Evaluating Robustness of Neural Networks with Mixed Integer Programming

by

Vincent Tjeng

B.S., Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2018

Certified by .....
Russ Tedrake
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Evaluating Robustness of Neural Networks with Mixed Integer Programming

by

Vincent Tjeng

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2018, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Neural networks have demonstrated considerable success on a wide variety of real-world problems. However, neural networks can 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. Our verifier finds minimum adversarial distortions two to three orders of magnitude more quickly than the state-of-the-art. We achieve this via tight formulations for non-linearities, as well as a novel presolve algorithm that makes full use of all information available. The computational speedup enables us to verify properties on convolutional networks with an order of magnitude more ReLUs than had been previously verified by any complete verifier, and we determine for the first time the *exact* adversarial accuracy of an MNIST classifier to perturbations with bounded l_∞ norm $\epsilon = 0.1$. On this network, we find an adversarial example for 4.38% of samples, and a certificate of robustness for the remainder. Across a variety of robust training procedures, we are able to certify more samples than the state-of-the-art *and* find more adversarial examples than a strong first-order attack for every network.

Thesis Supervisor: Russ Tedrake
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

None of this work would have been possible without the unceasing support of my advisor, Russ Tedrake. His knack for asking the right questions and discerning where the largest potential impact would be has ensured that I remain focused on what is most important.

I'd also like to thank the members of the Robot Locomotion Group for their insights and friendship. I spend every day at work surrounded by an incredible group of brilliant people, and I am continuously inspired by the work they do. Best of all, everyone in the group is incredibly generous with their time.

Finally, I would like to thank my family for a lifetime of love and support. Thanks to my mom and dad for their constant encouragement, and my sister for always being there to lend an ear.

Funding Acknowledgement This work was partially supported by Lockheed Martin Corporation under award number RPP2016-002.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural networks represent the state-of-the-art for classification of images [18, 39] and object localization in images [33]. 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 [36]. In the context of image classification, the perturbed image is often visually indistinguishable from the original but can be misclassified with high confidence.

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 the *minimum adversarial distortion* for each input [6]. An increase in mean minimum distortion indicates an improvement in robustness. Alternatively, we can determine *adversarial test accuracy* [2] with respect to a bounded

class of attacks. A classifier is considered robust for an input only if we can certify that no bounded perturbation causes a misclassification.[1]

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 [26].

1.1 Contributions

We make the following key contributions:

- We demonstrate that, despite considering the full combinatorial nature of the network, our verifier *can* succeed on deeper neural networks — including those with convolutional layers — when evaluating the robustness of these networks to bounded perturbations. Furthermore, we identify *why* we can succeed. Despite the large total number of non-linear units, as long as the input domain is bounded, 1) a large fraction of the non-linear units are provably active or provably inactive; 2) many labels can be efficiently eliminated from consideration.

- We determine — for the first time — the exact adversarial accuracy for MNIST classifiers with bounded l_∞ norm ϵ. For example, for $\epsilon = 0.1$, we *guarantee* an adversarial test accuracy for the classifier of 95.62%, and provide a valid adversarial example for the remaining test inputs. This accuracy represents an increase of 1.44 percentage points over the highest previous guarantee found in [27].

- We are able to certify more samples than the state-of-the-art *and* find more

[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.

adversarial examples than a strong attack across different networks trained with a variety of robust training procedures.

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

1.2 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, we consider the example of evaluating adversarial accuracy. As in [27], 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 the answer to some queries about the adversarial polytope may not be decidable using incomplete verifiers. In particular, incomplete verifiers can only certify robustness for a fraction of input that is robust; 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 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, 32], layer-by-layer approximations of the adversarial polytope [41], discretizing the search space [23], abstract interpretation [16], bounding the local Lipschitz constant [40], or bounding the activation of the ReLU with linear functions [40]. We highlight in particular the verifier presented by [27], 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, 28] or SMT [6, 14, 26, 34]. 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 revealed through our ablation testing. When compared to SMT-based approaches, our verifier is two to three orders of magnitude faster than the state-of-the-art verifier, Reluplex, on the task of finding minimum adversarial distortions.

Other authors have used complete verifiers to verify properties of MNIST classifiers, determining minimum adversarial distortions [6] or verifying robustness to bounded perturbations [10, 15]. However, the largest MNIST classifiers 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 complete verification is in training networks that are *designed* to be robust to bounded perturbations. Rather than simply optimizing over the loss at each input, robust training procedures optimize over an estimate of the worst-case loss over all bounded perturbations of the input.

