Scalable Verification of Quantized Neural Networks

Thomas A. Henzinger, Mathias Lechner, Đorđe Žikelić
IST Austria
Klosterneuburg, Austria
{tah,mlechner, dzikelic}@ist.ac.at

Abstract

Formal verification of neural networks is an active topic of research, and recent advances have significantly increased the size of the networks that verification tools can handle. However, most methods are designed for verification of an idealized model of the actual network which works over real arithmetic and ignores rounding imprecisions. This idealization is in stark contrast to network quantization, which is a technique that trades numerical precision for computational efficiency and is, therefore, often applied in practice. Neglecting rounding errors of such low-bit quantized neural networks has been shown to lead to wrong conclusions about the network's correctness. Thus, the desired approach for verifying quantized neural networks would be one that takes these rounding errors into account. In this paper, we show that verifying the bit-exact implementation of quantized neural networks with bit-vector specifications is PSPACE-hard, even though verifying idealized real-valued networks and satisfiability of bit-vector specifications alone are each in NP. Furthermore, we explore several practical heuristics toward closing the complexity gap between idealized and bit-exact verification. In particular, we propose three techniques for making SMT-based verification of quantized neural networks more scalable. Our experiments demonstrate that our proposed methods allow a speedup of up to three orders of magnitude over existing approaches.

Introduction

Deep neural networks for image classification typically consist of a large number of sequentially composed layers. Computing the output of such a network for a single input sample may require more than a billion floating-point operations (Tan and Le 2019). Consequently, deploying a trained deep neural network imposes demanding requirements on the computational resources available at the computing device that runs the network. Quantization of neural networks is a technique that reduces the computational cost of running a neural network by reducing the arithmetic precision of computations inside the network (Jacob et al. 2018). As a result, quantization has been widely adapted in industry for deploying neural networks in a resource-friendly way. For instance, Tesla's Autopilot Hardware 3.0 is designed for primarily running 8-bit quantized neural networks [1].

The verification problem for neural networks consists of checking validity of some input-output relation. More precisely, given two conditions over inputs and outputs of the network, the goal is to check if for every input sample which satisfies the input condition, the corresponding output of the neural network satisfies the output condition. Verification of neural networks has many important practical applications such as checking robustness to adversarial attacks (Szegedy et al. 2013; Tjeng, Xiao, and Tedrake 2019), proving safety in safety-critical applications (Huang et al. 2017) or output range analysis (Dutta, Chen, and Sankaranarayanan 2019), to name a few. There are many efficient methods for verification of neural networks (e.g. (Katz et al. 2017; Tjeng, Xiao, and Tedrake 2019; Bunel et al. 2018)), however most of them ignore rounding errors in computations. The few approaches that can handle the semantics of rounding operations are overapproximation-based methods, i.e., incomplete verification (Singh et al. 2018, 2019). The imprecision introduced by quantization stands in stark contrast with the idealization made by verification methods for standard neural networks, which disregards rounding errors that appear due to the network's semantics. Consequently, verification methods developed for standard networks are not sound for and cannot be applied to quantized neural networks. Indeed, recently it has been shown that specifications that hold for a floating-point representation of a network need not necessarily hold after quantizing the network (Giacobbe, Henzinger, and Lechner 2020). As a result, specialized verification methods that take quantization into account need to be developed, due to more complex semantics of quantized neural networks. Groundwork on such methods demonstrated that special encodings of networks in terms of Satisfiability Modulo Theories (SMT) (Clark and Cesare 2018) with bit-vector (Giacobbe, Henzinger, and Lechner 2020) or fixed-point (Baranowski et al. 2020) theories present a promising approach towards the verification of quantized networks. However, the size of networks that these tools can handle and runtimes of these approaches do not match the efficiency of advanced verification methods developed for standard networks like Reluplex(Katz et al. 2017) and Neurify (Wang et al. 2018a).

In this paper, we provide first evidence that the verifica-

[1] https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip

tion problem for quantized neural networks is harder compared to verification of their idealized counterparts, thus explaining the scalability-gap between existing methods for standard and quantized network verification. In particular, we show that verifying quantized neural networks with bit-vector specifications is PSPACE-hard, despite the satisfiability problem of formulas in the given specification logic being in NP. As verification of neural networks without quantization is known to be NP-complete (Katz et al. 2017), this implies that the verification of quantized neural networks is a harder problem.

We then address the scalability limitation of SMT-based methods for verification of quantized neural networks, and propose three techniques for their more efficient SMT encoding. First, we introduce a technique for identifying those variables and constraints whose value can be determined in advance, thus decreasing the size of SMT-encodings of networks. Second, we show how to encode variables as bit-vectors of minimal necessary bit-width. This significantly reduces the size of bit-vector encoding of networks in (Giacobbe, Henzinger, and Lechner 2020). Third, we propose a redundancy elimination heuristic which exploits bit-level redundancies occurring in the semantics of the network.

Finally, we propose a new method for the analysis of the quantized network's reachable value range, which is based on abstract interpretation and assists our new techniques for SMT-encoding of quantized networks. We evaluate our approach on two well-studied adversarial robustness verification benchmarks. Our evaluation demonstrates that the combined effect of our techniques is a speed-up of over three orders of magnitude compared to the existing tools.

