BDD4BNN: A BDD-Based Quantitative Analysis Framework for Binarized Neural Networks

Yedi Zhang[1], Zhe Zhao[1], Guangke Chen[1], Fu Song[1,2(✉)], and Taolue Chen[3]

[1] ShanghaiTech University, Shanghai, China
songfu@shanghaitech.edu.cn
[2] Shanghai Engineering Research Center of Intelligent Vision and Imaging, Shanghai, China
[3] Birkbeck, University of London, London, UK

Abstract. Verifying and explaining the behavior of neural networks is becoming increasingly important, especially when they are deployed in safety-critical applications. In this paper, we study verification and interpretability problems for Binarized Neural Networks (BNNs), the 1-bit quantization of general real-numbered neural networks. Our approach is to encode BNNs into Binary Decision Diagrams (BDDs), which is done by exploiting the internal structure of the BNNs. In particular, we translate the input-output relation of blocks in BNNs to cardinality constraints which are in turn encoded by BDDs. Based on the encoding, we develop a quantitative framework for BNNs where precise and comprehensive analysis of BNNs can be performed. We demonstrate the application of our framework by providing quantitative robustness analysis and interpretability for BNNs. We implement a prototype tool BDD4BNN and carry out extensive experiments, confirming the effectiveness and efficiency of our approach.

1 Introduction

Deep neural networks (DNNs) have achieved human-level performance in several tasks, and are increasingly being incorporated into various application domains such as autonomous driving [4] and medical diagnostics [53]. Modern DNNs usually contain a great many parameters which are typically stored as 32/64-bit floating-point numbers, and require a massive amount of floating-point operations to compute the output for a single input [60]. As a result, it is often challenging to deploy them on resource-constrained, embedded devices. To mitigate the issue, quantization, which quantizes 32/64-bit floating-points to low bit-width fixed-points (e.g., 4-bits) with little accuracy loss [23], emerges as a promising technique to reduce resource requirements. In particular, binarized neural networks (BNNs) [27] represent the case of 1-bit quantization using the bipolar binaries ±1. BNNs can drastically reduce memory storage and execution time with bit-wise operations, hence substantially improve the time and energy efficiency. BNNs have been demonstrated to achieve a high accuracy for a wide variety of applications [34,41,52].

This work is supported by the National Natural Science Foundation of China (NSFC) under Grants No.: 62072309, and an oversea grant from the State Key Laboratory of Novel Software Technology, Nanjing University (KFKT2018A16).

A. Silva and K. R. M. Leino (Eds.) CAV 2021, LNCS 12759, pp. 175–200, 2021.
https://doi.org/10.1007/978-3-030-81685-8_8

DNNs have been shown to lack robustness [11,14,36,49,59] and interpretability of the predictions they make [25,43]. Various formal techniques and heuristics have been proposed to analyze DNNs and interpret their behaviors, most of which focus on *real-numbered* DNNs only. Verification of *quantized* DNNs has not been thoroughly explored so far, although recent results have highlighted its importance: it was shown that a quantized DNN does not necessarily preserve the properties satisfied by the real-numbered DNN before quantization [14,22]. Indeed, the fixed-point number semantics effectively yields a discrete state space for the verification of quantized DNNs whereas real-numbered DNNs feature a continuous state space. The discrepancy could invalidate current verification techniques for real-numbered DNNs when they are directly applied to the quantized counterparts (e.g., both false negative and false positive could occur). Therefore, specialized techniques are required for rigorously verifying quantized DNNs.

Broadly speaking, the existing techniques for quantized DNNs make use of constraint solving which is based on either SAT/SMT or (reduced, ordered) binary decision diagrams (BDDs). A majority of work resorts to SAT/SMT solving. For the 1-bit quantization (i.e., BNNs), typically BNNs are transformed into Boolean formulas where SAT solving is harnessed [12,33,45,46]. Some recent work also studies variants of BNNs [28,48], i.e., BNNs with ternary weights. For quantized DNNs with multiple bits (i.e., fixed-points), it is natural to encode them as quantifier-free SMT formulas, e.g., using bit-vector and fixed-point theories [7,22,24], so that off-the-shelf SMT solvers can be leveraged. In another direction, BDD-based approaches currently can tackle BNNs only [54]. In a nutshell, they encode a BNN and an input region as a BDD, based on which various analyses can be performed via queries on the BDD. The crux of the approach is how to generate the BDD efficiently. In the work [54], the BDD is constructed by BDD learning [44], thus, currently limited to toy BNNs (e.g., 64 input size, 5 hidden neurons, and 2 output size) with relatively small input regions.

On the other hand, existing work mostly focuses on *qualitative* verification, which asks whether there exists an input x (in a specified region) for a neural network such that a property (e.g., local robustness) is violated. In many practical applications, checking only the existence is not sufficient. Indeed, for local robustness, such an (adversarial) input almost surely exists which makes a qualitative answer less meaningful. Instead, *quantitative* verification, which asks how often a property ϕ is satisfied or violated, is far more useful yet more challenging as it could provide a probabilistic guarantee of the behavior of neural networks. Such a quantitative guarantee is essential to certify, for instance, certain implementations of neural network based perceptual components against safety standards of autonomous vehicles [29,32]. Quantitative analysis of general neural networks, however, is challenging, hence received little attention and for which the results are rather limited so far. DeepSRGR [69] presented an abstract interpretation based quantitative robustness verification approach for DNNs which is sound but incomplete. For BNNs, approximate SAT model-counting solvers (♯SAT) are leveraged [6,47] based on the SAT encoding for the qualitative counterpart. Though probably approximately correct (PAC) style guarantees can be provided, verification cost is usually prohibitively high to achieve higher precision and confidence.

Main Contributions. We propose a BDD-based framework BDD4BNN to support quantitative analysis of BNNs. The main challenge is how to efficiently build BDDs from BNNs [47]. In contrast to previous work [54] which is learning-based and largely treats the BNN as a blackbox, we *directly* encode a BNN and the associated input region into BDDs. In a nutshell, a BNN is a sequential composition of multiple internal blocks and one output block. Each block comprises 3 layers and captures a function $f : \{+1, -1\}^n \to \{+1, -1\}^m$, where n (resp. m) denotes the number of inputs (resp. outputs) of the block. Technically, the function f can be alternatively rewritten as a function over the standard Boolean domain, i.e., $f : \{0, 1\}^n \to \{0, 1\}^m$. A key stepping-stone of our encoding is the observation that the i-th output y_i of the block can be captured by a cardinality constraint of the form $\sum_{j=1}^{n} \ell_j \geq k$ such that $y_i = +1 \Leftrightarrow \sum_{j=1}^{n} \ell_j \geq k$, where each literal ℓ_j is either x_j or $\neg x_j$ for the input variable x_j, and k is a constant. We then present an algorithm to encode a cardinality constraint $\sum_{j=1}^{n} \ell_j \geq k$ as a BDD with $O((n - k) \cdot k)$ nodes in $O((n - k) \cdot k)$ time. As a result, the input-output relation of each block can be encoded as a BDD, the composition of which yields the BDD for the entire BNN. A distinguished advantage of our BDD encoding lies in its support of incremental encoding. In particular, when different input regions are of interest, there is no need to construct the BDD of the entire BNN from scratch.

Encoding BNNs as BDDs enables a wide variety of applications in security analysis and decision explanation of BNNs. In this paper, we highlight two of them within our framework, i.e., robustness analysis and interpretability. It was shown that DNNs have been suffering from poor robustness to adversarial examples [49,50,59]. We consider two quantitative variants of the problem: (1) how many adversarial examples does the BNN have in the input region, and (2) how many of them are misclassified to each class? We further provide an algorithm to incrementally compute the (locally) maximal Hamming distance within which the BNN satisfies the desired robustness properties.

Interpretability is an issue arisen as a result of the blackbox nature of DNNs [25,43]. In application domains such as medical diagnosis, understanding the decisions made by DNNs is a must. We consider two problems: (1) why some inputs are (mis)classified into a class by the BNN and (2) are there any essential features in the input region that are common for all samples classified into a class?

Experimental Results. We implement our framework as a prototype tool BDD4BNN using the CUDD package [58], which scales to BNNs with up to 4 internal blocks, 200 hidden neurons, and 784 input size. To the best of our knowledge, it is the first work to precisely and quantitatively analyze such large BNNs that go significantly beyond the state-of-the-art. The experimental results show that BDD4BNN is significantly more efficient and scalable than the learning-based technique [54]. Furthermore, we demonstrate how BDD4BNN can be used in quantitative robustness analysis and decision explanation of BNNs. For quantitative robustness analysis, our experimental results show that BDD4BNN is considerably (5× to 1,340×) faster and more accurate than the state-of-the-art approximate ♯SAT-based approach [6]. It can also compute precisely the distribution of predicated classes of the images in the input region as well as the locally maximal Hamming distances on several BNNs. For decision explanation, we show the effectiveness of BDD4BNN in computing prime-implicant explanations and essential features of the given input region for some target classes. Note that this work focuses on quantitative verification and interpretability of BNNs and may underperform SAT/SMT-based methods [12,33,45,46] for qualitative verification of BNNs.

In general, our main contributions can be summarized as follows.

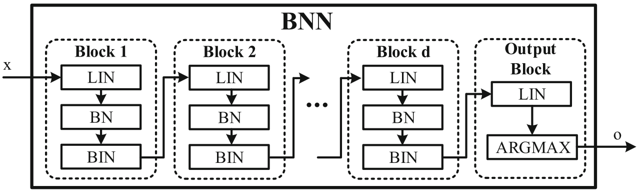


Fig. 1. Architecture of a BNN with $d + 1$ blocks

- We introduce a novel algorithmic approach for encoding BNNs into BDDs that exactly preserves the semantics of BNNs and supports incremental encoding.
- We propose a framework for quantitative verification of BNNs and in particular, we demonstrate the robustness analysis and interpretability of BNNs.
- We implement the framework as an end-to-end tool BDD4BNN and conduct thorough experiments on various BNNs, demonstrating the efficiency and effectiveness of BDD4BNN.

2 Preliminaries

In this section, we briefly introduce binarized neural networks (BNNs) and (reduced, ordered) binary decision diagrams (BDDs).

We denote by $\mathbb{R}$, $\mathbb{N}$, $\mathbb{B}$, and $\mathbb{B}_{\pm 1}$ the set of real numbers, the set of natural numbers, the standard Boolean domain $\{0, 1\}$ and the integer set $\{+1, -1\}$. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \cdots, n\}$. We will use $\boldsymbol{W}$, $\boldsymbol{W}', \ldots$ to denote (2-dimensional) matrices, $\boldsymbol{x}, \boldsymbol{v}, \cdots$ to denote (row) vectors, and $x, v, \ldots$ to denote scalars. We denote by $\boldsymbol{W}_{i,:}$ and $\boldsymbol{W}_{:,j}$ the i-th row and j-th column of the matrix $\boldsymbol{W}$. Similarly, we denote by $\boldsymbol{x}_j$ and $\boldsymbol{W}_{i,j}$ the j-th entry of $\boldsymbol{x}$ and $\boldsymbol{W}_{i,:}$ respectively. In this work, Boolean values $1/0$ will be used as integers $1/0$ in arithmetic computations without typecasting.

