INNAbstract: An INN-Based Abstraction Method for Large-Scale Neural Network Verification

Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, Mohamed Ghazel

INNAbstract: An INN-Based Abstraction Method for Large-Scale Neural Network Verification

Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, and Mohamed Ghazel, *Member, IEEE*

***Abstract*— Neural networks (NNs) have witnessed widespread deployment across various domains, including some safety-critical applications. In this regard, the demand for verifying means of such artificial intelligence techniques is more and more pressing. Nowadays, the development of evaluation approaches for NNs is a hot topic that is attracting considerable interest, and a number of verification methods have been proposed. Yet, a challenging issue for NN verification is pertaining to the scalability when some NNs of practical interest have to be evaluated. This work aims to present *INNAbstract*, an abstraction method to reduce the size of NNs, which leads to improving the scalability of NN verification and reachability analysis methods. This is achieved by merging neurons while ensuring that the obtained model (i.e., abstract model) overapproximates the original one. *INNAbstract* supports networks with numerous activation functions. In addition, we propose a heuristic for nodes' selection to build more precise abstract models, in the sense that the outputs are closer to those of the original network. The experimental results illustrate the efficiency of the proposed approach compared to the existing relevant abstraction techniques. Furthermore, they demonstrate that INNAbstract can help the existing verification tools to be applied on larger networks while considering various activation functions.**

***Index Terms*— Formal verification, neural network (NN) abstraction, neural networks (NNs), NN verification.**

I. Introduction

NEURAL networks (NNs) are one of the most widely used techniques in machine learning. They have been successfully applied to solve many complex problems in different domains, including automatic speech recognition, computer vision, and healthcare [1], [2]. The success of NNs motivated the academic and industrial communities to adopt this technique in safety-critical systems, such as autonomous cars and trains [3], [4] and air traffic collision avoidance systems [5]. The development of such systems requires a rigorous process of testing, verification, and validation to ensure that each (software) component in the system meets a set of functional and safety requirements. Evaluating the network's performance on a representative set of samples (test set) is a common way to assess its accuracy. However, this remains insufficient when such NN-based software is used in safety-critical applications since there is no guarantee of how the network would behave on other unseen samples. Indeed, recent studies have demonstrated the sensitivity and the vulnerability of NNs to adversarial attacks (i.e., small perturbations to the inputs that can drastically change the corresponding outputs). These perturbations are generally imperceptible to humans and easy to generate [6], [7]. For instance, Szegedy et al. [8] demonstrated that adding small noise to an image may lead the NN to misclassify the image that was, beforehand, correctly classified. For these reasons, NN verification has become a hot research topic and has witnessed a noticeable interest in the last few years [9], [10], [11].

Formal verification methods, such as model checking [12], satisfiability analysis (SAT) [13], reachability analysis, and abstract interpretation [14] have proven to be effective in offering formal proofs of the safety for both software and hardware systems. Recently, these techniques have been adopted to verify some properties of NNs. Verifying an NN consists of checking that the NN never violates some predefined properties. Namely, for a given input region defined by a set of constraints, we check whether the NN's generated output is necessarily within a safe output region [15]. Early works dealing with this issue relied on reducing the verification process to an optimization problem by encoding the input–output constraints (the property) and the network's behavior (the model) as a linear programming (LP) problem and then applying an LP-solver, such as mixed integer LP (MILP) [16], [17] or satisfiability modulo theory (SMT) solvers [18], [19], [20] to check the properties. NN verification is known to be an NP-complete problem [21], and the complexity of the verification algorithms increases exponentially with the number of neurons in the network. As a result, most of the existing verifiers encounter the state-space explosion problem. For instance, a straightforward application of SAT/SMT or MILP on a network of n *Relu* neurons may result in generating 2^n linear subproblems. One way to address this issue is to resort to approximation and abstraction methods. The general concept of these methods is to either linearly approximate the activation function, or merge some neurons of the network to create an abstract one that is a sound overapproximation of the original network [22]. The latter are often called model reduction techniques.

We should mention here another field known as *NN compression*, which is interested in merging neurons to build

Manuscript received 23 December 2022; revised 31 July 2023; accepted 11 September 2023. *(Corresponding author: Fateh Boudardara.)*

Fateh Boudardara and Abderraouf Boussif are with the Technological Research Institute Railenium, F-59308 Valenciennes, France (e-mail: fateh.boudardara@railenium.eu; abderraouf.boussif@railenium.eu).

Pierre-Jean Meyer and Mohamed Ghazel are with Universite Gustave Eiffel, COSYS-ESTAS, F-59666 Villeneuve d'Ascq, France (e-mail: pierre-jean.meyer@univ-eiffel.fr; mohamed.ghazel@univ-eiffel.fr).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2023.3316551.

Digital Object Identifier 10.1109/TNNLS.2023.3316551

smaller networks with comparable performance to the original one. In particular, methods based on quantization [23] and pruning [24] are extensively used in NN compression. These techniques aim to produce smaller networks to improve their efficiency in terms of memory usage and computational resources. This allows them to be deployed on resource-constrained devices. However, it is important to emphasize that NN compression does not provide any formal guarantee about the obtained reduced network. Consequently, the verification results on the reduced network cannot be lifted to the original network, and thus no inference can be made about the correctness of the original network. Unlike NN compression, NN abstraction (or model reduction) methods generate abstract networks that are used to enhance the verification process. Indeed, once the required properties have been successfully verified on an abstract network, we can lift this result to the original network, allowing for its safe deployment in the intended system.

In this article, we present INNAbstract, a model reduction method for NNs to enhance the scalability of NN operations, particularly in the context of verification and reachability analysis. The proposed method relies on merging a set of nodes (neurons) and calculating the abstract node's weights using mathematical formulas to ensure that the abstract model is an overapproximation of the original one (i.e., the output of the original network is always within the output range of the abstract network). Indeed, node merging allows for reducing the size of the NN and, hence, instead of verifying properties on the large original model, these properties can be checked on the reduced model with lower computation complexity. This helps improve the scalability of the reachability analysis and verification processes. Unlike other abstraction techniques, INNAbstract applies to NNs with various activation functions. To assess the efficiency and the scalability of the proposed approach, we conducted a series of experiments on a set of selected benchmarks from the literature.

Notice that this work is a revised and extended version of a workshop article [25], in which the general idea of the approach along with some early preliminary results was presented. Compared to the preliminary workshop version, in this article.

1) We present the theoretical foundations of the approach while extending it to support a wide range of activation functions (*Relu*, *Tanh*, *Bipolar Sigmoid*, etc.).
2) We provide the formal proofs of the underlying theoretical results, particularly regarding the assurance of the overapproximation relation.
3) We provide the pseudo-algorithms that implement the main steps of the abstraction process.
4) We propose a heuristic for efficiently selecting the nodes to be merged, to enhance the precision of the abstraction.
5) Finally, we present a software tool implementing the approach, which is used to conduct an extensive experimental study (on random-generated and well-known benchmark networks), to evaluate the efficiency of the approach. Furthermore, we compare our method to other relevant approaches from the literature.

The remainder of this article is structured as follows: Section II provides a brief overview of NN verification and abstraction works. Section III gives basic concepts and notations of NNs and recalls some concepts pertaining to the NN verification problem. Section IV presents the proposed abstraction approach along with the underlying pseudo-algorithms, and heuristics for selecting nodes to be merged. The description of the conducted experiments and some related discussion are given in Section V. Finally, Section VI concludes the article and proposes some guidelines for future works.

II. Related Works

The interest in NN verification can be traced back to the early 2000s [26]. However, the first concrete work that deals with this problem was proposed by Pulina and Tacchella [20], where they proposed an SMT-based verification method for Sigmoid feed-forward NNs. Katz et al. [21] introduced Reluplex for Relu networks. Reluplex combines the Simplex optimization method with the SAT technique to handle the nonlinearity of *Relu* constraints. Many verification methods based on SAT/SMT solvers for *Relu* NNs have been proposed [18], [19]. Other methods based on LP-solving encode the network behavior and the input–output constraints as an MILP problem and apply an adequate solver to address the verification problem [17], [27], [28]. Although these techniques are exact since they encode the complete behavior of the network and the property to be verified, they are severely limited to small networks due to the complexity and the nonlinearity of NNs [29].

Another line of research consists of applying the concept of abstract interpretation [14] in an attempt to build an abstract network that is easy to verify, and at the same time, overapproximates the behavior of the original network. The abstraction method must ensure that if the property holds on the abstract model, it necessarily holds on the original one. Various approaches based on convex and linear relaxation of the activation function have been proposed. Gehr et al. [30] applied abstract interpretation to verify NNs for the first time. They proposed an abstraction of the *Relu* activation function using zonotopes. This approach is lately improved further by proposing an abstraction for other activation functions, such as *Tanh* and *Sigmoid* [31], [32]. In other works, the problem of NN verification is formulated as a reachability problem that is solved by calculating the reachable set (region) of the network with respect to some given input region. Therefore, the verification is carried out by checking whether the obtained output region is included in some predefined safe region [33], [34], [35], [36]. Since reachability methods are incomplete and may generate coarse outputs, some optimization methods are proposed for networks with piecewise linear activation function, particularly for *Relu*-NN, to provide tighter bounds of the outputs [37], [38], [39], [40], [41]. Some recent works propose a convex relaxation of *Relu* networks by considering multiple nodes and the dependencies between them at the same time [42], [43]. In this article, we employed the interval bounds propagation (IBP) algorithm [34] as a reachability analysis technique to calculate the output range of the abstract and original networks.

As a subcategory of abstraction methods, model reduction methods' objective is to build an abstract (also called reduced) model by merging a set of neurons of the same layer. Hence, instead of applying the verification on the large original model, it is rather applied on the smaller abstract model [44], [45], [46]. Ashok et al. [44] proposed *DeepAbstract* as a model reduction method. DeepAbstract implements the *K-mean* clustering algorithm to rearrange *Relu* nodes of a layer in clusters. Next, each cluster is replaced by its representative node. The overapproximation is established by propagating the error introduced during the merging process and, then, the verifier *ERAN* [31] is brought into play for verification. Elboher et al. [46] proposed a model reduction method for *Relu*-NN. The main idea of the approach is to classify neurons into four different categories according to the sign of their outgoing weights (positive and negative), and whether increasing the value of the neuron at hand would increase or decrease the network's output (increasing and decreasing). Finally, model reduction is applied by merging nodes belonging to the same category. To evaluate the performance of the approach, Elboher et al. [46] used the *Marabou* tool [19] as a back-end verifier and compared the obtained results to those obtained by Marabou without applying the model reduction method. Recently, a similar method to the one of Elboher et al. [46] is proposed by Liu et al. [47], where the authors classify neurons into only two categories (increasing and decreasing) instead of four categories. In addition, they introduced a new operation called *FREEZE* to replace an increasing (resp. decreasing) neuron by its upper bound (resp. lower bound) and then propagate this constant through the network. Another approach for *Relu*-NN model reduction based on Interval NNs (INNs) was introduced by [45]. The abstract node is defined by its incoming and outgoing weights; its incoming weight is the interval-hull of the incoming weights of the corresponding set of merged nodes in the original network, and its outgoing weight is the interval-hull of the outgoing weights of the same set multiplied by its size (i.e., the number of nodes). Moreover, the big-M MILP encoding [16] is adopted to compute the output range of the obtained INN. A generalization of this approach is proposed by Sotoudeh and Thakur [48], where the abstraction can be performed using further abstract domains, such as Zonotopes, and is not restricted to intervals. Recently, Prabhakar defined NN-bisimulation that can be used to reduce the size of NN [49]. As a consequence of NN-bisimulation's conditions, which are far from being satisfied on real-world models, a more tolerated (mitigated) approach, namely δ-bisimulation, is proposed in the same article [49]. Refer to [22], for a comprehensive review of model reduction methods.

