Maximal Robust Neural Network Specifications via Oracle-guided Numerical Optimization

Anan Kabaha[1][0000−0002−0969−6169] and Dana Drachsler-Cohen[1][0000−0001−6644−5377]

Technion, Haifa, Israel {anan.kabaha@campus,ddana@ee}.technion.ac.il

Abstract. Analyzing the robustness of neural networks is crucial for trusting them. The vast majority of existing works focus on networks' robustness in ϵ-ball neighborhoods, but these cannot capture complex robustness specifications. We propose MaRVeL, a system for computing maximal non-uniform robust specifications that maximize a target norm. The main idea is to employ *oracle-guided numerical optimization*, thereby leveraging the efficiency of a numerical optimizer as well as the accuracy of a non-differentiable robustness verifier, acting as the oracle. The optimizer iteratively submits to the verifier candidate specifications, which in turn returns the closest inputs to the decision boundaries. The optimizer then computes their gradients to guide its search in the directions the specification can expand while remaining robust. We evaluate MaRVeL on several datasets and classifiers and show that its specifications are larger by 5.1x than prior works. On a two-dimensional dataset, we show that the average diameter of its specifications is 93% of the optimal average diameter, whereas the diameter of prior works' specifications is only 26%.

1 Introduction

Neural networks are susceptible to adversarial examples [14,21,48,37,46,15]. To understand the robustness level of neural networks, many works verify local robustness [43,28,34,18,3,31,41,38]. These works focus on analyzing the network's robustness at an ϵ-ball centered at a given input, where every input entry can be perturbed by up to $\pm\epsilon$. However, focusing only on this kind of neighborhood hinders the overall robustness level of the network. To illustrate, consider Figure 1 showing the decision boundaries of a small network (the black curves), taking two-dimensional inputs, and an input (the red dot). Its maximal ϵ-ball is bounded by the closest decision boundary and thus it is quite small (the blue square). This is because an ϵ-ball uniformly bounds all perturbations by the same ϵ.

This gave rise to works that compute maximal non-uniform robust neighborhoods [26,25]. These neighborhoods are defined by interval specifications, generalizing ϵ-balls, where each input entry is bounded by an interval. A robust interval specification is maximal if expanding any interval results in including an adversarial example. In other words, every interval is approaching a decision boundary. To pick among the multiple maximal robust specifications, it is common to maximize a given size metric (e.g., the L_1 or L_2 norm). Computing

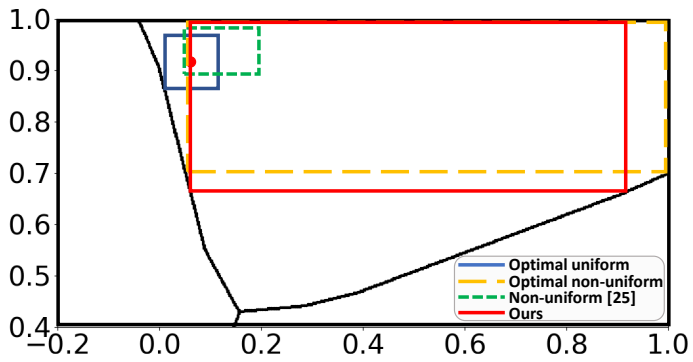

Fig. 1: A comparison of MaRVeL's maximal non-uniform specification to the maximal (uniform) ϵ-ball, the optimal non-uniform specification (computed by a naive approach), and the maximal non-uniform specification computed by [25].

maximal non-uniform specifications is challenging because (1) the search space is exponentially large and (2) determining whether an interval specification belongs to this space, i.e., whether it is robust, requires to call a robustness verifier, which takes non-negligible time. A naive optimal approach begins by computing all decision boundaries around the given input using a grid search. Accordingly, it computes all maximal robust interval specifications and returns the specification maximizing the size metric. However, this approach is highly time-consuming and impractical if the input dimension is high. Figure 1 shows an optimal non-uniform specification (the dashed yellow rectangle).

Existing works propose efficient approaches to compute maximal non-uniform specifications [26,25]. These approaches rely on numerical optimization, to search in the large space, and on an incomplete robustness analysis, to determine robustness of candidate specifications. This analysis overapproximates the network's computation with differentiable linear functions and thus it scales well to large networks and is amenable to first-order optimization. However, incomplete analysis suffers from precision loss. Hence, their specifications are not always maximal and are quite small. Figure 1 shows the maximal non-uniform specification computed by [25] (the dashed green rectangle). It is significantly smaller than the optimal specification and it does not reach any decision boundary. This raises the question: *Can we efficiently compute optimal maximal robust specifications?*

We present MaRVeL (**Ma**ximal **R**obustness **Ve**rification of Interva**L** specifications). Like prior works, MaRVeL relies on a numerical optimizer to look for a robust specification maximizing a given size metric. Unlike prior works, it relies on a MILP robustness verifier [38], which provides a more accurate analysis but is not differentiable. To employ first-order optimization, we propose a novel way to compute the analysis' gradient from the set of *weakest points*. These are inputs contained in the specification that are the closest to the decision boundaries, and they are computed during the robustness analysis. Based on this idea, MaRVeL

employs *oracle-guided numerical optimization*. At each iteration, it submits a specification to the robustness MILP verifier to obtain the weakest points. Accordingly, it computes the gradient and constructs the next specification. MaRVeL also employs counterexample-guided synthesis (CEGIS) to prune the search space based on non-robust specifications. If the verifier determines a specification is non-robust (i.e., it contains an adversarial example), then MaRVeL prunes the search space by restricting the relevant interval bounds. Figure 1 shows the specification computed by MaRVeL (the red rectangle), which is maximal and its average diameter is only 7% smaller than that of the optimal specification.

We evaluate MaRVeL on several benchmarks and compare to prior works [26,25]. First, we consider the two-dimensional synthetic dataset of [26] and show that MaRVeL's specifications are maximal and their average diameters is 93% of the average diameters of the optimal specifications, whereas the average diameters of prior works' specifications are at most 26%. Second, we consider popular datasets (MNIST, Fashion-MNIST, CIFAR-10, and Contagio/Virustotal [7,40]) and several networks, including convolutional networks. Results show that the average diameter of MaRVeL's specifications is 5.1x larger than that of prior works' specifications. We further show that the CEGIS component leads to 1.8x larger average diameters. The execution time of MaRVeL is 19.9x longer than that of prior works and 6.5x longer if MaRVeL terminates upon the first non-robust specification. The longer execution time is mostly because MaRVeL relies on a more accurate verifier. Lastly, we show that MaRVeL's specifications identify robustness attributes of the networks that ϵ-balls cannot identify and even prior works' specifications do not identify.

2 Preliminaries

In this section, we provide background on network classifiers and local robustness.

Neural network classifiers Given an input domain $\mathbb{R}^d$ and a set of classes $C = \{1, \ldots, c\}$, a classifier maps inputs to a score vector over the possible classes $D : \mathbb{R}^d \to \mathbb{R}^c$. We focus on classifiers in which every input entry has a minimum and maximum domain value. A fully-connected network consists of L layers. The first layer z_0 takes as input a vector from $\mathbb{R}^d$, denoted x, and it passes the input as is to the next layer (i.e., $z_{0,k} = x_k$). The last layer outputs a vector, denoted $D(x)$, consisting of a score for each class in C. The classification of the network for input x is the class with the highest score, $c' = \text{argmax}(D(x))$. The layers are functions, denoted $h_1, h_2, \ldots, h_L$, each taking as input the output of the preceding layer. The network's function is the composition of the layers: $D(x) = h_L(h_{L-1}(\cdots(h_1(x))))$. The function of layer m is defined by a set of processing units called neurons, denoted $z_{m,1}, \ldots, z_{m,k_m}$. Each neuron takes as input the outputs of all neurons in the preceding layer and outputs a real number. The output of layer m is the vector $(z_{m,1}, \ldots, z_{m,k_m})^T$ consisting of all its neurons' outputs. A neuron $z_{m,k}$ has a weight for each input $w_{m,k,k'}$ and a single bias $b_{m,k}$. Its function is computed by first computing the sum of the bias and the multiplication of

every input by its respective weight: $\hat{z}_{m,k} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot z_{m-1,k'}$. This output is then passed to an activation function σ to produce the output $z_{m,k} = \sigma(\hat{z}_{m,k})$. Activation functions are typically non-linear functions. In this work, we focus on the ReLU activation function, $\text{ReLU}(\hat{z}) = \max(0, \hat{z})$. We note that, for simplicity's sake, we explain our approach for fully-connected networks, but it extends to other architectures, e.g., convolutional networks.