2.1 Binarized Neural Networks

A binarized neural network (BNN) [27] is a neural network where weights and activations are predominantly binarized over the domain $\mathbb{B}_{\pm 1}$. In this work, we consider feed-forward BNNs. As shown in Fig. 1, a BNN can be seen as a sequential composition of several internal blocks and one output block. Each internal block comprises 3 layers: a linear layer (LIN), a batch normalization layer (BN), and a binarization layer (BIN). The output block comprises a linear layer and an ARGMAX layer. Note that the input/output of internal blocks and the input of the output block are all vectors over $\mathbb{B}_{\pm 1}$.

Table 1. Definitions of layers in BNNs, where $n_{d+2} = s$ and $\arg\max(\cdot)$ returns the index of the largest entry which occurs first.

Layer	Function	Parameters	Definition
LIN	$t_i^{lin} : \mathbb{B}_{\pm 1}^{n_i} \to \mathbb{R}^{n_{i+1}}$	Weight matrix: $\boldsymbol{W} \in \mathbb{B}_{\pm 1}^{n_i \times n_{i+1}}$ Bias (row) vector: $\boldsymbol{b} \in \mathbb{R}^{n_{i+1}}$	$t_i^{lin}(\boldsymbol{x}) = \boldsymbol{y}$ where $\forall j \in [n_{i+1}]$, $\boldsymbol{y}_j = \langle \boldsymbol{x}, \boldsymbol{W}_{:,j} \rangle + \boldsymbol{b}_j$
BN	$t_i^{bn} : \mathbb{R}^{n_{i+1}} \to \mathbb{R}^{n_{i+1}}$	Weight vectors: $\alpha \in \mathbb{R}^{n_{i+1}}$ Bias vector: $\gamma \in \mathbb{R}^{n_{i+1}}$ Mean vector: $\mu \in \mathbb{R}^{n_{i+1}}$ Std. dev. vector: $\sigma \in \mathbb{R}^{n_{i+1}}$	$t_i^{bn}(\boldsymbol{x}) = \boldsymbol{y}$ where $\forall j \in [n_{i+1}]$, $\boldsymbol{y}_j = \alpha_j \cdot (\frac{x_j - \mu_j}{\sigma_j}) + \gamma_j$
BIN	$t_i^{bin} : \mathbb{R}^{n_{i+1}} \to \mathbb{B}_{\pm 1}^{n_{i+1}}$	–	$t_i^{bin}(\boldsymbol{x}) = \boldsymbol{y}$ where $\forall j \in [n_{i+1}]$, $\boldsymbol{y}_j = \begin{cases} +1, & \text{if } \boldsymbol{x}_j \geq 0; \\ -1, & \text{otherwise.} \end{cases}$
ARGMAX	$t_{d+1}^{am} : \mathbb{R}^s \to \mathbb{B}^s$	–	$t_{d+1}^{am}(\boldsymbol{x}) = \boldsymbol{y}$ where $\forall j \in [s]$, $\boldsymbol{y}_j = 1 \Leftrightarrow j = \arg\max(\boldsymbol{x})$

Definition 1. *A BNN $\mathcal{N} : \mathbb{B}_{\pm 1}^{n_1} \to \mathbb{B}^s$ with s classes is given by a tuple of blocks $(t_1, \cdots, t_d, t_{d+1})$ such that $\mathcal{N} = t_{d+1} \circ t_d \circ \cdots \circ t_1$,*

- *for every $i \in [d]$, $t_i : \mathbb{B}_{\pm 1}^{n_i} \to \mathbb{B}_{\pm 1}^{n_{i+1}}$ is an internal block comprising a LIN layer t_i^{lin}, a BN layer t_i^{bn} and a BIN t_i^{bin} with $t_i = t_i^{bin} \circ t_i^{bn} \circ t_i^{lin}$,*
- *$t_{d+1} : \mathbb{B}_{\pm 1}^{n_{d+1}} \to \mathbb{B}^s$ is the output block comprising a LIN layer t_{d+1}^{lin} and an ARGMAX layer t_{d+1}^{am} with $t_{d+1} = t_{d+1}^{am} \circ t_{d+1}^{lin}$,*

where t_i^{bin}, t_i^{bn}, t_i^{lin} for $i \in [d]$, t_{d+1}^{lin} and t_{d+1}^{am} are given in Table 1.

Intuitively, a LIN layer is a linear transformation. A BN layer following a LIN layer is used to standardize and normalize the output of the LIN layer. A BIN layer is used to binarize the real-numbered output vector of the BN layer. In this work, we consider the sign function which is widely used in BNNs to binarize real-numbered vectors. An ARGMAX layer follows a LIN layer and outputs the index of the largest entry as the predicted class which is represented by a one-hot vector. (In case there is more than one such entry, the first one is returned.) Formally, given a BNN $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$ and an input $\boldsymbol{x} \in \mathbb{B}_{\pm 1}^{n_1}$, $\mathcal{N}(\boldsymbol{x}) \in \mathbb{B}^s$ is a one-hot vector in which the index of the non-zero entry is the predicated class.

2.2 Binary Decision Diagrams

A BDD [9] is a rooted acyclic directed graph where non-terminal nodes v are labeled by Boolean variables $\mathsf{var}(v)$ and terminal nodes (leaves) v are labeled with values $\mathsf{val}(v) \in \mathbb{B}$, referred to as the 1-leaf and the 0-leaf respectively. Each non-terminal node v has two outgoing edges: $\mathsf{hi}(v)$ meaning $\mathsf{var}(v) = 1$ and $\mathsf{lo}(v)$ meaning $\mathsf{var}(v) = 0$. We will also refer to $\mathsf{hi}(v)$ and $\mathsf{lo}(v)$ as the hi and lo children of v respectively. Moreover, assuming that $x_1, \cdots, x_m$ is the variable ordering, for each node v with $\mathsf{var}(v) = x_i$ and each $v' \in \{\mathsf{hi}(v), \mathsf{lo}(v)\}$ with $\mathsf{var}(v') = x_j$, we have $i < j$. In the graphical representation of BDDs, $\mathsf{hi}(v)$ and $\mathsf{lo}(v)$ are depicted by solid and dashed lines respectively. Multi-Terminal Binary Decision Diagrams (MTBDDs) are a variant of BDDs in which the

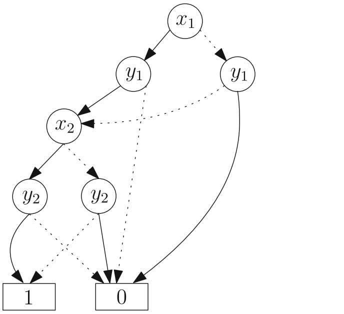


Fig. 2. The reduced BDD for $f(x_1, y_1, x_2, y_2) = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$

Table 2. Some basic BDD operations, where $op \in \{\textsc{And}, \textsc{Or}, \textsc{Xor}, \textsc{Xnor}\}$

Operation	Description
$v = \textsc{Var}(x)$	$f_v(x) = x$
$v = \textsc{Const}(1)$	$f_v = 1$
$v = \textsc{Const}(0)$	$f_v = 0$
$\textsc{Not}(v)$	$\neg f_v$
$\textsc{Apply}(v, v', op)$	$f_v \; op \; f_{v'}$
$\textsc{Exists}(v, X)$	$\exists X.f_v$
$\textsc{SatAll}(v)$	$\textsc{SatAll}(f_v)$
$\textsc{RelProd}(v, v')$	$f_v \circ f_{v'}$
$\text{ITE}(x, v, v')$	$(x \wedge v) \vee (\neg x \wedge v')$

terminal nodes are not restricted to be 0 or 1. A BDD is *reduced* if it (1) has only one 1-leaf and one 0-leaf, (2) does not contain a node v such that $\mathsf{hi}(v) = \mathsf{lo}(v)$, and (3) does not contain two distinct non-terminal nodes v and v' such that $\mathsf{var}(v) = \mathsf{var}(v')$, $\mathsf{hi}(v) = \mathsf{hi}(v')$ and $\mathsf{lo}(v) = \mathsf{lo}(v')$. For example, Fig. 2 shows the reduced BDD for the Boolean function $f(x_1, y_1, x_2, y_2) = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$. Hereafter, we assume that BDDs are reduced.

Bryant [9] showed that BDDs can serve as a canonical form of Boolean functions. Given a BDD over variables $x_1, \cdots, x_m$, each non-terminal node v with $\mathsf{var}(v) = x_i$ represents a Boolean function $f_v = (x_i \wedge f_{\mathsf{hi}(v)}) \vee (\neg x_i \wedge f_{\mathsf{lo}(v)})$. Operations on Boolean functions can usually be efficiently implemented via manipulating their BDD representations. A good variable ordering is crucial for the performance of BDD manipulations while the problem of finding an optimal ordering for a function is NP-hard. To store and manipulate BDDs efficiently, the nodes are stored in a hash table and the recent computed results are stored in a cache to avoid duplicated computations. In this work, we will use some basic BDD operations such as ITE for If-Then-Else, Xor for exclusive-OR, Xnor for exclusive-NOR (i.e., a Xnor $b = \neg(a$ Xor $b))$ and $\textsc{SatAll}(f_v)$ for the set of all solutions of the Boolean formula f_v. We denote by $\mathcal{L}(v)$ the set $\textsc{SatAll}(f_v)$. For easy reference, more operations are given in Table 2. By $op(v, v')$ we denote the operation $\textsc{Apply}(v, v', op)$.

3 BDD4BNN Design

3.1 BDD4BNN Overview

An overview of BDD4BNN is depicted in Fig. 3. BDD4BNN comprises four main components: Region2BDD, BNN2CC, BDD Model Builder, and Query Engine. For a fixed BNN $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$ and a region R of the input space of $\mathcal{N}$, BDD4BNN constructs the BDDs $(G_i^{out})_{i \in [s]}$ to encode the input-output relation of $\mathcal{N}$ in the region R, where the BDD G_i^{out} corresponds to the class $i \in [s]$. Technically, the region R is partitioned into s parts represented by $(G_i^{out})_{i \in [s]}$. For each property query, BDD4BNN analyzes $(G_i^{out})_{i \in [s]}$ and outputs the query result.

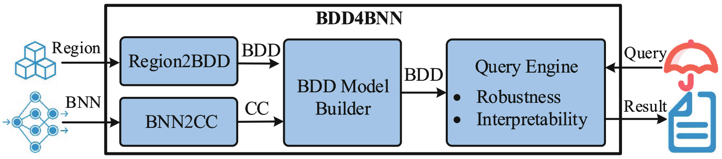
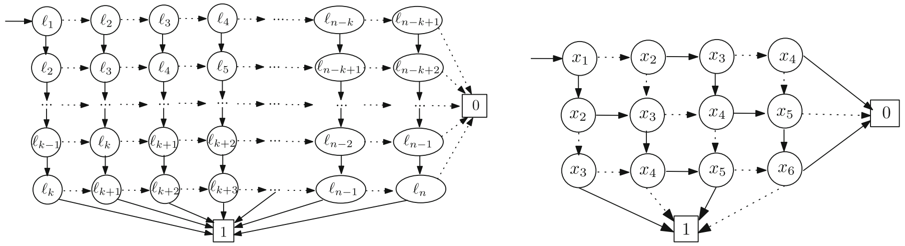


Fig. 3. Overview of BDD4BNN



(a) $\sum_{j=1}^{n} \ell_j \geq k$ (b) $x_1 + \neg x_2 + x_3 + \neg x_4 + x_5 + \neg x_6 \geq 3$

Fig. 4. Graphic representation of BDDs using Algorithm 1

The general workflow of our approach is as follows. First, Region2BDD builds up a BDD G_R^{in} from the region R which represents the desired input space of $\mathcal{N}$ for analysis. Second, BNN2CC transforms each block of the BNN $\mathcal{N}$ into a set of cardinality constraints (CCs) similar to [6,46]. Third, BDD Model Builder builds the BDDs $(G_i^{out})_{i\in[s]}$ from all the cardinality constraints and the BDD G_R^{in}. Finally, Query Engine answers queries by analyzing the BDDs $(G_i^{out})_{i\in[s]}$. Our Query Engine currently supports two types of application queries: robustness analysis and interpretability.

