Robustness Verification of Classification Deep Neural Networks via Linear Programming

Wang Lin[1,2,3] Zhengfeng Yang[2*] Xin Chen[3*] Qingye Zhao[3] Xiangkun Li[2] Zhiming Liu[4] Jifeng He[2]
[1] School of Information Science and Technology, Zhejiang Sci-Tech University
[2] Shanghai Key Lab of Trustworthy Computing, East China Normal University
[3] State Key Laboratory for Novel Software Technology, Nanjing University
[4] Center for Research and Innovation in Software Engineering, Southwest University
linwang@zstu.edu.cn, zfyang@sei.ecnu.edu.cn, chenxin@nju.edu.cn
qingyezhao@foxmail.com, xkli@stu.ecnu.cn, zhimingliu88@swu.edu.cn, jifeng@sei.ecnu.edu.cn

Abstract

There is a pressing need to verify robustness of classification deep neural networks (CDNNs) as they are embedded in many safety-critical applications. Existing robustness verification approaches rely on computing the over-approximation of the output set, and can hardly scale up to practical CDNNs, as the result of error accumulation accompanied with approximation. In this paper, we develop a novel method for robustness verification of CDNNs with sigmoid activation functions. It converts the robustness verification problem into an equivalent problem of inspecting the most suspected point in the input region which constitutes a nonlinear optimization problem. To make it amenable, by relaxing the nonlinear constraints into the linear inclusions, it is further refined as a linear programming problem. We conduct comparison experiments on a few CDNNs trained for classifying images in some state-of-the-art benchmarks, showing our advantages of precision and scalability that enable effective verification of practical CDNNs.

1. Introduction

Classification deep neural networks (CDNNs) are a special kind of deep neural networks that take in complicated, high-dimensional input, transform it through multiple layers of neurons and finally identify it as a specific output label or class. Acting as classifiers, they have been adopted in a variety of applications, such as adaptive control [31, 34], pattern recognition [28], image understanding [8, 18], cyber-security [29], speech and audio analysis [14, 15] and self-driving [26]. It is a growing trend that embedding CDNNs in safety-critical systems to make judgements like human experts [19].

Unfortunately, it has been reported by many researchers [3, 32] that CDNNs are unrobust with respect to perturbations. For example, applying minimal changes to the input image, being imperceptible to the human eye, can make the well-trained CDNN misclassify it.

The unrobust issue obviously raises potential safety issues for safety critical systems using neural networks and calls for effective verification techniques that can check the robustness of CDNNs [11, 17]. We state that a CDNN is robust when, given every possible input value within a specified region, its output is guaranteed to have the same classification label, which corresponds to the principal robust requirement of classification: minimal perturbations (restricted by a threshold) should not affect the classification result.

Given a CDNN, its robustness can be verified by estimating the range of the output set with respect to a given input region. Some methods are proposed to compute the output range of CDNNs of particular types [2, 6, 12, 22]. The method combining local gradient descent search with global mixed integer linear programming can handle the CDNNs with ReLU activation functions [9], which is further extended to treat a special class of perturbations [33]. The ployhedra manipulating based method is able to compute the exact output region of CDNNs with ReLU activation functions and an union of ployhedra as its input set [35].

In general, the computation of output set is very difficult since the theories to handle general activation functions are undecidable. The methods that transform the verification problem of CDNNs into programming receive most atten-

*Corresponding authors. This work was supported in part by the National Key Research and Development Program under Grant 2017YFB1001801, in part by the National Natural Science Foundation of China under Grant 61772203, 61602348, 61690204, 61632015, 61672435, 61732019, 61751210 and 61572441, in part by the Shanghai Natural Science Foundation under Grant 17ZR1408300, in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization, and in part by Southwest University under Grant SU116007.

tions in recent years. The method extending the Simplex method to deal with the constraints imposed by ReLU nodes is suggested to verify the ReLU equipped DNNs [17]. The approach integrating SAT solving with linear programming and linear approximation can handle CDNNs with piecewise linear activation functions [10].

The piece-wise linear feature of ReLU make the verification of ReLU equipped CDNNs much easier [5, 25]. For those nonlinear activation functions, such as sigmoid, approximation based methods are studied. The abstraction-refinement method introduces piece-wise linear functions to approximate sigmoid activation functions, encodes the network as a set of linear constraints and then solves them by the solver HYSAT [24]. The discretization based method uses layer-by-layer analysis and SMT techniques to verify CDNNs for image classification [16]. A simulation-based approach formulates the output range estimation problem into the maximal sensitivity, which can be computed via solving a chain of convex optimization problems [36].

As the error arising from approximation propagates along the network layer by layer, those approximation-based approaches are prone to fail when encountering practical CDNNs. How to provide adequate precision and scalability to satisfy the requirement of verifying practical CDNNs is the key challenge that verification methods must face with.

This paper proposes a novel method to robustness verification of CDNNs equipped with the activation function: sigmoid. Rather than computing the over-approximation of the output set with respect to the given input region, it transfers the robustness verification problem into the problem of finding the point in the input region that is easier to be assigned with a different label than any other points in the region and checks whether or not the point will be misclassified. To be specific, we build an equivalent optimization problem where the relation between the neurons in the networks is encoded as constraints and the objective function aims to find the minimal difference between the output values with respect to the desired label and the other labels. As the desired label should have the maximal value, the optimum of the optimization problem is positive if and only if the robustness property holds.

The optimization problem is further relaxed by replacing the nonlinear sigmoid function in constraints with its linear inclusions, so that linear programming (LP) solvers can work. Precisely, the sigmoid function is contained in a closed polyhedron produced by combining bound derivation with linear inclusion. The derived optimization problem constitutes a sufficient condition for verification as the feasible set of its constraints is a superset of the original one.