A related research field, known as NN compression, focuses also on reducing the size of NNs by merging neurons and eliminating certain weighted-edges without affecting their accuracy. Techniques such as pruning [24], quantization [23], and distillation [50] have gained significant attention in the field. These techniques are more valuable to build compressed NN models to effectively deploy them on resource-constrained devices. However, it is important to note that these techniques do not provide any formal guarantee about the obtained compressed models [51], making them unsuitable for verification purposes.

In the present article, we propose a model reduction method that provides formal guarantees on the obtained network and ensures that the reduced model overapproximates the original one. Compared to the aforementioned abstraction works, the abstraction technique elaborated in our work is more general. While most of the previously discussed works are limited to the *Relu* activation function, our method supports different types of activation functions. Moreover, compared to [44], where the abstraction depends on the set of input images on which one wants to check the robustness of the network, our approach generates abstract networks independent of any specific input domains. We should also mention the two techniques presented in [46] and [47] which focus on categorizing the weights of the network by analyzing their signs and impact on the output (increasing or decreasing). Both techniques require a preprocessing phase to classify all nodes of the networks. Additionally, these approaches can only merge nodes that belong to the same category. In contrast, our approach does not require any preprocessing phase, and if applied until saturation, a layer can be (theoretically) reduced to a single abstract node. Finally, we should highlight that the approach proposed in [45] is the closest to our technique since both are based on INN. However, there is a significant difference in our approach. Specifically, Prabhakar and Afzal [45] define the abstract weights as the interval-hull of the corresponding weights in the original network. In our method, we define the abstract outgoing weights as the sum of absolute values over the original weights and transfer their signs to the corresponding incoming weights. This approach helps build more precise abstract networks,[1] as can be observed through the experimental results discussed in this article.

III. Preliminaries

A. Deep NNs

Deep NNs (DNNs) are a graph-based machine-learning technique. Depending on the application domain, many DNN architectures have been proposed (CNNs, RNNs, etc.) [9], [52], [53]. In this article, we focus on feed-forward NNs [54] and we use the term DNN to refer to them.

A DNN is a set of interconnected nodes organized in layers. It receives (numerical) data from the input layer and propagates them through its *hidden* (i.e., intermediate) layers, up to the last one which is called the output layer. Fig. 1 depicts an example of an NN with a 4-D input layer, two hidden layers, and a 2-D output layer. The nodes are connected via weighted edges whose values are determined during the training phase.

Definition 1: For a DNN N, the set of nodes of a layer l_i is represented by S_i. S_0 is the input set, S_n is the output set, and $\forall i \in \{1, 2, \ldots, n-1\}$, S_i represents the set of nodes of the hidden layer l_i. Two successive layers, l_{i-1} and l_i, are connected via a weighted-matrix W_i, such that for $i \in \{1, 2, \ldots, n\}$, $w^i_{jk} = w(s_{i-1,k}, s_{i,j})$ denotes the weight

[1] In the sense that its output is closer to the output of the original network.

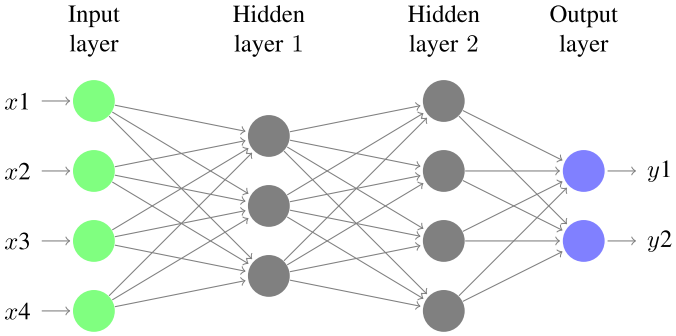


Fig. 1. Example of an NN.

of the edge connecting $s_{i-1,k} \in S_{i-1}$ to $s_{ij} \in S_i$, for all $k \in \{1, 2, \ldots, |S_{i-1}|\}$ and $j \in \{1, 2, \ldots, |S_i|\}$.

Definition 2: Given a node $s_{ij} \in S_i$, $i \in \{1, 2, \ldots, n\}$, its valuation $v(s_{ij}) = v_{ij}$ is calculated as follows:

$$v(s_{ij}) = \alpha\left(\sum_{s_{i-1,k} \in S_{i-1}} w(s_{i-1,k}, s_{ij}) \times v(s_{i-1,k})\right) \tag{1}$$

where $\forall s_{0j} \in S_0$, $j \in \{1, 2, \ldots, |S_0|\}$, $v(s_{0j}) = x_j$ is the value of the jth element of the input x, and $\alpha : \mathbb{R} \to \mathbb{R}$ is the activation function of the DNN.

Notice that several types of activation functions can be used with DNNs. For instance, (2) and (3) define *Relu* and *Tanh* functions, respectively,

$$\text{relu}(x) = \max(0, x); \quad x \in \mathbb{R} \tag{2}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \quad x \in \mathbb{R}. \tag{3}$$

From Definition 2, and with a slight abuse of notation, we use $v(S_i)$ or V_i to denote the valuation vector corresponding to all nodes of layer S_i, that is,

$$V_i = v(S_i) = [v(s_{i1}), v(s_{i2}), \ldots, v(s_{i|S_i|})]^\tau.$$

Definition 3: Given a DNN of n layers with its associated function: $N : \mathbb{R}^{|S_0|} \to \mathbb{R}^{|S_n|}$, its valuation for an input x is $N(x) = v(S_n)$.

In Definition 3, the function N is the composition of a set of vector functions f_i, where $i \in \{1, 2, \ldots, n\}$, that is, $N = f_n \circ f_{n-1} \circ \cdots \circ f_1$, where f_i is the function of layer l_i and is defined as follows:

$$f_i = \alpha\Big(\sum W_i \times v(S_{i-1})\Big) \tag{4}$$

where $W_i \in \mathbb{R}^{|S_i| \times |S_{i-1}|}$ is the weight matrix of layer l_i. In the remainder of this article, we exchangeably use w^i_{jk} or $w(s_{i-1,k}, s_{ij})$ to represent the weight of the edge connecting the node $s_{i-1,k} \in S_{i-1}$ to the node $s_{ij} \in S_i$.

Remark 1: For the sake of simplicity of formulas and proofs, we did not include biases explicitly in the definition of DNNs. In fact, the bias vector b_i of a layer l_i can be replaced by a weight vector connecting a new node $s_{i-1,b} \in S_{i-1}$ to all the nodes s_{ij} of S_i such that $v(s_{i-1,b}) = 1$ and $w(s_{i-1,b}, s_{ij}) = b_{ij}$. An illustrative example is given in Fig. 2.

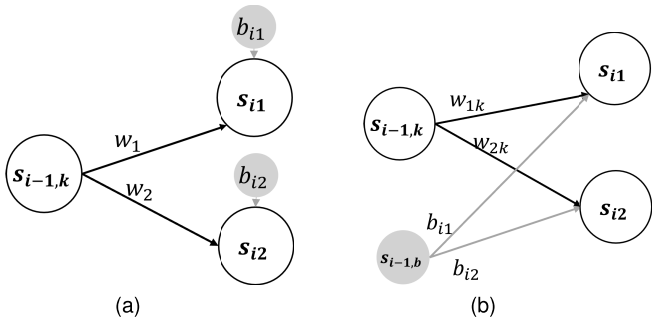


Fig. 2. Example explaining how to replace the bias vector with a weight vector. (a) Network with a bias vector. (b) Equivalent network to the one presented in (a) where biases are replaced by weights, s.t. $v(S_{i-1,b}) = 1$.

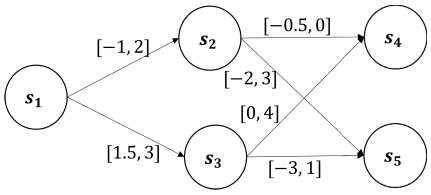


Fig. 3. Example of an INN of three layers.

Interval NN: A new structure to represent NNs, namely INNs, was introduced by Prabhakar and Afzal [45]. In INNs, a weight w of an edge is an interval $w = [w^l, w^u]$, such that $w^l \leq w^u$. Fig. 3 depicts an example of an INN composed of three layers. INN is a more general representation of DNN since any DNN can be transformed into an equivalent INN by defining its weights as $w = [w^l, w^u]$, s.t. $w^l = w^u$.

B. DNN Verification

Since the parameters of a DNN are determined during the training phase of the model on a limited set of data, DNNs may not cover the whole behavior of the modeled system. For example, a DNN model for image classification may misclassify some human-imperceptible perturbed images, which would cause significant damage if it is deployed in a safety-critical system. That is why verifying NNs is crucial.

In general terms, the goal of verifying a network is to check its correctness with respect to some desired properties. Despite the existence of many types of verification properties for DNN, most of the related works focus on the *invariant property*. For such property, NN verification aims to prove (or disapprove) that for a specific input region, the DNN output lies within some predefined output region. Accordingly, the verification problem of an NN $N : \mathbb{R}^{|S_0|} \to \mathbb{R}^{|S_n|}$ can be defined with the tuple $\langle N, \text{Pre}, \text{Post}\rangle$ such that Pre and Post are constraints on the inputs and the outputs of the DNN N, respectively. In other words, for an input region defined by a set of constraints Pre, the objective is to verify whether all the corresponding outputs are within the region defined by the set of constraints Post. This is formulated as

$$\forall x \in \mathbb{R}^{|S_0|}, \text{Pre}(x) \implies \text{Post}(N(x)). \tag{5}$$

Therefore, the verification process consists of either proving that the property defined in (5) is true, or providing

a counterexample $(x^*, y^*) \in \mathbb{R}^{|S_0|} \times \mathbb{R}^{|S_n|}$ such that $y^* = N(x^*)$, $\text{Pre}(x^*)$ holds true and $\text{Post}(y^*)$ is false.