In the rest of this section, we first introduce the key sub-component CC2BDD, which provides an encoding of cardinality constraints into BDDs. We then provide details of the components Region2BDD, BNN2CC, and BDD Model Builder. The Query Engine will be described in Sect. 4.

3.2 CC2BDD: Cardinality Constraints to BDDs

A *cardinality constraint* is a constraint of the form $\sum_{j=1}^{n} \ell_j \geq k$ over a vector $\boldsymbol{x}$ of Boolean variables with length n, where the literal ℓ_j is either $\boldsymbol{x}_j$ or $\neg\boldsymbol{x}_j$ for each $j \in [n]$. Note that constraints of the form $\sum_{j=1}^{n} \ell_j > k$, $\sum_{j=1}^{n} \ell_j \leq k$ and $\sum_{j=1}^{n} \ell_j < k$ are equivalent to $\sum_{j=1}^{n} \ell_j \geq k + 1$, $\sum_{j=1}^{n} \neg\ell_j \geq n - k$ and $\sum_{j=1}^{n} \neg\ell_j \geq n - k + 1$, respectively. We assume that 1 (resp. 0) is a special cardinality constraint that always holds (resp. never holds).

To encode $\sum_{j=1}^{n} \ell_j \geq k$ as a BDD, we observe that all the possible solutions of $\sum_{j=1}^{n} \ell_j \geq k$ can be compactly represented by a BDD-like graph shown in Fig. 4(a), where each node is labeled by a literal, and a solid (resp. dashed) edge from a node labeled by ℓ_j means that the value of the literal ℓ_j is 1 (resp. 0). Thus, each path from the ℓ_1-node to the 1-leaf through the ℓ_j-node (where $1 \leq j \leq n$) captures a set of valuations where ℓ_j followed by a (horizontal) dashed line is set to be 0 while ℓ_j followed by

Algorithm 1: BDD Construction for cardinality constraints

```
1 Proc CC2BDD($\mathsf{CC} : \sum_{j=1}^{n} \ell_j \geq k$)
2     $G_{k+1,1} = G_{k+1,2} = \cdots = G_{k+1,n-k+1} = \textsc{Const}(1)$;
3     $G_{1,n-k+2} = G_{2,n-k+2} = \cdots = G_{k,n-k+2} = \textsc{Const}(0)$;
4     for ($i = k;\ i \geq 1;\ i--$) do
5         for ($j = n-k+1;\ j \geq 1;\ j--$) do
6             if ($\ell_{i+j-1} == \boldsymbol{x}_{i+j-1}$) then $G_{i,j} = \mathrm{ITE}(\boldsymbol{x}_{i+j-1}, G_{i+1,j}, G_{i,j+1})$;
7             else $G_{i,j} = \mathrm{ITE}(\boldsymbol{x}_{i+j-1}, G_{i,j+1}, G_{i+1,j})$;
8     return $G_{1,1}$
```

a (vertical) solid line is set to be 1, and all the other literals which are not along the path can take arbitrary values. Clearly, for each of these valuations, there are at least k positive literals, hence the constraint $\sum_{j=1}^{n} \ell_j \geq k$ holds.

Based on the above observation, we build the BDD for $\sum_{j=1}^{n} \ell_j \geq k$ using Algorithm 1. It builds a BDD for each node in Fig. 4(a), row-by-row (the index i in Algorithm 1) and from right to left (the index j in Algorithm 1). For each node at the i-th row and j-th column, the label of the node must be the literal ℓ_{i+j-1}. We build the BDD $G_{i,j} = \mathrm{ITE}(\boldsymbol{x}_{i+j-1}, G_{i+1,j}, G_{i,j+1})$ if ℓ_{i+j-1} is of the form $\boldsymbol{x}_{i+j-1}$ (Line 6), otherwise we build the BDD $G_{i,j} = \mathrm{ITE}(\boldsymbol{x}_{i+j-1}, G_{i,j+1}, G_{i+1,j})$ (Line 7). Finally, we obtain the BDD $G_{1,1}$ that encodes the solutions of $\sum_{j=1}^{n} \ell_j \geq k$. Consider $x_1 + \neg x_2 + x_3 + \neg x_4 + x_5 + \neg x_6 \geq 3$, Fig. 4(b) shows its BDD by Algorithm 1.

Lemma 1. *For each cardinality constraint $\sum_{j=1}^{n} \ell_j \geq k$, a BDD G with $O((n-k) \cdot k)$ nodes can be computed in $O((n-k) \cdot k)$ time such that $\mathcal{L}(G)$ is the set of all the solutions of $\sum_{j=1}^{n} \ell_j \geq k$.*

Compared with prior works [8,42] which transform general arithmetic constraints into BDDs, we devise a dedicated BDD encoding algorithm for the cardinality constraints without applying reduction, hence it is more efficient.

3.3 Region2BDD: Input Regions to BDDs

In this paper, we consider the following two types of input regions.

- *Input region based on Hamming distance*. For an input $\boldsymbol{u} \in \mathbb{B}_{\pm 1}^{n_1}$ and an integer $r \geq 0$, $R(\boldsymbol{u}, r)$ denotes the set $\{\boldsymbol{x} \in \mathbb{B}_{\pm 1}^{n_1} \mid \mathsf{HD}(\boldsymbol{x}, \boldsymbol{u}) \leq r\}$, where $\mathsf{HD}(\boldsymbol{x}, \boldsymbol{u})$ denotes the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{u}$. Intuitively, $R(\boldsymbol{u}, r)$ includes the input vectors which differ from $\boldsymbol{u}$ by at most r positions.
- *Input region with fixed indices*. For an input $\boldsymbol{u} \in \mathbb{B}_{\pm 1}^{n_1}$ and a set of indices $I \subseteq [n_1]$, $R(\boldsymbol{u}, I)$ denotes the set $\{\boldsymbol{x} \in \mathbb{B}_{\pm 1}^{n_1} \mid \forall i \in [n_1] \setminus I.\ \boldsymbol{u}_i = \boldsymbol{x}_i\}$. Intuitively, $R(\boldsymbol{u}, I)$ includes the input vectors which differ from $\boldsymbol{u}$ only at the indices from I.

Note that both $R(\boldsymbol{u}, n_1)$ and $R(\boldsymbol{u}, [n_1])$ denote the entire input space $\mathbb{B}_{\pm 1}^{n_1}$.

Recall that each input sample is an element from $\mathbb{B}_{\pm 1}^{n_1}$. To represent the region R by a BDD, we transform each value ±1 into a Boolean value 1/0. To this end, for each input $\boldsymbol{u} \in \mathbb{B}_{\pm 1}^{n_1}$, we create a new sample $\boldsymbol{u}^{(b)} \in \mathbb{B}^{n_1}$ such that for every $i \in [n_1]$, $\boldsymbol{u}_i =$

$2\boldsymbol{u}_i^{(b)} - 1$. Therefore, $R(\boldsymbol{u}, r)$ and $R(\boldsymbol{u}, I)$ will be represented by $R(\boldsymbol{u}^{(b)}, r)$ and $R(\boldsymbol{u}^{(b)}, I)$, respectively. The transformation functions t_i^{lin}, t_i^{bn}, t_i^{bin} and t_{d+1}^{am} of the LIN, BN, BIN, and ARGMAX layers (cf. Table 1) will be handled accordingly. Note that for convenience, vectors over the Boolean domain $\mathbb{B}$ may be directly given by $\boldsymbol{u}$ or $\boldsymbol{x}$ when it is clear from the context.

Region Encoding Under Hamming Distance. Given an input $\boldsymbol{u} \in \mathbb{B}^{n_1}$ and an integer r, the region $R(\boldsymbol{u}, r)$ can be expressed by a cardinality constraint $\sum_{j=1}^{n_1} \ell_j \leq r$ (which is equivalent to $\sum_{j=1}^{n_1} \neg\ell_j \geq n_1 - r$), where for every $j \in [n_1]$, $\ell_j = \boldsymbol{x}_j$ if $\boldsymbol{u}_j = 0$, otherwise $\ell_j = \neg\boldsymbol{x}_j$. For instance, consider $\boldsymbol{u} = (1, 1, 1, 0, 0)$ and $r = 2$, we have:

$$\mathsf{HD}(\boldsymbol{u}, \boldsymbol{x}) = 1 \oplus \boldsymbol{x}_1 + 1 \oplus \boldsymbol{x}_2 + 1 \oplus \boldsymbol{x}_3 + 0 \oplus \boldsymbol{x}_4 + 0 \oplus \boldsymbol{x}_5 = \neg\boldsymbol{x}_1 + \neg\boldsymbol{x}_2 + \neg\boldsymbol{x}_3 + \boldsymbol{x}_4 + \boldsymbol{x}_5.$$

Thus, $R((1, 1, 1, 0, 0), 2)$ can be expressed by the cardinality constraint $\neg\boldsymbol{x}_1 + \neg\boldsymbol{x}_2 + \neg\boldsymbol{x}_3 + \boldsymbol{x}_4 + \boldsymbol{x}_5 \leq 2$, or equivalently $\boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3 + \neg\boldsymbol{x}_4 + \neg\boldsymbol{x}_5 \geq 3$.

By Algorithm 1, the cardinality constraint of $R(\boldsymbol{u}, r)$ can be encoded by the BDD $G_{\boldsymbol{u},r}^{in}$, such that $\mathcal{L}(G_{\boldsymbol{u},r}^{in}) = R(\boldsymbol{u}, r)$. Following Lemma 1, we get that:

Lemma 2. *For an input region R given by an input $\boldsymbol{u} \in \mathbb{B}^{n_1}$ and an integer r, a BDD $G_{\boldsymbol{u},r}^{in}$ with $O(r \cdot (n_1 - r))$ nodes can be computed in $O(r \cdot (n_1 - r))$ time such that $\mathcal{L}(G_{\boldsymbol{u},r}^{in}) = R(\boldsymbol{u}, r)$.*

Region Encoding Under Fixed Indices. Given an input $\boldsymbol{u} \in \mathbb{B}^{n_1}$ and a set of indices $I \subseteq [n_1]$, the region $R(\boldsymbol{u}, I) = \{\boldsymbol{x} \in \mathbb{B}^{n_1} \mid \forall i \in [n_1] \setminus I. \, \boldsymbol{u}_i = \boldsymbol{x}_i\}$ can be represented by the BDD $G_{\boldsymbol{u},I}^{in} \triangleq \textsc{And}_{i \in [n_1] \setminus I}\big((\boldsymbol{u}_i == 1) ? \textsc{Var}(\boldsymbol{x}_i) : \textsc{Not}(\textsc{Var}(\boldsymbol{x}_i))\big)$. Intuitively, $G_{\boldsymbol{u},I}^{in}$ states that the value at the index $i \in [n_1] \setminus I$ should be the same as the one in $\boldsymbol{u}$ while the value at the index $i \in I$ is unrestricted. For instance, consider $\boldsymbol{u} = (1, 0, 0, 0)$ and $I = \{3, 4\}$, we have:

$$R((1, 0, 0, 0), \{3, 4\}) = \{(1, 0, 0, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 0, 1, 1)\} = \boldsymbol{x}_1 \wedge \neg\boldsymbol{x}_2.$$

Lemma 3. *For an input region R given by an input $\boldsymbol{u} \in \mathbb{B}^{n_1}$ and indices $I \subseteq [n_1]$, a BDD $G_{\boldsymbol{u},I}^{in}$ with $O(n_1 - |I|)$ nodes can be computed in $O(n_1)$ time such that $\mathcal{L}(G_{\boldsymbol{u},I}^{in}) = R(\boldsymbol{u}, I)$.*

3.4 BNN2CC: BNNs to Cardinality Constraints

As mentioned before, to encode the BNN $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$ as BDDs, we transform the BNN $\mathcal{N}$ into cardinality constraints from which the desired BDDs $(G_i^{out})_{i \in [s]}$ are constructed. To this end, we first transform each internal block $t_i : \mathbb{B}_{\pm 1}^{n_i} \to \mathbb{B}_{\pm 1}^{n_{i+1}}$ into n_{i+1} cardinality constraints, each of which corresponds to one of the outputs of t_i. Then we transform the output block $t_{d+1} : \mathbb{B}_{\pm 1}^{n_{d+1}} \to \mathbb{B}^s$ into $s(s - 1)$ cardinality constraints, where one output class yields $(s - 1)$ cardinality constraints.

