Evaluating Robustness of Neural Networks with Mixed Integer Programming

Vincent Tjeng **Kai Xiao** **Russ Tedrake**
Massachusetts Institute of Technology
Cambridge, MA 02139
{vtjeng, kaix, russt}@mit.edu

Abstract

Neural networks have demonstrated considerable success on a wide variety of real-world problems. However, networks trained only to optimize for training accuracy can often be fooled by adversarial examples — slightly perturbed inputs that are misclassified with high confidence. Verification of networks enables us to gauge their vulnerability to such adversarial examples. We formulate verification of piecewise-linear neural networks as a mixed integer program. On a representative task of finding minimum adversarial distortions, our verifier is two to three orders of magnitude quicker than the state-of-the-art. We achieve this computational speedup via tight formulations for non-linearities, as well as a novel presolve algorithm that makes full use of all information available. The computational speedup allows us to verify properties on convolutional networks with an order of magnitude more ReLUs than networks previously verified by any complete verifier. In particular, we determine for the first time the *exact* adversarial accuracy of an MNIST classifier to perturbations with bounded l_∞ norm $\epsilon = 0.1$: for this classifier, we find an adversarial example for 4.38% of samples, and a certificate of robustness (to perturbations with bounded norm) for the remainder. Across all robust training procedures and network architectures considered, we are able to certify more samples than the state-of-the-art *and* find more adversarial examples than a strong first-order attack.

1 Introduction

Neural networks represent the state-of-the-art for classification of images [18, 40] and object localization in images [34]. However, networks that are trained only to optimize for training accuracy have been shown to be vulnerable to *adversarial examples*: perturbed inputs that are very similar to some regular input but for which the output is radically different [37].

There is now a large body of work proposing defense methods to produce classifiers that are more robust to adversarial examples. However, as long as a defense is evaluated only via attacks that find local optima (such as the Fast Gradient Sign Method (FGSM) [17] or Carlini and Wagner's attack (CW) [8]), we have no guarantee that the defense actually increases the robustness of the classifier produced. Defense methods thought to be successful when published have often been later found to be vulnerable to a new class of attacks. For instance, multiple defense methods are defeated in [7] by constructing defense-specific loss functions and in [1] by overcoming obfuscated gradients.

Fortunately, we *can* evaluate robustness to adversarial examples in a principled fashion. One option is to determine (for each test input) the minimum distance to the closest adversarial example, which we call the *minimum adversarial distortion* [6]. If our distance metric measures perceptual similarity well, an increase in mean minimum adversarial distortion indicates an improvement in robustness. Alternatively, we can determine the *adversarial test accuracy* [2], which is the proportion of the test

Preprint. Work in progress.

set for which no bounded perturbation causes a misclassification. If the bounded class of perturbations we have selected is meaningful, an increase in adversarial test accuracy indicates an improvement in robustness.[1]

In this paper, we present an efficient implementation of a mixed-integer linear programming (MILP) verifier for properties of piecewise-linear feed-forward neural networks. Our tight formulation for non-linearities and our novel presolve algorithm combine to minimize the number of binary variables in the MILP problem and dramatically improve its numerical conditioning. On a representative task of finding minimum adversarial distortions, we are two to three orders of magnitude faster than the state-of-the-art Satisfiability Modulo Theories (SMT) based verifier, Reluplex [27].

We make the following key contributions:

- We demonstrate that, despite considering the full combinatorial nature of the network, our verifier *can* succeed on larger neural networks — including those with convolutional layers — when evaluating the robustness of these networks to bounded perturbations.
- We identify *why* we can succeed on larger neural networks with thousands of units. Firstly, a large fraction of the ReLUs can be shown to be either always active or always inactive over the bounded input domain. Secondly, 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 allows us to eliminate it from consideration. We fully exploit both phenomena, reducing the overall number of non-linearities we need to consider.
- We determine — for the first time — the exact adversarial accuracy for MNIST classifiers with bounded l_∞ norm ϵ. For example, for a particular classifier trained to be robust to attacks with $\epsilon = 0.1$, we *guarantee* an adversarial test accuracy of 95.62%, and provide a valid adversarial example for the remaining 4.38% of test inputs. The adversarial test accuracy represents an increase of 1.44 percentage points over the highest previous guarantee found in [28].
- We are able to certify more samples than the state-of-the-art *and* find more adversarial examples than a strong attack across networks with different architectures trained with a variety of robust training procedures.

Our code is available at `github.com/vtjeng/MIPVerify.jl`

2 Related Work

Our work relates most closely to other work on verification of piecewise-linear neural networks; [5] provides a good overview of the field.

Verification procedures can be categorized as *complete* or *incomplete*. To understand the difference between the two categories of verification procedures, we consider the example of evaluating adversarial accuracy. As in [28], we call the exact set of all final-layer activations that can be achieved by applying a bounded perturbation to the input the *adversarial polytope*.

Incomplete verifiers reason over an outer approximation of the adversarial polytope. This means that, when using incomplete verifiers, the answer to some queries about the adversarial polytope may not be decidable. In particular, incomplete verifiers can only certify robustness for a fraction of robust input; the status for the remaining input is undetermined.

In contrast, complete verifiers reason over the exact adversarial polytope. Given sufficient time, a complete verifier can provide a definite answer to any query about the adversarial polytope. In the context of adversarial accuracy, complete verifiers will obtain a valid adversarial example or a certificate of robustness for *every* input. When a time limit is set, complete verifiers behave like incomplete verifiers, and resolve only a fraction of queries. However, complete verifiers do allow users to answer a larger fraction of queries by extending the set time limit.

Incomplete verifiers for evaluating network robustness employ a range of techniques, including duality [13, 33], layer-by-layer approximations of the adversarial polytope [42], discretizing the

[1]The two measures are related: a solver that can find certificates for bounded perturbations can be used iteratively (in a binary search process) to find minimum distortions.

search space [24], abstract interpretation [16], bounding the local Lipschitz constant [41], or bounding the activation of the ReLU with linear functions [41]. We highlight in particular the verifier presented by [28], which expresses an outer approximation of the adversarial polytope using a set of linear constraints, and generates robustness certificates by solving a linear program. Any certificate generated by this verifier will be quickly generated by our verifier, since we solve the same linear program as the first step of our verification procedure.

Complete verifiers typically employ either MILP solvers [10, 12, 15, 29] or SMT [6, 14, 27, 35]. When we compare our MILP-based approach with the state-of-the-art SMT-based approach (Reluplex) on the task of finding minimum adversarial distortions, we find that our verifier is two to three orders of magnitude faster. Our approach improves upon existing MILP-based approaches with a tighter formulation for non-linearities and a novel presolve algorithm that makes full use of all information available. The impact of these optimizations is significant: our ablation tests reveal an improvement in solve times of several orders of magnitude over a naïvely implemented MILP-based approach.

