Safety Verification and Robustness Analysis of Neural Networks via Quadratic Constraints and Semidefinite Programming:

By: Mahyar Fazlyab, Manfred Morari, George J. Pappas

Brief Overview

What we'll talk about today:

- Background on semidefinite programming
- Introduction to the problem of neural network verification
- Technical definitions and prerequisites
- Verification via SDP
- Experimental Results

What exactly is semidefinite programming?

First, general optimization

A general optimization problem takes the following form:

$$\begin{aligned} \text{minimize} \;\; & f(x) \\ \text{subject to} \;\; & g_i(x) \le b_i, \quad i \in [m] \end{aligned}$$

A related example: linear programming

A linear program of vectors in $\mathbb{R}^n$ is an optimization problem of the following sort:

$$\text{maximize } \; c^T x$$

$$\text{subject to } \; Ax \leq b$$

$$\text{and } \; x \geq 0$$

A related example: linear programming

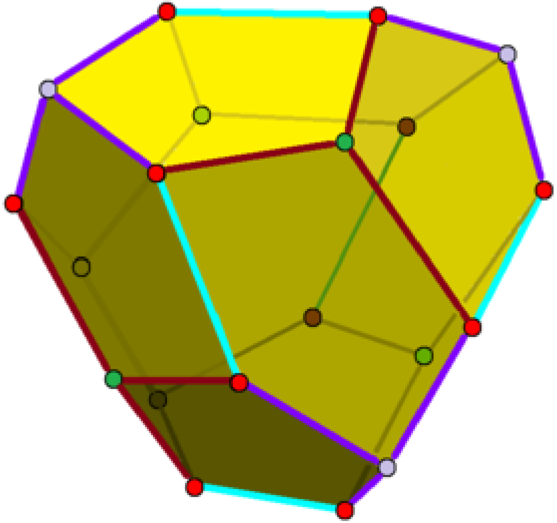


Figure: A polytope

Methods of solving LP's

- simplex algorithm
- criss-cross algorithm
- ellipsoid algorithm
- ... and many more

What does a semidefinite program look like?

First, some preliminaries:

- **Definition:** A matrix M is positive semidefinite if there are vectors $x^1, \ldots, x^n$ such that $m_{i,j} = x^i \cdot x^j$. We denote M is positive semidefinite as $M \succeq 0$.
- **Equivalently:** A real matrix M is positive semidefinite if, for any $x \in \mathbb{R}^n$, we have $x^T M x \geq 0$.

What does a semidefinite program look like?

- Let $\mathbb{S}^n$ be the set of all $n \times n$ symmetric matrices.
- Define an inner produce over $\mathbb{S}^n$ as $\langle S, T \rangle := tr(S^T T) = \sum_{i,j} S_{i,j} T_{i,j}$.

What does a semidefinite program look like?

Doing this, a semidefinite programming (SDP) problem looks as follows:

$$\text{Minimize } \langle S, X\rangle$$

$$\text{Subject to } \langle A_k, X\rangle \leq b_k, \quad k \in [m]$$

$$\text{and } X \succeq 0$$

Where we can replace $\leq$ on the second line with $=$ by using slack variables.

What does a semidefinite program look like?

Using the the definition of a semidefinite matrix (i.e. $m_{i,j} = x^i \cdot x^j$), we can rewrite to get:

$$\text{Minimize} \quad \sum_{i,j \in [n]} c_{i,j}(x^i \cdot x^j)$$

$$\text{Subject to} \quad \sum_{i,j \in [n]} a_{i,j,k}(x^i \cdot x^j) \leq b_k \ \text{ for all } k$$

This looks a lot more like the traditional linear programming (LP) example. Also, justifies the term quadratic constraints.

What does a semidefinite program look like

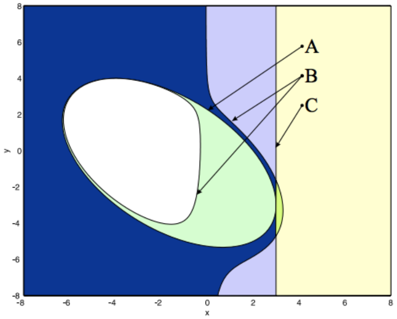


Figure: Feasible region for SDP

Linear Programs as SDP's

Suppose you are given a linear program defined by constant $c, a_1, \ldots, a_k \in \mathbb{R}^n$, and $b_1, \ldots, b_k \in \mathbb{R}$. We can concoct the matrices:

$$A_i = \mathrm{diag}(a_i) \;\; C := \mathrm{diag}(c)$$

To generate the program:

$$\text{minimize } \langle C, X \rangle$$

$$\text{subject to } \langle A_i, X \rangle \leq b_i$$

$$\text{and } x_{i,j} = 0 \text{ when } i \neq j, \quad X \succeq 0$$

Common Approaches

- Interior point methods
- First-order methods
- Many are approximate, all are slower than LP approaches!

