Training for Verification: Increasing Neuron Stability to Scale DNN Verification

Dong Xu[1(✉)], Nusrat Jahan Mozumder[1], Hai Duong[2], and Matthew B. Dwyer[1]

[1] University of Virginia, Charlottesville, VA 22904, USA
{dx3yy,nm8tm,matthewbdwyer}@virginia.edu
[2] George Mason University, Fairfax, VA 22030, USA
hduong22@gmu.edu

Abstract. With the growing use of deep neural networks(DNN) in mission and safety-critical applications, there is an increasing interest in DNN verification. Unfortunately, increasingly complex network structures, non-linear behavior, and high-dimensional input spaces combine to make DNN verification computationally challenging. Despite tremendous advances, DNN verifiers are still challenged to scale to large verification problems. In this work, we explore how the number of stable neurons under the precondition of a specification gives rise to verification complexity. We examine prior work on the problem, adapt it, and develop several novel approaches to increase stability. We demonstrate that neuron stability can be increased substantially without compromising model accuracy and this yields a multi-fold improvement in DNN verifier performance.

Keywords: neural network verification · neuron stability · pruning

1 Introduction

In recent years, there has been significant research on adapting formal verification to target deep neural network(DNN) model behavior. Approaches have been developed that incorporate a diverse range of algorithmic approaches including reachability [19,27,39–42,45,51,52], optimization [5,12,15,30,31,34,44,50], and search [1,7,9,21,26,46,47,49,58]. These techniques aim to verify the validity of a network's behavior for a wide range of inputs, e.g., perturbations of test samples that capture models of noise or malicious manipulation.

DNN verification is challenging due to the high input dimension of models, the ever-growing complexity of network layers, the inherent non-linearity of learned function approximations, and the algorithmically complex methods required to formulate the verification problem [25]. Several approaches [4,14,16,38] have been proposed to address the scalability issue, but as the results of recent DNN verifier competitions show scalability remains a challenge [2,22,32].

Stable neurons exhibit linear behavior and thereby have the potential to reduce DNN verification costs. Several researchers have explored how DNNs

B. Finkbeiner and L. Kovács (Eds.): TACAS 2024, LNCS 14572, pp. 24–44, 2024.
https://doi.org/10.1007/978-3-031-57256-2_2

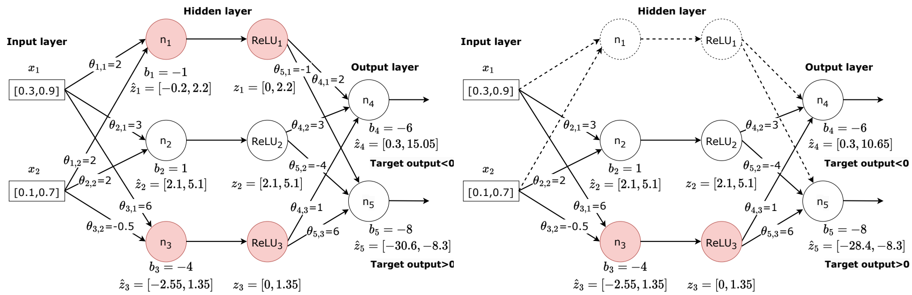
can be defined to increase the number of stable neurons and thereby facilitate verification. For example, one can incorporate a loss term that uses an estimate of neuron stability to train a network that can be verified more efficiently [53]. Another training time approach identifies neurons that are likely to be stable and active and replaces them with linear functions [10], while this approach requires customization of the verifier to show performance improvement.

Whereas prior work studied individual methods for increasing neuron stability in combination with individual verifiers, in this paper we conduct a broad exploratory study considering 18 different stabilizers paired with 3 state-of-the-art verifiers across DNNs for different datasets and comprising different architectures. We use three algorithmic approaches to increase stability: RS Loss [53] incorporates a stability-oriented loss term, Bias Shaping is a novel training time method that only modifies bias parameters to increase stability, and Stable Pruning is a novel approach that adapts structural DNN pruning [43] to increase stability. These are paired with *stability estimation* algorithms that operate at training time to guide them towards increasing stability. We develop 4 estimators based on prior work: **NIP** [53], **SIP** [46,47], **ALR** [56], and **ALRo** [57], and 2 novel estimators **SDD** and **SAD**.

Neuron instability can be a source of verification complexity for the two primary algorithmic approaches to DNN verification: abstraction-based methods and constraint-based methods. Abstraction-based verifiers [3, 17, 40, 42, 48] overapproximate neuron behavior, but when the approximation is too coarse – due to unstable neurons – the approximations must be refined which can slow down verification. Constraint-based verifiers [13, 23, 24, 44] are challenged by the disjunctive nature of constraints that encode unstable neurons. Orthogonal to these approaches, branch and bound techniques [9, 17, 48] are also sensitive to neuron stability since they need to generate sub-problems for each of the active phases of unstable neurons. In our exploratory study, we evaluate the performance of verifiers that span several of these algorithmic approaches and that also constitute the state-of-the-art based on their performance in the most recent VNN-COMP [32]. This allows us to assess the extent to which increasing neuron stability can improve the state-of-the-art.

In § 5 we report the findings of a study spanning 18 stable training algorithms, 3 state-of-the-art verifiers, 3 network architectures, and a large number of challenging property specifications. Our primary finding is that stable training can significantly increase the number of verifications problem solved – by as much as 5-fold – and significantly speed up verification – by as much as a factor of 14 – without compromising test accuracy or training time. Moreover, we find that if one is willing to tolerate a modest loss in test accuracy, then even greater improvement in verifier performance can be achieved.

The contributions of the work lie in a comprehensive evaluation of the potential for optimizing DNN verifier performance by increasing the number of stable neurons. More specifically, (1) we adapt RS Loss with different stability estimators and evaluate its performance across multiple verifiers and benchmarks; (2) we propose two novel approaches (Bias Shaping and Stable Pruning) to



(a) Original network (b) Stable Pruning

Fig. 1. Illustration of the applying Stable Pruning to verifying that a small original network outputs a pair of values where the first is negative and the second positive for inputs $\boldsymbol{x} \in [0.3, 0.9] \times [0.1, 0.7]$. Unstable neurons are shown in red and pruned neurons and their edges are dashed.

increase neuron stability and evaluate their performance across multiple verifiers and benchmarks; (3) we integrate these state-of-the-art neuron stabilizers into an open-source framework that supports experimentation with stability optimization by the DNN verification research community; and (4) show empirically that the performance of state-of-the-art verifiers can be significantly enhanced using stable training methods. These contributions set the stage for further work on *training for verification* that aim to further characterize the best stable training strategy for a given verifier and verification problem.

2 Overview

The popularity of the rectified linear unit (ReLU) activation function, $z = \max(\hat{z}, 0)$, which allows for more efficient training and inference [20, 29], has led verification researchers to target networks using them. In this section, we illustrate how ReLU leads to exponential verification costs and how training can mitigate that cost.

