T4V: Exploring Neural Network Architectures that Improve the Scalability of Neural Network Verification

Vivian Lin[1], Radoslav Ivanov[2⋆], James Weimer[1], Oleg Sokolsky[1], Insup Lee[1]

[1] PRECISE Center, University of Pennsylvania, Philadelphia, PA 19104, USA
{vilin,weimerj,sokolsky,lee}@seas.upenn.edu
[2] Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY, 12180
ivanor@rpi.edu

Abstract. This paper focuses on improving the scalability of NN verification through exploring which NN architectures lead to more scalable verification. We propose a general framework for incorporating verification scalability in the training process by identifying NN properties that improve verification and incentivizing these properties through a verification loss. One natural application of our method is robustness verification, especially using tools based on interval analysis, which have shown great promise in recent years. Specifically, we show that we can greatly reduce the approximation error of interval analysis by forcing all (or most) NNs to have the same sign. Finally, we provide an extensive evaluation on the MNIST and CIFAR-10 datasets in order to illustrate the benefit of training for verification.

1 Introduction

In the past several years, deep learning has shown great promise in traditionally challenging learning tasks such as image classification [38], natural language processing [5] and reinforcement learning [30]. Due to this impressive performance, neural networks (NNs) are also increasingly being used in safety-critical systems such as autonomous vehicles [2] and air traffic collision avoidance systems [21]. At the same time, however, researchers have discovered numerous robustness issues with NNs, e.g., adversarial examples where small input perturbations may lead to drastic changes in the NN outputs [37]. Since such issues might compromise the safety of NN-based systems, it is essential to verify the safety of NN components at design time, before these systems are deployed at large.

There has been a significant amount of work in the last few years on analyzing the safety of NNs [7,9,10,11,15,22,41,42,43,47]. However, although these methods employ a variety of techniques, scalability remains a major obstacle to applying any such method to realistic systems. For example, verifying even simple input-output properties such as interval constraints on the inputs and

⋆ Work was done while Radoslav was a postdoc fellow at the University of Pennsylvania.

outputs of fully-connected NNs with rectified linear units (ReLUs) is known to be NP-complete (exponential both in the number of inputs and the number of neurons in the NN) [22,34]. To get around this challenge, researchers have developed a number of useful heuristics such as combining interval analysis with linear programming [42] in order to make use of optimized solvers such as Gurobi [32].

As researchers continue to propose better tools through effective heuristics and implementations, an orthogonal approach to alleviating the scalability challenge is through exploring the properties of NN architectures that result in more scalable verification by existing tools and their corresponding heuristics [16,18]. However, there exists no formal general procedure of incorporating such properties into the training process and explicitly analyzing the trade-offs between verifiability (i.e., verification scalability) and the original property that the NN was trained for (e.g., classification accuracy).

In this paper, we propose to incorporate verifiability into the training process, through a method called *training for verification* (T4V). Such an approach can in principle be added to any training task. The high-level idea is as follows: suppose the original training loss, L_o, is designed to achieve a desired property, ϕ_o, such as classification accuracy and robustness; the goal is to identify an NN architecture property ϕ_v that results in more scalable verification and a corresponding loss L_v that can be added to L_o during training. Thus, by assigning different weights to L_o and L_v, one can explore the trade-off between verifiability and the original property and choose a desired setting.

To illustrate the process of T4V, we note that the verifiability property ϕ_v depends greatly on the specific verification approach. For example, as mentioned above, having a small Lipschitz constant is desirable for sampling-based methods [16]. In this work, we focus on approaches based on interval analysis since those tools have shown promising results in both open-loop [39,42,47] and closed-loop settings [18]. We have identified an important verifiability property related to interval analysis, namely that all (or most) NN weights in any layer should have the same sign. We show theoretically and through examples that if this property does not hold, then interval analysis can result in drastic overapproximation error. Finally, we propose a corresponding loss function, L_v, that promotes same-sign weights through penalizing negative weights and that is added to the original loss, L_o, through a weighted average.

We evaluate the proposed method by training a number of NN architectures on the MNIST [24] and CIFAR-10 [23] datasets where the goal is to verify robustness properties of the trained NN using the interval-analysis-based tool Fast-and-Complete [47]. We show that in all cases one might achieve significant improvements in verifiability at a small cost in robustness and accuracy. Eventually, an inflection point is reached after which the gains in verifiability are offset by prohibitive loss in robustness and accuracy. Although we perform this evaluation on robustness verification, our proposed T4V framework can also be used for verifying properties other than robustness. Some examples of these other applications can be found in Section 4.

In summary, this paper has three contributions: 1) a general framework for T4V that explicitly incorporates verifiability in the training process; 2) an illustration of the framework on the robustness verification problem; 3) extensive evaluation on the MNIST and CIFAR-10 datasets.

2 Related Work

A number of directions have been studied by existing work on the NN safety problem. One type of approach is to train NNs so that they are guaranteed to satisfy some safety property. For example, robustness certification techniques [13,29,33,44,45] aim to compute an upper bound on the robustness of a NN, then minimize this bound during training by incorporating it into the loss function. However, these works do not provide guarantees on unseen data. A related method [8] proposes predictor-verifier training (PVT), in which a predictor NN learns a task and a verifier network learns a robustness bound on the predictor NN. Although this approach empirically improves NN robustness to adversarial examples, the learned robustness bound only provides an approximation of the true bound. Finally, another area of interest has been in training NNs that are correct by construction [25,28]. However, these methods only provide guarantees for global properties and are unable to provide local robustness guarantees during training.

