Training Verification-Friendly Neural Networks via Neuron Behavior Consistency

Zongxin Liu[1,2], Zhe Zhao[3], Fu Song[1,4], Jun Sun[5], Pengfei Yang[6], Xiaowei Huang[7], Lijun Zhang[1,2]*

[1]Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]RealAI, Beijing, China
[4]Nanjing Institute of Software Technology, Nanjing, China
[5]Singapore Management University, Singapore
[6]College of Computer and Information Science, Software College, Southwest University, Chongqing, China
[7]The University of Liverpool, Liverpool, United Kingdom
liuzx@ios.ac.cn, zhe.zhao@realai.ai, songfu@ios.ac.cn, junsun@smu.edu.sg, ypfbest001@swu.edu.cn, xiaowei.huang@liverpool.ac.uk, zhanglj@ios.ac.cn

Abstract

Formal verification provides critical security assurances for neural networks, yet its practical application suffers from the long verification time. This work introduces a novel method for training verification-friendly neural networks, which are robust, easy to verify, and relatively accurate. Our method integrates neuron behavior consistency into the training process, making neuron activation states remain consistent across different inputs within a local neighborhood. This reduces the number of unstable neurons and tightens the bounds of neurons thereby enhancing the network's verifiability. We evaluated our method using the MNIST, Fashion-MNIST, and CIFAR-10 datasets with various network architectures. The experimental results demonstrate that networks trained using our method are verification-friendly across different radii and architectures, whereas other tools fail to maintain verifiability as the radius increases. Additionally, we show that our method can be combined with existing approaches to further improve the verifiability of networks.

Introduction

Neural networks are increasingly being applied in safety-critical domains such as autonomous driving (Urmson and Whittaker 2008) and flight control (Julian, Kochenderfer, and Owen 2019). However, they often struggle with a lack of robustness, as even minor perturbations to their inputs can lead to incorrect predictions (Bu et al. 2022; Chen et al. 2021; Song et al. 2021; Chen et al. 2023, 2022b; Zhao et al. 2021; Chen et al. 2022a). This is unacceptable in safety-critical applications, where consistent performance is imperative. Thus, it is desirable to develop methods to systematically advance the robustness verification of neural networks.

Existing methods for analyzing robustness can be broadly classified into two categories: empirical analysis through adversarial attacks and mathematical proof via formal verification. While adversarial attacks generate misleading examples, they merely demonstrate the presence of adversarial samples without affirming their absence. In contrast, formal verification ensures the correctness of neural networks using logical and mathematical methods, allowing us to verify their robustness formally. This rigorous verification is essential for safety-critical systems.

Advanced formal verification tools (Wang et al. 2021; Zhang et al. 2022a; Bak 2021; Katz et al. 2017), typically employ branch-and-bound algorithms for neural network verification. At the start of the verification process, abstract interpretation (Gehr et al. 2018; Mirman, Gehr, and Vechev 2018; Zhang et al. 2021, 2023; Guo et al. 2021) is usually used to abstract neurons. If the properties of the network remain undetermined after using symbolic propagation (Singh et al. 2019) to calculate neuron boundaries and applying MILP (Tjeng, Xiao, and Tedrake 2019; Tran et al. 2020; Zhang et al. 2022b; Zhang, Song, and Sun 2023; Zhang et al. 2024) or SMT methods (Ehlers 2017; Huang et al. 2017; Katz et al. 2017; Zhao et al. 2022; Liu et al. 2024a) for constraint solving, further branching is required. The branching process involves enumerating the activation states of unstable neurons, whose activation status cannot be determined through bound calculations. This introduces additional constraints, thereby refining the abstraction. Typically, neural networks contain numerous unstable neurons, and exploring the combinations of these activation states requires exponential time, which limits the widespread application of formal verification techniques in practice.

In addition to developing ever-more sophisticated methods for post-training verification, researchers have investigated the idea of training neural networks that are easier to verify, known as verification-friendly neural networks. Ideally, a training method for verification-friendly neural networks must satisfy the following requirements. First, (accuracy) the resultant neural network must have an accuracy comparable to that of neural networks trained conventionally. Second, (robustness) the resultant neural network must have improved robustness, which could be measured using existing adversarial attacks. Third, (verifiability) it must be easier to verify using existing or dedicated neural network verification techniques, which can be measured using the ef-

*Corresponding author.

fectiveness of selected neural network verification methods.

There are mainly two existing approaches to promoting the verification-friendliness of neural networks. One approach involves post-processing the network using methods that modify the weights of the networks (Xiao et al. 2019; Baninajjar, Rezine, and Aminifar 2024). The other involves altering the neural network design and training process with considerations for verification (Xiao et al. 2019; Narodytska et al. 2019; Xu et al. 2024). However, these methods still have limitations. The effectiveness of post-training methods is limited by the network itself. Additionally, certain post-training methods such as using MILP to make the network sparser (Baninajjar, Rezine, and Aminifar 2024), are computationally expensive and may not scale well for large networks. The ReLU Stable methods, which introduce ReLU Stable (RS) loss (Xiao et al. 2019) to reduce the number of unstable neurons, depend on the bounds of neurons. When the perturbation radius changes, it often causes these bounds to shift, affecting the verification efficiency of the network. Certified training, which is primarily based on the heavy Interval Bound Propagation (IBP) method (Mirman, Gehr, and Vechev 2018; Gowal et al. 2018; Zhang et al. 2019a; Xu et al. 2020) suffers from long training times and issues like gradient explosion or vanish problem. Adversarial training (Madry et al. 2018; Zhang et al. 2019b; Ganin et al. 2016; Zhu et al. 2017) typically increases network robustness but does not contribute to improving its verifiability.

In this work, we introduce a straightforward yet effective training method that enhances the verifiability of neural networks by enforcing the consistency of neuron behavior, which we refer to as neuron behavior consistency (NBC), throughout the training process as a regularization term. A neuron is called behavior consistent if its activation state remains the same within a given input neighborhood. By maximizing the consistency of neurons, the unstable neurons are decreased, reducing the search space of the verification process. NBC can also help tighten the bounds of neurons, as fewer unstable neurons introduce less error during bound calculation algorithms. Our approach can be scaled to larger networks compared to MILP-based methods. Moreover, the core of our method lies in the consistency of neuron behavior without relying on heavy IBP methods, which reduces training epochs and ensures that the trained network maintains verifiability across different perturbation radii.

