Abstraction-Based Proof Production in Formal Verification of Neural Networks (Extended Abstract)

Yizhak Yisrael Elboher[1], Omri Isac[1], Guy Katz[1], Tobias Ladner[2], Haoze Wu[3]

[1] The Hebrew University of Jerusalem, Israel
[2] Technical University of Munich, Germany
[3] Amherst College, USA

Abstract. Modern verification tools for deep neural networks (DNNs) increasingly rely on abstraction to scale to realistic architectures. In parallel, proof production is becoming a critical requirement for increasing the reliability of DNN verification results. However, current proof-producing verifiers do not support abstraction-based reasoning, creating a gap between scalability and provable guarantees. We address this gap by introducing a novel framework for proof-producing abstraction-based DNN verification. Our approach modularly separates the verification task into two components: proving the property on an abstract network, and proving the soundness of the abstraction with respect to the original DNN. The former can be handled by existing proof-producing verifiers, whereas we propose the first method for generating formal proofs for the latter. This preliminary work aims to enable scalable and trustworthy verification by supporting common abstraction techniques within a formal proof framework.

Keywords: Neural Networks, Formal Verification, Proof Production, Abstraction

1 Introduction

Deep Neural Networks (DNNs) [17, 30] have demonstrated exceptional performance in various domains, including vision [27], language [46], audio [45] and video [2] analysis, achieving state-of-the-art accuracy in complex tasks [39, 40]. However, despite their success, DNNs function as black-box models, making their decision-making processes difficult to interpret and trust [31, 41].

DNN verification [12,24,32] provides formal methods and tools (*verifiers*), to ensure or refute that DNNs comply with required specifications, offering formal guarantees of correctness. However, although verification algorithms are theoretically sound, their implementation can introduce potential issues [15, 23], compromising their soundness and undermining the confidence in the verifier.

A notable approach to tackle these issues is by producing formal *proofs*, i.e., mathematical objects that can be checked by an independent program and witness the verifier's correctness. Proof production was explored in SMT and SAT solvers [6, 18], and recently also in DNN verification [21, 43]. Although proofs enhance the reliability of the verification process, their generation limits

the scalability of the verifier in two ways: (i) the generated proofs tend to be large, which substantially increases the verifier's memory consumption; and (ii) some verifier optimization are not supported by the proof mechanism, and are disabled whenever proof generation is used — slowing down the verifier.

Scalability is also an intrinsic challenge in the core of formal verification, since a large, sometimes infinite, set of states should be scanned to formally guarantee or refute correctness. In particular, DNN verification is an NP-complete problem [24, 42], and modern solvers might solve verification queries in worst-case exponential time with respect to DNN size (number of neurons) [7]. A common attempt to overcome this obstacle is to apply *abstraction*. This well-established technique in formal verification [9–11] is used to manage the complexity of analyzing large systems by creating a simpler, abstract model that retains the essential properties of the original system. In the context of DNNs, abstraction has gained attention as a method to enhance verification scalability and efficiency [3, 13, 29]. Specifically, DNN abstraction involves the construction of a reduced or approximate representation of the network such that the verification of the abstract network provides meaningful guarantees for the original network. Using abstraction, verification tools can handle larger networks and more complex properties, making it a promising approach for scalable and efficient formal analysis of DNNs.

This work addresses two key challenges in DNN verification: enabling proof production for abstraction-based solvers and generating more compact proofs. While abstraction improves scalability by simplifying the network, existing proof-producing tools do not support it. To bridge this gap, we propose the notion of an *abstract proof*—a modular proof consisting of (1) a proof that the abstract network satisfies the query, and (2) a proof that the abstraction over-approximates the original network.

This approach extends proof support to scalable abstraction-based solvers and reduces proof size. Fig. 1 illustrates the improved proof workflow and expected efficiency gains compared to the standard approach.

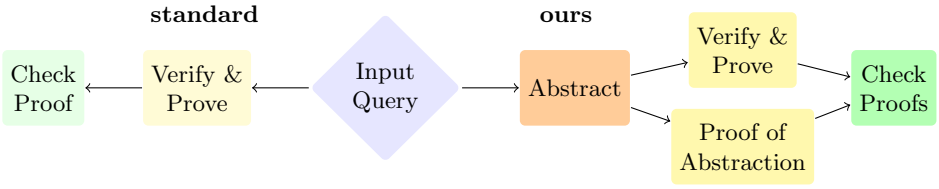


Fig. 1: Proof production flowchart: standard (left) versus ours (right). Bold colors represent cheaper operations.

Inspired by CEGAR [9], our main contributions are:

1. Introduction of the concept of an abstract proof for DNN verification.
2. Design of an abstraction-refinement mechanism for proof production.
3. Formalization of a verifiable proof of the abstraction process itself, using the Marabou DNN verifier [47] and the CORA abstraction engine [1].

The paper is organized as follows: Sec. 2 provides background on DNNs, DNN verification and abstraction, and proof production. Sec. 3 introduces our

modular framework for constructing abstract proofs, describes the challenge of aligning proofs between the original and abstract networks, and presents a general abstraction-refinement algorithm for efficient proof production. Sec. 4 explains the abstraction process in CORA, and adapts it to our formulation. Sec. 5 details our implementation using Marabou (for proof production) and CORA (for abstraction), including how to verify abstract networks and generate corresponding proofs. Sec. 6 reviews relevant literature, and Sec. 7 summarizes our contributions and outlines future work.

2 Preliminaries

2.1 Deep Neural Networks (DNNs)

A *Deep Neural Network (DNN)* is a parameterized function $f\colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$, composed of multiple layers of interconnected neurons. Each layer performs an affine transformation followed by a nonlinear activation function. Formally, given an input $\mathbf{x} \in \mathbb{R}^{n_0}$, the output $\mathbf{y} = f(\mathbf{x})$ is computed as follows:

$$\mathbf{h}_0 = \mathbf{x}, \qquad \mathbf{h}_k = \phi_k(\mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k), \qquad \mathbf{y} = \mathbf{h}_L, \qquad k \in [L]. \tag{1}$$

where $\mathbf{W}_k$ is the weight matrix, $\mathbf{b}_k$ is the bias vector, and ϕ_k is the nonlinear activation function (e.g., ReLU [36] or sigmoid) for the k-th layer. An illustration for a neural network is given in Fig. 2.

