ECE 285 Group Project 1–11

Neural Network Verification Via Semidefinite Relaxation

Aidan Epperly AEPPERLY@UCSD.EDU
UC San Diego Mathematics

Debalina Chowdhury DCHOWDHURY@UCSD.EDU
UC San Diego Electrical Engineering

Roshan Karande RSKARANDE@UCSD.EDU
UC San Diego Electrical Engineering

1. Abstract

Machine learning is a field that is growing increasingly important to society. One of the greatest flaws of many machine learning algorithms is a lack of robustness against adversarial attacks where a bad actors with knowledge of the neural network maliciously perturbs the input data of a neural network to cause a negative outcome. The solution to this problem is neural network verification, a procedure that proves that small changes in the input of a neural network will lead to correspondingly small changes in the output. In general, this problem is known to be NP-hard, but we will show how convex optimization, and in particular the powerful tool of semidefinite programming can be used to guarantee neural network verification.

2. Introduction

While just a few short years ago machine learning algorithms were mostly employed by governments and large tech firms, we have recently seen a rise in consumer available machine learning algorithms such as highly sophisticated chat bots and generative art bots. As machine learning algorithms become more and more widespread, the problem of robustness will become more and more pressing. While the robustness of machine learning algorithms is a large and difficult problem, there has been a great deal of recent progress in the field of neural network verification via semidefinite relaxations.

The neural network verification problem is to prove that for a given neural network, "small" perturbations in the input data will not cause "large" changes in the output. Without such a verification, neural networks can be highly vulnerable to adversarial attacks where some adversarial force with knowledge of the neural network could make a small but well chosen change to some input causing a neural network to catastrophically fail. One such example would be a neural network tasked with identifying stop signs for a self driving car. Without verification, it may be possible for an adversary to make some small, imperceptible change to a stop sign that never-the-less causes the self driving car to be unable to recognize it. We consider three different formulations for neural network verification, each of which is NP-hard, and show how they can be solved efficiently using semidefinite relaxations.

3. Related works

As with many modern applications of semidefinite programming, neural network verification can also be achieved using a linear programming relaxation as opposed to a semidefinite relaxation. These methods of neural network verification involving a linear programming relaxation are discussed in Wong and Kolter (2018); Dvijotham et al. (2018).

On the topic of semidefinite relaxation for neural network verification, all of the work we discuss in this paper focuses on simple convolutional neural networks. A paper by Dathathri et al. (2020) discusses verification methods larger neural networks. We also only focus on a pointwise method for proving robustness. A method for computing a Lipschitz bound can be found in a recent paper Shi et al. (2022) and a treatement of a probabilistic model can be found in a paper Pilipovsky et al. (2022).

4. Problem formulation

We will be reproducing the results from the paper *Efficient Neural Network Verification via Layer-based Semidefinite Relaxation and Linear Cuts* Batten et al. (2021). However, as this paper builds on the work of Raghunathan et al. (2018), we begin with a discussion of this paper. Then, we will discuss Fazlyab et al. (2020) which applies similar ideas to more complicated neural networks. Lastly, we will discuss the innovations of Batten et al. (2021) and discuss our numerical tests.

A Brief Note on Notation In order to unify notation throughout the various papers, we shall use the convention that if $x \in \mathbb{R}^d$, the x_i shall refer to the ith coordinate of x or the ith row if x is a matrix, and we shall use superscript x^i to denote that x^i is the ith term in a sequence.

4.1. Problem Formulation from Raghunathan et al. (2018)

We begin by discussing the problem formulation from *Semidefinite relaxations for certifying robustness to adversarial examples* Raghunathan et al. (2018). In this paper, the authors aim to prove robustness of ReLU neural networks (RNNs) against adversarial attacks using semidefinite relaxations.

