Robustness Verification for Classifier Ensembles

Dennis Gross[1], Nils Jansen[1], Guillermo A. Pérez[2(✉)], and Stephan Raaijmakers[3]

[1] Radboud University Nijmegen, Nijmegen, The Netherlands
[2] University of Antwerp, Antwerp, Belgium
guillermoalberto.perez@uantwerpen.be
[3] TNO and Leiden University, Leiden, The Netherlands

Abstract. We give a formal verification procedure that decides whether a classifier ensemble is robust against arbitrary randomized attacks. Such attacks consist of a set of deterministic attacks and a distribution over this set. The robustness-checking problem consists of assessing, given a set of classifiers and a labelled data set, whether there exists a randomized attack that induces a certain expected loss against all classifiers. We show the NP-hardness of the problem and provide an upper bound on the number of attacks that is sufficient to form an optimal randomized attack. These results provide an effective way to reason about the robustness of a classifier ensemble. We provide SMT and MILP encodings to compute optimal randomized attacks or prove that there is no attack inducing a certain expected loss. In the latter case, the classifier ensemble is provably robust. Our prototype implementation verifies multiple neural-network ensembles trained for image-classification tasks. The experimental results using the MILP encoding are promising both in terms of scalability and the general applicability of our verification procedure.

Keywords: Adversarial attacks · Ensemble classifiers · Robustness

1 Introduction

Recent years have seen a rapid progress in *machine learning* (ML) with a strong impact to fields like autonomous systems, computer vision, or robotics. As a consequence, many systems employing ML show an increasing interaction with aspects of our everyday life, consider autonomous cars operating amongst pedestrians and bicycles. While studies indicate that self-driving cars, inherently relying on ML techniques, make around 80% fewer traffic mistakes than human drivers [19], verifiable *safety* remains a major open challenge [5,13,23,26].

Research funded by the project NWA.BIGDATA.2019.002: "EXoDuS - EXplainable Data Science" and the FWO G030020N project "SAILor".

D. V. Hung and O. Sokolsky (Eds.): ATVA 2020, LNCS 12302, pp. 271–287, 2020.
https://doi.org/10.1007/978-3-030-59152-6_15

In the context of self-driving cars, for instance, certain camera data may contain noise that can be introduced randomly or actively via so-called adversarial attacks. We focus on the particular problem of such attacks in image classification. A successful attack perturbs the original image in a way such that a human does not recognize any difference while ML *misclassifies* the image. A measure of difference between the ground truth classification, for instance by a human, and a potentially perturbed ML classifier is referred as the *loss*.

A standard way to render image classification more robust against adversarial attacks is to employ a set of classifiers, also referred to as *classifier ensembles* [2,3,20,21]. The underlying idea is to obscure the actual classifier from the attacker. One possible formalization of the competition between an adversarial attacker and the ensemble is that of a zero-sum game [20]: The attacker chooses first, the ensemble tries to react to the attack with minimal loss — that is, choosing a classifier that induces maximal classification accuracy.

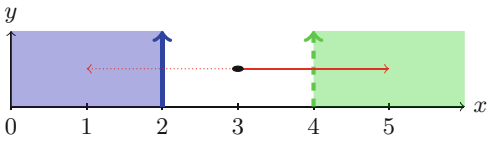

Fig. 1. We depict a single data point in $\mathbb{R}^2$ with label 1. The region to the left of the solid vertical line corresponds to points labelled with 2 by one classifier; the region to the right of the dashed line, those labelled with 2 by another classifier. Both (linear) classifiers label all other points in $\mathbb{R}^2$ with 1. (Hence, they correctly label the data point with 1.) The dotted attack, moving the data point left, induces a misclassification of the point by one of the classifiers. The solid attack, moving the data point right, induces a misclassification of the point by one of the classifiers. Note that every attack has a classifier which is "robust" to it, i.e. it does not misclassify the perturbed point. However, if the attacker chooses an attack uniformly at random, both of them misclassify the point with probability $1/2$.

In this setting, the attacker may need to use randomization to behave optimally (see Fig. 1, cf. [6]). Such an attack is called optimal if the *expected loss* is maximized regardless of the choice of classifier.

Inspired by previous approaches for single classifiers [12,15], we develop a formal verification procedure that decides if a classifier ensemble is *robust* against any randomized attack. In particular, the formal problem is the following. Given a set of classifiers and a labelled data set, we want to find a probability distribution and a set of attacks that induce an optimal randomized attack. Akin to the setting in [20], one can provide thresholds on potential perturbations of data points and the minimum shift in classification values. Thereby, it may happen that no optimal attack exists, in which case we call the classifier ensemble *robust*. Our aim is the development of a principled and effective method that is able to either find the optimal attack or prove that the ensemble is robust with respect to the predefined thresholds.

To that end, we first establish a number of theoretical results. First, we show that the underlying formal problem is **NP**-hard. Towards computational tractability, we also show that for an optimal attack there exists an upper bound on the number of attacks that are needed. Using these results, we provide an SMT encoding that computes suitable randomized attacks for a set of convolutional neural networks with ReLU activation functions and a labelled data set. In case there is no solution to that problem, the set of neural networks forms a robust classifier ensemble, see Fig. 2. Together with the state-of-the-art SMT solver Z3 [9], this encoding provides a complete method to solve the problem at hand. Yet, our experiments reveal that it scales only for small examples. We outline the necessary steps to formulate the problem as a mixed-integer linear programming (MILP), enabling the use of efficient optimization solvers like Gurobi [14].

Fig. 2. The verifier takes as input a set of classifiers, a set of labelled data points, the number of attacks, and the attack properties. If the verifier does not find a solution, we can be sure is robust against any attack with the specific properties. Otherwise, it returns the optimal attack.

In our experiments, we show the applicability of our approach by means of a benchmark set of binary classifiers, which were trained on the MNIST and German traffic sign datasets [10,25].

Related Work

It is noteworthy that there is some recent work on robustness checking of decision-tree ensembles [22]. However, their approach is based on abstract interpretation and is thus not complete. Other approaches for robustness checking of machine learning classifiers focus on single classifiers (see, e.g., [7,12,15,18,24]). Akin to our approach, some of these works employ SMT solving. In [4], MILP-solving is used for verification tasks on (single) recurrent neural networks. In contrast, our framework allows to compute attacks for classifier ensembles.

