2023 IEEE/ACM 11th International Conference on Formal Methods in Software Engineering (FormaliSE) | 979-8-3503-1263-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/FormaliSE58978.2023.00009

Verifying Binary Neural Networks on Continuous Input Space using Star Reachability

Mykhailo Ivashchenko
School of Computing
University of Nebraska
Lincoln, NE, United States
mivashchenko2@huskers.unl.edu

Sung Woo Choi
School of Computing
University of Nebraska
Lincoln, NE, United States
csw94056@gmail.com

Luan Viet Nguyen
Department of Computer Science
University of Dayton
Dayton, OH, United States
lnguyen1@udayton.edu

Hoang-Dung Tran
School of Computing
University of Nebraska
Lincoln, NE, United States
trhoangdung@gmail.com

***Abstract*—Deep Neural Networks (DNNs) have become a popular instrument for solving various real-world problems. DNNs' sophisticated structure allows them to learn complex representations and features. For this reason, Binary Neural Networks (BNNs) are widely used on edge devices, such as microcomputers. However, architecture specifics and floating-point number usage result in an increased computational operations complexity. Like other DNNs, BNNs are vulnerable to adversarial attacks; even a small perturbation to the input set may lead to an errant output. Unfortunately, only a few approaches have been proposed for verifying BNNs.**

This paper proposes an approach to verify BNNs on *continuous input space* using star reachability analysis. Our approach can compute both exact and overapproximate reachable sets of BNNs with Sign activation functions and use them for verification. The proposed approach is also efficient in constructing a complete set of counterexamples in case a network is unsafe. We implemented our approach in NNV, a neural network verification tool for DNNs and learning-enabled Cyber-Physical Systems. The experimental results show that our star-based approach is less conservative, more efficient, and scalable than the recent SMT-based method implemented in Marabou. We also provide a comparison with a quantization-based tool EEVBNN.

I. Introduction

Deep neural networks (DNNs) have become a popular technique for complex problems in various areas such as computer vision [1], and natural language processing [2]. However, real-world DNNs consist of large-size architecture with numerous parameters in a floating-point arithmetic format that complicates the deployment of DNNs on edge devices due to limited computation power and memory resources. To overcome this issue, simplified architectures, such as binary neural networks (BNNs) [3], [4] have been proposed. Unlike other DNNs, BNNs use only linear and Sign activation functions to reduce the number of floating-point arithmetic computations, which increases performance. Similar to DNNs, BNNs are vulnerable to adversarial attacks [5], [6] in which slightly changing the inputs can completely fool a well-trained and highly accurate network.

While being efficient and easy to deploy, BNNs are generally more challenging to train and verify because of a performance-accuracy trade-off [7]. Only a few verification methods have been proposed to deal with BNNs, and most of them require input quantization which omits an infinite number of possible input states. For instance, the BDD-based methods [8] perform quantitative robustness analysis of BNNs based on constructing equivalent binary decision diagrams from the networks with quantized input data. The EEVBNN tool [9] can perform exact verification for BNNs with quantized input space by converting the networks into SAT problems. It is important to emphasize that input quantization is an extra man-made step to ease neural network verification. By reducing an input space to a discrete set, quantization makes the verification process more efficient with respect to computational time and resources. However, input quantization causes a decrease in the accuracy of neural networks after training, especially in those used for control purposes [10], [11], because originally, the networks were trained to work on continuous input space. Verifying BNNs on continuous input space allows accounting for all possible input states, increasing the certainty of the performed verification. The SAT/SMT-based verification methods proposed in [12], [13] can verify BNNs on continuous input space, but they are not scalable.

This paper proposes a complementary approach for verifying BNNs without input quantization using star reachability [14], [15], i.e., directly dealing with continuous input space and the original BNNs. We extend the star set approach to perform exact and overapproximate analysis of Sign activation functions in BNNs. It is done by introducing a new *stepSign* operation for both exact and overapproximate analysis algorithms. We perform the exact analysis by applying the Sign operation to each neuron individually, while the overapproximate analysis uses an n-dimensional box as an approximation. Verification using exact reachability is sound and complete but computationally expensive. Meanwhile, verification with overapproximate reachability usually guarantees only the soundness of the results, but it is much less expensive in computation and offers better scalability. Interestingly, both soundness and completeness can be achievable using overapproximate reachability in many cases, with our new method performing backward counterexamples localization and random sampling. Extending from the original Star and ImageStar-based verification [14], [16], our proposed approach can verify both binary feedforward neural networks (BFFNNs) and binary convolutional neural networks (BCNNs) and is fully parallelizable to improve scalability.

We implemented the proposed approach in NNV, a veri-

979-8-3503-1263-8/23/$31.00 ©2023 IEEE
DOI 10.1109/FormaliSE58978.2023.00009

fication tool for DNNs and learning-enabled Cyber-Physical Systems [17]. We evaluate our approach in comparison with: the SMT-based [12] method implemented in Marabou [18] for BNNs with continuous input space, and the SAT-based method implemented in EEVBNN [9] with quantized input space. The experiments show that our approach is significantly faster than Marabou on their proposed benchmarks. For instance, the exact and overapproximate verification can be $3600\times$ and $5700\times$ faster than Marabou on a small network with 220 neurons. Additionally, our approach is also less conservative and more efficient than Marabou when dealing with severe L_∞ norm attacks, i.e., attacks with large disturbance bound δ. For example, Marabou reaches a timeout of 5,000 seconds when verifying the small network with $\delta > 1$ while our approach can prove the robustness of the network within 1 second. We note that our approach is more scalable than Marabou as it can verify larger networks that Marabou cannot verify. However, Star-based verification performs worse compared to the SAT-based method implemented in EEVBNN [9] on the given benchmarks. This result is understandable as EEVBNN quantizes the input space, while Star-based reachability is designed to operate with continuous input space. In sum, the main contributions of this paper are:

- The extension of the star reachability algorithms for verifying BNNs on continuous input space.
- The implementation of exact and overapproximate reachability algorithms in NNV that are publicly available for further evaluation and comparison [19].
- A thorough evaluation of the proposed approach in comparison with the existing ones on a set of benchmarks.

II. Background

A. Binary Neural Networks

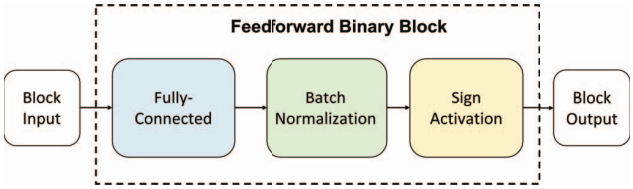
A k-layer binary neural network consists of an input layer, $k-1$ hidden layers, and an output layer. A BNN may include fully-connected, batch normalization, convolutional, pooling, and flatten layers. These layers can be combined with a *Sign layer* (performing sign operation defined in the following) to form a binary block. In this paper, we are interested in two types of binary blocks: the feedforward binary block and the convolutional binary block depicted in Figure 1. These blocks form a BFFNN and BCNN, respectively.

Definition II.1 (Sign function)**.** *Let x be an input, σ be the activation function, y be the result of application σ to x. For $\sigma(\cdot) = Sign(\cdot)$:*

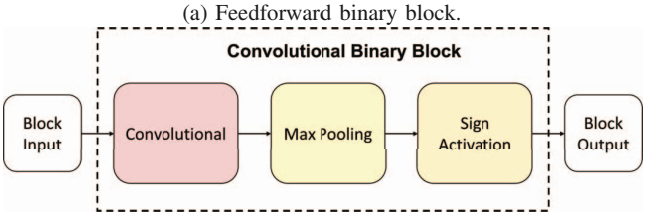
$$y = \sigma(x) \rightarrow \begin{cases} 1, x \geq 0; \\ -1, x < 0. \end{cases}$$

Definition II.2 (Execution of a BNN)**.** *Given a BNN of k layers $\{L_1, \ldots, L_k\}$, execution of the network computes an output y corresponding to an input vector x as follows:*

$$y = f_k(f_{k-1}(\ldots(f_1(x)))), \tag{1}$$



(a) Feedforward binary block.



