서울시립대학교
UNIVERSITY OF SEOUL

Evaluating Robustness of neural networks with mixed integer programming

정의 3-7 **선형계획법**

실수 공간 $\mathbb{R}^p$에서 정의된 벡터 $\boldsymbol{b}$, $\mathbb{R}^n$에서 정의된 벡터 $\boldsymbol{c}$, $\mathbb{R}^m$에서 정의된 벡터 $\boldsymbol{h}$와 행렬 $G \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{p \times n}$에 대하여, 다음의 최적화 문제를 **선형계획법**이라고 합니다.[9]

$$\begin{aligned} &\text{minimize}^{10} && \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ &\text{subject to} && (G\boldsymbol{x})_i \leq \boldsymbol{h}_i, \quad i = 1, \cdots, m \\ & && A\boldsymbol{x} = \boldsymbol{b} \end{aligned} \tag{3.5}$$

Problem definition

Optimization model 중 일부 변수(variables)가 정수(integer)라는 제한조건이 있을 때, 이를 integer program 이라 부른다.

$$
\begin{aligned}
\min_{x} \quad & f(x) \\
\text{subject to} \quad & x \in C \qquad \text{where } f : \mathbb{R}^n \to \mathbb{R}, \quad C \subseteq \mathbb{R}^n \quad \textit{and} \quad J \subseteq 1, \ldots, n. \\
& x_j \in \mathbb{Z}, j \in J
\end{aligned}
$$

앞선 식에서 J 가 다음을 만족 한다면, pure integer program 이라고 부른다.

$$J = \{\ 1, \ldots, n\ \}$$

이 절에서 논의되는 f 와 C 는 모두 convex라고 가정하도록 하자.

24-01 Definition · 모두를 위한 컨벡스 최적화 (convex-optimization-for-all.github.io)

Examples of Integer Programming

Knapsack problem

배낭 문제(Knapsack problem)은 배낭에 넣을 수 있는 부피가 한정되어 있어 배낭 안에 들어갈 item의 총 크기가 제약되어 있을 때, 최대의 가치(value)를 가지는 item들을 선택하도록 문제를 푸는 전통적인 조합 최적화 문제이다. 이 문제는 binary variable x 로 표현이 가능한데, j 번째 item을 선택했는 지 아닌지에 따라 x_j 가 0 혹은 1의 값을 가지게 된다.

$$
\begin{aligned}
\max_{x} \quad & c^{\mathsf{T}} x \\
\text{subject to} \quad & a^{\mathsf{T}} x \leq b \\
& x_j \in 0, 1, j = 1, \ldots, n
\end{aligned}
$$

c_j, a_j 는 각각 j 번째 item의 가치(value) 와 크기(volume) 를 나타낸다.

24-02 Examples of integer programs · 모두를 위한 컨벡스 최적화 (convex-optimization-for-all.github.io)

Contributions

- We demonstrate that, despite considering the full combinatorial nature of the network, our verifier can succeed at evaluating the robustness of larger neural networks, including those with convolutional and residual layers.
- We identify why we can succeed on larger neural networks with hundreds of thousands of units. First, a large fraction of the ReLUs can be shown to be either always active or always inactive over the bounded input domain. Second, since the predicted label is determined by the unit in the final layer with the maximum activation, proving that a unit never has the maximum activation over all bounded perturbations eliminates it from consideration. We exploit both phenomena, reducing the overall number of non-linearities considered.
- We determine for the first time the exact adversarial accuracy for MNIST classifiers to perturbations with bounded l1 norm. We are also able to certify more samples than the state-of-the-art and find more adversarial examples across MNIST and CIFAR-10 classifiers with different architectures trained with a variety of robust training procedures.

Definition of Robustness

Evaluating Adversarial Accuracy. Let $\mathcal{G}(x)$ denote the region in the input domain corresponding to all allowable perturbations of a particular input x. In general, perturbed inputs must also remain in the domain of valid inputs $\mathcal{X}_{valid}$. For example, for normalized images with pixel values ranging from 0 to 1, $\mathcal{X}_{valid} = [0, 1]^m$. As in Madry et al. (2018), we say that a neural network is robust to perturbations on x if the predicted probability of the true label $\lambda(x)$ exceeds that of every other label for all perturbations:

$$\forall x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid}) : \ \text{argmax}_i(f_i(x')) = \lambda(x) \tag{1}$$

Equivalently, the network is robust to perturbations on x if and only if Equation 2 is infeasible for x'.

$$(x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid})) \wedge \left(f_{\lambda(x)}(x') < \max_{\mu \in [1,n] \setminus \{\lambda(x)\}} f_\mu(x') \right) \tag{2}$$

where $f_i(\cdot)$ is the i^{th} output of the network. For conciseness, we call x *robust* with respect to the network if $f(\cdot)$ is robust to perturbations on x. If x is not robust, we call any x' satisfying the constraints a *valid* adversarial example to x. The *adversarial accuracy* of a network is the fraction of the test set that is robust; the *adversarial error* is the complement of the adversarial accuracy.