Adversarial training [17] optimizes over a *lower bound* of the loss provided by a valid 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 [37].

In contrast, certified training approaches [22, 27, 32] optimize over an *upper bound* of the loss (or a surrogate for the upper bound). At the end of the training procedure, the upper bound at a given input (if it is below a threshold) can be used as a certificate for 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.

Chapter 2

Verification as solving an MILP

2.1 Preliminaries

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.

Borrowing from the notation in [5], the general problem of verification is to determine whether some property $P(\cdot) : \mathbb{R}^n \to \{F, T\}$ 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, the property 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 densely-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 non-linearities in Section 2.5. [5] observes that batch normalization [25] or dropout [35] are also linear transformations at *evaluation time*, and we note that the "shortcut connections" used in architectures such as ResNet [21] are also linear.

Finally, we note that the 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, $\mathcal{X}_{valid} = [0, 1]^m$.

2.2 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 [27, 29], 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* in $\mathcal{G}(x)$:

$$\forall x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid}) : \ \text{argmax}_i(f_i(x')) = \lambda(x) \tag{2.1}$$

Equivalently, the network is robust to perturbations on x if and only if Equation 2.2 is infeasible.

$$(x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid})) \wedge \left(f_{\lambda(x)}(x') < \max_{\mu \in [1,n] \setminus \{\lambda(x)\}} f_\mu(x') \right) \tag{2.2}$$

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).

As long as $\mathcal{G}(x)$ (and $\mathcal{X}_{valid}$) can be expressed as the union of a set of polyhedra, this feasibility problem can be expressed as an MILP. In our case, the three robust training procedures we consider [27, 29, 32] are all designed to be robust to perturbations with bounded l_∞ norm, and the l_∞-ball of radius ϵ around each input x can indeed be succinctly represented by the set of linear constraints $\mathcal{G}(x) = \{x' \mid \forall i : -\epsilon \leq (x - x')_i \leq \epsilon\}$.

2.3 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:

$$\min_{x'} d(x', x) \tag{2.3}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{2.4}$$

$$x' \in \mathcal{X}_{valid} \tag{2.5}$$

More generally, we can also find the minimum *targeted* adversarial distortion, where we attempt to generate an adversarial examples that is classified in one of a set of target labels T.[1] The minimum targeted adversarial distortion is the solution to the optimization:

$$\min_{x'} d(x', x) \tag{2.6}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \in T \tag{2.7}$$

$$x' \in \mathcal{X}_{valid} \tag{2.8}$$

The most prevalent distance metrics in the literature for generating adversarial examples are the l_1 [8, 9], l_2 [36], and l_∞ [17, 31] 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 Section 2.4.

[1] We note that the minimum adversarial distortion is simply a special case of the minimum targeted adversarial distortion, with $T = [1, n] \backslash \{\lambda(x)\}$.

2.4 Expressing l_p Norms as the objective of an MIP Model

2.4.1 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 2.3-2.5 is equivalent to

$$\min_{x'} \sum_j \delta_j \tag{2.9}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{2.10}$$

$$x' \in \mathcal{X}_{valid} \tag{2.11}$$

$$\delta_j \geq x'_j - x_j \tag{2.12}$$

$$\delta_j \geq x_j - x'_j \tag{2.13}$$

2.4.2 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 2.3-2.5 is equivalent to

$$\min_{x'} \epsilon \tag{2.14}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{2.15}$$

$$x' \in \mathcal{X}_{valid} \tag{2.16}$$

$$\epsilon \geq x'_j - x_j \tag{2.17}$$

$$\epsilon \geq x_j - x'_j \tag{2.18}$$

2.4.3 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 2.3-2.5 is simply equivalent to

$$\min_{x'} \sum_j (x'_j - x_j)^2 \tag{2.19}$$

$$\text{subject to} \quad \text{argmax}_i(f_i(x')) \neq \lambda(x) \tag{2.20}$$

$$x' \in \mathcal{X}_{valid} \tag{2.21}$$

2.5 Formulating piecewise-linear Functions

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, tight formulations for general piecewise linear functions are available in [24].