The proposed method has two advantages over existing ones: 1. It verifies robustness by inspecting whether or not the worst point in the input region will be classified differently, and thus provides more precise estimation on robustness of CDNNs. 2. It can safely scale up to realistic sized networks which beyond the reach of existing ones. It is supported by the comparison experiments conducted on CDNNs generated from many state-of art benchmarks.

The main contributions of the paper are summarized as follows: 1. We propose a novel method for verifying robustness of CDNNs which converts the robustness verification problem into an equivalent optimization problem. 2. We suggest a new computational approach, by approximating the activation function sigmoid with a tightly enclosed region, and then leveraging LP solvers to solve it. 3. We conduct a comparison experiment on a set of benchmarks consisting of many state-of-the-art CDNNs, which shows our advantages on precision and scalability that enable verification of practical CDNNs.

The paper is organized as follows. Section 2 introduces some notions for CDNNs, and then describes the robustness verification problem. In Section 3, the problem of robustness verification is transformed to a nonlinear optimization problem and an LP based computational method is addressed to deal with it. We show the experiments on benchmarks, confirming the effectiveness of our algorithm in Section 4 before concluding in Section 5.

2. Preliminaries

Notations Let $\mathbb{N}$ and $\mathbb{R}$ be the set of natural numbers and the field of real number, respectively; $\digamma_n$ denotes the set consisting of the positive integer numbers not greater than n, i.e., $\digamma_n = \{1, 2, \ldots, n\}$, where n is a positive integer; $\mathcal{I}$ denotes the open interval $(0, 1)$, and $\mathcal{I}^s$ denotes the Cartesian product $\mathcal{I}^s = \mathcal{I} \times \cdots \times \mathcal{I}$ with the positive integer number s.

2.1. Deep Neural Networks

Deep neural networks (DNNs) consist of an input layer, an output layer, and multiple hidden layers in between. Neurons (so-called nodes), in a DNN are arranged in disjoint layers, with each neuron in one layer connected to the next layer, but no connection between neurons in the same layer. Furthermore, the output of each neuron in the hidden layer is assigned by a linear combination of the neuron outputs of the previous layer, and then applying a non-linear activation function. Formally, DNNs are defined as follows.

Definition 1 (Deep Neural Network) *A deep neural network $\mathcal{N}$ is a tuple $\langle L, X, W, B, \Phi \rangle$, where*

- *$L = \{L^{[0]} \ldots, L^{[n]}\}$ is a set of layers, where layer $L^{[0]}$ is the input layer, $L^{[n]}$ is the output layer, and $L^{[1]}, \ldots, L^{[n-1]}$ are the hidden layers. Each layer $L^{[k]}, 0 \leq k \leq n$ is associated with an s_k-dimensional vector space $\Psi_k \subseteq \mathbb{R}^{s_k}$, in which each dimension corresponds to a neuron.*

- $X = \{\mathbf{x}^{[0]}, \ldots, \mathbf{x}^{[n]}\}$, *where* $\mathbf{x}^{[k]}$ *is the vector corresponding to the values of the neurons in the layer* $L^{[k]}$ *for* $0 \leq k \leq n$.

- $W = \{W^{[1]}, \ldots, W^{[n]}\}$ *is the set of weight matrices. Each non-input layer* $L^{[k]}$ *with* $1 \leq k \leq n$ *has a weight matrix* $W^{[k]} \in \mathbb{R}^{s_k \times s_{k-1}}$, *and neurons in* $L^{[k]}$ *are connected to neurons from the preceding layer* $L^{[k-1]}$ *by the weight matrix* $W^{[k]}$.

- $B = \{\mathbf{b}^{[1]}, \ldots, \mathbf{b}^{[n]}\}$ *is the set of bias vectors. For each non-input layer* $L^{[k]}$ *with* $1 \leq k \leq n$, *the bias vector* $\mathbf{b}^{[k]} \in \mathbb{R}^{s_k}$ *is used to assigned bias values to the neurons in* $L^{[k]}$.

- $\Phi = \{\phi^{[1]}, \ldots, \phi^{[n]}\}$ *is a set of activation functions* $\phi^{[k]} : \Psi_{k-1} \to \Psi_k$, *one for each non-input layer* $L^{[k]}$ *with* $1 \leq k \leq n$. *Furthermore, the value vector* $\mathbf{x}^{[k]}$ *for the neurons in* $L^{[k]}$ *are determined by the value vector of* $\mathbf{x}^{[k-1]}$ *with the weight matrix* $W^{[k]}$, *the bias vector* $\mathbf{b}^{[k]}$ *and the activation function* $\phi^{[k]}$, *i.e.,* $\mathbf{x}^{[k]} = \phi^{[k]}(W^{[k]}\,\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]})$ *with the activation function* $\phi^{[k]}$ *being applied element-wise.*

Given a DNN $\mathcal{N} : \langle L, X, W, B, \Phi\rangle$, it is seen that the output of the $k-1$-th layer is the input of the k-th layer. From the mapping point of view, each non-input layer $L^{[k]}$, $1 \leq k \leq n$, defines a function $f_k : \mathbb{R}^{s_{k-1}} \to \mathbb{R}^{s_k}$, with

$$f_k(\mathbf{x}) = \phi^{[k]}(W^{[k]}\mathbf{x} + \mathbf{b}^{[k]}). \tag{1}$$

Therefore, the behavior of $\mathcal{N}$ is defined by the composition function $f : \mathbb{R}^{s_0} \to \mathbb{R}^{s_n}$, which is defined as

$$f(\mathbf{x}) = f_n(f_{n-1}(\cdots(f_1(\mathbf{x})))). \tag{2}$$

For the purpose of this paper we are assuming that all the weights matrices W and bias vectors B in $\mathcal{N}$ have fixed values. For an overview of weight and bias selection and a review of previous work, see [13]. Observing the composition of the given DNN, the difficulty in proving its properties is caused by the presence of activation functions. The activation function is generally a nonlinear function connecting the neuron values of the preceding layer to the ones of the following layer. There are some typical activation functions, such as the sigmoid, rectified linear unit (*ReLU*), and tanh functions[13]. In this paper, we only focus on DNNs with the widely used sigmoid activation function