In [11], Dreossi et al. propose a robustness framework which unifies the optimization and verification views on the robustness-checking problem and encompasses several existing approaches. They explicitly mention that their framework applies to *local robustness* and argue most of the existing work on finding adversarial examples and verifying robustness fits their framework. Our work, when we have a single classifier and a singleton data set, fits precisely into their framework. However, we generalize in those two dimensions by averaging over

the *robustness target value* (in their jargon) for all points in a data set, and by considering ensemble classifiers. This means that our point of view of the *adversarial environment* is neither that of a white-box attacker nor is it a black-box attacker. Indeed, we know the set of classifiers but we do not know what strategy is used to choose which (convex combination of) classifiers to apply. Our environment is thus a gray-box attacker.

2 Preliminaries

Let $\mathbf{x}$ be a vector $(x_1, \ldots, x_d) \in \mathbb{R}^d$. We write $\|\mathbf{x}\|_1$ for the "Manhattan norm" of $\mathbf{x}$, that is $\sum_{i=1}^{d} |\mathbf{x}_i|$.

We will make use of a partial inverse of the max function. Consider a totally ordered set Y and a function $f\colon X \to Y$. Throughout this work we define the arg max (arguments of the maxima) partial function as follows. For all $S \subseteq X$ we set $\arg\max_{s \in S} f(s) := m$ if m is the *unique* element of S such that $f(m) = \max_{s \in S} f(s)$. If more than one element of S witnesses the maximum then $\arg\max_{s \in S} f(s)$ is *undefined*.

A *probability distribution* over a finite set D is a function $\mu\colon D \to [0, 1] \subseteq \mathbb{R}$ with $\sum_{x \in D} \mu(x) = 1$. The set of all distributions on D is $Distr(D)$.

2.1 Neural Networks

We loosely follow the neural-network notation from [12,15]. A *feed-forward neural network* (NN for short) with d inputs and ℓ outputs encodes a function $f\colon \mathbb{R}^d \to \mathbb{R}^\ell$. We focus on NNs with *ReLU* activation functions. Formally, the function f is given in the form of

- a sequence $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(k)}$ of *weight matrices* with $\mathbf{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$, for all $i = 1, \ldots, k$, and
- a sequence $\mathbf{B}^{(1)}, \ldots, \mathbf{B}^{(k)}$ of *bias vectors* with $\mathbf{B}^{(i)} \in \mathbb{R}^{d_i}$, for all $i = 1, \ldots, k$.

Additionally, we have that $d_0, \ldots, d_k \in \mathbb{N}$ with $d_0 = d$ and $d_k = \ell$. We then set $f = g^{(k)}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$ where for all $i = 1, \ldots, k$ we define

$$g^{(i)}(\mathbf{x}) := \mathrm{ReLU}(\mathbf{W}^{(i)} g^{(i-1)}(\mathbf{x}) + \mathbf{B}^{(i)}),$$

and $g^{(0)}(\mathbf{x}) := \mathbf{x}$. The ReLU function on vectors $\mathbf{u}$ is the element-wise maximum between 0 and the vector entries, that is, if $\mathbf{v} = \mathrm{ReLU}(\mathbf{u})$ then $\mathbf{v}_i = \max(0, \mathbf{u}_i)$.

We sometimes refer to each $g^{(i)}$ as a *layer*. Note that each layer is fully determined by its corresponding weight and bias matrices.

2.2 Neural-Network Classifiers

A *data set* $X \subseteq \mathbb{R}^d$ is a finite set of (real-valued) data points $\mathbf{x} \in \mathbb{R}^d$ of dimension $d \in \mathbb{N}_{>0}$. A *classifier* $c\colon X \to [\ell]$ is a partial function that attaches to each data point a label from $[\ell] = \{1, \ldots, \ell\}$, the *set of labels*. We denote the set of all classifiers over X by $\mathbb{C}$. An NN-encoded classifier is simply a partial function $f\colon \mathbb{R}^d \to \mathbb{R}^k$ given as an NN that assigns to each data point $\mathbf{x} \in \mathbb{R}^d$ the label $\arg\max_{i \in [\ell]} h(i)$ where $h(i) = f(\mathbf{x})_i$. Intuitively, the label is the index of the largest entry in the vector resulting from applying f to $\mathbf{x}$. Note that if the image of x according to f has several maximal entries then the arg max and the output label are undefined.

Definition 1 (Labelled data set). *A* labelled data set $\mathcal{X} = (X, c_t)$ *consists of a data set* X *and a total classifier* c_t *for* X*, i.e.* c_t *is a total function.*

In particular, $c_t(\mathbf{x})$ is defined for all $\mathbf{x} \in X$ and considered to be the "ground truth" classification for the whole data set X.

3 Problem Statement

We state the formal problem and provide the required notation. Recall that in our setting we assume to have an ensemble of classifiers $C \subseteq \mathbb{C}$. Such an ensemble is attacked by a set of attacks that are selected randomly.

Definition 2 (Deterministic attack). *A* deterministic attack *for a labelled data set* (X, c_t) *and a classifier* $c\colon X \to [\ell]$ *is a function* $\delta\colon X \to \mathbb{R}^d$*. An attack* δ *induces a* misclassification *for* $\mathbf{x} \in X$ *and* c *if* $c(\mathbf{x} + \delta(\mathbf{x})) \neq c_t(\mathbf{x})$ *or if* $c(\mathbf{x} + \delta(\mathbf{x}))$ *is undefined. The set of all deterministic attacks is* Δ*. An attack is* ε*-bounded if* $\|\delta(\mathbf{x})\|_1 \leq \varepsilon$ *holds for all* $\mathbf{x} \in X$*.*

We sometimes call the value $\mathbf{x} + \delta(\mathbf{x})$ the *attack point*. Note that the classifier c is not perfect, that is, $c(\mathbf{x}) \neq c_t(\mathbf{x})$ for some $\mathbf{x} \in X$, already a zero-attack $\delta(\mathbf{x}) = 0$ leads to a misclassification.

