_{L-1} + \eta_{L-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & \mathrm{d}(\lambda_L) W_{L-1} & -2\mathrm{d}(\lambda_L) & \mathrm{d}(\lambda_L) b_{L-1} + \nu_L + \eta_L \\ -\nu_1^\top W_0 & -\nu_2^\top W_1 & -\nu_3^\top W_2 & -\nu_4^\top W_3 & & -\nu_{L-1}^\top W_{L-2} & -\nu_L^\top W_{L-1} & & \\ & +b_0^\top \mathrm{d}(\lambda_1) & +b_1^\top \mathrm{d}(\lambda_2) & +b_2^\top \mathrm{d}(\lambda_3) & \cdots & +b_{L-3}^\top \mathrm{d}(\lambda_{L-2}) & +b_{L-2}^\top \mathrm{d}(\lambda_{L-1}) & b_{L-1}^\top \mathrm{d}(\lambda_L) & -2\nu_1^\top b_0 - \cdots - 2\nu_L^\top b_{L-1} \\ & +\nu_1^\top + \eta_1^\top & +\nu_2^\top + \eta_2^\top & +\nu_3^\top + \eta_3^\top & \cdots & +\nu_{L-2}^\top + \eta_{L-2}^\top & +\nu_{L-1}^\top + \eta_{L-1}^\top & +\nu_L^\top + \eta_L^\top & \end{bmatrix}, \tag{27b}$$

$$M_{out} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & W_L^\top(-e_y + e_{y'}) \\ 0 & (-e_y^\top + e_{y'}^\top) W_L & 2(-e_y^\top + e_{y'}^\top) b_L \end{bmatrix}. \tag{27c}$$

An important observation is that, $M_{in} + M_{mid} + M_{out}$ is a tri-diagonal block symmetric matrix except the last column and last row.

Solve the following problem:

$$\begin{aligned} \text{minimize}_{\Gamma,\lambda,\nu,\eta} \quad & s \\ \text{s.t.} \quad & M_{in} + M_{mid} + M_{out} - sI \preceq 0 \\ & \nu_i \geq 0, \eta_i \geq 0, 1 \leq i \leq L \\ & \Gamma_i \geq 0, 0 \leq i \leq L. \end{aligned} \tag{28}$$

When the optimal $s < 0$, it means the condition $M_{in} + M_{mid} + M_{out} \preceq 0$ can be satisfied. Thus, the sample is safe. Otherwise, it might not be safe. The above problem is an LMI problem, so it can be solved by off-the-shelf solvers. We denote **LMI-SDP** to this approach.

Compare to the original paper, we introduce the interval constraints to all layers instead of just the input, which significantly tightens the results.

III. Gradient-Descent Based SDP

In this section, we propose a gradient-descent based SDP solving technique, which is built upon the formulation of **LMI-SDP** .

A. Fast Inverse Matrix Calculation of $M_{in} + M_{mid} + M_{out}$

1) Block-Matrix Inverse: Assume $\begin{bmatrix} M & v \\ v^\top & r \end{bmatrix} \in \mathbb{S}^{n\times n}$ is a symmetric matrix, where $M \in \mathbb{S}^{(n-1)\times(n-1)}, v \in \mathbb{R}^{n-1}, r \in \mathbb{R}$.

When the matrix is invertible, i.e., M^{-1} exists and $r - v^\top M^{-1} v \neq 0$, we have

$$\begin{bmatrix} M^{-1} + (r - v^\top M^{-1} v)^{-1} M^{-1} v v^\top M^{-1} & -(r - v^\top M^{-1} v)^{-1} M^{-1} v \\ -(r - v^\top M^{-1} v)^{-1} v^\top M^{-1} & (r - v^\top M^{-1} v)^{-1} \end{bmatrix} \tag{29}$$

is its inverse matrix.

Easily seen, the inverse calculation takes only $O(n^2)$ computation with M^{-1} is given.