Due to the challenges in training provably robust NNs, the formal verification of NNs after training is an important alternate approach to solving the NN safety problem. Several techniques exist to analyze the input-output properties of NNs: 1) casting the problem as a satisfiability modulo theory (SMT) program [15,22] or a mixed-integer linear program (MILP) [7,39,42,47]; 2) computing reachable sets for the NN outputs using various abstractions such as polyhedra [35], zonotopes [11], or star sets [41]; 3) estimating the NN Lipschitz constant [10] or the distance to the classification boundary for a given image [43] through relaxed linear or convex optimization programs. Furthermore, approaches have been developed for verifying the safety of closed-loop systems with NN controllers [6,16,17,18,19,20,36,40] by combining some of the above ideas with classical hybrid system reachability methods [4,14]. Despite this impressive progress, existing verification tools struggle to scale to the large size of NNs typical in real-world problems [22,34].

In response to the scalability issues of existing verification tools, a third type of approach to the NN safety problem has emerged. Specifically, many papers leverage various heuristics to make the formal verification of large NNs more attainable. For example, the verification tool Neurify [42] combines interval analysis with linear programming, which can be efficiently solved by existing optimization tools (e.g., Gurobi [32]). Other prior work is motivated by the observation that certain NN architectures are easier to verify by their proposed tools [16,18], e.g., NNs with smaller Lipschitz constants are better for sampling-based methods [16]. These techniques, which are perhaps the most relevant predecessors to T4V, train NNs to adopt properties that make them more amenable to verification by existing tools. One paper [46] identifies a heuristic, namely that NNs with ReLU activations are easier to verify when their ReLUs are stable for all possible perturbations of an input, and encodes it into the loss function during training. Another method [31] penalizes an interval bound on the output of the NN in the loss function, which (although originally intended to improve training stability in interval bound propagation [13]) inherently reduces the overapproximation error of interval analysis verification tools. Despite these advances, our T4V framework is the first formal general procedure for training NNs to adopt these properties and explictly analyzing the trade-offs that result.

3 Preliminaries and Problem Statement

This section first introduces the standard approach to NN training and verification, with a focus on robustness verification tools. Finally, we state the T4V problem.

3.1 Standard NN Training

The standard classification setting can be summarized as follows: given a training dataset, $\mathcal{Z} = \{(x_0, y_0), \ldots, (x_N, y_N)\}$, of N examples $\{(x_0, \ldots, x_N\}$ (e.g., images) and corresponding labels $\{y_0, \ldots, y_N\}$, an NN f is selected from a family of NN architectures $\mathcal{F}$ so as to minimize some loss, $L_o : \mathcal{F} \times \mathcal{Z} \to \mathbb{R}$, such as negative log likelihood or least squares [12]. Typically, the family $\mathcal{F}$ is a parameterized set of NN architectures such that selecting f amounts to choosing the NN parameters, θ, that minimize L_o:

$$\min_{\theta} \sum_{i=1}^{N} L_o(f_\theta, (x_i, y_i)). \tag{1}$$

A large number of NN architectures have been proposed in the last several years, including fully-connected, convolutional, residual, etc. To simplify the presentation, in this paper we consider standard feedforward NNs, which includes fully-connected and convolutional NNs. Formally, a feedforward NN f can be represented as a composition of its M layers:

$$f(x) = f_M \circ f_{M-1} \circ \cdots \circ f_1(x), \tag{2}$$

where each layer $f_i(x) = \sigma_i(W_i x + b_i)$ performs a linear function with weight matrix W_i and biases b_i, followed by a non-linear activation σ_i, such as the ReLU: $\sigma_i(x) = \max(0, x)$. Note that the NN parameters $\theta := \{W_0, b_0, \ldots, W_M, b_M\}$ are identified during training as illustrated in (1). A classifier $f_c(x)$ can then be constructed from the neural network $f(x)$ by taking the argmax of its last layer:

$$f_c(x) = \text{argmax}_i \ f_i(x), \tag{3}$$

where $f_i(x)$ is the i^{th} component of $f(x)$.

3.2 NN Verification

As discussed in the introduction, a number of formal verification methods have been developed recently to analyze various properties of trained NNs. The vast majority of these techniques were proposed for robustness analysis, as motivated by the discovery of adversarial examples [37]. In this work, we also focus on robustness verification as the main property of interest, though our proposed T4V method can be applied to any other property as well.

Robustness Property Intuitively, an NN is robust if perturbing its inputs by a small amount leads to a correspondingly small change in the output. Since it is not feasible to compute robustness bounds for all inputs, existing methods focus on verifying robustness around a given example. Specifically, given an example x and a perturbation bound ϵ, the robustness verification problem is to establish whether there exists an ϵ-bounded perturbation of x that leads to a change in the label predicted by f_c Formally, given f_c, x, and ϵ, the robustness property is stated as follows:

$$\forall x' \in \mathcal{B}_\epsilon(x), f_c(x') = f_c(x), \tag{4}$$

where $\mathcal{B}_\epsilon(x) = \{x' \mid \|x' - x\|_\infty \leq \epsilon\}$ is an L_∞ ball around x.[3] Thus, the bigger the ϵ for which (4) holds, the more robust f_c is.

Verification Methods As noted in the introduction, verifying (4) is NP-complete for fully-connected NNs with ReLU activations, exponential both in the number of inputs (i.e., the dimension of x) and in the number of neurons in f [22,34]. To circumvent this limitation, a variety of heuristics have been proposed, ranging from casting the problem as a MILP [7] to performing reachability analysis [41]. In this work, we focus on methods based on interval analysis [39,42,47] since they have shown great promise in the last couple of years [1].

We now provide a high-level overview of the tools Neurify [42] and Fast-and-Complete (FAC) [47], which employ similar approaches in terms of combining interval analysis with linear programming. Intuitively, the ultimate goal of these methods is to cast (4) as a relaxed LP and use an optimized solver such as Gurobi. Since ReLUs are non-linear, one needs to obtain a linear relaxation of each ReLU neuron. Such a linear relaxation can be obtained by computing bounds on the inputs to each ReLU (using interval analysis) and then approximating the ReLU with a linear function over those bounds [42].