Other authors have used complete verifiers to verify properties of MNIST classifiers, determining minimum adversarial distortions [6] and verifying robustness to bounded perturbations [10, 15]. However, the largest MNIST classifier that any of these papers verify has only 200 units. In contrast, our efficiently implemented verifier is able to handle a network with more than 4,000 units.

A complementary line of research to verification is in training networks that are *designed* to be robust to bounded perturbations. Regular training simply attempts to minimize the loss for each example. In contrast, robust training attempts to minimize the “worst-case loss” [28] for each example — that is, the maximum loss over all bounded perturbations of that example. Since calculating the exact worst-case loss can be computationally costly, robust training procedures typically instead minimize an estimate of the worst-case loss.

Adversarial training [17] minimizes a *lower bound* of the worst-case loss. This lower bound is provided by the loss at some adversarial example generated by heuristic attacks. Adversarial training has shown some promise, but networks trained to be robust to weaker attacks such as the FGSM [17] have been shown to remain vulnerable to stronger attacks [38]. The core issue here is that minimizing a lower bound of the loss does not guarantee that the loss itself is minimized.

In contrast, certified training approaches [23, 28, 33] minimize an *upper bound* of the worst-case loss (or a surrogate for the upper bound). At the end of the training process, the upper bound at a given input (if it is below a threshold) can be used as to certify the robustness of the network at that input. Unfortunately, the conservatism of the upper bound means that some input that *is* robust to bounded perturbations cannot be certified.

Complete verifiers such as ours can augment robust training procedures by resolving the status of input for which heuristic attacks cannot find an adversarial example but incomplete verifiers cannot find a certificate. This enables more accurate comparisons between different training procedures.

3 Background and Notation

We denote a neural network by the function $f(\cdot;\theta) : \mathbb{R}^m \to \mathbb{R}^n$ that is parameterized by a (fixed) vector of weights θ. For a classifier, the output layer has a neuron for each target class the network is designed to predict.

Verification as solving an MILP. Borrowing from the notation in [5], the general problem of verification is to determine whether some property P on the output of a neural network holds for all input in a bounded input domain $\mathcal{C} \subseteq \mathbb{R}^m$. For the verification problem to be expressible as solving an MILP, P must be expressible as the conjunction or disjunction of linear properties $P_{i,j}$ over some set of polyhedra $\mathcal{C}_i$, where $\mathcal{C} = \cup \mathcal{C}_i$.

In addition, $f(\cdot)$ must be composed of piecewise-linear layers. This is not a particularly restrictive requirement: piecewise-linear layers include layers that are linear transformations (such as fully-connected, convolution, and average-pooling layers) and layers that use piecewise-linear functions (such as ReLU or maximum-pooling layers). We provide details on how to express these piecewise-linear functions in Section 4.1. [5] observes that batch normalization [26] or dropout [36] are also linear transformations at *evaluation time*, and we note that the "shortcut connections" used in architectures such as ResNet [22] are also linear.

4 Formulating Robustness Evaluation of Classifiers as an MILP

In general, perturbed inputs must always remain in the domain of valid inputs $\mathcal{X}_{valid}$. For example, for normalized images with pixel values ranging from 0 to 1, $\mathcal{X}_{valid} = [0, 1]^m$.

Evaluating Adversarial Accuracy. Let $\mathcal{G}(x)$ denote the region in the input domain corresponding to all allowable perturbations of a particular input x. As in [28, 30], 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:

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

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

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

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 simply the complement of the adversarial accuracy.

As long as $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ can be expressed as the union of a set of polyhedra, the feasibility problem can be expressed as an MILP. The three robust training procedures we consider [28, 30, 33] are designed to be robust to perturbations with bounded l_∞ norm, and the l_∞-ball of radius ϵ around each input x can be succinctly represented by the set of linear constraints $\mathcal{G}(x) = \{x' \mid \forall i : -\epsilon \leq (x - x')_i \leq \epsilon\}$.

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:

$$\begin{aligned} & \min_{x'} d(x', x) && (3) \\ \text{subject to} \quad & \operatorname{argmax}_i(f_i(x')) \neq \lambda(x) && (4) \\ & x' \in \mathcal{X}_{valid} && (5) \end{aligned}$$

We can *target* the attack to generate an adversarial example that is classified in one of a set of target labels T by replacing Equation 4 with $\operatorname{argmax}_i(f_i(x')) \in T$.

The most prevalent distance metrics in the literature for generating adversarial examples are the l_1 [8, 9], l_2 [37], and l_∞ [17, 32] norms. All three can be expressed in the objective without adding any additional integer variables to the model [4]; full details can be found in Appendix A.3.

4.1 Formulating Piecewise-linear Functions in the Classifier

Tight formulations of the rectifier and maximum functions are critical to good performance of the MILP solver; we thus present these formulations in detail with accompanying proofs. For the interested reader, formulations for general piecewise linear functions are available in [25].

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.

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

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.

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

4.2 Presolve to Determine Bounds

We previously assumed that we had some element-wise bounds on the inputs to non-linearities. In practice, we have to carry out a presolve step to determine these bounds. Determining tight bounds is critical for problem tractability: tight bounds strengthen the problem formulation and thus improve solve times [39]. For instance, if we can prove that the phase of a ReLU is stable, we can avoid introducing a binary variable. More generally, loose bounds on input to some unit will propagate downstream, leading to units in later layers having looser bounds.

We used two procedures (in increasing order of speed) to determine bounds: LINEAR PROGRAMMING (LP), and INTERVAL ARITHMETIC (IA), also used in [10, 12, 28]. Implementation details are in Appendix A.4. Unfortunately, faster procedures achieve efficiency by compromising on tightness of bounds. We thus face a trade-off between higher *build times* (to determine tighter bounds to inputs to non-linearities), and higher *solve times* (to solve the main MILP problem in Equation 2 or Equation 3-5).

While a degree of compromise is inevitable, our knowledge of the non-linearities used in our network allows us to reduce average build times without affecting the strength of the problem formulation. The key observation is that, for piecewise-linear non-linearities, there are thresholds beyond which further refining a bound will not improve the problem formulation. For example, once the lower bound on the input to a ReLU can be shown to be positive, it is guaranteed to be stably active, and there is no need to attempt to prove a stronger lower bound.