For a DNN with ReLU activation functions, $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, comprised of k neurons, an inference, $\mathcal{N}(\boldsymbol{x})$, results in each neuron being either *active*, when $z = \max(\hat{z}, 0) = \hat{z}$, or *inactive*, when $z = \max(\hat{z}, 0) = 0$. The status of each neuron in a network during inference defines an activation pattern, $ap(\boldsymbol{x})$ – a Boolean vector of length k. Verifying a set of inputs, $\phi_{\boldsymbol{x}} \subseteq \mathbb{R}^n$, involves symbolically reasoning about the set of activation patterns, and the associated neuron outputs, for each $\boldsymbol{x} \in \phi_{\boldsymbol{x}}$. In the worst case, there are 2^k possible activation patterns which lead to the exponential complexity of ReLU verification [23].

For a given set of inputs, $\phi_{\boldsymbol{x}}$, a neuron, n_i, is *stable* and active if $\forall \boldsymbol{x} \in \phi_{\boldsymbol{x}} : ap(\boldsymbol{x})[i]$, and stable and inactive if $\forall \boldsymbol{x} \in \phi_{\boldsymbol{x}} : \neg ap(\boldsymbol{x})[i]$. A neuron's stability is dependent on the computation performed by its cone of influence [6] taking into account both $\phi_{\boldsymbol{x}}$ and the behavior of neurons on which n_i depends. In Fig. 1a, consider verification of a local robustness property centered at $\boldsymbol{x} = (0.6, 0.4)$ with a radius of $\epsilon = 0.3$ – so $\phi_{\boldsymbol{x}} = [0.3, 0.9] \times [0.1, 0.7]$. For such inputs, a single neuron, n_2, is stable – its pre-activation values are all positive, $\hat{z}_2 = [2.1, 5.1]$.

In §4, we define a set of techniques that aim to estimate which neurons are unstable during training and then bias the training process to stabilize them. Fig. 1b shows the application of one pair of those techniques to the original network and property. More specifically, the **NIP** estimator propagates interval approximations of neuron pre-activation values to estimate whether they are stable and then the STABLE PRUNING technique removes neurons that are stable and inactive. During training this method estimates the pre-activation value for n_1 to be $\hat{z}_1 = [-0.2, 2.2]$ which is nearly stable. STABLE PRUNING ranks neurons based on the distance they need to be shifted to be stable; for $\hat{z}_1$ that distance is 0.2. We adapt the iterative pruning approach of DropNet [43] to use this ranking. The intuition is that when a neuron is nearly stable it can be removed and in subsequent training, the parameters of the remaining neurons will adapt to compensate and preserve accuracy [18]. As illustrated in Fig. 1b, the number of unstable neurons is halved which can reduce verification costs.

3 Background & Related Work

Deep Neural Networks (DNN) are trained to accurately approximate a target function, $f : \mathbb{R}^n \to \mathbb{R}^m$. A network, $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, is comprised of a sequence of L hidden layers, $l_1, \ldots, l_L$, along with an input layer, $l_{in} = l_0$, and output layer, $l_{out} = l_{L+1}$(e.g. (a) in Fig. 1) Hidden layers are comprised of a set of *neurons* that accumulate a weighted sum of their inputs from the prior layer and then apply an *activation function* to determine how to non-linearly scale that sum to compute the output from the layer. Different activation functions have been explored in the literature, including: Rectified Linear Units (ReLU), Sigmoid, and Tanh.

Given a neural network architecture, $\mathcal{N}(\cdot)$, the network is *trained* to define *weight* values, denoted θ, and *bias* values, denoted b, that are associated with each neuron's input. A trained network defines for input $\boldsymbol{x}$, the output $\mathcal{N}(\boldsymbol{x}; \theta, b)$; when it is clear from the context we drop θ, b and write $\mathcal{N}(\boldsymbol{x})$.

Specifying DNN Properties Given a network $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, a property, ϕ, defines a set of constraints over the inputs, $\phi_{\boldsymbol{x}}$, and an associated set of constraints over the outputs, ϕ_y. Verification of $\mathcal{N} \models \phi$ seeks to prove: $\forall \boldsymbol{x} \in \mathbb{R}^n : \phi_{\boldsymbol{x}}(\boldsymbol{x}) \Rightarrow \phi_y(\mathcal{N}(\boldsymbol{x}))$.

Recent work has demonstrated that a general class of specifications, where $\phi_{\boldsymbol{x}}$ and $\phi_{\boldsymbol{y}}$ are defined as half-space polytopes, can be reduced to local robustness specifications [35, 36]. This means that the essential complexity of DNN verification is present when verifying simpler local robustness specifications, which state that $\forall \boldsymbol{x} \in c \pm \epsilon : \phi_y(\mathcal{N}(\boldsymbol{x}))$, for some constant input(centerpoint), c, and radius, ϵ, around it. Consequently, in §5, we explore the performance of verifiers on local robustness specifications.

Verifying DNN Properties The inherent complexity of the DNN verification problem arises from the non-linear expressive power of DNNs – so it is generally unavoidable. We explain the source of this complexity below for a network with L fully-connected layers, each with M neurons.

Let $\hat{z}_{i,j}$ denote the value computed for the input of neuron j in hidden layer i prior to the application of the activation function – the pre-activation value – and $z_{i,j}$ the post-activation value. For a ReLU activation function, $z_{i,j} = \max(\hat{z}_{i,j}, 0)$. The input to layer i is computed as the weighted sum of the output of the prior layer, using the learned weights θ, and bias b. The semantics of $\mathcal{N}(\boldsymbol{x}; \theta, b)$ is given by the constraints as shown in Eq. (1)

$$\bigwedge_{i \in [1,L], j \in [1,M]} \left(\hat{z}_{i,j} = \sum_{k \in [1,M]} (\theta_{i,j,k} \cdot z_{i-1,j}) + b_{i,j} \wedge z_{i,j} = \max(\hat{z}_{i,j}, 0) \right) \tag{1}$$

with additional constraints relating the $z_{L,j}$ to the output layer, l_{out}, and $\boldsymbol{x} = z_0$.

Computing $\mathcal{N}(\boldsymbol{x})$ for a single input value, $\boldsymbol{x}$, results in a pattern of ReLU activations in which each neuron is either *active*, $\max(\hat{z}_{i,j}, 0) = \hat{z}_{i,j}$, or *inactive*, $\max(\hat{z}_{i,j}, 0) = 0$. However, a property specification, ϕ, constrains l_{in} to define a set of input values, e.g., as in the case of local robustness $\boldsymbol{x} \in c \pm \epsilon$. Through Eq. (1), this may give rise to constraints on $\hat{z}_{i,j}$ that define values for which the neuron is both active, $\hat{z}_{i,j} \geq 0$, and inactive, $\hat{z}_{i,j} < 0$. When the set of pre-activation values spans 0 in this way, we say that neuron $n_{i,j}$ is *unstable*.