Once a linear relaxation of each ReLU is obtained, the entire LP is solved using Gurobi. If no counterexample is found, then the property is true. If a counterexample is found, one needs to check if it is a true or false positive: if it is a true positive, then the property is false; if the counterexample is a false positive, then the LP needs to be refined by splitting an overapproximated ReLU into its individual components and solving both resulting LPs (thus potentially leading to an exponential number of splits). Please consult prior work [42,47]

[3] We use L_∞ in the interest of clarity, though other norms can be used as well.

about various improvements in terms of which ReLUs to split first and how to obtain the tightest bounds $[l, u]$ for the input to each ReLU.

As noted above, this procedure has an exponential complexity in the worst case due to the possibility of having to split each ReLU in the NN. Thus, the run-time of such an approach can be greatly improved by tightening the bounds obtained using interval analysis, which is the goal of T4V.

3.3 Problem Statement

We now state the T4V problem, both in its general version as well as the specific instantiation related to robustness verification.

Problem 1 (Training for Verification). Suppose some base loss function L_o has been selected to achieve some property ϕ_o, e.g., classification accuracy, of an NN during training. Suppose also that a verification algorithm $\mathcal{A}$ is used to verify a given property ψ. The training for verification (T4V) problem is 1) to identify an NN property ϕ_v that results in more scalable verification of ψ by $\mathcal{A}$ and 2) incorporate ϕ_v into the NN during the training process using a corresponding loss function L_v.

Problem 2 (Training for Robustness Verification). The training for robustness verification problem is an instantiation of the T4V problem, where $\mathcal{A}$ is an interval-analysis-based verification algorithm [42,47] and ψ is a robustness property.

4 Training for Verification: High-Level Approach

This section provides our high-level approach to T4V. This method can in principle be applied to any training approach and any desired verification algorithm, although the specifics may vary as discussed below.

In a standard training setting, the NN parameters are selected so as to minimize some loss L_o, as discussed in Section 3. In classification problems, this loss is usually negative loss likelihood or cross entropy. If robustness is considered in addition to classification accuracy, then one can also add an additional term such as adversarial robustness [27] or interval bound propagation [13]. None of these methods consider verification during the training phase, however, since verification is usually performed post-hoc.

In this paper, we propose introducing verification considerations to the training process. Since NNs are usually highly overparameterized, there exist multiple NNs within the same family that achieve a similar loss L_o. Thus, it makes sense to choose an NN that not only achieves a low L_o but is also easy to verify. In order to do so, one needs to identify an NN property ϕ_v that leads to more scalable verification. The property ϕ_v is likely to be specific to the verification task and the tool being used, since different tools use different heuristics to tackle the scalability challenge.

Assuming for now that such a property ϕ_v is identified (examples are provided at the end of the section), the next task is to choose a corresponding loss function, L_v, that incentivizes this property during training. Given L_v, one can now modify the training loss by combining L_o and L_v, for example in a convex combination:

$$L = \alpha L_o + (1 - \alpha) L_v, \tag{5}$$

where $\alpha \in (0, 1)$. Thus, by varying α and observing the corresponding effect on L_o, the user can select the appropriate trade-off between verifiability and the original property. Note that in some cases, the loss L_v might even improve the original property, e.g., regularization is known to improve generalizability in classification tasks, especially with overparameterized models such as NNs [12]. We now provide two example properties ϕ_v and corresponding losses L_v.

Example 1 (Sampling-based Reachability Analysis). Reachability analysis is a useful technique in closed-loop verification, e.g., when NNs are used as controllers [16,18]. In this setting, the task is to compute the reachable set of outputs of the NN given a set of inputs. One way of approximating the reachable set is by constructing a polynomial approximation of the NN over the input set using polynomial regression [16]. In order to bound the error of the approximating polynomial, one could use a sampling-based method by making use of the NN's Lipschitz constant. Thus, NNs with lower Lipschitz constants would improve the scalability of the above method since fewer points would need to be sampled.

In the context of the proposed T4V method, the desired verifiability property ϕ_v is that the NN has a low Lipschitz constant. One way to incentivize this property is to introduce an L_2 regularizer on the NN weights W, i.e., $L_v = W^T W$. While prior work [16] has made the observation that lower Lipschitz constants improve verification scalability, we believe our proposed framework can make explicit the relationship between NN verifiability and performance as a controller. For example, for an NN used as an agent in a deep reinforcement learning problem, our framework can help understand the trade-off between NN verifiability and rewards earned by the NN agent.

Example 2 (Taylor-model-based Reachability Analysis). An alternative method to reachability analysis is to construct a polynomial approximation with error bounds (i.e., a Taylor model) for each neuron in the NN [19,20]. This method makes use of interval analysis to propagate the error bounds through the NN. Thus, the error bounds can grow quickly, especially in large NNs. One way to reduce the error growth, as discussed also in Section 5 in the context of robustness verification, is to ensure that all weights in a given layer have the same sign.

Thus, the property ϕ_v is that all weights are positive, without loss of generality. One way of promoting this property is through introducing a higher penalty on negative weights, explained further in Section 5.

The remainder of this paper considers in more detail another example of T4V, namely for the case of robustness verification using interval analysis.

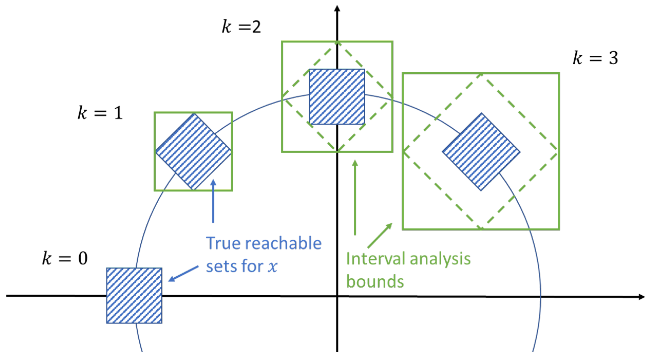


Fig. 1: Illustration of the growth of the approximation error incurred by interval analysis on the harmonic oscillator system.