The rest of this work is organized as follows: First, we provide background and discuss related works on the verification of neural networks and quantized neural networks. We then start with our contribution by showing that the verification problem for quantized neural networks with bit-vector specifications is PSPACE-hard. In the following section, we propose several improvements to the existing SMT-encodings of quantized neural networks. Finally, we present our experimental evaluation to assess the performance impacts of our techniques.

Background and Related Work

A neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that consists of several layers $f = l_1 \circ l_2 \circ \cdots \circ l_k$ that are sequentially composed, with each layer parameterized by learned weight values. Commonly found types of layers are linear

$$l(x) = Wx + b, W \in \mathbb{R}^{n_o \times n_i}, b \in \mathbb{R}^{n_o}, \tag{1}$$

ReLU $l(x) = \max\{x, 0\}$, and convolutional layers (LeCun et al. 1998).

In practice, the function f is implemented by floating-point arithmetic instead of real-valued computations. To distinguish a neural network from its approximation, we define an interpretation $[\![f]\!]$ as a map which assigns a new function to each network, i.e.

$$[\![]\!] : (\mathbb{R}^n \to \mathbb{R}^m) \to (\mathcal{D} \to \mathbb{R}^m), \tag{2}$$

where $\mathcal{D} \subset \mathbb{R}^n$ is the admissible input domain. For instance, we denote by $[\![f]\!]_{\mathbb{R}} : f \mapsto f$ the idealized real-valued abstraction of a network f, whereas $[\![f]\!]_{\text{float32}}$ denotes its floating-point implementation, i.e. the realization of f using 32-bit IEEE floating-point (Kahan 1996) instead of real arithmetic. Evaluating f, even under floating-point interpretation, can be costly in terms of computations and memory resources. In order to reduce these resource requirements, networks are usually quantized before being deployed to end devices (Jacob et al. 2018).

Formally, quantization is an interpretation $[\![f]\!]_{\text{int-}k}$ that evaluates a network f which uses k-bit fixed-point arithmetic (Smith et al. 1997), e.g. 4 to 8 bits. Let $[\mathbb{Z}]_k = \{0, 1\}^k$ denote the set of all bit-vectors of bit-width k. For each layer $l : [\mathbb{Z}]_k^{n_i} \to [\mathbb{Z}]_k^{n_0}$ in $[\![f]\!]_{\text{int-}k}$, we define its semantics by defining $l(x_1, \ldots, x_{n_i}) = (y_1, \ldots, y_{n_0})$ as follows:

$$x'_i = \sum_{j=1}^{n_i} w_{ij} x_j + b_i, \tag{3}$$

$$x''_i = \text{round}(x'_i, k_i) = \lfloor x'_i \cdot 2^{-k_i} \rfloor, \quad \text{and} \tag{4}$$

$$y_i = \max\{0, \min\{2^{N_i} - 1, x''_i\}\}, \tag{5}$$

Here, $w_{i,j}$ and b_i for each $1 \leq j \leq n_i$ and $1 \leq i \leq n_0$ denote the learned weights and biases of f, and k_i and N_i denote the bit-shift and the cut-off value associated to each variable y_i, respectively. Eq. (3) multiplies the inputs x_j with the weight values w_{ij} and adds the bias b_i, eq. (4) rounds the result to the nearest valid k-bit fixed-point value, and eq. (5) is a non-linear ReLU-N activation function [2].

An illustration of how the computations inside a network differ based on the used interpretation is shown in Fig. 1.

Verification of Neural Networks

The verification problem for a neural network and its given interpretation consists of verifying some input-output relation. More formally, given a neural network f, its interpretation $[\![f]\!]$ and two predicates φ and ψ over the input domain $\mathcal{D}$ and output domain $\mathbb{R}^m$ of $[\![f]\!]$, we want to check validity of the following formula (i.e. whether it holds for each $x \in \mathcal{D}$)

$$\varphi(x) \wedge [\![f]\!](x) = y \implies \psi(y). \tag{6}$$

We refer to the formula in eq. (6) as the formal specification that needs to be proved. In order to formally verify a neural network, it is insufficient to just specify the network without also providing a particular interpretation. A property that holds with respect to one interpretation need not necessarily remain true if we consider a different interpretation. For example, robustness of the real-valued abstraction does not imply robustness of the floating-point implementation of a network (Jia and Rinard 2020).

Ideally, we would like to verify neural networks under the exact semantics that are used for running networks on the end device, i.e., $[\![f]\!]_{\text{float32}}$ most of the time. However, as verification methods for IEEE floating-point arithmetic are

[2]Note that for quanitzed neural networks, the double-side bounded ReLU-N activation is preferred over the standard ReLU activation function (Jacob et al. 2018)