For image classification, most of the research works focus on DNN robustness [10]. In this case, for a given input image x_0 that is classified as c_{x_0}, the *Pre* constraints define a region μ that includes perturbed images around x_0, and the goal of a verification technique is to check whether all these images have the same class c_{x_0} using the network N. If so, we say the network is robust with respect to region μ.

A common approach to NN verification is to encode the network and the property to be checked as an LP problem. Then, off-the-shelf LP solvers, for example, SAT/SMT and MILP solvers, can be used to perform the verification process. However, methods based on this approach suffer from the state-space explosion problem. This issue primarily arises from the nonlinear behavior of DNNs induced by the activation functions, including simple functions such as piecewise linear functions [e.g., Relu (2)]. Indeed, a *Relu* neuron has to be split into two linear equations, namely, if $y = \text{relu}(x)$, then $y = 0$ when $x < 0$ and $y = x$ when x is positive. Thus, verifying a network of n *Relu* neurons leads to 2^n linear subproblems, and, accordingly, the complexity of the verification problem grows exponentially with the number of neurons. To cope with this inherent complexity problem, abstraction methods are employed to build smaller networks while conserving the relevant behavior of the original one. For instance, these methods can be used to generate abstract networks, which constitute an overapproximation of the original network by merging its nodes [45], as defined below.

Definition 4: A network $\bar{N}$ with its associated function $\bar{N} : \mathbb{R}^{|S_0|} \rightarrow \mathbb{R}^{|S_n|}$ is an overapproximation of a network N if the following relation is fulfilled:

$$\bar{N} \models P \implies N \models P. \tag{6}$$

The above definition means that any time the property P is satisfied by the abstract network $\bar{N}$, P is necessarily satisfied by N. Formula (6) can be combined with Formula (5) as follows:

$$\forall x \in \mathbb{R}^{|S_0|}, \text{Pre}(x) \wedge \text{Post}(\bar{N}(x)) \implies \text{Post}(N(x)). \tag{7}$$

It is worth noticing that if $\bar{N} \not\models P$, no conclusion can be inferred regarding N. In this case, we need to check whether the counterexample violating the property P on $\bar{N}$ (i.e., $\bar{N} \not\models P$) leads to $N \not\models P$. If for the generated counterexample, $\bar{N} \not\models P$ but $N \models P$, then this counterexample is called a *spurious* one and hence no conclusion can be made regarding the satisfiability of P by the original network N.

IV. Proposed Approach

The objective of our method is to reduce the size of a DNN by merging a set of nodes. Abstracting a set of nodes requires providing formulas to determine the incoming and outgoing weights of the obtained abstract node, and a proof ensuring that the reduced network over-approximates the original one. The broad idea of this approach is to define the output weights of the abstract node as the sum of the absolute values over the outgoing weights of the set of merged nodes and bring back their signs to be used with their incoming weights to calculate the new incoming weights of the abstract node. Fig. 4 provides an illustration of the abstraction procedure, where Fig. 4(a) represents the original network N, and Fig. 4(b) represents the obtained abstract network. Mathematical formulations and examples of the application of the proposed method are presented below. First, we start by giving a general formula for NNs with odd and monotone activation functions and then provide a modified version to support networks with the *Relu* activation function.

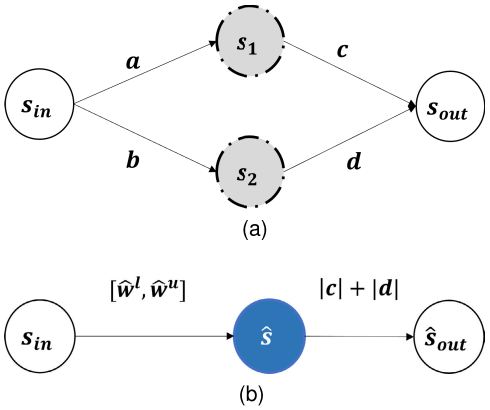


Fig. 4. Example explaining the main idea of the proposed approach, where the incoming weight to the abstract node $\hat{s}$ is: $\hat{w}^l = \min\{\text{sign}(c) \times a, \text{sign}(d) \times b\}$ and $\hat{w}^u = \max\{\text{sign}(c) \times a, \text{sign}(d) \times b\}$. (a) Original network. (b) Abstract network.

A. Model Reduction for Networks With Odd Activation Functions

Before discussing the details of the approach, let us first recall the definition of an odd function.

Definition 5: A function $g : \mathbb{R} \rightarrow \mathbb{R}$ is odd if

$$\forall x \in \mathbb{R} : g(-x) = -g(x).$$

Examples of some used odd activation functions are *Tanh*, *Bipolar Sigmoid*, *LeCun's Tanh*, the *identity function*, *Hard Tanh*, and so on [55].

The method consists of merging a set of nodes $\hat{S} \subseteq S_i$, where S_i is the set of nodes of hidden layer l_i, and replacing the subset $\hat{S}$ by an abstract node $\hat{s}$ that is determined by its weights as follows.

1) Incoming weights:
$\forall s_{i-1,k} \in S_{i-1}$, $w(s_{i-1,k}, \hat{s}) = [\hat{w}^l_k, \hat{w}^u_k]$, such that

$$\begin{cases} \hat{w}^l_k = \min\limits_{s_i \in \hat{S}, s_{i+1,j} \in S_{i+1}} \{\text{sign}(w(s_i, s_{i+1,j})) \times w(s_{i-1,k}, s_i)\} \\ \hat{w}^u_k = \max\limits_{s_i \in \hat{S}, s_{i+1,j} \in S_{i+1}} \{\text{sign}(w(s_i, s_{i+1,j})) \times w(s_{i-1,k}, s_i)\}. \end{cases} \tag{8}$$

2) Outgoing weights

$$\forall s_{i+1,j} \in S_{i+1}, w(\hat{s}, s_{i+1,j}) = \sum_{s_i \in \hat{S}} |w(s_i, s_{i+1,j})| \tag{9}$$

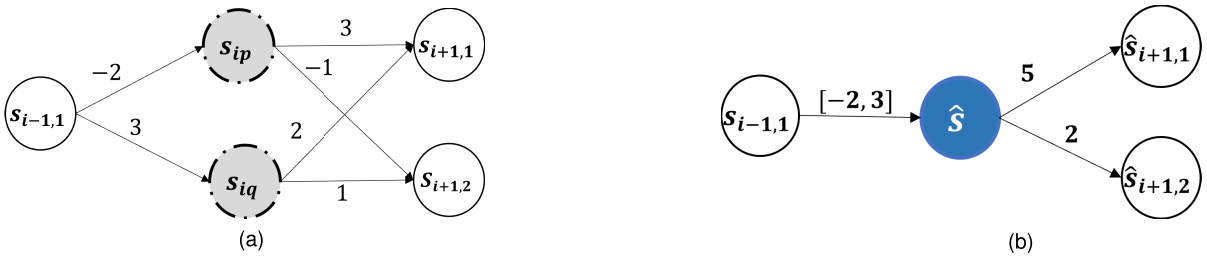


Fig. 5. Example of the abstraction method applied on two neurons of a hidden layer l_i. For $v(s_{i-1,1}) = 1$, then we have $v(s_{i+1,1}) = 0$ and $v(s_{i+1,2}) = 5$, $v(\hat{s}_{i+1,1}) = [-10, 15]$ and $v(\hat{s}_{i+1,2}) = [-4, 6]$. Hence, the overapproximation is fulfilled, since $v(s_{i+1,k}) \in v(\hat{s}_{i+1,k})$ for $k = 1, 2$. (a) Network N: s_{ip} and s_{iq} are to be merged. (b) Obtained abstract model $\bar{N}$.

Algorithm 1 Pseudo-Code of INNAbstract

```
procedure ABSTRACT(N, selectStrategy)
    N̄ ← duplicate(N)  ▷ Initialize the abstract network
    for i ← 1 to n − 1 do  ▷ Explore all layers
        P_i ← SelectNodes(N̄, i, selectStrategy)
        AbstractOneLayer(N̄, i, P_i)
    end for
    return N̄
end procedure
```

Algorithm 2 Pseudo-Code of the Abstraction Method Applied on One Hidden Layer l_i

```
procedure ABSTRACTONELAYER(N, i, P_i )  ▷ P_i a set containing the partitions
    for every subset Ŝ in P_i do
        create a node ŝ
        for s_{i−1,k} ∈ S_{i−1} do
            calculate ŵ(s_{i−1,k}, ŝ) using Equation 8
        end for
        for s_{i+1,j} ∈ S_{i+1} do
            calculate ŵ(ŝ, s_{i+1,j}) using Equation 9
        end for
        for s_i ∈ Ŝ do
            remove s_i
        end for
        add ŝ to S_i
    end for
end procedure
```

where $|.|$ is the absolute value, and sign is the sign function defined as: $\text{sign} : \mathbb{R} \to \{-1, 1\}$

$$\text{sign}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{10}$$

The main steps of the abstraction method (INNAbstract) are presented in Algorithm 1. The algorithm takes as input two parameters: the original network N and the nodes selection strategy. First, a copy of the original network denoted $\bar{N}$ is generated. Next, for each layer, a partition P_i of disjoint subsets of S_i is created using the *SelectNodes* procedure. Then the procedure *AbstractOneLayer*, presented in Algorithm 2, is applied for merging each subset $\hat{S}$ in the partition P_i and calculating the weights of the obtained abstract nodes. Finally, the algorithm returns the abstract network $\bar{N}$. *SelectNodes* may correspond to a random selection, or may implement some heuristics as will be discussed in Section IV-C.

An example of executing Algorithm 1 is given in Fig. 5. In this example, we consider that the network uses the identity activation function ($\alpha(x) = x$). The application of Algorithm 1 on a network N allows the generation of an abstract network that is an overapproximation of N. Based on Algorithm 1, the following results are stated and proved.

Proposition 1: Let N be an NN. Assume that the activation function of N is odd and monotone. Let $\hat{S}_i = (S_i \backslash \hat{S}) \cup \{\hat{s}\}$ be the set of nodes in layer i after applying the abstraction procedure on this layer. Then, we have

$$\forall v(S_{i-1}),\ v(S_{i+1}) \subseteq v(\hat{S}_{i+1}).$$

Proposition 2: Let N be an NN. Assume that the activation function of N is odd and monotone. Let $\bar{N}$ be the abstract network of N obtained using Algorithm 1. Then, we have

$$\forall x \in \mathbb{R}^{|S_0|}, N(x) \subseteq \bar{N}(x).$$

The proof of Proposition 1 is given in Appendix . The proof of Proposition 2 is straightforward and can be derived from the proof of Proposition 1 by propagating the abstraction from layer l_i to l_{i+1}, up to l_{n-1}.

