Reachability Analysis of Neural Networks with Uncertain Parameters

Pierre-Jean Meyer

Reachability Analysis of Neural Networks with Uncertain Parameters

Pierre-Jean Meyer *

* *Univ Gustave Eiffel, COSYS-ESTAS, F-59666 Villeneuve d'Ascq, France (e-mail: pierre-jean.meyer@univ-eiffel.fr)*

Abstract: The literature on reachability analysis methods for neural networks currently only focuses on uncertainties on the network's inputs. In this paper, we introduce two new approaches for the reachability analysis of neural networks with additional uncertainties on their internal parameters (weight matrices and bias vectors of each layer), which may open the field of formal methods on neural networks to new topics, such as safe training or network repair. The first and main method that we propose relies on existing reachability analysis approach based on mixed monotonicity (initially introduced for dynamical systems). The second proposed approach extends the ESIP (Error-based Symbolic Interval Propagation) approach which was first implemented in the verification tool Neurify, and first mentioned in the publication of the tool VeriNet. Although the ESIP approach has been shown to often outperform the mixed-monotonicity reachability analysis in the classical case with uncertainties only on the network's inputs, we show in this paper through numerical simulations that the situation is greatly reversed (in terms of precision, computation time, memory usage, and broader applicability) when dealing with uncertainties on the weights and biases.

Keywords: Uncertain systems, reachability analysis, neural network.

1. INTRODUCTION

In the recent years, artificial intelligence methods have grown very rapidly and spread to numerous application fields. Although such approaches often work well in practice, the usual statistical testing of their behavior (Kim et al., 2020) becomes insufficient when dealing with safety-critical applications such as autonomous vehicles (Xiang et al., 2018). In such application fields, we instead need to develop formal verification approaches to guarantee the desired safe behavior of the system. In the case of neural networks, most formal verification tools rely on reachability analysis methods or on solving optimization problems (Liu et al., 2021), and they aim to verify safety specifications in the form of input-output conditions: check if, given a set of allowed input values, the set of all outputs reachable by the network (or an over-approximation of this set) remains within safe bounds (Bak et al., 2021).

Note however that all the tools mentioned in Liu et al. (2021); Bak et al. (2021) focus on the safety verification of pre-trained neural networks, i.e. the network parameters (weight matrices, bias vectors) are assumed to be fixed and known, and they only consider uncertainties on the network's input (from the safety specification to be checked). In contrast, in this paper we are interested in expending the reachability-based verification methods to a whole set of neural networks, or equivalently to a neural network with additional uncertainties on all its internal parameters (weight matrices and bias vectors). Such methods would in turn allow us to connect this field of neural network verification to new topics and offer new ways to approach problems such as safe training (ensuring during training that the trained network satisfies the desired properties, see e.g. Gowal et al. (2018); Mirman et al. (2018)) and network repair (finding the smallest changes to apply to an unsafe network in order to ensure its safety, see e.g. Majd et al. (2021); Yang et al. (2022)). This paper thus introduces the first necessary step in the development of such verification tools: creating new methods for the reachability analysis of neural networks with bounded inputs, weights and biases.

Contributions We propose two new methods to compute interval over-approximations of the output set of a neural network with bounded uncertainties on its inputs, weight matrices and bias vectors. The first approach and main contribution is based on mixed-monotonicity reachability analysis (initially introduced for the analysis of dynamical systems (Meyer et al., 2021)). One of the main strength of this approach is its generality since it is applicable to neural networks with any Lipschitz-continuous activation functions, unlike most other approaches in the literature which are limited to piecewise-affine (Wang et al., 2018a; Katz et al., 2019; Botoeva et al., 2020; Xu et al., 2021), sigmoid-shaped (Henriksen and Lomuscio, 2020; Tran et al., 2020; Müller, 2022) or monotone increasing functions (Dvijotham et al., 2018; Raghunathan et al., 2018). The proposed algorithm applies mixed-monotonicity reachability analysis to each partial network within the main neural network, and then intersects their results to obtain tighter over-approximations than if the reachability analysis was applied only once to the whole network directly.

Since, to the best of our knowledge, other approaches solving the considered problem have not yet been pro-

posed in the literature, we introduce a second method to offer some elements of comparison with the above mixed-monotonicity approach. This second algorithm extends to uncertain neural networks the ESIP (Error-based Symbolic Interval Propagation) method described in Henriksen and Lomuscio (2020). Although this ESIP approach is more limited in terms of activation functions (only piecewise-affine and sigmoid-shaped functions), this method was chosen here because it was shown in Meyer (2022) to be very computationally efficient in the particular case of uncertainties only on the network's input. However in this paper, numerical simulations show that with additional uncertainties on the network parameters, the mixed-monotonicity approach outperforms the ESIP algorithm on all relevant criteria: tightness of over-approximations, computation time and memory usage.