5 Training for Robustness Verification

This section presents an illustration of the proposed T4V method, for the case of robustness verification using interval-analysis-based tools. We first identify a property ϕ_v that results in smaller approximation error due to interval analysis. We then introduce the corresponding verifiability loss, L_v, and show how to combine it with the original loss, L_o.

5.1 On the benefit of same-sign weights for interval analysis

As discussed in Section 3, interval analysis is a major component of state-of-the-art robustness verification tools. At the same time, interval analysis can introduce significant approximation error, especially in high-dimensional settings. Prior work [26] has discussed multiple reasons for the error growth, such as rotations, ill-conditioned matrices, etc.

To illustrate one such case of approximation error growth, consider the harmonic oscillator dynamical system [26]:

$$x_{k+1} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} x_k, \tag{6}$$

where $\phi = 45°$, and $x_0 \in [-1-\epsilon, -1+\epsilon] \times [-\epsilon, \epsilon]$. In this example, the initial set for x_0 is a box with size 2ϵ (ϵ can take on any positive value). With each step $k = 1, 2, 3, \ldots$, the reachable set for x_k is rotated by 45°. If one were to use interval analysis to approximate the reachable set, the approximation error would grow quickly over time, as shown in Figure 1, due to the rotation present in the harmonic oscillator system.

The harmonic oscillator example is instructive because an NN can be considered a dynamical system, where the neurons at layer k are x_k. Thus, for the purposes of analyzing error growth, we can view the NN description in (2) as a linear system of the sort:[4]

$$x_{k+1} = W_{k+1}x_k, \tag{7}$$

where $k = 0, \dots, M$, x_0 is the input to the NN, and W_k are the individual layer weights. Note that we ignore the bias terms b_k because they do not affect the shape of the sets.

With this intuition in mind, we can choose a property for each W_k and enforce it during training. Unfortunately, most properties that result in lower approximation error [26] are difficult to enforce as they would significantly constrain the NN architecture: e.g., eliminating rotations would mean that we a priori fix the directions of all rows of each W_k and only train their magnitudes. Hence, we have identified a new property that can be more easily incentivized during training as a soft constraint. Namely, interval analysis approximation error is reduced when all (or most) entries of each W_k have the same sign. The benefit of this property is illustrated in the following theorem.

Before stating the theorem, we introduce some notation relevant to interval analysis. Let $x \in [l, u]$, i.e., x lies in the (potentially multidimensional) interval $[l, u]$. When clear from context, we use the shorthand notation $[x] := [l, u]$. We say a matrix $A \in \mathbb{R}^{m \times n}$ is a *same-sign matrix* if all entries of A have the same sign. As a reminder, for a scalar constant $a \geq 0$, $a \times [l, u] = [al, au]$; when $a < 0$, $a \times [l, u] = [au, al]$. Finally, when $A \in \mathbb{R}^{m \times n}$ with rows $a_1, \dots a_m$ and $x \in \mathbb{R}^n$, we use the shorthand notation $A[x]$ to mean the interval vector $[y]$ where the first component is the interval $[l_1, u_1] = a_1^T[x]$, and so on.

Theorem 1. *Let $x \in [x]$ and let $A = A_M \dots A_1$ be the product of M matrices with compatible dimensions. If all A_i are same-sign matrices, then the set $A[x]$ is the same as the set $A_M \dots [A_1[x]]$.*

Proof. We prove the claim by induction on the number of matrices M. The claim is trivially true for $M = 1$.

Suppose that the claim is true for k, i.e., $A[x] = A_k \dots [A_1[x]]$. Now consider A_{k+1}. We need to show that $A_{k+1}[A[x]] = (A_{k+1}A)[x]$.

Let $[x] = [a, b]$, with a and b vectors. Note that all entries of each A and A_{k+1} must have the same sign, so we can write $A = \text{sign}(A)|A|$ and $A_{k+1} = \text{sign}(A_{k+1})|A_{k+1}|$, where $|A|$ denotes the element-wise absolute value of A.

Note that since $|A|$ has non-negative values, $[|A|[x]] = [|A|a, |A|b]$, which implies $A_{k+1}[A[x]] = \text{sign}(A)\text{sign}(A_{k+1})|A_{k+1}|[|A|a, |A|b]$. Similarly, since $|A_{k+1}|$ has non-negative values, $|A_{k+1}|[|A|a, |A|b] = [|A_{k+1}| \cdot |A|a, |A_{k+1}| \cdot |A|b]$.

Conversely, since $A_{k+1}A = \text{sign}(A_{k+1})\text{sign}(A)|A_{k+1}| \cdot |A|$, then $(|A_{k+1}| \cdot |A|)[x] = [|A_{k+1}| \cdot |A|a, |A_{k+1}| \cdot |A|b]$, which proves the claim. ■

Theorem 1 means that if we have a sequence of same-sign matrices $A_1, \dots, A_M$ applied to an interval vector $[x]$, using interval approximation after each multiplication is the same as premultiplying the matrices and using interval analysis

[4] Note that the ReLU activations may sometimes reduce the approximation error since all negative values are mapped to 0. However, they do not add any further error, so analyzing only the linear parts of the NN is an important first step.

only once. This is important because in an NN we cannot premultiply all layer weights due to the presence of activations. Thus, Theorem 1 means that if all weights have the same sign, then applying interval analysis at each layer does not result in drastic approximation error growth. Note that if the conditions of Theorem 1 do not hold, the error could grow substantially, as shown in the following example.

