A review of abstraction methods towards verifying neural networks

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

A review of abstraction methods towards verifying neural networks

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

August 30, 2024

Abstract

Neural networks as a machine learning technique are increasingly deployed in various domains. Despite their performances and their continuous improvement, the deployment of neural networks in safety-critical systems, in particular for autonomous mobility, remains restricted. This is mainly due to the lack of (formal) specifications and verification methods and tools that allow for getting sufficient confidence in the behavior of the neural network-based functions. Recent years have seen neural network verification getting more attention; and many verification methods were proposed, yet they are far from being applicable to real-world models. The main challenge of these methods is related to their computational complexity and the size of neural networks pertaining to complex functions. As a consequence, applying abstraction methods for neural network verification purposes is seen as a promising mean to cope with such issues. In general terms, the aim of abstraction is to build an *abstract* model by *omitting* some irrelevant details or some details that are not highly impacting w.r.t some considered features. Thus, the verification is made easier while preserving, to some extent, the relevant behavior for the properties to be examined on the original model. In this paper, we review both the abstraction techniques for activation functions and model size reduction approaches, with a particular focus on the latter. Throughout the paper, we briefly present the main idea of each approach, and then discuss their respective advantages and limitations. Finally, we provide some insights and guidelines to improve the discussed methods.

1 Introduction

Neural Network (NN) is one of the most popular machine learning techniques [16, 25]. The use of such an approach has shown fast progress during the last decade, giving rise to a noticeable enhancement of the technique, as witnessed by its successful achievements in various domains [30]. Nowadays, applications of NNs can be encountered in a wide range of domains, such as in financial transactions, trading, forecasting and fraud detection [30, 34]. In recent years, with the advances in terms of computational performances, NNs have been widely adopted in image recognition and object detection systems. Namely, they are increasingly investigated to be deployed for safety-critical applications, in particular for the design of environment monitoring and decision making functions in autonomous vehicles and trains [39, 50]. A software module of a safety-critical system needs to be certified before its deployment. Thus, it is required to develop methods to verify safety specifications and certify such NN-based software.

The earliest works that deal with the verification of NN models are based on the transformation of the model at hand into a system of linear equations that can be solved by means of available verification tools, namely SAT/SMT solvers [11, 21, 37] and MILP solvers [4, 8, 31, 46]. Although these methods are theoretically sound[1] and complete[2], they are limited to small-size neural networks due to the non-linearity of NN models. Indeed, the number of linear constraints grows exponentially with the number of neurons for which the activation functions need to be linearized, which may give rise to a state-space explosion problem. Therefore, verification methods based on over-approximation have been proposed to help mitigate this problem while preserving the soundness but not the completeness [10, 27, 38, 52, 53, 54] (see [51] for more details). Among these techniques, abstraction methods try to ease the verification problem by abstracting the activation function using linear bounds [11] or

[1] Whenever the method returns that the property holds, it indeed holds on the system.
[2] The verification method never returns "Unknown".

abstract domains [14, 42, 43], or by reducing the size of the network to improve the scaliblity of NN verification engines. In the latter case, a smaller and easier model to verify is generated from the original network [12, 36]; thus, instead of applying the verification method directly on the original model, the verification process can be enhanced by applying it on the reduced model.

Regarding the substantial interest in NN verification and the amount of existing methods for certifying NNs, many surveys and reviews on NN verification methods have been proposed in the literature. For instance, Leofante et al. [26] established three types of NN verification properties: equivalence, invertibility and invariance. They also provided a review of NN verification techniques based on constraints solving. Liu et al. [29] classified the existing verification methods into three basic categories: optimization, reachability and search-based verification techniques. Huang et al. [18] conducted a review about deep NN safety and trustworthiness. For NN verification, the authors distinguished between global and local properties. Regarding the guarantees of the verification technique, the survey classifies NN verification techniques into deterministic, approximative and statistical. According to [49], verification methods can be classified as geometric-based methods, MILP, SAT/SMT and optimization-based methods, even though MILP and SAT/SMT based verification methods can also be considered as particular cases of optimization techniques. Recently, Urban et al. [51] discussed the verification methods applied to machine learning. For NN verification, the authors proposed a classification of the existing methods into complete or incomplete methods with respect to the output of the verification process. Moreover, the review [51] summarizes formal verification approaches for different machine learning techniques such as support vector machine and decision trees.

Among all the surveys and reviews discussed above, and to the best of our knowledge, no existing work offers an overview on the abstraction methods for NN verification purposes. The aim of this work is to present a review on the existing activation function abstraction and model reduction methods in the literature for NN verification, and derive a critical discussion regarding these techniques. Concretely, for each discussed approach we will sketch out the main idea and analyze its advantages along with its drawbacks. For model reduction techniques, we will particularly highlight how each method can affect the verification process, and we will discuss further research directions in terms of these techniques. It is worth noticing that in this paper, we only consider NN abstraction methods that are used for verification purposes, *i.e.*, we do not include neural networks' compression techniques such as quantisation and edges pruning [17], since their goal is to build a compressed model to speed up the run-time execution, while preserving the model's accuracy but not necessarily its behavior.

The remainder of the paper is structured as follows: In Section 2, preliminary concepts and notations pertaining to neural networks are introduced, the verification problem of NNs is stated and an overview of the existing NN verification methods is provided. Section 3 reviews existing NN abstraction approaches, with a deeper focus on model reduction methods. Besides discussing the main features of the evoked techniques, some pointers to possible enhancements of the discussed methods will be provided. Finally, in Section 4 we recall the main findings through our review and outline some challenges and perspectives regarding NN abstraction.