We extend deterministic attacks by means of probabilities.

Definition 3 (Randomized attack). *A finite set* $A \subseteq \Delta$ *of deterministic attacks together with a probability distribution* $\mathbb{P} \in \mathit{Distr}(A)$ *is a* randomized attack $(A, \mathbb{P})$*. A randomized attack is* ε-bounded *if for all attacks* $\delta \in A$ *with* $\mathbb{P}(\delta) > 0$ *it holds that* $\|\delta(\mathbf{x})\|_1 \leq \varepsilon$ *for all* $\mathbf{x} \in X$*.*

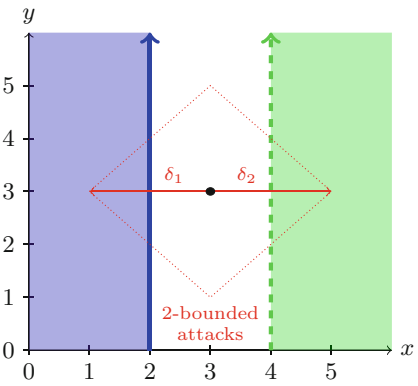

Fig. 3. The dotted diamond contains the set of all 2-bounded attacks in the setting described in Fig. 1. Both δ_1 and δ_2 are therefore 2-bounded (deterministic) attacks. Hence, any randomized attack with $A = \{\delta_1, \delta_2\}$ is also 2-bounded.

In general, a *loss function* $\ell\colon \mathbb{C} \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ describes the *penalty* incurred by a classifier with respect to a labelled data point and an attack. In this work, we will focus on the widely used zero-one loss.

Definition 4 (Zero-one loss function). *The* (0–1)-loss function $\ell_{0-1}\colon \mathbb{C} \times \mathbb{R}^d \times \mathbb{R}^d \to \{0,1\}$ *for a labelled data set* (X, c_t)*, a classifier* $c\colon X \to [\ell]$*, and a deterministic attack* $\delta \in A$ *is given by the following for all* $\mathbf{x} \in X$

$$\ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x})) = \begin{cases} 0 & \text{if } c(\mathbf{x} + \delta(\mathbf{x})) = c_t(\mathbf{x}) \\ 1 & \text{otherwise.} \end{cases}$$

In particular, the loss function yields one if the image of the classifier c is undefined for the attack point $\mathbf{x} + \delta(\mathbf{x})$. Note furthermore that the loss is measured with respect to the ground truth classifier c_t. Thereby, the classifier c and the zero function as deterministic attack do not necessarily induce a loss of zero with respect to c_t. This assumption is realistic as, while we expect classifiers to perform well with regard to the ground truth, we cannot assume perfect classification in realistic settings.

We now connect a randomized attack to an ensemble, that is, a finite set $C \subseteq \mathbb{C}$ of classifiers. In particular, we quantify the overall value a randomized attack induces with respect to the loss function and the ensemble.

Definition 5 (Misclassification value). *The* misclassification value *of a randomized attack* $(A, \mathbb{P})$ *with respect to a labelled data set* $\mathcal{X} = (X, c_t)$ *and a finite set of classifiers* $C \subseteq \mathbb{C}$ *is given by*

$$\mathbf{Val}(A, \mathbb{P}) := \min_{c \in C} \frac{1}{|X|} \sum_{\mathbf{x} \in X} \mathop{\mathbb{E}}_{\delta \sim \mathbb{P}} [\ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x}))]. \tag{1}$$

This value is the minimum (over all classifiers) mean expected loss with respect to the randomized attack and the classifiers from C. An *optimal adversarial*

attack against $C \subseteq \mathbb{C}$ with respect to a labelled data set (X, c_t) is a randomized attack which maximizes the value $\mathbf{Val}(A, \mathbb{P})$ in Eq. (1).

We are now ready to formalize a notion of robustness in terms of ε-bounded attacks and a *robustness bound* $\alpha \in \mathbb{R}$, as proposed in [20] for a set of classifiers.

Definition 6 (Bounded robustness). *A set of classifiers $C \in \mathbb{C}$ for a labelled data set (X, c_t) is called* robust bounded by ε and α *(ε, α-robust) if it holds that*

$$\forall (A, \mathbb{P}) \in 2^{\Delta} \times \mathit{Distr}(A).\, \mathbf{Val}(A, \mathbb{P}) < \alpha, \tag{2}$$

where the $(A, \mathbb{P})$ range over all ε-bounded randomized attacks.

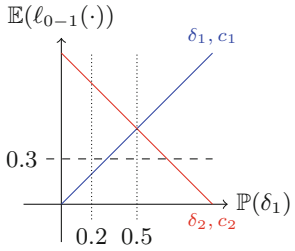

Fig. 4. Continuing with the example from Figs. 1 and 3, we now plot the expected loss per attack and the corresponding classifier. (That is, the classifier which misclassifies the perturbed point.) On the horizontal axis we have the probability x assigned to δ_1 and we assume $\mathbb{P}(\delta_2) = 1-x$. Note that $x = 0.2$ is such that the minimal expected loss, i.e. the misclassification value, is strictly less than 0.3. Indeed, one classifier manages to correctly classifier the perturbed point with probability 0.8 in this case. With $x = 0.5$ we see that the misclassification value is 0.5. Hence, the ensemble is not $(2, 0.5)$-robust.

In other words, an (ε, α)-robust ensemble is such that for all possible ε-bounded random attacks $(A, \mathbb{P})$, there is at least one classifier $c \in C$ from the ensemble such that $\sum_{\mathbf{x} \in X} \mathbb{E}_{\delta \sim \mathbb{P}}[\ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x}))] < \alpha |X|$. Conversely, an ensemble is not (ε, α)-robust if there is an ε-bounded randomized attack with a misclassification value of at least α.

