A Cascade of Checkers for Run-time Certification of Local Robustness

Ravi Mangal*[1] and Corina Păsăreanu[1,2]

[1] Carnegie Mellon University CyLab, Pittsburgh PA 15213, USA
[2] NASA Ames, Moffett Field CA 94035, USA
`{rmangal,pcorina}@andrew.cmu.edu`

Abstract. Neural networks are known to be susceptible to adversarial examples. Different techniques have been proposed in the literature to address the problem, ranging from adversarial training with robustness guarantees to post-training and run-time certification of local robustness using either inexpensive but incomplete verification or sound, complete, but expensive constraint solving. We advocate for the use of a run-time cascade of over-approximate, under-approximate, and exact local robustness checkers. The exact check in the cascade ensures that no unnecessary alarms are raised, an important requirement for autonomous systems where resorting to fail-safe mechanisms is highly undesirable. Though exact checks are expensive, via two case studies, we demonstrate that the exact check in a cascade is rarely invoked in practice. Code and data are available at `https://github.com/ravimangal/cascade-robustness-impl`.

Keywords: Neural networks · Local robustness · Run-time checks

1 Introduction

Software systems with neural network components are becoming increasingly common due to the new computational capabilities unlocked by neural networks. However, the susceptibility of neural network classifiers to adversarial examples is concerning, particularly for networks used in safety-critical systems. Adversarial examples [30] are inputs produced by applying small, imperceptible modifications to correctly classified inputs such that the modified input is classified incorrectly. This lack of *robustness* of neural networks to small changes in the inputs can not only be exploited by malicious actors [30,4] but also lead to incorrect behavior in the presence of natural noise [12]. Moreover, this phenomenon is widespread - neural networks trained for a variety of tasks like image recognition [30,4], natural language processing [14,1], and speech recognition [5,6,25] have been shown to exhibit the same weakness.

Recognizing the seriousness of the problem, the research community has been actively engaged in studying it. We now know that a neural classifier does not only need to be accurate (i.e., make correct predictions) but it also needs to be

* Corresponding Author

locally robust at all inputs of interest. A network is ϵ-locally robust at an input x if it makes the same prediction at all the inputs that lie within a ball of radius ϵ centered at x (assuming some distance metric defined over the input domain). Local robustness at an input ensures that small ($\leq \epsilon$) modifications to the input do not affect the network prediction.

In practice, ensuring that neural networks are robust has turned out to be a hard challenge. A number of approaches modify the neural network training process to encourage learning robust networks [20,34,26,19]. While such robustness-aware training can greatly reduce the susceptibility of neural networks to adversarial examples due to enhanced robustness, they provide no guarantee that the trained network is free from adversarial examples. For safety-critical applications, the existence of a single adversarial example can have disastrous consequences.

How can we ensure that a network is free from adversarial examples? One approach is to check, using a local robustness certification procedure, if a network is locally robust at every input of interest. This requires assuming that all inputs of interest are known a priori, an assumption that is unlikely to hold in practice. The only option then, to guarantee protection from adversarial examples, is to check at run-time if the network is locally robust at the evaluated input x. If the run-time check passes, we are assured that the input x cannot be adversarial (since even if x is an ϵ-perturbed input produced by an adversary, local robustness at x ensures that the network assigns it the same label as the original unperturbed input). If the check fails, then x is potentially an adversarial input, and one has to resort to some fail-safe mechanism like aborting execution or asking a human expert to make the prediction, both undesirable scenarios to be avoided as far as possible. Though a run-time check introduces a computational overhead, it is the only mechanism for ensuring that a safety-critical system does not misbehave due to adversarial examples.