$$\phi(x) = \frac{1}{1 + e^{-x}}. \tag{3}$$

We extend the definition of $\phi(x)$ to apply component-wise to vectors $\mathbf{x}$ as $\phi(\mathbf{x})$: $\big(\phi(x_1), \cdots, \phi(x_n)\big)^T$. Due to the property of the sigmoid function with $\phi(x) : \mathbb{R} \to \mathcal{I}$, the input and the output of the function f_k for the given input $\mathbf{x}^{[0]} \in \mathbb{R}^{s_0}$, can be restricted to

$$\mathbf{x}^{[k]} = f_k(\mathbf{x}^{[k-1]}) \in \mathcal{I}^{s_k}, \quad 1 \leq k \leq n.$$

According to the Universal Approximation Theorem [27], it guarantees that, in principle, a DNN $\mathcal{N}$ is able to approximate any nonlinear real-valued function. Despite the impressive ability of approximating nonlinear functions, much complexities represent in predicting the output behaviors of $\mathcal{N}$ due to the nonlinearity and non-convexity of DNNs.

2.2. Classification Deep Neural Networks and Robustness Property

In this subsection, we will introduce the notion of classification deep neural networks (CDNNs), and describe the robustness verification problem of CDNNs. CDNN is one of the primary applications in deep neural networks. In this type of networks, a CDNN will act as a *classifier*, which is used to specify which of labels some input belongs to. Given a CDNN $\mathcal{N} : \langle L, X, W, B, \Phi\rangle$ with an input $\mathbf{x}^{[0]}$, the activation for $\mathbf{x}^{[k]}$ in layer k is $\mathbf{x}^{[k]} = f_k(f_{k-1}(\cdots(f_1(\mathbf{x}^{[0]}))))$, the activation for $\mathbf{x}^{[n]}$ in the output layer can be propagated through the layers shown in (2), that is, $\mathbf{x}^{[n]} = f(\mathbf{x}^{[0]}) \in \mathcal{I}^{s_n}$.

The classification decision is asked to produce a function $\varrho : \mathbb{R}^{s_0} \to F_{s_n}$ based on the output value $\mathbf{x}^{[n]}$, assigning to $\mathbf{x}^{[0]}$ to label $\ell \in F_{s_n}$ by selecting the index corresponding to the largest element in $\mathbf{x}^{[n]}$. Therefore, the function ϱ is the composition of $\arg\max$ and f, i.e.,

$$\varrho(\mathbf{x}^{[0]}) = \arg\max(f(\mathbf{x}^{[0]})) = \underset{1 \leq \ell \leq s_n}{\arg\max}(\mathbf{x}^{[n]}). \tag{4}$$

To make the input $\mathbf{x}^{[0]}$ be classified, the largest element of the output $\mathbf{x}^{[n]}$ must be unique. In other words, $\ell = \varrho(\mathbf{x}^{[0]})$ implies that $\mathbf{x}_\ell^{[n]} > \mathbf{x}_{\tilde{\ell}}^{[n]}$ for all $\tilde{\ell} \neq \ell$.

Given a CDNN $\mathcal{N}$, we say that two inputs $\mathbf{x}^{[0]}$ and $\mathbf{y}^{[0]}$ have the same label if $\varrho(\mathbf{x}^{[0]}) = \varrho(\mathbf{y}^{[0]})$. This property for having the same label can also be generalized from the discrete inputs to the input region. More specifically, we say that the input region Θ has the same label if two arbitrary inputs $\mathbf{x}^{[0]}$ and $\mathbf{y}^{[0]}$ chosen from Θ, have the same label, namely,

$$\varrho(\mathbf{x}^{[0]}) = \varrho(\mathbf{y}^{[0]}), \quad \forall \mathbf{x}^{[0]}, \mathbf{y}^{[0]} \in \Theta.$$

Similar to the description in [16], this property that the given input region having the same label is called as *robustness*. In this situation, robustness specification of the given CDNN $\mathcal{N}$ is described over the specified input region Θ. We begin with a common definition for *robustness*.

Definition 2 (Robustness) *Given a classification deep neural network* $\mathcal{N}$ *with an input region* Θ, *the robustness property holds if and only if all inputs within the input region* Θ *have the same label, i.e.,*

$$\forall \mathbf{x}^{[0]},\ \mathbf{y}^{[0]} \in \Theta \implies \varrho(\mathbf{x}^{[0]}) = \varrho(\mathbf{y}^{[0]}). \tag{5}$$

Namely, if there exists an index ℓ, $1 \leq \ell \leq s_n$, such that

$$\varrho(\mathbf{x}^{[0]}) = \ell, \quad \forall\, \mathbf{x}^{[0]} \in \Theta \tag{6}$$

is satisfied, we say that $\mathcal{N}$ with respect to Θ is robust.

Given a CDNN $\mathcal{N}$, the problem of robustness verification is to decide whether $\mathcal{N}$ with respect to the input region Θ is robust. According to Definition 2, the target for robustness verification is to determine whether there exists an unified label (index) ℓ such that any output $\mathbf{x}^{[n]}$ produced by the input selected from Θ satisfies

$$\mathbf{x}_{\ell}^{[n]} > \mathbf{x}_{\tilde{\ell}}^{[n]} \quad \text{for all } \tilde{\ell} \neq \ell. \tag{7}$$

On the contrary, we say that $\mathcal{N}$ is unrobust if there exist two inputs $\mathbf{x}^{[0]}, \mathbf{y}^{[0]} \in \Theta$ such that

$$\arg\max(f(\mathbf{x}^{[0]})) \neq \arg\max(f(\mathbf{y}^{[0]}))$$

is satisfied. Hereafter we will always consider the input region Θ is a Cartesian product, that is, $\Theta = \Omega_1 \times \cdots \times \Omega_{s_0}$ where $\Omega_i = [a_i, b_i]$ is a closed interval bounded by $a_i, b_i \in \mathbb{R}$ for all $1 \leq i \leq s_0$.