We evaluate our method using Fashion-MNIST, MNIST, and CIFAR-10 datasets across different architectures at various perturbation radii. Our method outperforms others in stable neuron ratio and achieves up to a 450% speedup in verification time. Importantly, our method accelerates the verification while preserving the accuracy of the models, which is not commonly achieved by existing methods. In summary, our contributions are as follows:

- We introduce a method of training verification-friendly networks by integrating neuron behavior consistency.
- We evaluate our method on three well-known datasets. Experimental results show that networks trained using our method can maintain verification-friendly properties across different radii and different model architectures.
- We demonstrate that our method can be combined with existing methods to further improve the verifiability of networks, especially in the case of large networks.
- We show that our method accelerates the verification process while preserving model accuracy and robustness.

Preliminary

In this section, we introduce the background of neural network verification problems and the general branch and bound verification framework.

Neural Networks Verification Problems

Given a neural network $f : \mathbb{R}^{m_{\text{in}}} \to \mathbb{R}^{m_{\text{out}}}$, with m_{in} input neurons and m_{out} output neurons, the goal of the neural network verification problem is to determine whether the output of the network satisfies a set of output constraints $\mathcal{P}$ for all inputs that meet the input constraints $\mathcal{C}$, formally defined as:

Definition 1 (Neural Network Verification Problem)**.** *The neural network verification problem $\langle f, \mathcal{C}, \mathcal{P} \rangle$ is to determine whether:*

$$\forall \boldsymbol{x} \in \mathcal{C} \Rightarrow f(\boldsymbol{x}) \in \mathcal{P}, \tag{1}$$

where $\mathcal{C} \subseteq \mathbb{R}^{m_{\text{in}}}$ represents the input constraints and $\mathcal{P} \subseteq \mathbb{R}^{m_{\text{out}}}$ represents the output constraints.

We focus on the local robustness verification problem $\langle f, \mathcal{C}_\varepsilon(\boldsymbol{x}), \mathcal{P}_c \rangle$, which checks if the classification result c is robust within a l_∞-radius ε around input $\boldsymbol{x}$, where the input constraints $\mathcal{C}_\varepsilon(\boldsymbol{x})$ are defined as $\{\boldsymbol{x}' \in \mathbb{R}^{m_{\text{in}}} \mid \|\boldsymbol{x}' - \boldsymbol{x}\|_\infty \leq \varepsilon\}$ and the output constraints $\mathcal{P}_c$ are defined as $\{\boldsymbol{y} \in \mathbb{R}^{m_{\text{out}}} \mid \bigwedge_{i \neq c} \boldsymbol{y}_i - \boldsymbol{y}_c \leq 0\}$.

General Verification Framework

State-of-the-art methods for solving neural network verification problems (Wang et al. 2021; Zhang et al. 2022a; Katz et al. 2019; Bak 2021) are typically based on branch-and-bound algorithms, consisting of three critical components: constraint solving, bound calculation, and branch selection.

As shown in Figure 1, the verification process begins with the bound calculation, where the upper and lower bounds of neuron outputs are estimated under specified input constraints. If these bounds are sufficiently precise, the properties of the network can be directly verified. Due to non-linear activation functions such as ReLU, computing these bounds can be complex, which is a problem often addressed through neuron-wise abstraction (Singh et al. 2019; Bak 2021). This abstraction uses linear bounds to approximate neuron outputs, thereby simplifying the bound calculations.

When the bound calculation does not suffice to verify the network's properties, constraint solving is employed. This process involves determining if the constraints of the abstracted network can be satisfied, using Linear Programming (LP) or Mixed Integer Linear Programming (MILP) methods. Constraints typically include the input constraints $\mathcal{C}$, the negated output constraints $\neg\mathcal{P}$, and the constraints of the abstracted network itself. If these constraints are UNSAT (unsatisfiable), the property holds, proving that the network

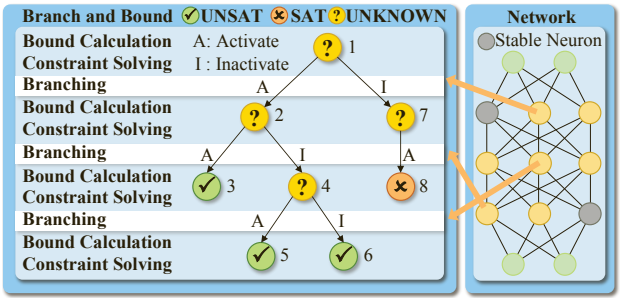

Figure 1: The Branch and Bound (BaB) verification process

can not be successfully attacked within $\mathcal{C}$; otherwise, the returned counterexample must be examined. If the counterexample is not a false positive caused by over-approximation, it indicates a violation of the network's properties. Otherwise, it suggests that the network is over-abstracted, necessitating branch selection to refine the abstraction.

Branch selection selects an unstable neuron (whose activation state cannot be determined through bound calculations) and splits it into active and inactive branches. Identifying the state of a neuron entails establishing new constraints that refine the abstraction. Branch selection strategies significantly affect verification efficiency. While branching may require the exploration of all unstable neurons in worst-case scenarios, minimizing these neurons can exponentially reduce the theoretical upper bound on branching.

Training Verification-Friendly Networks via Neuron Behavior Consistency

In this section, we propose our method, neuron behavior consistency, to train verification-friendly neural networks.

Given a network f with parameters θ and an underlying distribution $\mathcal{D}$, the traditional training process aims to optimize the parameters θ to minimize the expected loss:

$$\min_{\theta} \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}} (loss(\boldsymbol{x},\boldsymbol{y})), \tag{2}$$

where $(\boldsymbol{x}, y)$ denotes the input and target label sampled from the distribution $\mathcal{D}$, and $\boldsymbol{y}$ is the one-hot vector representation of the target label y, that is, a vector with a single 1 at the index of the target label and 0 elsewhere. The most common loss function is the cross-entropy function for classification tasks, defined as $\mathrm{CE}(f(\boldsymbol{x}), \boldsymbol{y}) = -\sum_i \boldsymbol{y}_i \log f(\boldsymbol{x})_i$.