While the problem of checking if a neural network is locally robust at an input is known to be NP-Complete for ReLU neural networks [15], a number of practical algorithms that variously balance the trade-off between precision of the check and efficiency have been proposed in the literature. Sound but incomplete (or *over-approximate*) algorithms guarantee that the network is locally robust whenever the check passes but not vice versa, i.e., they can report false positives [33,34,26,11,28,10,19]. Sound and complete (or *exact*) algorithms are guaranteed to either find a valid robustness certificate or a valid counterexample but they can be very inefficient [15,31]. Attack (or bug-finding or *under-approximate*) algorithms only aim to find counterexamples to robustness but can fail to find a counterexample even if one exists (and are therefore unable to provide a robustness certificate) [4,20,24,32,2,8].

When deploying local robustness checks at run-time, a common choice is to use over-approximate algorithms because of their computationally efficient nature [19]. But, due to their incompleteness, these algorithms can report false positives and unnecessarily require the use of the undesirable fail-safe mechanisms. On the other hand, while an exact check can avoid unnecessary alarms, these checks involve constraint solving and can be prohibitively expensive. In order to balance between the frequency of unnecessary alarms and the cost of a local robustness check, in this paper, we propose to use a cascade of local robustness checkers at run-time. In particular, we propose a cascade of checks that starts with an over-approximate check, followed by an under-approximate check (which is also computationally cheap), and ends with an exact check. This sequence ensures that if the first check fails, we first attempt to evaluate if the failure was due to a false positive or a true positive. If the under-approximate check (i.e., the attack algorithm) succeeds, then it was indeed a true positive and we are forced to resort to our undesirable fail-safe. However, if the attack also fails, we use the exact check that either returns a certificate of robustness or a counterexample. Notice that the exact check, which is computationally expensive, is invoked only if absolutely necessary. Though a local robustness check that combines an over-approximate algorithm with an exact algorithm has been proposed before [29], our approach differs in multiple ways. Most importantly, while the check from [29] closely integrates the over-approximate and exact algorithms, our approach is entirely agnostic to the internal implementation details of the checks being composed.

We have implemented our run-time cascade of checkers, and empirically evaluate it using two real-world case studies. Our evaluation suggests that, in practice, a cascaded checker can be a reasonable choice since the over-approximate and under-approximate checks are able to resolve most of the local robustness queries, and the expensive, exact check is rarely invoked.

The rest of the paper is organized as follows. In Section 2, we provide the necessary preliminaries and definitions. In Section 3, we briefly describe techniques for robustness-aware training of neural networks as well as the algorithms used for checking local robustness. In Section 4, we give more details about our run-time cascade of local robustness checkers. In Section 5, we present our two case studies and empirically evaluate our run-time cascade of checks. Finally, we conclude in Section 6.

2 Background

We present preliminaries and necessary definitions in this section.

Neural Networks. A neural network, $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^m$, is a function defined by a composition of linear and non-linear transformations, where θ refers to *weights* or parameters characterizing the linear transformations. As the internal details of neural networks are not relevant to much of this paper, we will by default omit the subscript θ, and treat f as a black-box function. Neural networks are used as classifiers by extracting *class predictions* from the output $f(x) : \mathbb{R}^m$, also called the *logits* of a network. Given a neural network f, we use the upper-case F to refer to the corresponding neural classifier that returns the top class:

$F = \lambda x.\, \text{argmax}_i\, f_i(x)$. For our purposes, we will assume that argmax returns a single index, $i^* \in [m]$[1]; ties may be broken arbitrarily

Adversarial Examples. An adversarial example for a neural classifier is the result of applying small modifications to a correctly classified *valid* input such that the modified input is classified incorrectly. Definition 1 below formalizes the notion of an adversarial example.

Definition 1 (Adversarial Example). *Given a neural classifier $F \in \mathbb{R}^d \to [m]$ and an input $x \in \mathbb{R}^d$, an input x' is an adversarial example with respect to an ℓ_p distance metric, and a fixed constant $\epsilon \in \mathbb{R}$ if*

$$||x - x'||_p \leq \epsilon \wedge F(x) \neq F(x')$$

Local Robustness. A classifier is protected from adversarial examples with respect to a valid input x if it is locally robust at x. As stated in Definition 2, a classifier is locally robust at x, if its prediction does not change in an ϵ-ball centered at x.