3. Robustness Verification of Classification Deep Neural Networks

Instead of computing the over-approximation of the output set, in this work we propose a direct method for attacking the robustness verification problem. Given a CDNN with the input region, we establish a nonlinear optimization problem equivalent to the original robustness verification problem. Namely, a sufficient and necessary condition for robustness verification is to inspect the most suspicious point in the input region is robust or not. (see Section 3.1). To alleviate the computational intractability for the practical CDNNs, a surrogate is given to relax the derived nonlinear optimization as a linear programming problem (see Section 3.2). Moreover, the positivity of the optimum for the relaxed LP problem suffices to guarantee the robustness property of the given network (see Section 3.3).

3.1. From Robustness Verification to Nonlinear Optimization

Given a CDNN $\mathcal{N} : \langle L, X, W, B, \Phi \rangle$ with an input region Θ, recall that the robustness satisfaction is to verify whether the existence of the identical label ℓ such that each output holds the condition (7). As a result, ℓ is easily obtained by checking an arbitrary output, i.e., $\ell = \arg\max f(\mathbf{x}^{[0]})$, where $\mathbf{x}^{[0]}$ is a random input selected from Θ. Depending on the label ℓ, robustness verification by checking the condition (7) is equivalent to verify

$$\min\{\mathbf{x}_{\ell}^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]},\ \tilde{\ell} \neq \ell\} > 0 \tag{8}$$

is satisfied for each output $\mathbf{x}^{[n]}$. In other words, $\mathcal{N}$ with Θ is robust if and only if

$$\min_{\mathbf{x}^{[n]}}\{\, \min\{\mathbf{x}_{\ell}^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]},\ \tilde{\ell} \neq \ell\}\,\} > 0. \tag{9}$$

From (9), one can construct the following nonlinear optimization problem whose objective is a linear piece-wise function:

$$\left.\begin{array}{ll} p^* = \min_{\mathbf{x}^{[k]}, \mathbf{z}^{[k]}}\{\ \mathbf{x}_{\ell}^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]},\ \tilde{\ell} \neq \ell\ \} & \\ \text{s.t.}\ \ a_i \leq \mathbf{x}_i^{[0]} \leq b_i & i = 1, \cdots, s_0, \\ \quad \mathbf{z}^{[k]} = W^{[k]}\,\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]} & k = 1, \cdots, n, \\ \quad \mathbf{x}^{[k]} = \phi(\mathbf{z}^{[k]}) & k = 1, \cdots, n. \end{array}\right\} \tag{10}$$

The following theorem shows the equivalent transformation between robustness verification and nonlinear optimization solving.

Theorem 1 *Let $\mathcal{N}$ be a classification deep neural network, and Θ be the given input region. Suppose $\mathbf{y}^{[0]}$ is selected randomly from Θ, and ℓ is the label classified by $\mathcal{N}$ with $\mathbf{y}^{[0]}$, i.e., $\ell = \varrho(\mathbf{y}^{[0]})$. Suppose p^* is the optimum of the optimization problem (10), established by $\mathcal{N}$, Θ and ℓ. Then $\mathcal{N}$ with respect to Θ is robust if and only if $p^* > 0$.*

Proof 1 *Observe that the constraints of the optimization problem (10) are the equivalent expression of $\mathcal{N} : \mathbf{x}^{[n]} = f(\mathbf{x}^{[0]})$. Thus, taking the equivalence between (7) and (9) yields the desired result.* □

In fact, the optimization problem (10) tries to find the point that are most likely to be unrobust in the input region. If the worst point is robust, all the other points should be robust too.

3.2. Linear Encoding for Deep Neural Networks

The subsection considers how to encode the behavior in CDNNs in terms of linear constraints so that highly scalable LP solvers can contribute to verify CDNNs of practical size. The encoding is based on the input-output behavior of every neuron in the network, and the main challenge is to handle the non-linearities, which are arising from the activation functions. To facilitate such linear relaxation, we start by computing the lower and upper bounds of all neurons via layer-by-layer estimation, precedes linear inclusion for activation functions within the yielded bounds.

Bound Derivation with Interval Analysis Without loss of generality, we consider a single layer $L^{[k]}$. As shown in Section 2, it is known that the values of neurons, $\mathbf{x}^{[k]}$ in $L^{[k]}$ can be computed by

$$\mathbf{x}^{[k]} = \phi(W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}), \tag{11}$$

where ϕ is applied element-wise. By introducing the auxiliary variables $\mathbf{z}^{[k]}$, the evaluation (11) can be rewritten as the composition of a linear mapping

$$\mathbf{z}^{[k]} = W^{[k]}\,\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}, \tag{12}$$

and a nonlinear mapping

$$\mathbf{x}^{[k]} = \phi(\mathbf{z}^{[k]}). \tag{13}$$

Let $D^{[0]} = \Theta$ be the interval vector representing the bounds of the input neurons, and let $D^{[k]}, 1 \leq k \leq n$ be the interval vector representing the bounds of the k-th layer neurons. The bounds of all neurons can be obtained by performing the procedure of the layer-by-layer analysis. More specifically, suppose $D^{[k-1]}$ is yielded from the previous layer, using the interval computation to (12) yields the range of $\mathbf{z}_j^{[k]}$, denoted by $[\alpha_{k,j}, \beta_{k,j}]$ for each $j = 1, \cdots, s_k$. Since ϕ is a monotonically increasing function, the i-th element of $\mathbf{x}^{[k]}$ can be bounded by $\phi(\alpha_{k,i}) \leq \mathbf{x}_i^{[k]} \leq \phi(\beta_{k,i})$. As a result, we have $\mathbf{x}^{[k]} \subseteq D^{[k]}$, where $D^{[k]}$ is the following interval vector

$$D^{[k]} = [\phi(\alpha_{k,1}), \phi(\beta_{k,1})] \times \cdots \times [\phi(\alpha_{k,s_k}), \phi(\beta_{k,s_k})].$$