The ordinary training objective does not impose constraints on neurons, potentially resulting in a large number of unstable neurons. We thus propose an alternative training objective that aims to maximize the consistency of neurons. A neuron is called consistent if its activation state remains the same within a given input neighborhood. Formally, given an input $\boldsymbol{x}$ and a neighboring input $\boldsymbol{x}'$, the consistency of the j-th neuron in the i-th layer $n_j^{(i)}$ is defined as:

$$\mathrm{NBC}(n_j^{(i)}, \boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 1, & \text{if } \mathrm{sign}(f^{(i)}(\boldsymbol{x})_j) = \mathrm{sign}(f^{(i)}(\boldsymbol{x}')_j), \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $f^{(i)}(\boldsymbol{x})_j$ denotes the pre-activation value of the j-th neuron of the i-th layer when fed with input $\boldsymbol{x}$. Intuitively,

Algorithm 1: Calculation of NBC

Input: Neural network f, input $\boldsymbol{x}$, neighbor input $\boldsymbol{x}'$
Output: Neural behavior consistency between $\boldsymbol{x}$ and $\boldsymbol{x}'$

$s \leftarrow 0$
for the i-th layer l_i in hidden layers **do**
  $\boldsymbol{v} \leftarrow [f^{(i)}(\boldsymbol{x})_1, f^{(i)}(\boldsymbol{x})_2, \ldots, f^{(i)}(\boldsymbol{x})_{m[i]}]^{\mathrm{T}}$
  $\boldsymbol{v}' \leftarrow [f^{(i)}(\boldsymbol{x}')_1, f^{(i)}(\boldsymbol{x}')_2, \ldots, f^{(i)}(\boldsymbol{x}')_{m[i]}]^{\mathrm{T}}$
  $nbc \leftarrow \frac{\boldsymbol{v}\cdot\boldsymbol{v}'}{|\boldsymbol{v}|\cdot|\boldsymbol{v}'|}$
  $s \leftarrow s + \frac{nbc}{\gamma[i]}$
end for
$s \leftarrow s - \mathrm{KL}(f(\boldsymbol{x})||f(\boldsymbol{x}'))$
return s

for any input within a given neighborhood, if the activation states of individual neurons are highly consistent, the calculated boundaries (upper and lower bounds) are more likely to be tight. This coherence may reduce the occurrence of unstable neurons in the neural network.

To maximize (minimize the negative) this consistency, we incorporate a regularization term into the optimization objective, which can be represented as:

$$\min_{\theta} \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D},\boldsymbol{x}'\in\mathcal{C}_\varepsilon(\boldsymbol{x})} [\mathrm{CE}(f(\boldsymbol{x}),\boldsymbol{y}) - \beta \sum_{n_i\in\mathcal{N}} \mathrm{NBC}(n_i,\boldsymbol{x},\boldsymbol{x}')], \tag{4}$$

where β is a hyperparameter that controls the importance of the regularization term, and $\mathcal{N}$ denotes the set of neurons in the network.

Adapting concepts from adversarial training, the loss function can be reformulated to maximize the minimal (minimize the negative minimal) consistency of neural behavior across different inputs within the neighborhood domain:

$$l^{\mathrm{NBC}}(\boldsymbol{x}, y) = \mathrm{CE}(f(\boldsymbol{x}),\boldsymbol{y}) - \beta \min_{\boldsymbol{x}'\in\mathcal{C}_\varepsilon(\boldsymbol{x})} \sum_{n_i\in\mathcal{N}} \mathrm{NBC}(n_i,\boldsymbol{x},\boldsymbol{x}'). \tag{5}$$

The consistency metric presented in Equation 3 is a discrete measure and cannot be directly integrated into the loss function. Therefore, we employ a continuous metric to approximate the consistency of neural behavior across different inputs within the neighborhood domain, as shown in Algorithm 1. This algorithm calculates the NBC for the neural network f when fed with $\boldsymbol{x}$ and $\boldsymbol{x}'$.

During the calculation of NBC, the NBC value s is initially set to zero. Then iterating over the hidden layers of f, the algorithm assesses the consistency of each neuron between the original and adversarial images, incrementally updating the NBC. The final NBC is computed as the sum of the scaled neuron consistencies across all layers.

Due to the characteristics of gradient backpropagation, the layer close to the output layer has a greater impact on the network's behavior. Therefore, we use the KL divergence as a consistency metric for the output layer, serving as a regularization term to ensure that the network's output remains consistent across different inputs. This can be calculated as:

$$\mathrm{KL}(f(\boldsymbol{x})||f(\boldsymbol{x}')) = \sum_i f(\boldsymbol{x})_i \log \frac{f(\boldsymbol{x})_i}{f(\boldsymbol{x}')_i}. \tag{6}$$

For the hidden layers, to prevent gradient explosion or vanishing, we use cosine similarity as a continuous metric to

Algorithm 2: Calculation of l^{NBC}

Input: Neural network f, input $\boldsymbol{x}$, label $\boldsymbol{y}$, perturbation ε, number of perturbation steps k, step size α, NBC regularization hyperparameter β
Output: Final NBC loss l^{NBC}

1: Generate a random starting point $\boldsymbol{x}' \in \mathcal{C}_\varepsilon(\boldsymbol{x})$
2: **for** i from 1 to k **do**
3: $\quad \Delta\boldsymbol{x} \leftarrow \frac{\partial \text{NBC}(f,\boldsymbol{x},\boldsymbol{x}')}{\partial \boldsymbol{x}'}$
4: $\quad \boldsymbol{x}' \leftarrow \boldsymbol{x}' - \alpha\Delta\boldsymbol{x}$
5: $\quad \boldsymbol{x}' \leftarrow \text{clip}(\boldsymbol{x}', \boldsymbol{x}, \varepsilon)$
6: **end for**
7: $l^{\text{NBC}} \leftarrow \text{CE}(f(\boldsymbol{x}), \boldsymbol{y}) - \beta \cdot \text{NBC}(f, \boldsymbol{x}, \boldsymbol{x}')$
8: **return** l^{NBC}

approximate the consistency of behavior. Cosine similarity outputs a value in the range of [0,1], making it more suitable for training neural networks with high-dimensional intermediate layers.