(b) Convolutional binary block.

Fig. 1: Binary blocks.

where f_i is the operation executed by the i^{th} layer. This operation is either affine mapping, sign activating, average pooling, or convolution.

Definition II.3 (Reachable Set of a BNN)**.** *The output (reachable set) $\mathcal{R}_\mathcal{N}$ of a BNN $\mathcal{N} = \{L_i\}$, $i = 1, 2, ..., k$ corresponding to a convex set I is defined incrementally as:*

$$\begin{aligned} \mathcal{R}_1 &\triangleq \{y^{[1]} \,|\, y^{[1]} = f_1(x), x \in I\}, \\ \mathcal{R}_2 &\triangleq \{y^{[2]} \,|\, y^{[2]} = f_2(y^{[1]}), y^{[1]} \in \mathcal{R}_1\}, \\ &\vdots \\ \mathcal{R}_\mathcal{N} = \mathcal{R}_k &\triangleq \{y^{[k]} \,|\, y^{[k]} = f_k(y^{[k-1]}), y^{[k-1]} \in \mathcal{R}_{k-1}\}, \end{aligned}$$

where $f_i(\cdot)$ is a function representing the operation of the i^{th} layer.

B. Set Representation

This section reviews the definitions of star [14], and ImageStar [17] set representation and their basic properties that are used to compute the reachable set of BNNs.

Definition II.4 (Generalized Star Set [14])**.** *A generalized star set (or simply star) Θ is a tuple $\langle c, V, P\rangle$ created based on input vector $x \in \mathbb{R}^n$, where $c \in \mathbb{R}^n$ is a center vector, $V = [v_1, v_2, \cdots, v_m]$ consists a set of m basis vectors $v \in \mathbb{R}^n$, and predicate $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ bounds. $P(\alpha) \triangleq C\alpha \leq d$ is a conjunction of p linear constraints, where $C \in \mathbb{R}^{p\times m}, d \in \mathbb{R}^p, \alpha$ is an unknown column vector with m entries. The set of states can be represented as:*

$$[\![\Theta]\!] = \{x \,|\, x = c + \sum_{i=1}^{m} \alpha_i v_i \text{ such that } P(\alpha) = \top\} \tag{2}$$

Remark II.1. *A star is an empty set, i.e., $\Theta = \emptyset$ if and only if the predicate $P(\alpha)$ is infeasible. Sometimes we will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ in the rest of this paper.*

Proposition II.1 (Affine Mapping of a Star)**.** *Given a star set $\Theta = \langle c, V, P\rangle$, an affine mapping of the star Θ with*

the affine mapping matrix W and offset vector b defined by $\bar{\Theta} = \{y \mid y = Wx + b,\ x \in \Theta\}$ is another star with the following characteristics: $\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P} \rangle,\ \bar{c} = Wc + b,\ \bar{v} = \{Wv_1, Wv_2, \cdots, Wv_m\},\ \bar{P} \equiv P$.

Figure 2 presents a toy example of the affine mapping operation over a 2D Star.

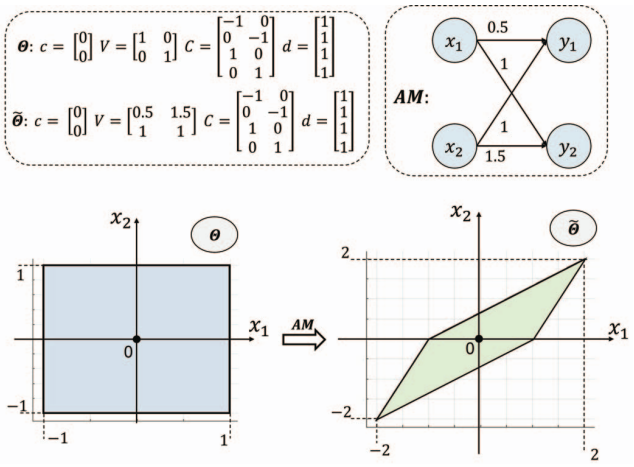


Fig. 2: A toy example of an Affine Mapping operation applied to a 2D Star. Star S is the input set, Star $\bar{S}$ is the output set, AM is the Affine Mapping operation with a 2×2 weight matrix and no bias.

Proposition II.2 (Star and Half-space Intersection)**.** *The intersection of a star $\Theta \triangleq \langle c, V, P \rangle$ and a half-space $\mathcal{H} \triangleq \{x \mid Hx \leq g\}$ is another star with following characteristics.*

$$\bar{\Theta} = \Theta \cap \mathcal{H} = \langle \bar{c}, \bar{V}, \bar{P} \rangle,\ \bar{c} = c,\ \bar{V} = V,\ \bar{P} = P \wedge P',$$
$$P'(\alpha) \triangleq (H \times V_m)\alpha \leq g - H \times c, V_m = [v_1\ v_2 \cdots v_m].$$

Proposition II.3 (Exact Range)**.** *Given a star set $\Theta = \langle c, V, P \rangle$, the range of the i^{th} state $x[i]$ of the star set can be found by solving the following linear programming optimization problems:*

$$x[i]_{min} = min(c[i] + \Sigma_{j=1}^{m} v_j[i]\alpha_j),\ s.t.\ P(\alpha) \triangleq C\alpha \leq d,$$
$$x[i]_{max} = max(c[i] + \Sigma_{j=1}^{m} v_j[i]\alpha_j),\ s.t.\ P(\alpha) \triangleq C\alpha \leq d.$$

Proposition II.4 (Estimated Range)**.** *Given a star set $\Theta = \langle c, V, P \rangle$, let l and u be the lower and upper bound vectors of the predicate variables, the range of the i^{th} state $x[i]$ of the star set can be estimated quickly without solving the linear programming optimization problems as follows:*

$$x[i]^{est}_{min} = c[i] + \sum_{\substack{j=1 \\ v_j[i] \geq 0}}^{m} l[i]v_j[i] + \sum_{\substack{k=1 \\ v_k[i] \leq 0}}^{m} u[i]v_k[i],$$
$$x[i]^{est}_{max} = c[i] + \sum_{\substack{j=1 \\ v_j[i] \geq 0}}^{m} u[i]v_j[i] + \sum_{\substack{k=1 \\ v_k[i] \leq 0}}^{m} l[i]v_k[i].$$

Reachability analysis of BNNs involves convolution and batch normalization operations that can be done efficiently using ImageStar [16]. Therefore, in this paper, we switch between ImageStar and Star representations when computing the reachable set of BNNs.

Definition II.5 (ImageStar [16])**.** *An ImageStar Θ_{IS} is a tuple $\langle c, V, P \rangle$, where $c \in \mathbb{R}^{h \times w \times nc}$ is the anchor image, $V = v_1, v_2, \cdots, v_m$ is a set of m images in $\mathbb{R}^{h \times w \times nc}$ called generator images, $P : \mathbb{R} \rightarrow \{\top, \bot\}$ is a predicate, and h, w, nc are the height, width and the number of channels of the images respectively. The generator images are arranged to form the ImageStar's $h \times w \times nc \times m$ basis array. The set of images represented by the ImageStar is given as follows:*

$$[\![\Theta_{IS}]\!] = \{x \mid x = c + \sum_{i=1}^{m} \alpha_i v_i \text{ such that } P(\alpha) = \top\} \quad (3)$$

Definition II.6 (ImageStar to Star)**.** *Let $\Theta_{IS} = \langle c, V, P \rangle$ be an ImageStar, where $c \in \mathbb{R}^{h \times w \times nc}$, $V = v_1, v_2, \cdots, v_m$ in $\mathbb{R}^{h \times w \times nc}$ are generator images, $P : \mathbb{R} \rightarrow \{\top, \bot\}$ is a predicate. There exists a sound conversion from Θ_{IS} to a Star $\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P} \rangle$, where:*

$$\bar{c} = reshape(c, (h \cdot w \cdot nc, 1)) \in \mathbb{R}^{(h \cdot w \cdot nc) \times 1},$$
$$\bar{V} = [\bar{v}_1, \bar{v}_2, \cdots, \bar{v}_m], \bar{v}_i = reshape(v_i, (h \cdot w \cdot nc, 1)),$$
$$\bar{P} = P.$$

Remark II.2. *A backward conversion from $\bar{\Theta}$ to Θ_{IS} can be performed similarly and is also sound.*

III. Reachability Analysis of BNNs

This section extends the reachability analysis methods of ReLU networks [14] to BNNs. Given a star input set, we compute the output set of a BNN layer-by-layer. We switch between ImageStar (Definition II.5) and Star set representations (Definitions II.4, II.6) to compute the reachable set of fully-connected, convolutional, max-pooling, and batch normalization layers. The reachability analysis of these layers can be found in [16]. In this paper, we focus on the reachability of a Sign layer with a star input set.