Unstable neurons require that verification approaches reason about the disjunctions present in Eq. (1). In the worst case, if all neurons are unstable, then there are 2^{L*M} different ways of resolving the disjunctions. More generally, for a property, ϕ, only a subset of neurons will be unstable, $U_\phi \subseteq L \times M$, and, as we discuss in §4, controlling the size of this subset is a means of reducing the cost of DNN verification.

Several approaches have been introduced to verify a DNN behavior in recent years [28]. One class of verifiers, including α,β-CROWN [48], NNEnum [3], ERAN [40], and MN-BAB [17] overapproximate ReLU behavior which allows them to efficiently calculate an overapproximation of Eq. (1), which we denote $\overline{\mathcal{N}}$. When $\overline{\mathcal{N}} \not\models \phi$ some techniques, like ERAN, simply return *unknown*, but others, like NNEnum, α,β-CROWN or MN-BAB, perform a case split on unstable neurons to refine the over-approximation. Another class of verifiers, including Marabou [24] and Planet [13], explore the space of case-splits to formulate separate constraint queries that constitute verification conditions. Here again, the number of possible case-splits leads to exponential complexity.

RS Loss [53] is a regularization technique that induces neuron stability in the training process. The RS Loss, L_R is blended with the regular training loss L_T to yield a weighted sum as the optimization target, $L = L_T + w_R \times L_R$, where w_R is the hyperparameter to control the degree of stabilization. The RS Loss term L_R is formulated as $L_R = \sum_{i=1}^{n} -\text{Tanh}(1 + \underline{\hat{z}_i} \times \overline{\hat{z}_i})$ where $\underline{\hat{z}}$ and $\overline{\hat{z}}$ are the lower and upper bounds of the pre-activation values. NRS Loss [59] is a variant of RS Loss that regularizes the pre-batch normalization (BN) bounds instead of pre-activation bounds. Whereas RS Loss indirectly biases the network toward neuron stability, in §4 we introduce Bias Shaping which directly manipulates neuron bias towards the same goal.

DropNet [43] is a structured model compression method to generate sparse and reduced neural networks based on the *lottery ticket hypothesis* [18]. According to the hypothesis, a dense network contains a sub-network that can match the test accuracy of the base network if trained in isolation. DropNet iteratively prunes a predefined percentage of less important neurons by setting their weights to zero. Although the iteration process is resource expensive, the flatness of the error landscape at the end of training limits the fraction of weights that can be pruned, hence sharp pruning at once reduces the network accuracy [33].

While the initial purpose of pruning was preserving network accuracy only, recent studies have revealed that pruning can significantly increase a network's robustness and scale robustness verification [59]. The removal of non-linearity from the insignificant neurons by converting them to linear functions has been proposed in literature [10]. However, the existence of linear activation functions in a network can sometimes result in unnecessary computational costs, as the networks are supposed to work on complex data and linear functions are incapable of handling the complexity. Also, special treatments are required to handle these non-standard architectures in network inference and verification. Thus, we propose to use iterative pruning to remove the redundant non-linearity from the network using the pre-activation values of the ReLU function during mini-batch training. In §4, we present a variant of DropNet named Stable Pruning that uses stability measures to determine how neurons should be pruned.

4 Approach

This section presents the two novel neuron stabilization methods: Bias Shaping and Stable Pruning, as well as six different stability estimators. Alg. 1 shows the general training iterations for a neural network with stabilizers(pairs of stabilization method, $\mathcal{A}$, and stability estimator, $\mathcal{B}$). The conventional neural network training process of a mini-batch is shown in Line 2.

Alg. 1: Training with Stabilizers

input : neural network $\mathcal{N}$, data loader D, stabilization method $\mathcal{A}$, stability estimator $\mathcal{B}$, ratio i, and step s

output : stabilized network $\mathcal{N}'$

1 **for** j, (X, Y) *in* D **do**

2   Train_Mini-Batch($\mathcal{N}, X, Y$)

3   **if** $j \equiv 0 \pmod{s}$ **then**

4         $\hat{Z} \leftarrow$ Estimate_Stability($\mathcal{B}$, $\mathcal{N}$)

5         $\mathcal{N}' \leftarrow$ Stablize($\mathcal{A}$, $\mathcal{N}$, $\hat{Z}$, i)

6 **return** $\mathcal{N}'$

Stabilizers are applied at every sth mini-batch (line 3). Line 4 determines each neuron's stability estimation by calculating their boundaries, $\hat{Z}$, using different estimators described in §4.1. Lastly, Line 5 applies the main stabilization algorithms, e.g. Bias Shaping (Alg. 2) and Stable Pruning (Alg. 3).

4.1 Neuron Stability Estimation

The neural network training process is performed on the data samples, while the verification process seeks to prove certain properties on an effectively unbounded set of inputs. Hence, there exists a gap between the two stages since a neuron that is stable on the training dataset is not guaranteed to also be stable based on the set of values described by the precondition of the verification problem. Guiding the training process to produce neural networks with more stable neurons in the verification stage requires reducing this gap. This is achieved by estimating neuron stability over a broader set of values representative of those encountered during the verification process and then stabilizing the unstable neurons.

We identify two general categories of neuron stability estimators that can be calculated during the training phase: **Sampled[S]** and **Reachability[R]** estimators. The sampled estimators consider a finite set of sampled data gathered directly or inferred from the training dataset. The reachability estimators operate on set propagations that generalize the training dataset. The six neuron stability estimators are defined as follows:

$$\mathcal{B}(D) = \{x | x = \beta(x') \wedge x' \sim D\}$$

where $\beta \in \{\mathbf{SDD}, \mathbf{SAD}, \mathbf{NIP}, \mathbf{SIP}, \mathbf{ALR}, \mathbf{ALRo}\}$ and D is the network training dataset distribution. The **SDD** (Sampled Dataset Distribution[**S**]) estimator uses the training mini-batch samples directly and takes advantage of the training process's forward propagations to determine whether neurons are stable. The **SAD** (Sampled Adjacent Distribution[**S**]) estimator samples from the robustness radii of the training mini-batch and runs extra forward propagations on the adjacent examples to determine the stability of neurons. The **NIP** (Naive Interval Propagation[**R**]) [53] estimator generates a set of intervals based on the mini-batch samples and the given robustness radii. However, instead of propagating exact samples, it propagates the intervals through the network. The **SIP** (Symbolic Interval Propagation[**R**]) [46, 47] extends **NIP** by using symbolic intervals instead of concrete intervals when propagating through the network. The symbolic intervals are concretized whenever neuron stability needs to be evaluated. The **ALR** and **ALRo** (Auto_LiRPA[**R**]) [56, 57] estimators further improve **SIP** by applying more precise but computationally expensive over-approximation constraints and parameterizing upper and lower bounds of hidden neurons to optimize objectives with respect to the property of interest. **ALRo** applies the α optimization [57] when compared to the base approach. Note that although many of these approaches were developed for other uses, the integration of them to induce stable neurons during training is novel.