Local robustness A safety property for neural networks that has drawn a lot of interest is *local robustness*. A network is locally robust at a given input if it does not change the classification under a given type of perturbation. Formally, given a classifier D, an input x and a neighborhood containing x, $I(x) \subseteq \mathbb{R}^d$, we say D is robust at $I(x)$ if: $\forall x' \in I(x).\ \text{argmax}(D(x')) = \text{argmax}(D(x))$. There are many robustness verifiers for neural networks. Most of them can analyze hyperrectangular neighborhoods, where each input entry (i.e., pixel, if the input is an image) is bounded in an interval $[l, u]$, where $l, u \in \mathbb{R}$. These neighborhoods capture popular robustness neighborhoods, e.g., ϵ-balls. Among the robustness verifiers, some are complete, i.e., for every neighborhood, they return *robust* or *non-robust*, while others are incomplete, i.e., they may also return *unknown*. Many complete verifiers rely on constraint solvers, e.g., SAT-solvers [10], SMT-solvers [18], or mixed-integer linear programming (MILP) solvers [38]. Incomplete verifiers often employ linear or convex relaxations to the network's non-linear computations to scale the analysis [29,34,1,41,43,3,13,33,31,32,28]. While complete verifiers tend to be slower than incomplete verifiers, today's MILP solvers are very efficient and can reason about relatively large networks. They also provide a natural way to trade-off accuracy with scalability, as we explain in Section 5.1.

3 Problem Definition

In this section, we define the problem of maximal robust specifications for neural networks. We then discuss the challenges, prior work and the current gap.

Robustness specifications We focus on interval specifications defining hyperrectangular neighborhoods. An interval specification is a sequence of intervals, each corresponding to an input entry and constraining its possible values. Formally, interval specifications are parameterized by an input x and take the form of: $I_{l_1,u_1,\ldots,l_d,u_d}(x) = [l_1, u_1], \ldots, [l_d, u_d]$, where $l_i \leq x_i \leq u_i$, for every $i \in [d]$. The specification's neighborhood contains all inputs bounded by the intervals: $N_{I_{l_1,u_1,\ldots,l_d,u_d}}(x) = \{x' \mid \forall i \in [d].\ x'_i \in [l_i, u_i]\}$. When it is clear from the context, we write I. If $x' \in N_I(x)$, we write $x' \in I$ and say x' is contained in I. We say I is a *robustness specification* for a classifier D if D is robust at $N_I(x)$. Our goal is to compute maximal robust specifications maximizing a given norm. Formally:

Definition 1 (Problem Definition). *Given a classifier D, a correctly classified input x and its class c_x, and a differentiable almost everywhere p-norm $||\cdot||_p$ (e.g., $p = 1, 2, \ldots$), the goal is to compute a specification $I_{l_1,u_1,\ldots,l_d,u_d}(x)$ satisfying:*

1. *D is robust at $N_{I_{l_1,u_1,\ldots,l_d,u_d}}(x)$.*
2. *For every interval specification I' expanding I, D is not robust at $N_{I'}(x)$.*
3. *$I_{l_1,u_1,\ldots,l_d,u_d}(x)$ maximizes $||\cdot||_p$, among specifications meeting 1 & 2.*

This problem is challenging for several reasons. First, it involves searching in high-dimensional space: a specification is a vector in $\mathbb{R}^{2d}$. Second, determining whether a specification belongs to the search space (namely, whether it is robust) involves querying a robustness verifier, which takes non-negligible time. Third, it involves identifying the decision boundaries of the classifier to determine that the specification is maximal. We note that for (uniform) ϵ-ball neighborhoods, computing the maximal neighborhood is significantly simpler. An ϵ-ball specification allows perturbations of each input entry by up to a given ϵ: $B_\epsilon(x) = [x_1 - \epsilon, x_1 + \epsilon], \ldots, [x_d - \epsilon, x_d + \epsilon]$. Namely, an ϵ-ball is defined by a real number ϵ. Thus, computing the maximal ϵ-ball of a given x is a search in a one-dimensional space. It can be done using a binary search, where each candidate ϵ' is submitted to a robustness verifier. Determining whether an ϵ-ball specification is maximal is also simpler and does not require estimating the decision boundaries: the maximal robust ϵ-ball is the one maximizing ϵ. However, as we demonstrate later, considering the more expressive interval specifications leads to revealing a more accurate perspective on the classifier's robustness level.

Prior work and current gap Two works address the problem of maximal robust interval specifications [26,25]. These works assume an incomplete robustness verifier that relies on linear relaxations to bound each neuron by linear bounds. They leverage the linear bounds to overapproximate the classifier's function $D(x)$ as a linear function of the inputs $\tilde{D}(x)$. This allows them to search for a maximal specification using numerical optimization guided by the gradient of $\tilde{D}$. While these approaches compute larger specifications than their counterpart maximal ϵ-ball specifications (as we show in Section 6), they suffer from precision loss. The precision loss stems both from the accumulated overapproximation error of the incomplete verifier's analysis and the inaccuracy of computing the gradient based on $\tilde{D}(x)$ and not the actual classifier's function $D(x)$. As a result, the computed specifications are not maximal. As demonstrated in Figure 1, existing approaches compute non-maximal specifications, which are also significantly smaller than the optimal specification. We note that although Figure 1 demonstrates one of the existing approaches, similar results are obtained for the other one. In this work, we propose a new approach for computing maximal robust specifications.

4 Key Idea: An Oracle-Guided Numerical Optimization

In this section, we present our key idea for computing maximal robust specifications, on which we later build to design MaRVeL. Our goal is to compute a maximal robust specification $I(x)$ maximizing a given norm $||\cdot||_p$. To this end, we rely on a MILP verifier, which loses less precision than verifiers relying on linear relaxations. However, the computation of this verifier is not differentiable and

thus not amenable to numerical optimization, as proposed by prior works [26,25]. On the other hand, numerical optimization is very efficient for (differentiable) maximization problems, and thus we wish to leverage it for searching for candidate specifications. We draw inspiration from program synthesis and propose to rely on *oracle-guided numerical optimization*.

In oracle-guided numerical optimization, we have two entities: the numerical optimizer and the verifier, which interact iteratively. At every iteration, the numerical optimizer computes a new candidate specification and then submits it to the verifier. The verifier checks whether the specification defines a robust neighborhood and returns information to the optimizer that guides it in which directions the current specification can expand (if it is robust) or should shrink (if it is not robust). The process terminates when the optimizer does not have more directions to expand. It then returns the last candidate specification that is robust, according to the verifier. We next formalize the optimization problem that the optimizer solves to compute a maximal robust specification. We then explain at a high-level how the optimizer solves the optimization problem and describe the information provided by the verifier to guide the optimization.

4.1 The Optimization Problem

Ideally, we would like the optimizer to solve a constrained optimization problem over specifications, where the maximization function is the p-norm of the specification and the constraints are that the specification is valid (i.e., contains x) and robust. We note that, in this section, we ignore the domain constraints, bounding the input entries by minimum and maximum values, because they are enforced differently (explained in Section 5.3). Expressing that the specification is valid is straightforward: we require $x_i \geq l_i$ and $x_i \leq u_i$, for every $i \in [d]$. Expressing that the specification is robust is more subtle because it requires to enforce that the network classifies every input contained in the specification as c_x (i.e., x's class):

$$\forall x' \in I(x).\ \text{class}(D(x')) = c_x$$

However, this constraint is not differentiable, because we rely on a MILP-based encoding of the network's computation, to avoid precision loss. Thus, we rewrite this constraint into a term, which is easier for differentiation, preserving the constraint's semantics. We begin with an equivalent constraint requiring that the difference between c_x's score and the maximal score of any other class is positive:

$$\forall x' \in I(x).\ D(x')_{c_x} - \max\{D(x')_{c'} \mid c' \neq c_x\} > 0$$

Next, to eliminate the for-all operator, which is generally not supported by numerical optimizers, we rewrite this constraint by requiring that the minimum value of the above difference is positive:

$$\min\{D(x')_{c_x} - \max\{D(x')_{c'} \mid c' \neq c_x\} \mid x' \in I(x)\} > 0$$

This constraint has the same semantics: if the minimal value of this difference is positive, then the specification is robust, and otherwise, it is not robust. We call the minimal difference the *robustness level*. We next define it formally.

Definition 2 (Robustness Level). *Given a classifier D, a correctly classified input x and its class c_x, and a specification $I(x)$, the* robustness level *of $I(x)$ is* $RL(I(x)) = \min\{D(x')_{c_x} - \max\{D(x')_{c'} \mid c' \neq c_x\} \mid x' \in I(x)\}$.

Lastly, since such constraint is challenging for a numerical optimizer, we relax it by adding the robustness level as an additional term to the maximization function (such relaxation is common, for example, to compute adversarial examples [4,6,39,37]). By aiming to maximize the robustness level, the optimizer guides its search towards robust specifications. Overall, the optimizer computes a maximal robust specification by solving the following optimization problem:

$$\begin{aligned} \max_{I_{l_1,u_1,\ldots,l_d,u_d}} \; & ||I_{l_1,u_1,\ldots,l_d,u_d}||_p + \lambda \cdot RL(I_{l_1,u_1,\ldots,l_d,u_d}) \\ \text{subject to} \quad & \\ & x_i \geq l_i \qquad \forall i \in [d] \\ & x_i \leq u_i \qquad \forall i \in [d] \end{aligned} \tag{1}$$

Here, λ is the balancing term, which we define in Section 5.2. This constrained problem aims to maximize both the specification's size and its robustness level.