A. Exact Reachability

Similar to ReLU networks, we can compute the exact reachable set of a Sign layer with a star input set by applying a sequence of *stepSign* operations. Let $Sign_i$ be the stepSign operation applied to the i^{th} neuron. Then the reachable set of a Sign layer L with n-neurons and a star input set Θ can be computed precisely as:

$$\mathcal{R}_L = Sign_n(Sign_{n-1}(\ldots(Sign_1(\Theta)))).$$

The stepSign operation computes intermediate reachable sets of a Sign layer by applying the Sign activation function (Definition II.1) to a specific neuron. Depending on the sign of that neuron, the output can be 1 or -1. Given a star input set Θ, the range of the input x_i to the i^{th} neuron can be computed exactly using Proposition II.3 or overapproximately

estimated by Proposition II.4. Using the exact or estimated range, we can determine the sign of the neuron input. The following proposition describes the stepSign operation at i^{th} neuron given a star input set Θ.

Proposition III.1 (exact stepSign operation)**.** *Given a star input set $\Theta = \langle c, V, P\rangle$, let l_i and u_i be the lower bound and upper bound of the i^{th} input, the stepSign operation at the i^{th} neuron produces an intermediate reachable set as follows:*

- *If $0 \leq l_i \leq x_i$, then $Sign_i(\Theta) = \bar{\Theta} = \langle \bar{c}, \bar{V}, P\rangle, \bar{c} = c, \bar{c}(i) = 1, \bar{V} = V, \bar{V}(i,:) = 0$.*
- *If $x_i \leq u_i < 0$, then $Sign_i(\Theta) = \bar{\Theta} = \langle \bar{c}, \bar{V}, P\rangle, \bar{c} = c, \bar{c}(i) = -1, \bar{V} = V, \bar{V}(i,:) = 0$.*
- *If $l_i \leq x_i \leq u_i$, $l_i < 0$ and $u_i \geq 0$, then $Sign_i(\Theta) = [\bar{\Theta}_1, \bar{\Theta}_2]$, in which $\bar{\Theta}_1 = \langle \bar{c}_1, \bar{V}_1, P \wedge x_i \geq 0\rangle, \bar{c}_1 = c, \bar{c}_1(i) = 1, \bar{V}_1 = V, \bar{V}_1(i,:) = 0$, and $\bar{\Theta}_2 = \langle \bar{c}_2, \bar{V}_2, P \wedge x_i < 0\rangle, \bar{c}_2 = c, \bar{c}_2(i) = -1, \bar{V}_2 = V, \bar{V}_2(i,:) = 0$.*

Proof. $\forall x = [x_1, x_2, \ldots, x_i, \ldots, x_n]^T \in \Theta$, the stepSign operation at the i^{th} neuron applies the Sign activation function only on x_i and keeps other inputs the same. Depending on the sign of x_i, its corresponding output can be 1 or -1. Therefore, if $0 \leq l_i \leq x_i$, applying the sign activation function at x_i maps its value to $y_i = 1$. The corresponding output vector is $y = [x_1, x_2, \ldots, y_i = 1, \ldots, x_n]^T \in \bar{\Theta}$ defined above. Similar derivation applies for $x_i \leq u_i < 0$. The most interesting case is when $l_i \leq x_i \leq u_i$, $l_i < 0$, and $u_i \geq 0$, i.e., x_i is either negative or positive. The stepSign operation splits this into two sub-cases $x_i \geq 0$ and $x_i < 0$ and processes as normal. This results in two new star sets in the intermediate reachable set defined above. □

Figure 3 presents a toy example of the exact stepSign operation applied to a 2D Star.

Lemma III.1. *The number of star sets in the exact reachability of a Sign layer grows exponentially in the worst case.*

Proof. Let n be the dimension of the star input set to a Sign layer. Applying exact reachability on this input set may produce 2^n new star sets in the worst case if splitting happens at all individual inputs. □

B. Overapproximate Reachability

The exact reachability suffers from the explosion in the number of star sets, which leads to large memory consumption, more computation time, and limited scalability. Overapproximate reachability can overcome this challenge. The idea is to overapproximate the sign activation function to neurons where splitting occurs, i.e., we overapproximate two new star sets $\bar{\Theta}_1$ and $\bar{\Theta}_2$ in Proposition III.1 by a single star set $\bar{\Theta}$. This can be done by introducing a new predicate variable α_{m+1} to represent the output y_i (can be -1 or 1 in this case). We overapproximate y_i by adding new constraints $-1 \leq \alpha_{m+1} \leq 1$, i.e., overapproximate two points -1 and 1 by a segment $[-1, 1]$. The following proposition describes how an overapproximate stepSign operation works.

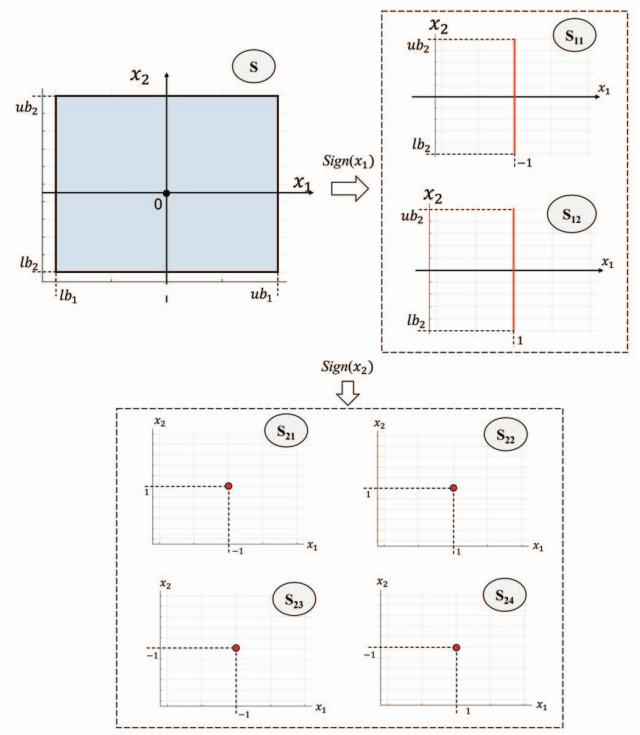


Fig. 3: A toy example of the exact analysis *stepSign* operation performed on a 2D Star set. $Sign(x_1)$ applies the sign operation over the x_1 axis. $Sign(x_2)$ applies the sign operation over the x_2 axis. *stepSign*$(S) = \{S_{21}, S_{22}, S_{23}, S_{24}\}$.