ReLU neural networks as vector-valued functions. An L-layer RNN is a nonlinear vector valued function of the form

$$\begin{aligned} f(x^0) &:= W^L x^L + b^L, \\ x^{i+1} &:= \mathrm{ReLU}(\hat{x^{i+1}}), \\ \hat{x^{i+1}} &:= W^i x^i + b^i, \end{aligned}$$

where W^i is a linear map from $\mathbb{R}^{n_i}$ to $\mathbb{R}^{n_{i+1}}$, b^i is a bias vector in $\mathbb{R}^{n_i}$, $n_0 := d$ is the starting dimension, and $n_{L+1} := m$ is the output dimension. Given an RNN f, we can define a classification network where x_0 is assigned to the class associated with the highest output of the network:

$$y := \mathrm{argmax}_{i=1,\dots,m} f(x^0)_i.$$

We will only concern ourselves with this problem.

Adversarial robustness. In this paper, we consider a *clean* test input $\overline{x}$ and an *adversarially perturbed* input which we will denote by $A(\overline{x})$. We assume that the perturbed input is contained in the l_∞ norm ball of radius $\epsilon > 0$ from $\overline{x}$. If the correct classification of $\overline{x}$ is $\overline{y}$, then the *worst-case margin* of an incorrect classification y of $\overline{x}$ is

$$l_y^\star(\overline{x}, \overline{y}) := \max_{A(x) \in B_\epsilon(\overline{x})} (f(A(x))_y - f(A(x))_{\overline{y}}).$$

If $l_y^\star(\overline{x}, \overline{y}) < 0$, we call $(\overline{x}, \overline{y})$ certifiably robust. Computing $l_y^\star(\overline{x}, \overline{y})$ is generally very difficult, so we instead use a semidefinite relaxation $L_y(\overline{x}, \overline{y}) \geq l_y^\star(\overline{x}, \overline{y})$.

ReLU constraints as quadratic constraints. To obtain the semidefinite relaxation, the authors first note that we can rewrite $l_y^\star(\overline{x}, \overline{y})$ as

$$\begin{aligned} l_y^\star(\overline{x}, \overline{y}) = \max_{x^0, \ldots, x^L} & (c_y - c_{\overline{y}})^T x^L + b_{\overline{y}} - b_y \\ \text{subject to } & x^{i+1} = \mathrm{ReLU}(W^i x^i + b^i),\ i = 1, \ldots, L, && \text{(1a)} \\ & ||x^0 - \overline{x}||_\infty \leq \epsilon, && \text{(1b)} \end{aligned}$$

where $c_y = W_y^L$, $b_y = b_y^L$, and x^0 is the initial input $A(x)$. Constraint (1a) can then be relaxed to

$$\begin{aligned} x^{i+1} \geq 0, x^{i+1} \geq W^i x^i + b^i, \quad i = 1, 2, \ldots, L && \text{(2a)} \\ x^{i+1} \odot (x^{i+1} - W^i x^i - b^i) = 0, \quad i = 1, 2, \ldots, L. && \text{(2b)} \end{aligned}$$

In general, if $l^i \leq x^i \leq u^i$ for $l^i, u^i \in \mathbb{R}^{n_i}$, we can rewrite this constraint as

$$x^i \odot x^i \leq (l^i + u^i) \odot x^i - l^i \odot u^i.$$

This allows us to write constraint (1b) in the above form for $l^0 = \overline{x} - \epsilon \mathbf{1},\ u^0 = \overline{x} + \epsilon \mathbf{1}$. We can also obtain bounds $l^i \leq x^i \leq u^i$ using the recursion

$$\begin{aligned} l^0 = \overline{x} - \epsilon \mathbf{1}, u^0 = \overline{x} + \epsilon \mathbf{1}, \\ l^{i+1} = [W^i]_+ l^i + [W^i]_- u^i, u^{i+1} = [W^i]_+ u^i + [W^i]_- l^i, \end{aligned}$$

where $([M]_+)_{ij} = \max(M_{ij}, 0)$ and $([M]_-)_{ij} = \min(M_{ij}, 0)$.