Linear Inclusion Let us explain how to compute linear inclusion for the activation function $y = \phi(x)$ within the domain $[a, b]$. Let us first choose the midpoint ξ of the interval $[a, b]$, and the Taylor expansion for $\phi(x)$ yields a linear approximation at the point ξ,

$$\phi(x) = p(x) + r(x),$$

where $p(x) = \phi(\xi) + \phi'(\xi)(x - \xi)$ is a linear approximation of $\phi(x)$ at the point ξ, and $r(x)$ is the error function. By exploring interval evaluation [23], we can obtain the inclusion, denoted by $[\underline{r}, \overline{r}]$, representing the range of $r(x)$ evaluated at the domain $[a, b]$.

To summarize, the procedure called *LI*, combining bound derivation with linear approximation, is applicable to linear inclusion for the activation function $\phi(x)$, namely,

$$\textit{LI:} \quad \phi(a) \leq \phi(x) \leq \phi(b),\ p(x) + \underline{r} \leq \phi(x) \leq p(x) + \overline{r}. \tag{14}$$

As mentioned above, suppose the range of $z_j^{[k]}$ is $[\alpha_{k,j}, \beta_{k,j}]$ produced from the previous layer. For each $1 \leq j \leq s_k$ calling *LI* procedure to $\mathbf{x}_j^{[k]} = \phi(\mathbf{z}_j^{[k]})$, yields the associated linear inequalities:

$$\phi(\alpha_{k,j}) \leq \mathbf{x}_j^{[k]} \leq \phi(\beta_{k,j}), \tag{15}$$

$$p_j^{[k]}(\mathbf{z}_j^{[k]}) + \underline{r}_{k,j} \leq \mathbf{x}_j^{[k]} \leq p_j^{[k]}(\mathbf{z}_j^{[k]}) + \overline{r}_{k,j}. \tag{16}$$

By calling *LI* procedure component-wise to the vector $\mathbf{x}^{[k]}$, the nonlinear constraint (13) can be relaxed to the following set of linear constraints

$$C_k : \begin{cases} \mathbf{x}^{[k]} \in D^{[k]}, \\ \mathbf{x}^{[k]} - p^{[k]}(\mathbf{z}^{[k]}) - \underline{\mathbf{r}}^{[k]} \geq 0, \\ \mathbf{x}^{[k]} - p^{[k]}(\mathbf{z}^{[k]}) - \overline{\mathbf{r}}^{[k]} \leq 0, \end{cases} \tag{17}$$

where $p^{[k]}(\mathbf{z}^{[k]}) = \phi(\xi^{[k]}) - \phi'(\xi^{[k]})(\mathbf{z}^{[k]} - \xi^{[k]})$ and $\xi^{[k]}$ is the midpoint vector for the range of $\mathbf{z}^{[k]}$, $\underline{\mathbf{r}}^{[k]} = (\underline{r}_{k,1}, \cdots, \underline{r}_{k,s_k})^T$, and $\overline{\mathbf{r}}^{[k]} = (\overline{r}_{k,1}, \cdots, \overline{r}_{k,s_k})^T$.

Using (12) and (17), we provide a way for linear encoding the input-output behavior of layer $L^{[k]}$. Thus, the set of linear constraints encoding the network $\mathcal{N}$ can be defined as: $C = \bigcup_{k=1}^{n} C_k$. In this situation, it follows from the above linear encoding that the equivalent optimization problem (10) for robustness verification can be relaxed into the following optimization problem:

$$\left.\begin{array}{ll} p_r^* = \min_{\mathbf{x}^{[k]}, \mathbf{z}^{[k]}} \{\ \mathbf{x}_\ell^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]},\ \tilde{\ell} \neq \ell\ \} & \\ \text{s.t.} \quad \mathbf{x}^{[k]} \in D^{[k]}, & k = 0, \cdots, n \\ \quad \mathbf{z}^{[k]} = W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}, & k = 1, \cdots, n, \\ \quad \mathbf{x}^{[k]} - p^{[k]}(\mathbf{z}^{[k]}) - \underline{\mathbf{r}}_k \geq 0, & k = 1, \cdots, n \\ \quad \mathbf{x}^{[k]} - p^{[k]}(\mathbf{z}^{[k]}) - \overline{\mathbf{r}}_k \leq 0, & k = 1, \cdots, n. \end{array}\right\} \tag{18}$$

Remark 1 *In comparison with the optimization problem (10), the feasible set of (18) is the superset of the one of (10), which results in the optimum of (18) is the lower bound of the optimum of (10), i.e., $p_r^* \leq p^*$.*

3.3. Transform to Linear Programming

The objective function of the optimization problem (18) is piecewise-linear. To make it amenable to LP solvers, it is transformed into an equivalent optimization problem by introducing an auxiliary variable t:

$$\left.\begin{array}{ll} p_r^* = \max\ t & \\ \text{s.t.} \quad \mathbf{x}_\ell^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]} \geq t, \quad \tilde{\ell} \neq \ell, & \\ \quad \mathbf{x}^{[i]} \in D^{[k]}, & k = 0, \cdots, n \\ \quad \mathbf{z}^{[k]} = W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}, & k = 1, \cdots, n \\ \quad \mathbf{x}^{[k]} - p^{[k]}(\mathbf{z}^{[k]}) - \underline{\mathbf{r}}_k \geq 0, & k = 1, \cdots, n \\ \quad \mathbf{x}^{[k]} - p^{[k]}(\mathbf{z}^{[k]}) - \overline{\mathbf{r}}_k \leq 0, & k = 1, \cdots, n \end{array}\right\} \tag{19}$$