Proposition III.2 (overapproximate stepSign operation)**.** *Given a star input set $\Theta = \langle c, V, P\rangle$, let l_i and u_i be the lower bound and upper bound of the i^{th} input, the overapproximate stepSign operation at the i^{th} neuron produces an intermediate reachable set as follows:*

- *If $0 \leq l_i \leq x_i$, then $Sign_i(\Theta) = \bar{\Theta} = \langle \bar{c}, \bar{V}, P\rangle, \bar{c} = c, \bar{c}(i) = 1, \bar{V} = V, \bar{V}(i,:) = 0$.*
- *If $x_i \leq u_i < 0$, then $Sign_i(\Theta) = \bar{\Theta} = \langle \bar{c}, \bar{V}, P\rangle, \bar{c} = c, \bar{c}(i) = -1, \bar{V} = V, \bar{V}(i,:) = 0$.*
- *If $l_i \leq x_i \leq u_i$, $l_i < 0$ and $u_i \geq 0$, then $Sign_i(\Theta) \subset \bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}\rangle$, in which: $\bar{c} = c, \bar{c}(i) = 0, \bar{V} = [V\ v_{m+1}], v_{m+1} = e_i, \bar{V}(i, 1:m) = 0, \bar{P} = P \wedge -1 \leq \alpha_{m+1} \leq 1$.*

Proof. $\forall x = [x_1, x_2, \ldots, x_i, \ldots, x_n]^T \in \Theta$, $x_i = c(i) + \Sigma_{i=1}^{m} \alpha_j v_j(i)$. When $l_i \geq 0$ or $u_i < 0$, exact reachability produces only a single star set. Thus, no overapproximation is needed. When $l_i \leq x_i \leq u_i$, $l_i < 0$, and $u_i \geq 0$, i.e., x_i is either negative or positive. The corresponding output y_i is either -1 or 1. This can be overapproximated by a single set $y_i \subset Y = \{\tilde{y}_i = \alpha_{m+1} | -1 \leq \alpha_{m+1} \leq 1\}$. The overapproximate output can be represented in a star form as $\tilde{y}_i = 0 + \Sigma_{i=1}^{m} 0 \times \alpha_j + \alpha_{m+1} e_i$, where e_i is the i^{th} standard basis vector of the $\mathbb{R}^n$ space. Note that other individual outputs are the same as their corresponding inputs. Consequently,

a single star set $\bar{\Theta}$ defined above can be constructed to overapproximate the i^{th} output set in this case. Figure 4 presents a toy example of the approximate stepSign operation applied to a 2D Star. □

Remark III.1. *The number of new predicate variables and their associated constraints grows linearly in overapproximate reachability of a Sign layer.*

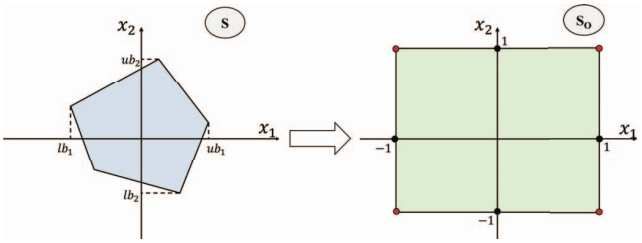


Fig. 4: A toy example of the overapproximate analysis *stepSign* operation performed on a 2D Star set. The input Star S is overapproximated with a 2D box, where $x_1 \in [-1, 1], x_2 \in [-1, 1]$. $stepSign(S) = S_O$.

C. Reachability Algorithm

Algorithm 1 summarizes the reachability analysis of a BNN $\mathcal{N}$ with k layers. We compute the reachable set layer-by-layer. The last layer's reachable set is returned as the output set of the network. We note that reachability analyses of fully-connected, batch normalization, and convolutional layers are done exactly regardless of reachability methods, i.e., exact or overapproximate [16]. This means the reachability method selection only applies to max-pooling and sign layers.

Algorithm 1 Reachability of BNN

1: **procedure** REACH($\mathcal{N}, \Theta, method$) ▷ Network, star input set, reachability method
2: $\mathcal{R} \leftarrow \Theta$
3: **for** $<i = 1$ to $k>$ **do**
4: $L_i \leftarrow \mathcal{N}.Layers(i)$
5: $\mathcal{R} \leftarrow L_i.reach(\mathcal{R}, method)$ ▷ Reachability of an individual layer
6: **return** $\mathcal{R}$ ▷ The network's reachable set

Implementation Highlight. In the exact analysis, a max-pooling or a Sign layer may produce multiple output sets from an input set. Therefore, we exploit the power of parallel computing to process multiple inputs simultaneously at a specific layer to speed up the verification. In addition, we usually use estimated ranges to determine the sign of individual inputs in the reachability of a Sign layer to minimize unnecessary optimization time in the analysis. For example, if we know the estimated lower bound of x_i is $\tilde{l}_i \geq 0$, then we do not need to find its exact lower bound l_i for the analysis as it is always non-negative $x_i \geq l_i \geq \tilde{l}_i \geq 0$. Finally, we note that if a BNN is a BFFNN, a more efficient implementation using Depth First Search (DFS) with exact reachability [15] can be used to verify the network. Compared to the Breadth-First Search (BFS) implementation in this paper to handle both BFFNNs and BCNNs, DFS is faster and more memory-efficient in searching a counterexample when verifying the network. The algorithm will stop immediately once a counterexample is found.

IV. Verification

Using the reachable set computed in the previous section, verifying the safety of BNNs defined in the following is straightforward.

Definition IV.1 (Safety Verification of a BNN)**.** *Given a binary neural network $\mathcal{N}$, and an unsafe specification $\mathcal{U}$ defined by a set of linear constraints on the network's outputs $\mathcal{U} \triangleq \{y \,|\, Cy \leq d\}$, the network is called to be safe corresponding to the input set Θ, if and only if $\mathcal{R} \cap \mathcal{U} = \emptyset$, where $\mathcal{R}$ is the network's reachable set, i.e., $\mathcal{R} = \mathcal{N}(\Theta)$. Otherwise, the neural network is unsafe.*

Counterexamples Construction. Similar to verification of ReLU networks [14], we can construct a complete set of counterexamples that makes a BNN unsafe if the exact reachability method is used. This is described in the following lemma.

Lemma IV.1. *Let $\mathcal{R} = [\Theta_1, \Theta_2, \ldots, \Theta_N]$ be the exact reachable set of a BNN $\mathcal{N}$ with a star input set $\Theta = \langle c, V, P \rangle$, i.e., $\mathcal{R} = \mathcal{N}(\Theta)$, and $\mathcal{U} \triangleq \{y \,|\, Cy \leq d\}$ be the unsafe specification of the network. If the network is unsafe, i.e., $\mathcal{R} \cap \mathcal{U} \neq \emptyset$, then a complete set of counterexample inputs $\mathcal{C}$ is computed as follows:*

- $\forall k = 1, 2, \ldots, N,\ \Theta_k \cap \mathcal{U} = \Theta'_k = \langle c'_k, V'_k, P'_k \rangle \neq \emptyset$ (Proposition II.2)
- $\mathcal{C}_k = \langle \Theta.c, \Theta.V, P'_k \rangle,\ \mathcal{C} \leftarrow \mathcal{C}_k$.

Proof. In the exact reachability of a BNN, the input set and output set are defined based on the same set of predicate variables unchanged in the computation. When splitting occurs, new constraints on the predicate variables are added (for example, Proposition III.1 for a Sign layer). Therefore, a star set in the network's reachable set contains all constraints appearing in the input set, i.e., $\Theta_k.P_k \subseteq \Theta.P$. When a star set Θ_k in the network's reachable set intersects with the unsafe region $\mathcal{U}$, the intersection is an unsafe output set of the network, which is also a star set $\Theta'_k = \langle c'_k, V'_k, P'_k \rangle$ (Proposition II.2). Importantly, we have $P'_k \subseteq P_k \subseteq P$. Therefore, any input vectors corresponding to any predicate vectors $\alpha = [\alpha_1, \ldots, \alpha_m]^T \in P'_k$ cause the network to be unsafe. In other words, the star $\mathcal{C}_k = \langle \Theta.c, \Theta.V, P'_k \rangle$ is a set of counterexamples of the network. We can construct a complete set of counterexamples by checking the intersection of all Star sets in the reachable set with the unsafe region $\mathcal{U}$. □