2.5.1 Formulating ReLU

Consider a ReLU with input x and output y. We have $y = \max(x, 0)$. In addition, we assume that we have some bounds on the value of the input, $l \leq x \leq u$.

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 $a = \mathbb{1}_{x \geq 0}$. The constraint $y = \max(x, 0)$ is then equivalent to the following set of linear and integer constraints:

$$y \leq x - l(1-a) \tag{2.22}$$

$$y \geq x \tag{2.23}$$

$$y \leq u \cdot a \tag{2.24}$$

$$y \geq 0 \tag{2.25}$$

$$a \in \{0, 1\} \tag{2.26}$$

The proof is as follows.

When $a = 0$, the constraints in Equation 2.24 and 2.25 are binding, and together imply that $y = 0$. The other two constraints are not binding, since Equation 2.23 is no stricter than Equation 2.25 when $x < 0$, while Equation 2.22 is no stricter than Equation 2.24 since $x - l \geq 0$. We thus have $a = 0 \implies y = 0$.

When $a = 1$, the constraints in Equation 2.22 and 2.23 are binding, and together imply that $y = x$. The other two constraints are not binding, since Equation 2.25 is no stricter than Equation 2.23 when $x > 0$, while Equation 2.24 is no stricter than Equation 2.22 since $x \leq u$. We thus have $a = 1 \implies y = 0$.

This formulation for rectified linearities is sharp [38] 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 our bounds on each individual element of z to improve the problem formulation.

2.5.2 Formulating the Maximum Function.

Consider a unit whose output y is the maximum of its inputs $x_1, x_2, \ldots, x_m$. We have $y = \max(x_1, x_2, \ldots, x_m)$. As in the previous section, we assume that we have some bounds on the value of the input, $l_i \leq x_i \leq u_i$.

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 henceforth assume without loss of generality that $u_i > l_{max}\ \forall i$. 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 $u_{max,-i} \triangleq \max_{j \neq i}(u_j)$. The constraint $y = \max(x_1, x_2, \ldots, x_m)$ is then equivalent to the following set of linear and integer constraints:

$$y \leq x_i + (1 - a_i)(u_{max,-i} - l_i)\ \forall i \tag{2.27}$$

$$y \geq x_i\ \forall i \tag{2.28}$$

$$\sum_{i=1}^{m} a_i = 1 \tag{2.29}$$

$$a_i \in \{0, 1\} \tag{2.30}$$

The proof is as follows.

Equation 2.29 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 2.27 and 2.28 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 2.28 is not binding since $a_i = 0$ implies x_i is not the (unique) maximum value. Furthermore, we have chosen an appropriate constant term such that Equation 2.27 is not binding, since $x_i + u_{max,-i} - l_i \geq u_{max,-i} \geq y$. This completes our proof.

2.6 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 [38]. For instance, if we can

prove that the phase of a ReLU is stable, we can avoid introducing a binary variable. More generally, looser bounds on input to some unit will propagate downstream, leading to units in later layers having looser bounds.

2.6.1 Procedures to Determine Bounds

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 (as determined by 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.

In our descriptions, we focus on computing an upper bound on v; computing lower bounds follows a similar process. All the information required to determine the best possible bounds on a variable 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, they each can take on any value — within the existing bounds — 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 matching 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 will still be valid.

MILP

MILP 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. This procedure is also used in [10, 15].

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

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 the MILP 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.

LP

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

IA

IA selects V_I to be the parents of v; this is simply interval arithmetic [30]. In other words, bounds on v are determined solely by considering the bounds on the variables in the previous layer. This procedure is also used in [10, 12, 27].

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

$$\hat{z}_i \geq W_i^- u_{z_{i-1}} + W_i^+ l_{z_{i-1}} \tag{2.31}$$

$$\hat{z}_i \leq W_i^+ u_{z_{i-1}} + W_i^- l_{z_{i-1}} \tag{2.32}$$

$$W_i^+ \triangleq \max(W_i, 0) \tag{2.33}$$

$$W_i^- \triangleq \min(W_i, 0) \tag{2.34}$$

$$l_{z_{i-1}} \leq z_{i-1} \leq u_{z_{i-1}} \tag{2.35}$$

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.

2.6.2 Progressive Bounds Tightening

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 2.2 or Equation 2.3-2.5).

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 inactive, 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. 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. Finally, we also allow users to specify a *maximum build effort*, which is 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 such as LP provides sufficiently strong bounds for quick solves.