Now... Lets talk about some other stuff related to SDP's and semidefinite matrices.

Linear Matrix Inequalities (LMI's)

Recall, we used $X \succeq 0$ to denote that X is a positive semidefinite matrix. More generally, we can relate matrices X and Y by $X \succeq Y$ if and only if $X - Y \succeq 0$. Likewise, we can write $0 \succeq X$ if X is negative semidefinite. We can define Linear Matrix Inequalities (LMI) as expressions of the form:

$$\sum_{i=1}^{n} A_i \succeq \sum_{j=1}^{m} B_j$$

Quadratically Constrained Quadratic Programs (QCQP)

A general class of optimization problems of the following sort:

$$\text{minimize } \; x^T P_0 x + q_0^T x$$

$$\text{subject to } \; x^T P_i x + q_i^T x + r_i \leq 0$$

$$\text{and } \; Ax = b$$

In general, solving these is NP-Hard. However, if we restrict P_0 and the P_i's to semidefinite matrices, can use SDP to solve! Likewise, can incorporate LMI as in SDP!

So… why are SDP's and optimization relevant to neural network (NN) verification?

High-level problem overview

- Neural networks (NN's) are being used to emulate human decisions (e.g. in self-driving cars, face-recognition systems)
- NN's are still a relatively black-box tool.
- We want a better way to reason about how NN's classify large sets of inputs.
- That is, a way to prove security properties of NN's.

More detailed overview (Geometric)

Viewing a NN as a function $f : \mathbb{R}^n \to \mathbb{R}^m \ldots$

- Can view NN security properties as geometric constraints on inputs and outputs.
- For a region $X \subset \mathbb{R}^n$, another region $Y := f(X) \subset \mathbb{R}^m$, for some $S \subset \mathbb{R}^m$, we may ask $\mathbb{1}[Y \subset S]$.
- e.g., if our NN f is a binary spam classifier, we may ask "is f(non spam messages) $\subset \{0\}$", where 0 indicates not spam.

More detailed overview (Optimization)

We can all see robustness guarantees as an optimization problem:

- Suppose f is a NN for a classification problem.
- Suppose we have a pair (x, y) and want to ensure for any point $A(x) \in B_\epsilon(x)$, $f(x) = f(A(x))$.
- Suppose NN returns the value $\arg\max\{f(x)_i\}$ where i ranges over all labels.
- Can write as optimization problem $\mathrm{maximize}_{y' \in [n] \setminus \{y\}} f(A(x))_{y'} - f(A(x))_y$.

Why this question isn't easy

While answering this yes/no question may seem straightforward, it is not!

- Computing $f(X)$ exactly is computationally hard!
- Moreover, general NN's are not convex! Hard to use optimization techniques.

So how do we address this

- In general, we will try to over approximate $f(X)$ with some set $\tilde{Y} \subset \mathbb{R}^m$, and check "is $\tilde{Y} \subset S$ instead.
- Question is then: how do we balance conservatism with tightness? (NN's are highly non-linear)
- Important as we can have false positives (i.e. $Y \subset S$, but $\tilde{Y} \not\subset S$)
- How do we propogate X through the layers without being too conservative?

Key observation!

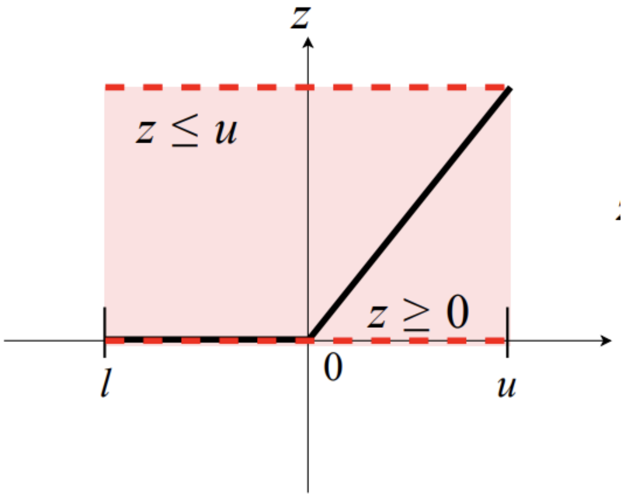
We just need to bound the activation functions!