Finding counterexamples with overapproximate reachability. As the exact reachability of a BNN is expensive, the overapproximate reachability is usually used to verify the safety of a large network. It produces a single star set $\mathcal{R} = \langle c', V', P' \rangle$, which is an overapproximation of the exact reachable set of

the network. If the overapproximate reachable set $\mathcal{R}$ intersects with the unsafe region $\mathcal{U}$, it creates a *potentially unsafe output set* $\mathcal{R}_{unsafe} = \langle c', V', P_{unsafe} \rangle$ (Proposition II.2). However, the actual outputs of the network are not necessarily in this potentially unsafe output set due to the overapproximation error. Therefore, to prove that the network is unsafe, we need to find a single actual output y_{actual} that relies upon the potentially unsafe output set, i.e., $y_{actual} \in \mathcal{R}_{unsafe}$. To do that, we randomly sample the P_{unsafe} to create a collection of predicate vectors $\alpha' = [\alpha_1, \alpha_2, \ldots, \alpha_m, \alpha_{m+1}, \ldots, \alpha_M]^T$. Here we assume that $(M - m)$ new predicate variables are introduced in the overapproximate reachability process. We take the first part of α' to form a collection of predicate vectors of the input set $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_m]$. From these predicate vectors, we construct a collection of input vectors $x = c + \Sigma_{i=1}^{m} \alpha_i v_i$. We execute the network to find the outputs corresponding to these input vectors. Finally, we check if there is an obtained output that relies upon the potentially unsafe output set $\mathcal{R}$. If yes, then the network is unsafe. If not, then the network's safety is unknown. The number of sampling iterations we perform per example equals 50.

Soundness and Completeness. The soundness and completeness of our verification approach using star reachability are given in the following lemma.

Lemma IV.2. *Verification of BNNs is sound and complete if the exact reachability method is used. Otherwise, it is sound but may or may not be complete.*

Proof. Using the exact reachability, we can always prove if the network is safe or construct a complete set of counterexamples when the network is unsafe (Lemma IV.1). Therefore, the verification of BNNs is sound and complete in this case. When the overapproximate reachability method is used, we may prove the network is safe using the overapproximate reachable set. However, when the overapproximate reachable set intersects with the unsafe region, we may or may not find counterexamples (the method is discussed above). Therefore, the verification is sound but may or may not be complete when using the overapproximate reachability. □

V. Evaluation

The proposed verification approach is implemented in NNV [17], a verification tool for deep neural networks and learning-enabled cyber-physical systems. It is evaluated and compared with the SMT-based approach [12] implemented in Marabou [18] on a set of benchmarks. The experiments were performed on a computer with the following configurations: Intel® Core™ i7-6950X CPU @ 3.00GHz × 20 processors, 62.7 GiB memory, 64-bit Ubuntu 18.04.6 LTS OS. Note that we did not use parallel computing to speed up our method when comparing it to Marabou and EEVBNN.

A. Star Reachability vs. Marabou SAT-based Method

Experiment Set Up. The SMT-based method [12] implemented in Marabou [18] presents verification results for two neural networks: a BFFN (MLP0) and a BCNN (XNOR0). MLP0 is trained to classify images from the MNIST dataset [20]. MNIST consists of 28×28 handwritten images from 0 to 9. XNOR0 is trained to classify images from the FMNIST [21] dataset. FMNIST consists of 28×28 images of clothes from ten different classes (each class is assigned an index from 0 to 9). The MLP experiment includes 500 examples of L_∞ norm attack for 10 disturbance values (50 examples per value): $\delta = \{0.1, 0.15, 0.2, 0.3, 0.5, 1, 3, 5, 10, 15\}$, i.e., $\|x' - x\| \leq \delta$, where x and x' are the original and the adversarial images, respectively. The XNOR experiment includes 300 examples of L_∞ norm attack for 6 disturbance values (50 examples per value): $\delta = \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$. The presented experiments for MLP0 and XNOR0 use the same examples provided by the Marabou paper [12]. Note that for the MLP0 experiment, we omit some examples as their classification labels do not correspond to respective ground-truth labels. To evaluate the scalability of the proposed approach, we trained 4 additional models for the experiment. The networks' information is presented in Table I. Note that we use the same Marabou configuration as the original paper: weighted sum layer elimination and polarity-based splitting are turned on, and symbolic bound tightening and LP relaxation are turned off. Note that the first binary block in every model omits the Sign layer to improve verification.

Network	Architecture	Accuracy	Type
MLP0	$(50 \times 4) : (10 \times 2)$	90%	BFFNN
XNOR0	3CB:1FB	75%	BCNN
MLP1	$(200 \times 2) : (100 \times 2) : (50 \times 2) : (10 \times 2)$	96%	BFFNN
MLP2	$(200 \times 3) : (100 \times 2) : (50 \times 2) : (10 \times 2)$	96%	BFFNN
MLP3	$(200 \times 3) : (100 \times 3) : (50 \times 2) : (10 \times 2)$	96%	BFFNN
MLP4	$(200 \times 4) : (100 \times 3) : (50 \times 2) : (10 \times 2)$	96%	BFFNN

b

TABLE I: The architectures of MLP networks. MLP0, XNOR0 are the networks proposed by [12]. MLP1-4 are the additionally trained networks. "CB" is a convolutional binary block. "FB" is a feedfoward binary block.

Verification Results. The verification results of the SMT-based method's proposed benchmarks (the Marabou method), MLP0 and XNOR0, are presented in Tables II and IV. We note that on the given examples, the exact reachability algorithm does not run into an exponential explosion. For this reason, the number of solved examples for the exact and overapproximate algorithms is the same. The same analysis for additionally trained networks MLP1-4 is given in Table III. A mismatch between the star reachability (NNV) and the SMT-based (Marabou) approaches and a potential soundness bug are illustrated in Figure 5.

Timing Performance. Our thorough experiments show that the proposed verification approach using star reachability outperforms the SMT-based method on all benchmarks. The exact Star method is significantly faster than the SMT-based approach when verifying BFFNN networks. For example, on the MLP0 network of 220 neurons (Table II), the exact star method is $2261\times$ and $2888\times$ faster than Marabou when proving UNSAT (robust or safe) and SAT (not robust or not safe) examples with the disturbance bound $\delta = 1$. Importantly,

δ	Marabou					Exact-Star				Approx-Star			
	Time(s)		#Sol			Time(s)		#Sol		Time(s)		#Sol	
	UN	S	UN	S	UK	UN	S	UN	S	UN	S	UN	S
0.1	7.42	68	47	1	0	0.8	0.8	48	0	0.5	0.5	48	0
0.15	13.9	16	40	2	0	0.9	0.9	41	1	0.5	0.5	41	1
0.2	13.06	115	43	1	0	0.9	0.9	44	0	0.5	0.5	44	0
0.3	69.69	128	40	3	0	0.9	0.9	42	1	0.5	0.5	42	1
0.5	457.29	314	33	9	0	0.9	0.9	41	1	0.5	0.5	41	1
1	1809.09	2889	25	13	8	0.8	0.8	42	4	0.5	0.5	42	4
3	TO	2432	0	25	14	0.8	0.8	36	3	0.5	0.5	36	3
5	TO	702	0	43	0	0.8	0.8	25	18	0.5	0.5	25	18
10	TO	441	0	40	0	0.8	0.8	18	22	0.5	0.5	18	22
15	TO	528	0	49	0	0.8	0.8	20	29	0.5	0.5	20	29

TABLE II: Verification results of MLP0 network. *Notation*: δ represents disturbance bound values, Time(s) represents average computation time for each value of δ, '#Sol' is the number of solved examples, UN = UNSAT indicates that the network is robust (safe) under the attack, S = SAT indicates that the network is not robust (unsafe), UK = UNKNOWN indicates that the example wasn't solved in the given time, TO = Timeout set as 5000 seconds. The red color is used to highlight the difference in the number of verified examples. The blue color is used to highlight the difference in verification time.

a large disturbance bound δ does not affect much the timing performance of the exact star method while significantly damaging the performance of Marabou. For example, with $\delta = 10$, the exact star method can still prove the robustness of the MLP0 network for all examples within 1 second while Marabou reaches a timeout set as 5000 seconds for all $\delta \geq 3$. Impressively, with a large network MLP4 of 1220 neurons (Table I) and a large disturbance bound $\delta = 15$, our exact verification approach can still prove the robustness of the network for all 50 examples with an average time of ≈ 3.2 seconds for each example (Table III). Note that Marabou reaches timeout set as 5000 seconds for all newly trained networks even with a small disturbance bound, e.g., MLP1 with $\delta = 0.1$. We want to emphasize that the exact verification approach does not perform well with BCNNs. The main reason is many splittings occur at max-pooling layers, which leads to the explosion in time and memory consumption in reachable set computation [16].

