Deep Neural Networks Abstraction using An Interval Weights Based Approach

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

Deep Neural Networks Abstraction using An Interval Weights Based Approach

Fateh Boudardara [1], Abderraouf Boussif[1], Mohamed Ghazel[2], and Pierre-Jean Meyer[2]

[1]Technological Research Institute Railenium, Valenciennes, France
[2]Univ Gustave Eiffel, COSYS-ESTAS, Villeneuve d'Ascq, France

Abstract

In this work, we present a Neural Network (NN) abstraction approach to reduce the state-space (number of nodes) of NN towards solving the non-scalability of NN formal verification approaches. The main idea consists in merging neurons on the NN layers in order to build an abstract model that over-approximates the original one. Concretely, the outgoing weights of the abstract network are computed as the sum of the absolute value of the weights on the original one, while the incoming weights are intervals determined based on the signs of the outgoing and the incoming weights of the original model.

1 Introduction

Due to the tremendous success of deep neural networks (DNNs), they are increasingly deployed in safety-critical systems, such as autonomous cars and trains. However, these systems must meet some specific safety requirements before their deployment. Therefore, many concerns about the safety of DNNs have been raised recently. In fact, recent studies demonstrated the vulnerability of DNNs [1], thus the domain of neural networks verification are becoming more popular and attractive. Several formal verification methods are adjusted and applied to check some properties on DNNs, such as safety and robustness. Originally, the verification problem of DNNs was transformed to an optimization problem and solved using Mixed-Integer Linear Programming and SAT/SMT solvers [2, 3]. Many other methods were developed for instance abstract interpretation [4], reachability [5] and others.

Unfortunately, the developed techniques cannot scale to verify large models because of the high complexity of DNNs. Model reduction methods that are considered as abstraction methods and consist of reducing the size of the model while preserving some relevant behaviors [6, 7, 8], are seen as a promising remedy to the problem of scalability of the existing NN verification methods. A model reduction approach ensures that whenever the property holds on the reduced model, it must hold on the original. In this paper we present a method that is based on converting the original NN to an interval NN (INN). The reduced model is constructed by taking the interval hull of the incoming weights and the sum of the outgoing weights in such a way that the outputs of the original network are always included in those of the abstract one. The presented method supports both *Tanh*-NN and *Relu*-NN. A succinct presentation of the proposed approach and the preliminary obtained results are presented here below; further details about the approach and the experiments are presented in [9].

A neural network is a sequence of connected layers. The first layer is the input layer, followed by one or more hidden layers and an output layer. Each neuron s_{ij} in S_i [1] of a hidden layer receives data from its predecessor layer, calculate its activated value using Equation 1, and forward the result to its successor layer.

$$v(s_{ij}) = \alpha(\sum_{s \in S_{i-1}} w(s, s_{ij}) \times v(s) + b_{s_{ij}}) \quad (1)$$

In Equation 1, $w(s, s_{ij})$ is the weight of the edge connecting $s \in S_{i-1}$ to $s_{ij} \in S_i$, $b_{s_{ij}}$ is the bias of the node s_{ij}, and α is a predefined activation function. Our method supports *Relu*-NN ($\alpha(x) = Relu(x) = max(0, x)$) and *Tanh*-NN ($\alpha(x) = Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$).

(a) The original network

(b) The abstract network

Figure 1: An example explaining the main idea of the proposed approach.

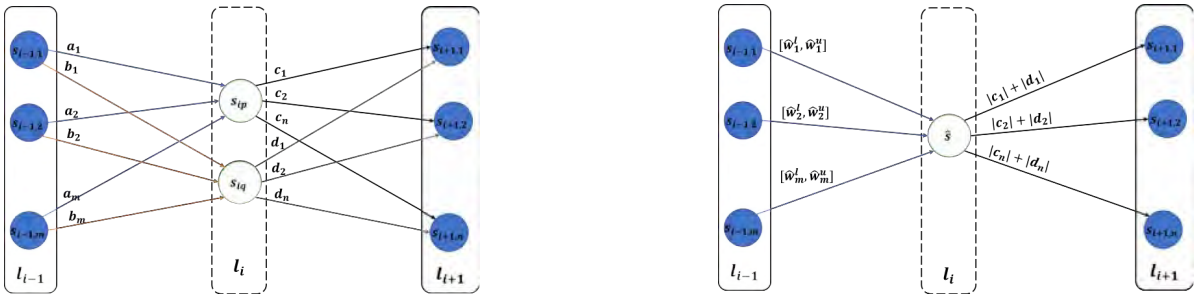
2 Proposed model reduction method

Model reduction for NNs, as a sub-category of NN abstraction, is a concept of reducing the size of NNs by merging some neurons while guaranteeing that the original model N satisfies the property P whenever this property is satisfied by the abstract model $\bar{N}$, i.e., $\bar{N} \models P \implies N \models P$.