2 Background

2.1 Neural networks

A neural network is a sequence of interconnected *layers* $\{l_1, l_2, ..., l_n\}$. When the number of layers is important, the term *Deep Neural Networks* is used. In an NN, each layer holds one or many nodes, called *neurons*. The first layer l_1 is called the *input layer*, the last one l_n is the *output layer* and the remaining layers $l_i : 2 \leq i \leq n-1$ are referred to as *hidden layers*. Likewise, the nodes in the hidden layers are called *hidden nodes*. Each hidden node is associated with a *bias* and an *activation function*. The nodes of a layer $l_i \in \{l_2, l_3, ..., l_n\}$ are connected to the nodes of the previous layer via *weighted* edges. That is to say, a neuron of layer l_i receives data from layer l_{i-1}, calculates the weighted sum of this data and adds a bias. An activation function is then applied and the result is forwarded to interconnected neurons of the next layer l_{i+1} (more details are given below). The propagation of data from the input layer to the output layer, passing through multiple hidden layers, is called "feed-forward propagation". An NN is built upon a training phase that aims to recognize and encode the underlying input-output relationship (correlation) of a data set. To evaluate an NN model, the accuracy, which is the rate of correct predictions, is calculated. Figure 1 shows a neural network of 4 layers: an input

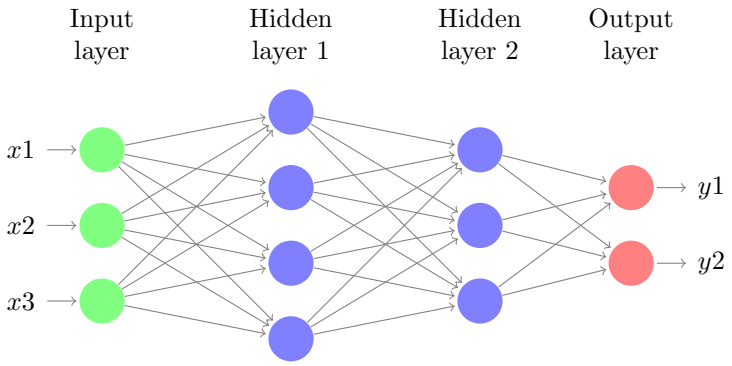


Figure 1: Example of a neural network

layer of 3 inputs, two hidden layers of 4 and 3 nodes, respectively, and a 2-node output layer.

An NN model can indeed be seen as a function $\mathcal{N} : D_x \to D_y$, where D_x is the input domain and D_y is the output domain of the model. For image classification for example, D_x is a matrix of pixel values representing an image, D_y is the set of all possible classes of these images. As an NN model consists of a sequence of n layers, $\mathcal{N}$ can be considered as a composition of a set of functions $\{f_1, f_2, ..., f_n\}$ where f_i, $1 \leq i \leq n$ is the corresponding function of layer l_i. This can be written, formally, as: $\mathcal{N}(x) = f_n(f_{n-1}(...(f_1(x))...)$, where f_1 is the identity function. In the following, we give some formal definitions pertaining to NN concepts and properties that will be used later on in this paper.

Definition 2.1. For a layer $l_i : i \in \{1 \dots n\}$, we define the set of neurons of l_i by S_i, with $|S_i|$ the number of neurons in the layer l_i. And for a neuron $n_{ij} \in S_i$, its value w.r.t to an input x is $v_{ij}(x)$. For simplicity, when x is not specific, we use v_{ij} instead of $v_{ij}(x)$.

Let $n_{ij} \in S_i$ be a neuron of a hidden layer l_i, its value v_{ij} is calculated in two steps:

1. **Affine transformation**: calculates the sum of previous layer's outputs modulated by the weights assigned to the corresponding edges, plus the bias. This can be formulated as:

$$z_{ij} = \sum_{k=1}^{k=|S_{i-1}|} w_{j,k}^{i-1} \times v_{i-1,k} + b_{ij}$$

   where $w_{j,k}^{i-1}$ is the weight of the edge connecting the nodes $n_{i-1,k}$ and n_{ij}, and b_{ij} is the bias of the node n_{ij}. Note that z_{ij} is also called the *pre-activation* value of n_{ij}.

2. **Activation function**: the final value v_{ij}, also called the value after activation, is determined by applying an activation function σ to z_{ij}, i.e. $v_{ij} = \sigma(z_{ij})$.

The two steps are summarized in equation (1). The obtained value v_{ij} is the output value of n_{ij} which will be forwarded to the next layer l_{i+1}. Figure 2 illustrates these steps on an example.

$$v_{ij} = \sigma\left(\sum_{k=1}^{k=|S_{i-1}|} w_{j,k}^{i-1} \times v_{i-1,k} + b_{ij}\right) \tag{1}$$

The calculation of the NN output $y = \mathcal{N}(x)$ for a given input x, is done by successively applying these operations, layer by layer, from the input to the output layer.

Depending on the application, there exists several activation functions: *Sigmoid*, *Tanh*, *Relu*, etc. [55]. Relu (for Rectified Linear Unit), as defined in equation (2), is a piece-wise linear function that has linear behaviors in $]-\infty, 0]$ and in $[0, +\infty[$. The ReLU activation function is widely used in NN, and due its simple form and its piece-wise linear behaviour, the majority of the existing neural network verification and abstraction approaches consider models with this activation function [18, 21, 31].

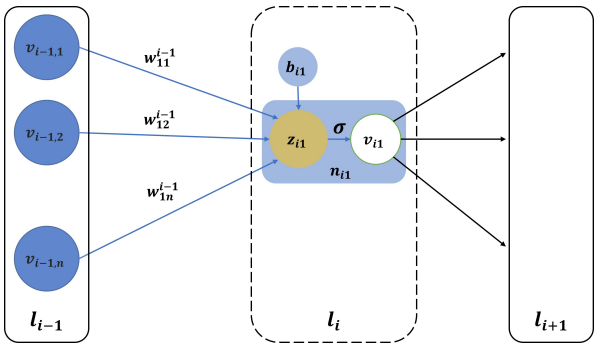


Figure 2: An example showing the connection between a neuron of l_i and l_{i-1}

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

Remark (Weights). In this paper, the weight of an edge connecting $n_{ik} \in S_i$ to a node $n_{i+1,j} \in S_{i+1}$ is written as w_{jk}^i or $w(n_{ik}, n_{i+1,j})$.