4 Theoretical Results

In this section we establish two key results that carry essential practical implication for our setting. First, we show that in order to obtain an optimal randomized attack, only a bounded number of deterministic attacks is needed.[1] Thereby, we

[1] An analogue of this property had already been observed by Perdomo and Singer in [20, Section 3] in the case when classifiers are chosen randomly.

only need to take a bounded number of potential attacks into account in order to prove the α-robustness of a set of classifiers and a given labelled data set. Second, we establish that our problem is in fact **NP**-hard, justifying the use of SMT and MILP solvers to (1) compute any optimal randomized attack and, more importantly, to (2) prove robustness against any such attack.

4.1 Bounding the Number of Attacks

In the following, we assume a fixed labelled data set (X, c_t). For every classifier $c \in C$ and every deterministic attack $\delta \in \Delta$, let us write $M_c(\delta)$ to denote the value $\sum_{x \in X} \ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x}))$. Observe that $0 \leq M_c(\delta) \leq |X|$ for all $c \in C$ and $\delta \in \Delta$. Furthermore, for all $c \in C$ and randomized attacks $(A, \mathbb{P})$ it holds that:

$$\sum_{\mathbf{x} \in X} \mathop{\mathbb{E}}_{\delta \sim \mathbb{P}}[\ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x}))] = \sum_{\mathbf{x} \in X} \sum_{\delta \in A} \mathbb{P}(\delta) \cdot \ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x}))$$

$$= \sum_{\delta \in A} \mathbb{P}(\delta) \underbrace{\left(\sum_{\mathbf{x} \in X} \ell_{0-1}(c, \mathbf{x}, \delta(\mathbf{x})) \right)}_{= M_c(\delta)} = \sum_{\delta \in A} \mathbb{P}(\delta) \cdot M_c(\delta)$$

We get that Eq. (2) from Definition 5 is false if and only if the following holds.

$$\exists (A, \mathbb{P}) \in 2^{\Delta} \times \mathit{Distr}(A). \forall c \in C. \sum_{\delta \in A} \mathbb{P}(\delta) \cdot M_c(\delta) \geq \alpha |X| \tag{3}$$

Proposition 1 (Bounded number of attacks). *Let $\alpha \in \mathbb{R}$ and consider the labelled data set (X, c_t) together with a finite set of classifiers $C \subseteq \mathbb{C}$. For all randomized attacks $(A, \mathbb{P})$, there exists a randomized attack $(A', \mathbb{P}')$ such that*

- $|A'| \leq (|X| + 1)^{|C|}$,
- $\mathbf{Val}(A', \mathbb{P}') = \mathbf{Val}(A, \mathbb{P})$, *and*
- $(A', \mathbb{P}')$ *is ε-bounded if $(A, \mathbb{P})$ is ε-bounded.*

Proof. We proceed by contradiction. Let $(A, \mathbb{P})$ be an ε-bounded randomized attack with a misclassification value of α such that $|A| > (|X| + 1)^{|C|}$. Further suppose that $(A, \mathbb{P})$ is minimal (with respect to the size of A) amongst all such randomized attacks. It follows that there are attacks $\delta, \delta' \in A$ such that $M_c(\delta) = M_c(\delta')$ for all $c \in C$. We thus have that

$$\mathbb{P}(\delta) \cdot M_c(\delta) + \mathbb{P}(\delta') \cdot M_c(\delta') = (\mathbb{P}(\delta) + \mathbb{P}(\delta')) \cdot M_c(\delta).$$

Consider now the randomized attack $(A', \mathbb{P}')$ obtained by modifying $(A, \mathbb{P})$ so that $\mathbb{P}(\delta) = \mathbb{P}(\delta) + \mathbb{P}(\delta')$ and δ' is removed from A. From the above discussion and Eq. (3) it follows that $(A', \mathbb{P}')$, just like $(A, \mathbb{P})$, has a misclassification value of α. Furthermore, since $A' \subseteq A$, we have that $(A', \mathbb{P}')$ is ε-bounded and that $|A'| < |A|$. This contradicts our assumption regarding the minimality of $(A, \mathbb{P})$ amongst ε-bounded randomized attacks with the same value α. □

4.2 NP Hardness of Non-robustness Checking

It is known that checking whether linear properties hold for a given NN with ReLU activation functions is **NP**-hard [15]. We restate this using our notation.

Proposition 2 (From)[15]**, Appendix I).** *The following problem is* **NP***-hard: Given an NN-encoded function* $f\colon \mathbb{R}^n \to \mathbb{R}^m$ *and closed nonnegative intervals* $(I_k)_1^n, (O_\ell)_1^m$*, decide whether there exists* $\mathbf{x} \in \prod_{k=}^{n} I_k$ *such that* $f(\mathbf{x}) \in \prod_{\ell=1}^{m} O_\ell$*.*

Intuitively, determining whether there exists a point in a given *box* — that is, a hypercube defined by a Cartesian product of intervals — from $\mathbb{R}^n$ whose image according to f is in a given box from $\mathbb{R}^m$ is **NP**-hard. We will now reduce this to the problem of determining if there is a randomized attack such that its misclassification value takes at least a given threshold.

Theorem 1. *The following problem is* **NP***-hard: For a labelled data set* $\mathcal{X} = (X, c_t)$*, a set* C *of classifiers, and a value* $\alpha \in \mathbb{Q}$*, decide if there exists an* ε*-bounded randomized attack* $(A, \mathbb{P})$ *w.r.t.* $\mathcal{X}$ *and* C *such that* $\mathbf{Val}(A, \mathbb{P}) \geq \alpha$*.*

Proof. We use Proposition 2 and show how to construct, for any NN-encoded function $g\colon \mathbb{R}^n \to \mathbb{R}$ and any constraint $\ell \leq g(\mathbf{x}) \leq u$, two classifiers c_ℓ, c_u such that a single deterministic attack δ causes $\mathbf{0}$, the single data point, to be misclassified by both c_ℓ and c_u if and only if the constraint holds. Note that the ε bound can be chosen to be large enough so that it contains the box $\prod_{k=1}^{n} I_k$ and that the identity function over nonnegative numbers is NN-encodable, that is, using the identity matrix as weight matrix $\mathbf{W}$ and a zero bias vector $\mathbf{B}$. For every instance of the problem from Proposition 2 we can therefore construct $2(n+m)$ NNs that encodes all input and output constraints: $2n$ of them based on the identity function to encode input constraints and $2m$ based on the input NN from the given instance. It follows that determining if there exists an ε-bounded deterministic attack δ that causes $\mathbf{0}$ to be *simultaneously misclassified* by a given classifier ensemble is **NP**-hard. Hence, to conclude, it suffices to argue that the latter problem reduces to our robustness-value threshold problem. The result follows from Lemmas 1 and 2. □

Enforcing Interval Constraints. Let $g\colon \mathbb{R}^n \to \mathbb{R}$ be an NN-encoded function and $\ell, u \in \mathbb{R}$ with $\ell \leq u$. Consider now the constraint $\ell \leq g(\mathbf{x}) \leq u$. Henceforth we will focus on the labelled data set (X, c_t) with $X = \{\mathbf{0}\}$ and $c_t(\mathbf{x}) = 1$.

