Interval Weight-Based Abstraction for Neural Network Verification

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

Interval weight-based abstraction for neural network verification

Fateh Boudardara[1][0000−0001−5771−7676], Abderraouf Boussif[1], Pierre-Jean Meyer[2], and Mohamed Ghazel[1,2]

[1] Technological Research Institute Railenium, 180 rue Joseph-Louis Lagrange, F-59308, Valenciennes, France
{fateh.boudardara,abderraouf.boussif}@railenium.eu
[2] Univ Gustave Eiffel, COSYS-ESTAS, 20 rue Élisée Reclus, F-59666, Villeneuve d'Ascq, France {pierre-jean.meyer,mohamed.ghazel}@univ-eiffel.fr

Abstract. In recent years, neural networks (NNs) have gained much maturity and efficiency, and their applications have spread to various domains, including some modules of safety-critical systems. On the other hand, recent studies have demonstrated that NNs are vulnerable to adversarial attacks, thus a neural network model must be verified and certified before its deployment. Despite the number of existing formal verification methods of neural networks, verifying a large network remains a major challenge for these methods. This is mostly due to the scalability limitations of these approaches and the non-linearity introduced by the activation functions in the NNs. To help tackle this issue, we propose a novel abstraction method that allows the reduction of the NN size while preserving its behavioural features. The main idea of the approach is to reduce the size of the original neural network by merging neurons belonging to the same layer, and defining the new weights as intervals and sums of absolute values of those of the merged neurons. The approach allows for producing an abstract (i.e., reduced) model that is smaller and simpler to verify, while guaranteeing that this abstract model is an over-approximation of the original one. Our early experiments show that the approach enhances the scalability when performing verification operations, such as output range computation, on the abstract model.

Keywords: Neural network abstraction · Neural network verification · Over-approximation · Output range computation

1 Introduction

Neural networks (NNs) are a machine learning technique that is extensively integrated today in several domains, such as financial transactions and trading, image recognition and object detection [13]. Moreover, NNs are increasingly deployed in safety-critical systems such as autonomous vehicles and trains [3,18]. The standard evaluation methods of neural networks, that rely on running a series of tests on a finite subset of sample input data, cannot provide any

guarantee on the unseen samples. Indeed, these methods have proven to be sensitive and easy to fool by applying imperceptible perturbations. For instance, some undesired behaviours can be generated by applying small perturbations on the inputs [12,22]. This raises the issue about how these models can be verified and certified when it comes to deploy them in safety-critical systems.

Being given the accomplishments of formal methods in proving safety features in different software and hardware systems including in safety-critical applications [2,4], the idea of applying these methods on NNs has attracted a lot of attention; significant progress has been achieved in recent years and many NN verification methods and tools are developed [8,23]. In fact, NN verification methods can be classified into two main groups: complete and incomplete. Complete verification methods, also called exact methods, encode the exact behaviour of the model and the property to verify as linear programming (LP) problem, and then apply an adequate LP-solver to perform the verification. Mixed-Integer Linear Programming (MILP) and SAT/SMT based methods are the main techniques in this category of NN verification methods [5,6,9,11,15]. Because of the non-linearity of NNs (related to non-linear activation functions), these methods are able to verify only small networks. In contrast, incomplete methods construct an abstract model, generally by linearizing the activation function using abstract domains [20], linear functions [6] or some quadratic functions [25]. These methods are more scalable and can verify larger NNs, but since they actually investigate over-approximations of the model, they suffer from spurious counterexamples.

The main challenge of the aforementioned works lies in their scalability; indeed, the existing methods do not scale to verify large NNs [8,23]. To handle this issue, some model reduction methods are proposed [1,7,16,17,19,21]. The broad concept of these methods is to reduce the size of the neural network by merging some similar-behaving neurons while guaranteeing an over-approximation of the original network. This ensures that the property at hand holds on the original network whenever it holds on the reduced one.

In this work, we propose a novel NN reduction method to enhance the scalability of NN verification techniques. The method consists in merging nodes in the same layer based on formulas for calculating their incoming and outgoing weights. The obtained model $\overline{N}$ over-approximates[3] the behavior of the original one N, i.e., for every input x, the output y of N is included in the set of outputs $\overline{Y}$ of $\overline{N}$. The obtained network $\overline{N}$ is called an interval neural network since it may have interval weights [17]. To evaluate the efficiency of the proposed approach, we implemented it as a Python framework and used it to build the abstract model.