Semidefinite relaxation. If we consider the polynomial lifting matrix $P := vv^T$, where $v = (1, x^0, x^1, \ldots, x^L)$, then all the constraints become linear in the entries of P. For notational convenience, we allow P is be indexed by monomials. For instance, in the case where $L = 1$, we have

$$P = \begin{bmatrix} P[1] & P[(x^0)^T] & P[(x^1)^T] \\ P[x^0] & P[x^0(x^0)^T] & P[x^0(x^1)^T] \\ P[x^1] & P[x^1(x^0)^T] & P[x^1(x^1)^T] \end{bmatrix}.$$

If we further relax the problem to just require $P \succeq 0$, we obtain the SDP relaxation

$$\begin{aligned} L_y^\star(\overline{x}, \overline{y}) := \max_P \ & (c_y - c_{\overline{y}})^T P[x^L] + b_y - b_{\overline{y}} \\ \text{subject to} \ & \text{for } i = 1, \dots, L \\ & P[x^{i+1}] \geq 0, \ P[x^{i+1}] \geq W^i P[x^i] + b^i, && \text{(3a)} \\ & \operatorname{diag}(P[x^{i+1}(x^{i+1})^T] - W^i P[x^i (x^{i+1})^T]) \\ & \quad - b^i \odot P[x^{i+1}] = 0, && \text{(3b)} \\ & \operatorname{diag}(P[x^i (x^i)^T]) - (l^i + u^i) \odot P[x^i] \\ & \quad + l^i \odot u^i \leq 0, && \text{(3c)} \\ & P[1] = 1, \ P \succeq 0. && \text{(3d)} \end{aligned}$$

This SDP relaxation, $L_y^\star(\overline{x}, \overline{y})$, provides an upper bound on $l_y^\star(\overline{x}, \overline{y})$, as the SDP relaxation allows for a larger set of feasible P matrices. Thus if $L_y(\overline{x}, \overline{y})$ is negative, then $l_y(\overline{x}, \overline{y})$ must also be negative. Thus, while there may be false negatives, crucially there will be no false positives. Thus, if the SDP relaxation above certifies the robustness of an RNN, then it is provably robust.

SDP Relaxation Versus Linear Programming There have existed linear programming relaxations for verifying the robustness of neural networks as discusses in Wong and Kolter (2018); Dvijotham et al. (2018). However, especially on certain classes of ReLU neural networks, the LP relaxations perform much worse than the SDP relaxation proposed in this paper.

Table 1: Fraction of non-certified examples on MNIST data set. Taken from Raghunathan et al. (2018)

	Grad-NN [23]	LP-NN [29]	PGD-NN
PGD-attack	15%	18%	9%
SDP-cert (this work)	**20%**	**20%**	**18%**
LP-cert	97%	22%	100%
Grad-cert	35%	93%	n/a

However, as we shall see later, there are some nice features of the LP relaxation that can be put to good use in an SDP relaxation.

4.2. Problem Formulation from Fazlyab et al. (2020)

Up until now, we have only discussed ReLU neural networks. While ReLU is a popular activation function, it is not the only option. The paper *Safety Verification and Robustness Analysis of Neural Networks via Quadratic Contraints and Semidefinite Programming* by Fazlyab et al. (2020) deals with the case for non-ReLU neural networks. We will now discuss the problem formulation as they describe it.

Problem Statement Let $f : \mathbb{R}^d \to \mathbb{R}^m$ be a nonlinear vector-valued function defined by a feed-forward neural network (note that we have not specified the activation function). Given a set $\mathcal{X} \subseteq \mathbb{R}^d$ of points, let

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

We let $\mathcal{S}_y \subseteq \mathbb{R}^m$ denote the *safety specification set*, which is to say that the network will be considered robust if $\mathcal{Y} \subseteq \mathcal{S}_y$. This inclusion is difficult to determine as it requires perfect knowledge of a likely-nonconvex set $\mathcal{Y}$. Instead, we will aim to find some convex set $\overline{y} \supseteq \mathcal{Y}$ such that the inclusion $\overline{\mathcal{Y}} \subseteq \mathcal{S}_y$ is easy to compute. While this setup is much more general than that of Raghunathan et al. (2018), we end up studying identical cases. Thus, to keep the notation consistent, we pick some fixed point $\overline{x} \in \mathbb{R}^d$, some $\epsilon > 0$, and let

$$\begin{aligned}\mathcal{X} &= \left\{x \in \mathbb{R}^d : ||x - \overline{x}||_\infty \leq \epsilon\right\}, \\ \overline{y} &= \text{argmax}_{i=1,\dots,m} f(\overline{x})_i, \\ \mathcal{S}_y &= \left\{z \in \mathbb{R}^m : z_{\overline{y}} \geq z_i,\ \forall i = 1, \dots, m\right\}.\end{aligned}$$

Given these definitions, the problem formulation becomes identical to the one described above. We will generally avoid using $\mathcal{S}_y$, and instead consider $\mathcal{S}_x := f^{-1}(\mathcal{S}_y)$ and study the inclusion of $\mathcal{X}$ in $\mathcal{S}_x$.