To balance the impact of layers with different numbers of neurons, we scale the NBC value by the factor $\gamma[i]$ at line 6 of Algorithm 1. Our intuition is to prioritize layers with smaller dimensions, as their more consistent behavior can help propagate constraints through the network. Additionally, layers near the input and output often have fewer neurons. Applying constraints to these layers directly influences the forward and backward propagation processes, which can accelerate the convergence of the target loss. In contrast, over-constraining the middle layers, which contain more neurons and extract more complex features, could limit the model's expressive power. More generally, applying stricter penalties to layers may decrease accuracy but increase the proportion of stable neurons. For these reasons, we mainly impose constraints on layers with fewer neurons. In our experiments, we apply the factor $\gamma[i] = 2^{r[i]}$ to balance the impact of layers with different numbers of neurons, where $r[i]$ means the number of neurons in the i-th hidden layer $m[i]$ is the $r[i]$-th smallest number in all the numbers of neurons in hidden layers.

Algorithm 2 outlines the NBC loss calculation process. This algorithm uses the idea of adversarial training to find an adversarial input $\boldsymbol{x}'$ that minimizes the NBC loss between the input $\boldsymbol{x}$ and $\boldsymbol{x}'$. The algorithm starts by selecting a random adversarial input $\boldsymbol{x}'$ within the ε-neighborhood of the original input $\boldsymbol{x}$. The adversarial input is then iteratively perturbed over k steps, with each perturbation adjusting the adversarial input based on the gradients of the NBC concerning $\boldsymbol{x}'$, scaled by the step size α. The adversarial input is clipped after each perturbation step to ensure it remains within the perturbation range allowed by the original input. The final NBC loss is computed as the summation of the cross-entropy loss and the negative sum of scaled NBC (calculated according to Algorithm 1) between the original input and the adversarial input. A detailed discussion of the hyperparameters γ in Algorithm 1 and β in Algorithm 2 is provided in supplementary material (Liu et al. 2024b).

We train the network to maximize the consistency of neuron behavior across varied inputs within the neighborhood, thereby reducing the number of unstable neurons and tightening neuron bounds. As a result, the network trained with the NBC loss is friendly to formal verification methods, as it exhibits fewer unstable neurons and more precise bounds.

Evaluation

In this section, we evaluate our method to address the following research questions:
RQ1: Can networks trained with our method maintain verification-friendly properties across various network architectures and perturbation radii?
RQ2: Can our method be effectively integrated with existing training methods?
RQ3: How does the performance of our method compare to existing methods when their accuracies are close?

Experimental Setup

Experiments are conducted on a server with 128 Intel Xeon Platinum 8336C CPUs, 128GB memory, and four NVIDIA GeForce RTX 4090 GPUs, running Debian GNU/Linux 10 (Buster). We use Python 3.11.7 and PyTorch 2.1.2 for implementation. Other settings are as follows.
Dataset. Networks are trained on three widely used datasets: MNIST (LeCun et al. 1998), Fashion-MNIST (Xiao, Rasul, and Vollgraf 2017), and CIFAR-10 (Krizhevsky 2009).
Network Architecture. We select neural networks of varying sizes from VNN-COMP (Bak, Liu, and Johnson 2021; Müller et al. 2023) to assess the effectiveness of the methods used. For the MNIST and Fashion-MNIST datasets, we chose the M1 (cnn_4_layer), M2 (relu_stable), and M3 (conv_big) models, with approximately 0.16M, 0.17M, and 1.9M parameters, respectively. For the CIFAR-10 dataset, we select the C1 (marabou_medium), C2 (marabou_large), and C3 (conv_big) models, containing about 0.17M, 0.34M, and 2.4M parameters, respectively. Network architectures are detailed in supplementary material (Liu et al. 2024b).
Baselines. We choose Relu Stable (Xiao et al. 2019) as a baseline, as it shares the most similarities with our approach. TRADES (Zhang et al. 2019b) and Madry (Madry et al. 2018), two commonly used adversarial training methods, are selected to show that directly using our method can at least achieve robustness comparable to classical robust training methods and combining our method with existing methods can improve verification performance.
Training. The batch size is set to 128, and using the Adam optimizer. For RQ1, networks are trained under default settings for 400 epochs. For RQ2, each network is trained for 200 epochs using the original method, followed by an additional 200 epochs combining the original method with our approach, or vice versa. For RQ3, we train a base model using the CE loss, then fine-tune it separately using RS, Madry, TRADES, and our method, ensuring that each method maintains accuracy within a specified range.
Verification. We use α, β-CROWN, a state-of-the-art verification tool that performs the best in VNN-COMP competitions, to verify the properties of the trained networks. For each dataset, we select k images from each of the 10 categories in the test set. For each image $\boldsymbol{x}$ and its ground

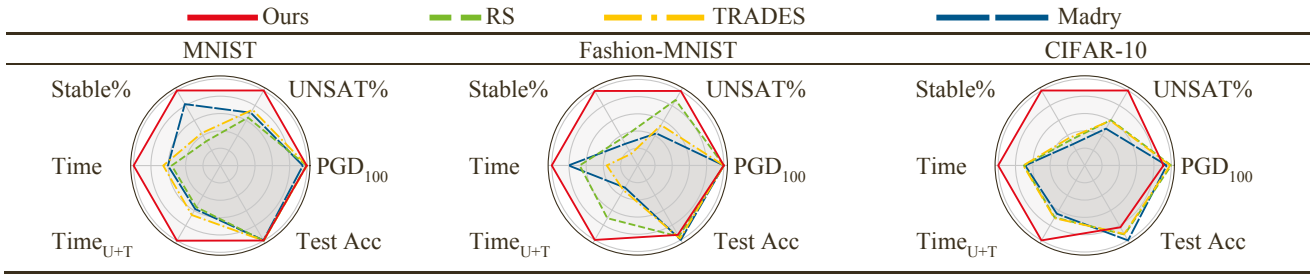

Figure 2: Overview of last epoch results. For each metric at each perturbation radius, the best-performing value is defined as 100, with other values scaled proportionally. We average all the scaled values of each model at each perturbation radius in each dataset to obtain the final score. For each metric, the larger value indicates better performance.