With regard to the related works, the closest approaches to our work are those proposed by Prabhakar and Afzal [17] and Elboher et al. [7]. The former is based on taking the interval hull of the incoming and the outgoing weights of the merged neurons. In our method, we replace the outgoing weights by the sum of the absolute value of the weights to enhance the precision. Elboher et

[3] With a slight abuse of notation, we write $N \subseteq \overline{N}$.

al. [7] abstract the outgoing weights by taking their sum after ensuring that they have all the same sign (positive or negative). In addition, the abstraction of incoming weights is done by either taking their *min* or *max* depending on the category of the merged neurons (only neurons belonging to the same category can be merged); hence an important preprossessing phase is required before applying the abstraction. In our approach, no preprossessing phase is required and, theoretically, all neurons of the same layer may be merged together. In addition, and unlike these approaches that support only NNs with $Relu$, our approach can abstract networks with $Tanh$ and $Relu$ activation functions; furthermore, it can be extended to cover a wide range of activation functions.

The remaining of this paper is structured as follows: in Section 2 we provide a technical background on NNs and their verification. Section 3 is devoted to presenting our abstraction approach. Section 4 is dedicated to the numerical evaluation of the approach based on the ACAS Xu benchmark. Finally, section 5 provides some concluding remarks and future work directions.

2 Background

2.1 Neural Networks

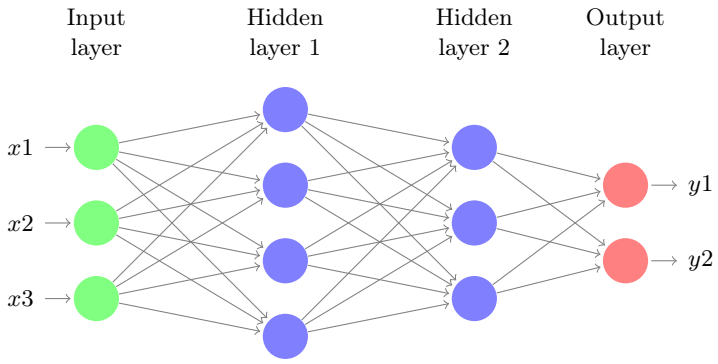


Fig. 1: Example of a neural network

A neural network N is a set of layers $L = \{l_0, l_1, ...l_n\}$ where l_0 and l_n are the input and the output layer, respectively, and $H_l = \{l_i : 1 \leq i \leq n-1\}$ is the set of hidden layers. Each layer l_i contains a set of nodes S_i, such that $\forall\, 1 \leq i \leq n$, a node $s_{ij} \in S_i$ is connected to all the nodes $s_{i-1,k}$ of the predecessor layer l_{i-1} with weighted edges $w^i_{jk} = w(s_{i-1,k}, s_{ij})$. Accordingly, the value $v(s_{ij})$ of a node

$s_{ij} \in S_i : 1 \leq i \leq n$ can be calculated using Equation 1.

$$\begin{cases} z(s_{ij}) = \sum_{s \in S_{i-1}} w(s, s_{ij}) \times v(s) + b_{s_{ij}} \\ v(s_{ij}) = \alpha(z(s_{ij})) \end{cases} \tag{1}$$

where $\alpha : \mathbb{R} \rightarrow \mathbb{R}$ is the activation function of node s_{ij}, and $b_{s_{ij}}$ is its bias. Calculating the output of N for a given value of the input x is performed by calculating $v(s_{nj}), \forall s_{nj} \in S_n$. This can be done by initializing S_0 with the given values of x (x is a vector and $|S_0| = |x|$), and then repeatedly applying Equation 1 to each hidden node $s_{ij} \in S_i, i \in \{1, 2, ..., n\}$. In this paper, and for the sake of simplicity and readability, z_{ij} can be used to denote the value of a node $s_{ij} \in S_i$ before activation instead of $z(s_{ij})$, similarly, the value after activation can be denoted as v_{ij}. Besides, the value of a hidden layer l_i for a given input is represented by the column vector $V_i = [v_{i1}, v_{i2}, ..., v_{i|S_i|}]^T$.