With this in mind, we adopt a progressive bounds tightening approach: we begin by determining coarse bounds using fast procedures and only spend time refining bounds using procedures with higher computational complexity if doing so could provide additional information to improve the problem formulation.[2] We also allow users to specify a *maximum build effort* — the procedure with the highest computational complexity that will be used to determine bounds. This is useful in the case that a fast procedure with low complexity provides sufficiently strong bounds for quick solves.

5 Experiments

Dataset. All experiments are carried out on classifiers for the MNIST dataset of handwritten digits.

Architectures. We conduct experiments on a range of feed-forward networks. In all cases, ReLUs follow each layer except the output layer. **MLP-**$m\times[n]$ refers to a multilayer perceptron with m hidden layers and n units per hidden layer. We further abbreviate MLP-$1\times[500]$ and MLP-$2\times[200]$ as **MLP-A** and **MLP-B** respectively. **CNN** refers to the ConvNet architecture used for the robust MNIST classifier in [28]. The network has two convolutional layers (stride length 2) with 16 and 32 filters respectively (size 4×4 in both layers), followed by a fully-connected layer with 100 units.

Training Methods. We conduct experiments on networks trained with a regular loss function and networks trained to be robust. Networks trained to be robust are identified by a prefix corresponding to the method used to approximate the worst-case loss: **LPd**[3] when the dual of a linear program is used, as in [28]; **SDPd** when the dual of a semidefinite relaxation is used, as in [33]; and **Adv** when adversarial examples generated via Projected Gradient Descent (PGD) are used, as in [30].

Full details on each network verified are provided in Appendix B.1.

Experimental Setup. We run experiments on a modest 8 CPUs@2.20 GHz with 8GB of RAM. Appendix B.2 provides additional details about the computational environment. Maximum build effort is LP. Unless otherwise noted, we report a timeout if solve time for some input exceeds 1200s.

5.1 Performance Comparisons

We evaluate the performance of our verification approach on the task of finding minimum targeted adversarial distortions. Approaches included for comparison are 1) `Reluplex` [27], a complete verifier also able to find the true minimum distortion; and 2) LP[4], `Fast-Lip`, `Fast-Lin` [41], and

[2] As a corollary, we always use only IA for the output of the first layer, since the independence of network input implies that IA is provably optimal for that layer.

[3] This is unrelated to the procedure to determine bounds named LP.

[4] This is unrelated to the procedure to determine bounds named LP, or the traning procedure LPd.

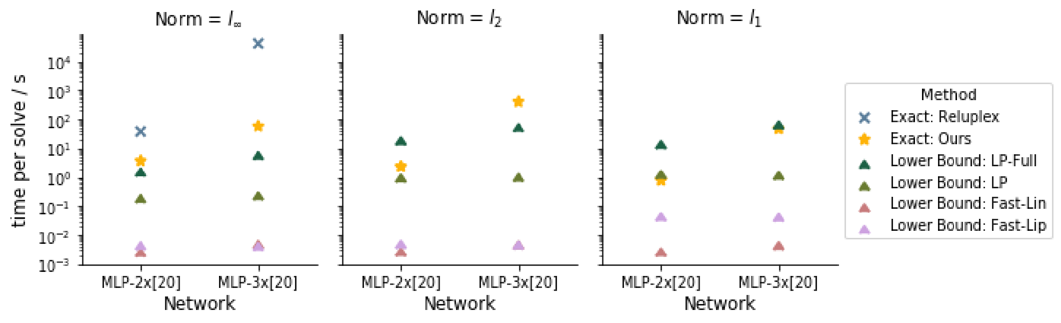


Figure 1: Average times for determining bounds on / exact values of the minimum targeted adversarial distortion. We improve on the speed of the state-of-the-art complete verifier `Reluplex` by two to three orders of magnitude. Results for methods other than ours are from [41]; results for `Reluplex` were only available in [41] for the l_∞ norm.

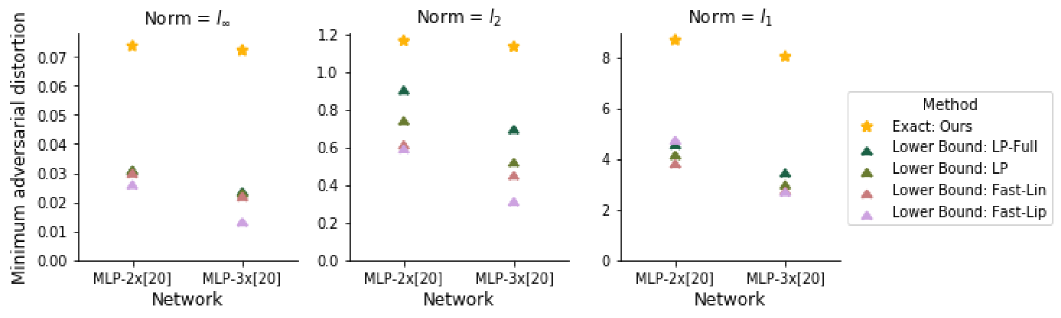


Figure 2: Average of bounds on / exact values of the minimum targeted adversarial distortion. The gap between the true minimum adversarial distortion and the best lower bound is significant and increases for deeper networks. Again, results for methods other than ours are from [41].

`LP-full` [28], incomplete verifiers that provide a certified *lower* bound on the minimum distortion. We remove the time limit of 1200s for our method for these experiments.

For each sample in the MNIST test set, our complete verifier finds the true minimum distortion by providing an adversarial example with that distortion, *and* a certificate that no input closer than that is adversarial.

To ensure that we are able to accurately compare the verification times for our method with those reported for the methods in [41], we did our best to match the authors' experimental setup. Specifically, we used the same network weights, test samples, and target classes as in [41]. In addition, we conducted the experiments in single thread mode as the authors do. Finally, as in [41], the results presented represent an average over the first 100 samples of the MNIST test set, excluding the samples (among these first 100) that are incorrectly classified.

Verification Times. Figure 1 presents average verification times per sample. On the l_∞ norm, we improve on the speed of the state-of-the-art complete verifier `Reluplex` by two to three orders of magnitude. For the l_1 norm, our speed is even comparable to `LP-full` — a method which only provides a lower bound.

Minimum Targeted Adversarial Distortions. Figure 2 compares lower bounds provided by the incomplete verifiers to the exact value we obtain. Even on these small networks, the gap between the best lower bound and the true minimum adversarial distortion is significant. This corroborates the observation in [33] that incomplete verifiers provide weak bounds if the network they are applied to is not optimized for that verifier. For example, under the l_∞ norm, the best certified lower bound is less than half of the true minimum distortion. To put this in context: with incomplete verifiers, we might only be able to verify a network to be robust to perturbations with l_∞ norm-bound $\epsilon = 0.05$, when the network is in fact robust to perturbations with l_∞ norm-bound $\epsilon = 0.1$.