Model		M1				M2				M3			
Method		T	M	R	O	T	M	R	O	T	M	R	O
Test Acc.		**99.0**	98.2	98.9	98.8	99.1	98.3	**99.2**	98.9	99.1	99.3	**99.3**	99.0
ε=0.1	UNSAT%	96.6	95.0	96.4	**97.2**	96.8	93.0	96.9	**97.0**	94.5	87.8	86.0	**96.6**
	Stable%	72.0	84.7	68.0	**84.7**	69.4	81.5	65.9	**86.6**	78.4	69.3	48.4	**92.1**
	Time (s)	3.6	4.7	4.0	**3.0**	3.7	5.2	3.6	**3.2**	9.2	19.0	20.6	**4.0**
	$\text{Time}_{\text{U+T}}$ (s)	3.5	4.5	3.9	**2.9**	3.6	5.0	3.5	**3.1**	8.9	19.1	20.7	**3.9**
	PGD_{100}	97.6	95.9	**97.7**	**97.7**	97.8	94.3	**97.9**	97.9	98.0	97.6	**98.2**	97.9
ε=0.2	UNSAT%	89.5	86.7	79.0	**92.2**	85.6	71.3	71.8	**91.8**	2.3	0.2	1.5	**46.3**
	Stable%	34.8	**65.7**	25.2	64.5	27.7	66.1	20.5	**68.0**	17.2	10.0	4.3	**68.7**
	Time (s)	13.7	12.8	28.0	**7.3**	20.8	30.2	39.1	**8.7**	111.8	113.0	113.1	**68.4**
	$\text{Time}_{\text{U+T}}$ (s)	14.1	13.1	29.7	**7.3**	21.6	33.8	41.2	**8.8**	117.5	119.8	118.3	**71.9**
	PGD_{100}	**95.7**	91.8	95.0	95.6	96.0	88.8	95.8	**96.4**	**96.1**	95.5	96.0	96.0
ε=0.3	UNSAT%	29.4	35.5	7.0	**60.1**	13.3	8.2	3.9	**47.4**	**0.0**	**0.0**	**0.0**	**0.0**
	Stable%	6.2	40.7	2.4	**47.8**	2.9	37.2	1.4	**53.9**	0.1	0.0	0.0	**52.5**
	Time (s)	81.1	64.3	94.3	**47.8**	95.4	84.5	99.8	**63.5**	108.4	**106.6**	108.3	107.7
	$\text{Time}_{\text{U+T}}$ (s)	91.5	80.0	113.8	**55.0**	108.5	113.0	116.6	**72.9**	**120.0**	**120.0**	**120.0**	**120.0**
	PGD_{100}	**93.1**	83.3	89.3	91.3	**92.8**	80.0	91.8	91.4	**92.2**	91.3	91.6	91.9

Table 1: Networks trained with $\varepsilon = 0.3$ on MNIST datasets. The best results are highlighted in bold. Verified under ε=0.1, 0.2, 0.3. T:TRADES, M:Madry, R:ReLU Stable, O:Ours.

truth label y, we verify the property that the network's output label remains y for input $\boldsymbol{x}$ under each perturbation ε. We set $k = 100$ and a timeout of 120 seconds for MNIST and Fashion-MNIST, and $k = 20$ with a timeout of 180 seconds for CIFAR-10.

Metrics. The metrics used in our evaluation are as follows:

- UNSAT%: The percentage of properties verified to hold (UNSAT), indicating the network's overall verification effectiveness and robustness.
- Stable%: The average percentage of stable neurons, calculated as: $\frac{\sum_1^N s_i}{N}$, where s_i is stable neuron ratio of the i-th property, and N is the total number of properties.
- Time: The average time required to verify a property.
- $\text{Time}_{\text{U+T}}$: The average time is taken to verify properties that result in UNSAT or Timeout. This metric is more indicative of the efficiency of the verification process, as it excludes the time taken to verify properties that are SAT, which can be verified quickly by attacking the network.
- PGD_{100}: The network's accuracy under a PGD attack for 100 steps with a perturbation ε.

Results of the Last Epoch

Figure 2 shows the overall evaluation results of networks trained on MNIST, Fashion-MNIST, and CIFAR-10. Since some metric values are too large or small, we set the best-performing value to 100, with other values scaled proportionally for better visualization. For the metrics Time and $\text{Time}_{\text{U+T}}$, we use the reciprocal of the values to make the visualization more intuitive. Therefore, the larger the metric value in the table, the better the performance.

We observe that our method outperforms others in terms of verification time, stable neuron ratio, and verified ratio (UNSAT%). While our method slightly lags in accuracy and PGD accuracy, it generally maintains comparable accuracy to the other methods. This result represents a comprehensive evaluation across various perturbation radii and models, indicating that networks of various architectures trained using our method are more verification-friendly at different perturbation radii than those trained using other methods.

Table 1 shows the detailed results for networks trained on MNIST. Detailed results for Fashion-MNIST and CIFAR-10 are provided in the supplementary material. As shown

Model		M1			M2			M3		
Method		T*	M*	R*	T*	M*	R*	T*	M*	R*
Test Acc.		-0.1	-3.9	-3.5	-0.1	-5.7	-2.8	-0.1	-0.4	-0.5
ε=0.1	UNSAT%	**+0.6**	-7.6	-4.3	-0.2	-11.2	-3.8	**+2.0**	**+9.0**	**+10.6**
	Stable%	**+12.7**	**+12.8**	**+24.4**	**+16.4**	**+15.8**	**+25.6**	**+13.7**	**+25.3**	**+44.4**
	Time (s)	**-0.7**	**-1.1**	**-0.7**	+1.1	+0.9	**-0.3**	**-3.4**	**-14.8**	**-16.3**
	$\text{Time}_{\text{U+T}}$ (s)	**-0.6**	**-1.4**	**-1.0**	+1.1	+0.8	**-0.5**	**-3.4**	**-15.1**	**-16.7**
	PGD_{100}	**+0.1**	-6.1	-4.3	-4.0	-7.9	-3.5	-0.2	**+0.3**	-0.4
ε=0.2	UNSAT%	**+2.7**	-10.0	**+9.4**	**+4.9**	-11.8	**+18.0**	**+43.0**	**+83.2**	**+87.0**
	Stable%	**+29.7**	**+28.1**	**+57.9**	**+37.9**	**+26.2**	**+60.1**	**+51.0**	**+70.6**	**+74.6**
	Time (s)	**-6.4**	**-6.8**	**-23.9**	**-7.2**	**-12.3**	**-34.7**	**-41.9**	**-88.0**	**-97.3**
	$\text{Time}_{\text{U+T}}$ (s)	**-6.8**	**-7.8**	**-26.0**	**-7.7**	**-11.5**	**-37.2**	**-44.4**	**-94.4**	**-102.5**
	PGD_{100}	-0.2	-9.9	-4.2	-2.2	-12.8	-4.1	-0.2	**+0.4**	-0.3
ε=0.3	UNSAT%	**+30.7**	**+16.8**	**+72.4**	**+30.4**	-0.6	**+75.1**	0.0	**+4.4**	**+38.9**
	Stable%	**+41.6**	**+46.2**	**+69.1**	**+47.7**	**+39.5**	**+64.9**	**+51.7**	**+57.4**	**+53.8**
	Time (s)	**-33.3**	**-44.1**	**-86.2**	**-27.2**	**-32.6**	**-90.6**	**-0.5**	**-3.9**	**-40.8**
	$\text{Time}_{\text{U+T}}$ (s)	**-36.5**	**-52.0**	**-105.9**	**-31.2**	**-6.4**	**-107.4**	0.0	**-4.4**	**-44.7**
	PGD_{100}	-1.5	-15.2	-3.0	**+1.1**	-21.6	-4.3	**+0.2**	**+1.0**	-0.4