Prabhalar and Afzal [17] introduced a new representation of neural networks called interval neural networks (INNs). The weights of edges in INNs are intervals $w = [w^l, w^u]$ instead of scalars as in classic NNs. It is worth noticing that the classic neural networks can be considered as a particular case of INNs where $w^l = w^u$. The general structure along with the operations on INNs are similar to those on NNs.

As mentioned before, α in Equation 1 is an activation function. Recall that there are various types of activation functions which are used in NNs. Equations 2 and 3 represent *Relu* and *Tanh* activation functions, respectively.

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

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

A neural network can be seen as a function $\mathcal{N} : \mathbb{R}^{|S_0|} \rightarrow \mathbb{R}^{|S_n|}$, such that: $\mathcal{N}(x) = f_n(f_{n-1}(...(f_1(x))...)$, where f_i is the corresponding function of layer l_i, for $1 \leq i \leq n$. Fig. 1 depicts a network of 3 inputs, 2 hidden layers and 2 outputs. Its associated function is: $\mathcal{N} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$, s.t: $\mathcal{N}(x) = f_3(f_2(f_1(x)))$.

Remark 1. Notice that a neural network with α activation function (denoted as α-NN) means that the same function α is applied on all its hidden layers. For instance, a *Relu*-NN (resp. *Tanh*-NN) has only *Relu* (resp. *Tanh*) activation functions.

2.2 Verification of Neural Networks

Formal verification is the process of checking the correctness of a system model $\boldsymbol{M}$ with respect to a set of specifications, i.e., to check whether or not a model of a system satisfies a set of requirements (specifications) [4]. A formal verification method is used to prove that some property $\boldsymbol{P}$ holds on a system-model $\boldsymbol{M}$

(a) The original network (b) The abstract network

Fig. 2: An example explaining the main idea of the proposed approach.

(denoted as $M \models P$); otherwise, some counterexamples illustrating that $\boldsymbol{P}$ is not satisfied by $\boldsymbol{M}$ ($M \not\models P$) can be issued.

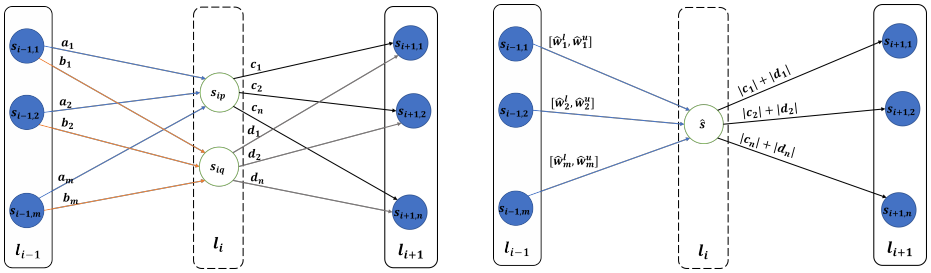
Similarly, NN verification consists of determining whether the NN model $\boldsymbol{N}$ satisfies some property $\boldsymbol{P}$, which is generally defined by a set of constraints on its input and output. That is to say, the NN verification problem can be defined by the tuple $\langle N, Pre, Post\rangle$ where $\boldsymbol{N}$ is the NN model, $\boldsymbol{Pre}$ is the set of input constraints and $\boldsymbol{Post}$ is the set of output constraints [14]. Let us denote by $\mathcal{N} : \mathbb{R}^{|S_0|} \rightarrow \mathbb{R}^{|S_n|}$ the associated function of $\boldsymbol{N}$, a verification property $\boldsymbol{P}$ on $\boldsymbol{N}$ can be formally expressed as:

$$\forall x \in \mathbb{R}^{|S_0|}, Pre(x) \implies Post(\mathcal{N}(x)) \tag{4}$$