Definition 2 (Local Robustness). *A neural classifier $F \in \mathbb{R}^d \to [m]$ is (ϵ, ℓ_p)-locally robust at $x \in \mathbb{R}^d$ if,*

$$\forall x' \in \mathbb{R}^d.\ ||x - x'||_p \leq \epsilon \implies F(x') = F(x)$$

Here we consider robustness and input modifications with respect to l_p norms, commonly used in the literature, but our approach extends also to other modifications, which are not necessarily captured with l_p norms.

Certified Run-time Defense Against Adversarial Examples. Before deployment, we can evaluate the susceptibility of a trained neural classifier to adversarial examples by checking its local robustness at inputs in the training and test datasets. However, this provides no guarantee of immunity from adversarial examples on unseen inputs. Checking local robustness of a classifier during run-time can provide such a guarantee. If the classifier is locally robust at the input x under evaluation, then x cannot be an adversarial example. Even if x is an ϵ-perturbed input generated by an adversary, the local robustness certificate ensures that the classifier's prediction is not affected by the perturbation. However, if the local robustness check fails, to be safe, one has to assume that x is potentially an adversarial example that can cause the classifier to misbehave, and resort to using a fail-safe mechanism (like aborting execution or deferring to a human expert) designed to handle this scenario. Defending against adversarial examples at run-time via local robustness checks is a well-known technique [7,19].

[1] $[m] := \{0, \ldots, m-1\}$

3 Ensuring Local Robustness

There are two primary approaches to ensure that neural classifiers are locally robust. One approach is to train the classifiers in a robustness-aware manner. The other approach is to check for local robustness violations by the classifier at run-time. In this section, we provide a brief overview of techniques for robustness-aware training of classifiers as well as algorithms for checking local robustness.

3.1 Local Robustness via Training

Standard training of neural classifiers is typically framed as the following optimization problem:

$$\theta^\star = \operatorname*{argmin}_\theta \sum_{i=1}^{i=n} L(F_\theta(x_i), y_i) \tag{1}$$

where F is a family of neural networks parameterized by $\theta \in \mathbb{R}^p$, $(x_i, y_i) \in \mathbb{R}^d \times [m]$ is a labeled training sample, and L is a real-valued *loss function* that measures how well F_θ "fits" the training data. $F_{\theta^\star}$ is the final trained model. To train models in a robustness-aware manner, the optimization objective is modified in order to promote both accuracy and local robustness of the trained models. A very popular robustness-aware training approach, first proposed by [20], formulates the following min-max optimization problem:

$$\theta^\star = \operatorname*{argmin}_\theta \sum_{i=1}^{i=n} \max_{\delta \in \mathbb{B}(0,\epsilon)} L(F_\theta(x_i + \delta), y_i) \tag{2}$$

where $\mathbb{B}(0, \epsilon)$ is a ball of radius ϵ centered at 0. Intuitively, this optimization objective captures the idea that we want classifiers that perform well even on adversarially perturbed inputs. The inner maximization problem aims to find the worst-case adversarially perturbed version of a given input that maximizes the loss, formalizing the notion of an adversary attacking the neural network. On the other hand, the outer minimization problem aims to find classifier parameters so that the worst-case loss given by the inner optimization problem is minimized, capturing the idea that the trained classifier needs to be immune to adversarial perturbations. Solving the optimization problem in Equation 2 can be very computationally expensive, and most practically successful algorithms only compute approximate solutions to the inner maximization problem. They either compute an under-approximation (lower bound) of the maximum loss [20] or compute an over-approximation (upper bound) of the maximum loss [34,21,26].