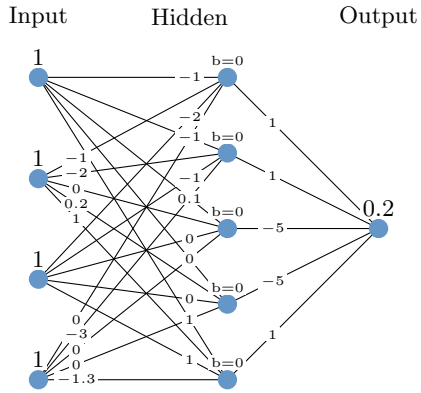


Fig. 2: A neural network f_1 for which $f_1(1,1,1,1) = 0.2$. All biases are 0.

2.2 DNN Verification

DNN Verification aims to solve verification queries. A *verification query* is a triplet $\langle f, \mathcal{P}, \mathcal{Q} \rangle$ where $f\colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ is a DNN, $\mathcal{P} \subset \mathbb{R}^{n_0}$ is an input property and $\mathcal{Q} \subset \mathbb{R}^{n_L}$ is an output property. The *DNN Verification Problem* is the problem of deciding whether there exists an input satisfying $\mathcal{P}$ for which its output of f satisfies $\mathcal{Q}$:

$$\exists \mathbf{x}.\ \mathbf{x} \in \mathcal{P} \wedge f(\mathbf{x}) \in \mathcal{Q}. \tag{2}$$

Typically, $\mathcal{Q}$ is a set that characterizes an undesired behavior, such as vulnerability of f to adversarial perturbations or danger conditions. If an input $\mathbf{x} \in \mathcal{P}$ is found whose output $f(\mathbf{x}) \in \mathcal{Q}$, we say the query is `SAT`, and $\mathbf{x}$ serves as a counterexample to the desired property. Otherwise, if no such input exists, we say the query is `UNSAT`, and thus the desired property is valid. To ease notation, we also denote the latter case as $\texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q} \rangle)$. This can be captured as follows:

$$\texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q} \rangle) \equiv \forall \mathbf{x}.\ \mathbf{x} \in \mathcal{P} \implies f(\mathbf{x}) \notin \mathcal{Q}. \tag{3}$$

For simplicity, we assume for this work that $\mathcal{P}$ is a hyperrectangle, although our approach can be generalized to any closed set $\mathcal{P}$, e.g., by over-approximating $\mathcal{P}$ with a bounding box thereof. An example of a verification query is $\langle f_1, \mathcal{P}_1, \mathcal{Q}_1 \rangle$ where f_1 is the network in Fig. 2, $\mathcal{P}_1 = \{(1 \pm \epsilon, 1 \pm \epsilon, 1 \pm \epsilon, 1) \mid \epsilon \in [0, 0.1]\}$ and $\mathcal{Q}_1 = \mathbb{R}_{\leq 0}$.

2.3 DNN Abstraction for Formal Verification

To accelerate DNN verification, it is often beneficial to reduce the size of the network through abstraction. However, since the verification target is the original DNN, one must ensure that any conclusions drawn from the abstract (simplified) model also apply to the original. The over-approximation requirement is represented as follows, where $f, \widehat{f}$ are the original and abstract networks, respectively:

$$\texttt{unsat}(\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle) \implies \texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q} \rangle).$$

If the abstraction is too coarse to yield a conclusive result, the abstract model is iteratively *refined* – made more precise over-approximation – until the verification query can be resolved correctly.

Our framework is designed to be compatible with a wide range of abstraction methods. In this work, we focus on CORA [1], a MATLAB toolbox used for formal verification of neural networks via reachability analysis. With its recent abstraction-refinement extension [29], CORA improves performance by replacing the original network with a smaller abstract model that is iteratively refined as needed. We leverage CORA as a backend for abstraction in our framework. Other approaches to network abstraction are discussed in Sec. 6.

2.4 Proof Production for DNN Verification

As a satisfiability problem, proving a `SAT` DNN verification query is straightforward, using a satisfying assignment that could be checked by evaluation over the network. Proving `UNSAT`, however, is more complicated due to the NP-hardness of the DNN verification problem [25, 42]. Thus, bookkeeping the whole proof may require large memory consumption, even for small DNNs.

In this work, we focus on the proof producing version of Marabou [21, 47], a state of the art DNN verifier, which encodes verification queries as satisfiability problems, utilizing satisfiability modulo theories (SMT) solving and linear programming (LP) to analyze properties of interest. It handles nonlinear activation functions, such as ReLU, through case-splitting and relaxation techniques. Marabou proof of `UNSAT` is represented by a *proof-tree*. By construction, its size heavily relies on the number of splits performed by Marabou, which could be exponentially large in the number of neurons.

3 Method

We intend to accelerate verification by applying abstraction to reduce the DNN size and prove the property over the abstract DNN. However, a proof over the abstract network alone is insufficient – it does not guarantee that the property holds for the original network. To overcome this, we introduce a general framework for constructing end-to-end proofs that remain sound while leveraging abstraction.

3.1 Proving Abstraction-Based DNN Verification

Since verifying DNNs using abstraction consists of two main components — constructing the abstraction and verifying the abstract network — our proof method for UNSAT follows the same modular structure. This modularity ensures that our approach remains agnostic to the underlying DNN verifier and abstraction technique, making it broadly applicable. Specifically, it enables combining any DNN verification tool capable of producing proofs with any abstraction method that comes with a corresponding proof rule.