2.2 Verification of neural networks

Formal verification is the domain of proving or disproving that a system meets certain formal specifications and properties. A verification problem is defined as:

$$M \models P \, ? \tag{3}$$

which is equivalent to answering the question: does the system model $\boldsymbol{M}$ satisfy the property $\boldsymbol{P}$? Depending on the verification technique, the system has to be modelled (e.g., state transition model) and the specifications need to be expressed respecting some specific syntax (e.g., temporal logic). The aim of a verification technique is to prove that $\boldsymbol{P}$ holds on $\boldsymbol{M}$ or generate a counterexample witnessing the violation of $\boldsymbol{P}$. Many verification techniques, such as model-checking, SAT/SMT, abstract interpretation, and theorem proving have been broadly and successfully applied to verify software-intensive systems [2, 6].

Accordingly, formal verification for NN can be defined as in formula (3), where $\boldsymbol{M}$ is the NN model and $\boldsymbol{P}$ is the property to be checked, which is generally a mathematical formula constituted of a set of constraints on the inputs and the outputs of the network.

According to Leofante et al. [26], for a given NN represented by its corresponding function $\mathcal{N} : D_x \to D_y$, the NN verification problem can be stated as follows:

- Define $pre(x)$ and $post(y)$ as a set of constraints on the input x (preconditions) and the output y (postconditions), respectively. Here, $pre(x)$ and $post(y)$ are sorted first order logic formulas.
- For all x satisfying the preconditions $pre(x)$, verify whether or not $\mathcal{N}(x)$ fulfills the postconditions $post(\mathcal{N}(x))$.

This can be formulated as follows:

$$\forall x \in D_x, pre(x) \implies post(\mathcal{N}(x)) \tag{4}$$

Example 2.1. By taking $D_x = \mathbb{R}^2$ and $D_y = \mathbb{R}$ as the input and output domains of some given NN, the verification problem defined by formula (4) can be instantiated as:

$$\begin{cases} pre(x) : x_1 \in [l_1, u_1] \wedge x_2 \in [l_2, u_2], \text{ with } x = (x_1 \; x_2)^T \\ post(\mathcal{N}(x)) : \mathcal{N}(x) \geq c \end{cases}$$

where $l_i, u_i, c \in \mathbb{R}$, and $l_i \leq u_i$. The verification problem of this example thus aims to check that for all input x in the 2-dimensional interval defined in the precondition, the corresponding output $\mathcal{N}(x)$ is lower-bounded by c as in the postcondition.

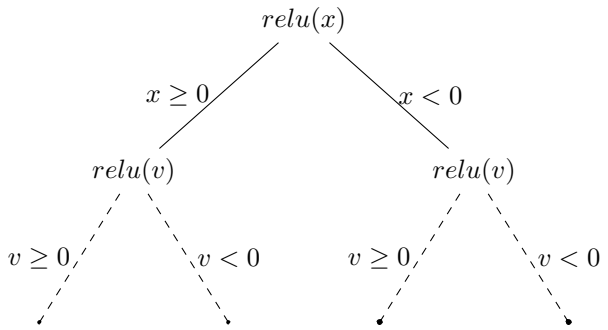


Figure 3: An example of state-space explosion. For two Relu nodes, case splitting leads to four linear subproblems.

Example 2.2. To verify the robustness property of a classification network, i.e., to check for a classification problem that the network assigns the same label (class) c_i to all inputs within a small region surrounding x_0, the verification problem can be formulated using formula (4) as follows:

$$\begin{cases} \exists x_0 \in D_x : \mathcal{N}(x_0) = c_i \\ pre(x) : \|x - x_0\|_p \leq \epsilon \\ post(\mathcal{N}(x)) : \mathcal{N}(x) = c_i \end{cases}$$

where $\|.\|_p$ is a given norm.

Verifying properties of NNs is increasingly receiving more attention and many approaches have been proposed in recent years [18, 29]. The straightforward verification way consists of encoding the NN behavior, as well as the property to be checked, as a system of linear equations, and then use an appropriate engine to perform the verification process. For instance SAT/SMT and MILP encoding are widely used to verify NNs properties [4, 9, 19, 21, 22, 31, 46]. These methods are also called *complete* because they encode the exact behavior of the network. However, since most of the common activation functions are nonlinear, this kind of verification methods does not scale in the case of large neural networks, and suffers from state-space explosion. For example for the piece-wise linear activation function Relu, each Relu node has to be split into two linear constraints, i.e.: if $y = relu(x)$, then $y = 0$ when $x < 0$ and $y = x$ when x is positive. Therefore, solving a verification problem of a network of n Relu nodes leads to solving 2^n linear sub-problems as illustrated in Figure 3. To address this issue, several approaches based on abstraction have been proposed. The next section provides more details about this category of techniques.

3 Abstraction approaches for neural network verification

In order to overcome the drawbacks of complete verification methods for NN, some abstraction approaches are proposed. The main idea behind these approaches consists in generating an abstract model from the original network ensuring that whenever the property $\boldsymbol{P}$ holds on the abstract model $\overline{\boldsymbol{N}}$, it necessarily holds on the original one $\boldsymbol{N}$, i.e.,:

$$\overline{N} \models P \implies N \models P. \tag{5}$$

However, these approaches may fail to provide any conclusion on the original network when the property is violated on the abstract model. This is in fact due to spurious counterexamples. Namely, when the property does not hold, a counterexample (CE) on the abstract model is generated, but due to the over-approximation of the abstract model, this CE might not correspond to any real behavior in the original model (i.e., spurious counterexample).

Concretely, the abstraction of NN can be performed in two different manners:

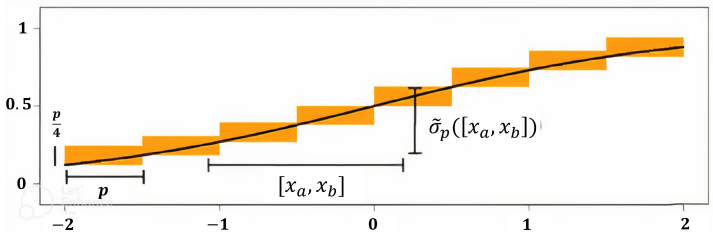


Figure 4: The activation function Sigmoid (σ) and its abstraction in $x \in [-2, 2]$. The solid line represents $y = \sigma(x)$ and each small region (yellow rectangles) is an over-approximation of y [37].