Suppose p_r^* and $\tilde{p}_r^*$ are optimums of the optimization problem (18) and (19) respectively. Since $p_r^* = \min\{\mathbf{x}_\ell^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]}\}$, and $\mathbf{x}_\ell^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]} \geq \tilde{p}_r^*$, for all $\tilde{\ell} \neq \ell$, we have $p_r^* \geq \tilde{p}_r^*$. On the other hand, $\tilde{p}_r^*$ is the maximum value satisfying the set of constraints $\{\mathbf{x}_\ell^{[n]} - \mathbf{x}_{\tilde{\ell}}^{[n]} \geq t, \tilde{\ell} \neq \ell\}$, thus $\tilde{p}_r^* \geq p_r^*$. Combining the two facts, it follows that $p_r^* = \tilde{p}_r^*$.

The above transformation between the optimization problems derives a sufficient condition for robustness verification of CDNNs.

Theorem 2 *Given a CDNN $\mathcal{N} : \langle L, X, W, B, \Phi \rangle$ with the input region Θ. Suppose (19) is the linear programming problem established as above, and p_r^* is an optimum of (19). If $p_r^* > 0$, then $\mathcal{N}$ is robust.*

Proof 2 *For the given $\mathcal{N}$ with the input region Θ, one may establish the optimization problem (10). Due to the equivalence between (18) and (19), and in combination with Remark 1, p_r^* is the lower bound of the optimum of (10). Taking $p_r^* > 0$ into account, the optimum of (10) is positive. Finally, it follows from Theorem 1 that $\mathcal{N}$ is robust.* □

Theorem 2 guarantees when $p_r^* > 0$, $\mathcal{N}$ is robust with Θ. Consider the case $p_r^* <= 0$, and the corresponding worst point is $\tilde{\mathbf{x}}^{[0]}$, it is required to verify $\tilde{\mathbf{x}}^{[0]}$ using $\mathcal{N}$, denoted as $\varrho(\tilde{\mathbf{x}}^{[0]})$. If $\varrho(\tilde{\mathbf{x}}^{[0]}) \neq \ell$, that is $\tilde{\mathbf{x}}^{[0]}$ is misclassified, $\tilde{\mathbf{x}}^{[0]}$ provides a counterexample to show that $\mathcal{N}$ is not robust. But if $\varrho(\tilde{\mathbf{x}}^{[0]}) = \ell$, the robustness of $\mathcal{N}$ remains undetermined as we cannot predict the other bad points are robust or not.

To verify the robustness property of the given network $\mathcal{N}$, we have established a relaxed linear programming problem (19), which can be solved by using conventional algorithms such as the interior-point method [4]. Furthermore, the positivity of the optimum of (19) suffices to verify the robustness of $\mathcal{N}$. Detailed procedures are summarized in Algorithm 1.

Algorithm 1: *RobustVerifier* (Robustness verification for a CDNN)

Input: A CDNN $\mathcal{N}$; An input region Θ.
Output: bRobust; bUnrobust.

```
1  bRobust ← FALSE;  bUnrobust ← FALSE;
2  Determine the label ℓ:
3      Choose a random element x^[0] from Θ, i.e., x^[0] ∈ Θ;
4      ℓ ← ϱ(x^[0]);
5  D^[0] ← Θ;
6  C ← {};
7  for k = 1 : n do
8      D^[k] is computed from D^[k−1] by bound derivation;
9      C_k ← LinearEncoding(N, D^[k]);
10     C ← C ∪ C_k;
11 Construct the linear programming problem as in (19);
12 Call an LP solver to obtain the optimum p_r^* and its optimizer: (x̃^[k], z̃^[k]);
13 if p_r^* > 0 then
14     bRobust ← TRUE;
15 else
16     if ϱ(x̃^[0]) ≠ ℓ then
17         bUnrobust ← TRUE;
```

RobustVerifier takes as inputs a CDNN $\mathcal{N}$ with an input region Θ. In line 1, two Boolean flags are defined, i.e., bRobust and bUnrobust. The procedure first determines the label ℓ of the $\mathcal{N}$ by executing an element chosen from Θ, and then initializes the set of the linear constraints (line 5 and 6). Line 7 to 10 obtain the linear constraints approximating the activation functions. Line 11 constructs the corresponding linear programming, and its solution can be found by calling an LP solver (line 12).

RobustVerifier returns one of the following results: if bRobust is TRUE then $\mathcal{N}$ with Θ is robust (line 14), and if bUnrobust is TRUE then $\mathcal{N}$ with Θ is unrobust since two inputs $\mathbf{x}^{[0]}$ and $\tilde{\mathbf{x}}^{[0]}$ have the different label. Otherwise, *RobustVerifier* fails to determine whether $\mathcal{N}$ is robust.

Remark 2 *Suppose $D^{[n]}$ is the range of the output layer L_n obtained by bound derivation (line 8), denoted by $D^{[n]} = [l_1, u_1] \times \cdots [l_{s_n}, u_{s_n}]$. Once $l_\ell > u_{\tilde{\ell}}$ for each $\tilde{\ell} \neq \ell$, it is to say that $\mathcal{N}$ with the input region Θ has the same label ℓ. In this case,* Robustverifier *ends and bRobust is set to TRUE. Therefore, this simple method for computing the lower and upper bounds of the output layer, called as* RobustLU*, can also be used to verify the robustness property of $\mathcal{N}$ with Θ.*

4. Evaluation

In the evaluation, we use image classification networks generated from three popular image datasets designed for image classification, as target CDNNs for robustness verification. Those image classification networks include networks trained for classifying 10 classes of hand-written images of digits 0-9 from the database MNIST [20] , 43 classes of German traffic signs from the database (GTSRB) [30] , and 101 classes of images from the Caltech 101 dataset [21] . All the image classification networks are built from the neural networks library Keras [7] with a deep learning package Tensorflow [1] as its backend.