Our method constructs a proof that consists of two independent parts. First, given candidate DNNs $f, \widehat{f}$ and properties $\mathcal{P}, \mathcal{Q}$, we establish that if $\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle$ is UNSAT, then so is $\langle f, \mathcal{P}, \mathcal{Q} \rangle$. This forms the *proof of over-approximation*, i.e., proof of abstraction correctness, which ensures that verification results transfer from the abstract network to the original one. The second part is the verification proof for the abstract network, i.e., the proof that $\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle$ is indeed UNSAT. These two proofs, when combined, yield the *abstract proof* following the proof rule in Fig. 3. Even though this rule is a private case of implication elimination (modus ponens), we define it to clearly indicate our modular approach.

$$\mathsf{abs-proof}: \quad \frac{\texttt{unsat}(\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle) \implies \texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q} \rangle) \quad \texttt{unsat}(\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle)}{\texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q} \rangle)}$$

Fig. 3: Proof rule for proving DNN verification with abstraction

As a proof system for verifying DNNs (i.e., the top right part of the rule) has been introduced in prior work [21], our focus in this paper is twofold: (1) showing how $\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle$ can be reduced to a verification query for DNNs; and (2) constructing the proof of over-approximation (i.e., the top left part). We exemplify how both are done in Sec. 5.1 and in Sec. 5.2, respectively.

3.2 Main Algorithm

The naive approach to improve proof production through abstraction-refinement (described in Appendix A, Alg. 2) abstracts and iteratively refines the model and verifies the query with proof production. This approach has a classical drawback: Like any abstraction refinement procedure, the operation performed during abstraction might be redundant in the case of spurious SAT. Moreover, each doubtful proof generation attempt is expensive.

We suggest Alg. 1 to overcome these problems. Unless UNSAT is accepted, correctness is just verified but is not proved. In addition to avoiding redundant

proof attempts and improving the performance in each iteration, another advantage is that any verifier can be applied to verify the property, making the procedure more modular, not limited to verifiers with proof production capabilities or specific limited configuration that support proof production.

The methods `prove-over-approximation` and `verify-with-proofs` represent proving the over-approximation and producing proof in the query on the abstract network, and are explained in more detail in Sec. 5. $\langle p_a, p_q \rangle$ is a concatenation of the two proofs into a full proof.

Algorithm 1 Proof Production with Abstraction

Input: f, $\mathcal{P}$, $\mathcal{Q}$. **Output:** proof that $\texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q} \rangle)$, or counterexample.

$\widehat{f} = \texttt{abstract}(f, \mathcal{P})$
while true **do**
  result, example = $\texttt{verify}(\widehat{f}, \mathcal{P}, \mathcal{Q})$
  if result == `SAT` and example is not spurious **then**
    return result, example
  else if result == `UNSAT` **then**
    $p_a = \texttt{prove-over-approximation}(\widehat{f}, f, \mathcal{P}, \mathcal{Q})$
    $p_q = \texttt{verify-with-proofs}(\widehat{f}, \mathcal{P}, \mathcal{Q})$
    if p_a and p_q were successfully generated **then**
      return `UNSAT`, $\langle p_a, p_q \rangle$
    else
      $\widehat{f} = \texttt{refine}(\widehat{f}, f)$
    end if
  end if
end while

4 Abstraction in CORA

We provide an overview on how abstraction in CORA works and how it can be integrated into the verification process. We refer the reader to [1, 29] for additional details.

Given a neural network f as in (1) and an input set $\mathcal{P}$, the exact output set $\mathcal{Y}^* = f(\mathcal{P})$ is computed by

$$\mathcal{H}_0^* = \mathcal{P}, \qquad \mathcal{H}_k^* = \phi_k(\mathbf{W}_k \mathcal{H}_{k-1}^* + \mathbf{b}_k), \qquad \mathcal{Y}^* = \mathcal{H}_L^*, \qquad k \in [L]. \tag{4}$$

These exact sets are generally expensive to compute [24]. Thus, we enclose the output of each layer $\mathcal{H}_k \supseteq \mathcal{H}_k^*$. In this work, we only consider the set $\mathcal{H}_k$ to be represented as interval bounds, although more sophisticated set representations exist [4, 16, 26, 28, 35].

Since DNNs usually contain a large number of neurons per layer, their verification can be computationally expensive as well. Thus, [29] suggests a construction of an abstract network that soundly merges neurons with similar bounds to reduce the network size, which in turn decreases the verification time by decreasing the computation time. The bounds are determined by a one-step look-ahead algorithm using interval bound propagation (IBP). In particular, we compute

the output interval bounds of layer k as follows [29, Alg. 2]:

$$\mathcal{I}_k = \phi_k(\mathbf{W}_k \cdot \texttt{bounds}(\mathcal{H}_{k-1})) + \mathbf{b}_k) \supseteq \mathcal{H}_k, \tag{5}$$

where $\texttt{bounds}(\cdot)$ computes the interval bounds of the given set $\mathcal{H}_{k-1}$. In order to preserve soundness, multiple neurons with similar bounds are merged and the resulting error is bounded by adapting the bias term in the next layer, converting them from scalars into intervals. These bias intervals bound the deviation between the abstract network and the original network of each layer in the network and, thus, also the output of both networks.

More formally, given a neural network, [29, Prop. 4] defines a way to merge the neurons in the k-th layer, constructing the weights and biases such that the output of the $k+1$-th layer of the original neural network is contained in the output of the $k+1$-th layer of the abstract network. Notice that the terminology in [29] splits each layer into two layers, namely the linear layer and the nonlinear layer, and indexes them separately. Here, we similarly treat each layer as having two parts, but do not handle these as different layers.

Proposition 1 (Neuron Merging [29, Prop. 4]). *Given a nonlinear hidden layer $k \in [L-1]$ of a network f with n_k neurons, output interval bounds $\mathcal{I}_k \supseteq \mathcal{H}_k^*$, a merge bucket $\mathcal{B} \subset [n_k]$ containing the indices of the merged neurons, and $\bar{\mathcal{B}} = [n_k] \setminus \mathcal{B}$, we can construct an abstract network $\widehat{f}$, where we remove the merged neurons by adjusting the layers k and $k+1$ as follows:*

$$\begin{aligned} \widehat{\mathbf{W}}_k &= \mathbf{W}_{k(\bar{\mathcal{B}},\cdot)}, & \widehat{\mathbf{b}}_k &= \mathbf{b}_{k(\bar{\mathcal{B}})}, & \widehat{\mathcal{I}}_k &= \widehat{\mathcal{I}}_{k(\bar{\mathcal{B}})}, \\ \widehat{\mathbf{W}}_{k+1} &= \mathbf{W}_{k+1(\cdot,\bar{\mathcal{B}})}, & \widehat{\mathbf{b}}_{k+1} &= \mathbf{b}_{k+1}, & \widehat{\mathcal{I}}_{k+1} &= \mathbf{W}_{k+1(\cdot,\mathcal{B})}\mathcal{I}_{k(\mathcal{B})}. \end{aligned}$$

where for $\mathcal{S} \in \{\mathcal{B}, \bar{\mathcal{B}}\}$, $\square_{(\mathcal{S},\cdot)}$ and $\square_{(\cdot,\mathcal{S})}$ represent the rows and columns with the indices in $\mathcal{S}$ in lexicographic order, respectively. The interval bounds $\widehat{\mathcal{I}}_k$ require us to extend the formulation for a neural network f as in (1) to an abstract network $\widehat{f}$, as illustrated in Fig. 4.