3 Proposed Approach

3.1 Main Idea

In this paper, we propose a method to construct an over-approximation of NNs by merging neurons that belong to the same layer. To explain the general idea of the method, let us refer to Fig. 2 where the original network N is presented on Fig. 2a and its abstract network $\overline{N}$ after merging nodes s_1 and s_2 is presented on Fig. 2b. The challenge is how to determine the weights of the edges connecting the node $\hat{s}$ to the previous and the next layer, while ensuring that $y = v(s)$ is included in $\bar{y} = \hat{v}(\bar{s})$ for all possible values of s_{in} (keeping in mind that the values of nodes of the abstract network $\overline{N}$ are intervals). The value of s is $y = \alpha(c \times v(s_1) + d \times v(s_2))$, and the value of its associated abstract node $\bar{s}$ is $\bar{y} = \alpha(\hat{w}_2\hat{v}(\hat{s}))$, s.t: $\hat{v}(\hat{s}) = \alpha(\hat{w}_1 x)$, where $\hat{w}_2$ is the weight connecting $\hat{s}$ to $\bar{s}$ and $\hat{w}_1$ the weight $w(s_{in}, s)$. Our goal is to calculate $\bar{y}$ in such a way that $y \in \bar{y}$. We can define the abstract incoming weight of $\hat{s}$ as $\hat{w}_1 = [min(a, b), max(a, b)]$ to ensure that $v(s_1) \in \hat{v}(\hat{s})$ and $v(s_2) \in \hat{v}(\hat{s})$. The next step is to define $\hat{w}_2$. One way to do that is to sum c and d, i.e., $\hat{w}_2 = c + d$. However, in case of $c = -d$, the sum would be zero (which leads to always having $\bar{y} = 0$). To avoid that, we take the sum of the absolute value of c and d, and we transfer the signs of c and d backward to the previous layer's weights. The formula for calculating the interval weights will be provided in the sequel for $Tanh$ and $Relu$ activation functions.



(a) A sub neural network containing three hidden layers, we want to merge the two nodes of layer l_i.

(b) The sub neural network after abstraction. Here $\hat{w}_k^l, \hat{w}_k^u : 1 \leq k \leq m$ are the weights calculated using formula (6)

Fig. 3: An illustration of our abstraction method applied on a hidden layer l_i. The model on the right is the abstraction of the one on the left, where the node $\hat{s}$ is obtained upon merging s_{ip} and s_{iq}.

For the sake of clarity, we firstly provide the abstraction formula for the case of merging two neurons. Then, we discuss the general formula when it comes to the case of merging a set of neurons.

Let us consider the example in Fig. 3 showing a sub-network that contains three hidden layers with the same activation function α, and we aim to merge neurons s_{ip} and s_{iq} of layer l_i. The incoming weights to s_{ip} and s_{iq} are denoted by a_k and b_k, respectively. Formally, $a_k = w(s_{i-1,k}, s_{ip})$ and $b_k = w(s_{i-1,k}, s_{iq})$ for each $s_{i-1,k} \in S_{i-1}$. Analogously, we denote the outgoing weights by c_j and d_j such that $c_j = w(s_{ip}, s_{i+1,j})$ and $d_j = w(s_{iq}, s_{i+1,j})$, for all $s_{i+1,j} \in S_{i+1}$.

Definition 1. *We define the sign function in this paper as follows:* $sign : \mathbb{R} \rightarrow \{-1, 1\}$

$$sign(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{5}$$

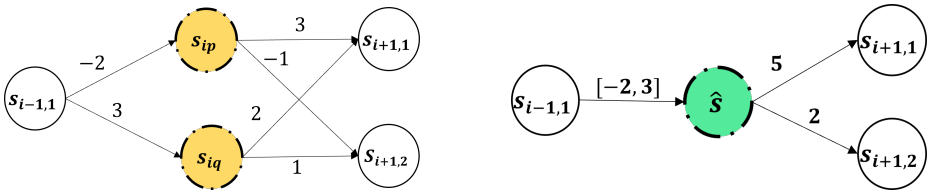
3.2 Abstraction for NNs with *Tanh*

In this part, we suppose that the used activation function $\alpha = tanh$. The abstraction of the two nodes of l_i is obtained by applying the steps presented in Procedure 1.