Table 2: Networks trained with each method combined with our method on the MNIST dataset with $\varepsilon = 0.3$. Improved results are highlighted in bold. T*: TRADES+Ours, M*: Madry+Ours, R*: ReLU Stable+Ours.

in Table 1, traditional adversarial training methods enhance network robustness, yet verifiability declines as the perturbation radius increases. This suggests the need for new training techniques that support effective verification.

Regarding UNSAT%, our method consistently outperforms others, especially at higher perturbation radii. For instance, when $\varepsilon = 0.2$, our verified ratio reached 46.3%, more than 20 times that of the next best-performing method and over 231 times that of the least effective method on the M3 model. Unlike other methods, where UNSAT% significantly drops as the radius increases, our method maintains a high verified ratio across all tested radii.

As for stable neurons, our method outperforms others in most models and perturbation settings. In the M3 model at $\varepsilon = 0.3$, while competing methods showed almost no stable neurons, our approach maintained a stable neuron ratio above 50%. Higher proportion of stable neurons increases the likelihood of successful verification with extended verification time, as the search space is significantly reduced.

Our method generally requires less verification time. An exception is the M3 model at a 0.3 perturbation radius, where verification failed to confirm all properties. In contrast, the model trained by Madry quickly verified many properties as violated, reducing verification time. Additionally, under 100-step PGD attacks, our method achieved accuracy comparable to adversarial training methods. Same results also hold for other two datasets, see (Liu et al. 2024b).

Answer to RQ1: Our method maintains high stable neuron ratios, UNSAT%, and robustness across various perturbation radii. Overall, networks trained with our method preserve their verification-friendly properties across different network architectures and radii.

Combination with Existing Methods

Table 2 shows the results of networks trained with our method combined with other methods on MNIST dataset. The results consistently demonstrate that combining our method with existing approaches improves the ratio of stable neurons. In most cases, this combination also increases the ratio of verified properties (UNSAT%) and significantly reduces the time required for verification.

It is worth noting that when our method is combined with a baseline method, the performance is particularly outstanding in larger network structures. For example, in the M3 model. Our method combined with the Madry and RS methods respectively increased the verified ratio by 83.2% and 87.0% at $\varepsilon = 0.2$, while also reducing verification time by 94.4 seconds and 102.5 seconds. These results indicate that our method makes it possible to verify larger network structures. The same results are also reflected in the Fashion-MNIST and CIFAR-10 datasets provided in the supplementary material.

Combining our method with existing approaches leads to a slight trade-off in overall accuracy and adversarial accuracy. Nevertheless, this trade-off is accompanied by a significant improvement in the verification process and the consistency of neuron behavior. The improvements are especially pronounced for larger models, suggesting that our method facilitates the verification of more complex and larger network architectures, which can be challenging for current verification tools. Additionally, similar results are also observed in the Fashion-MNIST and CIFAR-10 datasets, as shown in the supplementary material.

Answer to RQ2: Our method, when combined with other methods, improves the verification-friendly properties of the network. However, a trade-off between accuracy and verifiability is required.

Comparison under Close Accuracy

The accuracy of networks trained with default parameters varied significantly on the CIFAR-10 dataset. To compare the performance of networks with similar accuracy, we fine-

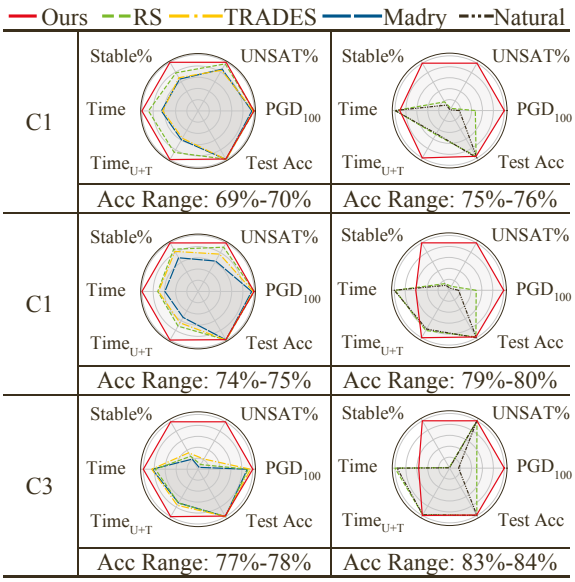

Figure 3: Networks trained with $\varepsilon = 2/255$ on CIFAR-10 dataset and evaluated on $\varepsilon = 2/255$. Methods unable to achieve a given accuracy range are omitted. 'Natural' means the network is trained with only cross-entropy loss.

tuned the parameters to ensure that the accuracy of each method remained within a certain range.

Figure 3 shows the results of networks trained on CIFAR-10. Larger perturbation radii are very challenging for current verification methods, so we chose a smaller perturbation radius to train and evaluate the networks.

For UNSAT%, our method performs best across different network models and different accuracy ranges and generally requires less verification time for timeouts and UNSAT results than other methods, indicating that our networks are easier to verify. When the models reach higher accuracy, our method has a slightly longer average time, as other methods are more easily to find counterexamples using attack methods, while our method is less vulnerable to attacks.

Our accuracy under PGD is generally higher than that of other methods, with a few exceptions where it is slightly lower than the best-performing method. Our method consistently maintains a higher ratio of stable neurons than other methods. Notably, in the C3 model, our method maintains a high stable neuron ratio of up to 80.1% while preserving high verification accuracy. This indicates that the upper bound of the search space is significantly reduced.