The broad idea of our method is to merge neurons of hidden (intermediate) layers and compute the incoming weights of the abstract node as the convex interval hull of its incoming weights before abstraction multiplied by the sign of its outgoing weights. On the other hand, the outgoing weights of the abstract node are computed as the sum of the absolute value of its corresponding outgoing weights on the original network. Figure 1 illustrates the main idea approach. Notice that $sign$ is a function defined as follow: $sign : \mathbb{R} \to \{-1, 1\}$

$$sign(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2}$$

2.1 Model reduction for NN with *Tanh* activation function



(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 (3).

Figure 2: 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 simplicity and without lose of generality, let consider the network in Figure 2a, and assume that we want to merge the two nodes s_{ip} and s_{ip}. The obtained abstract network is presented in Figure 2b, where $\hat{s}$ is abstract node after merging s_{ip} and s_{ip}. The incoming weights of $\hat{s}$ have the form of intervals and they are calculated as follows:

$$\begin{cases} \hat{w}_k^l = \min\limits_{1 \leq j \leq n} \{sign(c_j)\, a_k, sign(d_j)\, b_k\} \\ \hat{w}_k^u = \max\limits_{1 \leq j \leq n} \{sign(c_j)\, a_k, sign(d_j)\, b_k\} \end{cases} \tag{3}$$

and its outgoing weights are the sum of the absolute value of the corresponding outgoing weights of s_{ip} and s_{ip}. Algorithm 1 summarizes the essential steps of the model reduction for neural networks with *Tanh* (*Tanh*-NN).

[1] S_i is the set of neurons of layer l_i, and $s_{ij} \in S_i$ is the j^{th} neuron of S_i

Algorithm 1 Proposed model reduction procedure for *Tanh*-NN

1: create a node $\hat{s}$
2: select s_{ip} and s_{iq}
3: calculate the incoming weights to $\hat{s}$ using Equation 3
4: calculate the outgoing weights: $\hat{w}(\hat{s}, s_{i+1,j}) = |c_j| + |d_j|$
5: replace s_{ip} and s_{iq} with $\hat{s}$

2.2 Model reduction for NN with *Relu* activation function

The *Relu* function is a piece-wise linear function, it eliminates the negative values (set them to zero) and returns only positive values. This particularity prevents the application of the model reduction method present in Algorithm 1 on *Relu*-NNs. Algorithm 2 depicts the update of Algorithm 1 to support *Relu*-NNs, where c_j^* (resp. d_j^*) presented in line 4 in Algorithm 2 is the outgoing weight c_j (resp. d_j) such that $sign(c_j*)\ a_k = \min\limits_{1\leq j\leq n}\{sign(c_j)\ a_k\}$ (resp. $sign(d_j*)\ b_k = \min\limits_{1\leq j\leq n}\{sign(d_j)\ b_k\}$).

Algorithm 2 Proposed model reduction procedure for *Relu*-NN

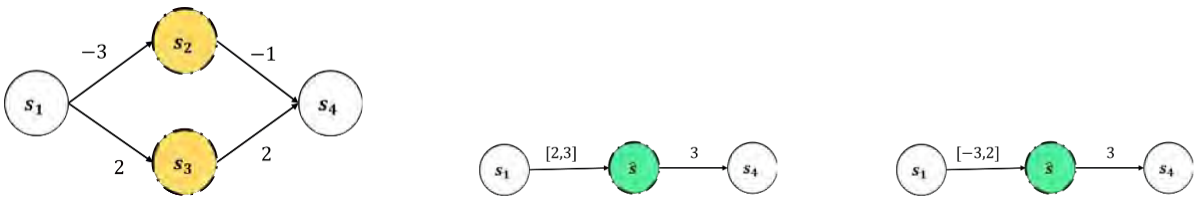
1: create a node $\hat{s}$
2: select s_{ip} and s_{iq}
3: **for** outgoing weight of s_{ip} and s_{iq} **do**
4: calculate c_j^* and d_j^*
5: **end for**
6: calculate the incoming weights to $\hat{s}$ using Algorithm 3
7: calculate the outgoing weights: $\hat{w}(\hat{s}, s_{i+1,j}) = |c_j| + |d_j|$
8: replace s_{ip} and s_{iq} by $\hat{s}$