Procedure 1

1. *Create a new node* $\hat{s}$
2. *Calculate the incoming weights of* $\hat{s}$: $\forall s_{i-1,k} \in S_{i-1} : w(s_{i-1,k}, \hat{s}) = [\hat{w}_k^l, \hat{w}_k^u]$, *such that:*

$$\begin{cases} \hat{w}_k^l = \min\limits_{1 \leq j \leq n} \{sign(c_j)\, a_k, sign(d_j)\, b_k\} \\ \hat{w}_k^u = \max\limits_{1 \leq j \leq n} \{sign(c_j)\, a_k, sign(d_j)\, b_k\} \end{cases} \tag{6}$$



(a) The original model N: s_{ip} and s_{iq} of layer l_i are to be merged.

(b) The obtained model $\overline{N}$ after abstraction.

Fig. 4: An example of the abstraction method applied on two neurons of hidden layer l_i. Let us take $v(s_{i-1,1}) = 2$, then we have $v(s_{i+1,1}) = 0$ and $v(s_{i+1,2}) = 10$, $\hat{v}(s_{i+1,1}) = [-20, 30]$ and $\hat{v}(s_{i+1,2}) = [-8, 12]$. Hence, the over-approximation is fulfilled, since $v(s_{i+1,k}) \in \hat{v}(s_{i+1,k})$ for $k = 1, 2$.

where $n = |S_{i+1}|$ is the the number of neurons of l_{i+1}.

3. *Calculate the outgoing weights of $\hat{s}$, namely $\forall s_{i+1,j} \in S_{i+1}$:*

$$\hat{w}(\hat{s}, s_{i+1,j}) = |c_j| + |d_j| \tag{7}$$

4. *Calculate the biases of $\hat{s}$: $b_{\hat{s}} = [b^l_{\hat{s}}, b^u_{\hat{s}}]$, such that: $b^l_{\hat{s}} = \min_{1 \le j \le n} \{sign(c_j) b_{s_{ip}}, sign(d_j) b_{s_{iq}}\}$ and $b^u_{\hat{s}} = \max_{1 \le j \le n} \{sign(c_j) b_{s_{ip}}, sign(d_j) b_{s_{iq}}\}$, where $b_{s_{ip}}$ and $b_{s_{iq}}$ are the biases of s_{ip} and s_{iq}, respectively.*
5. *Remove s_{ip} and s_{iq} from S_i, add $\hat{s}$ to S_i and connect $\hat{s}$ to the nodes in l_{i-1} and l_{i+1} using the calculated weights.*

Procedure 1 can be applied repeatedly on the same layer to merge pairs of neurons, and can be iterated on multiple layers. An example depicting the execution of our abstraction method is given in Fig. 4.

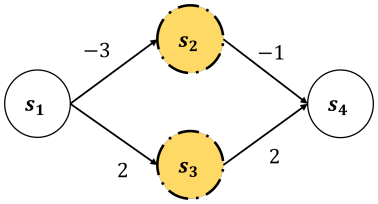
Proposition 1. *Let N be a NN with Tanh activation function. The application of Procedure 1 on a hidden layer $l_i : i \in \{1, 2, \ldots, |N| - 1\}$, guarantees that for every possible value of $v_{i-1}(s), s \in S_{i-1}$: $v_{i+1} \in \hat{v}_{i+1}$.* ■

Due to the limit on the number of pages, the proofs are omitted from this version of the paper.

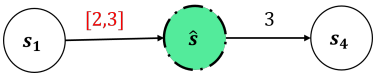
Up to now, we have considered the procedure of abstraction through the successive merging of two neurons each time. Hereafter, we propose the generalized procedure to merge more than two nodes at the same time. This can be done by updating equations 6 and 7, as follows:

Procedure 2

Lets denote by $\hat{S} \subseteq S_i$ the set of neurons to merged:



(a) A fragment of the original model N: we want to merge s_1 and s_2.



(b) The obtained partition after abstraction procedure defined by Equations 6 and 7.

Fig. 5: A counter-example of applying Procedure 1 on a *Relu* NNs.