For each vector-valued function t, we denote by $t_{\downarrow j}$ the (scalar-valued) function returning the j-th entry of the output of t.

Transformation for Internal Blocks. Consider the internal block $t_i : \mathbb{B}_{\pm1}^{n_i} \to \mathbb{B}_{\pm1}^{n_{i+1}}$ for $i \in [d]$. Recall that for every $j \in [n_{i+1}]$ and $\boldsymbol{x} \in \mathbb{B}_{\pm1}^{n_i}$, $t_{i\downarrow j}(\boldsymbol{x}) = t_i^{bin}(t_i^{bn}(\langle \boldsymbol{x}, \boldsymbol{W}_{:,j}\rangle + \boldsymbol{b}_j))$, and each value ±1 of an input $\boldsymbol{u} \in \mathbb{B}_{\pm1}^{n_1}$ is replaced by $1/0$ (cf. Sect. 3.3). To be consistent, the function $t_{i\downarrow j} : \mathbb{B}_{\pm1}^{n_i} \to \mathbb{B}_{\pm1}$ is reformulated as the function $t_{i\downarrow j}^{(b)} : \mathbb{B}^{n_i} \to \mathbb{B}$ such that for every $\boldsymbol{x} \in \mathbb{B}^{n_i}$, $t_{i\downarrow j}^{(b)}(\boldsymbol{x}) = 0.5 \times (t_i^{bin}(t_i^{bn}(\langle 2\boldsymbol{x} - \mathbf{1}, \boldsymbol{W}_{:,j}\rangle + \boldsymbol{b}_j)) + 1)$, where $\mathbf{1}$ denotes the vector of 1's with the width n_i.

Let $C_{i,j}$ be the following cardinality constraint:

$$C_{i,j} \triangleq \begin{cases} \sum_{k=1}^{n_i} \ell_k \geq \lceil \frac{1}{2} \cdot (n_i + \mu_j - \boldsymbol{b}_j - \frac{\gamma_j \cdot \sigma_j}{\alpha_j})\rceil, & \text{if } \alpha_j > 0; \\ 1, & \text{if } \alpha_j = 0 \wedge \gamma_j \geq 0; \\ 0, & \text{if } \alpha_j = 0 \wedge \gamma_j < 0; \\ \sum_{k=1}^{n_i} \neg\ell_k \geq \lceil \frac{1}{2} \cdot (n_i - \mu_j + \boldsymbol{b}_j + \frac{\gamma_j \cdot \sigma_j}{\alpha_j})\rceil, & \text{if } \alpha_j < 0; \end{cases}$$

where for every $k \in [n_i]$, ℓ_k is $\boldsymbol{x}_k$ if $\boldsymbol{W}_{k,j} = +1$, and ℓ_k is $\neg\boldsymbol{x}_k$ if $\boldsymbol{W}_{k,j} = -1$.

Proposition 1. $t_{i\downarrow j}^{(b)} \Leftrightarrow C_{i,j}$.

Proof refers to [71].

Transformation for the Output Block. For the output block $t_{d+1} : \mathbb{B}_{\pm1}^{n_{d+1}} \to \mathbb{B}^s$, since $t_{d+1} = t_{d+1}^{am} \circ t_{d+1}^{lin}$, then for every $j \in [s]$, we can reformulate $t_{d+1\downarrow j} : \mathbb{B}_{\pm1}^{n_{d+1}} \to \mathbb{B}$ as the function $t_{d+1\downarrow j}^{(b)} : \mathbb{B}^{n_{d+1}} \to \mathbb{B}$ such that for every $\boldsymbol{x} \in \mathbb{B}^{n_{d+1}}$, $t_{d+1\downarrow j}^{(b)}(\boldsymbol{x}) = t_{d+1\downarrow j}(2\boldsymbol{x} - \mathbf{1})$.

For every $j' \in [s] \setminus \{j\}$, we define the cardinality constraint $C_{d+1,j'}$ as follows:

$$C_{d+1,j'} \triangleq \begin{cases} \sum_{k=1}^{n_{d+1}} \ell_{d+1,k} \geq \frac{1}{4}(\boldsymbol{b}_{j'} - \boldsymbol{b}_j + \sum_{k=1}^{n_{d+1}}(\boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'})) + 1 + \sharp\texttt{Neg}, \\ \qquad \text{if } j' < j \text{ and } \frac{1}{4}(\boldsymbol{b}_{j'} - \boldsymbol{b}_j + \sum_{k=1}^{n_{d+1}}(\boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'})) \text{ is an integer;} \\ \sum_{k=1}^{n_{d+1}} \ell_{d+1,k} \geq \lceil \frac{1}{4}(\boldsymbol{b}_{j'} - \boldsymbol{b}_j + \sum_{k=1}^{n_{d+1}}(\boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'}))\rceil + \sharp\texttt{Neg}, \quad \text{otherwise;} \end{cases}$$

where $\sharp\texttt{Neg} = |\{k \in [n_{d+1}] \mid \boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'} = -2\}|$, $\ell_{d+1,k}$ is $\boldsymbol{x}_{d+1,k}$ if $\boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'} = +2$, $\ell_{d+1,k}$ is $\neg\boldsymbol{x}_{d+1,k}$ if $\boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'} = -2$, and $\ell_{d+1,k}$ is 0 if $\boldsymbol{W}_{k,j} - \boldsymbol{W}_{k,j'} = 0$.

Proposition 2. $t_{d+1\downarrow j}^{(b)} \Leftrightarrow \bigwedge_{j' \in [s], j' \neq j} C_{d+1,j'}$.

Proof refers to [71].

For each internal block $t_i : \mathbb{B}_{\pm1}^{n_i} \to \mathbb{B}_{\pm1}^{n_{i+1}}$, we denote by $\text{BNN2CC}(t_i)$ the cardinality constraints $\{C_{i,1}, \cdots, C_{i,n_{i+1}}\}$. For each output class $j \in [s]$, we denote by $\text{BNN2CC}^j(t_{d+1})$ the cardinality constraints $\{C_{d+1,1}, \cdots C_{d+1,j-1}, C_{d+1,j+1}, \cdots, C_{d+1,s}\}$. By applying the above transformation to all the blocks of the BNN $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$, we obtain its cardinality constraint form $\mathcal{N}^{(b)} = (t_1^{(b)}, \cdots, t_d^{(b)}, t_{d+1}^{(b)})$ such that for each $i \in [d]$, $t_i^{(b)} = \text{BNN2CC}(t_i)$, and $t_{d+1}^{(b)} = (\text{BNN2CC}^1(t_{d+1}), \cdots, \text{BNN2CC}^s(t_{d+1}))$. Given an input $\boldsymbol{u} \in \mathbb{B}^{n_1}$, we denote by $\mathcal{N}^{(b)}(\boldsymbol{u})$ the index $j \in [s]$ such that all the cardinality constraints in $\text{BNN2CC}^j(t_{d+1})$ hold under the valuation $\boldsymbol{u}$. It is straightforward to verify:

Theorem 1. *$\boldsymbol{u} \in \mathbb{B}_{\pm1}^{n_1}$ is classified into the class j by the BNN $\mathcal{N}$ iff $\mathcal{N}^{(b)}(\boldsymbol{u}^{(b)}) = j$.*

Example 1. Consider the BNN $\mathcal{N} = (t_1, t_2)$ with one internal block t_1 and one output block t_2 as shown in Fig. 5 (left-bottom), where the elements of the Weight matrix $\boldsymbol{W}$ are associated to the edges, and the other parameters are given in the left-up table. The transformation functions of blocks t_1 and t_2 are given in the right-up table, and their cardinality constraints are given in the right-bottom table.

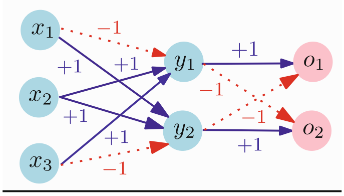
For instance, for each input $\boldsymbol{x} \in \mathbb{B}^3_{\pm 1}$, $y_1 = \text{sign}(-x_1 + x_2 + x_3 + 2.7)$, i.e., $y_1 = +1 \Leftrightarrow -x_1 + x_2 + x_3 + 2.7 \geq 0$. By replacing x_i with $2 \times x_i^{(b)} - 1$ and $x_1^{(b)}$ with $1 - \neg x_1^{(b)}$, we have: $y_1 = +1 \Leftrightarrow (-x_1^{(b)} + x_2^{(b)} + x_3^{(b)} + 0.85 \geq 0) \Leftrightarrow (\neg x_1^{(b)} + x_2^{(b)} + x_3^{(b)} \geq 0.15)$. Thus we get $y_1^{(b)} \Leftrightarrow \neg x_1^{(b)} + x_2^{(b)} + x_3^{(b)} \geq 1$ (note that $y_1^{(b)} = 0 \Leftrightarrow \neg x_1^{(b)} + x_2^{(b)} + x_3^{(b)} < 1$). Similarly, we can deduce that $o_1 \Leftrightarrow y_1 - y_2 \geq 0.7$, and thus $o_1 \Leftrightarrow y_1^{(b)} - y_2^{(b)} \geq 0.35 \Leftrightarrow y_1^{(b)} + \neg y_2^{(b)} \geq 2$.

3.5 BDD Model Builder

The construction of the BDDs $(G_i^{out})_{i \in [s]}$ from the BNN $\mathcal{N}^{(b)}$ and the input region R is done iteratively throughout the blocks. Initially, the BDD for the first block is built, which can be seen as the input-output relation for the first internal block. In the i-th iteration, as the input-output relation of the first $(i-1)$ internal blocks has been encoded into the BDD, we compose this BDD with the BDD for the block t_i which is built from its cardinality constraints $t_i^{(b)}$, resulting in the BDD for the first i internal blocks. Finally, we obtain the BDDs $(G_i^{out})_{i \in [s]}$ of the BNN $\mathcal{N}$, with respect to the input region R.