We note that although an optimal solution to this problem may be a non-robust specification, this does not affect the overall soundness of our approach. This is because every candidate is submitted to a sound verifier, and eventually we return the maximal specification that is robust, according to the verifier.

4.2 Solving the Optimization Problem

To solve the optimization problem, the optimizer runs stochastic gradient descent (SGD). At every SGD iteration, the optimizer computes the gradient of the maximization problem and accordingly updates the current specification by a small step: $I \mapsto I + \eta \cdot \nabla(||I||_p + \lambda \cdot RL(I))$. Afterwards, it clips the specification to respect the validity constraints $x_i \geq l_i$ and $x_i \leq u_i$. The main question is how to compute the gradient of the maximization function. Since the norm is differentiable almost everywhere, the challenge is only in computing the gradient of $RL(I)$. Our idea is to rely on the robustness level that the MILP verifier computes as part of its analysis, and in particular on the *inputs* defining the robustness level. That is, the inputs minimizing the difference between the score of c_x and the maximal score of any other class. We call these inputs the *weakest points*. We next define them formally.

Definition 3 (Weakest Points). *Given a classifier D, a correctly classified input x and its class c_x, and a specification $I(x)$, the* weakest points *of $I(x)$ is the following set of inputs $\hat{\mathcal{W}} \subseteq \mathbb{R}^d$: $\hat{\mathcal{W}} = \{x' \in I(x) \mid RL(x') = RL(I(x))\}$, where for every $x' \in \mathbb{R}^d$, we define $RL(x') = D(x')_{c_x} - \max\{D(x')_{c'} \mid c' \neq c_x\}$.*

We next explain how the weakest points enable the optimizer to compute the gradient of $RL(I)$. By the definition of robustness level, we have $RL(I) = RL(\hat{\mathcal{W}})$. In particular, their gradients are equal: $\nabla RL(I) = \nabla RL(\hat{\mathcal{W}})$. Thus, to compute

the gradient of $RL(I)$, the optimizer computes the gradients of the weakest points, which is typically a very small set. Computing the gradient at a single point $x' \in \hat{\mathcal{W}}$ is a simple standard computation involving a forward pass and a backward pass over the classifier D. The gradient of $\hat{\mathcal{W}}$ is the average of the weakest points' gradients. In practice, the gradient of the weakest points may direct to other decision boundaries, which are close to inputs with a low robustness level, but not the lowest. To avoid it, we identify the set of classes with a low robustness level $C' \subseteq C \setminus \{c_x\}$ and obtain from the verifier's analysis the weakest points of every class $c' \in C'$, namely the inputs minimizing the robustness level for every class in C'. Formally, given the set of classes with a low robustness level C', its set of weakest points is $\mathcal{W} = \{\mathcal{W}_{c'} \mid c' \in C'\}$, where for $c' \in C'$, $\mathcal{W}_{c'} = \{x' \in I(x) \mid D(x')_{c_x} - D(x')_{c'} = RL_{c'}(I(x))\}$ and $RL_{c'}(I(x)) = \min\{D(x')_{c_x} - D(x')_{c'} \mid x' \in I(x)\}$. The optimizer constructs a weighted gradient from all the points in $\bigcup \mathcal{W}$ (defined in Section 5.2).

5 MaRVeL: Computing Maximal Robust Specifications

In this section, we present MaRVeL, our algorithm to compute maximal robust interval specifications. MaRVeL builds on oracle-guided numerical optimization (Section 4). Figure 2 shows its operation. MaRVeL takes as arguments a classifier D, an input x and its class c_x. Throughout execution, it maintains three specifications: the current specification I, the last verified robust specification I^r, and the termination specification I^f, keeping the maximal bounds. Initially, I is the specification containing only x, and I^r and I^f are undefined. MaRVeL operates iteratively to maximize the optimization problem of Equation (1). It begins at the Verify step. This step begins with a call to a fast incomplete verifier to identify the set of classes with low robustness levels C'. For every $c' \in C'$, it encodes the verification task as a MILP and submits it to a MILP solver. The solver returns, for every class $c' \in C'$, the weakest points and their robustness level $\mathcal{W}_{c'}$. MaRVeL then continues to the Progress step to decide how to advance the computation. If I is robust, I^r is updated. Otherwise, MaRVeL resets I to the previous I^r and updates I^f using CEGIS, to prevent expanding in the maximal directions. It further updates the balancing factor λ_0 (described later), if I is not sufficiently larger than the previous I^r or if I is not robust. Then, MaRVeL checks the termination conditions. MaRVeL terminates in one of the following cases: (1) if x is misclassified (in which case, I is set to an undefined I^r and thus has $\perp$), (2) all bounds are maximal (I^f has no $\perp$), or (3) the balancing factor λ_0 is below a predetermined threshold ($\lambda_0 < \lambda_{min}$). If MaRVeL does not terminate, it continues to the Optimize step. This step first computes the specification size's gradient, the robustness level's gradient (from the weakest points), and the value of λ. Accordingly, it updates I. Lastly, it employs clipping to I based on the validity constraints (i.e., $l_i \leq x_i \leq u_i$) and enforces the bounds in I^f. When one of the termination conditions is true, the last robust specification I^r is returned. We next explain these steps, show an example, and discuss correctness.

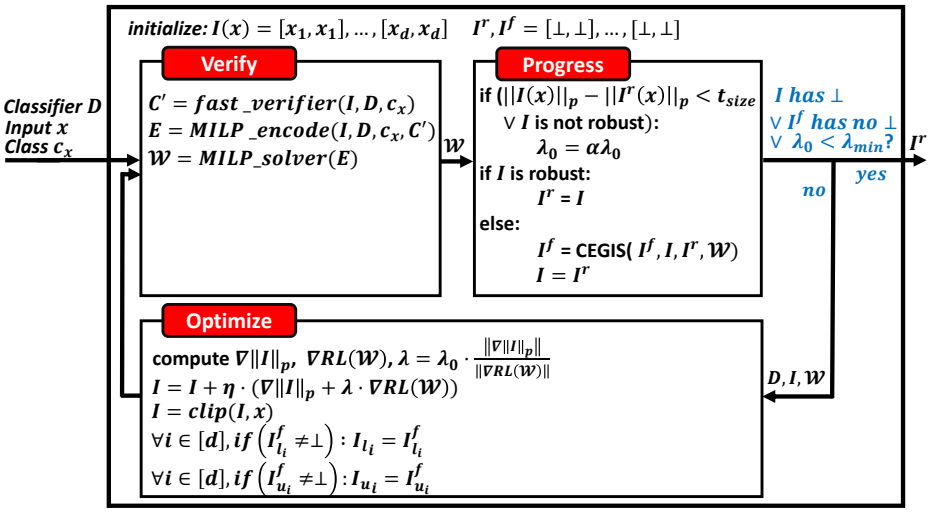

Fig. 2: MaRVeL: System description.

5.1 The Verify Step

For the verifier, MaRVeL relies on the MILP encoding of an existing MILP-based robustness verifier [38]. We begin with a short background on its encoding, which is necessary to understand MaRVeL's encoding and optimizations, and then describe MaRVeL's call to the fast incomplete verifier and the optimizations that MaRVeL employs.

Background: A MILP robustness verifier The MILP verifier [38] encodes the robustness analysis as MILPs, which are then submitted to a MILP solver. We begin by describing the encoding of the network's computation. Given a network, the encoding associates to each neuron $z_{m,k}$ the following (we abuse notation, for simplicity's sake): (1) a real-valued variable $\hat{z}_{m,k}$ for the affine computation, (2) a real-valued variable $z_{m,k}$ for the ReLU computation, (3) concrete lower and upper bounds $l_{m,k}, u_{m,k} \in \mathbb{R}$, and (4) a boolean variable $a_{m,k} \in \{0, 1\}$. For every neuron, it adds the following constraints, capturing the neuron's computation:

$$\hat{z}_{m,k} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot z_{m-1,k'} \qquad z_{m,k} \geq 0 \qquad z_{m,k} \geq \hat{z}_{m,k}$$

$$z_{m,k} \leq u_{m,k} \cdot a_{m,k} \qquad z_{m,k} \leq \hat{z}_{m,k} - l_{m,k}(1 - a_{m,k})$$

The concrete bounds are computed before encoding the network, for example using interval arithmetic or a fast incomplete verifier (which, as mentioned, MaRVeL runs before the MILP verifier). These constraints capture the network's computation precisely, without any approximation. Note that because the encoding relies on boolean variables, the overall function is a step function, and thus

not differentiable. Given an interval specification $I_{l_1,u_1,\ldots,l_d,u_d}(x)$, the encoding adds the constraints: $z_{0,k} \geq l_k$ and $z_{0,k} \leq u_k$, for every $k \in [d]$. To encode robustness of c_x with respect to class $c' \neq c_x$, i.e., show that the score of c_x is higher than that of c', it adds a minimization function: $\min(z_{L,c_x} - z_{L,c'})$. An optimal solution to this MILP is the robustness level of c'. The set of inputs obtaining this robustness level is the set of weakest points of c', $\mathcal{W}_{c'}$. Overall, to show local robustness, i.e., show robustness of c_x with respect to every class $c' \neq c_x$, the verifier submits to a MILP solver $|C| - 1$ MILPs. If all optimal solutions (i.e., robustness levels) are positive, the classifier is locally robust at the given interval specification. Otherwise, it is not robust. That is, this encoding provides a sound and complete local robustness analysis [38]. In addition to the robustness levels, the MILP solver can also return the sets of weakest points.