Quadratic Constraints One of the main improvements of this paper over the last is the development of a method to find reasonably tight quadratic constraints that can be used to replace the activation function constraints in the optimization problem. We will now go through the derivation process and the resulting SDP.

Given any set $Z \subseteq \mathbb{R}^d$, we define $\mathcal{P}_Z$ to be the set of all symmetric indefinite matrices P such that

$$(1, z)^T P (1, z) \geq 0, \quad \forall z \in Z.$$

Then, $Z \subseteq \bigcap_{P \in \mathcal{P}_Z} \left\{z \in \mathbb{R}^d : (z, 1)^T P\,(z, 1) \geq 0\right\}$. In such a case, Z is said to satisfy the QC defined by $\mathcal{P}_Z$. Similarly, if $\phi : \mathbb{R}^n \to \mathbb{R}^n$ is a function, then we let $\mathcal{Q}_Z^\phi$ be the set of all indefinite symmetric matrices Q such that

$$(z, \phi(z), 1)^T Q (z, \phi(z), 1) \geq 0, \quad \forall z \in Z.$$

In this case, ϕ is said to satisfy the QC defined by $\mathcal{Q}_Z^\phi$ on Z. Given some ϕ and X, we aim to explicitly compute $\mathcal{Q}_Z^\phi$. To do this, the authors consider three types of functions.

The first is so-called sector boundedness, where a function $\varphi : \mathbb{R} \to \mathbb{R}$ fixing 0 is said to be *sector bounded* in the sector $[\alpha, \beta]$ if

$$(\varphi(x) - \alpha x)(\varphi(x) - \beta x) \leq 0.$$

For a vector valued function, we consider n by n matrices A and B such that $A \succeq B$ in the Lowner ordering. A vector valued function $\phi : \mathbb{R}^n \to \mathbb{R}^n$ is said to be sector bounded in the sector $[A, B]$ if

$$(\phi(x) - Ax)^T (\phi(x) - Bx) \leq 0 \iff \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T \begin{bmatrix} -(A^T B + B^T A) & A^T + B^T & 0 \\ A + B & -2I & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0.$$

The authors then consider so-called slope restricted functions. A function $\phi : \mathbb{R}^n \to \mathbb{R}^n$ is slope restricted in the sector $[\alpha, \beta]$, where α and β are real numbers, if for any $x, y \in \mathbb{R}^n$, we have

$$(\phi(x) - \phi(y) - \alpha(x - y))^T (\phi(x) - \phi(y) - \beta(x - y)) \leq 0.$$

All of the commonly used activation functions are slope restricted, and many of them are sector bounded, so analysis of these functions is often useful for analysing neural networks. Assuming that φ is slope restricted on $[\alpha, \beta]$, we obtain the result that $\phi = (\varphi(x_1), \ldots, \varphi(x_n))$ satisfies the QC

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T \begin{bmatrix} -2\alpha\beta T & (\alpha+\beta)T & 0 \\ (\alpha+\beta)T & -2T & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0,$$

for all $x \in \mathbb{R}^n$ and

$$T = \sum_{1 \leq i < j \leq n} \lambda_{ij}(e_i - e_j)(e_i - e_j)^T,\ \lambda_{ij} \geq 0.$$

Lastly, the authors consider the case that the vector valued function ϕ is bounded by $a \leq \phi(x) \leq b$ for all $x \in \mathbb{R}^n$. In this case,

$$\begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2D & D(a+b) \\ 0 & (b+a)^T D & -2a^T Db \end{bmatrix} \begin{bmatrix} x \\ \phi(x) \\ 1 \end{bmatrix} \geq 0,$$

where D is diagonal and nonnegative.