Table 1: Adversarial accuracy of 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 method in the paper cited. 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. Error is over the full MNIST test set of 10,000 samples; mean verification time includes solves reaching the time limit.
† The PGD error we report differs significantly from that in [33] since we impose the constraint that $x' \in \mathcal{X}_{valid}$.

Network	ϵ	Test Error	Certified Bounds on Adversarial Error					Mean Time / s
			Lower Bound		Upper Bound		No Gap?	
			PGD	Ours	SOA	Ours		
LPd-CNN	0.1	1.89%	4.11%	**4.38%**	5.82%[28]	**4.38%**	✓	3.52
Adv-CNN	0.1	0.96%	4.10%	**4.21%**	—	**7.21%**		135.74
Adv-MLP-B	0.1	4.02%	9.03%	**9.68%**	15.41%[13]	**9.68%**	✓	3.69
SDPd-MLP-A	0.1	4.18%	†11.51%	**14.36%**	34.77%[33]	**30.81%**		312.43
LPd-CNN	0.2	4.23%	9.54%	**10.68%**	17.50%[28]	**10.68%**	✓	7.32
LPd-CNN	0.3	11.40%	22.70%	**25.79%**	35.03%[28]	**25.79%**	✓	5.13
LPd-CNN	0.4	26.13%	39.22%	**48.98%**	62.49%[28]	**48.98%**	✓	5.07

5.2 Determining Adversarial Accuracy of Robust Networks

We use our verifier to determine the adversarial accuracy of classifiers trained by a range of robust training procedures. Table 1 presents the test error and estimates of the adversarial error for these classifiers.[5] We verified a range of networks trained to be robust to attacks with bounded l_∞ norm $\epsilon = 0.1$, as well as networks trained to be robust to larger attacks of $\epsilon = 0.2, 0.3$ and 0.4.

Lower bounds on the adversarial error are proven by providing adversarial examples for input that is not robust. We compare the number of samples for which we successfully find adversarial examples to the number for PGD, a strong first-order attack. Upper bounds on the adversarial error are proven by providing certificates of robustness for input that is robust. We compare our upper bounds to the previous state-of-the-art for each network.

While performance depends on the training method and architecture, we improve on both the lower and upper bounds for *every* network tested.[6] For lower bounds, we successfully find an adversarial example for every test sample that PGD finds an adversarial example for. In addition, we observe that PGD 'misses' some valid adversarial examples: it fails to find these adversarial examples even though they are within the norm bounds. As the last three rows of Table 1 show, PGD misses for a larger fraction of test samples when ϵ is larger. We also found that PGD is far more likely to miss for some test sample if the minimum adversarial distortion for that sample is close to ϵ; this observation is discussed in more depth in Appendix C.

For upper bounds, we improve on the bound on adversarial error even when the upper bound on the worst-case loss — which is used to generate the certificate of robustness — is *explicitly* optimized for during training (as is the case for LPd and SDPd training).

Most importantly, we are able to determine the **exact adversarial accuracy** for LPd-CNN and Adv-MLP-B for all ϵ tested, finding either a certificate of robustness or an adversarial example for *every* test sample. For LPd-CNN and Adv-MLP-B, running our verifier over the full test set takes approximately 10 hours — the same order of magnitude as the time to train each network on a single GPU. Better still, verification of individual samples is fully parallelizable.

[5] As mentioned in Section 2, complete verifiers will obtain either a valid adversarial example or a certificate of robustness for every input given enough time. However, we do not *always* have a guarantee of robustness or a valid adversarial example for every test sample since we terminate the optimization at 1200s to provide a better picture of how our verifier performs within reasonable time limits.

[6] On SDPd-MLP-A, the verifier in [33] finds certificates for 372 samples for which our verifier reaches its time limit.

Table 2: 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 $\epsilon = 0.1$, and are aggregated over the full MNIST test set.

Network	Mean Time / s	Average Number of Labels Eliminated	Number of ReLUs			
			Possibly Unstable	Provably Stable		Total
				Active	Inactive	
LPd-CNN	3.52	6.57	121.18	1552.52	3130.30	4804
Adv-CNN	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
SDPd-MLP-A	312.43	0.00	297.66	73.85	128.50	500

Table 3: Results of ablation testing for our verifier. The task was to determine the adversarial accuracy of LPd-CNN to perturbations with l_∞ norm-bound $\epsilon = 0.1$. In each case, we removed a single optimization made in our verifier. *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.
†Some bounds are reusable across different input since input domain is no longer restricted; we thus exclude the initial build time required (3593s) to determine reusable bounds. ‡Ablation test was run only on first 100 samples.

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

5.2.1 Observations on Determinants of Verification Time

Ceteris paribus, we might expect verification time to be correlated to the total number of ReLUs, since the solver may need to explore both possibilites for the phase of each ReLU. However, there is clearly more at play: even though LPd-CNN and Adv-CNN have identical architectures, verification time for Adv-CNN is two orders of magnitude higher.

The key lies in the restricted input domain $\mathcal{G}(x)$ for each test sample x. When input is restricted to $\mathcal{G}(x)$, we can prove that many ReLUs are stable (with respect to $\mathcal{G}$). Furthermore, we can eliminate some labels from consideration by proving that the upper bound on the output neuron corresponding to that label is lower than the lower bound for some other output neuron. As the results in Table 2 show, a significant number of ReLUs can be proven to be stable, and a significant number of labels can be eliminated from consideration. Rather than being correlated to the *total* number of ReLUs, solve times are instead more strongly correlated to the number of ReLUs that are not provably stable, as well as the number of labels that cannot be eliminated from consideration.

For networks not trained to be robust, we observed that very few ReLUs can be proven to be stable with respect to $\mathcal{G}$. We posit that the large number of ReLUs that can be proven to be stable with respect to $\mathcal{G}$ for a robust classifier is linked to the increased local stability of the classifier (with respect to perturbations described by $\mathcal{G}$), even as the classifier retains global expressiveness. We explore this connection between robust training and the stability of ReLUs in greater detail in Appendix D.

5.3 Ablation Testing

We isolated the impact of each optimization in our implementation by conducting thorough ablation tests. Results are reported in Table 3. When removing *progressive tightening*, we directly used the procedure specified by the maximum build effort (LP in this case), skipping less computationally complex procedures. 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 [10, 12, 29].