Details of the three datasets are listed below:

- *Modified National Institute of Standards and Technology database*. The MNIST database is a large collection of hand-written images designed for training various image processing systems in which each image is of size 28×28 and in black and white. It has 60,000 training input images, belonging to one of 10 labels, i.e. from the digit 0 to the digit 9.

- *German Traffic Signs Recognition Benchmark*. The GTSRB is a large image set of traffic signs devised for the single-image, multi-class classification problem. It consists of over 50,000 images of traffic signs, belonging to 43 different classes. In GTSRB, each image is of size 32×32 and has three channels (Red, Green and Blue).

- *Caltech-101 Dataset*. The Caltech-101 dataset consists of images belonging to 101 different classes where the number of images in each class varies from 40 to 800. Most images are of the resolution 300×200 pixel and have three channels.

In [24], the tool *NEVER* is proposed to verify a special type of neural networks with sigmoid activation functions, called Multi-Layer Perceptrons (MLPs). A MLP is a degeneration of CDNN as it restricts activation functions only to

be located on the output neurons. We generalized the algorithm to treat CDNNs, called *Generalized-NEVER* (*GNEVER*).

We have implemented the proposed algorithms *RobustVerifier*, *RobustLU* (see Remark 2) and *Generalized-NEVER* as MATLAB programs. The LP problems generated from robust verification are settled by the LP solver *linprog*. The following experimental results were obtained by running them on an Ubuntu 18.04 LTS computer with a 3.2GHz Intel(R) Xeon Gold 6146 processor and 128 GB RAM.

4.1. Performance

In the section, we evaluate the performance of the three tools by varied perturbations and network structures.

The CDNN trained from the MNIST dataset is used for performance evaluation whose input layer has 784 neurons and output layer consists of 10 neurons. For each image in MNIST, a set of pixels with a range of possible perturbation of these pixels is specified, which forms an input region Θ for the input layer.

Given a CDNN $\mathcal{N}$ and an input region Θ, robustness verification requires to check whether the robustness property holds with respect to Θ, i.e., all inputs within the input region Θ are assigned to the same label. Figure 1 shows a configuration of input region: a set of perturbations (with disturbance radius $\epsilon = 0.5$) to a block of size 5×5 on the top left corner of each image.

Figure 1. The MNIST CDNN is robust w.r.t. perturbed images.

Table 1 lists the comparison results of the three tools *RobustVerifier*, *RobustLU* and *GNEVER* with varied perturbations. Here, the input region is a block of size 5×5 on the top left corner of the image; ϵ denotes the disturbance radius, that is, $\epsilon = 0.2$ means perturbations varying within the range $[-0.2, 0.2]$ are imposed on the original image; r_1, r_2 and r_3 record the recognition rates of the tools *RobustVerifier*, *RobustLU* and *GNEVER*, respectively. Here, the recognition rate is defined as dividing the number of robust images plus adversarial examples reported by a specific tool by the total number of images under verification.

It can be shown from Table 1 that under all circumstances, the tool *RobustVerifier* outperforms the other two tools, while the tool *RobustLU* provides better performance than the tool *GNEVER*. For example, consider the case of $\epsilon = 0.2$, the correct recognition rates of three tools are

Table 1. Performance on varied perturbations

ϵ	$r1$	$r2$	$r3$
0.10	95.94%	92.56%	90.39%
0.15	92.16%	84.57%	82.00%
0.20	87.90%	83.88%	64.65%
0.25	79.43%	67.74%	65.78%
0.30	70.56%	65.32%	45.74%

87.90%, 83.88% and 64.65%, of the total 10,000 tests, respectively. And when ϵ grows up to 0.3, the correct recognition rates are reduced to 70.56%, 65.32% and 45.74%, respectively. The performance result complies with our discussion of approximation precision in descending order. For the same tool, seeing the results in the same column, the performance of verification tools decreases when the perturbations become bigger, since the increasement of perturbations speeds up the accumulation of approximation errors.

Table 2 lists the comparison results on varied network structures. Here, the input region is defined as a 5×5 block on the top left corner and the disturbance radius ϵ is fixed to 0.20; n denotes the number of layers in CDNNs and in each hidden layer the number s_k of neurons varies with $40 \leq s_k \leq 100$.

Table 2. Performance on varied network structure

n	$r1$	$r2$	$r3$
5	87.90%	83.88%	64.65%
6	72.33%	51.84%	50.24%
7	61.33%	43.11%	29.07%
8	50.58%	29.02%	27.79%
9	45.90%	33.28%	31.63%
10	34.65%	14.19%	12.40%

Table 2 shows that the tool *RobustVerifier* performs best in all cases and *RobustLU* performs better than *GNEVER*. And with the increasement of hidden layers, inspecting the results line by line, the performance of verification tools decreases. It is the approximation error accumulated layer by layer mainly responsible for performance decreasement. The tight approximation technique adopted helps our tool to be less affected.

4.2. Precision vs performance

In this subsection, we evaluate the three tools with larger CDNNs and investigate how does the precision affect the performance of verifying practical CDNNs.

For the CDNN generated from GTSRB, we specify the input region as a 3×3 block on the center of each image with the disturbance radius $\epsilon = 0.05$ on one of the RGB (Red Green Blue) channels.

For this configuration of input region, the tool *RobustVerifier* can verify that the perturbed images fall into the same class as their original ones while the other two cannot.