Chapter 3

Experimental Results

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×[500] and MLP-2×[200] as **MLP-A** and **MLP-B** respectively. **CNN** refers to the ConvNet architecture used for the robust MNIST classifier in [27]. The network has two convolutional layers (stride length 2) with 16 and 32 filters respectively (size 4×4 in both layers), followed by a hidden 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**[1] when the dual of a linear program is used, as in [27]; **SDPd** when the dual of a semidefinite relaxation is used, as in [32]; and **Adv** when adversarial examples generated via Projected Gradient Descent (PGD) are used, as in [29]. Full details on all networks used are provided in Appendix 3.

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

Experimental Setup. We construct the MILP models in Julia [3] using JuMP [11], with the model solved by the commercial solver Gurobi 7.5.2 [19]. All experiments were run on a KVM virtual machine with modest specifications: 8 virtual CPUs running on shared hardware, with Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processors, and 8GB of RAM. We set the maximum build effort to LP. Unless otherwise noted, we terminate the optimization and report a timeout if solve time exceeds 1200s.

Additional Details on 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 [40], 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 [27], 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×[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.
  - SDPd-MLP-1×[500] is the classifier in [32], 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`. Parameters selected were `batch_size=20, starting_epsilon=0.01, epochs=200, seed=0`.

3.1 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` [26], a complete verifier also able to find the true minimum distortion; and 2) `LP`[2], `Fast-Lip`, `Fast-Lin` [40], and `LP-full` [27], 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 proof 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 [40], we did our best to match the authors' experimental setup. Specifically, we used the same network weights, test samples, and target classes as in [40]. In addition, we conducted the experiments in single thread mode as the authors do. Finally, as in [40], the results presented represent an average over the first 100 samples of the MNIST test set, with samples among the first 100 that are incorrectly classified excluded from the average.

3.1.1 Verification Times

Figure 3-1 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, and our method appears to scale better for deeper networks. For the l_1 norm, our speed is even comparable to `LP-full` — a method which only provides a lower bound.

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

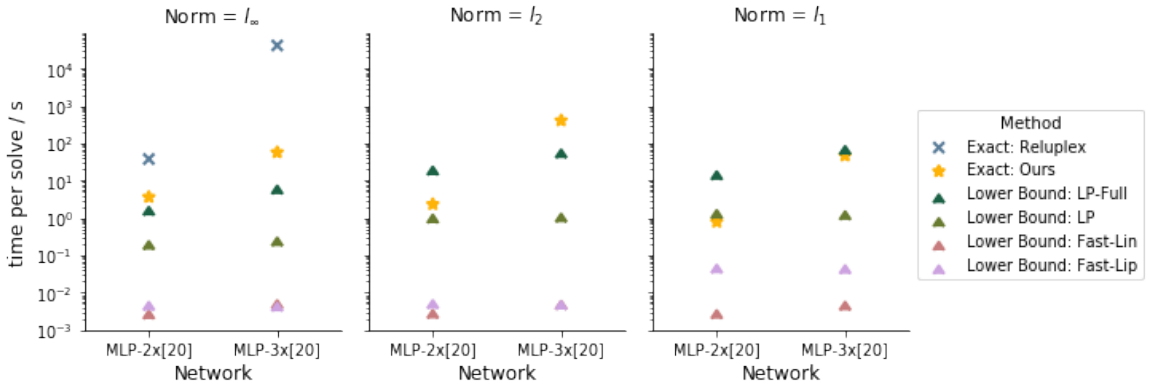

Figure 3-1: Average times for determining exact values or bounds on the minimum targeted adversarial distortion. We improve on the speed of the state-of-the-art complete verifier `Reluplex` by two to three orders of magnitude. Results for methods other than ours are from [40]; results for `Reluplex` were only available in [40] for the l_∞ norm.

3.1.2 Bounds on Minimum Targeted Adversarial Distortions

Figure 3-2 compares lower bounds provided by the incomplete verifiers to the exact value we obtain.

Even on the small networks we are considering, the gap between the best certified lower bound and the true minimum adversarial distortion is significant. This corroborates the observation in [32] 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; thus, a network designed to be robust to perturbations with l_∞ norm-bound $\epsilon = 0.2$ might only be verifiable to $\epsilon = 0.1$.