When removing *progressive tightening*, solve times match those of the control since the final MILP model generated is identical. However, build times more than double: we lose the performance advantage of skipping a costly LP solve when IA would have provided sufficient information. When removing *using restricted input domain* or *using asymmetric bounds*, solve times are 3-4 orders of magnitude higher since the formulation size increases significantly. In particular, when removing *using asymmetric bounds*, we introduce a binary variable for each ReLU, since we are no longer able to prove that any ReLU is in the stable phase. These tests emphasize how critical it is to make full use of the information available on bounds when building the MILP model.

6 Discussion

In this paper, we present a complete verifier for piecewise-linear neural networks built on expressing verification as a mixed integer linear program. While our verifier *is* reasonably fast, we encourage users to use even faster methods (such as heuristic attacks or incomplete verifiers) as a first pass, using our verifier only for cases that remain undecided. We do expect our verifier to become more useful when larger attacks are allowed, since the gap between lower bounds provided by heuristic attacks and upper bounds provided by incomplete verifiers increases with attack size.

We have focused on evaluating networks against the class of perturbations that they are *designed* to be robust to. However, *defining* a class of perturbations that better captures images perceptually similar to the original image remains an important direction of research. We note that our verifier *is* able to handle new classes of perturbations as long as the set of perturbed images can be described by a union of polytopes in the input space. One such class of perturbations that we can handle is convolutions of the original input image.

We close with some thoughts on improving verifiability of neural networks. As discussed in Section 5.2.1, increasing the number of ReLUs that can be proven to be locally stable makes verification faster. We also observed that sparsifying weights promotes verifiability: as discussed in Appendix E, simply naïvely sparsifying SDPd-MLP-A reduces the number of timeouts by an order of magnitude without significantly affecting test error. Adopting a more principled sparsification approach (for example, l_1 regularization during training, or pruning and retraining [21]) could potentially further increase verifiability without compromising on the true adversarial accuracy.

Acknowledgments

This work was partially supported by Lockheed Martin Corporation under award number RPP2016-002. We would like to thank Robin Deits, Tobia Marcucci, Peter R. Florence, Yunzhu Li, Joey Huchette, Tally Portnoi and Nalini Singh for several useful conversations. We would also like to thank Eric Wong, Aditi Raghunathan, Jonathan Uesato, Huan Zhang and Tsui-Wei Weng for sharing the networks verified in this paper, and Guy Katz, Nicholas Carlini and Matthias Hein for discussing their results.

References

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[2] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.

[3] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Piecewise linear neural network verification: A comparative study. *arXiv preprint arXiv:1711.00455*, 2017.

[6] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Ground-truth adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[7] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. *arXiv preprint arXiv:1705.07263*, 2017.

[8] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

[9] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*, 2017.

[10] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

[11] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[12] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

[13] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018.

[14] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[15] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *arXiv preprint arXiv:1712.06174*, 2017.

[16] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[18] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[19] Gurobi. Gurobi guidelines for numerical issues, 2017.

[20] Gurobi. Gurobi optimizer reference manual, 2017.

[21] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[24] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[25] Joey Huchette and Juan Pablo Vielma. Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools. *arXiv preprint arXiv:1708.00050*, 2017.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[27] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.

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

[29] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[31] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. SIAM, 2009.

[32] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.

[33] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[35] Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards verification of artificial neural networks. In *MBMV*, pages 30–40, 2015.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[38] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[39] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1):3–57, 2015.

[40] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1058–1066, 2013.

[41] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

[42] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multi-layer neural networks. *arXiv preprint arXiv:1708.03322*, 2017.

A Formulating the MILP model

A.1 Formulating ReLU in an MILP Model

We reproduce our formulation for the ReLU below.

$$y \leq x - l(1-a) \tag{8}$$
$$y \geq x \tag{9}$$
$$y \leq u \cdot a \tag{10}$$
$$y \geq 0 \tag{11}$$
$$a \in \{0, 1\} \tag{12}$$

We consider two cases.

Recall that a is the indicator variable $a = \mathbb{1}_{x \geq 0}$.

When $a = 0$, the constraints in Equation 10 and 11 are binding, and together imply that $y = 0$. The other two constraints are not binding, since Equation 9 is no stricter than Equation 11 when $x < 0$, while Equation 8 is no stricter than Equation 10 since $x - l \geq 0$. We thus have $a = 0 \implies y = 0$.

When $a = 1$, the constraints in Equation 8 and 9 are binding, and together imply that $y = x$. The other two constraints are not binding, since Equation 11 is no stricter than Equation 9 when $x > 0$, while Equation 10 is no stricter than Equation 8 since $x \leq u$. We thus have $a = 1 \implies y = x$.

This formulation for rectified linearities is sharp [39] if we have no further information about x. This is the case since relaxing the integrality constraint on a leads to (x, y) being restricted to an area that is the convex hull of $y = \max(x, 0)$. However, if x is an affine expression $x = w^T z + b$, the formulation is no longer sharp, and we can add more constraints using bounds we have on z to improve the problem formulation.

A.2 Formulating the Maximum Function in an MILP Model

We reproduce our formulation for the maximum function below.

$$y \leq x_i + (1 - a_i)(u_{max,-i} - l_i) \; \forall i \tag{13}$$
$$y \geq x_i \; \forall i \tag{14}$$
$$\sum_{i=1}^{m} a_i = 1 \tag{15}$$
$$a_i \in \{0, 1\} \tag{16}$$

Equation 15 ensures that exactly one of the a_i is 1. It thus suffices to consider the value of a_i for a single variable.

When $a_i = 1$, Equations 13 and 14 are binding, and together imply that $y = x_i$. We thus have $a_i = 1 \implies y = x_i$.

When $a_i = 0$, we simply need to show that the constraints involving x_i are never binding regardless of the values of $x_1, x_2, \ldots, x_m$. Equation 14 is not binding since $a_i = 0$ implies x_i is not the (unique) maximum value. Furthermore, we have chosen the coefficient of a_i such that Equation 13 is not binding, since $x_i + u_{max,-i} - l_i \geq u_{max,-i} \geq y$. This completes our proof.