The fast verifier The MILP verifier is generally an efficient approach for exact analysis. However, since MaRVeL invokes it at every iteration, it becomes highly time consuming, especially for large networks. Thus, at every iteration, MaRVeL attempts to reduce the number of MILPs by pruning classes whose robustness level is not low. Thus, before the MILP encoding, MaRVeL runs DeepPoly [33]. DeepPoly is an incomplete robustness verifier, relying on linear relaxations to scale the analysis. As part of its analysis, DeepPoly computes, for every output neuron, real-valued lower and upper bounds bounding the possible values. MaRVeL relies on these bounds to compute, for every class $c' \neq c_x$, a lower bound on the robustness level. Then, it constructs the set C' of all classes whose robustness level's lower bound is not positive. Note that the MILP verifier need not check the other classes to determine local robustness. If $C' = \emptyset$ (it may happen for very small specifications), it adds to C' the class with the minimal robustness level.

The MILP verifier MaRVeL encodes a MILP for each class in C', as described, and submits them to a MILP solver. To reduce execution times, it employs two optimizations. First, it employs a *partial* MILP encoding. That is, it encodes only part of the neurons using boolean variables, and the rest are overapproximated with DeepPoly's linear constraints (which lose precision). This allows MaRVeL to trade-off precision with scalability. Specifically, MaRVeL limits the number of neurons that are encoded precisely at every layer to n_m (a hyper-parameter). There are several heuristics to determine which neurons require precise encoding [34,43,41,42]. MaRVeL employs a common heuristic. It picks for every layer the n_m neurons with the largest overapproximation error, i.e., the largest difference between their upper bound and lower bound, as determined by DeepPoly. Second, many MILP solvers support anytime computations. Thus, MaRVeL runs the MILP solver with a predetermined timeout T_{MILP}. If the solver reaches this timeout, it returns the current optimal solution. We note that these optimizations do not affect the soundness of MaRVeL, but may reduce its accuracy.

5.2 The Optimize Step

The Optimize step begins by computing the gradient of the maximization function: $||I_{l_1,u_1,\ldots,l_d,u_d}||_p + \lambda \cdot RL(I_{l_1,u_1,\ldots,l_d,u_d})$. This computation follows the description

at Section 4.2, and we next explain it in detail. The gradient is a vector in $\mathbb{R}^{2d}$, defining for every l_i and u_i its derivative. Computing the gradient of $\nabla||I||_p$ is straightforward, for $p \geq 1$. For example, if $p = 2$, namely $||I||_2 = \sum_{i=1}^{d}(u_i - l_i)^2$, then $\nabla||I||_2 = (2l_1 - 2u_1, 2u_1 - 2l_1, \ldots, 2l_d - 2u_d, 2u_d - 2l_d)^T$. To compute the gradient of the robustness level $\nabla RL(I)$, MaRVeL computes $\nabla RL(\mathcal{W})$. To this end, it computes for each input $x' \in \mathcal{W}_{c'}$, where $\mathcal{W}_{c'} \in \mathcal{W}$, the gradient of $RL(x')$. This gradient is $\nabla_{x'}(z_{L,c_x} - z_{L,c'})$, and it can be computed as standard, using a forward pass and a backward pass over the classifier D. Given the gradient of $RL(x')$, which is a vector $(\dot{x}'_1, \ldots, \dot{x}'_d)^T \in \mathbb{R}^d$, MaRVeL defines for every $i \in [d]$ the derivative of l_i and u_i to be $\dot{x}'_i$. Theoretically, the gradient of the points in $\bigcup \mathcal{W}$ is the component-wise average. However, the weakest points of different classes have different robustness levels. Taking the average gradient assigns the same importance to all points, even if some are closer to a decision boundary than others. Instead, MaRVeL computes a weighted average: $\nabla RL(\mathcal{W}) = \frac{1}{|\bigcup \mathcal{W}|} \sum_{x' \in \bigcup \mathcal{W}} \frac{exp(-RL(x'))}{\sum_{\bar{x} \in \bigcup \mathcal{W}} exp(-RL(\bar{x}))} \cdot \nabla_{x'} RL(x')$. This weighted average assigns higher weights to the gradients of inputs with lower robustness levels. Having defined the gradients of each component, the overall gradient is the sum $\nabla||I||_p + \lambda \cdot \nabla RL(I)$. This is then normalized, as standard, by dividing it by its norm: $||\nabla||$. Our balancing term λ is a function of the gradients' ratio: $\lambda = \lambda_0 \cdot \frac{||(\nabla||I||_p)||}{||(\nabla RL(I))||}$, where λ_0 is initialized to a predetermined factor and decreases during the optimization. To allow the specifications expand at a reasonable rate, if I is not robust or $||I||_p - ||I^r||_p < t_{\text{size}}$, for a threshold t_{size}, MaRVeL multiplies λ_0 by a constant $\alpha \in (0, 1)$. This update directs the optimizer to assign more weight to the specification's size term in the next iterations.

Specification update After computing the gradient, the specification update is a standard SGD step: $I \mapsto I + \eta \cdot (\nabla||I||_p + \lambda \cdot \nabla RL(I))$, where η is a small constant. The intuitive meaning of a single step is that MaRVeL updates the specification with the goal of increasing its size while expanding the bounds away from the current weakest points. After that, the specification is clipped to satisfy the validity constraints. Namely, every u_i that is smaller than x_i is set to x_i, and every l_i that is greater than x_i is set to x_i. Then, the specification is aligned with the maximal and minimal bounds in I^f. Namely, every u_i or l_i that has a value in I^f is set to its value in I^f. The domain constraints bounding the input entries by minimum and maximum values are enforced through I^f, as we next explain.

5.3 CEGIS at the Progress Step

Lastly, we explain the CEGIS operation at the Progress step. Its goal is to leverage non-robust specifications to identify when MaRVeL reaches maximal bounds and thereby prevent the optimizer from proposing non-robust specifications. This operation draws inspiration from counterexample-guided inductive synthesis (CEGIS), where a program synthesizer prunes its search space after obtaining a counterexample from the oracle or user [36,17]. As described, if a specification is not robust, MaRVeL discards it and continues from the last robust specification.

However, before discarding it, MaRVeL computes a set of maximal bounds and updates I^f accordingly. The maximal bounds are computed from the weakest points that are adversarial examples (if a specification is not robust, some of the weakest points are adversarial examples). By restricting these bounds, future specifications will not include these adversarial examples. Moreover, a clever restriction will also eliminate very close adversarial examples that otherwise will be discovered in the following iterations, thereby slowing down the computation.

We begin with several observations and afterward explain how MaRVeL leverages them to construct I^f. Consider two consecutive specifications $I^r_{l^r_1,u^r_1,\ldots,l^r_d,u^r_d}$ and $I_{l_1,u_1,\ldots,l_d,u_d}$, where I^r is robust and I is not. The MILP verifier computes for I the set of weakest points. Because I is not robust, at least one of them is an adversarial example, denoted $x' = (x'_1, \ldots, x'_d)^T$. For every $i \in [d]$, one of the following holds: (1) $x'_i \in (u^r_i, u_i]$, (2) $x'_i \in [l_i, l^r_i)$, or (3) $x'_i \in [l^r_i, u^r_i]$. Because x' is an adversarial example and I^r is robust, there exists $i \in [d]$, for which cases (1) or (2) hold. We define $B_{x'}$ to be the set of bounds satisfying cases (1) or (2):

$$B_{x'} = \{U_i \mid x'_i \in (u^r_i, u_i]\} \cup \{L_i \mid x'_i \in [l_i, l^r_i)\}$$

Our first observation is that if, for future specifications, we prohibit *any* bound in $B_{x'}$ from reaching its respective value in I, then x' is not part of future specifications' neighborhoods. To eliminate all weakest points, it is sufficient to eliminate a single bound for each of them. Thus, the most permissive restriction on future specifications is a minimal hitting set over all $B_{x'}$-s. Namely, $B = \text{argmin}_{B \in \mathcal{B}}|B|$, where $\mathcal{B} = \{B \subseteq \{L_1, U_1, \ldots, L_d, U_d\} \mid \forall x' \in \bigcup \mathcal{W}.\ B_{x'} = \emptyset \vee B_{x'} \cap B \neq \emptyset\}$. We can prove that if MaRVeL removes the most permissive restriction B at every iteration in which I is not robust, then MaRVeL returns a maximal robust specification (Section 5.5, Theorem 1).