4.2 Bias Shaping

To increase the number of stable neurons in the neural network, we adapt training to ensure the same polarity of lower and upper bounds of neuron pre-activation values. In Eq. (1), the pre-activations of the current ReLU function are controlled by the parameters of the neural network and the post-activations of the previous layer. The weighted-sum term depends on the weights, bias, and the post-activations of the previous layer. The pre-activation values can be easily manipulated by changing the bias term. We refer to this as BIAS SHAPING, as described in Alg. 2.

Alg. 2: BIAS SHAPING

---

input : neural network $\mathcal{N}$, stability estimation boundaries $\hat{Z}$, ratio i
output : stabilized network $\mathcal{N}'$

1 $\underline{\hat{Z}}, \overline{\hat{Z}} \leftarrow$ GET_BOUNDS($\hat{Z}$)
2 $N_u \leftarrow \{n_i \text{ in } \mathcal{N} \text{ where } \underline{\hat{z}}_i < 0 \wedge \bar{\hat{z}}_i > 0\}$
3 $Z_u \leftarrow \{\textsc{Min}(-\underline{\hat{z}_{n_i}}, \overline{\hat{z}_{n_i}}) \text{ where } n_i \in N_u\}$
4 $\gamma \leftarrow \textsc{Sort}(Z_u)[|\hat{Z}| \times i]$
5 **for** n_i *in* N_u **do**
6   **if** $(\overline{\hat{z}_i} < \gamma) \wedge (\overline{\hat{z}_i} < -\underline{\hat{z}_i})$ **then**
7         $n_i.b \leftarrow n_i.b - \overline{\hat{z}_i}$
8   **else if** $|\underline{\hat{z}_i}| < \gamma$ **then**
9         $n_i.b \leftarrow n_i.b - \underline{\hat{z}_i}$
10 $\mathcal{N}' \leftarrow$ LOAD_PARAMETERS(N_u)
11 **return** $\mathcal{N}'$

Instead of using just the native pre-activation of the mini-batch samples, the stability estimators are applied to further close the gap between neuron stability during training and verification. Alg. 2 takes the set of stability estimations for all neurons, $\hat{Z} = [\hat{z}_1, \hat{z}_2, ..., \hat{z}_m]$, the neural network $\mathcal{N}$ with m neurons $(n_1, n_2, ..., n_m)$, and the ratio i as inputs. Line 1 calculated the lower and upper bounds of the estimation $\hat{Z}$. Using those bounds, the algorithm first finds the unstable neurons of the input network (line 2). Next, those neurons are ranked based on their distance to zero(lines 5 - 9), and the smallest subset of neurons will be selected for shaping if their distances are less than an adaptive threshold γ (lines 3, 4). Note that the number of selections is controlled by a parameter i – a percentage of neurons would be shaped at a time. Each neuron's bias term of the subset is modified by (a) shifting left by the value of the upper bound if the upper bound is closer to zero (line 7); or (b) shifting right by the absolute value of lower bound if the lower bound is closer to zero (line 9). As a result, the stabilized network is created by loading the new parameters at line 10.

4.3 Stable Pruning

Inspired by the DropNet [43] approach, we developed a new pruning method to reduce unstable neurons, named STABLE PRUNING as shown in Alg. 3. It uses iterative structured pruning to modify the global weight matrix by selectively masking neurons. Its novel criteria target specifically unstable neurons for masking. STABLE PRUNING sets weight and bias to zero to softly "remove" the neuron from the network, allowing back-propagation to recover accuracy loss by the harsh parameter modifications.

Given the stability estimation $\hat{z}$ for a neuron, $\underline{\hat{z}}$ and $\bar{\hat{z}}$ denote the lower and upper bounds respectively. When lower bound $\underline{\hat{z}}$ is greater than 0, although the neuron is stable-active, it cannot be pruned without changing the network's behavior, as the ReLU function is treated as an identity function. When $\bar{\hat{z}}$ is less than 0, the ReLU function is treated as a zero-function, and this neuron can be removed safely (line 3). In order to prune unstable neurons with minimal effects on network behavior, Stable Pruning ranks the unstable neurons by the distance between $\bar{\hat{z}}$ and 0, from smallest to largest (line 4), and a subset of neurons (also controlled by the ratio parameter, i) will be selected for pruning if their distances are less than an adaptive threshold γ (line 5). Initially, all neurons are enabled in the mask, m, (line 1) and those that fall below the threshold are updated to be removed from the network (line 6). Finally, the stabilized network is generated by applying the pruning mask on the network (line 7).

4.4 Implementation

Alg. 3: Stable Pruning

input : neural network $\mathcal{N}$, stability estimation boundaries $\hat{Z}$, ratio i
output : stabilized network $\mathcal{N}'$

1 $m = \{1\}^{|n|}$
2 $\bar{\hat{Z}} \leftarrow$ Get_Upper_Bound($\hat{Z}$)
3 $m[\bar{\hat{Z}} \leq 0] \leftarrow 0$
4 $Z'_u = \mathrm{sort}(\bar{\hat{Z}} > 0)$
5 $\gamma = Z_u[|Z'_u| \times i]$
6 $m[\hat{Z} < \gamma] \leftarrow 0$
7 $\mathcal{N}' \leftarrow \mathcal{N} \bigodot m$
8 **return** $\mathcal{N}'$

We implemented all of the above techniques, including: **SDD**, **SAD**, **NIP**, **SIP**, **ALR**, **ALRo**, RS Loss (§3), Bias Shaping (§4.2), and Stable Pruning (§4.3), into the OCTOPUS framework. OCTOPUS allows training neural networks with stabilizer methods and stability estimators, including their free combinations. It can be easily applied to different datasets and network architectures and presents a rich hyper-parameter space that can be tuned by hand or algorithmically, e.g., by search methods. RS Loss [53] is reimplemented to support all the additional neuron stability estimators. The **SIP** estimator uses the Symbolic Interval Analysis Library developed in [46], and the **ALR** and **ALRo** estimators integrate the Auto_LiRPA Library [57]. OCTOPUS also allows combinations of various neuron stabilizers and estimators, i.e., training with multiple stabilizers sequentially or simultaneously. The framework is built for ease of extension to adopt new techniques and is available at both FigShare [54] and GitHub.[3]

5 Evaluation

We explore two research questions to understand how stabilizers can be beneficial for DNN verification:
RQ1. How effective are the stabilizers in increasing the proportion of stable neurons?
RQ2. How effective are stabilizers in enhancing DNN verification performance?

[3] OCTOPUS GitHub link: `https://github.com/edwardxu0/octopus`

Tab. 1. Experimental parameter space

Parameters	Choices
Architectures	**M2**: MNIST_FC2(FC(256)×2), **M6**: MNIST_FC6(FC(256)×6)) **C3**: CIFAR2020(Conv(32,5,2), Conv(128,4,2), FC(250))
Verifiers	α,β-CROWN, MN-BAB, NNEnum
Properties	[0,1,. . . ,9]
Epsilon Radii	**M2**, **M6**:[12e-3, 14e-3, 16e-3, 18e-3, 20e-3] **C3**:[18e-4, 20e-4,22e-4, 24e-4, 26e-4]
Stabilization Methods	Baseline, **B**ias Shaping, **R**S Loss, Stable **P**runing
Stability Estimators	**SDD**, **SAD**, **NIP**, **SIP**, **ALR**, **ALRo**
Seeds	[0,1,2,3,4]

5.1 Study Design

To answer these questions, we design a broad study considering different neural network architectures, specifications, and verifiers. Tab. 1 shows the full experimental parameter space we consider across the research questions.