Lower-Bound Constraint. We obtain c_ℓ by adding to the NN encoding of g a new final layer with weight and bias vectors

$$\mathbf{W} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} \ell \\ 0 \end{pmatrix}$$

to obtain the NN-encoded function $\underline{g}\colon \mathbb{R}^n \to \mathbb{R}^2$. Note that $\underline{g}(\mathbf{v}) = (\ell, g(\mathbf{v}))^\intercal$ for all $\mathbf{v} \in \mathbb{R}^n$. It follows that $c_\ell(\mathbf{v}) = 2$ if $g(\mathbf{v}) > \ell$ and $c_\ell(\mathbf{v})$ is undefined if $g(\mathbf{v}) = \ell$. In all other cases the classifier yields 1.

Upper-Bound Constraint. To obtain c_u we add to the NN encoding of g two new layers. The corresponding weight matrices and bias vectors are as follows.

$$\mathbf{W}^{(1)} = (1),\ \mathbf{B}^{(1)} = (-u),\ \mathbf{W}^{(2)} = \begin{pmatrix} 0 \\ -1 \end{pmatrix},\ \mathbf{B}^{(2)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Let us denote by $\overline{g}\colon \mathbb{R}^n \to \mathbb{R}^2$ the resulting function. Observe that $\overline{g}(\mathbf{v}) = (1, \max(0, 1 - \max(0, g(\mathbf{v}) - u)))^\intercal$ for all $\mathbf{v} \in \mathbb{R}^n$. Hence, we have that $c_u(\mathbf{v})$ is undefined if and only if $g(\mathbf{v}) \leq u$ and yields 1 otherwise.

Lemma 1. *Let $g\colon \mathbb{R}^n \to \mathbb{R}$ be an NN-encoded function and consider the constraint $\ell \leq g(\mathbf{x}) \leq u$. One can construct NN-encoded classifiers c_ℓ and c_u, of size linear with respect to g, for the labelled data set $(\{\mathbf{0}\}, \{\mathbf{0} \mapsto 1\})$ such that the deterministic attack $\delta\colon \mathbf{0} \mapsto \mathbf{v}$*

- *induces a misclassification of $\mathbf{0}$ with respect to c_ℓ if and only if $\ell \leq g(\mathbf{v})$ and*
- *it induces a misclassification of $\mathbf{0}$ with respect to c_u if and only if $g(\mathbf{v}) \leq u$.*

We now show how to modify the NN to obtain classifiers c_ℓ, c_u such that $\mathbf{x}$ is misclassified by both c_ℓ and c_u if and only if the constraint holds.

Enforcing Universal Misclassification. A randomized attack with misclassification value 1 can be assumed to be deterministic. Indeed, from Eq. (3) it follows that for any such randomized attack we must have $M_c(\delta) = |X|$ for all $\delta \in A$ and all $c \in C$. Hence, we can choose any such $\delta \in A$ and consider the randomized attack $(\{\delta\}, \{\delta \mapsto 1\})$ which also has misclassification value 1.

Lemma 2. *Consider the labelled data set $\mathcal{X} = (X, c_t)$ with the finite set of classifiers $C \subseteq \mathbb{C}$. There exists an ε-bounded randomized attack $(A, \mathbb{P})$ with $\mathbf{Val}(A, \mathbb{P}) = 1$ if and only if there exists a deterministic attack δ such that*

- *$\|\delta(\mathbf{x})\|_1 \leq \varepsilon$ for all $\mathbf{x} \in X$ and*
- *for all $\mathbf{x} \in X$ and all $c \in C$ we have that either $c(\mathbf{x} + \delta(\mathbf{x}))$ is undefined or it is not equal to $c_t(\mathbf{x})$.*

With Lemmas 1 and 2 established, the proof of Theorem 1 is now complete.

5 SMT and MILP Encodings

In this section, we describe the main elements of our SMT and MILP encodings to compute (optimal) randomized attacks or prove the robustness of classifier ensembles. We start with a base encoding and will afterwards explain how to explicitly encode the classifiers and the loss function.

5.1 Base Problem Encoding

First, we assume a labelled data set $\mathcal{X} = (X, c_t)$, the attack bound ε, and the robustness bound α are input to the problem. In particular, the data set $X = \{\mathbf{x}^1, \ldots, \mathbf{x}^{|X|}\} \subseteq \mathbb{R}^d$ has data points $\mathbf{x}^j = (x_1^j, \ldots, x_d^j) \in \mathbb{R}^d$ for $1 \leq j \leq |X|$. Furthermore, we assume the number $|A|$ of attacks that shall be computed is fixed. Recall that, to show that the set of classifiers is robust, we can compute a sufficient bound on the number of attacks — see Sec. 4.1.

For readability, we assume the classifiers C and the loss function ℓ_{0-1} are given as functions that can directly be used in the encodings, and we will use the absolute value $|x|$ for $x \in \mathbb{R}$. Afterwards, we discuss how to actually encode classifiers and the loss function. We use the following variables:

- For the attacks from A, we introduce $\delta_1, \ldots, \delta_{|A|}$ with $\delta_i \in \mathbb{R}^{|X| \times d}$ for $1 \leq i \leq |A|$. Specifically, δ_i shall be assigned all attack values for the i-th attack from A. That is, δ_i^j is the attack for the data point $\mathbf{x}^j = (x_1^j, \ldots, x_d^j) \in \mathbb{R}^d$ with $\delta_i^j = (\delta_i^{j,1}, \ldots, \delta_i^{j,d})$ for $1 \leq j \leq |X|$.
- We introduce $p_1, \ldots, p_{|A|}$ to form a probability distribution over deterministic attacks; p_i is assigned the probability to execute attack δ_i.