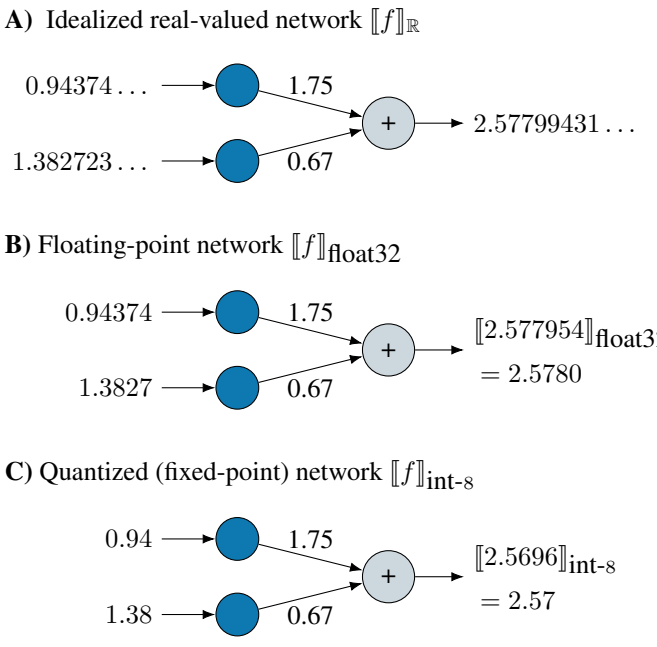

Figure 1: Illustration of how different interpretations of the same network run with different numerical precision. **A)** $[\![f]\!]_{\mathbb{R}}$ assumes infinite precision. **B)** $[\![f]\!]_{\text{float32}}$ rounds the mantissa according on the IEEE 754 standard. **C)** $[\![f]\!]_{\text{int-8}}$ rounds to a fixed number of digits before and after the comma. (Note that this figure serves as a hypothetical example in decimal format.)

extremely inefficient, research has focused on verifying the idealized real-valued abstraction $[\![f]\!]_{\mathbb{R}}$ of f. In particular, efficient methods have been developed for a popular type or networks that only consist of linear and ReLU operations (Figure 2 a) (Katz et al. 2017; Ehlers 2017; Tjeng, Xiao, and Tedrake 2019; Bunel et al. 2018). The piecewise linearity of such ReLU networks allows the use of Linear Programming (LP) techniques, which make the verification methods more efficient. The underlying verification problem of ReLU networks with linear inequality specifications was shown to be NP-complete in the number of ReLU operations (Katz et al. 2017), however advanced tools scale beyond toy networks.

Although these methods can handle networks of large size, they are building on the assumption that

$$[\![f]\!]_{\text{float32}} \approx [\![f]\!]_{\mathbb{R}}, \tag{7}$$

i.e. that the rounding errors introduced by the IEEE floating-point arithmetic of both the network and the verification algorithm can be neglected. It has been recently shown that this need not always be true. For example, Jia and Rinard (Jia and Rinard 2020) crafted adversarial counterexamples to the floating-point implementation of a neural network whose idealized interpretation was verified to be robust against such attacks, by exploiting subtle numerical differences between $[\![f]\!]_{\text{float32}}$ and $[\![f]\!]_{\mathbb{R}}$.

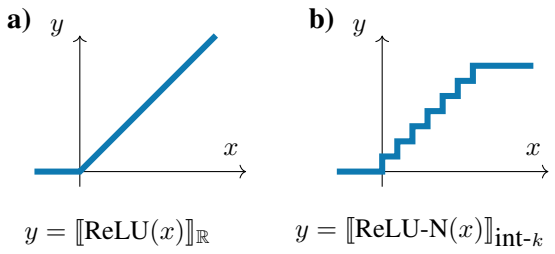

Figure 2: Illustration of **a)** the ReLU activation function under real-valued semantics, and **b)** ReLU-N activation under fixed-point semantics (right).

Verification of Quantized Neural Networks

The low numerical precision of few-bit fixed-point arithmetic implies that $[\![f]\!]_{\text{int-}k} \neq [\![f]\!]_{\mathbb{R}}$. Indeed, (Giacobbe, Henzinger, and Lechner 2020) constructed a prototypical network that either satisfies or violates a formal specification, depending on the numerical precision used to evaluate the network. Moreover, they observed such discrepancy in networks found in practice. Thus, no formal guarantee on $[\![f]\!]_{\text{int-}k}$ can be obtained by verifying $[\![f]\!]_{\mathbb{R}}$ or $[\![f]\!]_{\text{float32}}$. In order to verify fixed-point implementations of (i.e. quantized) neural networks, new approaches are required.

Fig. 2 depicts the ReLU activation function for idealized real-valued ReLU networks and for quantized ReLU networks, respectively. The activation function under fixed-point semantics consists of an exponential number of piecewise constant intervals thus making the LP-based techniques, which otherwise work well for real-valued networks, extremely inefficient. So the approaches developed for idealized real-valued ReLU networks cannot be efficiently applied to quantized networks. Existing verification methods for quantized neural networks are based on bit-exact Boolean Satisfiability (SAT) and SMT encodings. For 1-bit networks, i.e., binarized neural networks, Narodytska et al. (Narodytska et al. 2018) and (Cheng et al. 2018) proposed to encode the network semantics and the formal specification into an SAT formula, which is then checked by an off-the-shelf SAT solver. While their approach could handle networks of decent size, the use of SAT-solving is limited to binarized networks, which are not very common in practice.

(Giacobbe, Henzinger, and Lechner 2020) proposed to verify many-bit quantized neural network by encoding their semantics and specifications into quantifier-free bit-vector SMT (QF_BV) formulas. The authors showed that, by reordering linear summations inside the network, such monolithic bit-vector SMT encodings could scale to the verification of small but interestingly sized networks.

(Baranowski et al. 2020) introduced an SMT theory for fixed-point arithmetic and showed that the semantics of quantized neural networks could be encoded in this theory very naturally. However, as the authors only proposed prototype solvers for reference purposes, the size of the verified networks was limited.