- Outside of the activation functions, NN's just consist of affine functions. Depending on X, we can usually just compute $f(X)$ if f is affine.
- Examples of activation functions: $\mathrm{Relu}(x) := \max\{0, x\}$, $\tanh(x)$, $\sigma(x)$.

What are approaches for doing this?

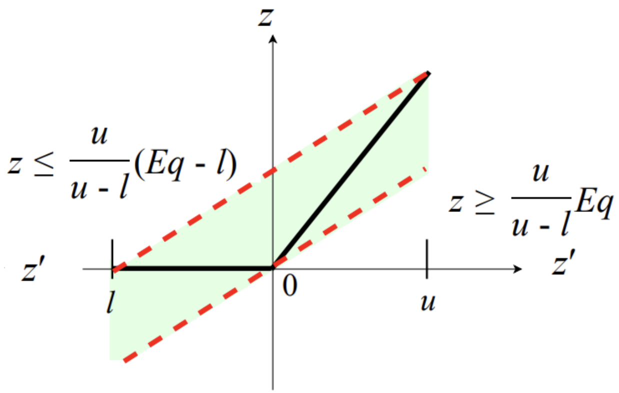
- Symbolic linear relaxation and propagation
- Naive concretization
- Linear relaxation
- ... and quadratic constraints! (QC's)

What are approaches for doing this?



(a) Naive concretizaion

What are approaches for doing this?



(b) Symbolic linear relaxation

What are approaches for doing this?

And now... the subject of the talk... quadratic constraints (QC's) via SDP!

NN Notation

Throughout, we will assume a NN f takes its inputs as a subset $X \subset \mathbb{R}^{n_x}$ and has outputs as a set $Y \subset \mathbb{R}^{n_y}$

- We denote an input as $x^0 = x$, the value passed from layer i to $i+1$ as $x^{i+1} := \phi(W^i x^i + b^i)$, and $f(x) := W^l x^l + b^l$ if the NN has l layers.
- We are assuming that the NN has an activation $\phi(x) := [\varphi(x_1), \ldots, \varphi(x_{d_i})]^T$ where $\varphi : \mathbb{R} \to \mathbb{R}$ is a 1-D activation fn.

Overview of developing QC's

We need to develop the following notions to use QC's to verify NN robustness.

(1) How to overestimate X using a set (potentially infinite) of QC's.

(2) Given QC's for a previous layer and an activation function ϕ, specify QC's for the subsequent layer.

(3) How to specify QC's for a safety set in the output layer.

(4) How to use SDP to check that, given input and transition constraints, the safety QC's in the output layer hold true.

Specifying QC's on the input layer

Quadratic relaxation: For hyperractangle $= \{x \in \mathbb{R}^d : \underline{x} \leq x \leq \overline{x}\}$, the quadratic constraints can be phrased as:

$$(x - \underline{x})^T(\overline{x} - x) \geq 0$$

But how could we encode it into a semidefinite matrix so that we can solve it by SDP optimization?

Specifying QC's on the input layer

Note that the inequality $\underline{x} \leq x \leq \overline{x}$ can be written as, for all $\Gamma_{i,j} \geq 0$, $1 \leq i, j \leq d$ (d is the input dimension):

$$\Gamma_{i,j}(x_i - \underline{x}_i)^T(\overline{x}_j - x_j) \geq 0$$

Specifying QC's on the input layer

Note that the inequality $\underline{x} \leq x \leq \overline{x}$ can be written as, for all $\Gamma_{i,j} \geq 0$, $1 \leq i, j \leq d$ (d is the input dimension):

$$\Gamma_{i,j}(x_i - \underline{x}_i)^T(\overline{x}_j - x_j) \geq 0 \tag{1}$$

We can construct matrix P with $\Gamma = [\Gamma_{i,j}] \in \mathbb{R}^{d\times d}$:

$$P := \left\{ \begin{bmatrix} -(\Gamma + \Gamma^T) & \Gamma\underline{x} + \Gamma^T\overline{x} \\ \underline{x}^T\Gamma^T + \overline{x}^T\Gamma & -\underline{x}^T\Gamma^T\overline{x} - \overline{x}^T\Gamma\underline{x} \end{bmatrix} \right\}$$