Algorithm 3 Computation of the incoming weights for *Relu*-NN

1: **if** $sign(a_k) \neq sign(c_j^*)$ or $sign(b_k) \neq sign(d_j^*)$ **then**
2: Use Equation 3
3: **else if** $sign(a_k) = sign(c_j^*)$ and $sign(b_k) = sign(d_j^*)$ **then**
4: Use Equation 4
5: **end if**

$$\begin{cases} \hat{w}_k^l = \min\{a_k, b_k\} \\ \hat{w}_k^u = \max\limits_{1\leq j\leq n}\{sign(c_j)\ a_k, sign(d_j)\ b_k\} \end{cases} \tag{4}$$

Figure 3 shows an example of merging two neurons s_2 and s_3 of the original network presented in Figure 3a. In the case the network uses the *Tanh* activation function, its abstract network is the network in Figure 3b obtained by applying Algorithm 1. If we suppose that the original network (Figure 3a) is a *Relu*-NN, Algorithm 2 is applied and the corresponding abstract network is presented in Figure 3c.

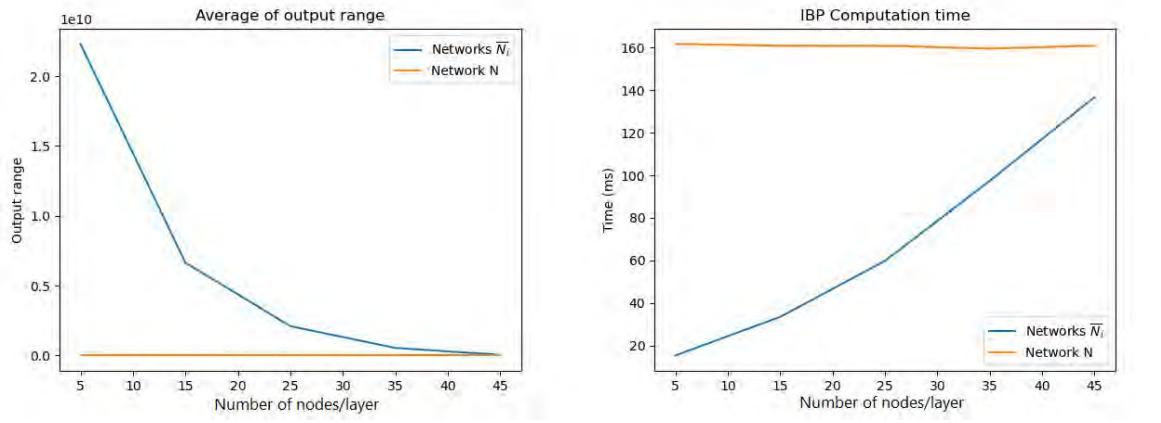


(a) The original network
(b) The abstract model for *Tanh*-NN
(c) The abstract model for *Relu*-NN

Figure 3: An example of the abstraction method applied on two neurons s_2 and s_3 of a hidden layer l_i.

We implemented Algorithm 1 and 2 as a Python framework and we conducted a series of experiments on the ACAS Xu benchmark [2]. For the output range computation we considered the property ϕ_5 as defined in [2]. We

examined the performance of our approach by varying the size of layers of the abstract network $(5, 15, 25, 35, 45)$. The selection of neurons to be merged is performed randomly, and for each abstract network we calculated the abstraction time, the output range using Interval Bound Propagation (IBP) algorithm [5] and IBP computation time. We compared the average output range and the IBP computation time over 50 random runs for each abstract model with the results obtained on the original model as shown in Figure 4a and 4b.



(a) Output range

(b) IBP computation time

Figure 4: Comparison of the output range and the IBP computation time on the original network and different abstract networks.

The obtained results showed that there is a trade-off between the total number of abstract nodes and the precision of the obtained abstract model. Having more nodes in the abstract network increases its precision, and also its IBP computation time. Notice that IBP is one of the fastest verification methods, and yet its computation time is significantly higher compared to the abstraction time of our approach (its is not provided to shortage of space and it will be presented on the poster).

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint*, 2013.

[2] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[3] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *Proc. 10th NASA Formal Methods*, pages 121–138, 2018.

[4] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[5] Weiming Xiang, Hoang-Dung Tran, Xiaodong Yang, and Taylor T Johnson. 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.

[6] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. *arXiv preprint*, 2020.

[7] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In *International Conference on Computer Aided Verification*, pages 43–65. Springer, 2020.

[8] Pranav Ashok, Vahid Hashemi, Jan Křetínskỳ, and Stefanie Mohr. Deepabstract: Neural network abstraction for accelerating verification. In *International Symposium on Automated Technology for Verification and Analysis*, pages 92–107. Springer, 2020.

[9] Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, and Mohamed Ghazel. Interval weight-based abstraction for neural network verification. In *Fifth International Workshop on Artificial Intelligence Safety Engineering (Accepted)*, pages 1–16, 2022.