Our second observation is that, in practice, the most permissive restriction results in high execution times, especially for high-dimensional specifications. This is because adversarial examples are not sporadic and often multiple adversarial examples appear in the same region [8]. Thus, while eliminating a single bound removes a particular adversarial example, it does not eliminate the adversarial region. Although eventually all adversarial examples in this region are removed, it requires many iterations in which I^r is not updated, causing a time waste.

Computing a minimal set of bounds defining adversarial regions is not trivial. Instead, we overapproximate it with the union of the bounds: $\tilde{B} = \bigcup_{x' \in \bigcup \mathcal{W}} B_{x'}$. While this is the most restrictive approach, we empirically observe that among all approaches we experimented with, it leads to a minimal number of iterations until the optimizer again computes a candidate specification which is discovered as robust. We believe the reason is that the SGD's step size is very small, and thus if a bound is included in any $B_{x'}$, it should be restricted. When experimenting with less restrictive approaches (e.g., computing a minimal hitting set or restricting bounds based on their frequencies), MaRVeL required many more iterations to restrict all necessary bounds. During these iterations, I^r remains the same, because the specifications are not robust. Consequently, when limiting MaRVeL

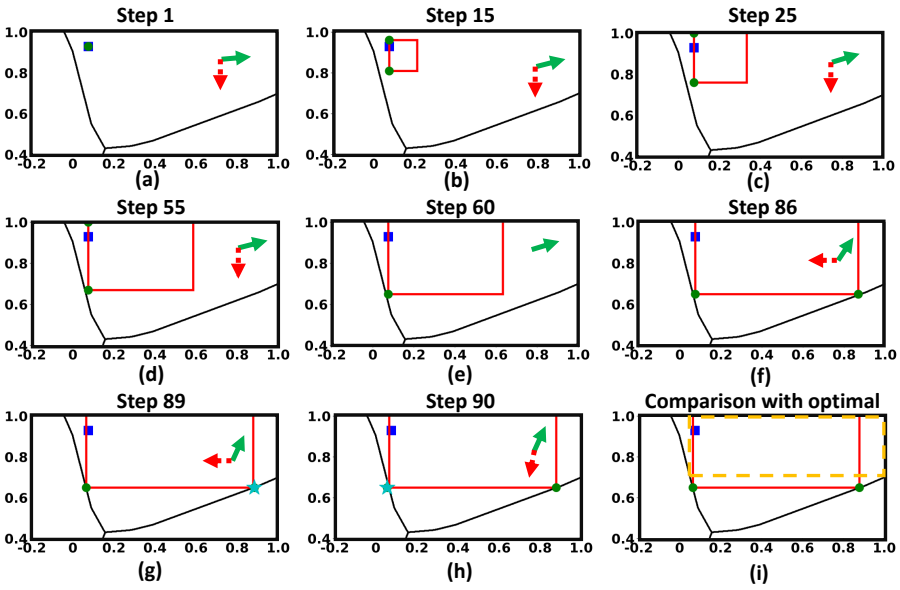

Fig. 3: A running example for computing MaRVeL's specification at Figure 1.

with a one hour timeout, the average diameter of the less restrictive approaches is at best 80% of the average diameter of the specifications computed with $\tilde{B}$.

MaRVeL builds on these observations to compute the maximal bounds. As described, it maintains a termination specification I^f, keeping for each bound a maximal or a minimal value. Initially, all bounds in I^f are undefined. At every iteration in which the verifier determines that I is not robust, I^f is updated based on the weakest points that are adversarial examples. To this end, MaRVeL first computes $\tilde{B}$. Then, for every $U_i \in \tilde{B}$, it sets in I^f at index u_i the value $I^r_{u_i}$, and for every $L_i \in \tilde{B}$, it sets in I^f at index l_i the value $I^r_{l_i}$. While we could set the bounds in I^f to the respective values in I minus a small constant, in practice this does not eliminate the adversarial region. Additionally, because MaRVeL advances I by small steps, the difference between $I^r_{u_i}$ and I_{u_i} is very small.

We note that I^f is also updated when bounds reach their maximal or minimal domain value. For example, assume the input domain is $[0,1]^d$. If the verifier determines a specification I is robust, then for every $u_i = 1$ and $l_i = 0$ in I, their respective value in I^f is updated to 1 or 0 (respectively). This is required to guarantee termination (Section 5.5, Lemma 1).

5.4 An End-to-End Example

We next exemplify MaRVeL for the specification presented at Figure 1. In this example, the classifier D is a fully-connected network, taking two-dimensional inputs in the range $[-1, 1]$ and consisting of three layers, each with ten neurons.

The input is $x = (0.075, 0.93)^T$ and its class is $c_x = 7$. Given these arguments, MaRVeL computes a maximal robust specification with respect to the L_1 norm ($p = 1$). Figure 3 visualizes the key steps in MaRVeL's verification process. Every figure shows the following. The black curves show the decision boundaries. The blue square shows the input x. The red rectangle shows the current specification I. The green dots show the weakest points, and the light blue stars show the weakest points that are adversarial examples. The gradient is shown as two arrows (to simplify its visualization): the green arrow shows the gradient of the upper bounds and the dashed red arrow shows the gradient of the lower bounds.

At step 1, MaRVeL initializes the specification I to contain only the input x: $I_1 = [0.075, 0.075], [0.93, 0.93]$. It also initializes $I^r = I^f = [\bot, \bot], [\bot, \bot]$. The Verify step runs DeepPoly and determines that only $c' = 8$ has to be checked by the (MILP) verifier. That is, instead of nine MILPs, only one is encoded and submitted to the verifier. The verifier returns $\mathcal{W}_8 = \{x\}$ and its robustness level, which is positive. Namely, I is robust. No termination condition is true, and so MaRVeL computes the gradient of $\mathcal{W}_8 = \{x\}$. Accordingly, it expands the specification. Then, it clips to ensure that the specification contains x. Step 15 shows a very similar scenario only that there are two weakest points for $c' = 8$.

At step 25, the verifier returns that $I = [0.075, 0.32], [0.75, 1]$ is robust. Because one of the bounds reaches its maximal domain value, its respective value in I^f is updated: $I^f = [\bot, \bot], [\bot, 1]$. The specification is expanded as before.

At step 55, $I = [0.075, 0.61], [0.65, 1]$ and it approaches the decision boundary of class 8. The weakest point is $(0.075, 0.66)^T$. The gradient of this point directs to expand I only in the right-up direction (demonstrated also in step 60).

At step 86, I approaches the decision boundary of class 6. The verifier returns $\mathcal{W} = \{\{(0.065, 0.65)^T\}_8, \{(0.87, 0.64)^T\}_6\}$. The first point corresponds to $c' = 8$ and the second point to $c' = 6$. At step 89, I is not robust and one of the weakest points is an adversarial example: $(0.88, 0.649)^T$. MaRVeL constructs $\tilde{B} = \{U_1, L_2\}$ and updates their respective values in I^f based on their values in the last I^r: $I^f = [\bot, 0.878], [0.649, 1]$. At step 90, I is not robust and one of the weakest points is an adversarial example: $(0.055, 0.649)^T$. MaRVeL constructs $\tilde{B} = \{L_1\}$ and updates its respective bound in I^f based on the last I^r: $I^f = [0.065, 0.878], [0.649, 1]$. At this point, there is no direction that MaRVeL can expand. Thus, MaRVeL terminates and returns the last robust specification $I^r = [0.065, 0.878], [0.649, 1]$. The figure shows that the specification is maximal: expanding any bound results in including an adversarial example. Figure 3(i) compares MaRVeL's specification with the optimal one (the dashed yellow rectangle), whose average diameter is larger by only 7%.

5.5 Correctness and Running Time

In this section, we discuss correctness and running time analysis.

Correctness By MaRVeL's operation, it is sound because if it returns a defined specification (without $\bot$), it must have been verified by the MILP verifier, which provides a sound robustness analysis [38]. Under the following conditions the

returned specification is maximal: (1) MaRVeL relies on the minimal hitting set B, (2) the step sizes are small enough (as standard in numerical optimization), (3) the MILP verifier is precise (i.e., the optimizations of the partial MILP and the anytime computations do not reduce its accuracy), and (4) MaRVeL terminates because I^f has no $\perp$. We next formalize this in a theorem.

Theorem 1. *Let D, x and c_x be arguments to MaRVeL. If MaRVeL relies on the minimal hitting set, the step sizes are small, the MILP verifier is precise, and MaRVeL completes because I^f has no $\perp$, then its specification is maximal: expanding any bound that has not reached its maximal or minimal possible value results in including an adversarial example.*

Proof (Sketch). Assume MaRVeL's maximization function was $\max ||I||_p$. Then, at every iteration, the gradient is positive for any upper bound and negative for any lower bound, because an L_p norm is a monotonically increasing function[1]. Thus, at every iteration, the SGD step updates the current specification by increasing every u_i and decreasing every l_i that are not limited by I^f or the validity constraints. Thus, if a bound u_i stops increasing (or a bound l_i stops decreasing) and if it is not because of the validity constraints or because u_i has reached its maximal domain value, then it is because u_i prevents an adversarial example. This is guaranteed since the MILP verifier is precise. Because the step sizes are small, the bound of u_i is maximal (or the bound of l_i is minimal). Because MaRVeL relies on the minimal hitting set B, every adversarial example is prevented by limiting a single bound and no bound can be omitted from B without including an adversarial example. Thus, if the optimization is completed, it must be that every bound is preventing an adversarial example or has reached the maximal or minimal possible value. A similar reasoning applies to our maximization function with the robustness level, thanks to the adaptive definition of λ_0. Recall that if the specification size increases too slowly or the specification is not robust, then λ_0 decreases. Thereby, MaRVeL assigns more weight to the specification's size term. Thus, the optimization process cannot terminate without attempting to increase every bound, due to the gradient of $||I||_p$. Hence, if the optimization is completed, it must be that every bound is preventing an adversarial example or has reached the maximal or minimal possible value.