- *Activation function abstraction* (AF abstraction): to ease the verification process, non-linear activation functions of the NN are over-approximated by a set of linear constraints [14, 37, 42, 43].
- *NN model reduction*: abstracting the network model by merging some nodes in order to reduce the size of the network, and thus improve the scalability of existing verification tools.

A detailed survey of these methods is given in Sections 3.1 and 3.2, respectively.

Remark (Refinement). Some works consider improving the *incomplete* verification methods by ruling out as many spurious CE as possible by introducing a refinement phase. In other words, the verification method refines the abstract model iteratively until we can prove either the property holds or the generated CE exhibits a real behavior on the original model [11, 47, 48, 52, 53].

3.1 Abstraction of the activation function

The key challenge of NN verification is pertaining to the non-linearity of activation functions. AF abstraction-based verification approaches are applied to handle this issue by over-approximating the activation functions with linear constraints.

The earliest work dealing with NN verification problem was introduced by Pulina et al. [37]. In this work, authors divided the *Sigmoid* function into small regions, then a linear over-approximation is computed for each region, as shown in Figure 4.

With the same spirit, Ehlers [11] proposed a precise *Relu*-abstraction technique where *Relu* is replaced by a system of linear constraints (see Figure 5) and hence the verification problem of NN is reformulated as a linear programming (LP) problem that can be solved using classic LP solvers. The approach in [11] was implemented in a tool called Planet and brings the LP toolkit GLPK into play along with the Minisat solver for verification.

Gehr et al. [14] applied an *abstract interpretation* method [7] on NN for the first time. They proposed a framework called AI^2 (Abstract Interpretation for Artificial Intelligence) that soundly over-approximates NN operations by means of *zonotope* abstract domain[3]. The approach can be extended to support other abstract domains. AI^2 can handle feedforward and convolution neural networks (CNN) with *Relu* and *max-pooling* functions. The approach in [14] was extended by Singh et al. [42] to support *Sigmoid* and *Tanh* activation functions. This is accomplished by means of abstract transformers based on zonotopes for each function. As an example, the abstraction of *Relu* is given in Figure 6.a.

Furthermore, Singh et al. [43] proposed a new method, called *DeepPoly*, based on Abstract Interpretation by introducing a new abstract domain. DeepPoly combines floating point polyhedra and intervals. Each neuron is represented by its concrete and symbolic upper and lower bounds. Moreover, Singh et al. [43] introduced abstract transformers for popular NN operations: affine transformation, *Relu*, *Sigmoid*, *Tanh* and *Max-pooling* to propagate the inputs successively through the layers of the network. For *Relu*, two different abstractions are proposed as shown in Figures 6.b et 6.c. The approach supports both feedforward and convolution NN.

While the previous works consider only a single neuron, some others try to define sound approximations of a set of neurons, jointly. Singh et al. [41] introduced a new method that provides an

[3] An abstract domain is a set of logical constraints that define a geometric shape. The most popular abstract domains are: box (or Interval), zonotope and polyhedra. For example, a zonotope abstract domain [15] Z is defined by a set of constraints z_i, s.t: $z_i = a_i + \sum_{j=1}^{m} b_{ij}\epsilon_j$, where $\epsilon_j \in [l_i, u_i]$ is an error term and a_i, b_{ij} are constants.

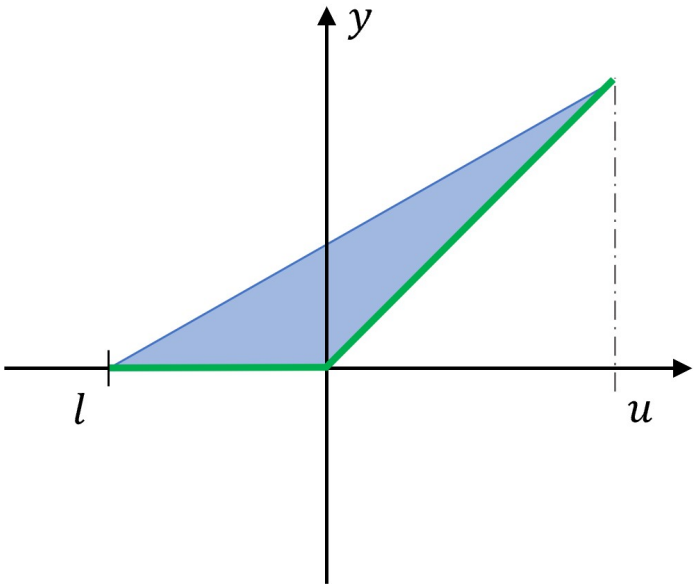


Figure 5: The abstraction of the *Relu* activation function proposed in [11]. The *Relu* ($d = relu(c)$) is represented by the black line and its over-approximation on the range $c \in [l, u]$ by the filled area.

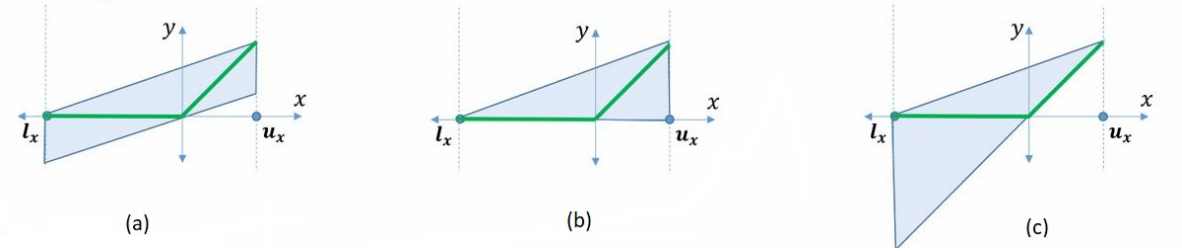


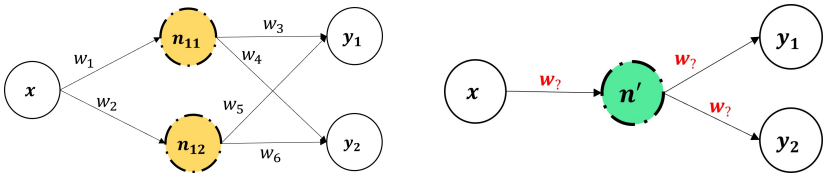
Figure 6: Relu activation function abstractions using different abstract domains

approximation of k *Relu* nodes (in the same layer) at a time in order to capture dependencies of the *Relu* inputs. First, the k nodes are selected and then the convex relaxation of the group of nodes is calculated. The framework has a parameter k which represents the number of *Relu* nodes to be considered together. A more general framework, based on [41], was recently proposed by Müller et al. [33]. The framework, called *PRIMA* (PRecIse Multi-neuron Abstraction), computes the convex over-approximation of a set of k outputs of arbitrary activation function, including *Relu*, *Sigmoid* and *Tanh*. The approach decomposes the n activations into overlapping groups of size k, then calculates the convex approximation of the octahedral over-approximation for each group i. Finally, it takes the union of all the obtained output constraints. These constraints combined with the encoding of the whole NN are used for verification.