Related work Both methods proposed in this paper to tackle the reachability analysis problem on an uncertain neural network are generalizations of the methods presented in the particular case without uncertainties on the weights and biases of the network in Meyer (2022) for the mixed-monotonicity approach, and in Henriksen and Lomuscio (2020) for the ESIP approach. To the best of our knowledge, the only other publication attempting to consider a similar problem in the literature is Zuo et al. (2014). On the other hand, while the authors of this work indeed consider reachability analysis of uncertain neural networks, they do it in a very different setting of neural ordinary differential equations which does not allow us to provide any theoretical or numerical comparison with our approach on discrete models of feedfoward neural networks. As mentioned above, many existing works on safety verification of neural networks also rely on various algorithms and set representations for reachability analysis (see e.g. those listed in the survey paper Liu et al. (2021) or the neural network verification competition Bak et al. (2021)). However, all these works currently only apply their reachability methods to pre-trained neural networks, and thus without any uncertainty on the weight matrices and bias vectors as we consider in this paper.

This paper is organized as follows. Section 2 introduces the considered neural network model and defines the reachability analysis problem. Section 3 describe the first and main contribution of this paper, solving the considered problem with mixed-monotonicity reachability analysis. The second approach based on ESIP (Error-based Symbolic Interval Propagation) is introduced in 4. Finally, Section 5 provides numerical simulations to compare both algorithms and to highlight the advantages of the mixed-monotonicity approach.

2. PROBLEM DEFINITION

Given $\underline{x}, \overline{x} \in \mathbb{R}^n$ with $\underline{x} \leq \overline{x}$, the interval $[\underline{x}, \overline{x}] \subseteq \mathbb{R}^n$ is the set $\{x \in \mathbb{R}^n \mid \forall i \in \{1, \dots, n\},\ \underline{x}_i \leq x_i \leq \overline{x}_i\}$.

We consider an L-layer feedforward neural network defined as

$$x^l = \Phi(W^l x^{l-1} + b^l),\ \forall l \in \{1, \dots, L\} \tag{1}$$

with uncertain input vector $x^0 \in [\underline{x^0}, \overline{x^0}] \subseteq \mathbb{R}^{n_0}$, and uncertain weight matrix $W^l \in [\underline{W^l}, \overline{W^l}] \subseteq \mathbb{R}^{n_l \times n_{l-1}}$ and bias vector $b^l \in [\underline{b^l}, \overline{b^l}] \subseteq \mathbb{R}^{n_l}$ for each layer $l \in \{1, \dots, L\}$. The function Φ is defined as the componentwise application of a scalar and Lipschitz-continuous activation function. For simplicity of presentation, the activation function Φ is assumed to be identical for all layers.

In this paper, we are interested in the robustness of the neural network with respect to the uncertainties on its input x^0, weights W^l and biases b^l. Since the output set of the network cannot be computed exactly due to the nonlinearities in the activation function Φ, we use a simpler set representation (mutli-dimensional interval) to over-approximate this output set. Relying on such over-approximations ensures that any safety property satisfied on the computed interval is guaranteed to also be satisfied on the real output set of the neural network. This reachability analysis problem is formalized as follows.

Problem 1. Given the L-layer neural network (1) and the uncertainty sets $[\underline{x^0}, \overline{x^0}] \subseteq \mathbb{R}^{n_0}$, $[\underline{W^l}, \overline{W^l}] \subseteq \mathbb{R}^{n_l \times n_{l-1}}$ and $[\underline{b^l}, \overline{b^l}] \subseteq \mathbb{R}^{n_l}$ for all $l \in \{1, \dots, L\}$, find an interval $[\underline{x^L}, \overline{x^L}] \subseteq \mathbb{R}^{n_L}$ over-approximating the output set of (1):

$$\left\{ x^L \text{ in } (1) \,\middle|\, \begin{matrix} x^0 \in [\underline{x^0}, \overline{x^0}], W^l \in [\underline{W^l}, \overline{W^l}], \\ b^l \in [\underline{b^l}, \overline{b^l}], \forall l \in \{1, \dots, L\} \end{matrix} \right\} \subseteq [\underline{x^L}, \overline{x^L}].$$

The secondary goal is to find over-approximations that are as close to the real output set as possible. In this paper, we introduce two new approaches addressing this reachability analysis problem of neural networks with uncertain parameters, which has not been explored in the literature yet. The first and main contribution in Section 3 is based on mixed-monotonicity reachability analysis. The second proposed approach in Section 4 relies on Error-based Symbolic Interval Propagation (ESIP).