Using these three classes of QCs, one can find quadratic constraints to use in place of the activation function constraints for the SDP relaxation.

SDP Relaxation Let f be an L layer neural network of the form

$$\begin{aligned} f(x^0) &:= W^L x^L + b^L, \\ x^{i+1} &:= \varphi(\hat{x^{i+1}}), \\ \hat{x^{i+1}} &:= W^i x^i + b^i, \end{aligned}$$

with $\varphi : \mathbb{R} \to \mathbb{R}$ being extended to $\mathbb{R}^n$ entrywise. Define $\mathbf{x} = (x^0, \ldots, x^L)$ and let $\mathbf{E}^k$ be such that $x^k = \mathbf{E}^k \mathbf{x}$. Moreover, let

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

$$\mathbf{B} = \begin{bmatrix} 0 & I_{n_1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I_{n_{L-1}} & 0 \\ 0 & 0 & \cdots & 0 & I_{n_L} \end{bmatrix}$$

Then, $f(x^0) = W^L \mathbf{E}^L \mathbf{x} + b^L$

$$\begin{aligned} f(x^0) &= W^L \mathbf{E}^L \mathbf{x} + b^L, \\ \mathbf{B}\mathbf{x} &= \varphi(\mathbf{A}\mathbf{x} + \mathbf{b}). \end{aligned}$$

If $\mathcal{X}$ satisfies the QC defined by $\mathcal{P}_{\mathcal{X}}$, $\mathcal{Z} = \{\mathbf{A}\mathbf{x} + \mathbf{b} : x \in \mathcal{X}\}$, and φ satisfies the QC defined by $\mathcal{Q}_{\mathcal{Z}}^{\varphi}$, then for

$$M_{\text{in}}(P) = \begin{bmatrix} \mathbf{E}^0 & 0 \\ 0 & 1 \end{bmatrix}^T P \begin{bmatrix} \mathbf{E}^0 & 0 \\ 0 & 1 \end{bmatrix}$$
$$M_{\text{mid}}(P) = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{B} & 0 \\ 0 & 1 \end{bmatrix}^T Q \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{B} & 0 \\ 0 & 1 \end{bmatrix}$$
$$M_{\text{out}}(P) = \begin{bmatrix} \mathbf{E}^0 & 0 \\ W^L\mathbf{E}^L & b^L \\ 0 & 1 \end{bmatrix}^T S \begin{bmatrix} \mathbf{E}^0 & 0 \\ W^L\mathbf{E}^L & b^L \\ 0 & 1 \end{bmatrix}$$

and S a given symmetric matrix

$$M_{\text{in}}(P) + M_{\text{mid}}(Q) + M_{\text{out}}(S) \preceq 0$$

for $(P, Q) \in (\mathcal{P}_{\mathcal{X}}, \mathcal{Q}_{\mathcal{Z}}^{\varphi})$ implies

$$(x, f(x), 1)^T S(x, f(x), 1) \leq 0, \quad \forall x \in \mathcal{X}.$$

Thus, if the set S_x is inner approximated by a finite intersection of QCs coming from matrices S_i for $i = 1, \ldots, k$, then the above can be used to check $\mathcal{X} \subseteq S_x$.

4.3. Problem Formulation from Batten et al. (2021)

The problem formulation from *Efficient Neural Network Verification via Layer-based Semidefinite Relaxation and Linear Cuts* Batten et al. (2021) is almost identical to that of Dathathri et al. (2020). The authors consider two well known convex relaxations of the verification problem and show how a combination of the two provides a provably better bound. The first is the same SDP relaxation from Dathathri et al. (2020), and the second is the LP relaxation from Wong and Kolter (2018). As we have already derived the SDP relaxation, we instead derive the LP relaxation.

Linear Programming Relaxation We again return to the ReLU constraint $x^{i+1} = \mathrm{ReLU}(W^i x^i + b^i)$. This time, instead of using a quadratic constraint to do a relaxation, we will replace the ReLU constraint with its convex hull. To see how this works, we first consider the one dimensional case. If we look at the convex hull of the graph of $\mathrm{ReLU}(x)$ for $x \in \mathbb{R}$, then this will be an infinitely large set. Thus, we instead suppose that $l \leq x \leq u$ for some $l \leq u$.