The classifier ensemble C is not ε, α-robust as in Definition 6 if and only if the following constraints are satisfiable.

$$\forall c \in C. \quad \sum_{j=1}^{|X|} \sum_{i=1}^{|A|} \left(p_i \cdot \ell_{0-1}(c, \mathbf{x}^j, \delta_i^j) \right) \geq \alpha \cdot |X| \tag{4}$$

$$\forall i \in \{1, \ldots, |A|\}, j \in \{1, \ldots, |X|\}. \quad \sum_{k=1}^{d} |\delta_i^{j,k}| \leq \varepsilon \tag{5}$$

$$\sum_{i=1}^{|A|} p_i = 1 \tag{6}$$

$$\forall i \in \{1, \ldots, |A|\}. \quad p_i \geq 0 \tag{7}$$

Indeed, (4) enforces the misclassification value to be at least α; (5) ensures an ε-bounded randomized attack; finally, by (6) and (7) the probability variables induce a valid probability distribution.

Specific Encodings. For the SMT encoding, we can make use of the $\max(\cdot)$ native to implement the absolute value. In particular for the MILP, however, we employ so-called "big-M" tricks to encode max functions and a (restricted) product operation (cf. [7]). Specifically, the product is required to obtain the value resulting from the multiplication of the loss function and probability variables.

As an example of "big-M" trick, suppose we have variables $a \in \mathbb{Q} \cap [0, 1], b \in \{0, 1\}$, and a constant $M \in \mathbb{Q}$ such that $M > a + b$. We introduce a variable $c \in \mathbb{Q} \cap [0, 1]$ and add the following constraints which clearly enforce that $c = ab$.

$$c \geq a - M(1 - b), \quad c \leq a + M(1 - b), \quad c \leq 0 + Mb \; .$$

Note that M can be chosen to be the constant 2 in this case.

Encoding the Loss Function. We encode the zero-one loss function from Definition 4 as an if-then-else expression making use of the ground truth classifier c_t.We introduce one variable $\ell^c_{i,j}$ per classifier $c \in C$ for all attacks $\delta_i \in A$ and all datapoints $\mathbf{x}^j \in X$. In the SMT encoding, we can then define

$$\ell^c_{i,j} = ITE((c(\mathbf{x}^j + \delta^j_i) = c_t(\mathbf{x}^j)), 0, 1) \tag{8}$$

so that $\ell^c_{i,j} = \ell_{0-1}(c, \mathbf{x}^j, \delta^j_i)$. In our MILP encoding we have to simulate the *ITE* primitive using constraints similar to the ones mentioned above.

5.2 Classifier Encoding

As mentioned in the preliminaries, neural networks implement functions by way of layer composition. Intuitively, the input of a layer is by a previous layer. When fed forward, input values are multiplied by a weight, and a bias value will be added to it. Matrix operations realized by a neural network can thus be encoded as linear functions. For max-pooling operations and the ReLU activation function, one can use the native $\max(\cdot)$ operation or implement a maximum using a "big-M trick". For this, a suitable constant M has to be obtained beforehand (cf. [7]). We also use a (discrete) convolution operation, as a linear function.

6 Experiments

In the previous section, we showed that our problem of neural network robustness verification is **NP**-hard. Meaningful comparison between approaches, therefore, needs to be experimental. To that end, we use classifiers trained on multiple image data sets and report on the comparison between the SMT and MILP encodings. In what follows, we analyze the running time behavior of the different verifiers, the generated attacks and the misclassification value for the given data points, and whether a set of classifiers is robust against predefined thresholds.

6.1 Experimental Setup

For each experiment, we define a set of classifiers C, our data points $\mathcal{X}$, the number of attacks $|A|$, and both the ε- and α-values. Then, we generate the attacks A and the probability distribution $\mathbb{P}$ using SMT and MILP solvers. If no randomized attack $(A, \mathbb{P})$ is found (UNSAT), we have shown that our set of classifiers is ε, α-robust with respect to the predefined thresholds.

Toolchain. Our NN robustness verifier[2], is implemented as part of a Python 3.x toolchain. We use the SMT solver Z3 [9] and the MILP solver Gurobi [14] with their standard settings. To support arbitrary classifiers, we created a generic pipeline using the TensorFlow API, and support Max-Pooling layers, convolutional layers, and dense layers with ReLU activation functions [1]. We focus on binary classifiers by defining certain classification boundaries. We train the classifiers using the Adam optimizer [17] as well as stochastic gradient descent.

[2] Available at https://tinyurl.com/ensemble-robustness.

Table 1. SMT versus MILP

Benchmark Information								SMT		MILP	
ID	Name	$\|C\|$	$\|A\|$	$\|\mathcal{X}\|$	dim	α	ε	Time	$\mathbf{Val}(A, \mathbb{P})$	Time	$\mathbf{Val}(A, \mathbb{P})$
2	mnist_0_1	3	2	4	7×7	0.2	100	-TO-	–	12.43	0.25
4	mnist_0_1_2convs	3	2	4	8×8	0.4	100	-TO-	–	53.92	0.4
7	mnist_0_1	3	2	4	8×8	0.4	1000	-TO-	–	0.34	0.4
8	mnist_0_1	3	2	4	8×8	0.9	1000	-TO-	–	50.09	0.9
9	mnist_0_1	3	3	4	8×8	0.9	60	-TO-	–	34.02	1
13	mnist_4_5	3	4	4	10×10	0.9	50	-TO-	–	144.32	1
14	mnist_7_8	3	4	4	6×6	0.1	60	-TO-	–	18.94	0.25
16	mnist_4_5	3	2	4	10×10	0.1	1000	155.73	0.38	101.16	0.1
17	mnist_4_5	3	3	4	10×10	0.1	80	403.25	0.25	101.47	0.25
18	mnist_4_5	3	2	4	10×10	0.15	80	216.65	0.38	44.26	0.15
19	mnist_4_5	3	2	4	10×10	0.2	100	156.63	0.38	54.36	0.25
22	mnist_7_8	3	2	4	6×6	0.9	0.1	-TO-	–	4	robust
26	traffic_signs	3	2	4	10×10	1	0.01	-TO-	–	17	robust
27	traffic_signs	3	2	4	10×10	1	0.1	-TO-	–	-TO-	–