3. MIXED MONOTONICITY

3.1 Mixed-monotonicity reachability analysis

We first introduce the reachability analysis method for a general static function $y = f(x)$, which will then be applied multiple times to the various partial networks within (1) in the following sections. This result is a straightforward generalization to static functions $y = f(x)$ of the reachability analysis approach for discrete-time system $x^+ = f(x)$ proposed in Meyer et al. (2021). It relies on the boundedness assumption of the derivative (also called Jacobian matrix in the paper) of function f, which is satisfied by any Lipschitz-continuous function.

Proposition 1. Consider the function $y = f(x)$ with output $y \in \mathbb{R}^{n_y}$ and bounded input $x \in [\underline{x}, \overline{x}] \subseteq \mathbb{R}^{n_x}$. Assume that its derivative f' is bounded: for all $x \in [\underline{x}, \overline{x}]$, $f'(x) \in [\underline{J}, \overline{J}] \subseteq \mathbb{R}^{n_y \times n_x}$; and denote as J^* the center of these derivative bounds. For each output dimension $i \in \{1, \dots, n_y\}$, define input vectors $\underline{\xi^i}, \overline{\xi^i} \in \mathbb{R}^{n_x}$ and row vector $\alpha^i \in \mathbb{R}^{1 \times n_x}$ such that for all $j \in \{1, \dots, n_x\}$,

$$(\underline{\xi^i}_j, \overline{\xi^i}_j, \alpha^i_j) = \begin{cases} (\overline{x}_j, \underline{x}_j, \max(0, \overline{J}_{ij})) & \text{if } J^*_{ij} < 0, \\ (\underline{x}_j, \overline{x}_j, \min(0, \underline{J}_{ij})) & \text{if } J^*_{ij} \geq 0. \end{cases}$$

Then for all $x \in [\underline{x}, \overline{x}]$ and $i \in \{1, \dots, n_y\}$, we have:

$$f_i(x) \in \left[f_i(\underline{\xi^i}) - \alpha^i(\underline{\xi^i} - \overline{\xi^i}),\quad f_i(\overline{\xi^i}) + \alpha^i(\underline{\xi^i} - \overline{\xi^i}) \right].$$

Intuitively, the output bounds are obtained by computing for each output dimension the images for two diagonally opposite vertices of the input interval, then expanding these bounds with an error term when the bounds on the derivative f' spans both negative and positive values. Proposition 1 can thus provide an interval over-approximation of the output set of any function as long as bounds on the derivative f' are known. Obtaining such bounds for a neural network is made possible by computing local bounds on the derivative of its activation functions, as detailed in Section 3.2.

3.2 Local bounds of activation functions

Proposition 1 and the main algorithm in Section 3.3 are applicable to neural networks with any Lipschitz-continuous activation function Φ. This is indeed a sufficient condition for the derivative of the whole network description (1) to be bounded. On the other hand, knowing the values of these derivative bounds is required to apply Proposition 1 to the neural network. To avoid asking users of this method to compute themselves the derivative bounds of their neural network, we restrict our framework to a subset of Lipschitz-continuous activation functions for which we provide a method to automatically define local bounding functions for the derivative of a given activation function.

Assumption 1. Let $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$ and consider a scalar activation function Φ whose derivative is defined as $\Phi' : \mathbb{R}_\infty \to \mathbb{R}_\infty$, and where $\Phi'(x) \in \{-\infty, +\infty\}$ only if $x \in \{-\infty, +\infty\}$. The global arg min and arg max $\underline{z}, \overline{z} \in \mathbb{R}_\infty$ of Φ' are known, and Φ' is a 3-piece piecewise-monotone function as follows:

- non-increasing on $(-\infty, \underline{z}]$ until reaching its global minimum $\min_{x \in \mathbb{R}_\infty} \Phi'(x) = \Phi'(\underline{z})$;
- non-decreasing on $[\underline{z}, \overline{z}]$ until reaching its global maximum $\max_{x \in \mathbb{R}_\infty} \Phi'(x) = \Phi'(\overline{z})$;
- and non-increasing on $[\overline{z}, +\infty)$.

When $\underline{z} = -\infty$ (resp. $\overline{z} = +\infty$), the first (resp. last) monotone segment is squeezed into a singleton at infinity and can thus be ignored.