2) Tri-Diagonal Matrix Inverse: For a invertible tri-diagonal block matrix

$$\begin{bmatrix} A_{1,1} & A_{1,2} & & & & & & \\ A_{2,1} & A_{2,2} & A_{2,3} & & & & & \\ & A_{3,2} & A_{3,3} & A_{3,4} & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & A_{(n-1),(n-2)} & A_{(n-1),(n-1)} & A_{(n-1),n} \\ & & & & & A_{n,(n-1)} & A_{n,n} \end{bmatrix}, \tag{30}$$

similar with simple tri-diagonal block matrix, we can do LU-decomposition $A = LU$ and meanwhile solve $Lx = b$ using Algorithm 1.

Algorithm 1: LU-Decomposition of Tri-Diagonal Block Matrix

Data: Tri-diagonal block matrix A

Result: Upper triangular block matrix U; lower triangular block matrix L.

$U_{1,1} \leftarrow A_{1,1}, b'_1 = b_1$;
for $i \leftarrow 2, \ldots, n$ **do**
  $L_{i,(i-1)} \leftarrow A_{i,(i-1)} U^{-1}_{(i-1),(i-1)}$;
  $U_{i,i} \leftarrow A_{i,i} - L_{i,(i-1)} A_{(i-1),i}$;
  $U_{(i-1),i} \leftarrow A_{(i-1),i}$;
  $b'_i \leftarrow b_i - L_{i,(i-1)} b'_{i-1}$;
end

The generated lower triangular block matrix L, and upper triangular block matrix U are shaped as below:

$$L = \begin{bmatrix} I & & & & \\ L_{2,1} & I & & & \\ & \ddots & \ddots & & \\ & & L_{(n-1),(n-2)} & I & \\ & & & L_{n,(n-1)} & I \end{bmatrix}, U = \begin{bmatrix} U_{1,1} & U_{1,2} & & & \\ & U_{2,2} & U_{2,3} & & \\ & & \ddots & \ddots & \\ & & & U_{(n-1),(n-1)} & U_{(n-1),n} \\ & & & & U_{n,n} \end{bmatrix}. \tag{31}$$

and $b' = L^{-1} b$.

Recursively let $b = e_i$ and solve $Ux = L^{-1} b$, then we can calculate the inverse of A.

Time Complexity Analysis. Without loss of generality, assume the block matrix $A \in \mathbb{R}^{n\times n}$ is composed of k $\mathbb{R}^{m\times m}$ matrix.

Then, we separate LU-deformation and b'_i calculation.

- One-time LU-decomposition takes $O(k \cdot m^3) = O(nm^2)$ computation.
- For each b, calculating $b' = L^{-1} b$ requires $O(k \cdot m^2)$, then solving $Ux = L^{-1} b$ requires another $O(k \cdot m^2)$. And there are $O(n)$ b's. As the result, the overall time complexity is $O(n \cdot k \cdot m^2) = O(n^2 m)$.

Finally, inverse calculation requires $O(n^2 m)$.

3) Case of $M_{in} + M_{mid} + M_{out}$: As we previously showed, $M_{in} + M_{mid} + M_{out}$ is a tri-diagonal block matrix except the last row and column, so combining the above two analysis, we can conclude that the inverse computation can be done in time complexity $O(n^2 m)$.

B. Compute Dominant Eigenvalue by Power Method

A well-known and useful approach for dominant eigenvalue computation is power method.

Starting from a random non-zero vector v_0, we compute

$$v_k = \frac{Av_{k-1}}{\|v_{k-1}\|}, \tag{32a}$$

and

$$\mu_k = v_k^\top A v_k. \tag{32b}$$

As $k \to \infty$, $\mu_k \to \lambda_{e1}$ which is the eigenvalue with largest norm (i.e., dominant eigenvalue). Moreover, under two additional conditions, v_k converges to the associated eigenvector.

In our scenario, A is a real symmetric matrix, thus the power method can identify the eigenvalue with largest absolute value.

C. Internal-Point Methods

The classical way to solve SDP problem (without loss of generality, consider LMI problem) is internal-point methods (2).