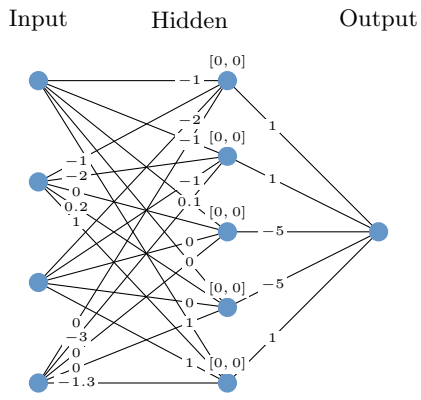


Fig. 4: $\widehat{f}_1$, the extension of f_1 with the new formulation for abstract networks, where the biases of f_1 (zeros) are converted to intervals (singletones).

Given an input $\mathbf{x} \in \mathcal{P}$, the output $\widehat{\mathcal{Y}} = \widehat{f}(\mathbf{x})$ is computed by

$$\widehat{\mathcal{H}}_0 = \{\mathbf{x}\}, \quad \widehat{\mathcal{H}}_k = \phi_k(\widehat{\mathbf{W}}_k \widehat{\mathcal{H}}_{k-1} \oplus \widehat{\mathbf{b}}_k \oplus \widehat{\mathcal{I}}_k), \quad \widehat{\mathcal{Y}} = \widehat{\mathcal{H}}_L, \quad k \in [L]. \tag{6}$$

where all $\widehat{\mathcal{I}}_k$ are initialized with $\{\mathbf{0}\}$, or equivalently $[\mathbf{0}, \mathbf{0}]$, and $\oplus$ denotes the Minkowski sum of two sets, i.e., given $\mathcal{S}_1, \mathcal{S}_2 \subset \mathbb{R}^n$, $\mathcal{S}_1 \oplus \mathcal{S}_2 = \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1,\ s_2 \in \mathcal{S}_1\}$. If either summand of the Minkowski sum is given as a vector, it is implicitly converted to a singleton. The interval biases $\widehat{\mathbf{b}}_k \oplus \widehat{\mathcal{I}}_k$ capture the error between the abstract network and the original network. Thus, initially, it holds:

$$\forall \mathbf{x} \in \mathcal{P} \colon f(\mathbf{x}) \in \widehat{f}(\mathbf{x}) = \{f(\mathbf{x})\}. \tag{7}$$

As an abstract network outputs a set instead of a single vector, we also have to generalize (3) to abstract networks:

$$\texttt{unsat}(\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle) \equiv \forall \mathbf{x}.\ \mathbf{x} \in \mathcal{P} \implies \widehat{f}(\mathbf{x}) \cap \mathcal{Q} = \emptyset. \tag{8}$$

This formulation enables us the following corollary:

Corollary 1. *Given an input set $\mathcal{P}$, an abstract neural network $\widehat{f}$ where Prop. 1 is applied to all layers $k' \leq k \in [L]$, we can merge the neurons in layer k using Prop. 1 such that for the obtained abstract network $\widehat{f}'$, it holds that*

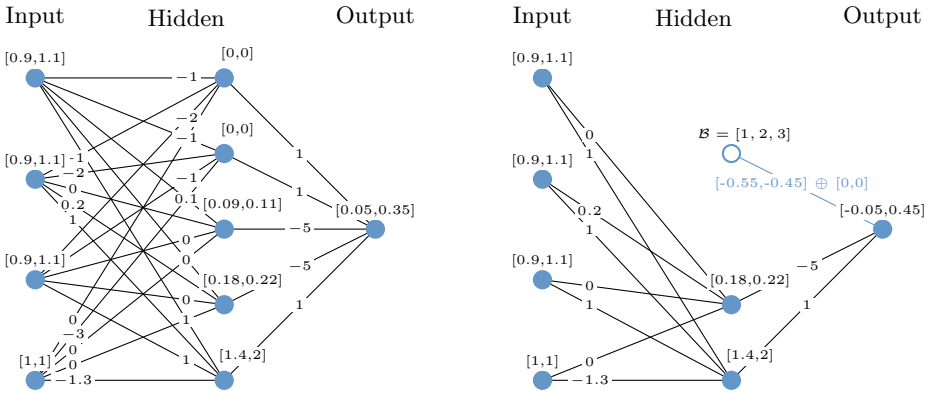
$$\forall \mathbf{x} \in \mathcal{P} \colon \widehat{f}(\mathbf{x}) \subseteq \widehat{f}'(\mathbf{x}).$$

In particular, it holds that

$$\texttt{unsat}(\langle \widehat{f}', \mathcal{P}, \mathcal{Q} \rangle) \implies \texttt{unsat}(\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle).$$

Proof. Let $\widehat{\mathcal{H}}_k$ and $\widehat{\mathcal{H}}'_k$ denote the output of the k-th layer of $\widehat{f}$ and $\widehat{f}'$, respectively. Please note that Prop. 1 only alters layer k and $k+1$, thus, all other layers are identical between $\widehat{f}$ and $\widehat{f}'$ (6). In particular, we know that $\widehat{\mathcal{H}}_{k-1} = \widehat{\mathcal{H}}'_{k-1}$ holds. We now show that $\widehat{\mathcal{H}}_{k+1} \subseteq \widehat{\mathcal{H}}'_{k+1}$ holds by contradiction. Let us assume that there exist a $\bar{\mathbf{h}}_{k-1} \in \widehat{\mathcal{H}}_{k-1}$ for which the respective $\bar{\mathbf{h}}_{k+1} \in \widehat{\mathcal{H}}_{k+1}$ but $\bar{\mathbf{h}}_{k+1} \notin \widehat{\mathcal{H}}'_{k+1}$. However, this cannot be true as the values of the merged neurons are captured by $\widehat{\mathcal{I}}_{k+1}$, which is computed over-approximative using IBP ((5), Prop. 1), and all remaining neurons are kept equal (Prop. 1). Thus, $\widehat{\mathcal{H}}_{k+1} \subseteq \widehat{\mathcal{H}}'_{k+1}$ holds, which directly shows that $\widehat{f}(\mathbf{x}) \subseteq \widehat{f}'(\mathbf{x})$ as all subsequent layers are again identical. The implication directly follows due to the subset relation and (8).