The annual VNN-COMP DNN verification competition [2, 22, 32] provides a range of benchmarks with standard network and property formats to evaluate state-of-the-art verifiers. These benchmarks cover a variety of network architectures and activation functions. This *architectural variety* evaluates verifiers' applicability across a range of network graph operations, e.g. ResNets with skip connections, max-pooling layers, non-linear activation, and domain-specific networks. Benchmarks also vary in *scale* with some having large numbers of layers, neurons, and parameters under the assumption that this will yield challenging benchmarks.

We conducted an exploratory study of the VNN-COMP 2022 benchmarks and found that 1156 of 1288 (89%) could be solved within 30 seconds. Nearly all of the solved problems were proven (UNSAT) with coarse over-approximation or falsified (SAT) with adversarial attacks. Such benchmarks do not exhibit the exponential complexity that is inherent in DNN verification [23]. To address this limitation, we designed a set of benchmarks that are better suited to assessing DNN verification algorithm performance.

Selecting Networks A retrospective analysis of VNN-COMP benchmarks determined that small weakly-regularized networks exhibit exponential complexity and medium-sized with large numbers of neurons are hard to scale for precise methods, such as branch and bound [8]. Of course, large weekly-regularized networks with large numbers of neurons are even harder, but it was found that these incur significant memory requirements which makes experimentation challenging, e.g., due to hardware limitations. Based on this analysis, we focus on three small and medium-sized networks with traditional network architectures selected from the VNN-COMP 2022 benchmarks, since these proved capable of forcing verifier algorithms to cope with exponential complexity.

Selecting Properties Rather than focusing on a variety of structurally distinct property specifications, we exploit the fact that general reachability proper-

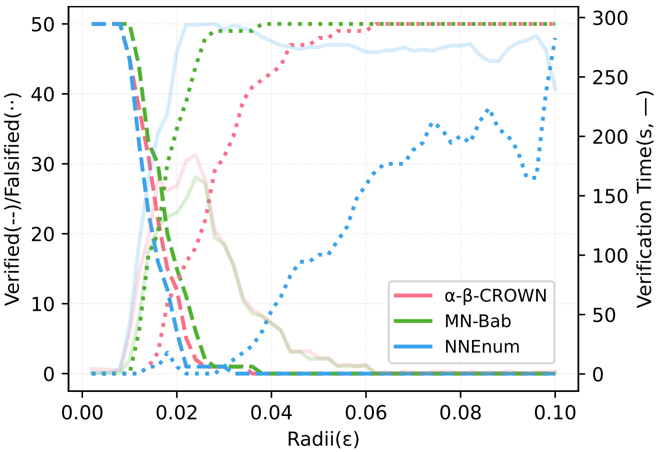


Fig. 2. Solved problems and verification time vs. epsilon radii

ties can be reduced to local robustness properties [37]. This allows us to vary the verification problem difficulty by controlling the robustness property's epsilon-radius. Conceptually, we know that verification problems with sufficiently small (large) radii will be verified (falsified) – a radius of 0 is trivially verified and a radius comprising the full input domain requires that a network produce a constant output. Verifier developers have incorporated techniques, like applying adversarial attacks and using coarse overapproximations, to quickly handle such cases [3, 48]. To sidestep these verification fast paths and exercise the core verification algorithms in our study, we select epsilon values for properties as follows.

For each network, we conducted a preliminary study with varying radii to assess the difficulty of the verification problems. Fig. 2 shows the results for **M2** on 50 different center-points with the three verifiers. The dashed lines show the number of verified problems and the dotted lines the number of falsified problems (left y-axis). We observe the trend that small epsilon leads to uniformly verified problems and large epsilon to uniformly falsified problems. Moreover, one can observe low verification times (right y-axis) in these extreme epsilon regimes, due to the fast path optimizations.

Our strategy for selecting harder verification properties is to choose a sample of radii around the point where the number of verified and falsified problems crossover, e.g., 0.018 in this plot for MN-BAB. We choose the crossover point of the best verifier who solved the most problems to design the radii shown in Tab. 1. This leads to a balance in verification ground truth between SAT and UNSAT answers, and these more challenging problems force the underlying algorithms to more precisely model network behavior, e.g., splitting of unstable neurons into branch and bound cases.

Selecting Verifiers Unlike other research that focuses on improving the performance of a single verifier with a single customized pruning techniques [10, 53, 59], our goal is to explore how the space of stabilization strategies impact a range of verification approaches. Towards this goal, we select the three best-performing verifiers from VNN-COMP 2022 [32] that were available: α,β-CROWN, MN-BAB, and NNENUM [4]. Improving the performance of these verifiers will extend the state-of-the-art in scalable DNN verification.

Network Training Stabilizers are incorporated into training, so we use a baseline(BASELINE) trained without any stabilizers using the Adam optimizer with a 10^{-3} learning rate and 0.99 decay for 20 epochs. All stabilizers are customizable with hyperparameters, as described in §3 and §4. We use the well-tuned parameter for RS LOSS introduced in [53], and perform a binary search of the parameter space for BIAS SHAPING and STABLE PRUNING. To elaborate, RS LOSS uses always-active scheduling with 10^{-4} weight parameter; BIAS SHAPING uses interval scheduling activated every 5/25/50 mini-batches and adjusts 2%/5%/5% of unstable neurons each time it is applied for **M2/M6/C2** architectures respectively; STABLE PRUNING undertakes an interval scheduling that is activated for every 5/50/50 mini-batches with a pruning ratio of 2%/5%/5% respectively. The resulting neural networks with the largest test accuracy of the last five epochs are selected for verification. To account for stochasticity in training, we train each network 5 times and report the mean data for each.

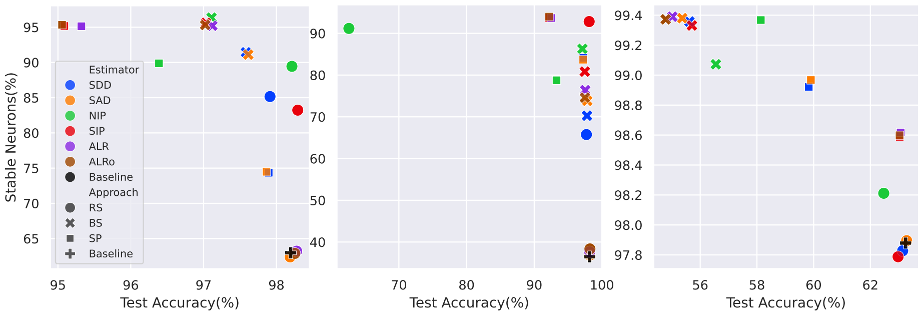
These choices for the space of experiments yield a total of 1,215 training tasks and 36,450 verification tasks. Each training task is run with one GTX 1080 Ti GPU with 11G VRAM. Each verification task is run with 8GB of memory on one core of the Intel Xeon Gold 6130 CPU @ 2.10GHz with a timeout of 300 seconds. The total CPU time spent on training and verification across our experiments is 1858 and 1052 hours, respectively.