Formally, for the problem defined as Equation 10, but more general by discarding linear constraints:

$$\begin{aligned} &\text{minimize}_x \ c^\top x, \\ &\text{s.t.} \qquad x_1 F_1 + x_2 F_2 + \cdots + x_n F_n + G \preceq 0, \end{aligned} \tag{33}$$

still with $F_i, G \in \mathbb{S}^k$, the solving approach is using log-determinant barrier function:

$$\text{minimize}_x \ L(x) = \text{minimize}_x \ tc^\top x - \ln\det\left(-\sum_{i=1}^{n} x_i F_i - G\right), \tag{34}$$

where t is the parameter controlling the tightness of the barrier. The convention of internal-point methods is to increase t from a initial value then increase it exponentially $t \leftarrow \mu t$. For each t, the objective is minimized by Newton's method. Then after the change of t, the previous optimal point is used as the starting point.

Phase I Methods. To execute internal-point methods, firstly a feasible solution is needed. This is achieved by consider the following alternative problem:

$$\begin{aligned} &\text{minimize}_s \ s, \\ &\text{s.t.} \ \sum_{i=1}^{n} x_i F_i + G \preceq sI, \end{aligned} \tag{35}$$

and stops when $s \leq 0$.

For this problem, the initial point could be any x. But s should be set to $\lambda_{\max}\left(\sum_{i=1}^{n} x_i F_i + G\right) + \epsilon_s$ where $\lambda_{\max}(\cdot)$ stands for the largest eigenvalue, and ϵ_s is used to prevent singularity. We can calculate an upper bound of $\lambda_{\max}$ from power method (Section III-B).

Gradient. Derived from (2) and (1), the gradient of $L(x)$ (Equation 34) is given by

$$\nabla_{x_i} L(x) = tc_i + \text{tr}(S^{-1} F_i), \tag{36}$$

where $S = -\sum_{i=1}^{n} x_i F_i - G$ or $S = -\sum_{i=1}^{n} x_i F_i - G + sI$ for problem in Equation 33 and Equation 35 respectively.

Proof. Without loss of generality, consider $L(x) = tc^\top x - \ln\det\left(-\sum_{i=1}^n x_i F_i - G\right)$.

$$
\begin{aligned}
&\nabla_{x_i} \ln\det \overbrace{\left(-\sum_{i=1}^n x_i F_i - G\right)}^{S} \\
=&\frac{1}{\det S}\frac{\partial \det S}{\partial x_i} \\
=&\frac{1}{\det S}\sum_{i=1}^k\sum_{j=1}^k \left((-F_i)\otimes S^*\right)_{ij} \\
=&-\sum_{i=1}^k\sum_{j=1}^k S^{-\top}\otimes F_i \\
=&-\operatorname{tr}(S^{-1}F_i) \qquad\qquad \left(\text{by } \operatorname{tr}(A^\top B) = \sum_{i=1}^n\sum_{j=1}^n A_{ij}B_{ij}\right)
\end{aligned} \tag{37}
$$

Thus $\nabla_{x_i} L(x) = tc_i + \operatorname{tr}(S^{-1}F_i)$. □

D. Extension to Quadratic Constraints (This theoretical part is now unuseful.)

Extend the problem specified in Equation 33 by permitting quadratic coefficients on x_i:

$$
\begin{aligned}
&\text{minimize}_x\ c^\top x, \\
&\text{s.t.} \qquad x_1F_1 + x_2F_2 + \cdots + x_nF_n + x_1^2H_1 + x_2^2H_2 + \cdots + x_n^2H_n + G \preceq 0,
\end{aligned} \tag{38}
$$

with the additional matrices $H_i \in \mathbb{S}^k$. We can again compute the barrier function with tightness parameter t, then give its closed-form gradient expression using the technique similar to Equation 37:

$$
\begin{aligned}
L(x) = tc^\top x - \ln\det \overbrace{\left(-\sum_{i=1}^n x_i F_i - \sum_{i=1}^n x_i^2 H_i - G\right)}^{S}, \\
\nabla_{x_i} L(x) = tc_i + \operatorname{tr}\left(S^{-1}(F_i + 2x_iH_i)\right).
\end{aligned} \tag{39}
$$

When S^{-1} is known, the gradient can be computed in $O(k^2)$.