such that making P semidefinite in constraint 2 is the same as the quadratic constraint 1:

$$\begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0 \tag{2}$$

Specifying QC's on the input layer

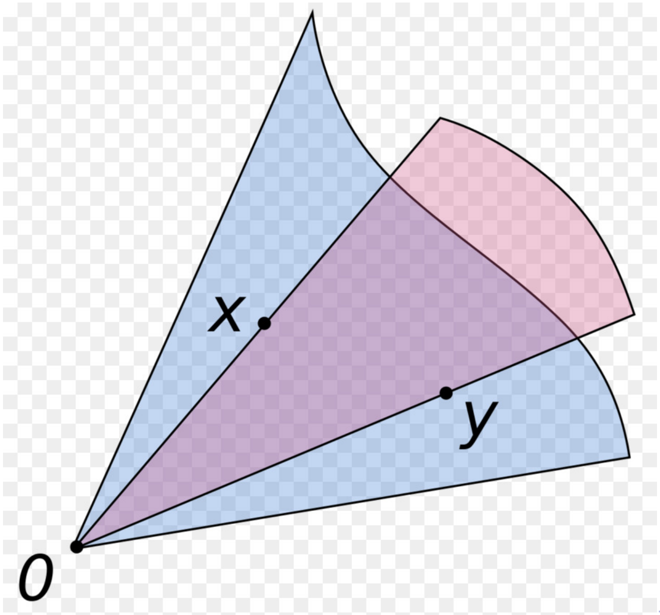
Definition: Suppose we are given a region $X \subset \mathbb{R}^d$. Let $\mathcal{P} \subset \mathbb{S}^{d+1}$ be all matrices such that, for all $P \in \mathcal{P}$, we have:

$$\begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0$$

We say an input region X satisfies QC's defined by $\mathcal{P}$. Here, $\mathcal{P}$ is a convex cone.

Note a **convex cone** is a set C such that $x, y \in C$ and $\alpha, \beta > 0$ yields $\alpha x + \beta y \in C$.

Convex Cone



Convex Cone and Semidefinite Constraints

The set of matrix $X \succeq 0$ ($X \in \mathbb{R}^{n\times n}$) is a convex cone in $\mathbb{R}^{n(n+1)/2}$.
Therefore, semidefinite matrix constraint can be treated as a convex constraint during SDP optimization procedure.

What’s next

So we have a way of specifying input constraints!
How do we handle output constraints?

Output Constraints

For a safety property of NN, it can be specified as a polytope constraint:

$$\mathcal{S} := \bigcap_{i=1}^{m} \{y : a_i^T y - b_i \leq 0\}$$

where y is the output of the NN and b is the worst-case upper bound predefined as safe.

Output Constraints

For a safety property of NN, it can be specified as a polytope constraint:

$$\mathcal{S} := \bigcap_{i=1}^{m} \{y : a_i^T y - b_i \leq 0\} \tag{3}$$

We can construct matrix S_i:

$$S_i := \begin{bmatrix} 0 & a_i \\ a_i^T & -2b_i \end{bmatrix}$$

such that restricting it to be **not positive**$(S \preceq 0)$ as constraint 4 has the same effect as constraint 3:

$$\begin{bmatrix} y \\ 1 \end{bmatrix}^T S_i \begin{bmatrix} y \\ 1 \end{bmatrix} \leq 0 \tag{4}$$

Output Constraints

Support output specifications which can be inner-approximated by finitely many quadratic inequalities $S_1, \ldots, S_m$, i.e.:

$$\mathcal{S} := \bigcap_{i=1}^{m} \left\{ y \in \mathbb{R}^{n_y} : \begin{bmatrix} y \\ 1 \end{bmatrix}^T S_i \begin{bmatrix} y \\ 1 \end{bmatrix} \leq 0 \right\}$$

What's left now? Quadratic constraints for activation functions!

QC for activation functions

Slope-restricted nonlinearities: A nonlinear function $\phi(x)$ is slope-restricted on $[\alpha, \beta]$ $(0 \leq \alpha \leq \beta)$ if the following condition holds:

$$(\phi(x)-\phi(x')-\alpha(x-x'))^T(\phi(x)-\phi(x')-\beta(x-x')) \leq 0$$

for any two pairs $(x, \phi(x))$ and $(x', \phi(x'))$.