B. Model Reduction Method for Relu-NN

The piecewise linear function *Relu* [see (2)] is monotone; however, it is not odd, thus the model reduction method presented in Section IV-A is not straightforwardly applicable on *Relu*-NN. Concretely, applying the approach presented in Section IV-A on a *Relu*-NN does not guarantee the overapproximation, that is, the output of the original network may not be within the range of the output of the reduced network (see the counterexample in Fig. 6).

In this section, we present a relaxation of the method presented in Section IV-A, so it can support *Relu*-NN. The idea is to check whether the incoming weight to every node $s \in \hat{S}$ has the same sign as its corresponding outgoing weight, and if so, the lower bounds are calculated using the following equation instead of (8):[2]

$$\begin{cases} \hat{w}_k^l = \min\limits_{s_i \in \hat{S}, s_{i+1,j} \in S_{i+1}} \{w(s_{i-1,k}, s_i)\} \\ \hat{w}_k^u = \max\limits_{s_i \in \hat{S}, s_{i+1,j} \in S_{i+1}} \{\text{sign}(w(s_i, s_{i+1,j})) \times w(s_{i-1,k}, s_i)\}. \end{cases} \tag{11}$$

[2]The formula for calculating the upper bounds of the incoming weights is the same as NN with odd and monotone, that is, Equation 9.

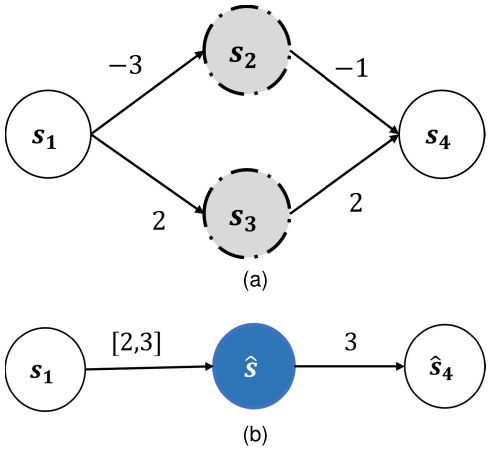


Fig. 6. Counterexample of applying Algorithm 2 on a *Relu*-NN. Assume that $v(s_1) = 2$, then $v(s_4) = 8$ and $v(\hat{s}_4) = [12, 18]$, hence $v(s_4) \notin v(\hat{s}_4)$. (a) Network N: s_2 and s_3 are to be merged. (b) Obtained abstract model $\bar{N}$.

Algorithm 3 Pseudo-Algorithm of the Abstraction Method Applied on One *Relu* Hidden Layer l_i

procedure ABSTRACT1RELULAYER(N, i, $\hat{S}$) ▷ N: the network, i : the layer's number
  for every subset $\hat{S}$ in P_i **do**
    create a node $\hat{s}$
    for $s_{i-1,k} \in S_{i-1}$ **do**
      calculate $\hat{w}(s_{i-1,k}, \hat{s})$ using Equation 8 and determine $w^*_{i+1,j}$
      if Equation 12 is true **then**
        update $\hat{w}(s_{i-1,k}, \hat{s})$ using Equation 11
      end if
    end for
    for $s_{i+1,j} \in S_{i+1}$ **do**
      calculate $\hat{w}(\hat{s}, s_{i+1,j})$ using Equation 9
    end for
    for $s_i \in \hat{S}$ **do**
      remove s_i
    end for
    add $\hat{s}$ to S_i
  end for
  replace all the scalar-weights w of layer l_i by an interval $[min(w, 0), w]$
end procedure

For two nodes $s_{i-1,k} \in S_{i-1}$ and $s \in \hat{S} \subseteq S_i$, we define $w^*_{i+1,j} \in \{w((s, s_{i+1,j})) : s_{i+1,j} \in S_{i+1}\}$ as the weight minimizing $\{\text{sign}(w(s, s_{i+1,j})) \times w(s_{i-1,k}, s)\}$. For a subset $\hat{S} \subseteq P_i$ of nodes to be abstracted, if all the concrete nodes $s \in \hat{S}$ satisfy in the following equation below, then the incoming weights of the abstract node $\hat{s}$ are calculated using (11)

$$\bigwedge_{s \in \hat{s}} \big(\text{sign}(w^*_{i+1,j}) = \text{sign}(w(s_{i-1,k}, s))\big). \tag{12}$$

The second step consists of replacing every scalar weight w of layer l_i by an interval weight $\hat{w} = [\min(w, 0), w]$. This is done at the end of the abstraction process of layer l_i and before switching to layer l_{i+1}. The main steps of abstracting a set of nodes of a hidden layer l_i of a *Relu*-NN are summarized in Algorithm 3. The algorithm can be applied on multiple layers as presented in Algorithm 4.

Algorithm 4 Pseudo-Algorithm of INNAbstract for *Relu*-NN

procedure ABSTRACT(N, *selectStrategy*)
  $\bar{N} \leftarrow duplicate(N)$ ▷ Initialize the abstract network
  for $i \leftarrow 1$ to $n - 1$ **do** ▷ Explore all layers
    $\hat{S} \leftarrow SelectNodes(\bar{N}, i, selectStrategy)$
    $Abstract1ReluLayer(\bar{N}, i, \hat{S})$
  end for
  return $\bar{N}$
end procedure

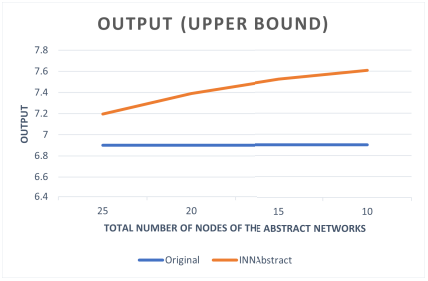


Fig. 7. Output range results on *Tanh*-NN.

In the following, we state the main results for the abstraction of *Relu*-NN. Due to the limit on the number of pages, the proof of Proposition 3 is omitted from the article. We should mention that proof of Proposition 4 can be derived from that of Proposition 3 by propagating the abstraction from layer l_i to l_{i+1}, up to l_{n-1}.

Proposition 3: Let N be an NN. Assume that *Relu* is the activation function of N. Let $\hat{S}_i = (S_i \backslash \hat{S}) \cup \{\hat{s}\}$ be the set of nodes upon applying the abstraction on S_i. Then, we have

$$\forall v(S_{i-1}), \quad v(S_{i+1}) \subseteq v(\hat{S}_{i+1}).$$

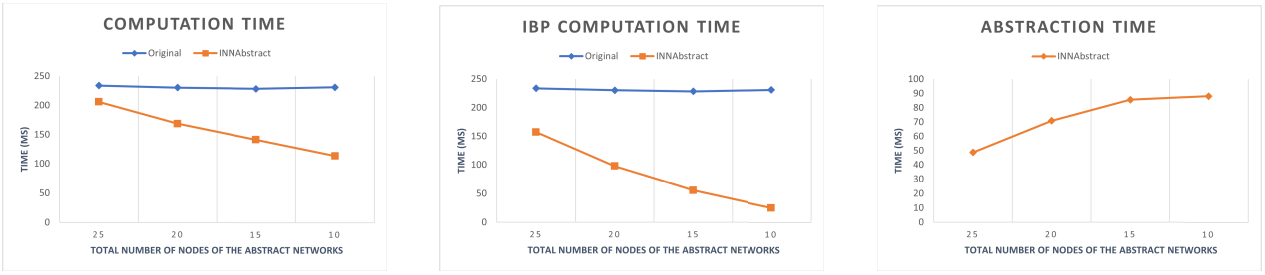
Proposition 4: Let N be an NN. Assume that *Relu* is the activation function of N. Let $\bar{N}$ be the abstract network of N obtained using Algorithm 4. Then, we have

$$\forall x \in \mathbb{R}^{|S_0|}, \quad N(x) \subseteq \bar{N}(x).$$

C. Heuristic for Nodes Selection

The precision of the reduced model depends mostly on the features of the merged nodes. One of the most important features is the sign of the outgoing weights since we use this feature to calculate the weights of the abstract node. Therefore, we propose a heuristic to select the set of nodes to be merged on each layer based on the sign of their outgoing weights. Recall that the sign of outgoing weights of the initial network affects the sign of both incoming and outgoing weights of the abstract network after merging (multiplying incoming weights by this sign, and taking the absolute value for outgoing weights). Therefore, intuitively, it seems more likely that the abstract network will be closer to the original one when the outgoing weights are positive.

In our approach, we propose a heuristic for ***nodes selection*** giving priority to nodes having positive outgoing weights, that



(a) (b) (c)

Fig. 8. Total computation time, IBP, and abstraction time results on *Tanh*-NN. (a) Total computation time. (b) IBP computation time. (c) Abstraction time.

Algorithm 5 Nodes Selection Strategy Based on Heuristics

1: **procedure** NODESELECTIONHEURIS(N, i, nbAbst)
2: $W \leftarrow$ the outgoing weight matrix of layer l_i
3: $W_{sign} \leftarrow \text{sign}(W)$ ▷ calculate the sign matrix of W
4: $sum_{rows} \leftarrow sum(W_{sign}, axis = 1)$ ▷ Sum by row
5: $sort(sum_{rows})$
6: save the first $nbAbst$ nodes in a set $\hat{S}$
7: return $\hat{S}$
8: **end procedure**

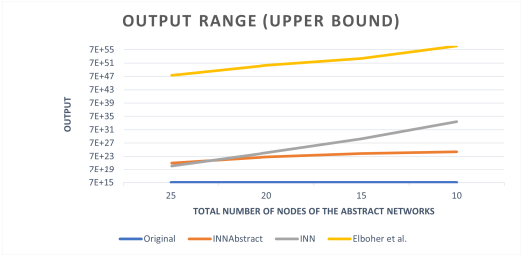


Fig. 9. Output range results on *Relu*-NN.

is, the nodes that have more positive weights are to be merged first. This is done by calculating the *sign* of the outgoing weight matrix of the hidden layer l_i, summing the obtained matrix by rows, and then sorting the obtained list. Finally, the selection of the nodes to be merged is done by picking nbAbst nodes from the top of this ordered list, where nbAbst is a parameter of NodeSelectionHeuris procedure that is summarized in Algorithm 5. Notice that NodeSelectionHeuris is used within the procedure *SelectNodes* in Algorithms 1 and 4 if selectStrategy = heursitics.

V. EXPERIMENTS AND EVALUATION

For evaluation purposes, we implemented the approach as a Python software prototype. The implementation includes three main modules.