$\mathbf{b}_1$	α_1	μ_1	γ_1	σ_1	$\mathbf{b}_2$
0.2	0.02	−0.5	0.02	2	−0.8
−0.5	−0.03	1.5	−0.03	3	0.6

$\mathcal{N}(x)$
$y_1 = \text{sign}(-x_1 + x_2 + x_3 + 2.7)$
$y_2 = \text{sign}(-x_1 - x_2 + x_3 - 1)$
$o_1 \Leftrightarrow y_1 - y_2 \geq 0.7$
$o_2 \Leftrightarrow y_1 - y_2 < 0.7$



Cardinality Constraint Encoding
$y_1^{(b)} \Leftrightarrow \neg x_1^{(b)} + x_2^{(b)} + x_3^{(b)} \geq 1$
$y_2^{(b)} \Leftrightarrow \neg x_1^{(b)} + \neg x_2^{(b)} + x_3^{(b)} \geq 2$
$o_1 \Leftrightarrow y_1^{(b)} + \neg y_2^{(b)} \geq 2$
$o_2 \Leftrightarrow y_1^{(b)} + \neg y_2^{(b)} < 2$

Fig. 5. An illustrating example

Design Choice. There are several design choices for efficiency consideration which we discuss as follows. First of all, to encode the input-output relation of an internal block t_i into BDD from its cardinality constraints $t_i^{(b)} = \{C_{i,1}, \cdots, C_{i,n_{i+1}}\}$, we need to compute $\text{AND}_{j \in [n_{i+1}]}\text{CC2BDD}(C_{i,j})$. A simple and straightforward approach is to initially compute a BDD $G = \text{CC2BDD}(C_{i,1})$ and then iteratively compute the conjunction $G = \text{AND}(G, \text{CC2BDD}(C_{i,j}))$ of G and $\text{CC2BDD}(C_{i,j})$ for $2 \leq j \leq n_{i+1}$.

Alternatively, we use a divide-and-conquer strategy to recursively compute the BDDs for the first half and the second half of the cardinality constraints respectively, and then apply the AND-operation. Our preliminary experimental results show that the latter approach often performs better (about 2 times faster) than the former one, although they generate the same BDD.

Second, constructing the BDD directly from the cardinality constraints $t_i^{(b)} = \{C_{i,1}, \cdots, C_{i,n_{i+1}}\}$ becomes prohibitively costly when n_i and n_{i+1} are large, as the BDDs CC2BDD($C_{i,j}$) for $j \in [n_{i+1}]$ need to consider all the inputs in $\mathbb{B}^{n_i}$. To improve efficiency, we apply feasible input propagation. Namely, when we construct the BDD for the block t_{i+1}, we only consider its possible inputs with respect to the output of the block t_i. Our preliminary experimental results show that the optimization could significantly improve the efficiency of the BDD construction.

Third, instead of encoding the input-output relation of the BNN $\mathcal{N}$ as a sole BDD or MTBDD, we opt to use a family of s BDDs $(G_i^{out})_{i\in[s]}$, each of which corresponds to one output class of $\mathcal{N}$. Recall that each output class $i \in [s]$ is represented by $(s-1)$ cardinality constraints. Then, we can build a BDD G_i for the output class i, similar to the BDD construction for internal blocks. By composing G_i with the BDD of the entire internal blocks, we obtain the BDD G_i^{out}. Building a single BDD or MTBDD for the BNN is possible from $(G_i^{out})_{i\in[s]}$, but our approach gives the flexibility especially when a specific target class is interested, which is common for robustness analysis.

Algorithm 2: BDD Construction for BNNs

```
Proc BNN2BDD(BNN : $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$, Region : $R(\boldsymbol{u}, \tau)$)
    $G^{in} = G^{in}_{\boldsymbol{u},\tau}$ (cf. Section 3.3); $\mathcal{N}^{(b)} = (t_1^{(b)}, \cdots, t_d^{(b)}, t_{d+1}^{(b)})$ (cf. Section 3.4);
    for $(i = 1;\ i \le d;\ i++)$ do
        $G'$ =Block2BDD($t_i^{(b)}, G^{in}, i$);
        $G^{in}$ = Exists($G', \boldsymbol{x}^i$) ;        // $\boldsymbol{x}^i$ denote input variables of $t_i^{(b)}$
        $G = (i == 1)\ ?\ G' :$ RelProd($G, G'$);
    for $(i = 1;\ i \le s;\ i++)$ do
        $G_i$ =Block2BDD($t^{(b)}_{d+1\downarrow i}, G^{in}, d+1$);
        $G_i^{out}$ = RelProd($G_i, G$);
    return $(G_i^{out})_{i\in[s]}$
Proc Block2BDD(CCs : $\{C_m, \cdots, C_n\}$, InputSpace : $G^{in}$, BlkIndex : $i$)
    if $n == m$ then
        $G_1$ =CC2BDD($C_m$) (cf. Algorithm 1);
        $G$ = And($G_1, G^{in}$);
        if $i \neq d+1$ then $G$ = Xnor($\boldsymbol{x}_m^{i+1}, G$);
    else
        $G_1$ =Block2BDD($\{C_m, \cdots, C_{\lfloor\frac{n-m}{2}\rfloor+m}\}, G^{in}, i$);
        $G_2$ =Block2BDD($\{C_{\lfloor\frac{n-m}{2}\rfloor+m+1}, \cdots, C_n\}, G^{in}, i$);
        $G$ = And($G_1, G_2$);
    return $G$
```

Overall Algorithm. The overall BDD construction procedure is shown in Algorithm 2. Given a BNN $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$ with s output classes and an input region $R(\boldsymbol{u}, \tau)$, the algorithm outputs the BDDs $(G_i^{out})_{i\in[s]}$, encoding the input-output relation of the BNN $\mathcal{N}$ with respect to the input region $R(\boldsymbol{u}, \tau)$.

The procedure BNN2BDD first builds the BDD representation $G^{in}_{\boldsymbol{u},\tau}$ of the input region $R(\boldsymbol{u}, \tau)$ and the cardinality constraints from BNN $\mathcal{N}^{(b)}$ (Line 1). The first for-loop builds a BDD encoding the input-output relation of the entire internal blocks w.r.t. $G^{in}_{\boldsymbol{u},\tau}$. The second for-loop builds the BDDs $(G_i^{out})_{i\in[s]}$, each of which encodes the input-output relation of the entire BNN for a class $i \in [s]$ w.r.t. $G^{in}_{\boldsymbol{u},\tau}$. The procedure Block2BDD receives the cardinality constraints $\{C_m, \cdots, C_n\}$, a BDD G^{in} representing the feasible inputs of the block and the block index i as inputs, and returns a BDD G. If $i = d+1$, namely, the cardinality constraints $\{C_m, \cdots, C_n\}$ are from the output block, the resulting BDD G encodes the subset of $G^{in}_{\boldsymbol{u},\tau}$ that satisfy all the cardinality constraints $\{C_m, \cdots, C_n\}$. If $i \neq d + 1$, then the BDD G encodes the input-output relation of the Boolean function $f_{m,n}$ such that for every $\boldsymbol{x}^i \in \mathcal{L}(G^{in})$, $f_{m,n}(\boldsymbol{x}^i)$ is the truth vector of the cardinality constraints $\{C_m, \cdots, C_n\}$ under the valuation $\boldsymbol{x}^i$. When $m = 1$ and $n = n_{i+1}$, $f_{m,n}$ is the same as $t_i^{(b)}$, hence $\mathcal{L}(G) = \{\boldsymbol{x}^i \times \boldsymbol{x}^{i+1} \in G^{in} \times \mathbb{B}^{n_{i+1}} \mid t_i^{(b)}(\boldsymbol{x}^i) = \boldsymbol{x}^{i+1}\}$. Detailed explanation refers to [71].

Theorem 2. *Given a BNN $\mathcal{N}$ with s output classes and an input region $R(\boldsymbol{u}, \tau)$, we can compute s BDDs $(G_i^{out})_{i\in[s]}$ such that the BNN $\mathcal{N}$ classifies an input $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$ into the class $i \in [s]$ iff $\boldsymbol{x}^{(b)} \in \mathcal{L}(G_i^{out})$.*

Algorithm 2 explicitly involves $O(d+s)$ RelProd-operations, $O(s^2 + \sum_{i\in[d]} n_i)$ And-operations and $O(d)$ Exists-operations.

4 Applications: Robustness Analysis and Interpretability

In this section, we present two applications within BDD4BNN, i.e., robustness analysis and interpretability of BNNs.

4.1 Robustness Analysis

Definition 2. *Given a BNN $\mathcal{N}$ and an input region $R(\boldsymbol{u}, \tau)$, the BNN is (locally) robust w.r.t. the region $R(\boldsymbol{u}, \tau)$ if each sample $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$ is classified into the same class as the ground-truth class of $\boldsymbol{u}$.*

An adversarial example in the region $R(\boldsymbol{u}, \tau)$ is a sample $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$ such that $\boldsymbol{x}$ is classified into a class, that differs from the ground-truth class of $\boldsymbol{u}$.

As mentioned in Sect. 1, qualitative verification which checks whether a BNN is robust or not is insufficient in many practical applications. In this paper, we are interested in *quantitative* verification of robustness which asks *how many adversarial examples are there in the input region of the BNN for each class*. To answer this question, given a BNN $\mathcal{N}$ and an input region $R(\boldsymbol{u}, \tau)$, we first obtain the BDDs $(G_i^{out})_{i\in[s]}$ by applying Algorithm 2 and then count the number of adversarial examples for each class in the input region $R(\boldsymbol{u}, \tau)$. Note that counting adversarial examples amounts to computing $|R(\boldsymbol{u}, \tau)| - |\mathcal{L}(G_g^{out})|$, where g denotes the ground-truth class of $\boldsymbol{u}$, and $|\mathcal{L}(G_g^{out})|$ can be computed in time $O(|G_g^{out}|)$.

In some applications, more refined analysis is needed. For instance, it may be acceptable to misclassify a dog as a cat, but unacceptable to misclassify a tree as a car. This suggests that the robustness of BNNs may depend on the classes to which samples are misclassified. To capture this, we consider the notion of targeted robustness.

Definition 3. *Given a BNN $\mathcal{N}$, an input region $R(\boldsymbol{u}, \tau)$, and the class t, the BNN is t-*target-robust *w.r.t. the region $R(\boldsymbol{u}, \tau)$ if every sample $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$ is never classified into the class t. (Note that we assume that the ground-truth class of $\boldsymbol{u}$ differs from the class t.)*

The quantitative verification problem of t-target-robustness of a BNN asks *how many adversarial examples in the input region $R(\boldsymbol{u}, \tau)$ are misclassified to the class t by the BNN $\mathcal{N}$*. To answer this question, we first obtain the BDD G_t^{out} by applying Algorithm 2 and then count the number of adversarial examples by computing $|\mathcal{L}(G_t^{out})|$.

Note that, if one wants to compute the (locally) maximal safe Hamming distance that satisfies a robustness property for an input sample (e.g., the proportion of adversarial examples is below a threshold), our framework can incrementally compute such a distance without constructing the BDD models of the entire BNN from scratch.

Definition 4. *Given a BNN $\mathcal{N}$, input region $R(\boldsymbol{u}, r)$ and threshold $\epsilon \geq 0$, r_1 is the (locally) maximal safe Hamming distance of $R(\boldsymbol{u}, \tau)$, if one of the follows holds:*

- *if $Pr(R(\boldsymbol{u}, r)) > \epsilon$, then $Pr(R(\boldsymbol{u}, r_1)) \leq \epsilon$ and $Pr(R(\boldsymbol{u}, r')) > \epsilon$ for $r' : r_1 < r' < r$;*
- *if $Pr(R(\boldsymbol{u}, r)) \leq \epsilon$, then $Pr(R(\boldsymbol{u}, r_1 + 1)) > \epsilon$ and $Pr(R(\boldsymbol{u}, r')) \leq \epsilon$ for $r' : r < r' \leq r_1$;*

where $Pr(R(\boldsymbol{u}, r))$ is the probability $\frac{\sum_{i \in [s]. i \neq g} |\mathcal{L}(G_i^{out})|}{|R(\boldsymbol{u}, r)|}$ for g being the ground-truth class of $\boldsymbol{u}$, assuming a uniform distribution of adversarial samples.