Example 3. Let $x \in [[-0.01, 0.01], [-0.01, 0.01]]^\top$. Let $v_1 = [1, 1]^\top, v_2 = [3, 3]^\top, v_3 = [5, -3]^\top$. Suppose we want to compute $v_3^\top [v_1\ v_2]^\top [x]$.

Note that $(v_3^\top [v_1\ v_2]^\top)[x] = [-4\ \ -4][x] = [-0.08, 0.08]$. On the other hand, if we apply interval analysis sequentially, we get
$v_3^\top [[v_1\ v_2]^\top [x]] = v_3^\top [[-0.02, 0.02], [-0.06, 0.06]]^\top = [-0.28, 0.28]$, which is a significantly larger interval.

5.2 Incentivizing Same-Sign Weights During Training

There are multiple ways to achieve the same-sign property during the training process. One approach would be to use constrained optimization and train the NN under the constraint that all weights in a layer have the same sign. However, such a constraint might be too limiting and might significantly affect training performance. Thus, we choose instead to relax these constraints using mechanisms inspired by the common Lagrangian relaxations from optimization [3]. Specifically, we incorporate our same-sign weights constraint directly in the loss through a larger penalty on negative weights. While it may be possible to achieve better training results using a scheme where some layers have only negative weights and others have only positive weights, we leave this analysis for future work. Specifically, the verification loss has the form:

$$L_v = \sum_{i=1}^{M} \gamma \|W_i^n\| + (1-\gamma)\|W_i^p\|, \tag{8}$$

where $W_i^n = \min(0, W_i), W_i^p = \max(0, W_i)$, $\| \cdot \|$ denotes the Frobenius norm, and $\gamma \in [0, 1]$ is a hyperparameter that determines how big the penalty on negative weights should be. Note that computing W_i^p and W_i^n requires a max and min function, similar to the ReLU implementation. We also emphasize that our verification loss L_v, when incorporated with the original L_o loss via (5), is reminiscent of the standard weight norm penalizations used in machine learning. Such regularization techniques have been shown to improve generalizability [12], and although we have repurposed them to achieve an imbalance of weight distribution (weighted more heavily towards negative weights or positive weights), our loss L_v is likely to improve generalization over a non-regularized NN. This is also observed in our experiments.

The choice of γ in (8) depends on the specific training task. Since NNs are overparameterized for many benchmarks such as MNIST and CIFAR-10, NNs can be well trained even with a γ that is close to 1. However, as the complexity of the training task increases, it is possible that even small deviations of γ past

0.5 may significantly hurt training performance. In our experiments, we chose $\gamma = 0.9$ since we were able to train NNs with very small negative weights while maintaining high accuracy.

Finally, note that a more effective strategy may be to also assign different weights to different layers, so as to recognize their relative importance. In particular, larger layers, both in terms of the number of inputs and number of neurons, result in larger approximation error when using interval analysis (interval analysis does not maintain relationships between variables, which accrues additional error in higher dimensions). Hence, it is natural to introduce a larger penalty for larger layers. Furthermore, earlier layers are more important in terms of error growth since the error is magnified as the analysis progresses through the following layers. Since a formal analysis of these factors is not straightforward, we assign the same weight to all layers and leave this investigation for future work.

6 Experiments

This section provides an evaluation of the proposed T4V method on the MNIST [24] and CIFAR-10 datasets [23]. MNIST is a dataset of 28×28 grayscale images of handwritten digits, and CIFAR-10 is a dataset of $3 \times 32 \times 32$ color images of 10 classes of objects such as airplane, automobile, etc. We use the FAC tool [47] in the verification evaluation, which is described at a high level in Section 3.[5]

6.1 Experimental Setup

We illustrate the benefit of our approach in two different scenarios, one in which the original loss is purely classification-oriented, and one in which the original loss promotes both classification accuracy and robustness. For each scenario, we evaluate the effect of T4V on NN verifiability. We also evaluate NN robustness, as the T4V process could improve verifiability by reducing robustness (a non-robust network can be trivially proven unsafe via counterexample). Our results alleviate such concerns. We show that in both scenarios, adding the verification loss L_v improves verifiability, often at little or no cost in accuracy and robustness.

Scenario 1: Classification accuracy In the first scenario, the original loss is cross-entropy loss, denoted by L_{CE}, which is a standard loss in classification tasks [12]. In the interest of space, we omit the formal definition of cross-entropy; intuitively, for each example x, L_{CE} tries to minimize the difference between the NN's output label "distribution" (e.g., a softmax layer) and the true label's distribution (given as a one-hot encoding, for example). Thus, the final T4V loss in this case is:

$$L := \alpha L_{CE} + (1 - \alpha) L_v. \tag{9}$$

[5] All code used to produce the experiments can be found at `https://github.com/vwlin/T4V`

Scenario 2: Classification accuracy and robustness In the second scenario, the original loss enforces not only classification accuracy but also robustness. This scenario illustrates two points: 1) the proposed method is general and can be used for a variety of original losses; 2) since the proposed L_v loss naturally improves robustness over a non-regularized network (as discussed in Section 5), we show that it can improve verifiability even in settings where the NN is already quite robust.

There are a number of existing methods that are aimed at improving NN robustness such as adversarial training [27] and interval bound propagation (IBP) [13]. In this work, we use IBP since it performs fairly well on the MNIST and CIFAR-10 datasets; we leave the evaluation over other robustness losses for future work. At a high level, IBP works as follows: for a given batch of B training examples $x_1, \ldots, x_B$, an interval of size $2\epsilon_{IBP}$ is created around each point; then we propagate the intervals $[x_1 - \epsilon_{IBP}, x_1 + \epsilon_{IBP}], \ldots, [x_B - \epsilon_{IBP}, x_B + \epsilon_{IBP}]$ through the NN using interval analysis; the size of the output interval is used as the IBP loss, L_{IBP}. Note that, in addition to improved robustness, IBP also alleviates the approximation error of interval analysis. Thus, the final loss becomes

$$L := \alpha(\beta L_{CE} + (1-\beta)L_{IBP}) + (1-\alpha)L_v, \tag{10}$$

where β is a hyperparameter regulating the relative weight of L_{IBP}.