Figure 2. The GTSRB CDNN is robust w.r.t. perturbed images.

Figure 3. Adversarial examples for the GTSRB CDNN.

Figure 2 shows 4 pairs of original and perturbed images of the configuration that has been verified to be robust. Then we try to verify larger input region by enlarging the disturbance radius or the size of perturbation block. Unfortunately, neither of the two tools can verify the perturbed images to be robust under those configurations. For the input region, a 3×3 block put on the central with the disturbance radius $\epsilon = 0.1$ on one of the RGB channels, our tool can return many adversarial examples corresponding to a misclassification of perturbed images. Figure 3 presents some adversarial examples, of which the input region is set with disturbance radius $\epsilon = 0.1$ on one channel to the central block of size 3×3.

Compared with that of the MNIST database, the CDNN of GTSRB has to identify more classes, 43 vs 10. Thus, it is more sensitive to perturbations inborn. A verification method unable to provide enough precision does not work when encountering CDNNs of this kind.

We also conduct experiments on the CDNN yielded from the Caltech 101 dataset. The trained CDNNs have no less than 180,000 neurons, while the input region is set as a block of size 40×40 on the center of image with the disturbance radius $\epsilon = 0.5$. Several pairs of original and perturbed images are shown in Figure 4.

Figure 5 lists the result of performance vs network structure. Here, the X axis denotes the number of layers of CDNNs while the Y axis records the rate of successful verification. For each network structure, ten different configurations of the number of neurons are chosen for experiments where the number of neurons of each hidden layer is

Figure 4. The trained CDNN is robust w.r.t. perturbed images.

generated randomly from the range [40,150]. The average successful verification rates of ten experiments are shown in the figure.

For the tool GEVER, its performance varies between 23% and 81%. It drops sharply with the increase of hidden layers as the error introduced by its approximation propagates layer by layer very quickly. For all experiments, our tool *RobustVerifier* can give exact results to more than 82% disturbed images. Note that, the verification results depend not only on the specific verification method but also on the quality of CDNNs being verified. The CDNNs of 7 or 8 layers seems not to be as good as the others. In this experiment, our tool *RobustVerifier* performs best and shows its ability of treating real CDNNs.

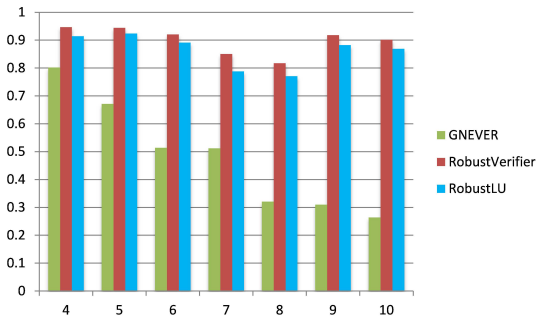

Figure 5. Performance vs network structure on Caltech-101.

5. Conclusion

We have presented a novel method for attacking robustness verification of classification deep neural networks with sigmoid activation functions. To make it amenable, we started with converting the verification problem into an equivalent nonlinear optimization problem, proceeded by solving a linear programming problem yielded from the linear relaxation for nonlinear activation functions. Our LP based approach has been implemented inside the tool *RobustVerifier* and validated on several state-of-the-art CDNNs for realistic images. The performance results highlight the potential of the approach in providing formal guarantees about the robustness of CDNNs of significant size.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[2] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Proceedings of the 29 Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2613–2621, 2016.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 387–402, 2013.

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

[5] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Piecewise linear neural network verification: A comparative study. *CoRR*, abs/1711.00455, 2017.

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

[7] François Chollet. Keras. `https://keras.io.`, 2015.

[8] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012.

[9] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *Proceedings of the 10th International Symposium on NASA Formal Methods (NFM)*, pages 121–138, 2018.

[10] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286, 2017.

[11] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, 107(3):481–508, 2018.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2011.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Kun Han, Dong Yu, and Ivan Tashev. Speech emotion recognition using deep neural network and extreme learning machine. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 223–227, 2014.

[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[16] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, pages 3–29, 2017.

[17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, pages 97–117, 2017.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1106–1114, 2012.

[19] Zeshan Kurd and Tim Kelly. Establishing safety criteria for artificial neural networks. In *Proceedings of the 7th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES)*, pages 163–169, 2003.

[20] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

[21] Fei-Fei Li, Robert Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[22] Awni Y. Hannun Maas, Andrew L. and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[23] Ramon Edgar Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. Cambridge University Press, 2009.

[24] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV)*, pages 243–257, 2010.

[25] Luca Pulina and Armando Tacchella. Challenging SMT solvers to verify neural networks. *AI Communications*, 25(2):117–135, 2012.

[26] Sebastian Ramos, Stefan K. Gehrig, Peter Pinggera, Uwe Franke, and Carsten Rother. Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pages 1025–1032, 2017.

[27] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In

Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN), pages 632–640, 2006.

[28] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[29] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *Proceedings of the 24th USENIX Security Symposium*, pages 611–626, 2015.

[30] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

[31] Changyin Sun, Wei He, Weiliang Ge, and Cheng Chang. Adaptive neural network control of biped robots. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(2):315–326, 2017.

[32] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR 2014)*, 2014.

[33] Vincent Tjeng and Russ Tedrake. Verifying neural networks with mixed integer programming. *CoRR*, abs/1711.07356, 2017.

[34] Tong Wang, Huijun Gao, and Jianbin Qiu. A combined adaptive neural network and nonlinear model predictive control for multirate networked industrial process control. *IEEE Transactions on Neural Networks and Learning Systems*, 27(2):416–425, 2016.

[35] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *CoRR*, abs/1712.08163, 2017.

[36] Weiming Xiang, Hoang-Dung Tran, , and Taylor T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 99:1–7, 2018.