An example of abstraction is shown in Fig. 5. Given the input set $\mathcal{P}_1$, the output bounds of the abstract network $\widehat{f}'_1$ contain the output bounds of the basic abstract network $\widehat{f}_1$. The first three hidden neurons have similar bounds, making them a merge bucket $\mathcal{B} = \{1, 2, 3\}$, and are thus merged into an abstract neuron (in white). The set of bounds of the bucket ([0,0], [0,0] and [[0.09,0.11]]) is embedded into the bias ([0,0]) of the neuron in the output layer using IBP ($1 \cdot [0, 0] + 1 \cdot [0, 0] - 5 \cdot [0.09, 0.11] = [-0.55, -0.45]$) and Minkowski sum ($\oplus$), resulting with output bounds of $[-0.05, 0.45] = -5 \cdot [0.18, 0.22] + 1 \cdot [1.4, 2] + [-0.55, -0.45]$.



(a) Basic abstract network $\widehat{f}_1$ (b) After CORA-abstraction $\widehat{f}_1'$

Fig. 5: Example of abstraction. The basic abstract network $\widehat{f}_1$ (left) is reduced to another abstract network $\widehat{f}_1'$ (right).

5 Proving CORA Abstraction and Marabou Verification

In this work, we focus on the proof-producing version of Marabou [21, 47], a state-of-the-art verification tool with the ability to produce proofs of its `UNSAT` results. The abstraction process used in this work was suggested in [29] and is implemented to improve set-based DNN verification and is part of CORA [1]. In this section, we explain how to implement `verify-with-proofs` and `prove-over-approximation` in Alg. 2 and Alg. 1 with these tools.

We start this section with explaining in 5.1 the details about the verification process in Marabou, and then show how to implement `verify-with-proofs` and apply Marabou on an abstract network obtained by the abstraction process in CORA. In 5.2, we show how to implement `prove-over-approximation` and produce a proof that the abstraction process is correct. By doing so, we accomplish both necessary results as outlined in Sec. 3.1.

5.1 Proving Correctness of Abstract Network Queries in Marabou

The abstract networks obtained by the abstraction process in CORA generalize DNNs. As a result, the verification process should be adapted from solving $\langle f, \mathcal{P}, \mathcal{Q} \rangle$ and proving (3) to solving $\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle$ and proving (8). In the following, we show how the latter can be represented as a query that the Marabou verifier supports. We then implement `verify-with-proofs`, since the proofs generated by Marabou during the verification of $\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle$ can be used as proofs for (8). Recall that Marabou handles verification queries by trying to find satisfying assignments to the linear constraints in f with LP methods [8, 19] and check the solution against the other, non-linear, constraints.

There are two differences to consider when using Marabou to solve queries over abstract networks: First, the output of an abstract network is a set of vectors and not a single vector. This does not require any change in the verification query, as Marabou is capable of handling constraints that define continuous sets

in both input and output. Second, the abstract network $\widehat{f}$ structure expresses biases as intervals or singletons, instead of scalars. To address this, we propose two options for encoding the verification query in a form supported by Marabou:

1. The architecture of the original network f is encoded in Marabou using equations. Since the backend LP solver used in Marabou supports linear inequalities, linear constraints in an abstract network are represented directly as inequalities, where the lower and upper bounds reflect the abstracted bias interval. More formally, suppose the bias term $\widehat{b}_{ki}$ in the abstract network $\widehat{f}$ (6) lies in the interval $[\widehat{b}^l_{ki}, \widehat{b}^u_{ki}]$. We can then express the linear constraint as the following pair of inequalities:

$$n_{ki} \geq \sum_j \mathbf{W}^{k-1}_{ji} x^{k-1}_j + \widehat{b}^l_{ki}, \quad \text{and} \quad n_{ki} \leq \sum_j \mathbf{W}^{k-1}_{ji} x^{k-1}_j + \widehat{b}^u_{ki}.$$

   As an example, consider the output neuron in the network f_1, which is originally encoded by the equation:

$$n_{\text{out}} = \sum_{i=1}^{5} \mathbf{W}^1_{i1} n_{1i} + 0.0,$$

   where n_{1i} denotes the i-th neuron in the hidden layer, and the final term is the bias. After converting f_1 into its abstract counterpart $\widehat{f}_1$, the output neuron is instead represented using the pair of inequalities:

$$n_{\text{out}} \geq \sum_{i=1}^{5} \mathbf{W}^1_{i1} n_{1i} + 0.0, \quad \text{and} \quad n_{\text{out}} \leq \sum_{i=1}^{5} \mathbf{W}^1_{i1} n_{1i} + 0.0.$$

   since its bias lies in the interval $[0, 0]$. After applying further abstraction to obtain $\widehat{f}'_1$, these inequalities are updated to reflect the new bounds:

$$n_{\text{out}} \geq \sum_{i=1}^{5} \mathbf{W}^1_{i1} n_{1i} - 0.05, \quad \text{and} \quad n_{\text{out}} \leq \sum_{i=1}^{5} \mathbf{W}^1_{i1} n_{1i} + 0.45.$$

2. The Marabou solver supports skip connections, as these can be represented as additional linear constraints, which are seamlessly handled by the underlying LP solver. Consequently, for each bias term $\mathbf{b}_{ki}$ in an abstract network $\widehat{f}$, we can introduce a fresh input variable p_{ki} that is connected via a skip connection of weight 1 directly to the neuron n_{ki}. The set $\mathcal{P}$ is then extended with a constraint that enforces the interval bounds associated with p_{ki}.
   For example, in the network $\widehat{f}'_1$, the bias of the output neuron is encoded through an additional input variable p_{out}, and $\mathcal{P}_1$ is updated to include the constraint $-0.05 \leq p_{\text{out}} \leq 0.45$.