1. *Incoming weights of the abstract node $\hat{s}$:*

$$\begin{cases} \hat{w}_k^l = \underset{s_i \in \hat{S}, s' \in S_{i+1}}{min} \{sign(w(s_i, s')) \times w(s_{i-1,k}, s_i)\} \\ \hat{w}_k^u = \underset{s_i \in \hat{S}, s' \in S_{i+1}}{max} \{sign(w(s_i, s')) \times w(s_{i-1,k}, s_i)\} \end{cases}$$

2. *Outgoing weights for each $s' \in S_{i+1}$:*

$$\hat{w}(\hat{s}, s') = \sum_{s_i \in \hat{S}} |w(s_i, s')|$$

Corollary 1. *Let N be a network with Tanh activation function, and $\overline{N}$ its abstract model obtained using Procedure 2 on one or multiple hidden layers. Then, the following holds: $N \subseteq \overline{N}$.* ■

Remark 2. Applying the proposed abstraction method until saturation would result in a network with a single neuron in each hidden layer, which would be a massive reduction of the size of the original network. However, it is plain to state that there is a trade-off, between the size reduction of the original model and the precision of the obtained abstract model, to be considered.

In the previous section, we considered networks with *Tanh* activation function. In the next section, the approach will be adjusted to handle *Relu*-networks.

3.3 Abstraction for NNs with *Relu*

We present in this section an adaptation of Formula 6 in order to extend our abstraction technique to NNs with *Relu*. The *Relu* eliminates the negative values. Hence, when we shift the signs of c_j and d_j back to previous layer (see Fig. 3), the lower bound of the obtained output may be greater than the original output (as shown in Fig. 5); namely assume that $v(s_1) = 1$, then $v(s_4) = 4 \notin \hat{v}(s_4) = [6, 9]$. To tackle this issue, we propose the following procedure to calculate **the lower bound** of the incoming weights for NNs with *Relu* activation function.

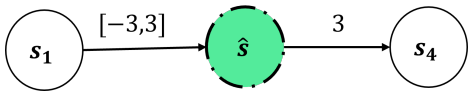


Fig. 6: An example illustrating the execution of Procedure 3 on a NN with *Relu*

Therefore, we update the main abstraction procedure, presented in section 3.1, by adding a condition that checks whether the incoming and the outgoing weights have the same signs. Adding such a condition guarantees that the abstract model over-approximates the behaviour of the original one. For instance, let us apply the new abstraction procedure on the same sub-network presented in Fig. 5a. Assume again that $v(s_1) = 1$, then the value of s_4 is $v(s_4) = 4$. Its abstract sub-network is presented in Fig. 6; and for the same value of s_1 $(v(s_1) = 1)$ $\hat{v}(s_4) = [0, 9]$; hence, $v(s_4) \in \hat{v}(s_4)$.

Procedure 3

1. *Create a new node* $\hat{s}$
2. *Let* c_j^* *(resp.* d_j^**) be the outgoing weight* c_j *(resp.* d_j*) such that* $sign(c_j*)a_k = \min_{1\leq j\leq n}\{sign(c_j)\, a_k\}$ *(resp.* $sign(d_j*)\, b_k = \min_{1\leq j\leq n}\{sign(d_j)\, b_k\}$*).*
   Calculate the incoming interval weights of $\hat{s}$ *as follows:* $\forall s_{i-1,k} \in S_{i-1}$, $w(s_{i-1,k}, \hat{s}) = [\hat{w}_k^l, \hat{w}_k^u]$*, such that:*
   (a) if $sign(a_k) \neq sign(c_j^*)$ *or* $sign(b_k) \neq sign(d_j^*)$*, then:*

$$\begin{cases} \hat{w}_k^l = \min\limits_{1\leq j\leq n}\{sign(c_j)\, a_k, sign(d_j)\, b_k\} \\ \hat{w}_k^u = \max\limits_{1\leq j\leq n}\{sign(c_j)\, a_k, sign(d_j)\, b_k\} \end{cases}$$

   (b) if $sign(a_k) = sign(c_j^*)$ *and* $sign(b_k) = sign(d_j^*)$ *then:*

$$\begin{cases} \hat{w}_k^l = \min\{a_k, b_k\} \\ \hat{w}_k^u = \max\limits_{1\leq j\leq n}\{sign(c_j)\, a_k, sign(d_j)\, b_k\} \end{cases}$$

3. *Calculate the outgoing weights of* $\hat{s}$ *as follows:*

$$\forall s_{i+1,j} \in S_{i+1}, \hat{w}(\hat{s}, s_{i+1,j}) = |c_j| + |d_j|$$

4. *Remove* s_{ip} *and* s_{iq} *from* S_i*, and add* $\hat{s}$ *to* S_i*. Connect* $\hat{s}$ *to the nodes in* l_{i-1} *and* l_{i+1} *using the calculated weights.*
5. *After applying the abstraction on all hidden layers, replace all the remaining scalar weights* $w \in \mathbb{R}$ *by an interval:* $[min(w, 0), max(w, 0)]$.