Moreover, even with quadratic constraints, the problem is still convex, which is given by the following theorem.

Theorem III.1. *Let $L : \mathbb{R}^n \to \mathbb{S}^k$ be the quadratic combination of symmetric matrices:*

$$
L(x) = \sum_{i=1}^n x_i F_i + \sum_{i=1}^n x_i^2 H_i + G, \tag{40}
$$

where $x \in \mathbb{R}^n$; $F_i, G \in \mathbb{S}^k$; and $H_i \in \mathbb{S}^k_{\succeq 0}$ for $1 \le i \le n$. Then, the region $\left\{x : L(x) \preceq 0\right\}$ is convex.

Proof. For any x, y, consider $L(^{(x+y)}/_2)$:

$$\begin{aligned} L(^{(x+y)}/_2) &= \frac{1}{2}\left(\sum_{i=1}^{n} x_i F_i + \sum_{i=1}^{n} y_i F_i\right) + \sum_{i=1}^{n}\left(\frac{x_i+y_i}{2}\right)^2 H_i + G \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} x_i F_i + \sum_{i=1}^{n} y_i F_i + \sum_{i=1}^{n}\left(\frac{1}{2}x_i^2 + \frac{1}{2}y_i^2 + x_i y_i\right) H_i + 2G\right) \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} x_i F_i + \sum_{i=1}^{n} y_i F_i + \sum_{i=1}^{n}\left(x_i^2 + y_i^2 - \frac{1}{2}(x_i - y_i)^2\right) H_i + 2G\right) \\ &= \frac{1}{2}\big(L(x) + L(y)\big) - \frac{1}{4}\sum_{i=1}^{n}(x_i - y_i)^2 H_i. \end{aligned} \tag{41}$$

If for vector $v \in \mathbb{R}^k$, $v^\top L(x) v \le 0$ and $v^\top L(y) v \le 0$, we have

$$\begin{aligned} v^\top L(^{(x+y)}/_2) v &= \frac{1}{2}\left(v^\top L(x) v + v^\top L(y) v\right) - \frac{1}{4}\sum_{i=1}^{n}(x_i - y_i)^2 v^\top H_i v \\ &\le \frac{1}{2}\left(v^\top L(x) v + v^\top L(y) v\right) \le 0. \end{aligned} \tag{42}$$

Because $L(x) \preceq 0, L(y) \preceq 0$, $L(^{(x+y)}/_2) \preceq 0$, which means the region $\left\{x : L(x) \preceq 0\right\}$ is convex. □

E. Apply to The Problem by Gradient Descent

Recall the LMI formalization theorem (Theorem II.1), we can propose to minimize s for the required radius size ϵ.

Otherwise, it might be unsafe.

The variables can be partitioned to two disjoint sets:

$$\mathcal{M} = \left\{(W_i, b_i) : 0 \le i \le L\right\}, \mathcal{V} = \left\{\Gamma_i; \lambda_k, \nu_k, \eta_k : 0 \le i \le L, 1 \le k \le L, \nu_k \ge 0, \eta_k \ge 0, \Gamma_i \ge 0\right\}. \tag{43}$$

Furthermore, we reduce the decision variables by constraining Γ_i's to be diagonal matrices $\mathrm{diag}(\gamma_i)$'s where $\gamma_i \ge 0, 0 \le i \le L$. This introduces additional imprecision but significantly decreases the number of decision variables from $O(m^2)$ to $O(m)$.