Both methods allow us to verify $\langle \widehat{f}, \mathcal{P}, \mathcal{Q} \rangle$ directly with Marabou; the first introduces more inequalities, whereas the second introduces more variables. The runtime differences between these approaches are studied in [8, Chap. 4].

5.2 Proving Correctness of the CORA Abstraction

After implementing `verify-with-proofs` in the previous section, we are left with implementing `prove-over-approximation` for the CORA abstraction. In this section, we describe this formalization.

A scheme of proof rules for a DNN with L layers is depicted in Fig. 6. As abstract DNNs are generalizations of DNNs, we first reason about any DNN and its trivial abstraction. For that, we use the proof rule $\mathsf{triv}-\mathsf{abs}$, based on (7). Recall that the correctness of the CORA abstraction is established based on Prop. 1 and Cor. 1, applied sequentially to all layers of the network. This yields the main part of the proof, depicted in the $\mathsf{base}-\mathsf{abs}$ and the $\mathsf{l_k}-\mathsf{abs}$ rules. $\mathsf{base}-\mathsf{abs}$ established the correctness of CORA for the first layer based on any hyperrectangle $\mathcal{I}_1$ bounding the input property $\mathcal{P}$. Then, using $\mathsf{l_k}-\mathsf{abs}$ repeatedly on each layer of the abstract network. Then, we can conclude the correctness of CORA using the $\mathsf{CORA}-\mathsf{L}$ rule for a DNN with L layers. Then, these rules can be integrated into the proof construction scheme described in Sec. 3.1. To ease notation, we assume that all networks' internal variables are well defined as in (1) and (6). In addition, we define the following:

$$
\begin{aligned}
f(\mathbf{x}) &= \phi_L(\mathbf{W}_L \cdots \phi_1(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)\cdots + \mathbf{b}_L),\\
\widehat{f}_0(\mathbf{x}) &= \phi_L(\mathbf{W}_L \cdots \ \phi_1(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1 \oplus \{\mathbf{0}\})\cdots \ + \mathbf{b}_L \oplus \{\mathbf{0}\}),\\
\widehat{f}_k(\mathbf{x}) &= \phi_L(\mathbf{W}_L \cdots \phi_k(\widehat{\mathbf{W}}_K \cdots \phi_1(\widehat{\mathbf{W}}_1\mathbf{x}+\widehat{\mathbf{b}}_1 \oplus \widehat{\mathcal{I}}_1)\cdots \widehat{\mathbf{b}}_k \oplus \widehat{\mathcal{I}}_k)\cdots + \mathbf{b}_L \oplus \{\mathbf{0}\}),\\
\widehat{f}(\mathbf{x}) &= \widehat{f}_L(\mathbf{x}) = \phi_L(\widehat{\mathbf{W}}_L \cdots \ \phi_1(\widehat{\mathbf{W}}_1\mathbf{x}+\widehat{\mathbf{b}}_1 \oplus \widehat{\mathcal{I}}_1)\cdots \ + \widehat{\mathbf{b}}_L \oplus \widehat{\mathcal{I}}_L)
\end{aligned}
$$

Proof checking. In order to check a proof witness for CORA abstraction, the checker needs to receive the original DNN verification query $\langle f, \mathcal{P}, \mathcal{Q}\rangle$, as well as the candidate abstract network $\widehat{f}$. Then, any intermediate abstract network $\widehat{f}_k$ can be constructed during the checking process based on $\widehat{f}$ and f.

$$
\mathsf{triv}-\mathsf{abs}: \ \frac{f \ \ \widehat{f}_0 \ \ \mathbf{x}\in\mathbb{R}^{n_0}}{f(\mathbf{x}) \in \widehat{f}_0(\mathbf{x})} \qquad \mathsf{base}-\mathsf{abs}: \ \frac{\widehat{f}_0 \ \ \widehat{f}_1 \ \ \mathbf{x}\in\mathcal{P}\subseteq\widehat{\mathcal{I}}_1}{\forall x\in\mathcal{P}.\ \widehat{f}_0(\mathbf{x}) \subseteq \widehat{f}_1(\mathbf{x})}
$$

$$
\mathsf{l_k}-\mathsf{abs}: \ \frac{\begin{array}{c} k\in\{2,\cdots,L-1\} \ \ \mathcal{B}\subset[n_k] \ \ \widehat{f}_k \ \ \widehat{f}_{k+1} \\ \widehat{\mathbf{W}}_k = \mathbf{W}_{k(\bar{B},\cdot)} \ \ \widehat{\mathbf{b}}_k = \mathbf{b}_{k(\bar{B})} \ \ \widehat{\mathbf{W}}_{k+1} = \mathbf{W}_{k+1(\cdot,\bar{B})} \ \ \widehat{\mathbf{b}}_{k+1} = \mathbf{b}_{k+1} \\ \widehat{\mathcal{I}}_k = \widehat{\mathcal{I}}_{k(\bar{\mathcal{B}})} \ \ \widehat{\mathcal{I}}_{k+1} = \mathbf{W}_{k+1(\cdot,\mathcal{B})}\mathcal{I}_{k(\mathcal{B})} \end{array}}{\forall x\in\mathcal{P}.\ \widehat{f}_k(\mathbf{x}) \subseteq \widehat{f}_{k+1}(\mathbf{x})}
$$

$$
\mathsf{CORA}-\mathsf{L}: \ \frac{f(\mathbf{x})\in\widehat{f}_0(\mathbf{x}) \ \ \forall x\in\mathcal{P}.\ \widehat{f}_0(\mathbf{x})\subseteq\widehat{f}_1(\mathbf{x}) \ \ \cdots \ \ \forall x\in\mathcal{P}.\ \widehat{f}_{L-1}(\mathbf{x})\subseteq\widehat{f}(\mathbf{x}) \ \ \mathcal{Q}\subset\mathbb{R}^{n_L}}{\mathtt{unsat}(\langle\widehat{f},\mathcal{P},\mathcal{Q}\rangle) \implies \mathtt{unsat}(\langle f,\mathcal{P},\mathcal{Q}\rangle)}
$$

Fig. 6: A scheme of proof rules for CORA abstraction of a DNN with L layers

6 Related Work

This work builds on two main pillars in DNN verification: proof production and abstraction.