Limitations of Neural Network Verification

The existing techniques for verification of idealized real-valued abstractions of neural networks have significantly increased the size of networks that can be verified (Ehlers 2017; Katz et al. 2017; Bunel et al. 2018; Tjeng, Xiao, and Tedrake 2019). However, scalability remains the key challenge hindering formal verification of neural networks in practice. For instance, even the largest networks verified by the existing methods (Ruan, Huang, and Kwiatkowska 2018) are tiny compared to the network architectures used for object detection and image classification (He et al. 2016).

Regarding the verification of quantized neural networks, no advanced techniques aiming at performance improvements have been studied so far. In this paper, we address the scalability of quantized neural network verification methods that rely on SMT-solving.

Hardness of Verification of Quantized Neural Networks

The size of quantized neural networks that existing verification methods can handle is significantly smaller compared to the real arithmetic networks that can be verified by the state-of-the-art tools like (Katz et al. 2017; Tjeng, Xiao, and Tedrake 2019; Bunel et al. 2018). Thus, a natural question is whether this gap in scalability is only because existing methods for quantized neural networks are less efficient, or if the verification problem for quantized neural networks is computationally harder.

In this section, we study the computational complexity of the verification problem for quantized neural networks. For idealized real arithmetic interpretation of neural networks, it was shown in (Katz et al. 2017) that, if predicates on inputs and outputs are given as conjunctions of linear inequalities, then the problem is NP-complete. The fact that the problem is NP-hard is established by reduction from 3-SAT, and the same argument can be used to show that the verification problem for quantized neural networks is also NP-hard. In this work, we argue that the verification problem for quantized neural networks with bit-vector specifications is in fact PSPACE-hard, and thus harder then verifying real arithmetic neural networks. Moreover, we show that this holds even for the special case when there are no constraints on the inputs of the network, i.e. when the predicate on inputs is assumed to be a tautology. The verification problem for a quantized neural network f that we consider consists of checking validity of a given input-output relation formula

$$[\![f]\!]_{\text{int-}k}(x) = y \implies \psi(y).$$

Here, $[\![f]\!]_{\text{int-}k}$ is the k-bit fixed point arithmetic interpretation of f, and ψ is a predicate in some specification logic over the outputs of $[\![f]\!]_{\text{int-}k}$. Equivalently, we may also check satisfiability of the dual formula

$$[\![f]\!]_{\text{int-}k}(x) = y \wedge \neg\psi(y). \tag{8}$$

In order to study complexity of the verification problem, we also need to specify the specification logic to which formula ψ belongs. In this work, we study hardness with respect to the fragment QF_BV2$_{bw}$ of the fixed-size bit-vector logic QF_BV2 (Kovásznai, Fröhlich, and Biere 2016). The fragment QF_BV2$_{bw}$ allows bit-wise logical operations (such as bit-wise conjunction, disjunction and negation) and the equality operator. The index 2 in QF_BV2$_{bw}$ is used to denote that the constants and bit-widths are given in binary representation. It was shown in (Kovásznai, Fröhlich, and Biere 2016) that the satisfiability problem for formulas in QF_BV2$_{bw}$ is NP-complete.

Even though QF_BV2$_{bw}$ itself allows only bit-vector operations and not linear integer arithmetic, we show that by introducing dummy output variables in $[\![f]\!]_{\text{int-}k}$ we may still encode formal specifications on outputs that are boolean combinations of linear inequalities over network's outputs. Thus, this specification logic is sufficiently expressive to encode formal specifications most often seen in practice. Let $y_1, \ldots, y_m$ denote output variables of $[\![f]\!]_{\text{int-}k}$. In order to encode an inequality of the form $a_1y_1 + \cdots + a_my_m + b \geq 0$ into the output specification, we do the following:

- Introduce an additional output neuron $\tilde{y}$ and a directed edge from each output neuron y_i to $\tilde{y}$. Let a_i be the weight of an edge from y_i to $\tilde{y}$, b be the bias term of $\tilde{y}$, $k-1$ be the bit-shift value of $\tilde{y}$, and $N = k$ be the number of bits defining the cut-off value of $\tilde{y}$. Then

  $$\tilde{y} = \text{ReLU-N}(\text{round}(2^{-(k-1)}(a_1y_1 + \cdots + a_my_m + b))).$$

  Thus, as we work with bit-vectors of bit-width k, $\tilde{y}$ is just the sign bit of $a_1y_1 + \cdots + a_sy_s + b$ preceded by zeros.

- As $a_1y_1 + \cdots + a_sy_s + b \geq 0$ holds if and only if the sign bit of $a_1y_1 + \cdots + a_sy_s + b$ is 0, in order to encode the inequality into the output specification it suffices to encode that $\tilde{y} = \mathbf{0}$, which is a formula expressible in QF_BV2$_{bw}$.

By doing this for each linear inequality in the specification and since the logical operations are allowed by QF_BV2$_{bw}$, it follows that we may use QF_BV2$_{bw}$ to encode boolean combinations of linear inequalities over outputs as formal specifications that are to be verified.

Our main result in this section is that, if ψ in eq. (8) is assumed to be a formula in QF_BV2$_{bw}$, then the verification problem for quantized neural networks is PSPACE-hard. Since checking satisfiability of ψ can be done in non-deterministic polynomial time, this means that the additional hardness really comes from the quantized neural networks.

Theorem 1 (Complexity of verification of QNNs)**.** *If the predicate on outputs is assumed to be a formula in* QF_BV2$_{bw}$*, the verification problem for quantized neural networks is* PSPACE*-hard.*

Proof sketch. Here we summarize the key ideas of our proof. For the complete proof, see the technical report.