An alternate formulation of the optimization objective for robustness-aware training is as follows:

$$\theta^\star = \operatorname*{argmin}_\theta \sum_{i=1}^{i=n} (L(F_\theta(x_i), y_i) + L_{rob}(F_\theta(x_i), \epsilon)) \tag{3}$$

where L_{rob} measures the degree to which classifier F_θ is ϵ-locally robust at x_i. Approaches using an optimization objective of this form [19,36] are required to check local robustness of the neural classifier in order to calculate L_{rob}. As long as the local robustness checking procedure is differentiable, any such procedure can be used. For our case studies (Section 5), we use an approach based on Equation 3, namely GloRo Nets [19], to train our classifiers. To calculate L_{rob}, GloRo uses a sound but incomplete check for local robustness based on calculating the Lipschitz constant of the neural network f.

3.2 Run-time Checks for Local Robustness

Algorithms that check if neural classifiers are locally robust come in three primary flavors:

1. *over-approximate* (or sound but complete) algorithms that guarantee local robustness of the neural network whenever the check passes but not vice versa, i.e., the check can fail even if the network is locally robust,
2. *under-approximate* (or attack) algorithms that generate counterexamples highlighting violations of local robustness but are not always guaranteed to find counterexamples even when they exist,
3. *exact* (sound and complete) algorithms that are guaranteed to either find a certificate of local robustness or a counterexample, but can be very computationally expensive.

Not only are such algorithms useful for checking local robustness at run-time, but they are also useful for evaluating the quality of the trained neural network pre-deployment. In particular, these checkers can be used to evaluate the local robustness of the neural network at the inputs in the training and test datasets. If the trained network lacks local robustness on these known inputs, it is unlikely to be locally robust on unknown inputs. We briefly survey local robustness checking algorithms in the rest of this section.

Over-approximate algorithms. A variety of approaches have been used for implementing over-approximate algorithms for checking local robustness. Algorithms using abstract interpretation [11,28] approximate the ϵ-ball in the input space with a polyhedron enclosing the ball and symbolically propagate the polyhedron through the neural network to get an over-approximation of all the possible outputs of the network when evaluated on points in the input polyhedron. This information can then be used to certify if the network is locally robust or not. Other algorithms frame local robustness certification as an optimization problem [3,9,26,34,31]. A key step in these algorithms is to translate the neural network into optimization constraints. This translation, if done in a semantics-preserving manner, can lead to intractable optimization problems. Instead, the translation constructs relaxed constraints that can be solved efficiently, at the cost of incompleteness. Another approach for over-approximate certification of local robustness relies on computing local or global Lipschitz constant for the neural network under consideration. The Lipschitz constant of a function upper bounds the rate of change of the function output with respect to its input. Given the Lipschitz constant of a neural network and its logit values at a particular input x, one can compute a lower bound of the radius of the ball centered at x within which the network prediction does not change. If this lower bound is greater than ϵ, then we have a certificate of ϵ-local robustness of the network at x. A number of certification algorithms are based on computing the Lipschitz constant [33,19], and we use one such approach [19] in our case studies in Section 5.

Under-approximate algorithms. Under-approximate algorithms, usually referred to as *attacks* in the adversarial machine learning community, can be divided into two major categories. White-box algorithms [13,4,20] assume that they have access to the internals of the neural network, namely the architecture and the weights of the network. Given such access, these algorithms frame the problem of finding counterexamples to local robustness for classifier F at input x as an optimization problem of the form,

$$\delta^\star = \operatorname*{argmax}_{\delta \in \mathbb{B}(0,\epsilon)} L(F(x+\delta), F(x)) \tag{4}$$

The counterexample is given by $x + \delta^\star$. Intuitively, the algorithms try to find a perturbed input $x' := x + \delta^\star$ such that x' is in the ϵ-ball centered at x, and the classifier output at x' is as different from x as possible (formalized by the requirement to maximize the loss L). This optimization problem is non-convex since F can be an arbitrarily complicated function, and in practice, attack algorithms use gradient ascent to solve the optimization problem. Due to the non-convex nature of the optimization objective, such algorithms are not guaranteed to find the optimal solution, and therefore, are not guaranteed to find a counterexample even if one exists. Black-box algorithms [24] are the other category of attack algorithms, and such algorithms only assume query access to the neural network, i.e., the algorithms can only observe the network's outputs on queried inputs, but do not have access to the weights and therefore, cannot directly access the gradients. In other words, black-box algorithms assume a weaker adversary than white-box algorithms. For our case studies, we use a white-box attack algorithm [20].