A.3 Expressing l_p norms as the Objective of an MIP Model

A.3.1 l_1

When $d(x', x) = \|x' - x\|_1$, we introduce the auxiliary variable δ, which bounds the elementwise absolute value from above: $\delta_j \geq x'_j - x_j, \delta_j \geq x_j - x'_j$. The optimization in Equation 3-5 is equivalent to

$$\min_{x'} \sum_j \delta_j \tag{17}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{18}$$

$$x' \in \mathcal{X}_{valid} \tag{19}$$

$$\delta_j \geq x'_j - x_j \tag{20}$$

$$\delta_j \geq x_j - x'_j \tag{21}$$

A.3.2 l_∞

When $d(x', x) = \|x' - x\|_\infty$, we introduce the auxiliary variable ϵ, which bounds the l_∞ norm from above: $\epsilon \geq x'_j - x_j, \epsilon \geq x_j - x'_j$. The optimization in Equation 3-5 is equivalent to

$$\min_{x'} \epsilon \tag{22}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{23}$$

$$x' \in \mathcal{X}_{valid} \tag{24}$$

$$\epsilon \geq x'_j - x_j \tag{25}$$

$$\epsilon \geq x_j - x'_j \tag{26}$$

A.3.3 l_2

When $d(x', x) = \|x' - x\|_2$, the objective becomes quadratic, and we have to use a Mixed Integer Quadratic Program (MIQP) solver. However, no auxiliary variables are required: the optimization in Equation 3-5 is simply equivalent to

$$\min_{x'} \sum_j (x'_j - x_j)^2 \tag{27}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{28}$$

$$x' \in \mathcal{X}_{valid} \tag{29}$$

A.4 Determining Tight Bounds on Decision Variables

Our framework for determining bounds on decision variables is to view the neural network as a computation graph G. Directed edges point from function input to output, and vertices represent variables. Source vertices in G correspond to the input of the network, and sink vertices in G correspond to the output of the network. The computation graph begins with defined bounds on the input variables (based on the input domain ($\mathcal{G}(x) \cap \mathcal{X}_{valid}$)), and is augmented with bounds on intermediate variables as we determine them. The computation graph is acyclic for the feed-forward networks we consider.

Since the networks we consider are piecewise-linear, any subgraph of G can be expressed as an MILP, with constraints derived from 1) input-output relationships along edges and 2) bounds on the values of the source nodes in the subgraph. Integer constraints are added whenever edges describe a non-linear relationship.

We focus on computing an upper bound on some variable v; computing lower bounds follows a similar process. All the information required to determine the best possible bounds on v is contained in the subtree of G rooted at v, G_v. (Other variables that are not ancestors of v in the computation graph cannot affect its value.) Maximizing the value of v in the MILP M_v corresponding to G_v gives the optimal upper bound on v.

We can reduce computation time in two ways. Firstly, we can prune some edges and vertices of G_v. Specifically, we select a set of variables with existing bounds V_I that we assume to be independent (that is, we assume that they each can take on any value independent of the value of the other variables in V_I). We remove all in-edges to vertices in V_I, and eliminate variables without children, resulting in the smaller computation graph G_{v,V_I}. Maximizing the value of v in the MILP M_{v,V_I} corresponding to G_{v,V_I} gives a valid upper bound on v that is optimal if the independence assumption holds.

We can also reduce computation time by relaxing some of the integer constraints in M_v to obtain a MILP with fewer integer variables M'_v. Relaxing an integer constraint corresponds to replacing the relevant non-linear relationship with its convex relaxation. Again, the objective value returned by maximizing the value of v over M'_v may not be the optimal upper bound, but is guaranteed to be a valid bound.

A.4.1 FULL

FULL considers the full subtree G_v and does not relax any integer constraints. The upper and lower bound on v is determined by maximizing and minimizing the value of v in M_v respectively. FULL is also used in [10, 15].

If solves proceed to optimality, FULL is guaranteed to find the optimal bounds on the value of a single variable v. The trade-off is that, for deeper layers, using FULL can be relatively inefficient, since solve times in the worst case are exponential in the number of binary variables in M_v.

Nevertheless, contrary to what is asserted in [10], we *can* terminate solves early and still obtain useful bounds. For example, to determine an upper bound on v, we set the objective of M_v to be to maximize the value of v. As the solve process proceeds, we obtain progressively better certified upper bounds on the maximum value of v. We can thus terminate the solve process and extract the best upper bound found at any time, using this upper bound as a valid (but possibly loose) bound on the value of v.

A.4.2 LINEAR PROGRAMMING (LP)

LP considers the full subtree G_v but relaxes all integer constraints. This results in the optimization problem becoming a linear program that can be solved more efficiently. LP represents a good middle ground between the optimality of FULL and the performance of IA.

A.4.3 INTERVAL ARITHMETIC (IA)

IA selects V_I to be the parents of v. In other words, bounds on v are determined solely by considering the bounds on the variables in the previous layer. We note that this is simply interval arithmetic [31].

Consider the example of computing bounds on the variable $\hat{z}_i = W_i z_{i-1} + b_i$, where $l_{z_{i-1}} \leq z_{i-1} \leq u_{z_{i-1}}$. We have

$$\hat{z}_i \geq W_i^- u_{z_{i-1}} + W_i^+ l_{z_{i-1}} + b_i \tag{30}$$

$$\hat{z}_i \leq W_i^+ u_{z_{i-1}} + W_i^- l_{z_{i-1}} + b_i \tag{31}$$

$$W_i^+ \triangleq \max(W_i, 0) \tag{32}$$

$$W_i^- \triangleq \min(W_i, 0) \tag{33}$$

IA is efficient (since it only involves matrix operations for our applications). However, for deeper layers, using interval arithmetic can lead to overly conservative bounds.

B Additional Experimental Details

B.1 Networks Used

The source of the weights for each of the networks we present results for in the paper are provided below.

- Networks not designed to be robust:
  - MLP-2×[20] and MLP-3×[20] are the MNIST classifiers in [41], and can be found at `https://github.com/huanzhang12/CertifiedReLURobustness`.
- Networks designed for robustness to perturbations with l_∞ norm-bound $\epsilon = 0.1$:
  - LPd-CNN is the MNIST classifier in [28], and can be found at `https://github.com/locuslab/convex_adversarial`.

- Adv-CNN was trained with adversarial examples generated by PGD. PGD attacks were carried out with l_∞ norm-bound $\epsilon = 0.1$, 8 steps per sample, and a step size of 0.334. An l_1 regularization term was added to the objective with a weight of 0.1 on the first convolution layer and 0.2 for the remaining layers.
- Adv-MLP-2$\times$[200] was trained with adversarial examples generated by PGD. PGD attacks were carried out with with l_∞ norm-bound $\epsilon = 0.15$, 200 steps per sample, and a step size of 0.1. An l_1 regularization term was added to the objective with a weight of 0.003 on the first layer and 0 for the remaining layers. Network courtesy of Jonathan Uesato.
- SDPd-MLP-1$\times$[500] is the classifier in [33], provided courtesy of the authors.