To prove PSPACE-hardness, we exhibit a reduction from TQBF which is known to be PSPACE-complete (Arora and Barak 2009). TQBF is the problem of deciding whether a quantified boolean formula (QBF) of the form $Q_1x_1.\, Q_2x_2.\, \ldots\, Q_nx_n.\, \phi(x_1, x_2, \ldots, x_n)$ is true, where each $Q_i \in \{\exists, \forall\}$ and ϕ is a quantifier-free formula in propositional logic over the variables $x_1, \ldots, x_n$. A QBF formula

is true if it admits a truth table for each existentially quantified variable x_i, where the truth table for x_i specifies a value in $\{0, 1\}$ for each valuation of those universally quantified variables x_j on which x_i depends (i.e. x_j with $j < i$). Thus, the size of each truth table is at most 2^k, where k is the total number of universally quantified variables in the formula.

In our reduction, given an instance of the TQBF problem $Q_1 x_1 . Q_2 x_2 . \ldots Q_n x_n . \phi(x_1, x_2, \ldots, x_n)$ we map it to the corresponding verification problem as follows. The interpretation $[\![f]\!]_{\text{int-}k}$ of the neural network f consists of $n+1$ disjoint gadgets $f_1, \ldots, f_n, g$, each having a single input and a single output neuron of bit-width 2^k. Note that bit-widths are given in binary representation, thus this is still polynomial in the size of the problem. We use these gadgets to encode all possible inputs to the QBF formula, whereas the postcondition in the verification problem encodes the quantifier-free formula itself. For a universally quantified variable x_i, the output of f_i is always a constant vector encoding the values of x_i in each of the 2^k valuations of universally quantified variables (for a fixed ordering of the valuations). For existentially quantified x_i, we use f_i and its input neuron to encode 2^k possible choices for the value of x_i, one for each valuation of universally quantified variables, and thus to encode the truth table for x_i. Finally, the gadget g is used to return a constant bit-vector $\mathbf{1}$ of bit-width 2^k on any possible input. The predicate ψ on the outputs is then defined as

$$\psi := (\phi_{bw}(y_1, \ldots, y_n) = \mathbf{1}),$$

where ϕ_{bw} is the quantifier-free formula in QF_BV2$_{bw}$ identical to ϕ, with only difference being that the inputs of ϕ_{bw} are bit-vectors of bit-width 2^k instead of boolean variables, and logical operations are also defined over bit-vectors (again, since bit-widths are encoded in binary representation, this is of polynomial size). The output of ϕ_{bw} is thus tested if it equals 1 for each valuation of universally quantified variables and the corresponding values of existentially quantified variables defined by the truth tables. Our construction ensures that any satisfying input for the neural networks induces satisfying truth tables for the TQBF instance and vice-versa, which completes the reduction. □

Theorem 1 is to our best knowledge the first theoretical result which indicates that the verification problem for quantized neural networks is harder than verifying their idealized real arithmetic counterparts. It sheds some light on the scalability gap of existing SMT-based methods for their verification, and shows that this gap is not solely due to practical inefficiency of existing methods for quantized neural networks, but also due to the fact that the problem is computationally harder. While Theorem 1 gives a lower bound on the hardness of verifying quantized neural networks, it is easy to see that an upper bound on the complexity of this problem is NEXP since the inputs to the verification problem are of size that is exponential in the size of the problem. Closing the gap and identifying tight complexity bounds is an interesting direction of future work.

Note though that the specification logic QF_BV2$_{bw}$ used to encode predicates over outputs is strictly more expressive than what we need to express boolean combinations of linear integer inequalities, which is the most common form of formal specifications seen in practice. This is because QF_BV2$_{bw}$ also allows logical operations over bit vectors, and not just over single bits. Nevertheless, our result presents the first step towards understanding computational hardness of the quantized neural network verification problem.

Improvements to Bit-Vector SMT-Encodings

In this section, we study efficient SMT-encodings of quantized neural networks that would improve scalability of verification methods for them. In particular, we propose three simplifications to the monolithic SMT encoding of eq. (3), (4), and (5) introduced in (Giacobbe, Henzinger, and Lechner 2020), which encodes quantized neural networks and formal specifications as formulas in the QF_BV2 logic : I) Remove dead branches of the If-Then-Else encoding of the activation function in eq. (5), i.e., branches that are guaranteed to never be taken; II) Allocate only the minimal number of bits for each bit-vector variable in the formula; and III) Eliminate sub-expressions from the summation in eq. (3). To obtain the information needed by the techniques I and II we further propose an abstract interpretation framework for quantized neural networks.

Abstract Interpretation Analysis

Abstract interpretation (Cousot and Cousot 1977) is a technique for constructing over-approximations to the behavior of a system. Initially developed for software verification, the method has recently been adapted to robustness verification of neural networks and is used to over-approximate the output range of variables in the network. Instead of considering all possible subsets of real numbers, it only considers an abstract domain which consists of subsets of suitable form (e.g. intervals, boxes or polyhedra). This allows modeling each operation in the network in terms of operations over the elements of the abstract domain, thus over-approximating the semantics of the network. While it leads to some impreision, abstract interpretation allows more efficient output range analysis for variables. Due to its over-approximating nature, it remains sound for verifying neural networks.