Exact algorithms. Exact algorithms for checking local robustness [15,16,31] encode a neural network's semantics as system of semantics-preserving constraints, and pose local robustness certification as constraint satisfaction problems. Though these algorithms are guaranteed to either find a certificate of robustness or a counterexample, they can be quite computationally expensive. An alternate approach for exact checking constructs a smoothed classifier from a given neural classifier using a randomized procedure at run-time [18,7,27,35]. Importantly, the smoothed classifier is such that, at each input, its local robustness radius can be calculated exactly using a closed-form expression involving the outputs of the smoothed classifier. However, the randomized smoothing procedure can be very expensive as it requires evaluating the original classifier on a large number of

	Finds certificates?	*Finds counterexamples?*	*Efficient?*
Over-approximate	✓	×	✓
Under-approximate	×	✓	✓
Exact	✓	✓	×

Table 1: Trade-offs made by the different flavors of local robustness checkers.

Algorithm 4.1: Cascaded local robustness checker

Inputs: A neural classifier $F \in \mathbb{R}^d \to [m]$, an input $x \in \mathbb{R}^d$, and local robustness radius $\epsilon \in \mathbb{R}$
Output: Certified status $b \in \{1, 0\}$

```
Cascade($F$ , $x$ , $epsilon$):
    cert := Over-approximate($F, x, \epsilon$)
    if ¬cert then
        cex := Under-approximate($F, x, \epsilon$)
        if ¬cex then
            $b$ := Exact($F, x, \epsilon$)
        else
            $b$ := 0
    else
        $b$ := 1
    return $b$
```

randomly drawn samples. For our case studies, we use an exact algorithm based on constraint-solving.

4 A Cascade of Run-time Local Robustness Checkers

The previous section demonstrates that one has many options when picking a local robustness checker to be deployed at run-time. Every option offers a different trade-off between the ability to certify local robustness and the efficiency of the check. Table 1 summarizes these trade-offs. For each flavor of local robustness checkers, the table shows if the checkers are able to produce certificates of local robustness, find counterexamples, and do so efficiently. Ideally, we would like a checker to possess all these characteristics but the NP-Complete nature of the problem makes this impossible. In light of these trade-offs, a common choice is to deploy an over-approximate checker [19]. Such checkers can falsely report that the neural network is not locally robust, causing unnecessary use of the fail-safe mechanisms.

We propose a local robustness checker that combines existing local robustness checkers of different flavors in a manner that brings together their strengths while mitigating their weaknesses. Our *cascaded* checker uses a sequence of over-approximate, under-approximate, and exact checkers. Algorithm 4.1 describes the cascaded checker. The inputs to the algorithm are the neural classifier F, the input x where we want to certify local robustness of F, and the radius ϵ to be certified. The algorithm first invokes an `Over-approximate` checker (line 2). If F is certified to be locally robust at x (i.e., `cert` equals 1), then we are done. Otherwise, the algorithm calls an `Under-approximate` checker (lines 3-4). If the under-approximate checker succeeds in finding a counterexample (i.e., `cex` equals 1), then we are done and know that F is not locally robust at x. Otherwise, an `Exact` checker is invoked (lines 5-6). The `Exact` checker either finds a proof of robustness ($b = 1$) or a counterexample ($b = 0$).

The cost of Algorithm 4.1, amortized over all the inputs seen at run-time, depends on the rate at with which the `Over-approximate` and `Under-approximate` checks succeed. If the cascaded checker has to frequently invoke the `Exact` checker, then one might as well directly use the `Exact` checker instead of the cascade. In practice, however, our empirical evaluation suggests that the `Exact` is rarely invoked (see Section 5). As a consequence, the cascaded checker is guaranteed to be sound and complete without incurring the high computational cost of an `Exact` checker.