If MaRVeL relies on $\tilde{B}$, we provide a lower bound on the number of dimensions in which the specification is maximal. Every time the specification I^f is updated, at least one of I^r's bounds is maximal, because $\tilde{B}$ is a hitting set. Given all (disjoint) sets $\tilde{B}_1, \ldots, \tilde{B}_k$ throughout the execution, the number of maximal bounds is at least k. This is a very loose lower bound, since in practice, several bounds tend to be maximal together, thereby inducing an adversarial region.

[1] There is an edge case where $l_i = u_i$, in which case the gradient is zero. There are standard corrections to guarantee that the gradient is monotonically increasing. For example, for the L_1 norm, which is the one currently supported in our implementation, the correction replaces the zero gradient by 1 (for u_i) or -1 (for l_i).

Running time Next, we analyze the running time of MaRVeL. We start with a lemma guaranteeing termination. Then, we analyze the running time of a single iteration of MaRVeL.

Lemma 1. *For every D, x and c_x, MaRVeL terminates.*

Proof (Sketch). At every iteration, one of the following holds:
- The current specification is robust: In this case, in most iterations, the size of the current specification is larger than previous specifications. We note that it may be that for a small number of iterations it is not the case, but then λ_0 decreases until the specification's size becomes large enough.
- The current specification is not robust: In this case, at least one of the bounds is set to a value (if it was $\perp$) or is tightened by the respective value in I^r. We note that a bound can be tightened in case MaRVeL relies on a more permissive set of bounds than $\tilde{B}$.

Because at every update of I^f at least one bound is set or tightened and because the step size is a discrete number, the number of iterations in which I^f is updated is finite. If at some iteration, I^f has no $\perp$, then MaRVeL terminates. Otherwise, it must be that at least one bound can continue increasing or decreasing. In this case, MaRVeL continues expanding the specification (by the definition of λ_0). If MaRVeL does not terminate because of I^f, even though every input entry is bounded by a minimum and maximum values, it must be that the specification's size increases too slowly, even when λ_0 continues decreasing. In this case, at some iteration, λ_0 decreases below λ_{min} and MaRVeL terminates.

The maximal running time of a single iteration of MaRVeL is the sum of $T_{\text{DeepPoly}} + |C-1| \cdot T_{\text{MILP}} + |\bigcup \mathcal{W}| \cdot T_D$, where T_{DeepPoly} is the execution time of the incomplete verifier DeepPoly, T_{MILP} is the execution time of the MILP verifier (recall that MaRVeL sets a timeout to the solver), and $|\bigcup \mathcal{W}| \cdot T_D$ is the time to compute the gradient of the weakest points, involving a forward pass and a backward pass to each over the classifier D. The other computations take a negligible time. Note that because $\mathcal{W}$ is computed by a MILP solver, $\bigcup \mathcal{W}$ is finite. The dominant factor of the running time is T_{MILP} (under reasonable choices). To mitigate it, MaRVeL solves the MILPs parallelly. Naturally, advances in complete robustness verification or MILP solvers can reduce MaRVeL's execution time.

6 Evaluation

In this section, we evaluate MaRVeL. We begin by describing our experiment setup and baselines and then present our experiments.

Experiment setup We implemented MaRVeL[2] in Python, as a module in ERAN[3], to easily integrate with DeepPoly and the MILP-based verification. MaRVeL

[2] https://github.com/ananmkabaha/MaRVeL.git
[3] https://github.com/eth-sri/eran

Table 1: The networks used in our experiments.

Dataset	Name	Architecture	#Neurons
MNIST	3×50	Fully-connected	100
	3×100	Fully-connected	200
	$Conv2$	Convolutional	2948
Fashion-MNIST	3×50	Fully-connected	100
	3×250	Fully-connected	500
	$Conv2$	Convolutional	2948
	$Conv3$	Convolutional	3664
CIFAR-10	3×400	Fully-connected	800
	$Conv2$	Convolutional	1188
	$Conv3$	Convolutional	4368
Contagio/Virustotal	3×50	Fully-connected	100
	3×100	Fully-connected	200

leverages ERAN's RefinePoly domain that runs DeepPoly and then the MILP verifier as described in Section 5.1. The MILP solver is Gurobi. Experiments ran on an Ubuntu 20.04.1 OS on a dual AMD EPYC 7713 server with 2TB RAM. We evaluated MaRVeL over several datasets. First, image datasets: MNIST [24] and Fashion-MNIST [44], consisting of 28×28 gray-scale images, and CIFAR-10 [20], consisting of $32 \times 32 \times 3$ colored images. Second, Contagio/Virustotal [7,40], a malware dataset consisting of malicious and benign PDF files, each with 135 features. Table 1 shows the different networks we used. Their activation function was ReLU. The $Conv2$ architecture comprised of two convolutional layers followed by two fully-connected layers, while $Conv3$ comprised of three convolutional layers followed by three fully-connected layers. We also used a toy synthetic dataset consisting of two-dimensional inputs [26], described later, to visualize the size of the specifications with respect to the decision boundaries. In our experiments, the norm is L_1 ($p = 1$), the balancing factor is $\lambda_0 = 0.99$, the number of precise neurons is $n_m = 200$, and the MILP timeout is $T_{\text{MILP}} = 100$ seconds.

Baselines We compare MaRVeL to existing works on computing maximal robust specifications [26,25]. Both approaches rely on CROWN [49], an incomplete robustness verifier, which overapproximates ReLU with linear constraints. They differ in the kind of specifications they compute. Liu et al. [26] compute non-uniform, symmetric robust specifications, defined by a non-negative vector ϵ: $I_\epsilon(x) = [x_1 - \epsilon_1, x_1 + \epsilon_1], \ldots, [x_d - \epsilon_d, x_d + \epsilon_d]$. Li et al. [25] build on [26] to compute non-uniform, asymmetric robust specifications, like our specifications. We used Liu et al.'s code [4], which supports only fully-connected networks, and extended their code to support Li et al.'s approach. We compare MaRVeL and

[4] https://github.com/liuchen11/CertifyNonuniformBounds

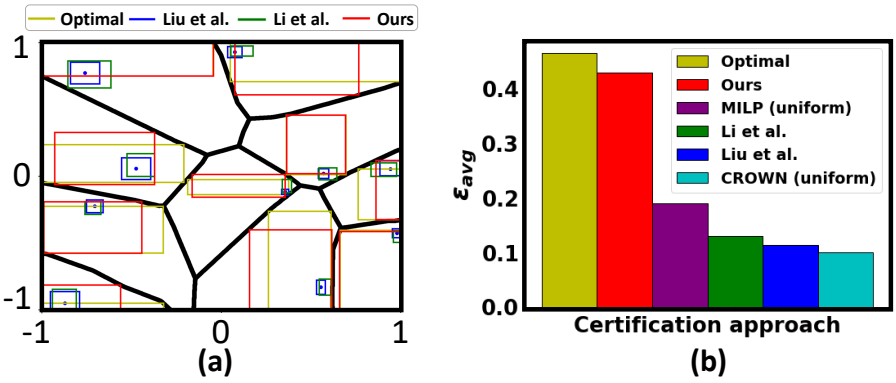

Fig. 4: A comparison of the maximal specifications computed by MaRVeL, by prior works, and by an impractical optimal approach, over a 2D synthetic dataset.

these works by measuring the specifications' average diameter $\epsilon_{avg} = \frac{\sum_{i=1}^{d} u_i - l_i}{d}$. Since MaRVeL and these works rely on different robustness verifiers, for a fair comparison, we also report the diameter ϵ_u of the maximal robust (uniform) ϵ-ball of CROWN and the MILP verifier. This is computed by a binary search running 15 steps and starting from $\epsilon_0 = 0.1$.