5.2 RQ1: Stabilizing Neurons

Stabilizers aim to linearize a portion of the behavior encoded by ReLU activation across the set of computations activated for a property precondition. In this experiment, we directly measure this by recording the percentage of neurons that are stable during verification. We also record model test accuracy to understand the trade-offs of the stabilization methods and stability estimators. Existing verifiers do not record the number of stable neurons, so we modified an open-source DNN verifier, NEURALSAT [11], to record the number of stable neurons computed during verification.

Fig. 3 presents the average test accuracy and the average number of stable neurons computed across the five training seeds for the three architectures across the stabilizers in the benchmark as described in §5.1. The black ✚ sign indicates the BASELINE (Baseline), the ● sign represents RS LOSS (RS), ✖ means the BIAS SHAPING (BS) method, and ■ is STABLE PRUNING (SP). Six different colors denote the different stability estimators. Across all three architectures, most techniques can increase the number of stable neurons, but some of the techniques

[4] Verinet performed well in the competition, but it required a custom solver that is not freely available.



(a) **M2** (b) **M6** (c) **C3**

Fig. 3. Stable neurons(%) vs. test accuracy(%) per model

lead to a loss in test accuracy. For the **M2** architecture, RS Loss with NIP can significantly increase the number of stable neurons by more than 26 percentage points without compromising accuracy. For **M6**, RS Loss yields an even greater increase of 55 percentage points but in combination with the SIP estimator. For the Convolutional **C3** network, a very high percentage of neurons are already stable so only marginal improvement can be achieved. Here the Stable Pruning method performs best while preserving accuracy, but it only yields a percentage point increase. For all of the architectures, if one is willing to sacrifice a degree of accuracy then further increases in stability can be achieved. For example, for **M2** bias shaping can achieve an additional 7 percentage point increase in stable neurons at the cost of just over 1 percentage point in test accuracy.

Incorporating stabilization in training can increase training time. Fig. 4 shows the average training time for **M2** normalized to the Baseline. The trend for **M6** is similar to the other architectures. The clear outlier in terms of cost is the **ALRo** estimator when used with RS Loss, which incurs more than a 5-fold increase in training time. This overhead even prevents RS Loss from practically training with **ALR** and **ALRo** on the **C3** architecture. The overhead of most of the other estimators is negligible, including those that yielded significant increases in stable neurons.

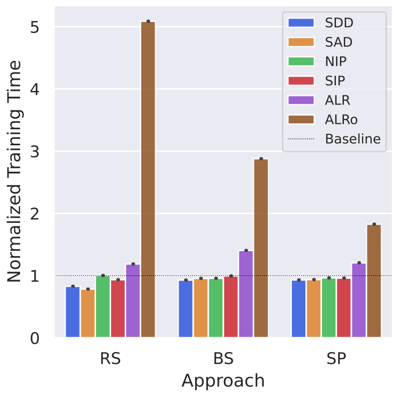
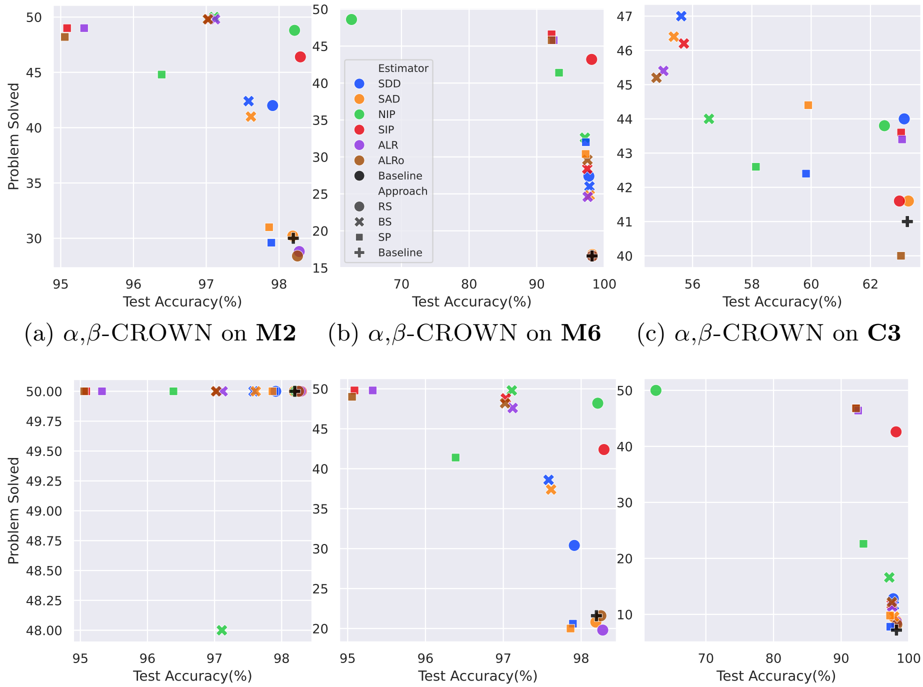


Fig. 4. Normalized training time

RQ1 Findings Across the study there are combinations of stabilization methods and stability estimators that are capable of increasing the number of stable neurons, in many cases substantially, without compromising test accuracy or training time.



(a) α,β-CROWN on **M2** (b) α,β-CROWN on **M6** (c) α,β-CROWN on **C3**

(d) MN-BAB on **M2** (e) NNENUM on **M2** (f) NNENUM on **M6**

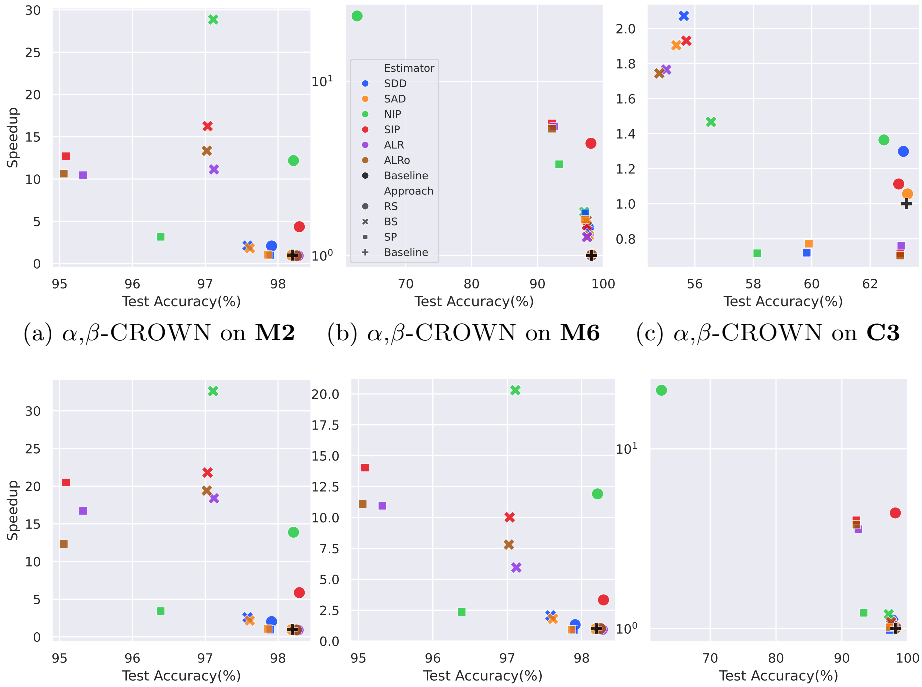
Fig. 5. Solved verification problems vs. test accuracy(%)