Answer to RQ3: Experiments show that our method is more verification-friendly than other methods at higher and close accuracy levels, as it consistently maintains a high ratio of stable neurons, greater robustness, and requires less verification time.

Related Work

Traditional adversarial methods such as TRADES (Zhang et al. 2019b) and Madry (Madry et al. 2018) use adversarial training to improve the robustness of neural networks. However, these methods primarily impose constraints on the network's output, while our method imposes neuron behavior constraints on all the network layers. Recent works, such as HYDRA(Sehwag et al. 2020), integrate the pruning process with adversarial training, using a criterion that considers adversarial robustness during the pruning decision. This leads to more compact yet robust models. Wu et al. (Wu, Xia, and Wang 2020) also found that perturbing the weight can improve the robustness of the network.

Certified training (De Palma et al. 2022; Mirman, Gehr, and Vechev 2018; Jovanovic et al. 2022; Zhang et al. 2019a; Xu et al. 2020) introduces interval bound propagation (IBP) into the training process to improve network robustness but suffers from long training times.

The ReLU Stable method (Xiao et al. 2019) ensures that the upper and lower bounds of the neurons have the same sign, pushing them away from zero in the same direction. This method can also incorporate ternary loss to train a SAT-friendly network (Narodytska et al. 2019). In contrast, our method emphasizes the stability of neurons before and after perturbation, focusing on the consistency of neuron behavior. Furthermore, RS loss requires additional calculations for each neuron's bounds, whereas our method does not require extra information, making it easier to implement.

Linearity grafting (Chen et al. 2022c) replaces unstable neurons with linear neurons to improve the robustness of the network. However, this method modifies the network architecture, whereas our method preserves the architecture.

The MILP-based method (Baninajjar, Rezine, and Aminifar 2024) uses MILP to post-process the network and make it sparse, improving verification efficiency. However, this may be time-consuming.

Pruning-based methods (Xiao et al. 2019; Xu et al. 2024) heuristically prune inactive or unstable neurons with little impact on the network's performance. The bias shaping method (Xu et al. 2024) changes the bias of unstable neurons during training to stabilize them. These methods are either post-processing techniques or training tricks that can be incorporated with other methods to improve the verifiability of networks further.

Conclusion

In this work, we propose a novel training method to develop a verification-friendly neural network by preserving neuron behavior consistency. Our experiments on the MNIST, Fashion-MNIST, and CIFAR-10 datasets demonstrate that our method consistently enhances the network's verification-friendliness, as evidenced by a higher stable neuron ratio, comparable robustness, and faster verification speed across different perturbation radii. Additionally, when combined with existing methods, our method shows improved verification efficiency. Our method also accelerates the verification process while maintaining model accuracy, which is a result that is rarely achieved by existing methods. In future work, we plan to explore the application of our method to more complex network architectures and datasets, as well as explain the underlying mathematical machinery of our method in more detail.

Acknowledgements

This work is partly supported by CAS Project for Young Scientists in Basic Research, Grant No.YSBR-040, ISCAS New Cultivation Project ISCAS-PYFX-202201, ISCAS Basic Research ISCAS-JCZD-202302 and the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (Award ID: MOET32020-0004).

References

Bak, S. 2021. nnenum: Verification of ReLU neural networks with optimized abstraction refinement. In *Proceedings of the 13th International Symposium on NASA Formal Methods (NFM)*, 19–36. Springer.

Bak, S.; Liu, C.; and Johnson, T. T. 2021. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. *CoRR*, abs/2109.00498.

Baninajjar, A.; Rezine, A.; and Aminifar, A. 2024. VNN: Verification-Friendly Neural Networks with Hard Robustness Guarantees. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2846–2856. PMLR.

Bu, L.; Zhao, Z.; Duan, Y.; and Song, F. 2022. Taking Care of the Discretization Problem: A Comprehensive Study of the Discretization Problem and a Black-Box Adversarial Attack in Discrete Integer Domain. *IEEE Trans. Dependable Secur. Comput.*, 19(5): 3200–3217.

Chen, G.; Chen, S.; Fan, L.; Du, X.; Zhao, Z.; Song, F.; and Liu, Y. 2021. Who is Real Bob? Adversarial Attacks on Speaker Recognition Systems. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P)*, 694–711.

Chen, G.; Zhang, Y.; Zhao, Z.; and Song, F. 2023. QFA2SR: Query-Free Adversarial Transfer Attacks to Speaker Recognition Systems. In *Proceedings of the 32nd USENIX Security Symposium*.

Chen, G.; Zhao, Z.; Song, F.; Chen, S.; Fan, L.; ; Wang, F.; and Wang, J. 2022a. Towards Understanding and Mitigating Audio Adversarial Examples for Speaker Recognition. *IEEE Trans. Dependable Secur. Comput.*, 1–17.

Chen, G.; Zhao, Z.; Song, F.; Chen, S.; Fan, L.; and Liu, Y. 2022b. AS2T: Arbitrary Source-To-Target Adversarial Attack on Speaker Recognition Systems. *IEEE Trans. Dependable Secur. Comput.*, 1–17.

Chen, T.; Zhang, H.; Zhang, Z.; Chang, S.; Liu, S.; Chen, P.-Y.; and Wang, Z. 2022c. Linearity grafting: Relaxed neuron pruning helps certifiable robustness. In *Proceedings of the International Conference on Machine Learning (ICML)*, 3760–3772. PMLR.

De Palma, A.; Bunel, R.; Dvijotham, K.; Kumar, M. P.; and Stanforth, R. 2022. IBP regularization for verified adversarial robustness via branch-and-bound. *arXiv preprint arXiv:2206.14772*.

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

Ganin, Y.; Ustinova, E.; Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; Marchand, M.; and Lempitsky, V. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1): 2096–2030.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *Proceedings of the IEEE symposium on security and privacy (SP)*, 3–18. IEEE.

Gowal, S.; Dvijotham, K.; Stanforth, R.; Bunel, R.; Qin, C.; Uesato, J.; Arandjelovic, R.; Mann, T. A.; and Kohli, P. 2018. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. *CoRR*, abs/1810.12715.

Guo, X.; Wan, W.; Zhang, Z.; Zhang, M.; Song, F.; and Wen, X. 2021. Eager Falsification for Accelerating Robustness Verification of Deep Neural Networks. In Jin, Z.; Li, X.; Xiang, J.; Mariani, L.; Liu, T.; Yu, X.; and Ivaki, N., eds., *Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 345–356. IEEE.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety verification of deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, 3–29. Springer.