Synthetic dataset We start by considering the toy synthetic dataset, presented by [26]. This dataset consists of two-dimensional inputs $x_1, x_2 \in [-1, 1]$ and ten classes $C = [1, \ldots, 10]$. We create training and test sets by randomly generating 9000 inputs and 1000 inputs, respectively. We consider a fully-connected network with two hidden layers, each with 10 ReLU neurons. After training, it reaches over 99% accuracy on the test set. Figure 4(a) shows the network's decision boundaries (the black curves). We run MaRVeL and both baselines on ten inputs. We further compare to a (highly impractical) optimal approach that computes all decision boundaries around the given input using a grid search, accordingly computes all maximal robust specifications, and returns the specification maximizing the L_1 norm. Figure 4(a) shows the maximal robust specifications of each approach. It shows that MaRVeL's specifications reach the decision boundaries and cannot be expanded in any dimension. In contrast, both prior works compute significantly smaller specifications. Part of the difference is attributed to the underlying verifier (MILP-based vs. CROWN). To illustrate this, Figure 4(b) shows the average diameter, over 100 inputs, of the specifications computed by the non-uniform approaches and of the maximal uniform ϵ-ball computed by the verifiers. The results show that the average diameter of MaRVeL is 93% of the optimal approach's average diameter, while the diameter of the prior works is only 26%. The average diameter of the maximal uniform ϵ-balls computed by CROWN is only 50% of that computed by the MILP verifier. On average, the execution time of MaRVeL is 12.1 seconds and the execution time of prior works is 7.8 seconds.

Table 2: A comparison of MaRVeL to the baselines over fully-connected networks.

Dataset	Network	MaRVeL					Li et al.		Liu et al.		
		$T^f[m]$	ϵ^f_{avg}	$T[m]$	ϵ_{avg}	ϵ_u	$T[m]$	ϵ_{avg}	$T[m]$	ϵ_{avg}	ϵ_u
Contagio	3×50	23.1	0.74	41.8	0.91	0.35	0.14	0.17	0.11	0.16	0.16
	3×100	11.9	0.31	28.2	0.40	0.26	0.24	0.15	0.22	0.15	0.14
MNIST	3×50	14.0	0.172	42.2	0.196	0.073	0.52	0.027	0.48	0.026	0.024
	3×100	12.4	0.093	32.6	0.10	0.068	4.2	0.025	3.12	0.022	0.020
F-MNIST	3×50	10.2	0.147	47.0	0.191	0.066	3.9	0.029	2.40	0.0295	0.024
	3×250	4.9	0.031	21.7	0.037	0.028	1.3	0.015	0.9	0.014	0.010
CIFAR-10	3×400	0.7	0.015	21.3	0.047	0.015	3.2	0.007	2.8	0.007	0.002

Real datasets We next evaluate MaRVeL over the image datasets and the malware dataset. We compare MaRVeL to our two baselines over the fully-connected networks, because their code does not support other architectures. For every network, we run each approach over 50 inputs with a one hour timeout. We measure the execution time in minutes T and the average diameter ϵ_{avg}. To understand the advantage of the CEGIS step, we compare to a variant of MaRVeL that terminates at the first iteration that I is not robust (its results are denoted by ^{f}). We note that the variant that simply removes the CEGIS step and runs MaRVeL as described (in particular, it continues to run even if it encounters non-robust specifications) obtains very close results to the variant we consider, given a one hour timeout, but it always reaches the timeout. We also report, for each approach's verifier, the average diameter of the maximal ϵ-ball ϵ_u. Table 2 shows the results. The results indicate that MaRVeL's average diameter is larger by 5.2x compared to Liu et al. and by 5x compared to Li et al.. Without the CEGIS step, MaRVeL's average diameter is larger by 3.8x compared to Liu et al. and by 3.7x compared to Li et al.. The average diameter of the maximal ϵ-ball is 3.3x larger for the MILP verifier. MaRVeL's average execution time is 34 minutes, and if it terminates upon encountering the first non-robust specification, it is 8 minutes. Table 3 shows the results for the convolutional networks. The results show that MaRVeL's average diameter is larger by 3.1x than the average diameter of the maximal ϵ-balls, and without CEGIS, it is larger by 1.4x.

Robustness interpretability We next show how our maximal specifications can expose interpretable robustness attributes of the network. We focus on networks for MNIST and Fashion-MNIST, consisting of gray-scale images with a single centered object (a digit or a fashion item). For each network, we run MaRVeL and both baselines on 50 images of different classes. For each approach, given the 50 specifications $\mathcal{I}$, we generate a heatmap image $y^{\mathcal{I}} \in \mathbb{R}^d$. A pixel in $y^{\mathcal{I}}$ is the average diameter of its interval: $y^{\mathcal{I}}_i = \frac{\sum_{I \in \mathcal{I}} u_i - l_i}{|\mathcal{I}|}$. A heatmap shows which pixels are more robust: the brighter the pixel the larger its average interval. Figure 5 shows the

Table 3: A comparison of MaRVeL to uniform ϵ-balls over convolutional networks.

Dataset	Network	MaRVeL				
		$T^f[m]$	ϵ^f_{avg}	$T[m]$	ϵ_{avg}	ϵ_u
MNIST	$Conv_2$	2.7	0.010	51.9	0.041	0.007
F-MNIST	$Conv_2$	4.2	0.032	30.1	0.063	0.016
	$Conv_3$	6.0	0.008	25.0	0.011	0.009
CIFAR-10	$Conv_2$	4.6	0.005	39.2	0.008	0.003
	$Conv_3$	3.45	0.003	32.2	0.005	0.003

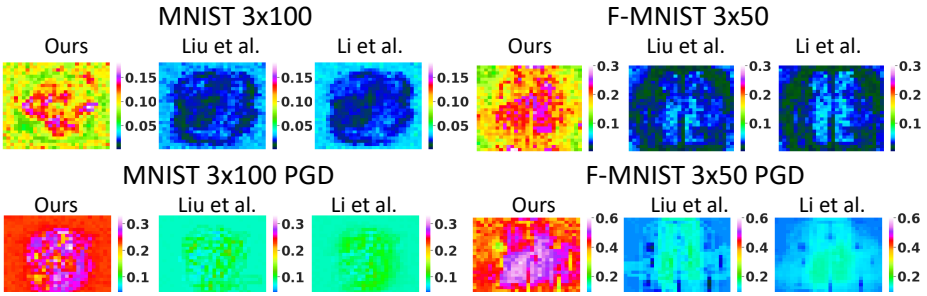

Fig. 5: The heatmaps of the specifications computed by MaRVeL and the baselines.

heatmaps of four networks, the first two are trained without defense and the other two are trained with the PGD defense [27]. The heatmaps demonstrate the following. First, MaRVeL computes larger diameters than the baselines, for all pixels. Second, the baselines' heatmaps suggest that the maximal robust diameters are obtained for the background pixels. In contrast, MaRVeL's heatmaps suggest that some object pixels have the largest diameters. This shows that MaRVeL's maximal specifications provide a better perspective on the network's robustness. Third, all approaches show that, as expected, networks trained with PGD are more robust than their undefended counterparts. However, MaRVeL provides a clearer distinction between the robustness of object pixels and background pixels.

Hyper-parameters Lastly, we discuss the effect of the hyper-parameters: (1) the balancing factor λ_0 and (2) n_m, the number of precise neurons at every layer, affecting the verifier's accuracy. In these experiments, we focus on the 3×50 MNIST network. We begin with the effect of λ_0. Recall that a very small value leads to ignoring the robustness level, while a very large value leads to ignoring the specification size. We consider the values: $\lambda_0 \in \{0.01, 0.25, 0.5, 0.75, 0.99, 2, 10\}$. For each value, we run MaRVeL on 50 images. Figure 6(a) shows the average diameter as a function of λ_0. The figure shows that the largest diameters are obtained for $\lambda_0 \approx 1$. Next, we study the effect of the number of precise neurons n_m. The larger its value, the more accurate the verifier, but the execution time is

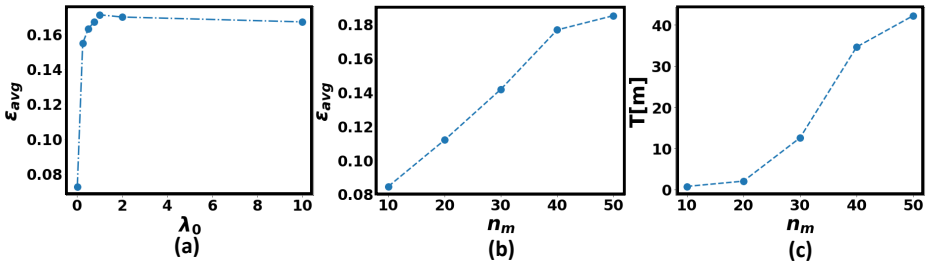

Fig. 6: The effect of the hyper-parameters of MaRVeL.

longer. We consider the values: $n_m \in \{10, 20, 30, 40, 50\}$ (note that for $n_m = 50$ all neurons are precisely encoded). For each value, we run MaRVeL on 50 images. Figure 6(b) and (c) show the average diameter and execution time as a function of n_m. The results show that the higher the value of n_m, the larger the average diameter. Naturally, the higher the value of n_m, the longer the execution time.

7 Related Work

In this section, we discuss the closest related work to ours.