Let us recall $M_{in} + M_{mid} + M_{out}$ (Equation 27), by combining the three matrices and s together, and constraining $\Gamma_i = \mathrm{diag}(\gamma_i)$, we have $-S = M_{in} + M_{mid} + M_{out} - sI =$

$$\begin{bmatrix}
-2\mathrm{d}(\gamma_0) - sI & W_0^\top \mathrm{d}(\lambda_1) & 0 & 0 & \cdots & 0 & 0 & 0 & -W_0^\top \nu_1 + \gamma_0 \otimes (l_0 + u_0) \\
\mathrm{d}(\lambda_1) W_0 & -2\mathrm{d}(\gamma_1 + \lambda_1) - sI & W_1^\top \mathrm{d}(\lambda_2) & 0 & \cdots & 0 & 0 & 0 & \lambda_1 \otimes b_0 - W_1^\top \nu_2 + \nu_1 + \eta_1 + \gamma_1 \otimes (l_1 + u_1) \\
0 & \mathrm{d}(\lambda_2) W_1 & -2\mathrm{d}(\gamma_2 + \lambda_2) - sI & W_2^\top \mathrm{d}(\lambda_3) & \cdots & 0 & 0 & 0 & \lambda_2 \otimes b_1 - W_2^\top \nu_3 + \nu_2 + \eta_2 + \gamma_2 \otimes (l_2 + u_2) \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & \mathrm{d}(\lambda_{L-1}) W_{L-2} & -2\mathrm{d}(\gamma_{L-1} + \lambda_{L-1}) - sI & W_{L-1}^\top \mathrm{d}(\lambda_L) & \lambda_{L-1} \otimes b_{L-2} - W_{L-1}^\top \nu_L + \nu_{L-1} + \eta_{L-1} + \gamma_{L-1} \otimes (l_{L-1} + u_{L-1}) \\
0 & 0 & 0 & 0 & \cdots & 0 & \mathrm{d}(\lambda_L) W_{L-1} & -2\mathrm{d}(\gamma_L + \lambda_L) - sI & \lambda_L \otimes b_{L-1} + \nu_L + \eta_L + \gamma_L \otimes (l_L + u_L) + W_L^\top(-e_y + e_{y'}) \\
[2pt/2pt] & \lambda_1^\top \otimes b_0^\top & \lambda_2^\top \otimes b_1^\top & \lambda_3^\top \otimes b_2^\top & \cdots & \lambda_{L-2}^\top \otimes b_{L-3}^\top & \lambda_{L-1}^\top \otimes b_{L-2}^\top & \lambda_L^\top \otimes b_{L-1}^\top & -s \\
-\nu_1^\top W_0 & -\nu_2^\top W_1 & -\nu_3^\top W_2 & -\nu_4^\top W_3 & \cdots & -\nu_{L-1}^\top W_{L-2} & -\nu_L^\top W_{L-1} & & -2\sum_{i=1}^{L} \nu_i^\top b_{i-1} \\
 & +\nu_1^\top + \eta_1^\top & +\nu_2^\top + \eta_2^\top & +\nu_3^\top + \eta_3^\top & \cdots & +\nu_{L-2}^\top + \eta_{L-2}^\top & +\nu_{L-1}^\top + \eta_{L-1}^\top & +\nu_L^\top + \eta_L^\top & -2\sum_{i=0}^{L} l_i^\top \gamma_i \otimes u_i \\
+\gamma_0^\top \otimes (l_0^\top + u_0^\top) & +\gamma_1^\top \otimes (l_1^\top + u_1^\top) & +\gamma_2^\top \otimes (l_2^\top + u_2^\top) & +\gamma_3^\top \otimes (l_3^\top + u_3^\top) & \cdots & +\gamma_{L-2}^\top \otimes (l_{L-2}^\top + u_{L-2}^\top) & +\gamma_{L-1}^\top \otimes (l_{L-1}^\top + u_{L-1}^\top) & +\gamma_L^\top \otimes (l_L^\top + u_L^\top) & +2(-e_y^\top + e_{y'}^\top) b_L \\
 & & & & & & & +(-e_y + e_{y'})^\top W_L &
\end{bmatrix}. \tag{44}$$

We can see that for set $\mathcal{M}, \mathcal{V}$ respectively, the matrix $-S$ is a linear combination from elements in the set. According to Theorem III.1, viewing from each set, the optimization problem is a convex problem.

We try to solve the optimization using gradient descent and update either $\mathcal{M}$ or $\mathcal{V}$. The benign convex property provides convergence guarantee for gradient descent.