QC for activation functions

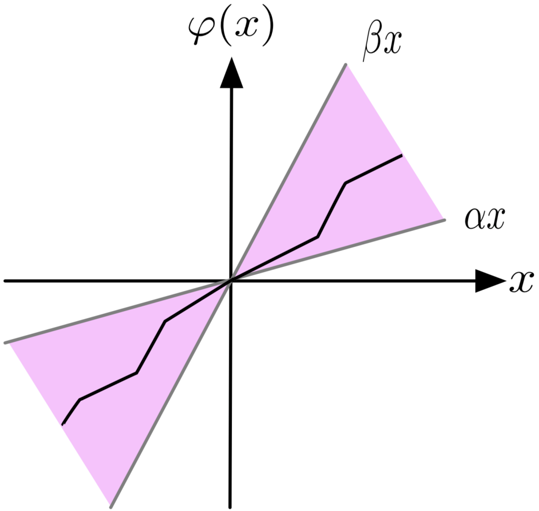


Figure: Slope restricted nonlinearity

QC for activation functions

Repeated nonlinearities (important): If we just use slope-restricted nonlinearities to encode the constraints of activation functions, we are actually ingoring dependency between the nodes in the earlier layers to the ones in the deeper layer. Therefore, for activation $\phi(x) = [\rho(x_1), ..., \rho(x_d)]^T$, slope restriction can be written as:

$$(\phi(x_i)-\phi(x_j)-\alpha(x_i-x_j)))(\phi(x_i)-\phi(x_j)-\beta(x_i-x_j))) \leq 0$$

for all $i, j = 1, ..., d, i \neq j$

QC for activation functions

Repeated nonlinearities (important): Such constraints can be rewritten as a semidefinite matrix constraint:

$$\begin{bmatrix} x \\ \phi \end{bmatrix}^T \begin{bmatrix} -2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2 \end{bmatrix} \begin{bmatrix} x \\ \phi \end{bmatrix} \geq 0$$

where $T = \sum_{1 \leq i \leq j \leq d} \lambda_{ij}(e_i - e_j)(e_i - e_j)^T$ with $\lambda_{ij} \geq 0$ and e_i is the i-th unit vector in $\mathbb{R}^d$.

QC for ReLU function

For a ReLU $\phi(x) = max(0, x)$ before considering repeated nonliearities, it can be relaxed into the following three quadratic constraints:

$$x\phi(x) = \phi^2(x), \phi(x) \geq x, \phi(x) \geq 0$$

QC for ReLU function

For a ReLU $\phi(x) = max(0, x)$, it can be relaxed into the following three quadratic constraints:

$$x\phi(x) = \phi^2(x), \phi(x) \geq x, \phi(x) \geq 0 \tag{5}$$

We can construct matrix Q such that restricting Q to be semidefinite as 6 will produce the same quadratic constraint as 5:

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T Q \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0 \tag{6}$$

QC for ReLU function

Formally, Q is defined as (not important):

$$Q := \begin{bmatrix} 0 & T & -v \\ T & -2T & v+\eta \\ -v^T & v^T+\eta^T & 0 \end{bmatrix}$$

Here $\eta, v \geq 0$ and T is semidefinite given by:

$$T = \sum_{i=1}^{d} \lambda_i e_i e_i^T + \sum_{i=1}^{d-1} \sum_{j>i}^{d} \lambda_{ij}(e_i - e_j)(e_i - e_j)^T$$

with $\lambda_i \in \mathbb{R}$ and $\lambda_{ij} \geq 0$.

Now we have semidefinite matrix constraints:
(1) P for input constraints;
(2) Q for activation constraints;
(3) S for output constraints;
Let's try to solve the NN verification problem.

SDP for NN

We can prove the safety property of NN with SDP problem by checking the feasibility of P and Q.

$$\textbf{Obj} : M_{in} + M_{mid} + M_{out} \preceq 0$$

M_{in}, M_{mid}, M_{out} are derived from P, Q, S by reshaping them into a huge matrix.
Conclusion: If there exist P and Q that makes it feasible, then we prove the safety property S always satisfied.

SDP for NN

Why exist $P, Q \Rightarrow$ proved safe?

$$\textbf{Obj} : M_{in} + M_{mid} + M_{out} \preceq 0$$

M_{in} semidefinite for sure!
M_{mid} semidefinite for sure!

SDP for NN

Why exist $P, Q =>$ proved safe?

$$M_{in} + M_{mid} + M_{out} \preceq 0$$

M_{in} semidefinite for sure!
M_{mid} semidefinite for sure!
M_{out} if obj is feasible, then not positive for sure!
Remember we define $S \preceq 0$ to be safe.