A technique based on symbolic propagation is proposed in [27] to enhance the precision of abstract interpretation-based approaches. In this work, every neuron is associated with a symbolic formula expressed using the activations of neurons in its previous layers. In [44], a combination of over-approximation techniques with linear relaxation methods is proposed so as to gain more precision of over-approximation techniques and the scalability of complete methods.

3.2 NN model reduction

The main objective of NN model reduction is to reduce the size of the NN model while guaranteeing some behavioral relation: the desired property $\boldsymbol{P}$ holds on the original model $\boldsymbol{N}$ whenever it holds on the reduced model $\overline{\boldsymbol{N}}$ as defined in equation (5). Figure 7 provides an illustrative example of the main idea behind model reduction applied on a small neural network.



(a) An NN before abstraction (the concrete NN model)

(b) The NN after model reduction (the two hidden nodes are merged)

Figure 7: Model reduction of a small neural network

Such a behavioral relation is obtained by ensuring that $\overline{N}$ is an over-approximation of N (i.e. all behaviors of N can be reproduced in $\overline{N}$). Therefore, the reduction process must carefully select the set of neurons to be merged (or removed), and determine how to calculate the weights of the new edges.

Prabhakar and Afzal [36] proposed a method based on Interval Neural Networks (INN) for output range analysis. In this method, the nodes of the same layer are merged while replacing the weights of their input edges by the interval hull of the incoming edges. In other words, the weights of incoming edges are replaced by $[\min(W_{in}), \max(W_{in})]$, where W_{in} are the values of the incoming weights to the nodes to be merged. The weights of the outgoing edges from these nodes are replaced by the interval hull multiplied by the number of merged nodes n: $n \times [\min(W_{out}), \max(W_{out})]$.

For the verification part, Prabhakar and Afzal [36] adapted INN to MILP big-M encoding [4] and used the Gurobi MILP solver for verification. The performance of this method is tested on the airborne collision avoidance ACAS Xu benchmark [20, 21]. The authors claim that the abstraction enhances the verification process. Namely, Gurobi was not able to verify a number of properties on the original model (no return), while the same properties have been successfully checked when Gurobi was applied on the abstract model.

In [45], Sotoudeh and Thakur, by introducing the notion of Abstract Neural Network (ANN), provided a formalization of a general abstraction approach. In ANN, the weights are represented using abstract domains. Accordingly, the approach proposed by Prabhakar and Afzal [36] can be considered as a particular instantiation of this approach using the interval abstract domain. Notice that the proposed approach supports a wide range of activation functions. Moreover, it can be instantiated using other convex abstract domains and it is not restricted to intervals as used in INNs [36]. The approach provides a generic formula to calculate the weight merging matrix $\overline{W}$ from the original weight matrix W and the partitions P^{in} and P^{out} of two successive abstract layers l_i and l_{i+1}, respectively. A partition P_i is a rearrangement of a set S_i of neurons, i.e., if $S_i = \{n_{i1}, n_{i2}, n_{i3}\}$, a possible partition of S_i would be $P_i = \{\{n_{i1}, n_{i2}\}, \{n_{i3}\}\}$, which means that n_{i1} and n_{i2} will be merged in the abstract network. $\overline{W}$ is the convex combination (calculated by a function g) of the partitioning combination matrix of P^{in} and P^{out}, denoted by C and D, respectively, and the weight matrix W, i.e., $\overline{W} = g(D, W, C)$. Next, the abstract weight matrix, denoted by W_{abs}, is built by applying a convex abstract domain α_A on the obtained $\overline{W}$: $W_{abs} = \alpha_A(\overline{W})$. The reduced model is obtained by applying the same procedure to every layer, iteratively. Therefore, the obtained reduced model is an over-approximation for any non-negative activation function that satisfies the Weakened Intermediate Value Property (WIVP). Although some activation functions can have negative values and others are not continuous (thus not WIVP), the authors of [45] claim that there is always a way to overcome these problems, as they showed for Leaky *Relu* and the *threshold* activation functions.

In [1], Ashok et al. apply K-means clustering algorithms to partition each hidden layer l_i into k_i subgroups, such that $k_i \leq |S_i|$, then replace each subgroup with its representative neuron. The abstraction method, called DeepAbstract, has three parameters: the original network N, a finite set of input-points X and a vector K_L which contains the number of nodes on each abstract layer. For each hidden layer l_i, the following steps are performed:

1. For every $x \in X$, calculate the value $v_{ij}(x)$ of each neuron in S_i,
2. Apply K-means to split each layer l_i into k_i clusters. Let C_{l_i} denote the set of clusters of l_i,
3. For each cluster $C \in C_{l_i}$:
   (a) Determine the representative neuron rep_C,

(b) Calculate the corresponding outgoing weights of rep_C:

$$\overline{W}^i_*, rep_C = \sum_{n_{ij} \in C} W^i_*, n_{ij}$$

(c) Replace all the neurons in C with rep_C.

Note that the representative neuron rep_C of a cluster C is the nearest neuron to the centroid of C, thus; the incoming weights of rep_C remain the same as the corresponding neuron before abstraction. All the other neurons from cluster C are omitted with their incoming edges.

In addition, Ashok et al. [1] provide a method to lift the verification results from the abstract model to the original one using the DeepPoly verification Algorithm[4]. A set of experiments were conducted to check the performance of DeepAbstract. Local robustness of some MNIST images was checked and the authors claim that the verification time was reduced by 25% when DeepPoly is combined with DeepAbstract.