Algorithm 3 shows the procedure to incrementally compute the maximal safe Hamming distance for a given threshold $\epsilon \geq 0$, input region $R(\boldsymbol{u}, r)$, and ground-truth class g of $\boldsymbol{u}$. Remark that $Pr(R(\boldsymbol{u}, r))$ may not be monotonic w.r.t. the Hamming distance r.

4.2 Interpretability

In general, interpretability addresses the question of *why some inputs in the input region are (mis)classified by the BNN into a specific class?* We consider the interpretability of BNNs using two complementary explanations, i.e., prime implicant explanations and essential features.

Definition 5. *Given a BNN $\mathcal{N}$, an input region $R(\boldsymbol{u}, \tau)$ and a class g, a* prime implicant explanation *(PI-explanation) of decisions made by the BNN $\mathcal{N}$ on the inputs $\mathcal{L}(G_g^{out})$ is a minimal set of literals $\{\ell_1, \cdots, \ell_k\}$ such that for every $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$, if $\boldsymbol{x}$ satisfies $\ell_1 \wedge \cdots \wedge \ell_k$, then $\boldsymbol{x}$ is classified into the class g by the BNN $\mathcal{N}$.*

Algorithm 3: Compute the maximal safe Hamming distance

```
Proc MaxHD(BNN : $\mathcal{N} = (t_1, \cdots, t_d, t_{d+1})$, Region : $R(\boldsymbol{u}, r)$, Threshold : $\epsilon$, Class : $g$)
    $(G_i^{out})_{i\in[s]}$ =BNN2BDD($\mathcal{N}, R(\boldsymbol{u}, r)$);
    if ($\frac{\sum_{i\in[s].i\neq g}|\mathcal{L}(G_i^{out})|}{|R(\boldsymbol{u},r)|} > \epsilon$) then // decrease $r$
        while ($r \geq 0$) do
            $r = r - 1$;
            $(G_i^{out})_{i\in[s]} = (\text{And}(G_{\boldsymbol{u},r}^{in}, G_i^{out}))_{i\in[s]}$;
            if ($\frac{\sum_{i\in[s].i\neq g}|\mathcal{L}(G_i^{out})|}{|R(\boldsymbol{u},r)|} \leq \epsilon$) then return $r$;
    else // increase $r$
        while ($r \leq n_1$) do // $n_1$ is the input size of the BNN $\mathcal{N}$
            $r = r + 1$;
            $(B_i^{out})_{i\in[s]}$ =BNN2BDD($\mathcal{N}, R(\boldsymbol{u}, r) \setminus R(\boldsymbol{u}, r-1)$);
            $(G_i^{out})_{i\in[s]} = (\text{Or}(B_i^{out}, G_i^{out}))_{i\in[s]}$;
            if ($\frac{\sum_{i\in[s].i\neq g}|\mathcal{L}(G_i^{out})|}{|R(\boldsymbol{u},r)|} > \epsilon$) then return $r - 1$;
    return $r$
```

Intuitively, a PI-explanation $\{\ell_1, \cdots, \ell_k\}$ indicates that $\{\mathsf{var}(\ell_1), \cdots, \mathsf{var}(\ell_k)\}$ are key features, namely, if fixed, the predication is guaranteed no matter how the remaining features change. Remark that there may be more than one PI-explanation for a set of inputs $\mathcal{L}(G_g^{out})$. When g is set to be the class of the benign input $\boldsymbol{u}$, a PI-explanation on G_g^{out} suggests why these samples are classified into g by the BNN $\mathcal{N}$.

Definition 6. *Given a BNN $\mathcal{N}$, an input region $R(\boldsymbol{u}, \tau)$ and a class g, the* essential features *for the inputs $\mathcal{L}(G_g^{out})$ are literals $\{\ell_1, \cdots, \ell_k\}$ such that every $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$, if $\boldsymbol{x}$ is classified into the class g by the BNN $\mathcal{N}$, then $\boldsymbol{x}$ satisfies $\ell_1 \wedge \cdots \wedge \ell_k$.*

Intuitively, the essential features $\{\ell_1, \cdots, \ell_k\}$ denote the key features such that all samples $\boldsymbol{x} \in R(\boldsymbol{u}, \tau)$ that are classified into the class g by the BNN $\mathcal{N}$ must agree on these features. Essential features differ from PI-explanations, where the former can be seen as a necessary condition, while the latter can be seen as a sufficient condition.

BDD libraries (e.g., CUDD [58]) usually provide APIs to identify prime implicants (e.g., Cudd_bddPrintCover and Cudd_FirstPrime) and essential variables (e.g., Cudd_FindEssential). Therefore, prime implicants and essential features can be computed via queries on the BDDs $(G_i^{out})_{i\in[s]}$.

5 Evaluation

We have implemented our framework as a prototype tool BDD4BNN based on the CUDD package [58]. BDD4BNN is implemented with Python as the front-end to preprocess BNNs and C++ as the back-end to perform the BDD encoding and analysis. In this section, we report the experimental results, including BDD encoding, robustness analysis based on hamming distance, and interpretability.

Experimental Setup. The experiments were conducted on a machine with Intel Xeon Gold 5118 2.3GHz CPU, 64-bit Ubuntu 20.04 LTS operating systems, 128 G RAM. Each BDD encoding executed on one core limited by 8-h.

Benchmarks. We use the PyTorch (v1.0.1.post2) deep learning platform provided by NPAQ [6] to train and test BNNs. We trained 12 BNN models (P1-P12) with varying sizes using the MNIST dataset [35]. The MNIST dateset contains 70,000 gray-scale 28 × 28 images (60,000 for training and 10,000 for testing) of handwritten digits with 10 classes. In our experiments, we downscale the images (28 × 28) to some selected input size n_1 (i.e., the corresponding image is of the size $\sqrt{n_1} \times \sqrt{n_1}$) and then binarize the normalized pixels of the images.

Details of the BNN models are listed in Table 3, each of which has 10 classes (i.e., $s = 10$). Column 1 shows the name of the BNN model. Column 2 shows the architecture of the BNN model, where $n_1 : \cdots : n_{d+1} : s$ denotes that the BNN model has $d + 1$ blocks, n_1 inputs and s outputs; the i-th block for $i \in [d + 1]$ has n_i inputs and n_{i+1} outputs with $n_{d+2} = s$. Recall that each internal block has 3 layers while the output block has 2 layers. Therefore, the number of layers ranges from 5 to 14, the dimension of inputs ranges from 9 to 784, and the number of hidden neurons per linear layer ranges from 10 to 100. Column 3 shows the accuracy of the BNN model on the test set of the MNIST dataset. (We can observe that the accuracy increases with the size of inputs, the number of layers, and the number of hidden neurons per layer.) We randomly choose 10 images from the training set of the MNIST dataset (one image per class) to evaluate our approach.

5.1 Performance of BDD Encoding

We evaluate BDD4BNN on the BNNs listed in Table 3 using different input regions.

BDD Encoding Using Full Input Space. We evaluate BDD4BNN on the BNNs (P1–P5), where $\mathbb{B}_{\pm1}^{n_1}$ is used as the input region. The results are shown in Table 4, where $|G|$ denotes the number of BDD nodes in the BDD manager. We can observe that both the execution time and the number of BDD nodes increase with the size of BNNs.

BDD Encoding Under Hamming Distance. We evaluate BDD4BNN on the BNNs (P5–P12). In this case, an input region is given by one of the 10 images and a Hamming distance r ranging from 2 to 6. The average results are shown in Table 5, where $[i]$ (resp. (i)) indicates the number of cases that BDD4BNN runs out of memory (resp. time). Overall, the execution time and the number of BDD nodes increase with r. BDD4BNN succeeded on all the cases when $r \leq 4$, 75 cases out of 80 when $r = 5$, and 48 cases out of 80 when $r = 6$. We observe that the execution time and number of BDD nodes increase with the number of hidden neurons (P6 vs. P7, P8 vs. P9, and P11 vs. P12), while the effect of the number of layers is diverse (P6 vs. P8 vs. P10, and P7 vs. P9). From P9 and P10, we observe that the number of hidden neurons per layer is likely the key impact factor of the efficiency of BDD4BNN. Interestingly, our tool BDD4BNN works well on BNNs with large input sizes (i.e., on P11 and P12).

Table 3. BNN benchmarks

Name	Architecture	Accuracy	Name	Architecture	Accuracy
P1	9:20:10	12.23%	P7	100:100:10	75.16%
P2	16:32:10	28.63%	P8	100:50:20:10	71.1%
P3	16:64:32:10	25.14%	P9	100:100:50:10	77.37%
P4	36:15:10:10	27.12%	P10	100:50:30:30:10	80.63%
P5	64:10:10	49.16%	P11	784:30:50:50:50:10	88.23%
P6	100:50:10	73.25%	P12	784:50:50:50:50:10	86.95%

Table 4. BDD encoding using full input space

Name	P1	P2	P3	P4	P5
Time (s)	≈0	0.78	28.21	10924.51	Timeout
$\|G\|$	288	18,864	17,636	152,830,875	–

These results demonstrate the efficiency and scalability of BDD4BNN on BDD encoding of BNNs. We remark that, compared with the learning-based approach [54], our approach is considerably more efficient and scalable. For instance, the learning-based approach takes 403 s to encode a BNN with 64 input size, 5 hidden neurons, and 2 output size when $r = 6$, while ours takes about 3 s even for a larger network P5.

5.2 Robustness Analysis

We evaluate BDD4BNN on the robustness of BNNs, including robustness analysis under different input regions and maximal safe Hamming distance computing.

Robustness Verification with Hamming Distance. We evaluate BDD4BNN on BNNs (P7, P8, P9, and P11) using the 10 images. The input regions are given by the Hamming distance r ranging from 2 to 4, resulting in 120 instances. To the best of our knowledge, NPAQ [6] is the only work that supports quantitative robustness verification of BNNs to which we compare BDD4BNN. Recall that NPAQ only provides PAC-style guarantees. Namely, it sets a tolerable error ε and a confidence parameter δ. The final estimated results of NPAQ have the bounded error ε with confidence of at least $1 - \delta$, i.e.,

$$Pr[(1+\varepsilon)^{-1}\texttt{RealNum} \leq \texttt{EstimatedNum} \leq (1+\varepsilon)\texttt{RealNum}] \geq 1-\delta \qquad (1)$$

In our experiments, we set $\varepsilon = 0.8$ and $\delta = 0.2$, as done in [6].