Proof production is a well-established area in formal verification, particularly within SAT and SMT solvers [5, 20, 37] among many others, where the generation of proofs or certificates serves to improve trust in automated verification results. These proofs can be independently checked, enhancing the reliability of verification pipelines. Despite its importance in traditional verification, proof production remains largely unexplored in the context of DNN verification, and most existing tools do not provide formal proofs as part of their output.

Abstraction [9–11] is a classical technique in formal verification, widely used to tackle scalability and complexity challenges. In the domain of DNN verification, abstraction has been actively studied through two main lenses. The first is *abstract interpretation*, which over-approximates neural network behavior using abstract domains [16, 44]. The second is *abstraction refinement*, where the verifier incrementally refines the abstraction based on counterexamples or property violations, improving precision over time [3, 13, 29, 33, 38]. These techniques have proven effective in improving both scalability and verification success rates.

However, the intersection of abstraction and proof production has received very limited attention, in both directions: how proofs can influence abstraction, and how abstraction can contribute to proof construction. An early example of the former, in the SAT domain, is the work of [20], where proofs are used to guide and refine the abstraction process.

As for the latter, our work is, to the best of our knowledge, among the first to investigate how abstraction mechanisms can be directly integrated into the production of formal proofs in the context of DNN verification. In a recent work [43], a Domain Specific Language (DSL), designed for defining and certifying the soundness of abstract interpretation DNN verifiers, is introduced and evaluated over several DNN verifiers. This work is focused on proving DNN verifiers that employ linear over approximations of activation functions, while our method focuses on separate proofs for neuron-merging abstraction process, and for the verification process. An interesting future work would be to formalize our scheme using the DSL of [43].

7 Conclusion and Future Work

This work in progress aims to bridge abstraction and proof production in DNN verification. On the one hand, incorporating abstraction enhances the efficiency and compactness of proof production in the formal verification of neural networks. On the other hand, proofs can be generated for verification tools that apply abstraction to improve performance. To achieve this, we enforce the abstraction process itself to be certified. By introducing a modular proof rule that separates the verification proof from the abstraction proof, we establish a foundation for generating complete proofs while using abstraction to aid in their construction. This modular approach allows integration with existing proof-producing verifiers for the verification component, while enabling the development of novel proof mechanisms specific to abstraction. We presented two general algorithms for abstraction-refinement-based proof production in DNN verification and demonstrated how these can be instantiated using current tools for both proof generation and abstraction.

The next steps of our work include the implementation and evaluation of our method with respect to proof size, verification time, and proof-checking time; over real-world benchmarks. Looking forward, we identify two promising directions for future work. First, integrating residual reasoning [14] could improve the effectiveness of abstraction refinement procedures. Second, leveraging abstract proofs within CDCL-based frameworks [22, 34] offers a compelling avenue for bridging abstraction and clause learning-based proof systems.

Acknowledgements

The research presented in this paper was partially funded by the project FAI under project number 286525601 funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG).

This work was partially funded by the European Union (ERC, VeriDeL, 101112713). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

This work was performed in part using high-performance computing equipment obtained under NSF Grant #2117377.

References

1. Althoff, M.: An Introduction to CORA 2015. In: 1st and 2nd Int. Workshop on Applied Verification for Continuous and Hybrid Systems, (ARCH). vol. 34, pp. 120–151 (2015)
2. Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., Schmid, C.: ViViT: A Video Vision Transformer. In: Int. Conf. on Computer Vision (ICCV). pp. 6816–6826 (2021)
3. Ashok, P., Hashemi, V., Křetínskỳ, J., Mohr, S.: DeepAbstract: Neural Network Abstraction for Accelerating Verification. In: Proc. 18th Int. Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 92–107 (2020)
4. Bak, S.: nnenum: Verification of relu neural networks with optimized abstraction refinement. In: Proc. 13th NASA Formal Methods Symposium (NFM). pp. 19–36 (2021)
5. Barbosa, H., Reynolds, A., Kremer, G., Lachnitt, H., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Viswanathan, A., Viteri, S., Zohar, Y., Tinelli, C., Barrett, C.: Flexible Proof Production in an Industrial-Strength SMT Solver. In: Proc. 11th Int. Joint Conf. on Automated Reasoning (IJCAR). pp. 15–35 (2022)
6. Barrett, C., de Moura, L., Fontaine, P.: Proofs in Satisfiability Modulo Theories. All about Proofs, Proofs for All **55**(1), 23–44 (2015)
7. Brix, C., Müller, M., Bak, S., Johnson, T., Liu, C.: First Three Years of the International Verification of Neural Networks Competition (VNN-COMP). Int. Journal on Software Tools for Technology Transfer pp. 1–11 (2023)
8. Chvátal, V.: Linear Programming. Macmillan (1983)
9. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. 12th Int. Conf. on Computer Aided Verification (CAV). pp. 154–169 (2000)

10. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems **16**(5), 1512–1542 (1994)
11. Cousot, P., Cousot, R.: Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). p. 238–252 (1977)
12. Ehlers, R.: Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In: Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA). pp. 269–286 (2017)
13. Elboher, Y., Gottschlich, J., Katz, G.: An Abstraction-Based Framework for Neural Network Verification. In: Proc. 32nd Int. Conf. on Computer Aided Verification (CAV). pp. 43–65 (2020)
14. Elboher, Y.Y., Cohen, E., Katz, G.: Neural network verification using residual reasoning. In: Proc. 20th Int. Conf. on Software Engineering and Formal Methods (SEFM). pp. 173–189 (2022)
15. Elsaleh, R., Katz, G.: DelBugV: Delta-Debugging Neural Network Verifiers. In: Proc. 23rd Int. Conf. Formal Methods in Computer-Aided Design (FMCAD). pp. 34–43 (2023)
16. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, E., Chaudhuri, S., Vechev, M.: AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In: Proc. 39th IEEE Symposium on Security and Privacy (S&P). pp. 3–18 (2018)
17. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, vol. 1. MIT press Cambridge (2016)
18. Griggio, A., Roveri, M., Tonetta, S.: Certifying Proofs for SAT-Based Model Checking. Formal Methods in System Design (FMSD) **57**(2), 178–210 (2021)
19. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), `https://www.gurobi.com`
20. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from Proofs. In: Proc. 31st ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). p. 232–244 (2004)
21. Isac, O., Barrett, C., Zhang, M., Katz, G.: Neural Network Verification with Proof Production. In: Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD). pp. 38–48 (2022)
22. Isac, O., Refaeli, I., Wu, H., Barrett, C., Katz, G.: Proof-Driven Clause Learning in Neural Network Verification (2025), Technical Report. `http://arxiv.org/abs/2503.12083`
23. Jia, K., Rinard, M.: Exploiting Verified Neural Networks via Floating Point Numerical Error. In: Proc. 28th Int. Static Analysis Symposium (SAS). pp. 191–205 (2021)
24. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Proc. 29th Int. Conf. on Computer Aided Verification (CAV). pp. 97–117 (2017)
25. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: a Calculus for Reasoning about Deep Neural Networks. Formal Methods in System Design (FMSD) (2021)
26. Kochdumper, N., Schilling, C., Althoff, M., Bak, S.: Open- and Closed-Loop Neural Network Verification Using Polynomial Zonotopes. In: Proc. 15th NASA Formal Methods Symposium (NFM). pp. 16–36 (2023)
27. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet Classification with Deep Convolutional Neural Networks. Advances in neural information processing systems (NeuRIPS) **25** (2012)

28. Ladner, T., Althoff, M.: Automatic abstraction refinement in neural network verification using sensitivity analysis. In: Proc. 26th ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC). pp. 1–13 (2023)
29. Ladner, T., Althoff, M.: Fully Automatic Neural Network Reduction for Formal Verification (2023), Technical Report. http://arxiv.org/abs/2305.01932
30. LeCun, Y., Bengio, Y., Hinton, G.: Deep Learning. Nature **521**(7553), 436–444 (2015)
31. Lipton, Z.C.: The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. Queue **16**(3), 31–57 (2018)
32. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for Verifying Deep Neural Networks. Foundations and Trends in Optimization **4**(3-4), 244–404 (2021)
33. Liu, J., Xing, Y., Shi, X., Song, F., Xu, Z., Ming, Z.: Abstraction and Refinement: Towards Scalable and Exact Verification of Neural Networks. ACM Transactions on Software Engineering and Methodology **33**(5), 1–35 (2024)
34. Liu, Z., Yang, P., Zhang, L., Huang, X.: DeepCDCL: A CDCL-based Neural Network Verification Framework. In: Proc. 18th Int. Symposium on Theoretical Aspects of Software Engineering (TASE). pp. 343–355 (2024)
35. Lopez, D.M., Choi, S.W., Tran, H.D., Johnson, T.T.: NNV 2.0: The Neural Network Nerification Tool. In: Proc. 35th Int. Conf. on Computer Aided Verification (CAV). pp. 397–412 (2023)
36. Nair, V., Hinton, G.E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: Proc. 27th Int. Conf. on Machine Learning (ICML). p. 807–814 (2010)
37. Niemetz, A., Preiner, M., Reynolds, A., Zohar, Y., Barrett, C., Tinelli, C.: Towards Bit-Width-Independent Proofs in SMT Solvers. In: Proc. 27th Int. Conf. on Automated Deduction (CADE). pp. 366–384. Springer (2019)
38. Ostrovsky, M., Barrett, C., Katz, G.: An abstraction-refinement approach to verifying convolutional neural networks. In: Proc. 20th Int. Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 391–396 (2022)
39. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning Transferable Visual Models From Natural Language Supervision. In: Proc. 38th Int. Conf. on Machine Learning (ICML) (2021)
40. Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Robust Speech Recognition via Large-Scale Weak Supervision. In: Proc. 40th Int. Conf. on Machine Learning (ICML) (2023)
41. Rudin, C.: Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. Nature machine intelligence **1**(5), 206–215 (2019)
42. Sälzer, M., Lange, M.: Reachability Is NP-Complete Even for the Simplest Neural Networks. In: Proc. 15th Int. Conf. on Reachability Problems (RP). pp. 149–164 (2021)
43. Singh, A., Sarita, Y.C., Mendis, C., Singh, G.: Automated verification of soundness of dnn certifiers. Proc. ACM on Programming Languages **9** (2025)
44. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An Abstract Domain for Certifying Neural Networks. In: Proc. 46th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). pp. 1–30 (2019)
45. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: WaveNet: A Generative Model for Raw Audio. In: Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW). p. 125 (2016)

46. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is All You Need. In: Proc. 31st Conf. on Advances in Neural Information Processing Systems (NeuRIPS). vol. 30 (2017)
47. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.: Marabou 2.0: A Versatile Formal Analyzer of Neural Networks. In: Proc. 36th Int. Conf. on Computer Aided Verification (CAV) (2024)

A Naive Algorithm

This appendix includes the naive algorithm to integrate proof production with abstraction and refinement.

Algorithm 2 Naive Proof Production with Abstraction

Input f, $\mathcal{P}$, $\mathcal{Q}$ **Output** proof that $\texttt{unsat}(\langle f, \mathcal{P}, \mathcal{Q}\rangle)$, or counterexample

1: $\widehat{f} = \texttt{abstract}(f, \mathcal{P})$
2: **while** true **do**
3: $p_a = \texttt{prove-over-approximation}(\widehat{f}, f, \mathcal{P}, \mathcal{Q})$
4: result, $p_q = \texttt{verify-with-proofs}(\widehat{f}, \mathcal{P}, \mathcal{Q})$
5: **if** result == `SAT` and example is not spurious **then**
6: **return** `SAT`, example
7: **else if** p_a and p_q were successfully generated **then**
8: **return** `UNSAT`, $\langle p_a, p_q\rangle$
9: **else**
10: $\widehat{f} = \texttt{refine}(\widehat{f}, f)$
11: **end if**
12: **end while**