Interval (Wang et al. 2018b; Tjeng, Xiao, and Tedrake 2019), zonotope (Mirman, Gehr, and Vechev 2018; Singh et al. 2018), and convex polytope (Katz et al. 2017; Ehlers 2017; Bunel et al. 2018; Wang et al. 2018a) abstractions have emerged in literature as efficient and yet precise choices for the abstract domains of real-valued neural networks. The obtained abstract domains have been used for output range analysis (Wang et al. 2018b), as well as removing decision points from the search process of complete verification algorithms (Tjeng, Xiao, and Tedrake 2019; Katz et al. 2017). One important difference between standard and quantized networks is the use of double-sided bounded activation functions in quantized neural networks, i.e., ReLU-N instead of ReLU (Jacob et al. 2018). This additional non-linear transition, on one hand, renders linear abstractions less effective, while on the other hand it provides hard upper bounds to each neuron, which bounds the over-approximation error. Consequently, we adopt interval abstractions (IA) on the quantized interpretation of a network

to obtain reachability sets for each neuron in the network. As discussed in (Tjeng, Xiao, and Tedrake 2019), using a tighter abstract interpretation poses a tradeoff between verification and pre-processing complexity.

Dead Branch Removal

Suppose that through our abstract interpretation we obtained an interval $[lb, ub]$ for the input x of a ReLU-N operation $y = \text{ReLU-N}(x)$. Then, we substitute the function by

$$\begin{cases} 0, & \text{if } ub \leq 0 \\ 2^N - 1, & \text{if } lb \geq 2^N - 1 \\ x, & \text{if } ub \geq 0 \text{ and } lb \leq 2^N - 1 \\ \max\{0, x\}, & \text{if } 0 < ub \leq 2^N - 1. \\ \min\{2^N - 1, x\}, & \text{if } 0 \leq lb < 2^N - 1. \\ \max\{0, \min\{2^N - 1, x\}\}, & \text{otherwise,} \end{cases}$$

which reduces the number of decision points in the SMT formula.

Minimum Bit Allocation

A k-bit quantized neural network represents each neuron and weight variable by a k-bit integer. However, when computing the values of certain types of layers, such as the linear layer in eq. (1), a wider register is necessary. The binary multiplication of a k-bit weight and a k-bit neuron value results in a number that is represented by $2k$-bits. Furthermore, summing up n such $2k$-bit integer requires

$$b_{\text{naive}} = 2k + \log_2(n) + 1 \tag{9}$$

bits to be safely represented without resulting in an overflow.

QF_BV2 reasons over fixed-size bit-vectors, i.e. the bit width of each variable must be fixed in the formula regardless of the variable's value. (Giacobbe, Henzinger, and Lechner 2020) showed that the number of bits used for all weight and neuron variables in the formal affects the runtime of the SMT-solver significantly. For example, omitting the least significant bit of each variable cuts the runtime on average by half. However, the SMT encoding of (Giacobbe, Henzinger, and Lechner 2020) allocates b_{naive} bits according to eq. (9) for each accumulation variable of a linear layer.

Our approach uses the interval $[lb, ub]$ obtained for each variable by abstract interpretation to compute the minimal number of bits necessary to express any value in the interval. As the signed bit-vector variables are represented in the two's complement format, we can compute the bit width b of variable x with computed interval $[lb, ub]$ by

$$b_{\text{minimal}} = 1 + \log_2(\max\{|lb|, |ub|\} + 1). \tag{10}$$

Trivially, one can show that $b_{\text{minimal}} < b_{\text{naive}}$, as $|ub| \leq 2^{2k}n$ and $|lb| \leq 2^{2k}n$.

Redundant Multiplication Elimination

Another difference between quantized and standard neural networks is the rounding of the weight values to the nearest representable value of the employed fixed-point format. Consequently, there is a considerable chance that two connections outgoing from the same source neuron will have the same weight value. For instance, assuming an 8-bit network and a uniform weight distribution, the chance of two connections having the same weight value is around 0.4% compared to the much lower $4 \cdot 10^{-8}\%$ of the same scenario happening in a floating-point network.

Moreover, many weight values express some subtle form of redundancy on a bit-level. For instance, both multiplication by 2 and multiplication by 6 contain a shift operations by 1 digit in their binary representation. Thus, computations

$$y_1 = 3 \cdot x_1 \qquad y_2 = 6 \cdot x_1 \tag{11}$$

can be rewritten as

$$y_1 = 3 \cdot x_1 \qquad y_2 = y_1 << 1, \tag{12}$$

where $<<$ is a binary shift to the left by 1 digit. As a result, a multiplication by 6 is replaced by a much simpler shift operation. Based on this motivation, we propose a redundancy elimination heuristic to remove redundant and partially redundant multiplications from the SMT formula. Our heuristic first orders all outgoing weights of a neuron in ascending order and then sequentially applies a rule-matching for each weight value. The rules try to find a simpler way to compute the multiplication of the weight and the neuron value by using already performed multiplications. The algorithm and the rules in full are provided in the technical report.

Note that a similar idea was introduced by (Cheng et al. 2018) in the form of a neuron factoring algorithm for the encoding of binarized (1-bit) neural networks into SAT formulas. However, the heuristic of (Cheng et al. 2018) removes redundant additions, whereas we consider bit-level redundancies in multiplications. For many-bit quantization, the probability of two neurons sharing more than one incoming weight is negligible, thus making such neuron factoring proposed in (Cheng et al. 2018) less effective.