Robustness verifiers and specifications Most existing robustness verifiers focus on ϵ-ball neighborhoods but can be easily extended to analyze interval specifications. These verifiers rely on various techniques, including overapproximation analysis [29,1,41] and in particular overapproximation by linear relaxations [43,3,13,33,31,32,28], simplex [18,19,11], mixed-integer linear programming (MILP) [38,23,34], and duality [9,30]. The closest works to ours present algorithms for computing non-uniform robust neighborhoods [26,25]. These works focus on image classification tasks, where one work computes non-uniform, symmetric robust specifications [26], and the other work computes non-uniform, asymmetric robust specifications [25]. Both works rely on the CROWN verifier [49], employing linear relaxations to the network's computation. Based on CROWN's linear constraints, they define a gradient to search for maximal robust specifications. The gradient computation is integrated as part of CROWN's analysis. In contrast, MaRVeL relies on a MILP verifier, providing a more accurate analysis, and the optimizer and verifier take turns. Another work focuses on non-uniform specifications that are defined by a transformation matrix, leveraging data correlations [12]. It relies on linear relaxations and duality to compute maximal non-uniform robust specifications for malware classification and spam detection tasks. In contrast, MaRVeL focuses on interval specifications. A different line of research computes adversarial regions, i.e., neighborhoods of adversarial inputs [8].

Optimization-guided search MaRVeL and prior works [26,25] rely on numerical optimization to compute maximal non-uniform robust specifications. Many works

rely on optimization to solve other robustness-related tasks. For example, for computing adversarial examples with uniform perturbation budgets [27,4,22,5,6] or non-uniform perturbation budgets [47,12], defending a network by adversarial training [45,27,35], or improving a verifier's precision by looking for spurious adversarial examples added during analysis [1,2].

Program synthesis MaRVeL builds on two common program synthesis techniques. First, it relies on oracle-guided synthesis [17,16], introduced for synthesizing programs by interaction with an oracle (e.g., a solver). Second, it leverages counterexample-guided inductive synthesis (CEGIS) [17,36], where a program synthesizer checks candidates with a solver, which either confirms or provides a counterexample. In the latter case, the synthesizer prunes the search space.

8 Conclusion

We present MaRVeL, an approach and a system for computing maximal non-uniform robust interval specifications, maximizing a target norm. The key idea is to rely on oracle-guided numerical optimization. This allows MaRVeL to rely on an accurate MILP verifier and thereby compute larger specifications than prior works. At each iteration, the optimizer submits a candidate specification to the MILP verifier. The verifier returns the *weakest points*, inputs minimizing the robustness level. Accordingly, the optimizer defines a gradient and computes a new candidate specification. To avoid wasting time on non-robust candidates, MaRVeL relies on CEGIS and restricts bounds that are part of non-robust specifications. We evaluate MaRVeL on several datasets and networks, including fully-connected and convolutional networks. We show it computes specifications with 5.1x larger average diameters compared to prior works, for fully-connected networks, and 2.6x larger average diameters compared to uniform ϵ-balls, computed with an accurate MILP verifier. We further show that CEGIS allows MaRVeL to compute specifications with 1.8x larger average diameters. We demonstrate that MaRVeL's specifications can identify input regions for which networks tend to be more robust or vulnerable as well as compare the robustness of different networks.

References

1. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri., S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In PLDI (2019)
2. Balunovic, M., Vechev, M.T.: Adversarial training and provable defenses: Bridging the gap. In ICLR (2020)
3. Boopathy, A., Weng, T., Chen, P., Liu, S., Dani., L.: Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In AAAI (2019)
4. Carlini, N., Wagner., D.A.: Towards evaluating the robustness of neural networks. In SP (2017)
5. Chen, P., Sharma, Y., Zhang, H., Yi, J., Hsieh., C.: EAD: elastic-net attacks to deep neural networks via adversarial examples. In AAAI (2018)

6. Chen, P., Zhang, H., Sharma, Y., Yi, J., Hsieh., C.: ZOO: zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In AISec Workshop (2017)
7. contagio: contagio 2010. contagio, pdf malware dump. http://contagiodump.blogspot.de/2010/08/malicious-documents-archivefor.html (2010)
8. Dimitrov, D.I., Singh, G., Gehr, T., Vechev, M.T.: Provably robust adversarial examples. In ICLR (2022)
9. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli., P.: A dual approach to scalable verification of deep networks. In UAI (2018)
10. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In ATVA (2017)
11. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In CAV (2020)
12. Erdemir, E., Bickford, J., Melis, L., Aydöre, S.: Adversarial robustness with non-uniform perturbations. In NeurIPS (2021)
13. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev., M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In SP (2018)
14. Goodfellow, I.J., Shlens, J., Szegedy., C.: Explaining and harnessing adversarial examples. In ICLR (2015)
15. Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., Madry., A.: Adversarial examples are not bugs, they are features. In NeurIPS (2019)
16. Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: Oracle-guided component-based program synthesis. In ICSE (2010)
17. Jha, S., Seshia, S.A.: A theory of formal synthesis via inductive learning. In Acta Informatica 54, 693–726 (2017)
18. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer., M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In CAV (2017)
19. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In CAV (2019)
20. Krizhevsky, A.: Learning multiple layers of features from tiny images. In CoRR, abs/1708.07747 (2009)
21. Kurakin, A., Goodfellow, I.J., Bengio., S.: Adversarial examples in the physical world. In ICLR Workshop (2017)
22. Kurakin, A., Goodfellow, I.J., Bengio., S.: Adversarial machine learning at scale. In ICLR (2017)
23. Lazarus, C., Kochenderfer, M.J.: A mixed integer programming approach for verifying properties of binarized neural networks. In IJCAI Workshop (2021)
24. Lecun, Y., Bottou, L., Bengio, Y., Haffner., P.: Gradient-based learning applied to document recognition. In Proceedings of the IEEE 1998;86(11):2278e324 (1998)
25. Li, C., Ji, S., Weng, H., Li, B., Shi, J., Beyah, R., Guo, S., Wang, Z., Wang, T.: Towards certifying the asymmetric robustness for neural networks: Quantification and applications. In TDSC (2021)
26. Liu, C., Tomioka, R., Cevher, V.: On certifying non-uniform bounds against adversarial attacks. In ICML (2019)
27. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu., A.: Towards deep learning models resistant to adversarial attacks. In ICLR (2018)

28. Müller, C., Serre, F., Singh, G., Püschel, M., Vechev, M.: Scaling polyhedral neural network verification on gpus. In MLSys (2021)
29. Qin, C., Dvijotham, K.D., O'Donoghue, B., Bunel, R., Stanforth, R., Gowal, S., Uesato, J., Swirszcz, G., Kohli, P.: Verification of non-linear specifications for neural networks. In ICLR (2019)
30. Raghunathan, A., Steinhardt, J., Liang., P.: Certified defenses against adversarial examples. In ICLR (2018)
31. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang., P.: A convex relaxation barrier to tight robustness verification of neural networks. In NeurIPS (2019)
32. Singh, G., Ganvir, R., Püschel, M., Vechev., M.T.: Beyond the single neuron convex barrier for neural network certification. In NeurIPS (2019)
33. Singh, G., Gehr, T., Püschel, M., Vechev., M.T.: An abstract domain for certifying neural networks. In POPL (2019)
34. Singh, G., Gehr, T., Püschel, M., Vechev., M.T.: Boosting robustness certification of neural networks. In ICLR (2019)
35. Sinha, A., Namkoong, H., Duchi, J.C.: Certifying some distributional robustness with principled adversarial training. In ICLR (2019)
36. Solar-Lezama, A., Tancau, L., Bodík, R., Seshia, S.A., Saraswat, V.A.: Combinatorial sketching for finite programs. In ASPLOS (2006)
37. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus., R.: Intriguing properties of neural networks. In ICLR (2014)
38. Tjeng, V., Xiao, K.Y., Tedrake., R.: Evaluating robustness of neural networks with mixed integer programming. In ICLR (2019)
39. Tu, C., Ting, P., Chen, P., Liu, S., Zhang, H., Yi, J., Hsieh, C., Cheng, S.: Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In AAAI (2019)
40. VirusTotal: Virustotal, a free service that analyzes suspicious files and urls and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware. https://www.virustotal.com/ (2004)
41. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana., S.: Efficient formal safety analysis of neural networks. In NeurIPS (2018)
42. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In USENIX (2018)
43. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In NeurIPS (2021)
44. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. In http://arxiv.org/abs/1708.07747 (2017)
45. Xie, C., Wu, Y., van der Maaten, L., Yuille, A.L., He, K.: Feature denoising for improving adversarial robustness. In CVPR (2019)
46. Yuan, X., He, P., Zhu, Q., Li., X.: Adversarial examples: Attacks and defenses for deep learning. In IEEE Trans. Neural Networks Learn. Syst. pages 2805-2824 (2019)
47. Zeng, H., Zhu, C., Goldstein, T., Huang, F.: Are adversarial examples created equal? A learnable weighted minimax risk for robustness under non-uniform attacks. In AAAI (2021)
48. Zhang, C., Benz, P., Imtiaz, T., Kweon., I.S.: Understanding adversarial examples from the mutual influence of images and perturbations. In CVPR (2020)
49. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In NeurIPS (2018)