• Networks designed for robustness to perturbations with l_∞ norm-bound $\epsilon = 0.2, 0.3, 0.4$:

  - LPd-CNN was trained with the code available at `https://github.com/locuslab/convex_adversarial` at commit `4e9377f`. Parameters selected were `batch_size=20, starting_epsilon=0.01, epochs=200, seed=0`.

B.2 Computational Environment

We construct the MILP models in Julia [3] using JuMP [11], with the model solved by the commercial solver Gurobi 7.5.2 [20]. All experiments were run on a KVM virtual machine with 8 virtual CPUs running on shared hardware, with Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processors, and 8GB of RAM.

C Which Adversarial Examples are Missed by PGD?

PGD succeeds in finding an adversarial example if and only if the starting point for the gradient descent is in the basin of attraction of some adversarial example. Since PGD initializes the gradient descent with a randomly chosen starting point within $\mathcal{G}(x) \cap \mathcal{X}_{valid}$, the success rate (with a single random start) corresponds to the fraction of $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ that is in the basin of attraction of some adversarial example.

Intuitively, the success rate of PGD should be inversely related to the magnitude of the minimum adversarial distortion $\hat{\delta}$: if $\hat{\delta}$ is small, we expect more of $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ to correspond to adversarial examples, and thus the union of the basins of attraction of the adversarial examples is likely to be larger. We investigate here whether our intuition is substantiated.

To obtain the best possible empirical estimate of the success rate of PGD for each sample, we would need to re-run PGD initialized with *multiple* different randomly chosen starting points within $\mathcal{G}(x) \cap \mathcal{X}_{valid}$.

However, since we are simply interested in the relationship between success rate and minimum adversarial distortion, we obtained a coarser estimate by binning the samples based on their minimum adversarial distortion, and then calculating the fraction of samples in each bin for which PGD with a *single* randomly chosen starting point succeeds at finding an adversarial example.

Figure 3 plots this relationship for four networks using the CNN architecture and trained with the same training method LPd but optimized for attacks of different size. Three features are clearly discernible:

- PGD is very successful at finding adversarial examples when the magnitude of the minimum adversarial distortion, $\hat{\delta}$, is small.
- The success rate of PGD declines significantly for all networks as $\hat{\delta}$ approaches ϵ.
- For a given value of $\hat{\delta}$, and two networks a and b trained to be robust to attacks with l_∞ norm-bound ϵ_a and ϵ_b respectively (where $\epsilon_a < \epsilon_b$), PGD is consistently more successful at attacking the network trained to be robust to smaller attacks, a, as long as $\hat{\delta} \ll \epsilon_a$.

The sharp decline in the success rate of PGD as $\hat{\delta}$ approaches ϵ is particularly interesting, especially since it is suggests a pathway to generating networks that *appear* robust when subject to PGD attacks of bounded l_∞ norm but are in fact vulnerable to such bounded attacks: we simply train the network to maximize the total number of adversarial examples with minimum adversarial distortion close to ϵ.

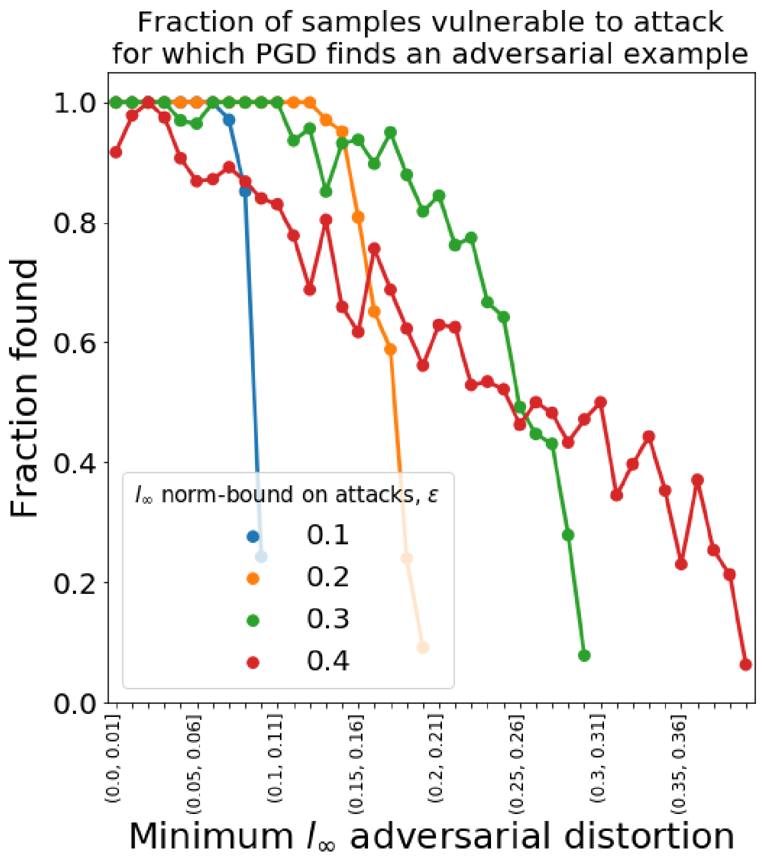


Figure 3: Fraction of samples in the MNIST test set vulnerable to attack for which PGD succeeds at finding an adversarial example. Samples are binned by their minimum adversarial distortion (as measured under the l_∞ norm), with bins of size 0.01. Each of these are LPd-CNN networks, and were trained to optimize for robustness to attacks with l_∞ norm-bound ϵ. For any given network, the success rate of PGD declines as the minimum adversarial distortion increases. Comparing networks, success rate declines for networks with larger ϵ even at the same minimum adversarial distortion.

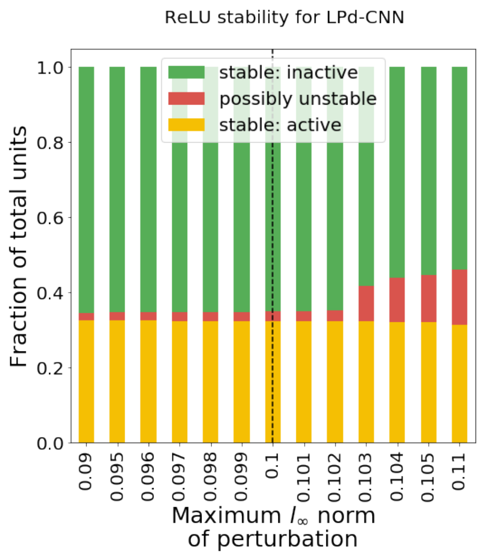
D Robust Training and ReLU stability

Networks that are designed to be robust need to balance two competing objectives. Locally, they need to be robust to small perturbations to the input. However, they also need to retain sufficient global expressiveness to maintain a low test error.