The graph of $\mathrm{ReLU}(x)$ will thus be compact and have a convex hull that is some triangle. The three sides will be given by the lines $y = 0$, $y = x$, and $y = k(x - l) + \mathrm{ReLU}(l)$ for $k = (\mathrm{ReLU}(u) - \mathrm{ReLU}(l))/(u - l)$. Extending this to the vector valued case, $x^{i+1} = \mathrm{ReLU}(W^i x^i + b^i)$ can be relaxed to

$$x^{i+1} \geq 0, \quad x^{i+1} \geq \hat{x}^{i+1}, \tag{4a}$$
$$x^{i+1} \leq k^i \odot (\hat{x}^{i+1} - \hat{l}^{i+1}) + \mathrm{ReLU}(\hat{l}^{i+1}), \tag{4b}$$
$$\hat{x}^{i+1} = W^i x^i + b^i, \quad \hat{l}^{i+1} \leq \hat{x}^{i+1} \leq \hat{u}^{i+1}, \tag{4c}$$

for $k^i = (\mathrm{ReLU}(\hat{u}^{i+1}) - \mathrm{ReLU}(\hat{l}^{i+1}))/(\hat{u}^{i+1} - \hat{l}^{i+1})$. The computation of $\hat{u}^{i+1}$ and $\hat{l}^{i+1}$ are covered in Wong and Kolter (2018) and can be derived using interval arithmetic and the norm ball constraint.

SDP Relaxation With Linear Cuts The first improvement made by the authors is to note that this LP relaxation has the advantage of being exact when a given neuron is always off or always on. In particular, if the upper bound is at most zero or the lower bound is at least 0, then the LP relaxation is exact. Thus, an improvment can be made by using the SDP relaxation derived in Raghunathan et al. (2018) and adding the "linear cut" constraint (4b). This yields the SDP

$$\begin{aligned}
\max_{P} \ & (c_y - c_{\overline{y}})^T P[x^L] + b_y - b_{\overline{y}} \\
\text{subject to} \ & \text{for } i = 1, \dots, L \\
& P[x^{i+1}] \geq 0, \ P[x^{i+1}] \geq W^i P[x^i] + b^i, && \text{(5a)} \\
& \operatorname{diag}(P[x^{i+1}(x^{i+1})^T] - W^i P[x^i (x^{i+1})^T]) \\
& \quad - b^i \odot P[x^{i+1}] = 0, && \text{(5b)} \\
& \operatorname{diag}(P[x^i (x^i)^T]) - (l^i + u^i) \odot P[x^i] \\
& \quad + l^i \odot u^i \leq 0, && \text{(5c)} \\
& P[x^{i+1}] \leq k^i \odot (W^i P[x^i] + b^i - \hat{l}^i) + \operatorname{ReLU}(\hat{l}^i), && \text{(5d)} \\
& P[1] = 1, \ P \succeq 0, && \text{(5e)}
\end{aligned}$$

where $k^i := \frac{\operatorname{ReLU}(\hat{u}^{i+1}) - \operatorname{ReLU}(\hat{l}^{i+1})}{\hat{u}^{i+1} - \hat{l}^{i+1}}$. The authors also derive an equivalent, but faster SDP by breaking into *layers*, defining a matrix $P^i := v^i (v^i)^T$ for each layer. This faster, layer-based SDP is given by