F. Gradient Descent on $\mathcal{V}$

View from $\mathcal{V}$ variables, the matrix S is a linear combination of the sub-matrices of each variable. Formally, we have

$$S = -\sum_{i=0}^{L}\sum_{j=1}^{n_i}\gamma_{ij}M_{\gamma,ij} - \sum_{i=1}^{L}\sum_{j=1}^{n_i}\lambda_{ij}M_{\lambda,ij} - \sum_{i=1}^{L}\sum_{j=1}^{n_i}\nu_{ij}M_{\nu,ij} - \sum_{i=1}^{L}\sum_{j=1}^{n_i}\eta_{ij}M_{\eta,ij} - C + sI. \tag{45}$$

Blocking $S^{-\top}$ in the same way as Eq. 44:

$$S^{-\top} = \begin{bmatrix} N_{00} & N_{01} & \cdots & N_{0L} & w_0 \\ N_{10} & N_{11} & \cdots & N_{1L} & w_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ N_{L0} & N_{L1} & \cdots & N_{LL} & w_L \\ [2pt/2pt] w_0^\top & w_1^\top & \cdots & w_L^\top & w_{L+1} \end{bmatrix}. \tag{46}$$

The internal-point methods with objective $L = ts - \ln\det S$ yields the gradients as follows:

$$\nabla_s L = t - \mathrm{tr}(S^{-1}), \tag{47a}$$

$$\nabla_{\gamma_i} L = -2\mathrm{diag}(N_{ii}) + 2w_i \otimes (l_i + u_i) - 2w_{L+1}(l_i \otimes u_i), \tag{47b}$$

$$\nabla_{\lambda_i} L = 2(W_{i-1} \otimes N_{i,i-1})_{:,\sum} + 2b_{i-1} \otimes w_i - 2\mathrm{diag}(N_{ii}), \tag{47c}$$

$$\nabla_{\nu_i} L = -2W_{i-1}w_{i-1} + 2w_i - 2w_{L+1}b_{i-1}, \tag{47d}$$

$$\nabla_{\eta_i} L = 2w_i. \tag{47e}$$

A property is that when $S \preceq 0$ but is approaching the boundary, $\mathrm{tr}(S^{-1}) > 0 \to +\infty$. Thus the gradient descent direction for s: $\mathrm{tr}(S^{-1}) - t$ will > 0 for small t. Therefore, we need to increase t iteratively just as classical internal-point methods: $t \leftarrow \mu t$. Except s, other gradients have no relation with t.

Time Complexity. The main bottle-neck lies in the computation of S^{-1}, which takes $O(n^2 m)$. Once S^{-1} is computed use aforementioned technique, the gradient calculation is fast - takes no more than $O(n^2)$ as seen from the above equations.

Take the internal-point method iterations into consideration. Therefore, the time complexity is $O(cn^2 m)$ where c is the number of iterations needed for convergence, which is typically in $[10, 100]$.

We denote **GD-SDP** to this approach.

IV. Further Extensions

Instead of viewing $\mathcal{V}$ as decision variables, we can also rewrite S as the linear combination of model parameters $\mathcal{M}$. So the similar gradient forms exist for model parameters as well.

The main idea is to use a neural network $g_\theta(\cdot)$ to predict the variables in $\mathcal{V}$ (see Equation 43) and train $g_\theta(\cdot)$ using gradient descent as showed before (Equation 47).

Formally, the input of $g_\theta(\cdot)$ are x_0, ϵ, and $\mathcal{M}$, and the output are elements in $\mathcal{V}$. The network parameters are denoted by θ.

The network structure could be similar to the one in (3), which is shown in Figure 1.