Minimum Adversarial Distortion

Evaluating Mean Minimum Adversarial Distortion. Let $d(\cdot, \cdot)$ denote a distance metric that measures the perceptual similarity between two input images. The minimum adversarial distortion under d for input x with true label $\lambda(x)$ corresponds to the solution to the optimization:

$$\min_{x'} d(x', x) \tag{3}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{4}$$

$$x' \in \mathcal{X}_{valid} \tag{5}$$

Formulating ReLU Let $y = \max(x, 0)$, and $l \leq x \leq u$. There are three possibilities for the *phase* of the ReLU. If $u \leq 0$, we have $y \equiv 0$. We say that such a unit is *stably inactive*. Similarly, if $l \geq 0$, we have $y \equiv x$. We say that such a unit is *stably active*. Otherwise, the unit is *unstable*. For unstable units, we introduce an indicator decision variable $a = \mathbb{1}_{x \geq 0}$. As we prove in Appendix A.1, $y = \max(x, 0)$ is equivalent to the set of linear and integer constraints in Equation 6.[3]

$$(y \leq x - l(1 - a)) \wedge (y \geq x) \wedge (y \leq u \cdot a) \wedge (y \geq 0) \wedge (a \in \{0, 1\}) \tag{6}$$

Formulating the Maximum Function Let $y = \max(x_1, x_2, \ldots, x_m)$, and $l_i \leq x_i \leq u_i$.

Proposition 1. Let $l_{max} \triangleq \max(l_1, l_2, \ldots, l_m)$. We can eliminate from consideration all x_i where $u_i \leq l_{max}$, since we know that $y \geq l_{max} \geq u_i \geq x_i$.

We introduce an indicator decision variable a_i for each of our input variables, where $a_i = 1 \implies y = x_i$. Furthermore, we define $u_{max,-i} \triangleq \max_{j \neq i}(u_j)$. As we prove in Appendix A.2, the constraint $y = \max(x_1, x_2, \ldots, x_m)$ is equivalent to the set of linear and integer constraints in Equation 7.

$$\bigwedge_{i=1}^{m} ((y \leq x_i + (1 - a_i)(u_{max,-i} - l_i)) \wedge (y \geq x_i)) \wedge \left(\sum_{i=1}^{m} a_i = 1\right) \wedge (a_i \in \{0, 1\}) \tag{7}$$

Progressive Bounds Tightening

- Tight bounds strengthen the problem formulation and thus improve solve times.
- If we can prove that the phase of a ReLU is stable, we can avoid introducing a binary variable.

Progressive Bounds Tightening

GETBOUNDSFORRELU(x, fs)

1 $\triangleright$ fs are the procedures to determine bounds, sorted in increasing computational complexity.
2 $l_{best} = -\infty$; $u_{best} = \infty$ $\triangleright$ initialize best known upper and lower bounds on x
3 **for** f in fs: $\triangleright$ carrying out progressive bounds tightening
4 **do** $u = f(x, boundType = upper)$; $u_{best} = \min(u_{best}, u)$
5 **if** $u_{best} \leq 0$ **return** (l_{best}, u_{best}) $\triangleright$ Early return: $x \leq u_{best} \leq 0$; thus $\max(x, 0) \equiv 0$.
6 $l = f(x, boundType = lower)$; $l_{best} = \max(l_{best}, l)$
7 **if** $l_{best} \geq 0$ **return** (l_{best}, u_{best}) $\triangleright$ Early return: $x \geq l_{best} \geq 0$; thus $\max(x, 0) \equiv x$
8 **return** (l_{best}, u_{best}) $\triangleright$ x could be either positive or negative.

The process of progressive bounds tightening is naturally extensible to more procedures. Kolter & Wong (2017); Wong et al. (2018); Dvijotham et al. (2018); Weng et al. (2018) each discuss procedures to determine bounds with computational complexity and tightness intermediate between IA and LP. Using one of these procedures in addition to IA and LP has the potential to further reduce build times.