5.3 RQ2: Enhancing Verification

RQ1 demonstrates the ability of stabilizers to increase the number of stable neurons across a space of verification problems. This question explores whether those increases lead to improvements in verifier performance. To assess the generalization of the stabilizers to variations of DNN properties, we verify 50 local robustness properties per trained network, pairing 10 center points with each of the 5 epsilon radii. We run the three selected state-of-the-art verifiers on each problem.

We measure two metrics to assess verification performance: (1) the number of problems, i.e., the network, center-point, and radii combination, each verifier can solve, i.e., produce either an SAT or UNSAT result, and (2) the time taken to solve those problems. Note that our metrics exclude runs that produce errors, exceed a 300-second timeout, or an 8GB memory bound. These metrics are standard for assessing verifier performance and while sometimes they are aggregated, as in PAR2 [55], we keep them separate here to explore them independently.

Fig. 5 shows six plots of the number of verification problems solved versus test accuracy across the three architectures using three of the verifiers. The trends in



(a) α,β-CROWN on **M2** (b) α,β-CROWN on **M6** (c) α,β-CROWN on **C3**

(d) MN-BAB on **M2** (e) NNENUM on **M2** (f) NNENUM on **M6**

Fig. 6. Verification time speedup vs. test accuracy(%)

these plots are largely consistent with the findings of RQ1 - when more neurons are stable the verifiers are more effective in solving problems. RS LOSS, with different estimators, increases the number of problems solved by factors up to 5.92 for these verifier network combinations without sacrificing test accuracy. As in RQ1, further performance improvements are possible by sacrificing accuracy. For example, on **M2** α,β-CROWN can improve by a factor of 1.67 using BIAS SHAPING with a reduction of 1 percentage point in accuracy.

The trends shown here are consistent with the performance of α,β-CROWN and NNENUM across the study, but MN-BAB exhibited different performance. For **M2** and **M6**, the baseline technique was able to solve all 50 problems so there is no opportunity for improvement, while almost all the stabilizers can maintain the 50 problems solved. Note that the implementation of MN-BAB just doesn't support the **C3** architecture. While the number of problems does not change for MN-BAB with stabilization as we discuss below its runtime is reduced.

Fig. 6 plots the verification time speedup over BASELINE against test accuracy for 6 verifier network pairs. We observe a similar trend to what was observed for the number of neurons stabilized and the number of verification problems solved

– stabilization can speed up verification without compromising test accuracy. For MN-BAB on **M2** while the number of problems solved did not change, using RS Loss with **NIP** yielded a factor of 14 speedup. For **M6** we see a speedup of up to a factor of 5 with NNEnum and for **C3** more modest speedups for α,β-CROWN. The MN-BAB plot also shows, as observed above, that further speedups – greater than 30 fold – can be achieved if one compromises accuracy by about 1 percentage point.

RQ2 Findings Stabilizing neurons during training can substantially increase the number of problems solved and reduce the time required to solve them by state-of-the-art DNN verifiers without compromising test accuracy. Further improvement in verifier performance can be achieved with a small sacrifice in test accuracy.

5.4 Discussion

The data show a significant degree of variability in the effectiveness of particular stable training approaches with verifiers and verification problems. Broadly speaking RS Loss seems to perform well when one is unwilling to sacrifice test accuracy, but the best estimator varies depending on the verifier and problem – with **SDD**, **NIP**, and **SIP** yielding the best performance. For the large Convolutional network, Stable Pruning also performs well without compromising test accuracy. We believe this to be consistent with the broader results from the field of structured pruning [18, 43], where it has been found that large networks tend to be over-parameterized and can thus accommodate significant pruning without compromising accuracy. While the study shows that many of the methods can yield benefits, we believe that it also demonstrates that certain stabilization approaches, e.g., RS Loss with **ALRo**, are too costly for use in practice. Further study should focus on how to select the best stable training approach, and its hyperparameters, to yield the best improvement for a given verifier and class of verification problems. We believe it will be fruitful to develop such *training for verification* approaches in concert with algorithmic and engineering improvements to verification algorithms.

5.5 Threats to Validity

The chief threats to internal validity relate to whether the collection of test accuracy, stable neurons, verification problems solved, and verification time were accurate. We tested the accuracy of all stabilizer-trained networks, cross-checked problem solutions across verifiers, and thoroughly tested our instrumentation of NeuralSAT for recording neuron stability. Regarding external validity, while our study was scoped to manage experimental costs, it spanned: 3 verifiers, 3 network architectures, 50 property specifications, and 5 seeds. We used fixed sets of training and stabilizer parameters per neural network architecture, which potentially underestimated the benefit that might be observed by customizing parameters. While broadening the study further would be a valuable direction for future work, the scope of the study is sufficient to support the finding that stabilizers can enhance DNN verification across a breadth of contexts.

6 Conclusion

Verifying neural networks is a challenging task due to their high computational complexity. In this work, we propose two novel approaches Bias Shaping and Stable Pruning, to enhance the scalability of DNN verifiers by inducing more stable neurons during the training process. In addition, we designed six neuron stability estimators to drive stability-oriented training. Across a significant study, we found that focusing on stability yields a viable method to achieve training for verification that can significantly improve the ability to solve problems and speed up state-of-the-art verifiers.

Besides the promising results, we identified more opportunities when working on this project. In the future, we plan to (1) extend our methods to real-world large neural network architectures; (2) explore automatic ways to tune hyperparameters that lead to better performance; (3) further enhance the stabilizers' performance while minimizing accuracy trade-offs; (4) study the applicability of stabilizer combinations; and lastly (5) study the verification algorithms to understand how to customize stabilizers to benefit the most.

Acknowledgment

This material is based in part upon work supported by National Science Foundation awards 1900676, 2019239, 2129824, 2217071, and 2312487.