3.2 Determining Adversarial Accuracy of Robust Networks

Our complete verifier enables us to determine the exact adversarial accuracy of any feed-forward piecewise-linear classifier given enough time. We attempt to do so on classifiers trained by a range of robust training procedures. Table 3.1 presents the test

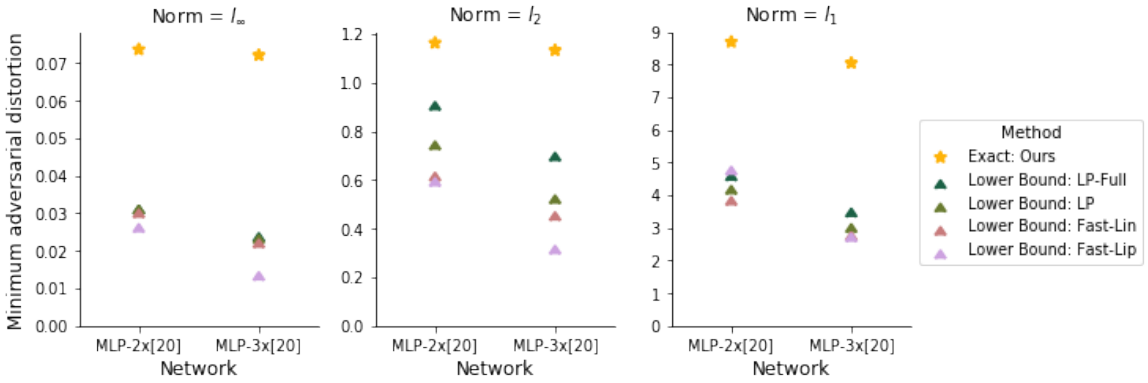

Figure 3-2: Average of exact values or bounds on the minimum targeted adversarial distortion. The gap between the true minimum adversarial distortion and the best lower bound is significant and increases for deeper networks. Again, results for methods other than ours are from [40].

error of these classifiers, along with estimates of the adversarial error.

Lower Bounds. 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. 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 our performance depends on the training method and architecture, we improve on both the upper and lower bounds for *every* network tested.[3] Notably, we improve on the upper 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). Our method also finds an adversarial example for every input that PGD finds one for. 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

[3]On SDPd-MLP-A, the verifier in [32] finds certificates for 372 samples for which our verifier reaches its time limit.

Table 3.1: Adversarial accuracy of classifiers to perturbations with l_∞ norm-bound ϵ. In every case, we improve 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 in the cited paper. 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 samples for which the solver reached the time limit. 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 [32] since we impose the constraint that $x' \in \mathcal{X}_{valid}$.

Network	ϵ	Test Error	Certified Bounds on Adversarial Error				
			Lower Bound		Upper Bound		No Gap?
			PGD	Ours	SOA	Ours	
LPd-CNN	0.1	1.89%	4.11%	**4.38%**	5.82%[27]	**4.38%**	✓
Adv-CNN	0.1	0.96%	4.10%	**4.21%**	—	**7.21%**	
Adv-MLP-B	0.1	4.02%	9.03%	**9.68%**	15.41%[13]	**9.68%**	✓
SDPd-MLP-A	0.1	4.18%	†11.51%	**14.36%**	34.77%[32]	**30.81%**	
LPd-CNN	0.2	4.23%	9.54%	**10.68%**	17.50%[27]	**10.68%**	✓
LPd-CNN	0.3	11.40%	22.70%	**25.79%**	35.03%[27]	**25.79%**	✓
LPd-CNN	0.4	26.13%	39.22%	**48.98%**	62.49%[27]	**48.98%**	✓

or an adversarial example for *every* test sample. As seen in Table 3.2, running our verifier for LPd-CNN and Adv-MLP-Bover 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.

Observations on Determinants of Verification Time

Ceteris paribus, we might expect verification time to be proportional to the total number of ReLUs, since the solver may need to explore both possibilites for the phase of each ReLU. However, as can be seen in Table 3.2, 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 restricted input domain partially explains this discrepancy. With a restricted input domain, we can eliminate some labels from consideration by proving that the upper bound on the corresponding output neuron is lower than the lower bound for some other output neuron. We can

Table 3.2: Mean verification time for determining adversarial accuracy of classifiers over the full MNIST test set of 10,000 samples, includes solves reaching the time limit but excluding samples that are incorrectly classified (since determining that these samples are not robust is trivial). The fraction of samples reaching the time limit is noted as 'fraction timed out'.