5 Case Studies

The practical effectiveness of our sound and complete cascaded local robustness checker primarily depends on the run-time overhead introduced by the checker. This overhead, in turn, depends on the success rate of the `Over-approximate` and the `Under-approximate` checkers in the cascade. Given that the `Exact` checker is significantly more computationally expensive than the other components of the cascade, it is essential that it only be invoked rarely to ensure that the average overhead of the cascade per input is low.

We conduct two case studies to evaluate the rate at which the `Over-approximate` and `Under-approximate` checkers succeed. In particular, we measure the percentage of test inputs that are resolved by the `Over-approximate`, the `Under-approximate`, and the `Exact` checks. For both the case studies, we are interested in local robustness with respect to the ℓ_2 distance metric. Moreover, we train the neural classifiers in a robustness-aware manner using the state-of-the-art GloRo Net framework [19] that updates the loss function in the manner described in Equation 3, and calculates the Lipschitz constant of the neural network in order to verify local robustness at an input. We also use this local robustness check based on Lipschitz constant as the `Over-approximate` check. For the `Under-approximate` check, we use the projected gradient descent (PGD) algorithm [20], as implemented in the CleverHans framework [23]. Finally, for the `Exact` check, we use the Marabou framework for neural network verification [16]. However, Marabou can only encode linear constraints, and so is unable to encode the ℓ_2 local robustness constraint. Instead, we use Marabou to check local robustness in the largest box contained inside the ℓ_2 ball of radius ϵ, and in the

Total queries	ϵ	*% certified by* `Over-approximate`	*% attacked by* `Under-approximate`	*% resolved by* `Exact`	*% unresolved*
10000	0.3	92.11	7.12	0.76	0.01
	1.58	45.02	49.52	5.28	0.18

Table 2: Percentage of inputs successfully handled by each check for MNIST.

smallest box containing the ϵ-ℓ_2 ball. If Marabou finds a counterexample for the first query, then we have a valid counterexample. Similarly, if Marabou is able to certify the second query, then we have a valid certificate. Though Marabou is no longer guaranteed to be sound and complete when used in the manner described, in our case studies, Marabou rarely fails to resolve the local robustness at an input.

5.1 MNIST

Our first case study uses the popular MNIST dataset [17] where the goal is to construct a neural classifier that can classify hand-written digits. Our neural network has three dense hidden layers with 64, 32, and 16 ReLU neurons in that order. We check ℓ_2 local robustness for an ϵ values of 0.3 and 1.58.

Table 2 shows the success rate of each of the local robustness checkers in our cascade. "Total queries" refers to the number of inputs in the test set used for evaluation. For ϵ value of 0.3, we see that the classifier is certified locally robust at 92.11% of the inputs by the `Over-approximate` check. Of the remaining 7.89% inputs, the `Under-approximate` check is able to find a counterexample for 7.12% of the inputs. As a result, only 0.77% of the 10000 inputs need to be checked with the `Exact` solver. Marabou is able to resolve 0.76% of the inputs, finding a counterexample in each case. Only 0.01% of the inputs, i.e., a single input, is not resolved by any of the checks (due to the fact that for ℓ_2 robustness queries, Marabou is not sound and complete). For ϵ value of 1.58, we see that the classifier is much less robust and the `Over-approximate` check is only able to certify 45.02% of the inputs. For all of the 5.28% inputs resolved by Marabou, it finds a counterexample. These results provide two interesting takeaways. First, the `Exact` checker is rarely invoked, suggesting that a cascaded checker is a reasonable choice in practice. Second, an `Exact` checker like Marabou is not only useful for finding certificates but also counterexamples.

5.2 SafeSCAD

Our second case study uses datasets produced as a part of the SafeSCAD [2] project. The project is concerned with the development of a driver attentiveness management system to support safe shared control of autonomous vehicles.