Table 5. BDD encoding under Hamming distance

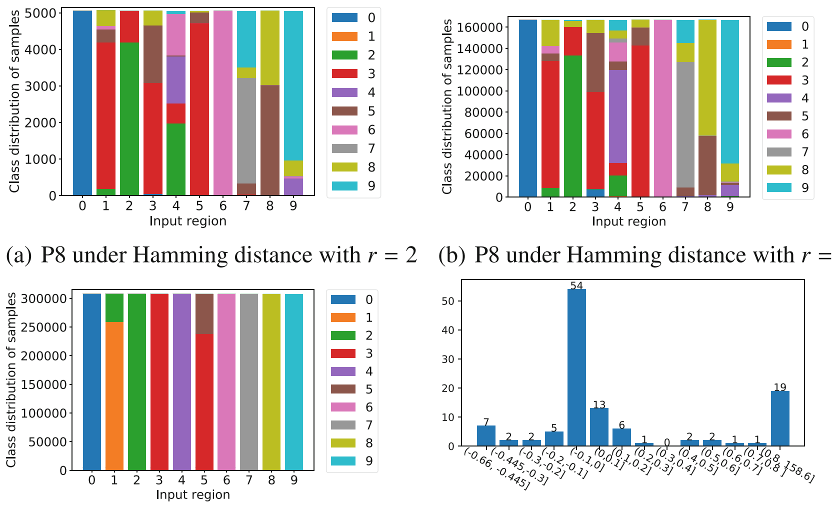
	r=2		r=3		r=4		r=5		r=6	
	Time(s)	$\|G\|$	Time(s)	$\|G\|$	Time(s)	$\|G\|$	Time(s)	$\|G\|$	Time(s)	$\|G\|$
P5	0.01	1,559	0.03	9,795	0.11	36,796	0.74	176,107	2.94	592,104
P6	0.25	4,670	4.17	84,037	109.26	1,018,571	2,292.5	11,375,842	(5) 17,811	41,883,970
P7	0.65	5,295	22.70	106,754	652.78	1,575,722	(1) 17,399	16,163,078	[10]	-
P8	0.14	6,147	1.95	125,226	44.51	1,668,027	1,146.8	20,519,582	(1) 12,491	172,369,297
P9	1.99	6,139	63.30	136,126	1,428.6	2,005,666	[1](3) 17,039	29,323,244	[10]	-
P10	0.30	4,630	4.87	100,054	101.41	1,603,920	1,909.9	19,844,299	(5) 20,484	173,316,483
P11	5.52	3,128	5.73	22,120	6.60	86,413	11.63	556,774	238.2	2,881,468
P12	12.4	5,693	12.87	49,996	16.92	493,820	403.09	5,739,602	(1) 11,058	16,241,733

Table 6. Robustness verification under Hamming distance

	r	NPAQ [6]			BDD4BNN			Diff	
		#(Adv)	Time(s)	Pr(adv)	#(Adv)	Time(s)	Pr(adv)	#(Adv)	Speed Up
P7	2	875	271.07	17.32%	1,806	0.65	35.76%	106.4%	416
	3	39,587	919.88	23.74%	65,054	22.71	39.01%	64.33%	40
	4	1,023,798	3,862.0	25.04%	1,501,691	661.79	36.73%	46.68%	5
P8	2	1,601	187.78	31.70%	2,261	0.14	44.76%	41.22%	1,340
	3	66,562	396.45	39.92%	64,372	1.96	38.60%	-3.29%	201
	4	1,636,070	1,861.7	40.02%	1,829,103	45.0	44.74%	11.80%	40
P9	2	1,214	363.44	24.03%	1,406	1.99	27.84%	15.82%	182
	3	51,464	3,763.6	30.86%	42,901	63.31	25.73%	-16.64%	58
	4	1,316,181	(1) 9,007.8	32.20%	3,968,609	1,505.0	97.08%	201.5%	5
P11	2	12,083	3,831.0	3.93%	28,736	5.52	9.34%	137.8%	693
	3	0	(2) 4,634.2	0%	0	5.68	0%	-	815
	4	0	(2) 7,979.1	0%	0	6.38	0%	-	1,250

The results on the average of the images are shown in Table 6. NPAQ ran out of time on 5 instances (which occur in P9 with $r = 4$ and P11 with $r = 3$ and $r = 4$), while BDD4BNN successfully verified all the 120 instances. Table 6 only shows the results of 115 instances that can be solved by NPAQ. Columns 3, 4, and 5 (resp. 6, 7, and 8) show the number of adversarial examples, the execution time, and the proportion of adversarial examples in the input region. Column 9 shows the error rate $\frac{\texttt{RealNum}-\texttt{EstimatedNum}}{\texttt{EstimatedNum}}$, where `RealNum` is from our result, and `EstimatedNum` is from NPAQ. Column 10 shows the speedup of BDD4BNN compared with NPAQ. Remark that the numbers of adversarial examples are 0 for P11 on input regions with $r = 3$ and $r = 4$ that can be solved by NPAQ. There do exist input regions for P11 that cannot be solved by NPAQ but have adversarial examples (see below). On BNNs that were solved by both NPAQ and BDD4BNN, BDD4BNN is significantly (5× to $1,340\times$) faster and more accurate than NPAQ. From Table 5 and Table 6, we also found that most of the verification time is spent on BDD encoding while the rest is usually less than 10 s.

Details of Robustness and Targeted Robustness. Figure 6(a) (resp. Fig. 6(b) and Fig. 6(c)) depicts the distributions of classes on P8 with Hamming distance $r = 2$ (resp. P8 with $r = 3$ and P11 with $r = 2$), where on the x-axis $i = 0, \cdots, 9$ denotes the input



(a) P8 under Hamming distance with $r = 2$ (b) P8 under Hamming distance with $r = 3$

(c) P11 under Hamming distance with $r = 2$ (d) Distribution of error rates of NPAQ

Fig. 6. Details of robustness verification with Hamming distance

region that is within the respective Hamming distance to the image of digit i (called i-region). We can observe that P8 is robust for the 0-region when $r = 2$ and robust for the 6-region when $r = 2$ and $r = 3$, but is not robust for the other regions. (Note P8 is not robust for 0-region when $r = 3$, which is hard to be visualized in Fig. 6(b) due to the small number of adversarial examples.) Most of the adversarial examples in the 1-region and 5-region are misclassified into the digit 3 by P8. P11 is not robust for the 1-region or the 5-region, but is robust for all the other regions. Though P8 and P11 are not robust on some input regions, indeed they are t-target-robust for many target classes t, e.g., P11 is t-target-robust for the 1-region when $t \neq 2$, and the 5-region when $t \neq 3$. (The raw data are given in [71].)

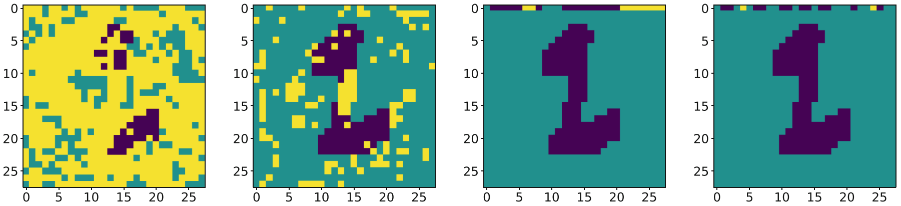
Quality Validation of NPAQ. Figure 6(d) shows the distribution of error rates of NPAQ, where the x-axis is the range of the error rate and the y-axis is the corresponding number of instances. There are 19 instances where the estimated number of adversarial examples exceeds $(1+\epsilon)$ of the real number of the adversarial examples and 7 instances where the estimated number of adversarial examples is less than $(1 + \epsilon)^{-1}$ of the real number of the adversarial examples. This means that out of 115 instances, only in 89 instances the estimated number is within the allowed range, which is less than $1 - \delta = 0.8$.

Maximal Safe Hamming Distance. As a representative of such an analysis, we evaluate BDD4BNN on 4 BNNs (P7, P8, P9, and P11) with 10 images for 2 robustness thresholds ($\epsilon = 0$ and $\epsilon = 0.03$). The initial Hamming distance r is 3. Intuitively, $\epsilon = 0$ (resp. $\epsilon = 0.03$) means that up to 0% (resp. 3%) samples in the input region can be adversarial.

Table 7 shows the results, where columns SD and Time give the maximal safe Hamming distance and the execution time, respectively. BDD4BNN solved 74 out of 80 instances. (For the remaining 6 instances, BDD4BNN ran out of time or memory, but

Table 7. Maximal safe Hamming distance

Image	P7				P8				P9				P11			
	$\epsilon = 0$		$\epsilon = 0.03$		$\epsilon = 0$		$\epsilon = 0.03$		$\epsilon = 0$		$\epsilon = 0.03$		$\epsilon = 0$		$\epsilon = 0.03$	
	SD	Time(s)	SD	Time(s)	SD	Time(s)	SD	Time(s)	SD	Time(s)	SD	Time(s)	SD	Time(s)	SD	Time(s)
0	1	15.09	**4**	10,845	2	0.51	**6**	Timeout	3	746.15	3	737.96	6	29.69	6	29.28
1	-1	19.96	-1	19.13	-1	2.84	-1	2.97	0	155.50	0	155.09	0	6.49	0	6.11
2	2	13.25	**3**	422.04	0	0.46	0	0.50	1	37.50	**4**	14,127	6	11,334	6	11,437
3	0	21.39	0	20.94	-1	1.92	-1	2.08	0	41.04	0	40.49	6	8,323.1	6	8,088.3
4	3	426.81	**5**	OOM	-1	2.41	-1	2.61	2	8.08	**5**	OOM	6	30.85	6	30.74
5	-1	15.60	-1	15.92	-1	0.68	-1	0.74	-1	22.54	-1	21.54	-1	7.03	-1	6.72
6	4	7,990.6	**5**	OOM	3	5.69	**4**	198.26	1	57.37	**4**	Timeout	6	44.57	6	45.12
7	-1	16.08	-1	15.90	-1	2.49	-1	2.52	1	89.49	**4**	Timeout	6	89.38	6	88.39
8	-1	19.02	-1	19.28	-1	1.71	-1	1.80	-1	80.16	-1	79.91	6	43.95	6	43.30
9	0	26.82	0	27.69	0	5.09	**1**	5.39	-1	109.04	-1	107.24	6	338.73	6	327.48



(a) EFs for class 2 (b) EFs for class 5 (c) PI for class 2 (d) PI for class 5

Fig. 7. Graphic representation of essential features and PI-explanations

it was still able to compute a larger safe Hamming distance.) We can observe that the maximal safe Hamming distance increases with the threshold ϵ on several BNNs and input regions. We can also observe that P11 is more robust than others, which is consistent with their accuracies (cf. Table 3). Remark that SD = -1 indicates that the input image itself is misclassified.

5.3 Interpretability

To demonstrate the ability of BDD4BNN on interpretability, we consider the analysis of the BNN P12 and the image $\boldsymbol{u}$ of digit 1.

Essential Features. For the input region given by the Hamming distance $r = 4$, we compute two sets of essential features for the inputs $\mathcal{L}(G_2^{out})$ and $\mathcal{L}(G_5^{out})$, i.e., the adversarial examples in the region $R(\boldsymbol{u}, 4)$ that are misclassified into the classes 2 and 5 respectively. The essential features are depicted in Figs. 7(a) and 7(b), where black (resp. blue) color means that the value of the corresponding pixel is 1 (resp. 0), and yellow color means that the value of the corresponding pixel can take arbitrary values. Figure 7(a) (resp. Fig. 7(b)) indicates that the inputs $\mathcal{L}(G_2^{out})$ (resp. $\mathcal{L}(G_5^{out})$) must agree on these black- and blue-colored pixels.