Experimental Evaluation

We create an experimental setup to evaluate how much the proposed techniques affect the runtime and efficiency of the SMT-solver. Our reference baseline is the approach of (Giacobbe, Henzinger, and Lechner 2020), which consists of a monolithic and "balanced" bit-vector formulation for the Boolector SMT-solver. We implement our techniques on top of this baseline. We limited our evaluation to Boolector, as other SMT-solvers supporting bit-vector theories, such as Z3 (De Moura and Bjørner 2008), CVC4 (Barrett et al. 2011), and Yices (Dutertre 2014), performed much worse in the evaluation of (Giacobbe, Henzinger, and Lechner 2020).

Our evaluation comprises of two benchmarks. Our first evaluation considers the adversarial robustness verification of image classifier trained on the MNIST dataset (LeCun et al. 1998). In particular, we check the l_∞ robustness of networks against adversarial attacks (Szegedy et al. 2013). Other norms, such as l_1 and l_2, can be expressed in bit-vector SMT constraints as well, although with potentially negative effects on the solver runtime. In the second evaluation, we repeat the experiment on the slightly more complex Fashion-MNIST dataset (Xiao, Rasul, and Vollgraf 2017) .

All experiments are run on a 14-core Intel W-2175 CPU with 64GB of memory. We used the boolector

Attack radius	Baseline (+ Lingeling)	Baseline (+ CaDiCal)	Ours
$\varepsilon = 1$	63 (63.6%)	92 (92.9%)	**99 (100.0%)**
$\varepsilon = 2$	0 (0.0%)	20 (20.2%)	**94 (94.9%)**
$\varepsilon = 3$	0 (0.0%)	2 (2.1%)	**71 (74.0%)**
$\varepsilon = 4$	0 (0.0%)	1 (1.0%)	**54 (55.7%)**

Table 1: Number of solved instances of adversarial robustness verification on the MNIST dataset. Absolute numbers and in percentages of checked instances in parenthesis.

Dataset	Baseline (+ Lingeling)	Baseline (+ CaDiCal)	Ours
MNIST	8803 |8789	2798 |3931	**5 |90**
Fashion-MNIST	6927 |6927	3105 |3474	**4 |49**

Table 2: Median |mean runtime of adversarial robustness verification process per sample. The reported values only account for non-timed-out samples.

(Niemetz, Preiner, and Biere 2015) with the SAT-solvers Lingeling(Biere 2017) (only baseline) and CaDiCal (Biere 2019) (baseline + our improvements) as SAT-backend.

Adversarial robustness specification can be expressed as

$$|x - x_i|_\infty \leq \varepsilon \wedge y = [\![f]\!]_{\text{int-}k}(x) \implies y = y_i, \qquad (13)$$

where (x_i, y_i) is a human labeled test sample and ε is a fixed attack radius. As shown in eq. (13), the space of possible attacks increases with ε. Consequently, we evaluate with different attack radii ε and study the runtimes individually. In particular, for MNIST we check the first 100 test samples with an attack radius of $\varepsilon = 1$, the next 100 test samples with $\varepsilon = 2$, and the next 200 test samples with $\varepsilon = 3$ and $\varepsilon = 4$ respectively. For our Fashion-MNIST evaluation, we reduce the number of samples to 50 per attack radius value for $\varepsilon > 2$ due to time and compute limitations.

The network studied in our benchmark consists of four fully-connected layers (784,64,32,10), resulting in 52,650 parameters in total. It was trained using a quantization-aware training scheme with a 6-bit quantization.

The results for the MNIST evaluation in terms of solved instances and median solver runtime are shown in Table 1 and Table 2 respectively. Table 3 and Table 2 show the results for the Fashion-MNIST benchmark. Both benchmark networks are publicly available [3]

Ablation Analysis

We perform an ablation analysis where we re-run our robustness evaluation with one of our proposed techniques disabled. The objective of our ablation analysis is to understand how the individual techniques affect the observed efficiency gains. Due to time and computational limitations we focus our ablation experiments to MNIST exclusively.

The results in Table 4 show the highest number of solved instances were achieved when all our techniques were enabled. Nonetheless, Table 4 demonstrate these gains are not

[3]https://github.com/mlech26l/qnn_robustness_benchmarks

Attack radius	Baseline (+ Lingeling)	Baseline (+ CaDiCal)	Ours
$\varepsilon = 1$	2 (2.3%)	44 (50.6%)	**76 (87.4%)**
$\varepsilon = 2$	0 (0.0%)	7 (7.8%)	**73 (81.1%)**
$\varepsilon = 3$	0 (0.0%)	1 (2.3%)	**27 (62.8%)**
$\varepsilon = 4$	0 (0.0%)	0 (0.0%)	**18 (40.9%)**

Table 3: Number of solved instances of adversarial robustness verification on the Fashion-MNIST dataset.

Method	Total solved instances	Cumulative runtime
No redundancy elim.	316 (80.8%)	7.7 h
No minimum bitwidth	315 (80.6%)	5.1 h
No ReLU simplify	88 (22.5%)	83.2 h
No Abstract interpret.	107 (27.4%)	126.0 h
All enabled	**318 (81.3%)**	7.9 h

Table 4: Ablation analysis on the MNIST dataset. The cumulative runtime corresponds to non-timed-out samples.

equally distributed across the three techniques. In particular, the ReLU simplification has a much higher contribution for explaining the gains compared to the redundancy elimination and minimum bitwidth methods. The limited benefits observed for these two techniques may be explain by the inner workings of the Boolector SMT-solver.