[2] Safety of Shared Control in Autonomous Driving

Total queries	ϵ	*% certified by* `Over-approximate`	*% attacked by* `Under-approximate`	*% resolved by* `Exact`	*% unresolved*
11819	0.05	54.36	35.77	8.49	1.38
	0.15	42.19	45.68	10.9	1.23

Table 3: Percentage of inputs successfully handled by each check for SafeSCAD.

Shared-control autonomous vehicles are designed to operate autonomously but can request the driver to take over control if the vehicle enters conditions that the autonomous system cannot handle. The goal of the driver attentiveness management system then is to ensure that drivers are alert enough to take over control whenever requested. This system uses a neural network for predicting the driver alertness levels based on inputs from specialized sensors that monitor key vehicle parameters (velocity, lane position, etc.) and driver's biometrics (eye movement, heart rate, etc.). We used driver data collected as part of a SafeSCAD user study carried out within a driving simulator [22]. Our neural network has four dense hidden layers with 50, 100, 35, and 11 ReLU neurons in that order. We check ℓ_2 local robustness for ϵ values of 0.05 and 0.15.

Table 3 shows the success rate of each of the local robustness checkers in our cascade. We see that the trained classifier is not as robust as the MNIST case, and, for ϵ value of 0.05, it is certified locally robust only at 54.36% of the inputs by the `Over-approximate` check. Of the remaining 45.64% inputs, the `Under-approximate` check is able to find a counterexample for 35.77% of the inputs. 9.87% of the 11819 inputs need to be checked with the `Exact` solver. Marabou is able to resolve 8.49% of the inputs, finding a counterexample for 8.47% of the inputs and finding a proof of robustness for 0.02% of the inputs. 1.38% of the inputs not resolved by any of the checks (due to the fact that for ℓ_2 robustness queries, Marabou is not sound and complete). The results for ϵ value of 0.15 can be read off from the table in a similar manner. Note that Marabou finds a counterexample for all of the 10.9% of the inputs resolved by it. These results largely mirror the findings from the MNIST case study. In particular, they show that even when the neural classifier trained in a robustness-aware manner is not locally robust on a large percentage of the test inputs, the `Over-approximate` and `Under-approximate` checkers are able to resolve most of the inputs, and the `Exact` solver is rarely invoked.

6 Conclusion

In this paper, we surveyed techniques for checking local robustness on neural networks and we advocated for a cascade of checkers that best leverages their strengths and mitigates their weaknesses. We demonstrated the cascade of checkers with two case studies. Our experiments demonstrate that the most expensive check (which involves formal methods) is seldom needed as the previous checkers in the cascade are often sufficient for providing a robustness guarantee or for finding a counterexample. Nevertheless, the expensive, formal methods check is still important when dealing with autonomous, safety-critical systems as it can help avoid unnecessarily resorting to the fail-safe mechanism. Furthermore, we show that the formal methods check is useful for not only providing a certificate but also for producing counterexamples which are hard to find with cheaper techniques. Future work involves experimenting with more case studies and applying cascades to reasoning about more natural perturbations that are not necessarily captured with l_p-bounded modifications.

Acknowledgments. This material is based upon work supported by DARPA GARD Contract HR00112020006 and UK's Assuring Autonomy International Programme.

References

1. Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.J., Srivastava, M., Chang, K.W.: Generating Natural Language Adversarial Examples. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. pp. 2890–2896. Association for Computational Linguistics, Brussels, Belgium (Oct 2018)
2. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In: International conference on machine learning. pp. 274–283. PMLR (2018)
3. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A.V., Criminisi, A.: Measuring neural net robustness with constraints. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. p. 2621–2629. NIPS'16, Curran Associates Inc., Red Hook, NY, USA (2016)
4. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 39–57. IEEE Computer Society, Los Alamitos, CA, USA (may 2017). `https://doi.org/10.1109/SP.2017.49`, `https://doi.ieeecomputersociety.org/10.1109/SP.2017.49`
5. Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., Zhou, W.: Hidden Voice Commands. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 513–530 (2016), `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/carlini`
6. Carlini, N., Wagner, D.: Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 1–7 (May 2018)
7. Cohen, J., Rosenfeld, E., Kolter, Z.: Certified adversarial robustness via randomized smoothing. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 1310–1320. PMLR (09–15 Jun 2019), `https://proceedings.mlr.press/v97/cohen19c.html`
8. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International conference on machine learning. pp. 2206–2216. PMLR (2020)
9. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., Kohli, P.: A dual approach to scalable verification of deep networks. In: Proceedings of the Thirty-Fourth

Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18). pp. 162–171. AUAI Press, Corvallis, Oregon (2018)
10. Fromherz, A., Leino, K., Fredrikson, M., Parno, B., Păsăreanu, C.: Fast geometric projections for local robustness certification. In: International Conference on Learning Representations (ICLR) (2021)
11. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18 (2018)
12. Gilmer, J., Ford, N., Carlini, N., Cubuk, E.: Adversarial examples are a natural consequence of test error in noise. In: International Conference on Machine Learning. pp. 2280–2289. PMLR (2019)
13. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6572
14. Jia, R., Liang, P.: Adversarial Examples for Evaluating Reading Comprehension Systems. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 2021–2031. Association for Computational Linguistics, Copenhagen, Denmark (Sep 2017)
15. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
16. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 443–452. Springer International Publishing, Cham (2019)
17. LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database (2010)
18. Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., Jana, S.: Certified robustness to adversarial examples with differential privacy. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 656–672. IEEE (2019)
19. Leino, K., Wang, Z., Fredrikson, M.: Globally-robust neural networks. In: International Conference on Machine Learning (ICML) (2021)
20. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018)
21. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: International Conference on Machine Learning. pp. 3578–3586. PMLR (2018)
22. Pakdamanian, E., Sheng, S., Baee, S., Heo, S., Kraus, S., Feng, L.: Deeptake: Prediction of driver takeover behavior using multimodal data. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. CHI '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3411764.3445563, https://doi.org/10.1145/3411764.3445563
23. Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A., Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.L., Li, Z., Sheatsley, R., Garg, A., Uesato, J., Gierke, W., Dong, Y., Berthelot, D., Hendricks, P., Rauber, J., Long, R.: Technical report on the cleverhans v2.1.0 adversarial examples library. arXiv preprint arXiv:1610.00768 (2018)

24. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia conference on computer and communications security. pp. 506–519 (2017)
25. Qin, Y., Carlini, N., Cottrell, G., Goodfellow, I., Raffel, C.: Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition. In: International Conference on Machine Learning. pp. 5231–5240 (May 2019), http://proceedings.mlr.press/v97/qin19a.html
26. Raghunathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=Bys4ob-Rb
27. Salman, H., Yang, G., Li, J., Zhang, P., Zhang, H., Razenshteyn, I., Bubeck, S.: Provably robust deep learning via adversarially trained smoothed classifiers. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. pp. 11292–11303 (2019)
28. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL) (Jan 2019)
29. Singh, G., Gehr, T., Püschel, M., Vechev, M.: Robustness certification with refinement. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=HJgeEh09KQ
30. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014), http://arxiv.org/abs/1312.6199
31. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=HyGIdiRqtm
32. Tramer, F., Carlini, N., Brendel, W., Madry, A.: On adaptive attacks to adversarial example defenses. Advances in Neural Information Processing Systems **33** (2020)
33. Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning. pp. 5276–5285. PMLR (2018)
34. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International Conference on Machine Learning. pp. 5286–5295. PMLR (2018)
35. Yang, G., Duan, T., Hu, J.E., Salman, H., Razenshteyn, I., Li, J.: Randomized smoothing of all shapes and sizes. In: International Conference on Machine Learning. pp. 10693–10705. PMLR (2020)
36. Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., Jordan, M.: Theoretically principled trade-off between robustness and accuracy. In: International conference on machine learning. pp. 7472–7482. PMLR (2019)