The overapproximate star reachability is faster than the exact one. More importantly, it performs well for both BFFNN and BCNN networks. For MLP0 BFFNN network (Table II), when $\delta = 1$, the overapproximate star method is $3618\times$ and $5778\times$ faster than Marabou on proving UNSAT and SAT examples. On XNOR0 BCNN network (Table IV), the overapproximate star method is $44.6\times$ and $3877.25\times$ faster than Marabou on proving UNSAT and SAT examples. Similar to the exact approach, large disturbance bounds do not affect much the timing performance of the overapproximate verification method.

Conservativeness. The conservativeness of a method can be quantified as the percentage of the provable cases (UNSAT + SAT) over the total number of examples (UNSAT + SAT + UNKNOWN). The smaller the resulting value, the more conservative the method is. On small networks, i.e., MLP0 (Table II) and XNOR0 (Table IV), our approach is less conservative than Marabou especially when dealing with a large disturbance bound δ. For example, on the MLP0 BFFNN network, for $\delta = 1$, Marabou proves $(25+13)/46 \approx 82.6\%$ of the cases while our approach (both exact and overapproximate) proves 100% of the cases. On the XNOR0 BCNN network, for $\delta = 0.3$, Marabou proves only $(20 + 7)/50 = 54\%$ while ours is $(30 + 12)/50 = 84\%$. Importantly, on large networks, i.e., MLP1-4 (Table III), Marabou cannot prove any cases, i.e., 0%, while our exact method proves 100% cases for MLP1-4 and the overapproximate one also proves $\geq 96\%$ cases for MLP1-4.

Scalability. The experiments show that our verification approach is more scalable than Marabou, especially when dealing with large disturbance bound. The exact verification method can handle a large BFFNN network of 1220 neurons (MLP4 in Table III) while Marabou can verify a small network of 220 neurons (MLP0) with small disturbance bound $\delta \leq 1$ (Table II) in the set timeout. The overapproximate verification method is more scalable than Marabou when dealing with both BFFNNs and BCNNs. It can handle large networks while Marabou cannot (Table III).

Inconsistency. When comparing the results with Marabou, we discovered that certain examples were classified as SAT even though the network was robust on given examples. For instance, when verifying the MLP0 network with a disturbance bound $\delta = 0.1$ (Table II), our verification approach proves that the network is robust (UNSAT) for all 48 examples. However, Marabou returns 47 UNSAT results and 1 SAT without generating a specific counterexample for this case. We have plotted the ranges of all outputs of the network (Figure 5) to intuitively justify the robustness in this case. The figure clearly shows that the network is robust as the output corresponding to digit 0 is still the maximum output range among others. Therefore, digit 0 is still classified correctly in this case. We have contacted Marabou's authors to query this mismatch, as there is no counterexample generated from Marabou for this case. However, we could not find out the reason when this paper was submitted. We also have experienced an unexpected behavior of Marabou when verifying MLP0 with large disturbance bound $\delta \geq 5$. Marabou reaches a timeout and returns all SAT results, which is very unexpected

Network	δ	Times	Marabou					Exact-Star				Approx-Star				
			#Sol			Time(s)		#Sol		Time(s)		#Sol				
			UN	S	UK	UN	S	UN	S	UN	S	UN	S	UK		
MLP1	0.1	TO	0	0	50	2.6	-	50	0	1.6	-	50	0	0		
	0.15	TO	0	0	50	2.6	2.78	49	1	1.59	1.56	49	1	0		
	0.2	TO	0	0	50	2.68	-	50	0	1.58	-	50	0	0		
	0.3	TO	0	0	50	2.52	2.58	46	4	1.6	1.6	46	3	1		
	0.5	TO	0	0	50	2.51	2.46	49	1	1.6	1.6	49	0	1		
	1	TO	0	0	50	2.55	2.57	49	1	1.56	1.57	49	1	0		
	3	TO	0	0	50	2.8	3.03	48	2	1.66	1.66	48	2	0		
	5	TO	0	0	50	3.25	3.14	46	4	1.68	1.7	46	4	0		
	10	TO	0	0	50	3.07	3.53	35	15	1.68	1.66	35	15	0		
	15	TO	0	0	50	2.77	2.8	31	19	1.67	1.67	31	19	0		
MLP2	0.1	TO	0	0	50	3.46	-	50	0	1.98	-	50	0	0		
	0.15	TO	0	0	50	3.39	-	50	0	1.93	-	50	0	0		
	0.2	TO	0	0	50	3.51	-	50	0	2.01	-	50	0	0		
	0.3	TO	0	0	50	3.46	3.3	49	1	2.09	2.2	49	0	1		
	0.5	TO	0	0	50	3.44	4.89	49	1	2.1	2.4	49	0	1		
	1	TO	0	0	50	4.3	5.38	49	1	2.3	2.4	49	0	1		
	3	TO	0	0	50	4.19	3.48	48	2	2.15	2.05	48	2	0		
	5	TO	0	0	50	3.4	3.54	46	4	2.13	2.13	46	4	0		
	10	TO	0	0	50	2.94	2.93	39	11	2.16	2.15	39	11	0		
	15	TO	0	0	50	3.03	3.02	32	18	2.2	2.2	32	17	0		
MLP3	0.1	TO	0	0	50	3.51	-	50	0	3.9	-	50	0	0		
	0.15	TO	0	0	50	3.42	-	50	0	3.8	-	50	0	0		
	0.2	TO	0	0	50	3.45	4.48	48	2	3.5	3.3	48	2	0		
	0.3	TO	0	0	50	3.44	3.27	49	1	3.5	3.2	49	1	0		
	0.5	TO	0	0	50	3.6	3.51	47	3	3.6	3.3	47	1	2		
	1	TO	0	0	50	4.36	4.71	48	2	3.8	3.5	48	1	1		
	3	TO	0	0	50	4.09	6.03	48	2	3.8	3.5	48	2	0		
	5	TO	0	0	50	3.5	3.04	47	3	3.6	3.6	47	3	0		
	10	TO	0	0	50	3.03	3.02	38	12	3.7	3.9	38	12	0		
	15	TO	0	0	50	3.03	3.14	26	24	3.6	3.8	26	24	0		
MLP4	0.1	TO	0	0	50	4.21	-	50	0	3.3	-	50	0	0		
	0.15	TO	0	0	50	4.23	4.08	49	1	3.2	3.3	49	0	1		
	0.2	TO	0	0	50	4.17	4.1	49	1	3.2	3.3	49	0	1		
	0.3	TO	0	0	50	4.13	4.52	47	3	3.1	3.2	47	1	2		
	0.5	TO	0	0	50	5.09	-	50	0	3.2	-	50	0	0		
	1	TO	0	0	50	4.9	5.06	48	2	3.6	3.6	48	0	2		
	3	TO	0	0	50	3.89	3.64	49	1	3.6	3.6	49	1	0		
	5	TO	0	0	50	3.63	3.59	46	4	3.6	3.7	46	3	1		
	10	TO	0	0	50	3.43	3.26	42	8	3.5	3.7	42	7	1		
	15	TO	0	0	50	3.15	3.14	35	15	1.16	1.17	35	15	0		

TABLE III: Verification results for MLP1-4. Notations are the same with that of Table II

..