Progressive Bounds Tightening

GETBOUNDSFORMAX(xs, fs)

1 ▷ fs are the procedures to determine bounds, sorted in increasing computational complexity.
2 $d_l = \{x : -\infty \text{ for } x \text{ in } xs\}$
3 $d_u = \{x : \infty \text{ for } x \text{ in } xs\}$
4 ▷ initialize dictionaries containing best known upper and lower bounds on xs
5 $l_{max} = -\infty$ ▷ l_{max} is the maximum known lower bound on any of the xs
6 $a = \{xs\}$
7 ▷ a is a set of active elements in xs that can still potentially take on the maximum value.
8 **for** f in fs: ▷ carrying out progressive bounds tightening
9 **do for** x in xs:
10 **if** $d_u[x] < l_{max}$
11 **then** $a.remove(x)$ ▷ x cannot take on the maximum value
12 **else** $u = f(x, boundType = upper)$
13 $d_u[x] = \min(d_u[x], u)$
14 $l = f(x, boundType = lower)$
15 $d_l[x] = \max(d_l[x], l)$
16 $l_{max} = \max(l_{max}, l)$
17 **return** (a, d_l, d_u)

Experiment

- MLP A: 1 hidden layer, 500 perceptrons per hidden layer.
- MLP B: 2 hidden layer, 200 perceptrons per hidden layer.
- CNN A: 2 convolutional layers, which have 16 and 32 filters, respectively.
- CNN B: 4 convolutional layers, which have 32, 32, 64, 64 filters, respectively.
- LP_d: the dual of a linear program
- SDP_d: the dual of a semidefinite relaxation
- ADV: Adversarial examples generated via Projected Gradient Descent (PGD)

Performance Comparisons: MILP

When removing *progressive tightening*, we directly use LP rather than doing IA first. When removing *using restricted input domain*, we determine bounds under the assumption that our perturbed input could be anywhere in the full input domain $\mathcal{X}_{valid}$, imposing the constraint $x' \in \mathcal{G}(x)$ only after all bounds are determined. Finally, when removing *using asymmetric bounds*, we replace l and u in Equation 6 with $-M$ and M respectively, where $M \triangleq \max(-l, u)$, as is done in Cheng et al. (2017); Dutta et al. (2018); Lomuscio & Maganti (2017). We carry out experiments on an MNIST classifier; results are reported in Table 1.

Table 1: Results of ablation testing on our verifier, where each test removes a single optimization. The task was to determine the adversarial accuracy of the MNIST classifier LP_d-CNN_A to perturbations with l_∞ norm-bound $\epsilon = 0.1$. *Build* time refers to time used to determine bounds, while *solve* time refers to time used to solve the main MILP problem in Equation 2 once all bounds have been determined. During solve time, we solve a linear program for each of the *nodes explored* in the MILP search tree.

†We exclude the initial build time required (3593s) to determine reusable bounds.

Optimization Removed	Mean Time / s			Nodes Explored		Fraction Timed Out
	Build	Solve	Total	Mean	Median	
(Control)	3.44	0.08	3.52	1.91	0	0
Progressive tightening	7.66	0.11	7.77	1.91	0	0
Using restricted input domain†	1.49	56.47	57.96	649.63	65	0.0047
Using asymmetric bounds	4465.11	133.03	4598.15	1279.06	105	0.0300

Performance Comparisons: Complete & Incomplete Verifiers

Verification Times, vis-à-vis the state-of-the-art SMT-based complete verifier `Reluplex`. Figure 1 presents average verification times per sample. All solves for our method were run to completion. On the l_∞ norm, we improve on the speed of `Reluplex` by two to three orders of magnitude.