References

1. Bak, S.: Execution-guided overapproximation (ego) for improving scalability of neural network verification. In: International Workshop on Verification of Neural Networks (2020)
2. 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)
3. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: International Conference on Computer Aided Verification. pp. 66–96. Springer (2020)
4. Baluta, T., Chua, Z.L., Meel, K.S., Saxena, P.: Scalable quantitative verification for deep neural networks. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). pp. 312–323. IEEE (2021)
5. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. Advances in neural information processing systems **29** (2016)
6. Biere, A., Clarke, E., Raimi, R., Zhu, Y.: Verifying safety properties of a powerpc-microprocessor using symbolic model checking without bdds. In: Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11. pp. 60–71. Springer (1999)

7. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34(04), pp. 3291–3299 (2020)
8. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (vnn-comp). International Journal on Software Tools for Technology Transfer pp. 1–11 (2023)
9. Bunel, R., Mudigonda, P., Turkaslan, I., Torr, P., Lu, J., Kohli, P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research **21**(2020) (2020)
10. Chen, T., Zhang, H., Zhang, Z., Chang, S., Liu, S., Chen, P.Y., Wang, Z.: Linearity grafting: Relaxed neuron pruning helps certifiable robustness. In: International Conference on Machine Learning. pp. 3760–3772. PMLR (2022)
11. Duong, H., Li, L., Nguyen, T., Dwyer, M.: A dpll (t) framework for verifying deep neural networks. arXiv preprint arXiv:2307.10266 (2023)
12. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI. vol. 1(2), p. 3 (2018)
13. 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)
14. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: International Conference on Computer Aided Verification. pp. 43–65. Springer (2020)
15. Fazlyab, M., Morari, M., Pappas, G.J.: Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. IEEE Transactions on Automatic Control (2020)
16. Feng, C., Chen, Z., Hong, W., Yu, H., Dong, W., Wang, J.: Boosting the robustness verification of dnn by identifying the achilles's heel. arXiv preprint arXiv:1811.07108 (2018)
17. Ferrari, C., Müller, M.N., Jovanovic, N., Vechev, M.T.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022), `https://openreview.net/forum?id=l_amHf1oaK`
18. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), `https://openreview.net/forum?id=rJl-b3RcF7`
19. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18. IEEE (2018)
20. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 315–323. JMLR Workshop and Conference Proceedings (2011)
21. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International conference on computer aided verification. pp. 3–29. Springer (2017)
22. Johnson, T.T., Liu, C.: Vnn-comp2020 report, `https://www.overleaf.com/read/rbcfnbyhymmy`
23. 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)

24. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
25. Khedher, M.I., Ibn-Khedher, H., Hadji, M.: Dynamic and scalable deep neural network verification algorithm. In: ICAART (2). pp. 1122–1130 (2021)
26. Khedr, H., Ferlez, J., Shoukry, Y.: Effective formal verification of neural networks using the geometry of linear regions. arXiv preprint arXiv:2006.10864 (2020)
27. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: International Static Analysis Symposium. pp. 296–319. Springer (2019)
28. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for verifying deep neural networks. Foundations and Trends® in Optimization **4**(3-4), 244–404 (2021)
29. Livni, R., Shalev-Shwartz, S., Shamir, O.: On the computational efficiency of training neural networks. Advances in neural information processing systems **27** (2014)
30. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint arXiv:1706.07351 (2017)
31. Lu, J., Kumar, M.P.: Neural network branching for neural network verification. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), `https://openreview.net/forum?id=B1evfa4tPB`
32. Müller, M.N., Brix, C., Bak, S., Liu, C., Johnson, T.T.: The third international verification of neural networks competition (vnn-comp 2022): summary and results. arXiv preprint arXiv:2212.10376 (2022)
33. Paul, M., Chen, F., Larsen, B.W., Frankle, J., Ganguli, S., Dziugaite, G.K.: Unmasking the lottery ticket hypothesis: What's encoded in a winning ticket's mask? In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), `https://openreview.net/pdf?id=xSsW2Am-ukZ`
34. Raghunathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net (2018), `https://openreview.net/forum?id=Bys4ob-Rb`
35. Shriver, D., Elbaum, S., Dwyer, M.: Artifact: Reducing dnn properties to enable falsification with adversarial attacks. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). pp. 162–163 (2021). https://doi.org/10.1109/ICSE-Companion52605.2021.00068
36. Shriver, D., Elbaum, S., Dwyer, M.B.: Dnnv: A framework for deep neural network verification. In: International Conference on Computer Aided Verification. pp. 137–150. Springer (2021)
37. Shriver, D., Elbaum, S., Dwyer, M.B.: Reducing dnn properties to enable falsification with adversarial attacks. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). pp. 275–287. IEEE (2021)
38. Shriver, D., Xu, D., Elbaum, S., Dwyer, M.B.: Refactoring neural networks for verification. arXiv preprint arXiv:1908.08026 (2019)
39. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. Advances in Neural Information Processing Systems **32**, 15098–15109 (2019)
40. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. NeurIPS **1**(4), 6 (2018)

41. Singh, G., Gehr, T., Püschel, M., Vechev, M.: Boosting robustness certification of neural networks. In: International Conference on Learning Representations (2018)
42. 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)
43. Tan, C.M.J., Motani, M.: Dropnet: Reducing neural network complexity via iterative pruning. In: International Conference on Machine Learning. pp. 9356–9366. PMLR (2020)
44. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), `https://openreview.net/forum?id=HyGIdiRqtm`
45. 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)
46. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. Advances in neural information processing systems **31** (2018)
47. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1599–1614 (2018)
48. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. Advances in Neural Information Processing Systems **34**, 29909–29921 (2021)
49. Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning. pp. 5276–5285. PMLR (2018)
50. 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)
51. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE transactions on neural networks and learning systems **29**(11), 5777–5783 (2018)
52. Xiang, W., Tran, H.D., Rosenfeld, J.A., Johnson, T.T.: Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In: 2018 Annual American Control Conference (ACC). pp. 1574–1579. IEEE (2018)
53. Xiao, K.Y., Tjeng, V., Shafiullah, N.M.M., Madry, A.: Training for faster adversarial robustness verification via inducing relu stability. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), `https://openreview.net/forum?id=BJfIVjAcKm`
54. Xu, D., Mozumder, N.J., Duong, H., Dwyer, M.B.: The OCTOPUS Framework $\models$ Training for Verification: Increasing Neuron Stability to Scale DNN Verification (1 2024). https://doi.org/10.6084/m9.figshare.24916248.v3
55. Xu, D., Shriver, D., Dwyer, M.B., Elbaum, S.: Systematic generation of diverse benchmarks for dnn verification. In: International Conference on Computer Aided Verification. pp. 97–121. Springer (2020)
56. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic perturbation analysis for scalable certified robustness and beyond. Advances in Neural Information Processing Systems **33** (2020)

57. 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 (2021), https://openreview.net/forum?id=nVZtXBI6LNn
58. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 4944–4953 (2018), https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html
59. Zhangheng, L., Chen, T., Li, L., Li, B., Wang, Z.: Can pruning improve certified robustness of neural networks? Transactions on Machine Learning Research (2022)