Proposition 2. *For a network* N *with the Relu activation function, applying the abstraction method defined in Procedure 3 on a layer* $l_i : i \in \{1, \ldots, |N| - 1\}$ *guarantees that for every possible value* $v_{i-1}(s)$ *of* $s \in S_{i-1}$ *and* $\forall s_{i+1,j} \in S_{i+1}$: $v(s_{i+1,j}) \in v(\hat{s}_{i+1,j})$.

■

Notice that in Propositions 1 and 2, we assume that the layers l_{i-1}, l_i and l_{i+1} have the same activation function (either *Tanh* or *Relu*).

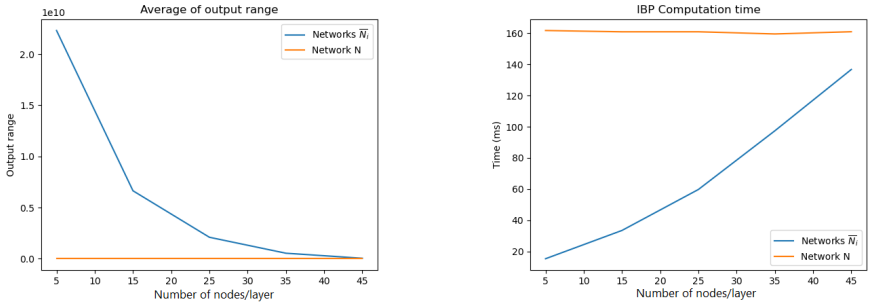
4 Early Experiments

We implemented our abstraction method as a Python framework while considering a NNET format reader [10]. The user can set the abstraction's parameters, such as the maximum number of nodes on each layer after abstraction and the strategy of nodes selection. In our analysis, we simply used random selection, but different nodes' selection strategies (using heuristics for instance) can be integrated. To evaluate the performance of our proposed method, we conducted a series of experiments on the ACAS Xu benchmark [10]. This benchmark is a set of 45 NNs pertaining to an airborne-collision avoidance system. Each network has 300 hidden nodes (6 hidden layers with 50 neurons each), 7 inputs and 5 outputs.

After uploading the model N from the NNET file[4], we generate 5 abstract models $\overline{N}_i$, $i \in \{1, 2, 3, 4, 5\}$ with 5, 15, 25, 35 and 45 nodes on all hidden layers, respectively. We also considered some constraints on the NN input, namely specified by property ϕ_5 as defined in [11]. Moreover, we used the Interval Bound Propagation (IBP) algorithm [24] for calculating the output range of N and and abstract networks $\overline{N}_i$. Over 50 random runs, we calculated the average of the abstraction time for each abstract network $\overline{N}_i$ and we compared the average of the output ranges (upper bound) and the IBP computation time of abstract networks to those of the original network N. The obtained results are summarized in Figures 7a, 7b, 8.

From Fig. 7a, we can observe that the precision of the abstract model highly depends on the number of the merged nodes, i.e., allowing for more abstract nodes leads to less precise abstract models. Contrarily, the output computation time is proportional to the number of nodes on each layer as shown in Fig. 7b).

[4] The network *ACASXU_ experimental_ v2a_ 1_ 1.nnet* is used in this work.