Elboher et al. [12] proposed an abstraction approach based on merging neurons of the same *category* (see hereafter) to build a smaller model so as to enhance the scalability of the existing verification tools. Regarding the verification property, which has the form: $P : \forall x \in pre(x) \implies y \leq c$, the aim of this approach is to build a reduced model $\overline{N}$ (its corresponding function is $\overline{\mathcal{N}}$), s.t $\forall x \in D_x$, $\overline{\mathcal{N}}(x) \geq \mathcal{N}(x)$. Therefore, $N \models P$ whenever $\overline{N} \models P$ (i.e., $\overline{\mathcal{N}}(x) \leq c$). First, each neuron is labelled according to the sign of its outgoing weights. A neuron is split if it has both positive and negative outgoing weights. Next, to guarantee that $\overline{N}$ is an over-approximation of N, the proposed method tries to increase the output of the abstract model by classifying each neuron as I or D. The class I means the output will increase by increasing the value of this neuron, while a neuron is marked as D when decreasing its value leads to increasing the output's value. Finally, the nodes of the same layer and the same category can be merged by summing up the weights of their outgoing edges and taking the *min* value of the the weights of their incoming edges if they are marked as D, or the *max* value for any I group of nodes. Moreover, some heuristics are proposed in [12] to enhance the abstraction process. The proposed method is applied on ACAS Xu networks while Marabou [22] is used as back-end verification tool. A comparison study between the abstraction method combined with Marabou and the vanilla version of Marabou was conducted, and the results showed that the abstraction method ~~helps~~ allows Marabou to verify more properties in less execution time.

A novel approach based on bisimulation [23] is proposed by Prabhakar [35]. The generated abstract neural network is equivalent, or bisimilar, to the original one. To guarantee the equivalence between N and $\overline{N}$, two neurons n_{ij} and n_{ik} to be merged must have the same activation function, the same bias value ($b_{ij} = b_{ik}$) and the same weights for each incoming edge respectively, i.e., $\forall n' \in S_{i-1}, w(n', n_{ij}) = w(n', n_{ik})$. Due to the strict conditions that, generally, do not hold in most of real networks, Prabhakar [35] extends the NN bisimulation to a more feasible relaxed method, called NN δ-bisimulation. Using NN δ-bisimulation ($\delta \in \mathbb{R}^+$), two nodes n_{ij} and n_{ik} in S_i can be merged if the following conditions are satisfied:

1. n_{ij} and n_{ik} have the same activation function
2. $|b_{ij} - b_{ik}| \leq \delta$
3. $\forall n' \in S_{i-1}, |w(n', n_{ij}) - w(n', n_{ik})| \leq \delta$

where $\delta \geq 0$. So the obtained network $\overline{N}$ is δ-bisimilar to network N.

Taking advantages of code refactoring [13], Shriver et al. [40] introduced the concept of refactoring neural networks to restructure the initial model and preserve its accuracy to enhance further operations on it, for instance verification. Concretely, NN refactoring consists of two steps: architecture transformation and distillation. The former applies some changes on the network's architecture by dropping or changing some layers and/or their types that are not supported by verification tools (e.g. residual blocks and convolutional layers). The latter updates the model's parameters: weights and biases, while preserving the original model's behavior, which is captured by its accuracy and test error according to Shriver et al. [40]. A tool called R4V was developed from this approach. R4V was tested on DAVE-2 [3] and DroNet [32] networks. The used verification tools are presented in Table 1. The

[4]Available at https://github.com/eth-sri/ERAN.

Name (if exists)	Pub. Year	Authors	Verification methods	Evaluation
R4V	2019	Shriver et al.	Relupex[21], ERAN[42], Neurify[52], Planet[11]	DAVE-2[3], DroNet[32]
INN	2019	Prabhakar et al.	MILP [31]	ACAS Xu [20, 21]
ANN	2020	Soutoudeh et al.	-	-
DeepAbstract	2020	Ashok et al.	ERAN	MNIST[24]
-	2020	Elboher et al.	Marabou[22]	ACAS Xu
Bisimulation	2021	Prabhakar	-	-

Table 1: A list of NN model reduction methods used for verification. The underscore symbol "−" is used to denote that no information is provided in the corresponding original paper.

results showed that applying the verification tools on the refactored model improves their scalability. For example, Planet [11] fails to check any property on DroNet within 24 hours. However, after refactoring the network, Planet was able to verify three out of the ten properties.

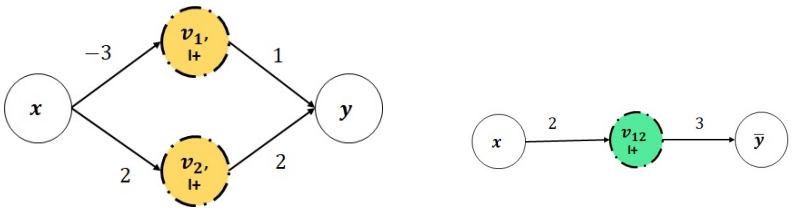
The main features of the above discussed neural networks reduction techniques are summarized in Table 1. The last two columns of the table contain verification methods and the data sets used during the evaluation of the abstraction method. Verification methods are those used during the evaluation of the abstraction in the original paper; notice that other methods can be used to verify the obtained abstract model.

It is worth mentioning that another family of techniques based on merging neurons and removing some edges without affecting the accuracy of the model exists in the literature. These techniques are called NN compression and acceleration, and their objective is to build a smaller network with low computational complexity, so that it can be embedded on devices with limited resources and used in real-time applications, while keeping the accuracy as high as possible [5, 17, 28]. Although both NN model reduction and NN compression strive to reduce the number of neurons, NN compression techniques cannot be used for verification, since the generated models do not fulfil the abstraction condition presented in formula (5). In other words, verifying a property P on the compressed network obtained by any compression method does not imply that the property does hold in the original network.