6.2 Implementation Details

For our evaluation on MNIST, we train three different fully-connected NN architectures of increasing size in order to illustrate the benefit of our approach: 1) an NN with 2 hidden layers and 50 neurons per hidden layer; 2) an NN with 2 hidden layers and 200 neurons per hidden layer; and 3) an NN with 5 hidden layers and 200 neurons per hidden layer. Without regularization, these NNs achieve classification accuracy of 95.6%, 95.8%, and 97.5%, respectively (as averaged over 3 trials with different random seeds).

To train the NNs for Scenario 2 on MNIST, for each NN we used values of β and ϵ_{IBP} that do not greatly affect the original L_{CE} loss; the specific values are $\epsilon_{IBP} = 0.04$ (normalized from the pixel range $[0, 255]$ to $[0, 1]$), and $\beta = 0.84, \beta = 0.96, \beta = 0.97$ for the three NNs, respectively. Note that bigger values of β are needed for larger NNs since interval analysis can result in large output intervals, as discussed in Section 5.

For our evaluation on CIFAR-10, we note that CIFAR-10 is a significantly harder dataset than MNIST, both due to the number of dimensions and to the richness of the images. In addition, since FAC does not support regularization such as dropout and batch normalization, it is challenging to train NNs with very high accuracy that can be encoded in the tool. Thus, we use the following architecture (as inspired by the CIFAR-Base model in the original FAC paper [47]): 1) a convolutional layer of 8 filters (with a kernel size of 4 and stride of 1); 2) a convolutional layer of 16 filters (with a kernel size of 4 and stride of 1); and

3) a fully-connected layer of 128 neurons. Note that when expanded, the convolutional layers have 6728 and 10816 neurons, respectively, which is significantly larger than the NNs used for MNIST. Without regularization, this NN achieves an average (over 3 trials) classification accuracy of 73.38%.

For both scenarios and datasets, to evaluate verification scalability, we randomly select 200 images from the test set and run the FAC tool with a selected perturbation size ϵ on each NN for each correctly classified image. An image is said to be verified if the verification completed within a timeout of 60s for the MNIST dataset and 120s for the CIFAR-10 dataset. We define verifiability to be the fraction of images that were verified by FAC. We evaluate robustness with respect to the same safety property that we verify with FAC (i.e., that an ϵ perturbation will not cause a change in classifier decision). We first ϵ-perturb 1000 randomly selected images from the test set (a superset of the 200 images used to evaluate verifiability) using the PGD attack [27]. We then measure robustness as the fraction of images for which the classification decision of the NN did not change after the attack. Note that this robustness measure is an overestimate of the NN's true robustness, as the PGD attack is not guaranteed to find the optimal adversarial perturbation for each image. Thus, for a closer estimate of robustness, we use 1000 images rather than 200 images. Finally, to obtain informative evaluations of verifiability and robustness, we select the perturbation ϵ (the same ϵ is used for both evaluations) so that the robustness verification task is neither overly trivial (ϵ too small) nor insurmountably difficult (ϵ too large) for the FAC tool to complete on each NN.

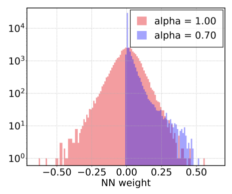
6.3 Distribution of Weights

For a subset of the NNs trained on MNIST for Scenarios 1 and 2, the distribution of learned weights are shown in Figure 2. We find that without our verification loss L_v (i.e., when $\alpha = 1.00$), the weights roughly follow a bell-shaped distribution centered at 0. In contrast, after training with L_v (i.e., when $\alpha < 1$), all negative weights are of nearly zero magnitude. For our NNs trained on CIFAR-10 for Scenarios 1 and 2, we show the distribution of learned weights in Figure 3. Just as with the NNs trained on MNIST, the addition of our verification loss L_v greatly reduces the magnitude of negative weights.

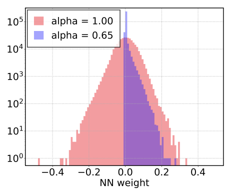
6.4 MNIST Evaluation

The MNIST evaluation on Scenario 1 is shown in Figure 4.[6] We observe that, as we decrease α (i.e., we assign more weight to L_v) verifiability increases significantly for each setup, reaching as high as 90% for the two-layer NNs. Also note that the increase in verifiability usually comes at the expense of some drop in accuracy (more pronounced in the five-layer NN). At the same time, for values of α very close to 1, one can obtain significant benefits in verifiability at

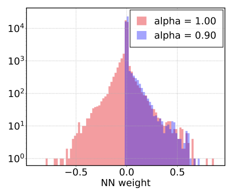
[6] For assessing the magnitude of ϵ, note that image pixels can only take discrete values between 0 and 255.



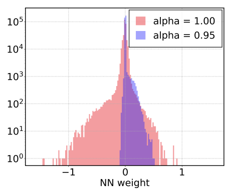
(a) 2 layers, 50 neurons per layer, Scenario 1.



(b) 4 layers, 200 neurons per layer, Scenario 1.

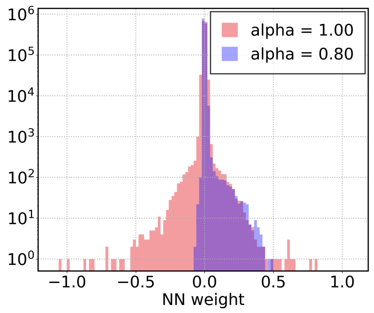


(c) 2 layers, 50 neurons per layer, Scenario 2.