SDP for NN

$$A = \begin{bmatrix} W^0 & 0 & \cdots & 0 & 0 \\ 0 & W^1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & W^{\ell-1} & 0 \end{bmatrix} \quad b = \begin{bmatrix} b^0 \\ b^1 \\ \vdots \\ b^{\ell-1} \end{bmatrix},$$

$$B = \begin{bmatrix} 0 & I & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I \end{bmatrix} \quad C = \begin{bmatrix} 0 & 0 & \cdots & W^\ell \end{bmatrix}. \quad (24b)$$

$$M_{in} = \begin{bmatrix} I_{n_x} & 0 \\ 0 & 1 \end{bmatrix}^\top P \begin{bmatrix} I_{n_x} & 0 \\ 0 & 1 \end{bmatrix},$$

$$M_{mid} = \begin{bmatrix} A & b \\ B & 0 \\ 0 & 1 \end{bmatrix}^\top Q \begin{bmatrix} A & b \\ B & 0 \\ 0 & 1 \end{bmatrix},$$

$$M_{out} = \begin{bmatrix} C & b^\ell \\ 0 & 1 \end{bmatrix}^\top S \begin{bmatrix} C & b^\ell \\ 0 & 1 \end{bmatrix},$$

Figure: How M_{in}, M_{mid}, M_{out} are derived from P, Q, S.

SDP for NN

Let $S = \{y | a^T y - b \leq 0\}$ to be the output constraints. We can also have certified upper bound by finding P and Q to minimize b:

$$min \quad b$$

$$s.t. \quad M_{in} + M_{mid} + M_{out} \preceq 0$$

Evaluation

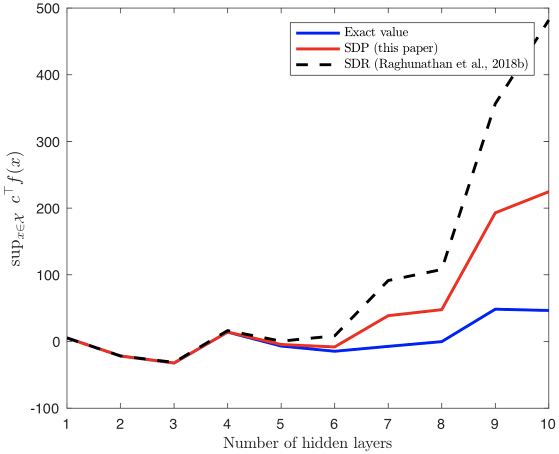


Figure: The certified upper bound in terms of different number of hidden layers. The network has 2 inputs, 10 outputs, and 100 hidden ReLU each layer.

Evaluation

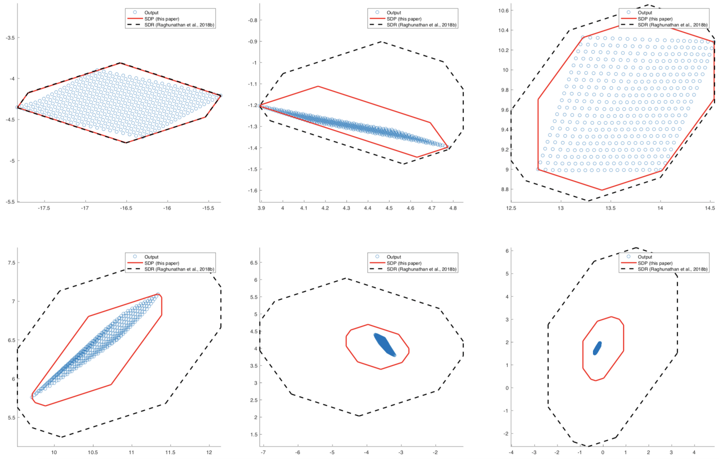


Figure: The output polytope obtained by SDP, SDR, and ground-truth output ranges. The number of hidden layers for each plot is 1, 2, 4, 6, 8, and 10.

Drawback

The main object function $M_{in} + M_{mid} + M_{out} \preceq 0$ is a huge matrix with $O(N^2)$ where N is the number of all hidden nodes. Therefore, SDP relaxation will take a lot of memory and cannot scale to large models.

Takeaway

(1) More meaningful constraints encoded in semidefinite matrix constraints, the better the tightness of the estimation will be.
(2) Require much more memory and computation to formulate and solve as SDP than linear programming formulation.

Questions?