3.3 Discussion

This section discusses the aforementioned model reduction methods, while highlighting their limitations and proposing some possible area of improvements. In order to fairly compare the efficiency of the discussed approaches, we analyze them according to three main criteria (with respect to the available information in the original papers): (*i*) the precision of the over-approximation, (*ii*) the minimal number of neurons that can be obtained when the reduction method is applied until saturation, and (*iii*) the efficiency regarding the verification time and the number of the verified properties on the reduced model versus the original one.

The abstraction method based on INNs, proposed by Prabhakar et al. [36] seems to be very useful when the problem of output range analysis is considered. An exhaustive application of this method leads to merge all neurons of each hidden layer and replace them by one abstract neuron. The results of their paper show that the precision depends highly on the number and the selection of the nodes to be merged. The method needs some improvements to be more precise since no study has been provided for neuron selection. In addition, operations on intervals may impact the precision of this method. MILP encoding is proposed to solve the verification problem on INNs, and to the best of our knowledge, no other verification method is proposed to verify INNs. Moreover, this method is restricted to abstract NNs with non-negative activation functions [45]. Consequently, Sotoudeh et al. [45] proposed some fundamentals to abstract any NNs with different activation functions using any convex abstract domain and which is not limited to intervals. In [45], the authors provide an example of abstraction based on octagons but no explanation was given of the meaning of using such abstract



(a) The concrete model N after the classification of nodes (I+ for increasing positive node)

(b) The abstract model $\overline{N}$

Figure 8: Counterexample of Elboher et al. [12] abstraction method

domain to represent the merged neurons. Moreover, the work would have been more relevant if it had included an evaluation study to concretely show how the ANN can be extended to deal with other abstract domains.

DeepAbstract, proposed by Ashok et al. [1], is parametrized by the number of clusters on each layer; if there are few clusters, the model will be more abstract and less precise. In addition, this method relies on the discrete input set X that is used during clustering phase and can only verify the robustness of the model on points within this set X. Ashok et al. [1] claim that the verification time was reduced by 25% when DeepAbstract is used along with DeepPoly, however, only 195 out of 200 images could be verified to be robust against 197/200 when DeepPoly is used without abstraction.

The abstraction-refinement proposed by Elboher et al. [12] boosted the Marabou verifier to check more properties (58 out of 90 property versus 35/90). Moreover, the abstraction method reduces the total query median runtime from 63671 seconds to 1045 seconds. As a consequence of the classification of neurons, this method can abstract a layer to four neurons at most. This is one of the main drawbacks of this method since only neurons belonging to the same category can be merged. It should also be mentioned that only properties in the form: $y \leq c$ are considered, although authors claim that the approach is adaptable to cope with various types of properties by adjusting the output layer. In addition, this method cannot be applied if some neurons have negative values. For instance, this method cannot be applied in hidden layers if the used activation function returns negative values such as sigmoid and Leaky Relu. For the same reason, the first hidden layer cannot be abstracted if the inputs are negative. An example demonstrating this case is given in Figure 8, where x is an input, y is the output. The NN in Figure 8.b is generated using Elboher et al.'s method [12], which is supposed to be an abstraction of the original model of Figure 8.a. Both $\overline{N}$ and N use the *Relu* activation function on the hidden layer. Although for negative inputs the output of $\overline{N}$ is always zero: $\forall x \leq 0$, $\overline{y} = 0$, the output of N is always positive, for instance, for $x = -1$, $y = 3$, thus the condition of the over-approximation $\forall x \in D_x : \overline{\mathcal{N}}(x) \geq \mathcal{N}(x)$ does not hold.

The NN bisimulation method proposed in [35] guarantees the equivalence between abstract and original models, thus offers an exact abstraction. However, the set of conditions are hard to satisfy on real neural network, especially the condition on weights. On the other hand, the relaxed version, NN-δ-bisimulation, looks more feasible but needs further improvements to keep trace of the verified property on the abstract model and lift it to provide guarantees on the original network.

In [40], Shriver et al. propose an efficient approach with a dedicated tool, called R4V, to simplify and compress NN models. The wide experimental study they performed with different verification tools and data sets shows that R4V offers actual benefits to overcome the limitations of some NN verification techniques. However, this method enables to verify properties on the refactored model and does not propose a way to lift these guarantees to the original model. In other words, it does not provide any guarantee of whether the property holds on the original model.

Regarding the challenges of neural network verification, developing a new general approach that overcomes the issues related to the existing abstraction methods mentioned above is necessary. The works [1, 35, 40] could be adopted and combined with some heuristics to select candidate neurons to be merged. For instance, the δ-bisimulation method [35] can be used to select similar nodes by analysing their incoming weights. Approach in [1] can be adapted using discretization of the input region, so that the nodes that are close to each other (in the same cluster) are good candidates for abstraction.

If we focus on the abstract weights generated by the approach introduced in [12], we notice that the outgoing weights are always the sum of the corresponding edges on the original model. However,

depending on the category of neurons, the abstract incoming weights is either the ***min*** or the ***max***. On the other hand, the INN abstraction [36] and the ANN [45] techniques calculate the scaling of the weight matrix by multiplying the obtained outgoing weights (using an abstract domain) by the number of the merged neurons. By examining the results of these three approaches, one way to abstract a neural network is to take the convex hull of the incoming weights and sum the outgoing weights of the merged neurons. In particular, for the INN-based abstraction method [36], the sum of the outgoing weights can be considered rather than the scaled convex hull of these weights; this should lead to a tighter weighted-interval, hence more precise abstract model can be generated.

4 Conclusion

In this work, we discussed the problem of neural network verification and we presented different existing techniques used to solve this problem. We showed that the abstraction of neural networks can be used to help tackle the non-linearity and the complexity of the generated models. Abstraction of neural networks can be applied in two levels: abstracting the activation function and reducing the network's size (model reduction). While the abstraction of activation functions aims to over-approximate the non-linear activation functions with linear constraints, model reduction is used to reduce the number of neurons of the network. Both categories are applied to improve the verification process as a whole. The abstraction has to be sound, meaning that the desired behavior of the original model must be maintained. In this paper we focused more on model reduction methods since, to the best of our knowledge, no survey about neural networks reduction for verification purposes has been introduced.