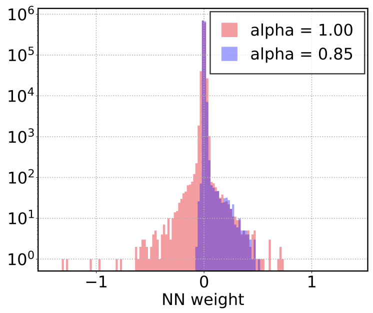


(d) 4 layers, 200 neurons per layer, Scenario 2.

Fig. 2: MNIST weight distributions on Scenarios 1 and 2. Each histogram is over a single trial. The vertical axes are on a log scale.



(a) Scenario 1.

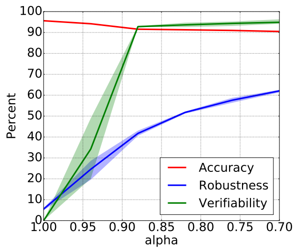


(b) Scenario 2.

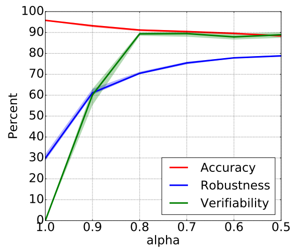
Fig. 3: CIFAR-10 weight distributions on Scenarios 1 and 2. Each histogram is over a single trial. The vertical axes are on a log scale.

only minor costs in accuracy. Finally, the proposed L_v also naturally improves robustness. However, the improvements in robustness and verifiability are not always correlated, as discussed next.

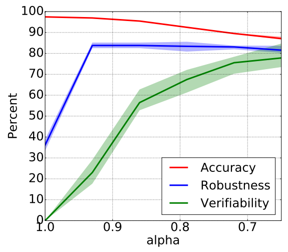
The evaluation on Scenario 2 is shown in Figure 5. Once again, the verifiability improves significantly as we decrease α. Note that, due to the addition of L_{IBP}, these NNs are much more robust (for larger ϵ) to input perturbations. In this case, the increase in verifiability comes at some cost in both accuracy and robustness, with the degree of this trade-off tunable by the hyperparameter α just as in Scenario 1. However, for the five-layer NN, the suitable range of α is much narrower than for the smaller networks. Due to the larger size of the five-layer NN, it is increasingly difficult as α decreases to balance the competing objectives of accuracy, low IBP loss, and low verification loss. The result of this complex loss is great variance across seeds when α is reduced beyond a certain threshold (analyzing the reasons for this variance, e.g., the importance of layer weighting in L_v, is left for future work).



(a) 2 layers, 50 neurons per layer, $\epsilon = 20$.

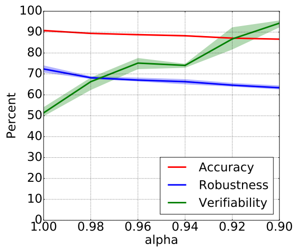


(b) 2 layers, 200 neurons per layer, $\epsilon = 15$.

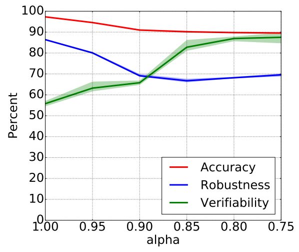


(c) 5 layers, 200 neurons per layer, $\epsilon = 8$.

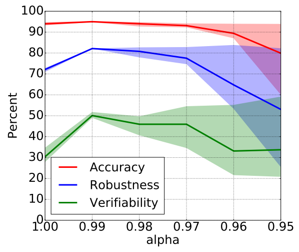
Fig. 4: MNIST evaluation on Scenario 1. All curves are averaged over 3 trials. Shaded regions indicate min/max outcomes for each setup.



(a) 2 layers, 50 neurons per layer, $\epsilon = 25$.



(b) 2 layers, 200 neurons per layer, $\epsilon = 20$.

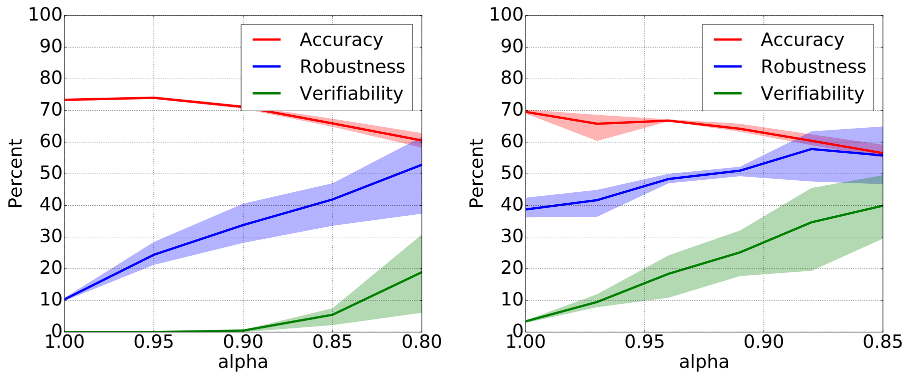


(c) 5 layers, 200 neurons per layer, $\epsilon = 15$.

Fig. 5: MNIST evaluation on Scenario 2. All curves are averaged over 3 trials. Shaded regions indicate min/max outcomes for each setup.

6.5 CIFAR-10 Evaluation

The CIFAR-10 evaluation is shown in Figure 6, where the verifiability definition is the same as in the case of MNIST (with a timeout per image of 120s). We observe the same overall trends as in the MNIST case – adding the L_v loss improves verifiability significantly, at some cost in accuracy (though robustness is improved in both scenarios). Note that there is more variance across seeds in the CIFAR-10 evaluation, most likely due to the challenging dataset and the larger size of the NNs. Finally, note that the robustness benefits of L_{IBP} are less pronounced than in the MNIST case – this is consistent with prior work where achieving robustness has been shown to be significantly harder for color images such as those in CIFAR-10 [27]. As part of future work, we will explore whether using other robustness losses leads to a better robustness/verifiability combination on CIFAR-10.