$$\begin{aligned}
\max_{P^0, \dots, P^{L-1}} \ & (c_y - c_{\overline{y}})^T P_{L-1}[x^L] + b_y - b_{\overline{y}} \\
\text{subject to} \ & \text{for } i = 1, \dots, L \\
& P^i[x^{i+1}] \geq 0, \ P^i[x^{i+1}] \geq W^i P^i[x^i] + b^i, && \text{(6a)} \\
& \operatorname{diag}(P^i[x^{i+1}(x^{i+1})^T] - W^i P^i[x^i (x^{i+1})^T]) \\
& \quad - b^i \odot P^i[x^{i+1}] = 0, && \text{(6b)} \\
& \operatorname{diag}(P^i[x^i (x^i)^T]) - (l^i + u^i) \odot P^i[x^i] \\
& \quad + l^i \odot u^i \leq 0, && \text{(6c)} \\
& P^i[x^{i+1}] \leq k^i \odot (W^i P^i[x^i] + b^i - \hat{l}^i) + \operatorname{ReLU}(\hat{l}^i), && \text{(6d)} \\
& P^i[1] = 1, \ P^i \succeq 0, && \text{(6e)} \\
& \text{for } j = 0, \dots, L-2, \\
& P^j[\hat{x}^{j+1}(\hat{x}^{j+1})^T] = P^{j+1}[\hat{x}^{j+1}(\hat{x}^{j+1})^T]. && \text{(6f)}
\end{aligned}$$

5. Experiment Results

We tried reproducing the results from the papers Raghunathan et al. (2018) (**SDR method**) and Batten et al. (2021) (**SDPNET method**) to verify the improvement in runtime and accuracy. The verification is done for $\epsilon = 0.1$ for a specific target on each sample. The repository — https://github.com/soc-ucsd/verification — was used to verify the results of the papers. A few bugs had to be fixed as the repository contained some redundant dependencies and deprecated or obsolete syntax.

The machine used to obtain the results had the following specifications — Processor (Intel i7 9750H), Memory (32 GB). The results have been verified on 20 samples due to computational constraints.

Figure 1: Sample run output.

After running the experiments, we had the following observations:

1. With **SDPNet method**, time(s) to verify one input instance is 110.4235 ± 35.82 seconds,
2. with **SDR method**, time(s) to verify one input instance is 1396.6515 ± 183.63 seconds,
3. **SDR method** is significantly slower in verification as compared to **SDPNet method**,

Sample	Target	Label	SDR Bound	SDR Time(sec)	SDPNet Bound	SDPNet Time(sec)
1	5	6	-8.2191	1256.3791	-8.9665	93.406
2	5	4	-6.7108	1242.9492	-8.6008	81.375
3	8	9	-1.8900	1372.6418	-3.2363	200.656
4	8	1	-2.9854	1748.2237	-4.2662	73.453
5	7	0	-7.3338	1363.0648	-9.2428	130.172

Figure 2: Results of the first few samples. Bound less than 0 implies verification.

4. As proposed in Batten et al. (2021), the bounds of **SDPNet method** are consistently better than the bounds of **SDR method**. We also observed that every instance that was verified with **SDR method** was also verified with **SDPNet method**.

While we attempted to implement the methods ourselves, we ran into the following problems:

1. The implementation of the papers from original authors is in Python for neural network work and MatLab (yalmip) for optimization. MatLab's Python interface is tempermental and does not work for newer versions of Python.

2. There is sparse documentation on the robust neural network training approaches used for model building.

3. The serialization format is Python and MatLab specific.

4. The authors implement the neural network in Tensorflow which if installed using a new version of Python may fail to interface with MatLab

5. We wanted to implement it in Julia as it has good support for building neural networks and also supports Mosek via JuMP. However, importing these serialized formats in our models proved to be challenging.

Many of the authors used a deep neural network for model building on datasets like MNIST. For deep networks, verification takes significantly longer. However, we know that shallow networks also achieve good accuracy on the MNIST dataset. It would be potentially interesting to test verification on shallow networks as that would yield faster results. In hindsight, a better approach would have been to verify results for simpler models with one or two hidden layers and thus few activations as this would help in faster prototyping.

References

Ben Batten, Panagiotis Kouvaros, Alessio Lomuscio, and Yang Zheng. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2184–2190. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/301. URL https://doi.org/10.24963/ijcai.2021/301. Main Track.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems*, 33:5318–5331, 2020.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, page 3, 2018.

Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 67(1):1–15, 2020.

Joshua Pilipovsky, Vignesh Sivaramakrishnan, Meeko M. K. Oishi, and Panagiotis Tsiotras. Probabilistic verification of relu neural networks via characteristic functions, 2022.

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

Zhouxing Shi, Yihan Wang, Huan Zhang, Zico Kolter, and Cho-Jui Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation, 2022.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*, pages 5286–5295. PMLR, 2018.