The Boolector SMT-solver (Niemetz, Preiner, and Biere 2015) is based on a portfolio approach which sequentially applies several different heuristics to find a satisfying assignment of the input formula (Wintersteiger, Hamadi, and De Moura 2009). In particular, Boolector starts with fast but incomplete local search heuristics and falls back to slower but complete bit-blasting (Clark and Cesare 2018) in case the incomplete search is unsuccessful (Niemetz, Preiner, and Biere 2019). Although our redundancy elimination and minimum bitwidth techniques simplify the bit-blasted representation of the encoding, it introduces additional dependencies between different bit-vector variables. As a result, we believe these extra dependencies make the local search heuristics of Boolector less effective and thus enabling only limited performance improvements.

Conclusion

We show that the problem of verifying quantized neural networks with bit-vector specifications on the inputs and outputs of the network is PSPACE-hard. We tackle this challenging problem by proposing three techniques to make the SMT-based verification of quantized networks more efficient. Our experiments show that our method outperforms existing tools by several orders of magnitude on adversarial robustness verification instances. Future work is necessary to explore quantized neural network verification's complexity with respect to different specification logics. On the practical side, our methods point to limitations of monolithic SMT-encodings for quantized neural network verification and suggest that future improvements may be obtained by integrating the encoding and the solver steps more tightly.

Acknowledgments

This research was supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award), ERC CoG 863818 (FoRM-SMArt), and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 665385.

References

Arora, S.; and Barak, B. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press. ISBN 978-0-521-42426-4. URL http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

Baranowski, M.; He, S.; Lechner, M.; Nguyen, T. S.; and Rakamaric, Z. 2020. An SMT Theory of Fixed-Point Arithmetic. In *International Joint Conference on Automated Reasoning (IJCAR)*, 13–31.

Barrett, C.; Conway, C. L.; Deters, M.; Hadarean, L.; Jovanović, D.; King, T.; Reynolds, A.; and Tinelli, C. 2011. CVC4. In *International Conference on Computer Aided Verification (CAV)*, 171–177. Springer.

Biere, A. 2017. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Balyo, T.; Heule, M.; and Järvisalo, M., eds., *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, 14–15. University of Helsinki.

Biere, A. 2019. CaDiCaL at the SAT Race 2019. In Heule, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, 8–9. University of Helsinki.

Bunel, R. R.; Turkaslan, I.; Torr, P.; Kohli, P.; and Mudigonda, P. K. 2018. A unified view of piecewise linear neural network verification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 4795–4804.

Cheng, C.-H.; Nührenberg, G.; Huang, C.-H.; and Ruess, H. 2018. Verification of Binarized Neural Networks via Inter-Neuron Factoring. In *Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)*, 279–290.

Clark, B.; and Cesare, T. 2018. Satisfiability Modulo Theories. In Clarke, E. M.; Henzinger, T. A.; Veith, H.; and Bloem, R., eds., *Handbook of model checking*, volume 10, chapter 11. Springer.

Cousot, P.; and Cousot, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL)*, 238–252.

De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

Dutertre, B. 2014. Yices 2.2. In *International Conference on Computer Aided Verification (CAV)*, 737–744. Springer.

Dutta, S.; Chen, X.; and Sankaranarayanan, S. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 157–168.

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

Giacobbe, M.; Henzinger, T. A.; and Lechner, M. 2020. How Many Bits Does it Take to Quantize Your Neural Network? In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 79–97.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

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

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2704–2713.

Jia, K.; and Rinard, M. 2020. Exploiting Verified Neural Networks via Floating Point Numerical Error. *arXiv preprint arXiv:2003.03021* .

Kahan, W. 1996. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE* 754(94720-1776): 11.

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

Kovásznai, G.; Fröhlich, A.; and Biere, A. 2016. Complexity of fixed-size bit-vector logics. *Theory of Computing Systems* 59(2): 323–376.

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

Mirman, M.; Gehr, T.; and Vechev, M. 2018. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, 3575–3583.

Narodytska, N.; Kasiviswanathan, S.; Ryzhyk, L.; Sagiv, M.; and Walsh, T. 2018. Verifying properties of binarized deep neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 6615–6624.

Niemetz, A.; Preiner, M.; and Biere, A. 2015. Boolector 2.0 system description. *Journal on Satisfiability, Boolean Modeling and Computation* 9: 53–58.

Niemetz, A.; Preiner, M.; and Biere, A. 2019. Boolector at the SMT Competition 2019. Technical report, Stanford University and JKU Linz.

Ruan, W.; Huang, X.; and Kwiatkowska, M. 2018. Reachability analysis of deep neural networks with provable guarantees. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2651–2659.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. 2018. Fast and effective robustness certification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 10825–10836.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3(POPL): 1–30.

Smith, S. W.; et al. 1997. *The scientist and engineer's guide to digital signal processing*, volume 14. California Technical Pub. San Diego.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* .

Tan, M.; and Le, Q. V. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 6105–6114.

Tjeng, V.; Xiao, K. Y.; and Tedrake, R. 2019. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient formal safety analysis of neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 6369–6379.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018b. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 1599–1614.

Wintersteiger, C. M.; Hamadi, Y.; and De Moura, L. 2009. A concurrent portfolio approach to SMT solving. In *International Conference on Computer Aided Verification (CAV)*, 715–720. Springer.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* .