The major changes we will make lie in two aspects:

- Network Output:
  The architecture in the figure only predicts one variable for each layer. This is due to the over-relaxation of

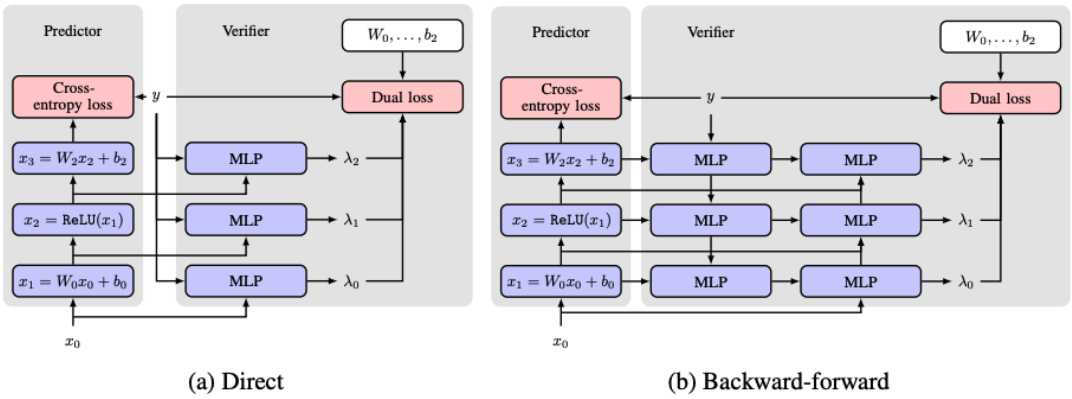

Figure 1. Predictor-verifier network architectures from (3).

their duality formulation. In our SDP-dual formulation, for input and each layer, the number of variables is in the same order of layer width. As the result, the number of parameters in the network will become a bit larger. To be precise, the number of outputs will be in the same order of number of neurons.

- Objective:
  According to the chain rule, the training objective is given by

$$\nabla_\theta L = \nabla_\gamma L \cdot \nabla_\theta \gamma + \nabla_\lambda L \cdot \nabla_\theta \lambda + \nabla_\nu L \cdot \nabla_\theta \nu + \nabla_\eta L \cdot \nabla_\theta \eta. \tag{48}$$

  Among them, $\nabla_\gamma L, \nabla_\lambda L, \nabla_\nu L$, and $\nabla_\eta L$ has analytic form as shown in Equation 47; $\nabla_\theta \Gamma, \nabla_\theta \lambda, \nabla_\theta \nu$, and $\nabla_\theta \eta$ is exactly the network gradients which can be computed using back propagation.

Training Robust Models. From Equation 47, we can see that the gradients for model weights W and b are available. They provide the direction of optimization for a more robust and easier verified model. We can combine these gradients along with the gradient given by normal loss to train a robust model. Formally,

$$\nabla_W (loss + kL) = \nabla_W loss + k \nabla_W L, \nabla_b (loss + kL) = \nabla_b loss + k \nabla_b L, \tag{49}$$

where k is a hyper parameter.

V. Implementation

We implement all the three verification techniques **Direct-SDP** , **LMI-SDP** , and **GD-SDP** in a united framework in Python. The SDP solver is SCS[1] with interfaces of CVXPY[2].

We also implemented Fast-Lin (8), Interval Bound Propagation (6) for activation bound computation. Fast-Lin is also used as the baseline for verification.

For tightness and correctness check, we use PGD attack to obtain a lower bound of the robust accuracy. The PGD attack is implemented by cleverhans[3] toolkit.

VI. Experiments

We prepare clever trained, PGD-adversarial trained, and provable-robust trained models on MNIST and CIFAR10. Currently, because of the long running time and bad scalability, we only test small models on MNIST.

Model A. The model is PGD-adversarial trained. The training ϵ is 0.1, while the verification ϵ is 0.03. It has only one hidden layer which contains 100 neurons.

Model B. The model is LP-Dual (9) trained. The training ϵ is 0.3, while the verification ϵ is 0.1. Its structure is `input (28x28) - conv (channel=4, kernel=4, stride=2) - conv (channel=8, kernel=4, stride=2) - flatten - linear (8x7x7, 100) - linear(100, 10)`.

[1]https://github.com/cvxgrp/scs

[2]https://www.cvxpy.org/tutorial/advanced/index.html

[3]https://github.com/tensorflow/cleverhans

Table I
COMPARISON OF **DIRECT-SDP** AND **LMI-SDP** . BASELINE (FAST-LIN), LOWER BOUND, AND THE EXACT VALUE ARE ALSO LISTED.