Table 2. MILP versus MaxMILP

Benchmark Information								MILP		MaxMILP	
ID	Name	$\|C\|$	$\|A\|$	$\|l\mathcal{X}\|$	dim	α	ε	Time	$\mathbf{Val}(A, \mathbb{P})$	Time	$\mathbf{Val}(A, \mathbb{P})$
1	mnist_0_1	3	2	4	7×7	0.1	1000	57.79	0.25	46.23	1*
3	mnist_0_1_2convs	3	2	4	8×8	0.2	1000	738.76	0.5	93.54	1*
7	mnist_0_1	3	2	4	8×8	0.4	1000	0.34	0.4	0.34	1*
10	mnist_0_1	3	4	4	8×8	0.9	60	51.39	1	51.39	1*
14	mnist_7_8	3	4	4	6×6	0.1	60	18.94	0.25	21.20	1
17	mnist_4_5	3	3	4	10×10	0.1	80	101.47	0.25	88.39	1
20	mnist_3_6	2	9	2	8×8	1	0.005	7	robust	7	robust
21	mnist_7_8	3	2	4	6×6	1	0.1	4	robust	4	robust
24	mnist_0_2	3	27	4	9×9	1	0.01	108	robust	108	robust
25	mnist_0_2	3	30	4	9×9	1	0.01	120	robust	120	robust
28	traffic_signs	3	3	4	10×10	1	0.01	45	robust	45	robust
29	traffic_signs	3	3	4	10×10	1	0.01	–TO–	-	–TO–	–

Data Sets. MNIST consists of 70 000 images of handwritten digits) [10] and is widely used for benchmarking in the field of machine learning [8,16]. We trained classifiers to have a test accuracy of at least 90%.

German traffic sign is a multi-class and single-image classification data set, containing more than 50 000 images and more than 40 classes [25]. Traffic sign recognition and potential attacks are of utmost importance for self-driving cars [27]. We extracted the images for "Give way" and "priority" traffic signs from the data set and trained classifiers on this subset to have an accuracy of at least 80%.

Optimal Attacks. As MILP inherently solves optimization problems, we augment the base encoding from Eqs (4)–(7) with the following objective function:

$$\max \quad \sum_{k=1}^{|C|} \sum_{j=1}^{|\mathcal{X}|} \sum_{i=1}^{|A|} \left(p_i \cdot \ell_{0-1}(c_k, \mathbf{x}^j, \delta_i^j) \right).$$

An optimal solution with respect to the objective **may** yield a randomized attack inducing the maximal misclassification value among all ε-bounded attacks.[3]

Alternative Attacks. Our method generates attacks taking the whole ensemble of classifiers into account, which is computationally harder than just considering single classifiers [12,15] due to an increased number of constraints and variables in the underlying encodings. To showcase the need for our approach, we implemented two other ways to generate attacks that are based on individual classifiers and subsequently lifted to the whole ensemble. Recall that we assume the attacker does not know which classifier from the ensemble will be chosen.

First, for the classifier ensemble C we compute — using a simplified version of our MILP encoding — an optimal attack δ_c for each classifier $c \in C$. Each such δ_c maximizes the misclassification value, that is, the loss, for the classifier c. The attack set $A_C = \{\delta_c \mid c \in C\}$ together with a uniform distribution $Distr_A$ over A_C form the so-called *uniform attacker* $(A_C, Distr_A)$.

Second, to compare with deterministic attacks, we calculate for each attack from A_C the misclassification value over all classifiers. The *best deterministic attack* is any attack from A_C inducing a maximal misclassification value.

6.2 Evaluation

We report on our experimental results using the aforementioned data sets and attacks. For all experiments we used a timeout of 7200 s (TO). Each benchmark has an ID, a name, the number $|C|$ of classifiers, the number $|A|$ of attacks, the size $|\mathcal{X}|$ of the data set, the dimension of the image (dim), the robustness bound α, and the attack bound ε. The names of the MNIST benchmarks are of the form "mnist_x_y", where x and y are the labels; the additional suffix "_$n$convs" indicates that the classifier has n convolutional layers. We provide an excerpt of our experiments, full tables are available in the appendix.

SMT versus MILP. In Table 1, we report on the comparison between SMT and MILP. Note that for these benchmarks, the MILP solver just checks the feasibility of the constraints without an objective function. We list for both solvers the time in seconds and the misclassification value $\mathbf{Val}(A, \mathbb{P})$, rounded to 2 decimal places, for the generated randomized attack $(A, \mathbb{P})$, if it can be computed before the timeout. If there is no solution, the classifier ensemble C is ε, α-robust, and instead of a misclassification value we list "robust".

[3] Note that we sum over classifiers instead of minimizing, as required in $\mathbf{Val}(A, \mathbb{P})$.

We observe that SMT is only able to find solutions within the timeout for small α and large ε values. Moreover, if the ensemble is robust, that is, the problem is not satisfiable, the solver does not terminate for any benchmark. Nevertheless, for some benchmarks (see Table 1, entries 16–19) the SMT solver yields a higher misclassification value than the MILP solver — that is, it finds a better attack. The MILP solver, on the other hand, solves most of our benchmarks mostly within less than a minute, including the robust ones. Despite the reasonably low timeout of 7200 s, we are thus able to verify the robustness of NNs with around 6 layers. Running times visibly differ for other factors such as layer types.

MILP versus MaxMILP. Table 2 compares some of the MILP results to those where we optimize the mentioned objective function, denoted by MaxMILP. The MILP solver Gurobi offers the possibility of so-called *callbacks*, that is, while an intermediate solution is not proven to be optimal, it may already be feasible. In case optimality cannot be shown within the timeout, we list the current (feasible) solution, and mark optimal solutions with $*$. The misclassification value for the MaxMILP solver is always 1. For robust ensembles, it is interesting to see that the MaxMILP encoding sometimes needs less time.