(a) Average of upper output range on dimension 1

(b) IBP computation time on the original and the reduced models

Fig. 7: Comparison between the output and computation time of the original and the abstract models

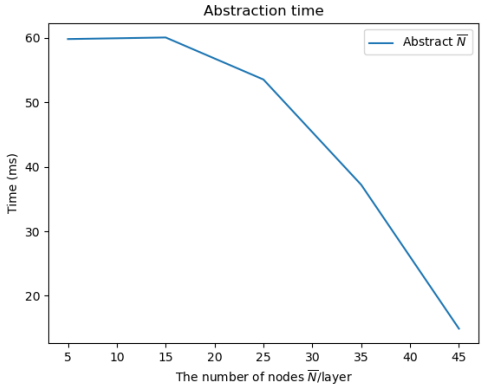


Fig. 8: Abstraction time for different size of abstract models.

It is straightforward to notice that the fewer the number of nodes of each layer, the faster the computation is performed. Although IBP is among the fastest verification methods, its computation time is significantly higher than the abstraction time which can be neglected if a more costly verification method is applied.

5 Conclusion

In this paper, we proposed a novel NN reduction method for the sake of enhancing the efficiency of various analysis operation that can be performed on NNs, such as, for instance, the computation of the output range (for invariant's checking for instance) or any other verification operations. Our method can be applied on both feed-forward *Tanh*-NN and *Relu*-NN. Yet, the approach can be extended to further activation functions. The model reduction approach guarantees that the abstract model is an over-approximation of the original one. Therefore, once some property is satisfied on the abstract model $\overline{N}$, it necessarily holds on the original one N.

The approach is implemented using Python, and an experimental study was conducted to analyse the efficiency and the precision of the generated abstract model. The conducted experiments on the basis of a state-of-the-art benchmark show how the precision of the abstract model is impacted by the size of its hidden layers. Furthermore, we showed that the proposed method can effectively reduce the output range computation time.

In the present paper, we proved the over-approximation of the abstract model w.r.t. the original one in the case of *Tanh* and *Relu* activation functions. In future work, we aim to extend the proof to consider the *Sigmoid*, *Leaky Relu* and *SELU* activation functions. In addition, in the present work, the nodes to be merged were selected randomly; but we intend to develop some nodes' selection heuristics that can improve the precision of the abstract model.

Acknowledgements 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. Ashok, P., Hashemi, V., Křetínskỳ, J., Mohr, S.: Deepabstract: Neural network abstraction for accelerating verification. In: International Symposium on Automated Technology for Verification and Analysis. pp. 92–107. Springer (2020)
2. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS press (2009)
3. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint (2016)
4. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., et al.: Handbook of model checking, vol. 10. Springer (2018)
5. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Proc. 10th NASA Formal Methods. pp. 121–138 (2018)
6. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 269–286. Springer (2017)
7. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: International Conference on Computer Aided Verification. pp. 43–65. Springer (2020)
8. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020)
9. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International conference on computer aided verification. pp. 3–29. Springer (2017)
10. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2016)
11. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International conference on computer aided verification. pp. 97–117. Springer (2017)
12. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: Artificial intelligence safety and security, pp. 99–112. Chapman and Hall/CRC (2018)
13. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
14. Leofante, F., Narodytska, N., Pulina, L., Tacchella, A.: Automated verification of neural networks: Advances, challenges and perspectives. arXiv preprint arXiv:1805.09938 (2018)

15. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint (2017)
16. Prabhakar, P.: Bisimulations for neural network reduction. arXiv preprint (2021)
17. Prabhakar, P., Afzal, Z.R.: Abstraction based output range analysis for neural networks. arXiv preprint (2020)
18. Ristić-Durrant, D., Franke, M., Michels, K.: A review of vision-based on-board obstacle detection and distance estimation in railways. Sensors **21**(10), 3452 (2021)
19. Shriver, D., Xu, D., Elbaum, S., Dwyer, M.B.: Refactoring neural networks for verification. arXiv preprint (2019)
20. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)
21. Sotoudeh, M., Thakur, A.V.: Abstract neural networks. In: International Static Analysis Symposium. pp. 65–88. Springer (2020)
22. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint (2013)
23. Urban, C., Miné, A.: A review of formal methods applied to machine learning. arXiv preprint (2021)
24. Xiang, W., Tran, H.D., Yang, X., Johnson, T.T.: Reachable set estimation for neural network control systems: A simulation-guided approach. IEEE Transactions on Neural Networks and Learning Systems **32**(5), 1821–1830 (2020)
25. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. arXiv preprint (2018)