Network	ϵ	Mean Time / s	Fraction timed out
LPd-CNN	0.1	3.52	0
Adv-CNN	0.1	135.74	0.0300
Adv-MLP-B	0.1	3.69	0
SDPd-MLP-A	0.1	312.43	0.1645
LPd-CNN	0.2	7.32	0
LPd-CNN	0.3	5.13	0
LPd-CNN	0.4	5.07	0

also determine that some ReLUs are stable by inspecting bounds on their input. For details, see Section 2.5.1. As the results in Table 3.3 show, for robust networks, a significant number of labels can be eliminated from consideration, and a significant number of ReLUs can be proven to be stable. Runtime is lower when more labels can be eliminated from consideration and more ReLUs can be proven to be stable.

Interestingly, networks not trained to be robust do not exhibit the same behavior — very few ReLUs can be proven to be stable even with a restricted input domain. We thus posit that, for robust classifiers, the small number of unstable ReLUs around test input is an indication that the classifier has been trained to be less expressive in the local neighborhood of points on the underlying data manifold, even as the classifier retains global expressiveness.

3.3 Ablation Testing

We isolated the impact of each optimization in our implementation by conducting thorough ablation tests. Results are reported in Table 3.4. When removing *progressive tightening*, we skip less computationally complex procedures, directly using the procedure specified by the maximum build effort (LP in this case). When removing

Table 3.3: Number of labels that can be eliminated from consideration, and average number of ReLUs in each phase when the input domain is restricted to the l_∞ ball of radius $\epsilon = 0.1$ around test input. Results are aggregated over the full MNIST test set. Mean verification time is inversely proportional to number of labels eliminated and proportional to number of possibly unstable ReLUs.

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-MLP-B	3.69	4.77	55.21	87.31	257.48	400
Adv-CNN	135.74	3.14	545.90	3383.30	874.80	4804
SDPd-MLP-A	312.43	0.00	297.66	73.85	128.50	500

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 Equations 2.22 and 2.24 with $-M$ and M respectively, where $M \triangleq \max(-l, u)$, as is done in [10, 12, 28].

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. In addition, higher build times when removing *using asymmetric bounds* demonstrate the vicious cycle of loose initial bounds causing later bounds to take more time to compute. These ablation tests emphasize how critical it is to make full use of the information available on bounds when building the MILP model.

Table 3.4: 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 feasibility problem in Equation 2.2.

[†]For Some bounds are reusable across different input since input domain is no longer restricted; we exclude the initial build time required (3593s) for these bounds. [‡]Ablation test 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

Chapter 4

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 perturbation region is piecewise-linear in x.

While our work has focused on robustness evaluation of neural networks where the training process is treated as a black box, we do have some thoughts on improving the verifiability of neural networks by augmenting training.

Firstly, as discussed in Section 3.2, increasing the number of ReLUs that can be proven to be locally stable makes verification quicker. One possible approach to improving verifiability is to reduce the number of ReLUs that are unstable during training. For example, we could design a differentiable estimator for the number of unstable ReLUs, adding a penalty for the estimated number of unstable ReLUs to the

loss function.

We also observed that sparsifying weights promotes verifiability: even naïvely sparsifying SDPd-MLP-A (by setting a fraction of weights with the smallest absolute value to 0) 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 [20]) could potentially further increase verifiability without compromising on the true adversarial accuracy.

In closing, we believe that, when the stakes are sufficiently high — for example, if the neural network is used within the control loop of an autonomous vehicle — practitioners should not simply be satisfied with a network that has high test accuracy on the unperturbed test set, but for which the adversarial accuracy is unknown. Instead, what is needed is a neural network that has high *provable* adversarial accuracy over a sufficiently rich set of allowable perturbations $\mathcal{G}(x)$.[1] We hope that our work will form one part of the toolkit used to generate neural networks that are robust to perturbations and by *proving* that the networks generated are indeed robust.

[1]We consider $\mathcal{G}(x)$ to be richer if it captures more images that humans would consider to be perceptually similar to the original image x.

Bibliography

[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.

[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] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2017.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

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

[26] 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.

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

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

[29] 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.

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

[31] 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.

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

[33] 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.

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

[35] 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.

[36] 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.

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

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

[39] 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.

[40] 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.

[41] 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.