δ	Marabou					Approx-Star				
	Time(s)		# Sol			Time(s)		# Sol		
	UN	S	UN	S	UK	UN	S	UN	S	UK
0.05	4.96	909.1	23	15	12	0.4	0.4	39	7	4
0.1	12.15	1,627.6	20	15	15	0.4	0.5	36	9	5
0.15	5	1,113.3	29	9	12	0.4	0.4	35	11	4
0.2	4.96	1,387.7	24	10	16	0.5	0.4	34	8	8
0.25	4.91	1,426	22	9	19	0.5	0.4	29	14	7
0.3	26.75	1,550.9	20	7	23	0.6	0.4	30	13	7

TABLE IV: Verification results of XNOR0 network. *Notation*: δ represents disturbance bound values, Time(s) represents average computation time for each value of δ, '#Sol' is the number of solved examples, UN = UNSAT indicates that the network is robust (safe) under the attack, S = SAT indicates that the network is not robust (unsafe), UK = UNKNOWN indicates that the example wasn't solved in the given time.

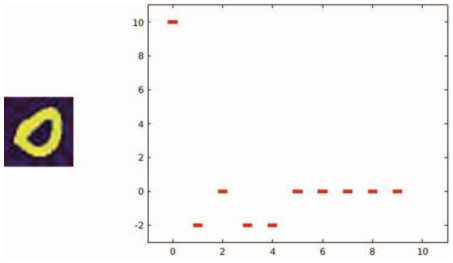


Fig. 5: MLP0 verification mismatch for $\delta = 0.1$. On this example, the network is correctly verified by Star (UNSAT) and is falsely verified by Marabou (SAT). The right plot presents the ranges of the output Star (lower and upper bounds).

behavior. For example, for $\delta = 10$, Marabou reaches timeout and returns 40 SAT results. However, in this case, we prove that 18 cases are UNSAT, and 22 cases are SAT.

B. Star Reachability vs. EEVBNN SAT-based Method

Experiment Set Up. The SAT-based method implemented in EEVBNN [9] presents results for four binary convolutional neural networks. They include two networks trained

using MNIST (**mnist-small** - 2 convolutional binary blocks and 2 feedforward binary blocks, **mnist-large** - 4 convolutional binary blocks and 3 feedforward binary blocks) and two networks trained using CIFAR10 [22] (**cifar10-small** and **cifar10-large** with similar architectures respectively). CIFAR10 consists of 60,000 32×32 color images in 10 classes, with 6,000 images per class.

The comparative experiments were conducted on 500 MNIST images for the MNIST-trained networks using the L_∞ norm attack for 2 disturbance values: $\delta = \{0.1, 0.3\}$, and on 500 CIFAR10 images for the CIFAR10-trained networks using the L_∞ norm attack for 2 disturbance values: $\delta = \{2/255, 8/255\}$. The details of the networks' architecture are presented in Table V.

Network	Architecture
mnist-small, cifar10-small	2CB:2FB
mnist-large, cifar10-small	4CB:3FB

TABLE V: The architectures of EEVBNN [9] networks. "CB" is a convolutional binary block. "FB" is a feedfoward binary block. Note that the given BCNNs do not include any pooling layers.

Verification Results. Note that on the given examples, the Star-based exact verification approach runs out of time and memory. For this reason, we only present the comparison with the overapproximate analysis algorithm. The verification results of the EEVBNN method's proposed benchmarks are presented in Table VI. Compared to EEVBNN, Star underperforms both with regard to the timing and the number of solved examples.

Timing Performance. The experiments show that EEVBNN can be from $4\times$ to $30\times$ faster than Star, depending on a model's size and the used disturbance value. For example, EEVBNN verifies all 500 examples for **cifar10-small** $6\times$ faster than Star. This happens because Star reachability is aimed at handling continuous input while EEVBNN works with the quantized one. Star's ability to operate in continuous space introduces a trade-off as its computational operations are more complex. In addition, EEVBNN tests its approach on 'solver-friendly' networks that contain high-sparsity weights.

Conservativeness. According to the experiments, EEVBNN solves all the examples, while Star is only able to solve $\approx 14\%$ for the MNIST-trained models and $\approx 88\%$ for CIFAR10-trained models. This indicates that EEVBNN is less conservative compared to the Star-based reachability method.

Drawbacks of the Proposed Approach. Although our approach is efficient and scalable for BFFNNs, it is not scalable for BCNNs with max-pooling layers. As analyzed in [16], when dealing with large disturbance bounds, more predicate variables and their associated generators are introduced in the reachability of a max-pooling layer. This causes an explosion in memory and computation time. It is worth emphasizing that BCNNs using average pooling can achieve the same (or even better) accuracy and are amenable to our verification approach [23]. We have tried analyzing BCNNs with average pooling using our approach. However, we could not compare with Marabou on these networks as Marabou does not currently support average pooling.

In addition, the given representation of Star cannot be efficiently used with quantized input space. For this reason, the method implemented in EEVBNN outperforms Star in terms of timing and conservativeness. However, we emphasize that Star reachability algorithms have been designed to work with continuous input space. While it requires the operations to be more computationally expensive, it allows Star to generalize better as it deals with continuous (infinite but bounded) input space instead of quantized input space with finite states like EEVBNN. In addition, quantization introduces various sources of errors (rounding, computational noise, etc.). All of this may not have an effect when verifying "basic" benchmarks like MNIST or CIFAR10 but could have a huge impact in real-world tasks. Note that EEVBNN also uses "solver-friendly BNNs". These BNNs' weights sparsity is artificially increased during the training process. Such an approach may also increase error accumulation. Thus, we believe that Star reachability is a good complementary approach when input quantization is not an option.

VI. Related Work

DNN verification. DNN verification approaches can be categorized into four main classes: SAT/SMT-based method [24], [25], optimization [26], [27], reachability analysis [14], [16], and abstraction [28], [29]. Reluplex [24] and Planet [30] are pioneers in the SMT-based approach. Based on Reluplex, the Marabou framework is developed to exploit the power of parallel computing in verification [25]. The SMT-based method has been extended to handle different types of neural networks recently, e.g., BNNs [12] considered in this paper. DNN verification can be converted into optimization problems. A notable technique is the mixed-integer linear programming (MILP) encoding for verifying ReLU networks [26], [27], [31]. An example of such conversion is presented in the EEVBNN framework [9]. It converts the verification problem into an SAT problem using the quantized input to boost the performance. DNN verification can also be transformed into a reachability analysis problem. Different abstract interpretation approaches have been proposed to construct the reachable set of a neural network and use it for verification. Typical representatives are polytope [32], [33], zonotope [29], relaxed-polytope [34]–[36], interval [37], [38], star [14], and imagestar [16]. Our approach proposed in this paper is an extension of the star set reachability [14] in combination with the ImageStar method [16] to verify BNNs.

BNN verification. BNNs are a sub-class of DNNs that has received some attention in the verification community recently. Recent works on BNN verification include both qualitative and quantitative analysis approaches. Quantitative verification focuses on how often the network's output satisfies or violates a pre-defined safety property. In contrast, the qualitative analysis

Network	δ	EEVBNN Time(s)		EEVBNN #Sol		Approx-Star Time(s)			Approx-Star #Sol		
		UN	S	UN	S	UN	S	UK	UN	S	UK
mnist-small	0.1	0.021	0.022	436	64	0.09	0.079	0.078	42	19	439
mnist-large	0.1	0.164	0.177	454	46	1.191	0.881	0.97	42	10	448
mnist-small	0.3	0.027	0.029	312	188	0.103	0.088	0.09	42	31	427
mnist-large	0.3	0.169	0.171	379	121	1.915	1.634	1.55	42	21	437
cifar10-small	2/255	0.046	0.047	128	372	0.273	0.267	0.27	57	228	215
cifar10-large	2/255	0.304	0.341	147	353	3.032	3.04	3.049	57	226	217
cifar10-small	8/255	0.064	0.068	94	406	1.131	1.152	1.174	57	270	173
cifar10-large	8/255	0.342	0.372	123	377	9.589	9.721	9.665	57	272	171

TABLE VI: Verification results for the networks presented by EEVBNN [9].

focuses on whether there exists an input that violates a predefined safety property which is the main focus of this paper. They include the SAT/SMT-based approaches that verify the two-valued BNNs that use a binary quantifying model {-1, 1} [13], [39]. We do not provide the comparison with the given approaches as we have not been able to obtain respective NN models for evaluation. Moreover, there are approaches that use quantized DNNs with multiple bits [40], [41].