Jovanovic, N.; Balunovic, M.; Baader, M.; and Vechev, M. T. 2022. On the Paradox of Certified Training. *Trans. Mach. Learn. Res.*

Julian, K. D.; Kochenderfer, M. J.; and Owen, M. P. 2019. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3): 598–608.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, 97–117. Springer.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M. J.; and Barrett, C. W. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proceedings of the 31th International Conference on Computer Aided Verification (CAV)*, 443–452.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. *Technical report*, 3–54.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11): 2278–2324.

Liu, J.; Xing, Y.; Shi, X.; Song, F.; Xu, Z.; and Ming, Z. 2024a. Abstraction and Refinement: Towards Scalable and Exact Verification of Neural Networks. *ACM Trans. Softw. Eng. Methodol.*, 33(5): 129:1–129:35.

Liu, Z.; Zhao, Z.; Song, F.; Sun, J.; Yang, P.; Huang, X.; and Zhang, L. 2024b. Training Verification-Friendly Neural Networks via Neuron Behavior Consistency. arXiv:2412.13229.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant

to Adversarial Attacks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

Mirman, M.; Gehr, T.; and Vechev, M. 2018. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 3578–3586. PMLR.

Müller, M. N.; Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2023. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. arXiv:2212.10376.

Narodytska, N.; Zhang, H.; Gupta, A.; and Walsh, T. 2019. In search for a SAT-friendly binarized neural network architecture. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Sehwag, V.; Wang, S.; Mittal, P.; and Jana, S. 2020. Hydra: Pruning adversarially robust neural networks. *Advances in Neural Information Processing Systems*, 33: 19655–19666.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. T. 2019. An abstract domain for certifying neural networks. *PACMPL*, 3(POPL): 41:1–41:30.

Song, F.; Lei, Y.; Chen, S.; Fan, L.; and Liu, Y. 2021. Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers. *Int. J. Intell. Syst.*, 36(9): 5210–5240.

Tjeng, V.; Xiao, K. Y.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *7th International Conference on Learning Representations (ICLR)*.

Tran, H.-D.; Yang, X.; Manzanas Lopez, D.; Musau, P.; Nguyen, L. V.; Xiang, W.; Bak, S.; and Johnson, T. T. 2020. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, 3–17. Springer.

Urmson, C.; and Whittaker, W. 2008. Self-Driving Cars and the Urban Challenge. *IEEE Intell. Syst.*, 23(2): 66–68.

Wang, S.; Zhang, H.; Xu, K.; Lin, X.; Jana, S.; Hsieh, C.-J.; and Kolter, J. Z. 2021. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34: 29909–29921.

Wu, D.; Xia, S.-T.; and Wang, Y. 2020. Adversarial weight perturbation helps robust generalization. *Advances in neural information processing systems*, 33: 2958–2969.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.07747.

Xiao, K. Y.; Tjeng, V.; Shafiullah, N. M. M.; and Madry, A. 2019. Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability. In *7th International Conference on Learning Representations (ICLR)*.

Xu, D.; Mozumder, N. J.; Duong, H.; and Dwyer, M. B. 2024. Training for Verification: Increasing Neuron Stability to Scale DNN Verification. In Finkbeiner, B.; and Kovács, L., eds., *Proceedings of the 30th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 24–44. Springer.

Xu, K.; Shi, Z.; Zhang, H.; Wang, Y.; Chang, K.-W.; Huang, M.; Kailkhura, B.; Lin, X.; and Hsieh, C.-J. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33: 1129–1141.

Zhang, H.; Chen, H.; Xiao, C.; Gowal, S.; Stanforth, R.; Li, B.; Boning, D.; and Hsieh, C.-J. 2019a. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*.

Zhang, H.; Wang, S.; Xu, K.; Li, L.; Li, B.; Jana, S.; Hsieh, C.; and Kolter, J. Z. 2022a. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. In *NeurIPS*.

Zhang, H.; Yu, Y.; Jiao, J.; Xing, E. P.; Ghaoui, L. E.; and Jordan, M. I. 2019b. Theoretically Principled Trade-off between Robustness and Accuracy. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, 7472–7482. PMLR.

Zhang, Y.; Chen, G.; Song, F.; Sun, J.; and Dong, J. S. 2024. Certified Quantization Strategy Synthesis for Neural Networks. In Platzer, A.; Rozier, K. Y.; Pradella, M.; and Rossi, M., eds., *Proceedings of the 26th International Symposium on Formal Methods (FM)*, 343–362. Springer.

Zhang, Y.; Song, F.; and Sun, J. 2023. QEBVerif: Quantization Error Bound Verification of Neural Networks. In Enea, C.; and Lal, A., eds., *Proceedings of the 35th International Conference on Computer Aided Verification (CAV)*, volume 13965, 413–437. Springer.

Zhang, Y.; Zhao, Z.; Chen, G.; Song, F.; and Chen, T. 2021. BDD4BNN: A BDD-Based Quantitative Analysis Framework for Binarized Neural Networks. In Silva, A.; and Leino, K. R. M., eds., *Proceedings of the 33rd International Conference on Computer Aided Verification*, 175–200.

Zhang, Y.; Zhao, Z.; Chen, G.; Song, F.; and Chen, T. 2023. Precise Quantitative Analysis of Binarized Neural Networks: A BDD-based Approach. *ACM Trans. Softw. Eng. Methodol.*, 32(3): 62:1–62:51.

Zhang, Y.; Zhao, Z.; Chen, G.; Song, F.; Zhang, M.; Chen, T.; and Sun, J. 2022b. QVIP: An ILP-based Formal Verification Approach for Quantized Neural Networks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 82:1–82:13. ACM.

Zhao, Z.; Chen, G.; Wang, J.; Yang, Y.; Song, F.; and Sun, J. 2021. Attack as defense: characterizing adversarial examples using robustness. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 42–55.

Zhao, Z.; Zhang, Y.; Chen, G.; Song, F.; Chen, T.; and Liu, J. 2022. CLEVEREST: Accelerating CEGAR-based Neural Network Verification via Adversarial Attacks. In Singh, G.; and Urban, C., eds., *Proceedings of the 29th International Symposium on Static Analysis (SAS)*, 449–473. Springer.

Zhu, J.-Y.; Park, T.; Isola, P.; and Efros, A. A. 2017. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2242–2251.