1) `Network_Reader`: reads the DNN model and the inputs' constraints. The tool supports both NNET and ONNX formats.
2) `Abstractor`: a proof-of-concept implementation of the presented approach, which is used to generate abstract networks from the initial one brought by the first module (Network_Reader).
3) `Analyzer`: we implemented and adjusted IBP method[3] [34] to support INN, and we used it to calculate the output range of the network.

The remaining of the section is dedicated to the empirical evaluation of our approach. It includes a description of the evaluation setup and a summary of the obtained results. A discussion of the obtained results is provided in Section V-C.

In fact, we conducted a set of experiments on networks with a *Tanh* activation function, which is odd and monotone, and also on *Relu*-NN. The evaluations include the following experimental parts.

1) *Evaluation on Random NN:* applying model reduction on randomly generated *Tanh*-NN and *Relu*-NN. To examine the impact of the network's size on the abstraction method, we generate a set of abstract networks with different sizes.
2) *Evaluation on Benchmarks:* applying model reduction on two benchmarks: MNIST[4] and ACAS Xu [21] for *Tanh*-NN and *Relu*-NN, respectively.
3) *Heuristic's Improvement:* applying our method with two different node selection strategies: random and based on the proposed heuristic, to assess the improvement brought by the latter. A set of *Relu*-NN of different sizes and depths is used.

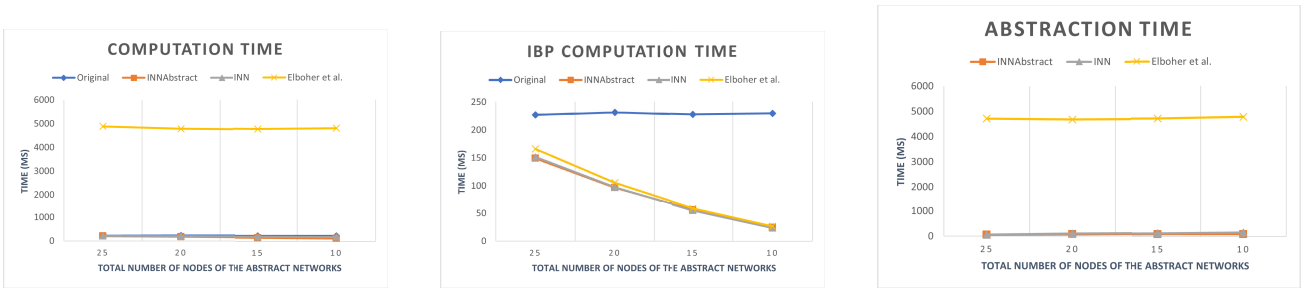
Additionally, we performed a comparison study between our method and two other abstraction techniques proposed in [46] and [45]. Since the latter support only the *Relu* activation function, we considered random *Relu*-NN and ACAS Xu benchmark networks. The obtained results are depicted iteratively in various tables and graphs throughout the article. In the sequel, we refer to the obtained results using our abstraction method, the method introduced in [46], and the method presented in [45], as *INNAbstract*, *Elboher* et al., and *INN*, respectively. The results obtained on the original initial network are denoted by *original*.

A. Experiments on Random NNs

In this first part, we perform an evaluation study on a set of randomly generated networks with the odd monotone activation function *Tanh*, and the *Relu* activation function.

[3]IBP is an algorithm to calculate the output range of NNs by propagating the inputs (intervals) through the network's layers.

[4]A database of handwritten digits, available in http://yann.lecun.com/exdb/mnist/



(a) (b) (c)

Fig. 10. Total computation time, IBP, and abstraction time results on *Relu*-NN. (a) Total computation time. (b) IBP computation time. (c) Abstraction time.

TABLE I

OUTPUT RESULTS ON *Tanh*-NN, $L = 20$ LAYERS

	Output range (UB)		Output range's width	
Nb. Nodes	Original	INNAbstract	Original	INNAbstract
25	6.90	7.19	13.78	14.39
20	6.90	7.39	13.78	14.78
15	6.90	7.52	13.78	15.05
10	6.90	7.61	13.78	15.21

TABLE II

ABSTRACTION TIME AND TOTAL COMPUTATION TIME ON *Tanh*-NN, $L = 20$ LAYERS

	Abstraction time (ms)	Total computation time (ms)	
Nb. Nodes	INNAbstract	Original	INNAbstract
25	48.77	233.66	205.86
20	70.91	230.21	168.47
15	85.59	228.23	141.19
10	88.07	230.97	113.64

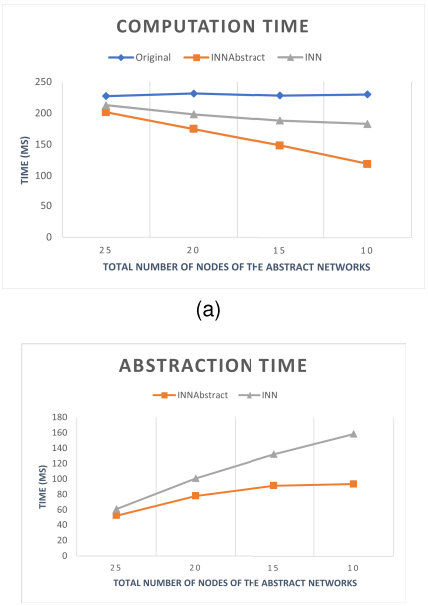
The generated networks have five inputs, $L = 20$ hidden layers with $n_l = 30$ nodes per layer, and one output. We apply our method to produce abstract networks of different sizes (different layer sizes): $\hat{n}_l \in \{25, 20, 15, 10\}$ as presented in Table I. Next, for an input $X^* \in [-1, 1]$, we apply a perturbation $\epsilon \in [0.01, 0.1]$ to generate input bounds $[X^* - \epsilon, X^* + \epsilon]$, then, we compute the output bounds of the original model as well as of the abstract networks using the IBP algorithm.

Over 50 runs, we calculate the average of the output's upper bound, the abstraction time, the IBP computation time, and the total computation time which represents the sum of the abstraction time and output range computation time on the abstract networks. The obtained results on *Tanh*-NN are presented in Tables I and II. Furthermore, we depict these results graphically in Figs. 7 and 8. For *Relu*-NN, we compare the results to the ones obtained using the abstraction method of Elboher et al. [46] and INN [45]. Finally, Figs. 9–11 provide more details about these results. A thorough discussion of the obtained results is provided in Section V-C.

B. Experiments on MNIST and ACAS Xu Benchmarks

1) MNIST: In this second series of experiments on *Tanh*-NN, we use the ONNX networks trained on the dataset of handwriting digits, namely MNIST. The networks are available on the VNN 2020 Competition's GitHub repository.[5]

[5]Available in: https://github.com/verivital/vnn-comp/tree/master/2020/NLN/benchmark/mnist/tanh

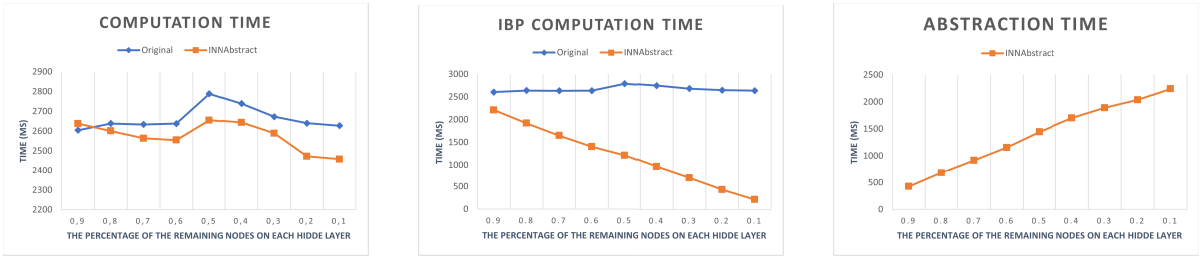


(a) (b)

Fig. 11. Total computation time and the abstraction time for INNAbstract and INN on *Relu*-NN. (a) Total computation time. (b) Abstraction time.

The networks have 784 inputs and 10 outputs; however, the size and the number of hidden layers may change. In this work, we perform our experiments on the network *tansig_200_100_50_onnx.onnx* which has three hidden layers with 200, 100, and 50 nodes, successively. We select an image from the MNIST dataset and then we apply a perturbation (ϵ) on each pixel using the L_∞ norm [29]. Next, we calculate the output range of the network using the IBP algorithm. Similarly, using the same input image and perturbation ϵ, we calculate the average of the output range of the reduced models over 50 runs. We also calculate the average of the IBP computation time for the original and reduced models and the average of the abstraction time.

Figs. 12 and 13 summarize the obtained results. Since the hidden layers do not have the same number of nodes, the X-axis in Figs. 12 and 13 represents the percentage of the remaining nodes after abstraction. For instance, the value 0.9 means that the number of the nodes in each hidden layer of the abstract network has been reduced by 10% (180, 90, and 45, successively).



(a) (b) (c)

Fig. 12. Total computation time, IBP, and abstraction time results on MNIST *Tanh*-NN. (a) Total computation time. (b) IBP computation time. (c) Abstraction time.

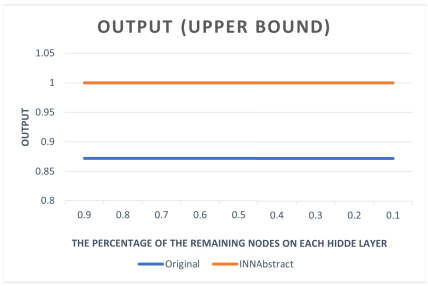


Fig. 13. Output range results on MNIST *Tanh*-NN.

2) ACAS Xu: We consider the ACAS Xu networks as a case study to evaluate our method on realistic *Relu*-NN. The benchmark ACAS Xu is an airborne-collision avoidance system that has 45 networks [5]. These networks are trained to provide advisories for an unmanned aircraft (ownership) to avoid a collision with another aircraft (intruder). Each network has seven inputs of sensor measurements, such as the speed of the ownership and the intruder, the distance between them and their directions, and so on. The network processes these inputs and returns five outputs that are scores of a possible advisory for the ownership: clear-of-conflict, strong/weak turn to the left, or strong/weak turn to the right.

For a given network and inputs' constraints of a specific property,[6] we compute the corresponding output range on the original network as well as on the generated abstract networks using the three methods, namely, our method (INNAbstract), INN [45], and the technique presented in Elboher et al. [46]. Over 50 runs, and for each network, we calculate the output upper bound average, the average of the IBP computation time, the abstraction time, and the total computation time. The results are shown in Figs. 14–16.

Through the performed experiments, we can observe that the computation time values obtained using Elboher et al.'s method were excessively high, making it challenging to compare the original network with the abstract networks generated using the INNAbstract and INN methods. For a comprehensive comparison, we include Figs. 11 and 15 to display the computation time values for the INNAbstract, INN, and original network on the *Relu*-NN and ACAS Xu benchmark, respectively.