Model	Clean Acc.	Upper Bound	Exact Value	Upper Bound		
		PGD	MILP	Fast-Lin	**Direct-SDP**	**LMI-SDP**
Model A	88.24%	86.27%	84.31%	47.06%	66.67%	60.78%
Model B	90.20%	82.35%	80.39%	47.06%	33.33%	0.00%

Table II
AVERAGE TIME COMPARISON.

Model	MILP	**Direct-SDP**	**LMI-SDP**
Model A	0.446 s	321.3 s	359.9 s
Model B	0.447 s	44 064.9 s	23 460.8 s

Based on attempted run, for **Direct-SDP** and **LMI-SDP** , we set the precision requirements to be 0.01 and 0.001 respectively, which assures no deterioration because of sub-optimal solution.

We use training samples from the 50th to the 100th for the experiments. In table I, we report the robust accuracy and the average running time per sample, for **Direct-SDP** and **LMI-SDP** . What we have faced when implementing **GD-SDP** is discussed later.

From Table I and II, we know that **Direct-SDP** not only offers tighter verification result, but also runs faster. While they both perform much better than linear inequality propagation, i.e., Fast-Lin, gaps still remain if we deem the PGD upper bound is tight in some degree.

*A. **GD-SDP** is Invalid.*

From the experiments, we found that *the gradient descent based approach **GD-SDP** could not calculate the SDP problem correctly*.

We suspect that it is because of the singularity of inverse matrix when the optimization process is approaching the barrier. The theoretical analysis shows that when the matrix S approaches the boundary of semidefinite, the eigenvalues of the inverse should be positive infinite. As the result, $\mathrm{tr}(S^{-1}) = \sum d_i \lambda_{S^{-1},i}$ should be positive infinite, and $\nabla_s L$ should be negative infinite (see Eq. 47), which strongly increases s to keep away from the boundary.

However, in gradient descent, probably because of precision loss, or probably because of strong negative effect after crossing the boundary, we do not see such drastic repulse during the gradient descent process. Therefore, the variables pass through the barrier easily and converge to an illegal solution.

To far, we do not know how to solve the issue using first-order methods. Second-order optimization like Newton method may solve the issue, but the denseness of Hessian matrix vanishes the efficiency of gradient descent.

*B. **LMI-SDP** is No Better than **Direct-SDP** .*

In addition to the tightness analysis (Table I) and runtime analysis (Table II), we also analyze the optimal s magnitude in III. From the results, we know that **LMI-SDP** is more vulnerable to numerical error because the average solution is much closer to 0 comparing with **Direct-SDP** . Taken all these factors in consideration, we think that **LMI-SDP** is not as good as **Direct-SDP** .

Table III
AVERAGE MAGNITUDE OF OPTIMAL s.

Direct-SDP	**LMI-SDP**
0.046	4.060

VII. Discussion and Conclusion

In this work, currently we show two takeaways:

1) **GD-SDP** is currently unable to accelerate **LMI-SDP** .
2) **LMI-SDP** is not as good as it was claimed in (5), and it cannot strictly beat **Direct-SDP** .

Recently, a UAI2019 paper (4) starts from **Direct-SDP** and does relaxation to achieve better scalability. We try to achieve tighter verification while maintaining scalabiltiy based on tighter base approach, i.e., **LMI-SDP** , which motivates our **GD-SDP** . However, the inherent shortage of **LMI-SDP** makes it hard as of now.

As the result, we plan to suspend this project now and turn to next ideas.

REFERENCES

[1] Can a determinant be differentiated? https://www.qc.edu.hk/math/Advanced%20Level/Diff_determinant.htm. Accessed: 2019-09-07.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.

[4] Krishnamurthy Dj Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. In *Proc. Uncertainty in Artificial Intelligence, UAI*, page 164, 2019.

[5] Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *arXiv preprint arXiv:1903.01287*, 2019.

[6] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[7] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10877–10887, 2018.

[8] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5273–5282, 2018.

[9] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.