PI-Explanations. For demonstration, we assume that the input region is given by the fixed set of indices $I = \{1, 2, \cdots, 28\}$ which denotes the first row of pixels of 28×28 images. We compute two PI-explanations of the inputs $\mathcal{L}(G_2^{out})$ and $\mathcal{L}(G_5^{out})$. The PI-explanations are depicted in Figs. 7(c) and 7(d). Figure 7(c) (resp. Fig. 7(d)) suggests that, by the definition of the PI-explanation, all the images in the region $R(\boldsymbol{u}, I)$ obtained by assigning arbitrary values to the yellow-colored pixels are always misclassified into the class 2 (resp. class 5), while changing one black-colored or blue-colored pixel would change the predication result since a PI-explanation is a minimal set of literals.

6 Related Work

In this section, we discuss the related work on qualitative/quantitative analysis and interpretability of DNNs. As there is a vast amount of literature regarding these topics, we will only discuss the most related ones to BDD4BNN.

Qualitative Analysis of DNNs. For real-numbered DNNs, various formal verification approaches have been proposed. Typical examples include constraint solving based approaches [17,26,30,31,51], optimization based approaches [10,13,15,16,40,61,67, 68], and program analysis based approaches [2,3,18,20,37–39,55–57,62–64,69].

Existing techniques for quantized DNNs are mostly based on constraint solving, in particular, SAT/SMT solving [12,33,45,46]. Following this line, verification of BNNs with ternary weights [28,48] and quantized DNNs with multiple bits [7,22,24] were also studied. Recently, the SMT-based framework Marabou for real-numbered DNNs [31] has also been extended to support BNNs [1].

Quantitative Analysis of DNNs. Comparing to qualitative analysis, quantitative analysis of neural networks is currently very limited. Two sampling-based approaches were proposed to certify the robustness for both DNNs and BNNs [5,65]. Yang et al. [69] proposed a spurious region-guided refinement approach for real-numbered DNN verification, claiming to be the first work of the quantitative robustness verification of DNNs with soundness guarantees.

Following the SAT-based qualitative analysis of BNNs [45,46], SAT-based quantitative analysis approaches were also proposed [6,21,47]. In particular, approximate SAT model-counting solvers are utilized. Shih et al. [54] also proposed a BDD-based approach to tackle BNNs, similar to our work in spirit. However, our approach is able to handle BNNs of considerably larger sizes than their learning-based method.

Interpretability of DNNs. Though interpretability of DNNs is crucial for explaining predictions, it is very challenging to tackle due to the blackbox nature of DNNs. There is a large body of work on the interpretability of DNNs (cf. [25,43] for a survey). Almost all the existing approaches are heuristic-based and restricted to finding explanations that are local in an input region. Some of them tackle the interpretability of DNNs by learning an interpretable model, such as binary decision trees [19,70] or finite-state automata [66]. In contrast to ours, they target at DNNs and only approximate the original model in the input region. The BDD-based approach [54] mentioned above has been used to compute the PI-explanation, but essential features were not considered therein.

7 Conclusion

In this paper, we have proposed a novel BDD-based framework for quantitative verification of BNNs. We implemented the framework as a prototype tool BDD4BNN and conducted extensive experiments on 12 BNN models with varying sizes and input regions. Experimental results demonstrated that BDD4BNN is more scalable than the existing BDD-learning based approach, and significantly more efficient and accurate than the existing SAT-based approach NPAQ. This work represents the first, but a key, step of the long-term program to develop an efficient and scalable BDD-based quantitative analysis framework for BNNs.

References

1. Amir, G., Wu, H., Barrett, C.W., Katz, G.: An SMT-based approach for verifying binarized neural networks. CoRR abs/2011.02948 (2020)
2. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: PLDI, pp. 731–744 (2019)
3. Ashok, P., Hashemi, V., Křetínský, J., Mohr, S.: DeepAbstract: neural network abstraction for accelerating verification. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 92–107. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_5
4. Baidu: Apollo (2021). https://apollo.auto
5. Baluta, T., Chua, Z.L., Meel, K.S., Saxena, P.: Scalable quantitative verification for deep neural networks. CoRR abs/2002.06864 (2020)
6. Baluta, T., Shen, S., Shinde, S., Meel, K.S., Saxena, P.: Quantitative verification of neural networks and its security applications. In: CCS, pp. 1249–1264 (2019)
7. Baranowski, M., He, S., Lechner, M., Nguyen, T.S., Rakamarić, Z.: An SMT theory of fixed-point arithmetic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 13–31. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_2
8. Bartzis, C., Bultan, T.: Construction of efficient BDDs for bounded arithmetic constraints. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 394–408. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_28
9. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)
10. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. J. Mach. Learn. Res. **21**, 42:1-42:39 (2020)
11. Chen, G., et al.: Who is real Bob? Adversarial attacks on speaker recognition systems. CoRR abs/1911.01840 (2019)
12. Cheng, C.-H., Nührenberg, G., Huang, C.-H., Ruess, H.: Verification of binarized neural networks via inter-neuron factoring. In: Piskac, R., Rümmer, P. (eds.) VSTTE 2018. LNCS, vol. 11294, pp. 279–290. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03592-1_16

13. Cheng, C.-H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 251–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_18
14. Duan, Y., Zhao, Z., Bu, L., Song, F.: Things you may not know about adversarial example: a black-box adversarial image attack. CoRR abs/1905.07672 (2019)
15. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) NFM 2018. LNCS, vol. 10811, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77935-5_9
16. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI, pp. 550–559 (2018)
17. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
18. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 43–65. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_3
19. Frosst, N., Hinton, G.E.: Distilling a neural network into a soft decision tree. In: Proceedings of the 1st International Workshop on Comprehensibility and Explanation in AI and ML (2017)
20. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI^2: safety and robustness certification of neural networks with abstract interpretation. In: S&P, pp. 3–18 (2018)
21. Ghosh, B., Basu, D., Meel, K.S.: Justicia: a stochastic SAT approach to formally verify fairness. CoRR abs/2009.06516 (2020)
22. Giacobbe, M., Henzinger, T.A., Lechner, M.: How many bits does it take to quantize your neural network? In: TACAS 2020. LNCS, vol. 12079, pp. 79–97. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45237-7_5
23. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: ICML, pp. 1737–1746 (2015)
24. Henzinger, T.A., Lechner, M., Žikelić, D.: Scalable verification of quantized neural networks (technical report). arXiv preprint arXiv:2012.08185 (2020)
25. Huang, X., et al.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev. **37**, 100270 (2020)
26. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
27. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: NeurIPS, pp. 4107–4115 (2016)
28. Jia, K., Rinard, M.: Efficient exact verification of binarized neural networks. In: NeurIPS (2020)
29. Kalra, N., Paddock, S.M.: Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp. Res. Part A Policy Pract. **94**, 182–193 (2016)
30. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5

31. Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
32. Koopman, P., Osyk, B.: Safety argument considerations for public road testing of autonomous vehicles. SAE Int. J. Adv. Curr. Pract. Mobility **1**, 512–523 (2019)
33. Korneev, S., Narodytska, N., Pulina, L., Tacchella, A., Bjorner, N., Sagiv, M.: Constrained image generation using binarized neural networks with decision procedures. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 438–449. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_27
34. Kung, J., Zhang, D.C., van der Wal, G.S., Chai, S.M., Mukhopadhyay, S.: Efficient object detection using embedded binarized neural networks. J. Signal Process. Syst. **90**(6), 877–890 (2018)
35. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010)
36. Lei, Y., Chen, S., Fan, L., Song, F., Liu, Y.: Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers. CoRR abs/2004.06954 (2020)
37. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: towards higher precision and faster verification. In: Chang, B.-Y.E. (ed.) SAS 2019. LNCS, vol. 11822, pp. 296–319. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_15
38. Li, R., et al.: PRODeep: a platform for robustness verification of deep neural networks. In: FSE, pp. 1630–1634 (2020)
39. Liu, W., Song, F., Zhang, T., Wang, J.: Verifying ReLU neural networks from a model checking perspective. J. Comput. Sci. Technol. **35**(6), 1365–1381 (2020)
40. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. CoRR abs/1706.07351 (2017)
41. McDanel, B., Teerapittayanon, S., Kung, H.T.: Embedded binarized neural networks. In: EWSN, pp. 168–173 (2017)
42. Minato, S.I., Somenzi, F.: Arithmetic Boolean expression manipulator using BDDs. Formal Methods Syst. Des. **10**(2), 221–242 (1997). https://doi.org/10.1023/A:1008643722423
43. Molnar, C., Casalicchio, G., Bischl, B.: Interpretable machine learning - A brief history, state-of-the-art and challenges. CoRR abs/2010.09337 (2020)
44. Nakamura, A.: An efficient query learning algorithm for ordered binary decision diagrams. Inf. Comput. **201**(2), 178–198 (2005)
45. Narodytska, N.: Formal analysis of deep binarized neural networks. In: IJCAI, pp. 5692–5696 (2018)
46. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: AAAI, pp. 6615–6624 (2018)
47. Narodytska, N., Shrotri, A., Meel, K.S., Ignatiev, A., Marques-Silva, J.: Assessing heuristic machine learning explanations with model counting. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 267–278. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_19
48. Narodytska, N., Zhang, H., Gupta, A., Walsh, T.: In search for a SAT-friendly binarized neural network architecture. In: ICLR (2020)
49. Papernot, N., McDaniel, P.D., Goodfellow, I.J., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: CCS, pp. 506–519 (2017)
50. Papernot, N., McDaniel, P.D., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: S&P, pp. 372–387 (2016)
51. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 243–257. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24

52. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
53. Shen, D., Wu, G., Suk, H.I.: Deep learning in medical image analysis. Annu. Rev. Biomed. Eng. **19**, 221–248 (2017)
54. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by Angluin-style learning. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 354–370. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_25
55. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: NeurIPS, pp. 15072–15083 (2019)
56. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: NeurIPS, pp. 10825–10836 (2018)
57. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. (POPL) **3**, 41:1–41:30 (2019)
58. Somenzi, F.: CUDD: CU decision diagram package (2015)
59. Szegedy, C., et al.: Intriguing properties of neural networks. In: ICLR (2014)
60. Tan, M., Le, Q.V.: EfficientNet: rethinking model scaling for convolutional neural networks. In: ICML, pp. 6105–6114 (2019)
61. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019)
62. Tran, H.-D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using ImageStars. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 18–42. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_2
63. Tran, H.-D., et al.: Star-based reachability analysis of deep neural networks. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 670–686. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_39
64. Wan, W., Zhang, Z., Zhu, Y., Zhang, M., Song, F.: Accelerating robustness verification of deep neural networks guided by target labels. CoRR abs/2007.08520 (2020)
65. Webb, S., Rainforth, T., Teh, Y.W., Kumar, M.P.: A statistical approach to assessing neural network robustness. In: ICLR (2019)
66. Weiss, G., Goldberg, Y., Yahav, E.: Extracting automata from recurrent neural networks using queries and counterexamples. In: ICML, pp. 5244–5253 (2018)
67. Wong, E., Kolter, J.Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: ICML, pp. 5283–5292 (2018)
68. Xiang, W., Tran, H., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. TNNLS **29**(11), 5777–5783 (2018)
69. Yang, P., et al.: Improving neural network verification through spurious region guided refinement. CoRR abs/2010.07722 (2020)
70. Zhang, Q., Yang, Y., Ma, H., Wu, Y.N.: Interpreting CNNs via decision trees. In: CVPR, pp. 6261–6270 (2019)
71. Zhang, Y., Zhao, Z., Chen, G., Song, F., Chen, T.: BDD4BNN: a BDD-based quantitative analysis framework for binarized neural networks. CoRR abs/2103.07224 (2021)