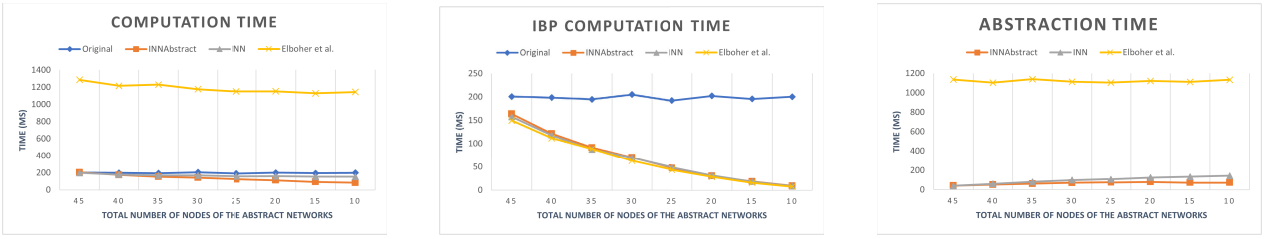
C. Discussion

As illustrated in Figs. 8, 10, 11, 12, 14, and 15, decreasing the size of the network (i.e., reducing the number of the remaining nodes) by performing more abstraction leads to a linear increase in the abstraction time and a significant decrease in the IBP computation time. As a result, the total computation time, defined as the sum of the abstraction time and the IBP computation time, is also decreased. Additionally, the IBP algorithm is always faster on the abstract networks than on the original ones. With the exception of the abstract networks that are generated using Elboher et al., the total computation time on the abstract networks of INNAbstract and INN is consistently lower than the computation time on the original ones, especially when the abstraction degree is significant [see Figs. 11(a) and 15(a)]. Moreover, the computation time of INNAbstract is always smaller than that of INNs. For example, in Fig. 15(a) when the number of the remaining nodes is 10, which represents a reduction of the original network's size by 80%, the whole process on the abstract models obtained using INNAbstract (abstraction + IBP) is two times faster than the IBP on the original model.

Moreover, all three model reduction methods exhibit similar IBP computation times, which decrease as the size of the abstract networks decreases. This can be explained by the fact that the IBP algorithm is performed on networks of the same size (same remaining nodes). However, by comparing the abstraction time of INNAbstract to the two other methods [shown in Figs. 10(c) and 14(c)] and more specifically to INNs [see Figs. 11(b) and 15(b)], we can observe that INNAbstract is faster. Namely, when considering the same number of abstract neurons, INNAbstract generates abstract networks in less time compared to the other methods. We can also observe that the Elboher et al. method is computationally expensive compared to INNAbstract and INN. This can be attributed to the preprocessing phase, which involves classifying neurons into different categories and potentially splitting numerous neurons. As a result, the network to be abstracted may have a larger size than the original one, which leads to increasing the abstraction time, significantly. Additionally, the abstraction time using our method increases linearly, and hence much more slowly compared to INNs [see Figs. 11(b) and 15(b)]. Moreover, the total computation time of INNAbstract is less than the total computation time of both methods. This clearly illustrates the efficiency of our method, which can handle larger networks without a significant increase in abstraction time.

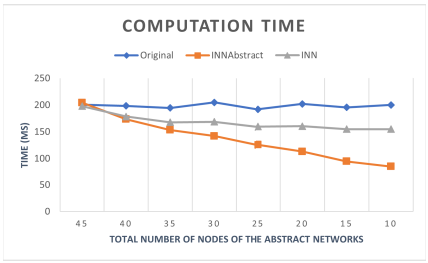
Figs. 7, 9, 13, and 16 show the results of the output range of the different abstractions. A general observation from Figs. 7, 9, and 16 shows that decreasing the size of the abstract network, which leads to less precise network, increases the

[6]In this article, we used the property ϕ_5 presented in [21], and the network ACASXU_experimental_v2a_1_1.nnet.

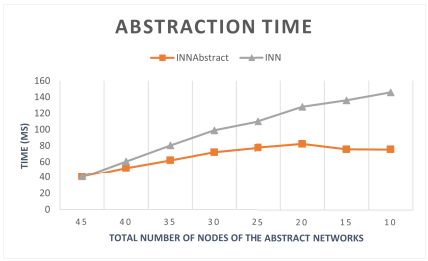


(a) (b) (c)

Fig. 14. Total computation time, IBP, and abstraction time results on ACAS Xu *Relu*-NN. (a) Total computation time. (b) IBP computation time. (c) Abstraction time.



(a)



(b)

Fig. 15. Total computation time and abstraction time results of INNAbstract and INN on ACAS Xu *Relu*-NN. (a) Total computation time. (b) Abstraction time.

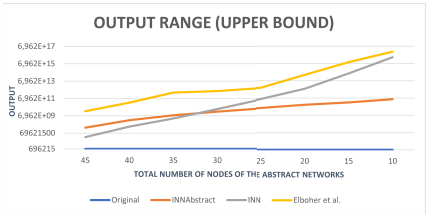


Fig. 16. Output range results on ACAS Xu *Relu*-NN.

upper bound of the output. Furthermore, when focusing on the output range results of the three abstraction methods, we can see that the output range's upper bounds of abstract networks obtained using INNAbstract are generally smaller than the output ranges of those obtained using Elboher et al. [46] and INNs [45]. Specifically, INNAbstract significantly outperforms Elboher et al. considering the different sizes of the abstract networks. Although the INN abstraction method initially shows slightly better performance when only a few neurons are merged, its performance drops drastically as the abstraction becomes more intensive (more neurons are merged), where the output range of the obtained abstract networks increases rapidly. In contrast, we can notice that the output range on abstract networks obtained using INNAbstract exhibits a slow increase with the number of remaining nodes, and INNAbstract outperforms the INN method when significant abstraction is applied.

In Fig. 13, we can observe that the output range of the abstract models is fairly close to the output range of the original one. Interestingly, the output's upper bound of different abstract networks does not change while varying the number of the remaining nodes. This is potentially related to the behavior of *Tanh* with large (resp. small) values, where the slope is almost null and the output of *Tanh* is basically 1 (resp. -1).

The main conclusion through these series of experiments is that: while the IBP computation time decreases, the abstraction time and the output range related to INNAbstract models exhibit a linear and slow increase with the number of merged nodes. Furthermore, we can observe that the total computation time (including both the abstraction and output range computation) on the abstract networks obtained using INNAbstract is consistently lower than the computation time on the original networks. Furthermore, according to the conducted experiments and the obtained results, INNAbstract significantly outperforms Elboher et al. [46] and INN [45] approaches in the sense that it produces tighter output bounds in less time. This highlights the effectiveness of INNAbstract in reducing the computation time, thus allowing for faster analysis and operations on NN.

It is worth recalling that the IBP is a very fast algorithm and allows for calculating an overapproximation of the outputs. Thus, the gain in precision and computation time will be more significant with other more precise methods.

D. Heuristic's Improvement

The objective of this section is to assess the performance of our model reduction method when combined with the proposed heuristic for node selection (Algorithm 5). To achieve this, we conduct a series of experiments on a set of randomly generated *Relu*-NN using two node selection strategies: random selection and heuristic-based selection. Namely, we generate various networks with different sizes by varying the number of layers L and the number of nodes on each layer n_l to examine their impact on the heuristic's performance. More details about the generation of these networks are given in Section V-A. Next, we calculate the output range of the obtained abstract networks and we determine the improvement rate (I_{rate}) in terms of precision using the proposed heuristic. This is done by calculating the percentage of the difference between the output range with respect to the output range of the abstraction method with the random nodes selection using the following formula:

$$I_{\text{rate}} = \frac{U_{\text{rand}} - U_{\text{heuris}}}{U_{\text{rand}}} \times 100$$

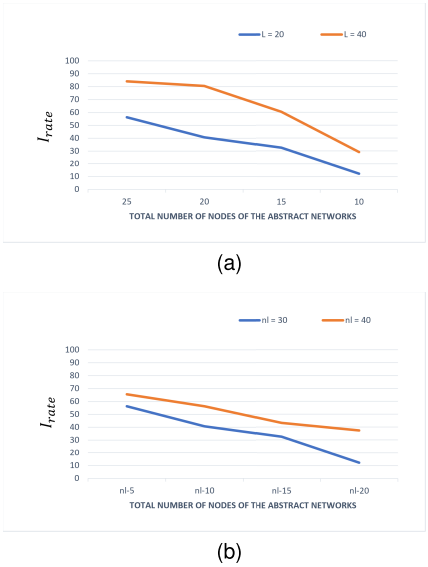


Fig. 17. Graph representing the improvement of the output range using the proposed heuristic. Results on networks with different numbers of (a) layers, $L = 20$ and $L = 40$, and (b) nodes per layer, $n_l = 30$ and $n_l = 40$.

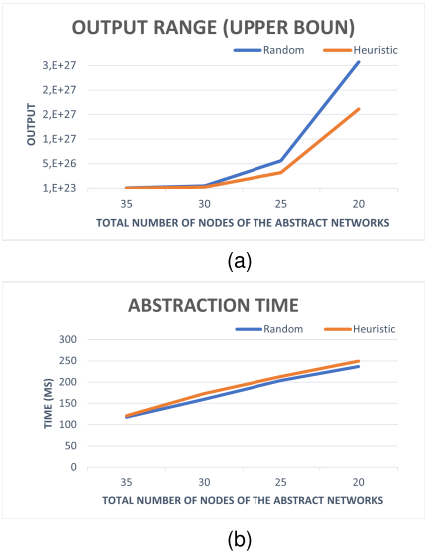


Fig. 18. Output range and abstraction time results on *Relu*-NN with random and heuristics-based selection strategy. (a) Output range (upper bound). (b) Abstraction time.

where U_{rand} and U_{heuris} are the upper bounds of the abstract network using INNABstract with the random and heuristic-based strategy for node selection, respectively.

Fig. 17 depicts the obtained results, where Fig. 17(a) represents the results on networks with different layers number ($L = 20$, $L = 40$) and the same number of nodes on hidden layers ($n_l = 30$), and Fig. 17(b) represents the results of networks whose layers number is set to 20 ($L = 20$) and have different hidden layer sizes ($n_l = 30$, $n_l = 40$).

As shown by the results presented in Fig. 17, the combination of the proposed heuristic with INNAbstract can enhance the precision of the generated abstract networks. Moreover, the performance of the heuristic depends on the number of layers and hidden nodes in the initial network. Specifically, the proposed heuristic performs better with large networks, since the population from where the nodes are selected is large, thus the algorithm has more options.