Through highlighting advantages and limitations of each model reduction method, we proposed some guidelines for a more general model reduction approach. As a perspective, we aim to develop a new NN-reduction method that takes the advantage of the INN abstraction developed in [36] (no constraints for merging neurons), and the precision of the approach proposed by Elboher et al. [12]. Evaluation of its performances will be conducted on several available benchmarks. (This last sentence is so vague that it seems really useless. I think you should either remove it or give a bit more details about what you plan to do.)

References

[1] P. Ashok, V. Hashemi, J. Křetínskỳ, and S. Mohr. Deepabstract: Neural network abstraction for accelerating verification. In International Symposium on Automated Technology for Verification and Analysis, pages 92–107. Springer, 2020.

[2] A. Biere, M. Heule, and H. van Maaren. Handbook of satisfiability, volume 185. IOS press, 2009.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

[4] C.-H. Cheng, G. Nührenberg, and H. Ruess. Maximum resilience of artificial neural networks. In International Symposium on Automated Technology for Verification and Analysis, pages 251–268. Springer, 2017.

[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282, 2017.

[6] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, et al. Handbook of model checking, volume 10. Springer, 2018.

[7] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pages 238–252, 1977.

[8] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari. Output range analysis for deep neural networks. arXiv preprint arXiv:1709.09130, 2017.

[9] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward neural networks. In Proc. 10th NASA Formal Methods, pages 121–138, 2018.

[10] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. In UAI, volume 1, page 3, 2018.

[11] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In International Symposium on Automated Technology for Verification and Analysis, pages 269–286. Springer, 2017.

[12] Y. Y. Elboher, J. Gottschlich, and G. Katz. An abstraction-based framework for neural network verification. In International Conference on Computer Aided Verification, pages 43–65. Springer, 2020.

[13] M. Fowler. Refactoring: improving the design of existing code. Addison-Wesley Professional, 2018.

[14] 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 2018 IEEE Symposium on Security and Privacy (SP), pages 3–18. IEEE, 2018.

[15] K. Ghorbal, E. Goubault, and S. Putot. The zonotope abstract domain taylor1+. In International Conference on Computer Aided Verification, pages 627–633. Springer, 2009.

[16] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. MIT press, 2016.

[17] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

[18] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review, 37:100270, 2020.

[19] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In International conference on computer aided verification, pages 3–29. Springer, 2017.

[20] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pages 1–10. IEEE, 2016.

[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 International Conference on Computer Aided Verification, pages 97–117. Springer, 2017.

[22] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In International Conference on Computer Aided Verification, pages 443–452. Springer, 2019.

[23] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. Information and computation, 94(1):1–28, 1991.

[24] Y. LeCun. The mnist database of handwritten digits. urlhttp://yann.lecun.com/exdb/mnist/, 1998.

[25] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. nature, 521(7553):436–444, 2015.

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

[27] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, and L. Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In International Static Analysis Symposium, pages 296–319. Springer, 2019.

[28] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing, 461:370–403, 2021.

[29] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758, 2019.

[30] 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, 234:11–26, 2017.

[31] A. Lomuscio and L. Maganti. An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint arXiv:1706.07351, 2017.

[32] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza. Dronet: Learning to fly by driving. IEEE Robotics and Automation Letters, 3(2):1088–1095, 2018.

[33] M. N. Müller, G. Makarchuk, G. Singh, M. Püschel, and M. Vechev. Prima: general and precise neural network certification via scalable convex hull approximations. Proceedings of the ACM on Programming Languages, 6(POPL):1–33, 2022.

[34] A. R. Pathak, M. Pandey, and S. Rautaray. Application of deep learning for object detection. Procedia computer science, 132:1706–1717, 2018.

[35] P. Prabhakar. Bisimulations for neural network reduction. arXiv preprint arXiv:2110.03726, 2021.

[36] P. Prabhakar and Z. R. Afzal. Abstraction based output range analysis for neural networks. arXiv preprint arXiv:2007.09527, 2020.

[37] L. Pulina and A. Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In International Conference on Computer Aided Verification, pages 243–257. Springer, 2010.

[38] A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. arXiv preprint arXiv:1801.09344, 2018.

[39] D. Ristić-Durrant, M. Franke, and K. Michels. A review of vision-based on-board obstacle detection and distance estimation in railways. Sensors, 21(10):3452, 2021.

[40] D. Shriver, D. Xu, S. Elbaum, and M. B. Dwyer. Refactoring neural networks for verification. arXiv preprint arXiv:1908.08026, 2019.

[41] G. Singh, R. Ganvir, M. Püschel, and M. Vechev. Beyond the single neuron convex barrier for neural network certification. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

[42] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. Advances in neural information processing systems, 31, 2018.

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

[44] G. Singh, T. Gehr, M. Püschel, and M. Vechev. Boosting robustness certification of neural networks. In International Conference on Learning Representations, 2019.

[45] M. Sotoudeh and A. V. Thakur. Abstract neural networks. In International Static Analysis Symposium, pages 65–88. Springer, 2020.

[46] V. Tjeng, K. Y. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In International Conference on Learning Representations, 2019.

[47] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson. Verification of deep convolutional neural networks using imagestars. In International Conference on Computer Aided Verification, pages 18–42. Springer, 2020.

[48] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson. Star-based reachability analysis of deep neural networks. In International Symposium on Formal Methods, pages 670–686. Springer, 2019.

[49] H.-D. Tran, W. Xiang, and T. T. Johnson. Verification approaches for learning-enabled autonomous cyber-physical systems. IEEE Design & Test, 2020.

[50] D. Trentesaux, R. Dahyot, A. Ouedraogo, D. Arenas, S. Lefebvre, W. Schön, B. Lussier, and H. Cheritel. The autonomous train. In 2018 13th Annual Conference on System of Systems Engineering (SoSE), pages 514–520. IEEE, 2018.

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

[52] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. arXiv preprint arXiv:1809.08098, 2018.

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

[54] E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In International Conference on Machine Learning, pages 5286–5295. PMLR, 2018.

[55] M. Zhu, W. Min, Q. Wang, S. Zou, and X. Chen. Pflu and fpflu: Two novel non-monotonic activation functions in convolutional neural networks. Neurocomputing, 429:110–117, 2021.