Table 3. Attacker comparison

Benchmark Information								UA	BDA	MaxMILP
ID	Name	$\lvert C\rvert$	$\lvert A\rvert$	$\lvert\mathcal{X}\rvert$	dim	Epsilon	Alpha	$\mathbf{Val}(A, \mathbb{P})$	$\mathbf{Val}(A, \mathbb{P})$	$\mathbf{Val}(A, \mathbb{P})$
3	mnist_0_1_2convs	3	2	4	8×8	0.2	1000	0.33	0.25	1*
7	mnist_0_1	3	2	4	8×8	0.4	1000	0.33	0.5	1*
8	mnist_0_1	3	2	4	8×8	0.9	1000	0.33	0.5	1*
9	mnist_0_1	3	3	4	8×8	0.9	60	0.33	0.5	1*
10	mnist_0_1	3	4	4	8×8	0.9	60	0.33	0.5	1*
11	mnist_4_5	3	2	4	10×10	0.2	100	0.33	0.25	1*
14	mnist_7_8	3	4	4	6×6	0.1	60	0.33	0.5	1
15	mnist_7_8	3	10	4	6×6	0.1	60	0.33	0.5	1
16	mnist_4_5	3	2	4	10×10	0.1	1000	0.33	0.75	1*
17	mnist_4_5	3	3	4	10×10	0.1	80	0.33	0.5	1
18	mnist_4_5	3	2	4	10×10	0.15	80	0.33	0.5	1*

In Table 3, we compare the MaxMILP method to the *uniform attacker* (UA) and the *best deterministic attacker* (BDA). What we can see is that the best deterministic attacker usually achieves higher misclassification values than the uniform attacker, but none of them are able to reach the actual optimum of 1.

Discussion of the Results. Within the timeout, our method is able to generate optimal results for medium-sized neural networks. The running time is mainly influenced by the number and type of the used layers, in particular, it is governed by convolutional and max-pooling layers: these involve more matrix operations than dense layers. As expected, larger values of the robustness bound α and smaller values of the attack bound ε typically increase the running times.

7 Conclusion and Future Work

We presented a new method to formally verify the robustness or, vice versa, compute optimal attacks for an ensemble of classifiers. Despite the theoretical hardness, we were able to, in particular by using MILP-solving, provide results for meaningful benchmarks. In future work, we will render our method more scalable towards a standalone verification tool for neural network ensembles. Moreover, we will explore settings where we do not have white-box access to the classifiers and employ state-of-the-art classifier stealing methods.

References

1. Abadi, M.: Tensorflow: learning functions at scale. In: Garrigue, J., Keller, G., Sumii, E. (eds.) ICFP, p. 1. ACM (2016)
2. Abbasi, M., Gagné, C.: Robustness to adversarial examples through an ensemble of specialists. In: ICLR (Workshop), OpenReview.net (2017)
3. Abbasi, M., Rajabi, A., Gagné, C., Bobba, R.B.: Toward adversarial robustness by diversity in an ensemble of specialized deep neural networks. In: Goutte, C., Zhu, X. (eds.) Canadian AI 2020. LNCS (LNAI), vol. 12109, pp. 1–14. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-47358-7_1
4. Akintunde, M.E., Kevorchian, A., Lomuscio, A., Pirovano, E.: Verification of RNN-based neural agent-environment systems. In: AAAI, pp. 6006–6013. AAAI Press (2019)
5. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in ai safety. CoRR abs/1606.06565 (2016)
6. Apt, K.R., Grädel, E.: Lectures in Game Theory for Computer Scientists. Cambridge University Press, Cambridge (2011)
7. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: NeurIPS, pp. 4795–4804 (2018)
8. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: extending MNIST to handwritten letters. In: IJCNN, pp. 2921–2926. IEEE (2017)
9. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
10. Deng, L.: The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)
11. Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: A formalization of robustness for deep neural networks. CoRR abs/1903.10033 (2019)
12. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
13. Freedman, R.G., Zilberstein, S.: Safety in AI-HRI: challenges complementing user experience quality. In: AAAI Fall Symposium Series (2016)
14. Gurobi Optimization Inc: Gurobi optimizer reference manual. http://www.gurobi.com (2013)
15. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5

16. Keysers, D.: Comparison and combination of state-of-the-art techniques for handwritten character recognition: topping the mnist benchmark. arXiv preprint arXiv:0710.2231 (2007)
17. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) ICLR (2015). http://arxiv.org/abs/1412.6980
18. Kwiatkowska, M.Z.: Safety verification for deep neural networks with provable guarantees (invited paper). In: CONCUR, LIPIcs, vol. 140, pp. 1–5. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
19. Nyholm, S.: The ethics of crashes with self-driving cars: a roadmap, ii. Philos. Compass **13**(7), e12506 (2018)
20. Perdomo, J.C., Singer, Y.: Robust attacks against multiple classifiers. CoRR abs/1906.02816 (2019)
21. Pinot, R., Ettedgui, R., Rizk, G., Chevaleyre, Y., Atif, J.: Randomization matters. how to defend against strong adversarial attacks. CoRR abs/2002.11565 (2020)
22. Ranzato, F., Zanella, M.: Robustness verification of decision tree ensembles. OVERLAY@AI*IA, **2509**, pp. 59–64 (2019). CEUR-WS.org
23. Science, N.: National Science Technology and Council: Preparing for the Future of Artificial Intelligence, T.C. (2016)
24. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019). https://doi.org/10.1145/3290354
25. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The German traffic sign recognition benchmark: a multi-class classification competition. In: IJCNN, pp. 1453–1460. IEEE (2011)
26. Stoica, I., et al.: A Berkeley view of systems challenges for AI. CoRR abs/1712.05855 (2017)
27. Yan, C., Xu, W., Liu, J.: Can you trust autonomous vehicles: contactless attacks against sensors of self-driving vehicle. DEF CON **24**(8), 109 (2016)