To illustrate the improvement in terms of precision through the proposed heuristics, we chose the configurations $L = 20$ and $n_l = 40$ to generate a random network of 20 layers, with 40 nodes in each hidden layer. We then compute the output range and the abstraction time of the resultant abstract networks, employing both random and heuristic selection strategies.

As depicted in Fig. 18(a), the output range obtained using INNAbstract in combination with the proposed heuristic is tighter compared to the output range obtained using INNAbstract with a random selection of nodes. Furthermore, the improvements become more significant when more nodes are merged. However, it is important to note that achieving these improvements comes at the cost of increased abstraction time, as illustrated in Fig. 18(b).

Finally, it is worth noticing that we have conducted the same experiments on *Tanh*-NN; however, the improvement rate remains negligible. This is mainly due to the nature of the *Tanh* function (illustrated by its hyperbolic behavior) which tends to push the output value to either end of the curve (1 or -1) due to its S-like shape, that is, in the region close to zero, if we slightly change the input value, the respective changes in the output are very large and vice versa.

VI. Conclusion

In this work, we presented INNAbstract, a model reduction method for NNs. The approach consists of merging nodes of the same layer and provides a formula to calculate the new weights ensuring that the output range of the original model is always included within that of the abstract one. The proposed approach supports NNs with *Relu* and odd-monotone activation functions, including *Tanh* and Bipolar Sigmoid. With the aim of enhancing the performance of the approach, we proposed a heuristic for node selection based on some features of the network's nodes. Furthermore, we carried out a series of experiments using randomly generated NNs in addition to some benchmarks to assess the performance of the proposed method. We also compared the performance of our method to other relevant works from the literature.

The results of the experimental study show that INNAbstract can effectively be used to reduce the size of large networks, leading to reduced computation costs, while providing formal guarantees that the generated abstract networks overapproximate the original ones. Additionally, the proposed heuristic allows for significantly improving the precision of the abstraction method. From the obtained results, the manipulation operations on abstract networks are clearly faster. However, there is a tradeoff between abstraction, precision, and computation time. Furthermore, on the basis of the conducted experiments, INNAbstract outperforms the other relevant abstraction methods presented in Elboher et al. [46] and Prabhakar and Afzal [45] in terms of both precision and computation time.

We plan to pursue this work in different directions. First, we want to integrate our model reduction method with

more precise NN-bound propagation and verification methods. Moreover, we will investigate the abstraction approach to support additional activation functions. Finally, we intend to explore more heuristics for node selection toward generating more precise abstract networks.

APPENDIX

We provide in this section the formal proof of the soundness of our proposed method for NNs with odd and monotone activation functions, namely Proposition 1. We would like to note that, due to page limitations, the proof of Proposition 3 is omitted from this version of the article.

PROOF OF PROPOSITION 1

Proposition 1: Let N be an NN. Assume that the activation function of N is odd and monotone. Let $\hat{S}_i = (S_i \backslash \hat{S}) \cup \{\hat{s}\}$ Be the set of nodes after applying the abstraction on S_i. Then, we have

$$\forall v(S_{i-1}),\, v(S_{i+1}) \subseteq v(\hat{S}_{i+1}).$$

■

In Proposition 1, $v(S_{i+1}) \subseteq v(\hat{S}_{i+1}$ means that $v(s_{i+1,j}) \in v(\hat{s}_{i+1,j}), \forall j \in \{1, 2, \ldots, m|\}$, where $m = |S_{i+1}| = |\hat{S}_{i+1}|$. Proposition 1 considers merging many nodes on a hidden layer l_i, and for the sake of simplicity, we provide the proof of merging two nodes. The generalization of this proof by considering multiple nodes can be done following the same steps.

Assume that we want to merge two hidden nodes s_1 and s_2 of a hidden layer l_i, such that l_{i-1}, l_i, and l_{i+1} all have an odd and monotone activation function, and let $\hat{s}$ denotes the obtained abstract node.

We denote by a_k (resp. b_k) the incoming weights of s_1 (resp. s_2) from node $s_{i-1,k} \in S_{i-1}$

$$\begin{cases} a_k = w(s_{i-1,k}, s_1) \\ b_k = w(s_{i-1,k}, s_2) \end{cases}$$

and we denote by c_j (resp. d_j) the outgoing weight of s_1 (resp. s_2) for each node $s_{i+1,j} \in S_{i+1}$, that is,

$$\begin{cases} c_j = w(s_1, s_{i+1,j}) \\ d_j = w(s_2, s_{i+1,j}). \end{cases}$$

The weights of the remaining edges connecting other nodes than s_1 and s_2 ($s \in S_i \setminus \{s_1, s_2\}$) to each node $s_{i+1,j} \in S_{i+1}$ are defined as follows:

$$w(s, s_{i+1,j}) = w_{s,s_j}.$$

We denote by $\hat{s}$ the obtained abstract node after merging s_1 and s_2 using Algorithm 2. The new abstract layer is denoted as $\hat{S}_i = \hat{s} \cup \{S_i \setminus \{s_1, s_2\}\}$. For clarity, we use v_k instead of $v_{i-1,k}$. Since the activation function α is monotone and odd, that is, $\forall x \in \mathbb{R} : \alpha(-x) = -\alpha(x)$ and

$$\forall x_1, x_2 \in \mathbb{R} : x_1 \leq x_2 \implies \alpha(x_1) \leq \alpha(x_2).$$

For each $j \in \{1, 2, \ldots, |S_{i+1}|\}$, we have

$$\begin{aligned} v_{i+1,j} &= \alpha(z_{i+1,j}) \\ &= \alpha\left(c_j \times \alpha\left(\sum_{k=1}^{m} a_k v_k\right) + d_j \times \alpha\left(\sum_{k=1}^{m} b_k v_k\right) \right. \\ &\qquad \left. + \sum_{s \in S_i \backslash s1,s2} w_{s,s_j} v(s)\right) \\ \hat{v}_{i+1,j} &= \alpha(\hat{z}_{i+1,j}) \\ &= \alpha\left((|c_j| + |d_j|) \times \alpha\left(\sum_{k=1}^{m} \hat{w}_k v_k\right)\right. \\ &\qquad \left. + \sum_{s \in S_i \backslash s1,s2} w_{s,s_j} v(s)\right) \end{aligned}$$

where $z_{i+1,j}$ (resp. $\hat{z}_{i+1,j}$) is the value of $s_{i+1,j}$ (resp. $\hat{s}_{i+1,j}$) before applying the activation function α, and $m = |S_{i-1}|$ is the number of nodes of layer l_{i-1} and $\hat{w}_k = [\hat{w}_k^l, \hat{w}_k^u]$ is the abstract weight connecting each node $s_k \in S_{i-1}$ to the abstract node $\hat{s} \in \hat{S}_i$ as defined in Algorithm 2. The main results is to prove that $v_{i+1,j} \in \hat{v}_{i+1,j}$.

By definition of $\hat{w}_k$, we have

$$\begin{cases} \hat{w}_k^l \leq \text{sign}(c_j) a_k \leq \hat{w}_k^u \\ \hat{w}_k^l \leq \text{sign}(d_j) b_k \leq \hat{w}_k^u \end{cases}$$

$\forall k = \{1, 2, \ldots, m\}, \forall j \in \{1, 2, \ldots, n\}$.

Thus, we have

$$\begin{cases} \hat{w}_k^l v_k \leq \text{sign}(c_j) a_k v_k \leq \hat{w}_k^u v_k, & \text{if } v_k \geq 0 \\ \hat{w}_k^u v_k \leq \text{sign}(c_j) a_k v_k \leq \hat{w}_k^l v_k, & \text{Otherwise.} \end{cases}$$

Two cases need to be proved, $v_k \geq 0$ and $v_k < 0$. We provide below a detailed proof for the case $v_k \geq 0$. The proof of the second case, that is, $v_k < 0$, is omitted and it is done similarly by just inverting the lower and upper bounds of $\hat{w}_k$ when multiplying it by v_k.

Assume that $v_k \geq 0$, multiplying the aforementioned inequality by $\text{sign}(c_j)$, we obtain

$$\begin{cases} \text{if } c_j \geq 0: \\ \quad \text{sign}(c_j)\hat{w}_k^l v_k \leq \text{sign}(c_j)\text{sign}(c_j) a_k v_k \leq \text{sign}(c_j)\hat{w}_k^u v_k \\ \text{Otherwise:} \\ \quad \text{sign}(c_j)\hat{w}_k^u v_k \leq \text{sign}(c_j)\text{sign}(c_j) a_k v_k \leq \text{sign}(c_j)\hat{w}_k^l v_k. \end{cases}$$

The proof contains two parts: $c_j \geq 0$ and $c_j < 0$. Here, we provide details of the proof for the first case: $\text{sign}(c_j) \geq 0$. The proof of the other case ($\text{sign}(c_j) < 0$) can be performed following the same logic, using the fact that α is odd, so $\alpha(\text{sign}(c_j) \times x) = \text{sign}(c_j) \times \alpha(x)$, and that we have $\text{sign}(c_j) \times c_j = |c_j|$.

For $\text{sign}(c_j) \geq 0$ and by taking the sum over all k, we have

$$\sum_{k=1}^{m} \text{sign}(c_j)\hat{w}_k^l v_k \leq \sum_{k=1}^{m} a_k v_k \leq \sum_{k=1}^{m} \text{sign}(c_j)\hat{w}_k^u v_k.$$

And since α is monotone increasing, we have

$$\alpha\left(\sum_{k=1}^{m}\text{sign}(c_j)\hat{w}_k^l v_k\right) \leq \alpha\left(\sum_{k=1}^{m} a_k v_k\right) \leq \alpha\left(\sum_{k=1}^{m}\text{sign}(c_j)\hat{w}_k^u v_k\right).$$

Since α is odd, we have $\alpha(\text{sign}(c_j) \times x) = \text{sign}(c_j) \times \alpha(x)$, thus the previous inequality is equivalent to

$$\text{sign}(c_j)\alpha\left(\sum_{k=1}^{m}\hat{w}_k^l v_k\right) \leq \alpha\left(\sum_{k=1}^{m} a_k v_k\right) \leq \text{sign}(c_j)\alpha\left(\sum_{k=1}^{m}\hat{w}_k^u v_k\right). \quad (13)$$

Multiplying equality (13) by c_j implies

$$|c_j| \times \alpha\left(\sum_{k=1}^{m}\hat{w}_k^l v_k\right) \leq c_j \times \alpha\left(\sum_{k=1}^{m} a_k v_k\right) \leq |c_j| \times \alpha\left(\sum_{k=1}^{m}\hat{w}_k^u v_k\right). \quad (14)$$