Another type of quantitative verification approaches utilizes binary decision diagram (BDD) construction from the given BNNs [8], [42]. BDD4BNN [8] is one of the latest novel approaches that use quantization and BDDs to certify the network's robustness. BDD4BNN encodes the input using cardinality constraints into a binary representation {0, 1} and further quantizes it into {-1, 1}. In addition, all neural networks' weights and biases are also quantized, which means that BDD4BNN uses only strictly binarized networks. Another tool that uses quantized input space is EEVBNN [9]. EEVBNN performs exact verification by converting the given BNNs into SAT problems. Quantization converts a continuous input space into a discrete input space and makes neural network verification much more efficient. However, input quantization is an extra man-made step that can cause a decrease in the accuracy performance of neural networks after training, especially for those used for control purposes, because originally, the networks were trained to work on continuous input space. Unlike EEVBNN and BDD4BNN, our star-based approach aims at verifying original neural networks under continuous input space without requiring the extra input quantization step. Therefore, our evaluation focuses on comparing the star-set approach and the SMT-based approach [12] implemented in Marabou, which was also developed to deal with continuous input space. We also provide a comparison with EEVBNN, which shows the limited performance of our approach when dealing with quantized input spaces.

VII. Conclusion

In this paper, we have extended the star reachability algorithms for verifying BNNs with continuous input space. The experiments showed that the proposed method is more efficient and scalable than the SMT-based approach implemented in Marabou. Our approach underperforms compared to the quantization-based technique proposed in EEVBNN. We emphasize that the current Star representation is not designed to handle quantized input space. However, we still include the comparison with EEVBNN to give a more complete picture of our approach. In future work, we will address the star set approach's drawbacks in dealing with the max-pooling layer, which requires a new, memory-efficient data structure for reachability analysis, e.g., sparse ImageStar. We will also work on designing a representation that would allow Star-based algorithms to be more efficient when operating in the quantized space. Additionally, the overapproximation of the Sign activation function will be improved to reduce the overapproximation error. Last but not least, as our method is efficient in constructing a complete set of counterexamples in case a network is unsafe, we will leverage it to develop a new repairing method for BNNs.

Acknowledgements

We appreciate Mr. Guy Amir from Hebrew University of Jerusalem for his help on explaining the BNN verification algorithm implemented in Marabou. We also appreciate Dr. Yedi Zhang from Shanghai Tech University for her explanation of the principles and implementation of the BDD4BNN approach. We also thank Mr. Kai Jia for providing us with expertise on the models that were used for evaluating the SAT-based method implemented in EEVBNN.

References

[1] J. Watson, M. Firman, A. Monszpart, and G. J. Brostow, "Footprints and free space from a single color image," 2020.

[2] N. Babanejad, A. Agrawal, A. An, and M. Papagelis, "A comprehensive analysis of preprocessing for word representation learning in affective tasks," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 5799–5810. [Online]. Available: https://aclanthology.org/2020.acl-main.514

[3] D. Shriver, S. Elbaum, and M. B. Dwyer, "Reducing dnn properties to enable falsification with adversarial attacks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 275–287.

[4] A. Bulat, B. Martinez, and G. Tzimiropoulos, "High-capacity expert binary networks," 2021.

[5] J. Chen, X. Wu, V. Rastogi, Y. Liang, and S. Jha, "Towards understanding limitations of pixel discretization against adversarial attacks," 2019.

[6] G. Liu, I. Khalil, A. Khreishah, and N. Phan, "A synergetic attack against neural network classifiers combining backdoor and adversarial examples," 2021.

[7] N. Narodytska, H. Zhang, A. Gupta, and T. Walsh, "In search for a sat-friendly binarized neural network architecture," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJx-j64FDr

[8] Y. Zhang, Z. Zhao, G. Chen, F. Song, and T. Chen, "BDD4BNN: a BDD-based quantitative analysis framework for binarized neural networks," in *International Conference on Computer Aided Verification*. Springer, 2021, pp. 175–200.

[9] K. Jia and M. Rinard, "Efficient exact verification of binarized neural networks," 2020. [Online]. Available: https://arxiv.org/abs/2005.03597

[10] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, "Performance guaranteed network acceleration via high-order residual quantization," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[11] S. Cao, L. Ma, W. Xiao, C. Zhang, Y. Liu, L. Zhang, L. Nie, and Z. Yang, "Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[12] G. Amir, H. Wu, C. Barrett, and G. Katz, "An SMT-based approach for verifying binarized neural networks," 2021.

[13] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," 2018.

[14] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *Formal Methods – The Next 30 Years*, M. H. ter Beek, A. McIver, and J. N. Oliveira, Eds. Cham: Springer International Publishing, 2019, pp. 670–686.

[15] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson, "Improved geometric path enumeration for verifying relu neural networks," in *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.

[16] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," in *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.

[17] H.-D. Tran, X. Yang, D. Manzanas, P. Musau, L. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.

[18] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The Marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification*, I. Dillig and S. Tasiran, Eds. Cham: Springer International Publishing, 2019, pp. 443–452.

[19] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 3–17.

[20] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[21] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," 2017.

[22] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2014. [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html

[23] H.-D. Tran, N. Pal, P. Musau, X. Yang, N. P. Hamilton, D. M. Lopez, S. Bak, and T. T. Johnson, "Robustness verification of semantic segmentation neural networks using relaxed reachability," in *33rd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2021.

[24] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," 2017.

[25] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification*, I. Dillig and S. Tasiran, Eds. Cham: Springer International Publishing, 2019, pp. 443–452.

[26] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," 2017.

[27] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[28] P. Prabhakar and Z. Rahimi Afzal, "Abstraction based output range analysis for neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[29] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf

[30] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," 2017.

[31] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," 2019.

[32] W. Xiang, H. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with relu activations," *CoRR*, vol. abs/1712.08163, 2017. [Online]. Available: http://arxiv.org/abs/1712.08163

[33] H.-D. Tran, P. Musau, D. Manzanas Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Parallelizable reachability analysis algorithms for feed-forward neural networks," in *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*, 2019, pp. 51–60.

[34] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3290354

[35] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," 2018.

[36] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks," 2018.

[37] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1599–1614. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi

[38] W. Xiang, H. Tran, and T. T. Johnson, "Specification-guided safety verification for feedforward neural networks," *CoRR*, vol. abs/1812.06161, 2018. [Online]. Available: http://arxiv.org/abs/1812.06161

[39] N. Narodytska, "Formal analysis of deep binarized neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 5692–5696. [Online]. Available: https://doi.org/10.24963/ijcai.2018/811

[40] M. Baranowski, S. He, M. Lechner, T. S. Nguyen, and Z. Rakamaric, "An smt theory of fixed-point arithmetic," in *Proceedings of the 10th International Joint Conference on Automated Reasoning (IJCAR)*, ser. Lecture Notes in Computer Science, N. Peltier and V. Sofronie-Stokkermans, Eds., vol. 12166. Springer, 2020, pp. 13–31.

[41] T. A. Henzinger, M. Lechner, and ore Žikelić, "Scalable verification of quantized neural networks (technical report)," 2020.

[42] A. Shih, A. Darwiche, and A. Choi, "Verifying binarized neural networks by angluin-style learning." in *SAT*, 2019, pp. 354–370. [Online]. Available: https://doi.org/10.1007/978-3-030-24258-9_25