(a) Scenario 1. (b) Scenario 2.

Fig. 6: CIFAR-10 evaluation on the two scenarios, $\epsilon = 3$. All curves are averaged over 3 trials. Shaded regions indicate min/max outcomes for each setup.

7 Discussion and Future Work

This paper presented an approach to training for verification. We proposed a way of incorporating a verification loss into the training process, thus significantly improving verification scalability. An evaluation was provided on MNIST and CIFAR-10 illustrating the generality and effectiveness of this technique.

Since T4V is a new area, there are a number of interesting directions for future work. As shown in Section 6, the proposed method introduces greater variance during the training process for larger NNs, so an important question is whether verifiability is fundamentally at odds with accuracy or whether it is simply a matter of finding the right local optimum during training. As a first step, we intend to investigate the effect on the training process of assigning different values of the γ hyperparameter at each layer.

Another interesting direction is to apply this technique in closed-loop settings. Since in closed-loop verification one needs to perform reasoning over multiple time steps, an extra challenge in this problem is that if the NN controller is not robust, verification is likely to fail even if the NN itself is "verifiable". Finally, we intend to apply the proposed method to other losses and verification tasks (e.g., semantic robustness) in order to investigate its benefits and limitations.

Acknowledgements This work was supported in part by DARPA/AFRL FA8750-18-C-0090 and ARO W911NF-20-1-0080. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Air Force Research Laboratory (AFRL), the Army Research Office (ARO), the Defense Advanced Research Projects

Agency (DARPA), the Department of Defense, or the United States Government.

References

1. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (vnn-comp 2021): Summary and results. arXiv preprint arXiv:2109.00498 (2021)
2. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
3. Boyd, S., Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
4. Chen, X., Abraham, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd. pp. 183–192. IEEE (2012)
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of machine learning research **12**(Aug), 2493–2537 (2011)
6. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: 22nd International Conference on Hybrid Systems: Computation and Control. pp. 157–168 (2019)
7. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods Symposium. pp. 121–138. Springer (2018)
8. Dvijotham, K., Gowal, S., Stanforth, R., Arandjelovic, R., O'Donoghue, B., Uesato, J., Kohli, P.: Training verified learners with learned verifiers. arXiv preprint arXiv:1805.10265 (2018)
9. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 269–286. Springer (2017)
10. Fazlyab, M., Robey, A., Hassani, H., Morari, M., Pappas, G.: Efficient and accurate estimation of lipschitz constants for deep neural networks. Advances in Neural Information Processing Systems **32** (2019)
11. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: Safety and robustness certification of neural networks with abstract interpretation. In: Security and Privacy (SP), 2018 IEEE Symposium on (2018)
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
13. Gowal, S., Dvijotham, K.D., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., Kohli, P.: Scalable verified training for provably robust image classification. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4842–4851 (2019)
14. Henzinger, T.A.: The theory of hybrid automata. In: Verification of digital and hybrid systems, pp. 265–292. Springer (2000)
15. Henzinger, T.A., Lechner, M., Zikelic, D.: Scalable verification of quantized neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35 (2021)
16. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–22 (2019)

17. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Case study: Verifying the safety of an autonomous racing car with a neural network controller. In: International Conference on Hybrid Systems: Computation and Control (2020)
18. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: 22nd ACM International Conference on Hybrid Systems: Computation and Control (2019)
19. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G., Lee, I.: Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In: International Conference on Computer Aided Verification. pp. 249–262. Springer (2021)
20. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1) (Dec 2020). https://doi.org/10.1145/3419742
21. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th. pp. 1–10. IEEE (2016)
22. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
23. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Master's thesis, University of Toronto (2009)
24. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
25. Lin, X., Zhu, H., Samanta, R., Jagannathan, S.: Art: abstraction refinement-guided training for provably correct neural networks. In: Formal Methods in Computer-Aided Design (2020)
26. Lohner, R.J.: On the ubiquity of the wrapping effect in the computation of error bounds. In: Perspectives on enclosure methods, pp. 201–216. Springer (2001)
27. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018)
28. Mell, S., Brown, O., Goodwin, J., Son, S.H.: Safe predictors for enforcing input-output specifications. arXiv preprint arXiv:2001.11062 (2020)
29. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: International Conference on Machine Learning. pp. 3578–3586. PMLR (2018)
30. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
31. Morawiecki, P., Spurek, P., Śmieja, M., Tabor, J.: Fast and stable interval bounds propagation for training verifiably robust models. In: European Symposium on Artificial Neural Networks (2020)
32. Optimization, G.: Gurobi optimizer, https://gurobi.com
33. Raghunathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: International Conference on Learning Representations (2018)
34. Sälzer, M., Lange, M.: Reachability is np-complete even for the simplest neural networks. In: International Conference on Reachability Problems. pp. 149–164. Springer (2021)
35. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)

36. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 147–156. ACM (2019)
37. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)
38. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1701–1708 (2014)
39. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2019)
40. Tran, H., Cai, F., Lopez, D.M., Musau, P., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Transactions on Embedded Computing Systems **18**(5s), 105 (2019)
41. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International Conference on Computer Aided Verification. pp. 3–17. Springer (2020)
42. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems. pp. 6367–6377 (2018)
43. Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning. pp. 5273–5282 (2018)
44. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International Conference on Machine Learning. pp. 5286–5295. PMLR (2018)
45. Wong, E., Schmidt, F., Metzen, J.H., Kolter, J.Z.: Scaling provable adversarial defenses. Advances in Neural Information Processing Systems **31** (2018)
46. Xiao, K.Y., Tjeng, V., Shafiullah, N.M., Madry, A.: Training for faster adversarial robustness verification via inducing relu stability. In: International Conference on Learning Representations (2019)
47. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2020)