Recalling that $\text{sign}(c_j) \times c_j = |c_j|$. The same reasoning remains valid when we replace a_k by b_k and c_j by d_j

$$|d_j| \times \alpha\left(\sum_{k=1}^{m}\hat{w}_k^l v_k\right) \leq d_j \times \alpha\left(\sum_{k=1}^{m} b_k v_k\right) \leq |d_j| \times \alpha\left(\sum_{k=1}^{m}\hat{w}_k^u v_k\right). \quad (15)$$

Summing inequalities (14) and (15) with the remaining incoming nodes values to $s_{i+1,j} \in S_{i+1} \setminus \{s_1, s_2\}$, we obtain

$$\begin{aligned}
&(|c_j| + |d_j|) \times \alpha\left(\sum_{k=1}^{m}\hat{w}_k^l v_k\right) + \sum_{s\in S_i \setminus s1,s2} w_{s,s_j} v(s) \\
&\leq c_j \times \alpha\left(\sum_{k=1}^{m} a_k v_k\right) + d_j \times \alpha\left(\sum_{k=1}^{m} b_k v_k\right) + \sum_{s\in S_i \setminus s1,s2} w_{s,s_j} v(s) \\
&\leq (|c_j| + |d_j|) \times \alpha\left(\sum_{k=1}^{m}\hat{w}_k^u v_k\right) + \sum_{s\in S_i \setminus s1,s2} w_{s,s_j} v(s). \quad (16)
\end{aligned}$$

And since α is monotone, by applying the activation α on the inequality (16), we obtain

$$\hat{v}(\hat{s}_{i+1,j})^l \leq v(s_{i+1,j}) \leq \hat{v}(\hat{s}_{i+1,j})^u.$$

Finally, we have

$$v_{i+1,j} \in \hat{v}_{i+1,j} \quad \forall j \in \{1, \ldots, n\}.$$

□

ACKNOWLEDGMENT

This research work contributes to the French collaborative project TASV (autonomous passengers service train), with Railenium, SNCF, Alstom Crespin, Thales, Bosch, and SpirOps. It was carried out in the framework of IRT Railenium, Valenciennes, France, and, therefore, was granted public funds within the scope of the French Program "Investissements d'Avenir."

REFERENCES

[1] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022.

[2] G. Litjens et al., "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.

[3] A. Bouguettaya, H. Zarzour, A. Kechida, and A. M. Taberkit, "Vehicle detection from UAV imagery with deep learning: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6047–6067, Nov. 2022.

[4] D. Ristić-Durrant, M. Franke, and K. Michels, "A review of vision-based on-board obstacle detection and distance estimation in railways," *Sensors*, vol. 21, no. 10, p. 3452, May 2021.

[5] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *Proc. IEEE/AIAA 35th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2016, pp. 1–10.

[6] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[7] K. Eykholt et al., "Robust physical-world attacks on deep learning visual classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1625–1634.

[8] C. Szegedy et al., "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Represent. (ICLR)* 2014, pp. 1–10.

[9] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[10] M. H. Meng et al., "Adversarial robustness of deep neural networks: A survey from a formal verification perspective," *IEEE Trans. Dependable Secure Comput.*, early access, May 30, 2022, doi: 10.1109/TDSC.2022.3179131.

[11] C. Urban and A. Miné, "A review of formal methods applied to machine learning," 2021, *arXiv:2104.02466*.

[12] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*, vol. 10. Cham, Switzerland: Springer, 2018.

[13] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*, vol. 185. Amsterdam, The Netherlands: IOS Press, 2009.

[14] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. 4th ACM SIGACT-SIGPLAN Symp. Princ. Program. Lang. (POPL)*, 1977, pp. 238–252.

[15] F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella, "Automated verification of neural networks: Advances, challenges and perspectives," 2018, *arXiv:1805.09938*.

[16] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *Proc. Int. Symp. Automated Technol. Verification Anal.* Cham, Switzerland: Springer, 2017, pp. 251–268.

[17] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," 2017, *arXiv:1706.07351*.

[18] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2017, pp. 3–29.

[19] G. Katz et al., "The marabou framework for verification and analysis of deep neural networks," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2019, pp. 443–452.

[20] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2010, pp. 243–257.

[21] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2017, pp. 97–117.

[22] F. Boudardara, A. Boussif, P. J. Meyer, and M. Ghazel, "A review of abstraction methods towards verifying neural networks," *ACM Trans. Embedded Comput. Syst.*, Aug. 2023, doi: 10.1145/3617508.

[23] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized CNN: A unified approach to accelerate and compress convolutional networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4730–4743, Oct. 2018.

[24] L. Gonzalez-Carabarin, I. A. M. Huijben, B. Veeling, A. Schmid, and R. J. G. van Sloun, "Dynamic probabilistic pruning: A general framework for hardware-constrained pruning at different granularities," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 8, 2022, doi: 10.1109/TNNLS.2022.3176809.

[25] F. Boudardara, A. Boussif, P.-J. Meyer, and M. Ghazel, "Interval weight-based abstraction for neural network verification," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Cham, Switzerland: Springer, 2022, pp. 330–342.

[26] B. J. Taylor, *Methods and Procedures for the Verification and Validation of Artificial Neural Networks*. New York, NY, USA: Springer, 2006.

[27] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–21.

[28] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods*. Cham, Switzerland: Springer, 2018, pp. 121–138.

[29] X. Huang et al., "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Comput. Sci. Rev.*, vol. 37, Aug. 2020, Art. no. 100270.

[30] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 3–18.

[31] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," in *Proc. ACM Program. Lang. (POPL)*, vol. 3, Jan. 2019, pp. 1–30.

[32] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–12.

[33] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.

[34] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 5, pp. 1821–1830, May 2021.

[35] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, Nov. 2018.

[36] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, 2018, pp. 1599–1614.

[37] H. Wu, A. Zeljić, G. Katz, and C. Barrett, "Efficient neural network analysis with sum-of-infeasibilities," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Cham, Switzerland: Springer, 2022, pp. 143–163.

[38] P. Henriksen and A. Lomuscio, "DEEPSPLIT: An efficient splitting method for neural network verification via indirect effect analysis," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, Aug. 2021, pp. 2549–2555, doi: 10.24963/ijcai.2021/351.

[39] J. Lan, Y. Zheng, and A. Lomuscio, "Tight neural network verification via semidefinite relaxations and linear reformulations," in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, no. 7, pp. 7272–7280.

[40] S. Wang et al., "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 29909–29921.

[41] A. De Palma et al., "Improved branch and bound for neural network verification via Lagrangian decomposition," 2021, *arXiv:2104.06718*.

[42] G. Singh, R. Ganvir, M. Püschel, and M. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[43] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev, "PRIMA: General and precise neural network certification via scalable convex hull approximations," in *Proc. ACM Program. Lang.*, vol. 6, 2022, pp. 1–33.

[44] P. Ashok, V. Hashemi, J. Křetínskỳ, and S. Mohr, "DeepAbstract: Neural network abstraction for accelerating verification," in *Proc. Int. Symp. Automated Technol. Verification Anal.* Cham, Switzerland: Springer, 2020, pp. 92–107.

[45] P. Prabhakar and Z. R. Afzal, "Abstraction based output range analysis for neural networks," in *Advances in Neural Information Processing Systems*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019.

[46] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2020, pp. 43–65.

[47] J. Liu, Y. Xing, X. Shi, F. Song, Z. Xu, and Z. Ming, "Abstraction and refinement: Towards scalable and exact verification of neural networks," 2022, *arXiv:2207.00759*.

[48] M. Sotoudeh and A. V. Thakur, "Abstract neural networks," in *Proc. Int. Static Anal. Symp.* Cham, Switzerland: Springer, 2020, pp. 65–88.

[49] P. Prabhakar, "Bisimulations for neural network reduction," in *Proc. Int. Conf. Verification, Model Checking, Abstract Interpretation*. Cham, Switzerland: Springer, 2022, pp. 285–300.

[50] T.-B. Xu and C.-L. Liu, "Deep neural network self-distillation exploiting data representation invariance," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 1, pp. 257–269, Jan. 2022.

[51] R. Mishra, H. P. Gupta, and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," 2020, *arXiv:2010.03954*.

[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[53] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[54] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, Dec. 1994.

[55] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*.

Fateh Boudardara received the B.Eng. degree from the High School of Computer Science, Algiers, Algeria, in 2015, and the master's degree in artificial intelligence and optimization from Erciyes University, Kayseri, Turkey, in 2019. He is currently pursuing the Ph.D. degree with the France's Railway Technological Research Institute, IRT Railenium, Valenciennes, France.

His research is primarily centered around artificial intelligence algorithms, with a specific focus on their verification and safety.

Abderraouf Boussif received the B.Eng. degree in system control engineering from the Polytechnic High School, Algiers, Algeria, in 2012, the master's degree in complex systems engineering from the École Normale Supérieure de Cachan, Cachan, Paris, in 2013, and the Ph.D. degree in safety system engineering from the University of Lille, Lille, France, in 2016.

He is a Researcher with the France's Railway Technological Research Institute, IRT Railenium, Valenciennes, France. His research interests are mainly in formal methods, model-based safety analysis, AI safety, and fault diagnosis of safety-critical systems, with a particular emphasis on railway control command and signaling systems.

Pierre-Jean Meyer received the master's degree in electrical engineering and automatic control from the Institut National Polytechnique de Toulouse, Toulouse, France, in 2011, and the Ph.D. degree in automatic control from Université Grenoble Alpes, Grenoble, France, in 2015.

He was a Post-Doctoral Researcher with the Automatic Control Department, KTH Royal Institute of Technology, Stockholm, Sweden, from 2015 to 2017, and the Electrical Engineering and Computer Sciences Department, University of California at Berkeley, Berkeley, CA, USA, from 2017 to 2020. Since 2021, he has been a Research Fellow with the COSYS-ESTAS Laboratory, Université Gustave Eiffel, Lille, France. His main research interests include safety verification using reachability analysis and abstraction-based control synthesis.

Mohamed Ghazel (Member, IEEE) received the master's degree in automatic control and industrial computer sciences from the École Centrale de Lille, Villeneuve-d'Ascq, France, in 2002, the Ph.D. degree in automatic control and industrial computer sciences from the University of Lille, Lille, France, in 2005, and the Habilitation à Diriger des Recherches (HDR) degree from the University Lille Nord de France, Lille, in 2014.

He is the Research Director with COSYS/ESTAS, Université Gustave Eiffel, (previously at IFSTTAR). He has been involved in several national and European research projects and acts as an expert for the European Commission in the framework of innovation programs. His research interests include the engineering, safety, and interoperability of transportation systems using discrete event models and formal methods.

Dr. Ghazel is a member of the IFAC TC 7.4 on Transportation Systems and IFAC TC9.2 on Systems and Control for Societal Impact.