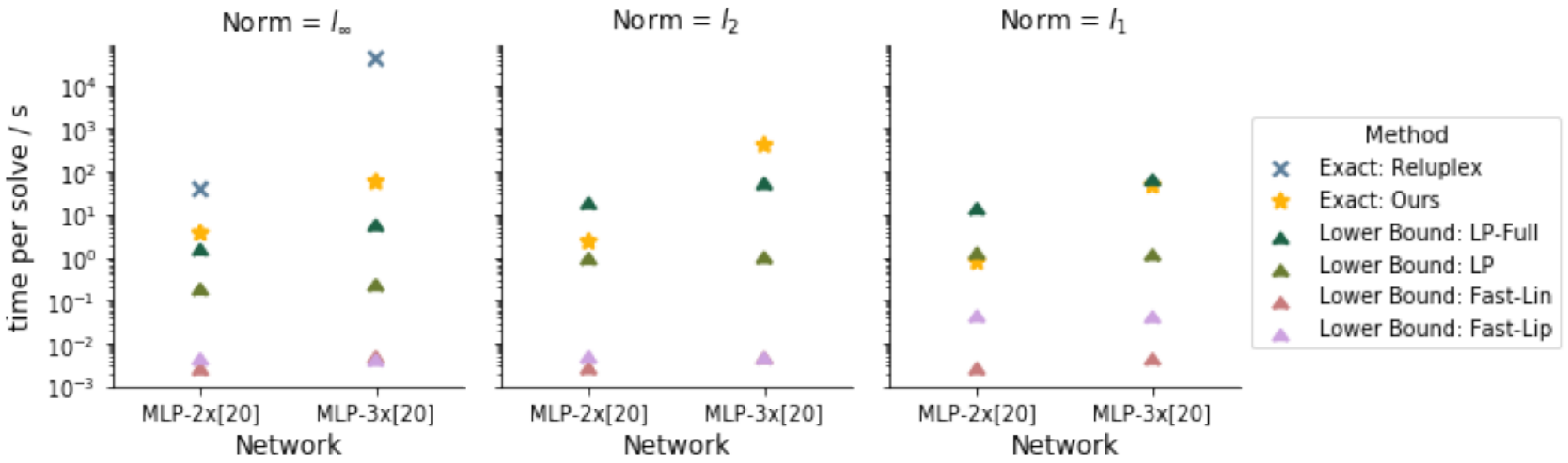

Figure 1: Average times for determining bounds on or exact values of minimum targeted adversarial distortion for MNIST test samples. We improve on the speed of the state-of-the-art complete verifier `Reluplex` by two to three orders of magnitude. Results for methods other than ours are from Weng et al. (2018); results for `Reluplex` were only available in Weng et al. (2018) for the l_∞ norm.

Determining Adversarial Accuracy

Table 2: Adversarial accuracy of MNIST and CIFAR-10 classifiers to perturbations with l_∞ norm bound ϵ. In every case, we improve on both 1) the lower bound on the adversarial error, found by PGD, and 2) the previous state-of-the-art (SOA) for the upper bound, generated by the following methods: [1] Kolter & Wong (2017), [2] Dvijotham et al. (2018), [3] Raghunathan et al. (2018). For classifiers marked with a ✓, we have a guarantee of robustness or a valid adversarial example for *every* test sample. Gaps between our bounds correspond to cases where the solver reached the time limit for some samples. Solve statistics on nodes explored are in Appendix F.1.

Dataset	Network	ϵ	Test Error	Certified Bounds on Adversarial Error					Mean Time / s
				Lower Bound		Upper Bound		No Gap?	
				PGD	Ours	SOA	Ours		
MNIST	$\text{LP}_\text{d}\text{-CNN}_\text{B}$	0.1	1.19%	2.62%	**2.73%**	4.45%[1]	**2.74%**		46.33
	$\text{LP}_\text{d}\text{-CNN}_\text{A}$	0.1	1.89%	4.11%	**4.38%**	5.82%[1]	**4.38%**	✓	3.52
	Adv-CNN_A	0.1	0.96%	4.10%	**4.21%**	—	**7.21%**		135.74
	Adv-MLP_B	0.1	4.02%	9.03%	**9.74%**	15.41%[2]	**9.74%**	✓	3.69
	$\text{SDP}_\text{d}\text{-MLP}_\text{A}$	0.1	4.18%	11.51%	**14.36%**	34.77%[3]	**30.81%**		312.43
	$\text{LP}_\text{d}\text{-CNN}_\text{A}$	0.2	4.23%	9.54%	**10.68%**	17.50%[1]	**10.68%**	✓	7.32
	$\text{LP}_\text{d}\text{-CNN}_\text{B}$	0.3	11.16%	19.70%	**24.12%**	41.98%[1]	**24.19%**		98.79
	$\text{LP}_\text{d}\text{-CNN}_\text{A}$	0.3	11.40%	22.70%	**25.79%**	35.03%[1]	**25.79%**	✓	5.13
	$\text{LP}_\text{d}\text{-CNN}_\text{A}$	0.4	26.13%	39.22%	**48.98%**	62.49%[1]	**48.98%**	✓	5.07
CIFAR-10	$\text{LP}_\text{d}\text{-CNN}_\text{A}$	$\frac{2}{255}$	39.14%	48.23%	**49.84%**	53.59%[1]	**50.20%**		22.41
	$\text{LP}_\text{d}\text{-RES}$	$\frac{8}{255}$	72.93%	76.52%	**77.29%**	78.52%[1]	**77.60%**		15.23

Determinants of Verification Time

Table 3: Determinants of verification time: mean verification time is 1) inversely correlated to the number of labels that can be eliminated from consideration and 2) correlated to the number of ReLUs that are not provably stable. Results are for $\epsilon = 0.1$ on MNIST; results for other networks are in Appendix F.2.

Network	Mean Time / s	Number of Labels Eliminated	Number of ReLUs			
			Possibly Unstable	Provably Stable		Total
				Active	Inactive	
$\text{LP}_\text{d}\text{-CNN}_\text{B}$	46.33	6.87	311.96	30175.65	17576.39	48064
$\text{LP}_\text{d}\text{-CNN}_\text{A}$	3.52	6.57	121.18	1552.52	3130.30	4804
Adv-CNN_A	135.74	3.14	545.90	3383.30	874.80	4804
Adv-MLP_B	3.69	4.77	55.21	87.31	257.48	400
$\text{SDP}_\text{d}\text{-MLP}_\text{A}$	312.43	0.00	297.66	73.85	128.50	500