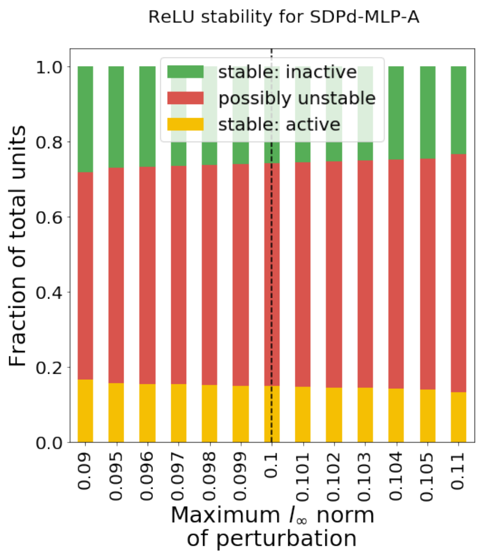
For the networks in Table 2, even though each robust training approach estimates the worst-case error very differently, all approaches lead to a significant fraction of the ReLUs in the network being provably stable with respect to perturbations with bounded l_∞ norm. In other words, for the input domain $\mathcal{G}(x)$ consisting of all bounded perturbations of the sample x, we can show for many ReLUs that the input to the unit is always positive (and thus the output is linear in the input) or always negative (and thus the output is always zero). As discussed in the main text, we believe that the need for the network to be robust to perturbations in $\mathcal{G}$ drives more ReLUs to be provably stable with respect to $\mathcal{G}$.

To better understand how networks can retain global expressiveness even as many ReLUs are provably stable with respect to perturbations with bounded l_∞ norm ϵ, we study how the number of ReLUs that are provably stable changes as we vary the size of $\mathcal{G}(x)$ by changing the maximum allowable l_∞ norm of perturbations. The results are presented in Figure 4.

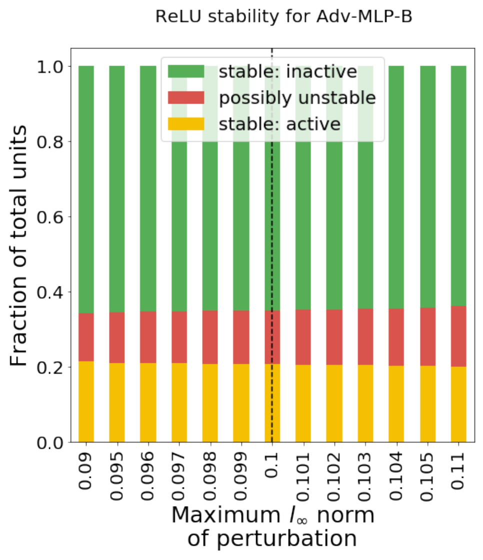
As expected, the number of ReLUs that cannot be proven to be stable increases as the maximum allowable l_∞ norm of perturbations increases. More interestingly, LPd-CNN is very sensitive to the $\epsilon = 0.1$ threshold, with a sharp increase in the number of ReLUs that cannot be proven to be stable when the maximum allowable l_∞ norm of perturbations increases beyond 0.102. An increase of the same abruptness is not seen for the other two networks.



(a) LPd-CNN. Note the sharp increase in the number of ReLUs that cannot be proven to be stable when the maximum l_∞ norm increases beyond 0.102.



(b) SDPd-MLP-A.



(c) Adv-MLP-B. Adversarial training alone is sufficient to significantly increase the number of ReLUs that are provably stable.

Figure 4: Comparison of provably ReLU stability for networks trained via different robust training procedures to be robust at $\epsilon = 0.1$, when varying the maximum allowable l_∞ norm of the perturbation. The results reported in Table 2 are marked by a dotted line. As we increase the maximum allowable l_∞ norm of perturbations, the number of ReLUs that cannot be proven to be stable increases across all networks (as expected), but LPd-CNN is far more sensitive to the $\epsilon = 0.1$ threshold.

E Sparsification and Verifiability

When verifying the robustness of SDPd-MLP-A, we observed that a significant proportion of kernel weights were close to zero. Many of these tiny weights are unlikely to be contributing significantly to the final classification of any input image. Having said that, setting these tiny weights to zero *could* potentially reduce verification time, by 1) reducing the size of the MILP formulation, and by 2) ameliorating numerical issues caused by the large range of numerical coefficients in the network [19].

We generated sparse versions of the original network to study the impact of sparseness on solve times. Our heuristic sparsification algorithm is as follows: for each fully-connected layer i, we set a fraction f_i of the weights with smallest absolute value in the kernel to 0, and rescale the rest of the weights such that the l_1 norm of the kernel remains the same.[7] Note that MLP-A consists of only two layers: one hidden layer (layer 1) and one output layer (layer 2).

Table 4: Effect of sparsification of SDPd-MLP-A on verifiability. Test error increases slightly as larger fractions of kernel weights are set to zero, but the certified upper bound on adversarial error decreases significantly as the solver reaches the time limit for fewer samples.

Fraction zeroed		Test Error	Certified Bounds on Adversarial Error		Mean Time / s	Fraction Timed Out
f_1	f_2		Lower Bound	Upper Bound		
0.0	0.00	**4.18%**	14.36%	30.81%	127.9	0.1645
0.5	0.25	4.22%	14.60%	25.25%	70.3	0.1065
0.8	0.25	4.22%	15.03%	**18.26%**	30.2	0.0323
0.9	0.25	4.93%	17.97%	18.76%	12.4	0.0079

Table 4 summarizes the results of verifying sparse versions of SDPd-MLP-A; the first row presents results for the original network, and the subsequent rows present results when more and more of the kernel weights are set to zero.

When comparing the first and last rows, we observe an improvement in both mean time and fraction timed out by an order of magnitude. As expected, sparsifying weights increases the test error, but the impact is not significant until f_1 exceeds 0.8. We also find that sparsification significantly improves our upper bound on adversarial error — to a point: the upper bound on adversarial error for $f_1 = 0.9$ is higher than that for $f_1 = 0.8$, likely because the true adversarial error has increased significantly.

Starting with a network that is robust, we have demonstrated that a simple sparsification approach can already generate a sparsified network with an upper bound on adversarial error significantly lower than the best upper bound that can be determined for the original network. Adopting a more principled sparsification approach could achieve the same improvement in verifiability but without compromising on the true adversarial error as much.

[7] Skipping the rescaling step did not appreciably affect verification times or test errors.