While the formulation of this assumption may seem restrictive (compared to the initial assumption of taking any Lipschitz-continuous activation function), it should be noted that the large majority of activation functions in the literature indeed have a derivative as described in Assumption 1, including all the less common non-monotone activation functions reviewed or introduced in Zhu et al. (2021). [1] Therefore, the mixed-monotonicity approach proposed in Section 3.3 has a much broader applicability than most neural network verification tools in the literature, which are most often restricted to ReLU and piecewise-affine activation functions (Wang et al., 2018a; Katz et al., 2019; Botoeva et al., 2020; Xu et al., 2021), occasionally able to consider sigmoid-shaped functions (Henriksen and Lomuscio, 2020; Tran et al., 2020; Müller, 2022), and very rarely dealing with general monotone activation functions (Dvijotham et al., 2018; Raghunathan et al., 2018).

[1] More details and examples on activation functions satisfying Assumption 1 are available in Meyer (2022) where this assumption was first introduced.

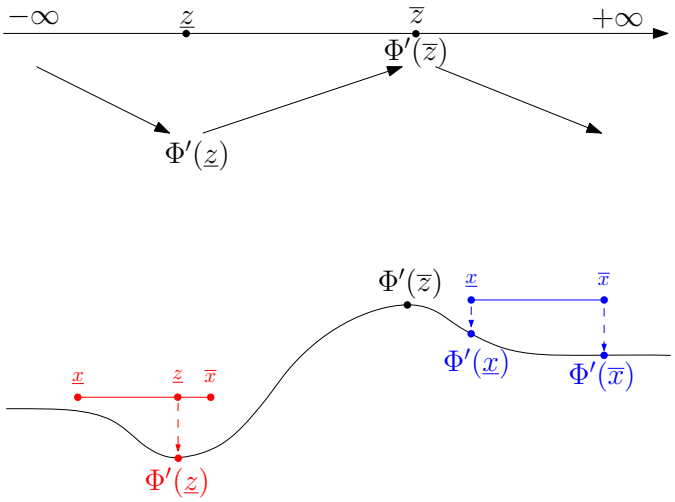

Fig. 1. *Top*: General shape for the activation function derivative according to Assumption 1. *Bottom*: Two examples for the computation of the local lower bound of Φ' depending on whether its global arg min $\underline{z}$ is contained in the input interval $[\underline{x}, \overline{x}]$ (in red) or not (in blue).

Proposition 2. Given an activation function Φ satisfying Assumption 1 and a bounded input domain $[\underline{x}, \overline{x}] \in \mathbb{R}$, the local bounds of the derivative Φ' on $[\underline{x}, \overline{x}]$ are given by:

$$\min_{x \in [\underline{x}, \overline{x}]} \Phi'(x) = \begin{cases} \Phi'(\underline{z}) & \text{if } \underline{z} \in [\underline{x}, \overline{x}], \\ \min(\Phi'(\underline{x}), \Phi'(\overline{x})) & \text{otherwise,} \end{cases}$$

$$\max_{x \in [\underline{x}, \overline{x}]} \Phi'(x) = \begin{cases} \Phi'(\overline{z}) & \text{if } \overline{z} \in [\underline{x}, \overline{x}], \\ \max(\Phi'(\underline{x}), \Phi'(\overline{x})) & \text{otherwise.} \end{cases}$$

In short, as long as the user provides Φ' and its global arg min and arg max ($\underline{z}$ and $\overline{z}$), Proposition 2 returns a local bounding function of Φ'. An illustration of Assumption 1 and Proposition 2 (for the lower bound of Φ') is provided in Fig. 1. The local bounding of Φ' in Proposition 2 is used in Section 3.3 for the computation of bounds on the Jacobian matrix of the neural network, which is required to apply the mixed-monotonicity reachability result from Proposition 1.

3.3 Main algorithm

In this section, we propose an approach using both Propositions 1 and 2 to solve Problem 1 and obtain the tightest possible interval over-approximation of the neural network output set that can be computed when applying mixed-monotonicity reachability analysis to (1). The proposed algorithm is inspired by the one introduced in Meyer (2022) in the particular case of a pre-trained neural network with only uncertainties on its input vector x^0.

Although Proposition 1 can be applied to any partial network (described as a subset of consecutive layers of (1)), we do not know in advance which decomposition into partial networks yields the best results. Algorithm 1 thus proposes an efficient way to apply this mixed-monotonicity reachability analysis on all possible network decompositions, while avoiding any redundant computation. To achieve this, we explore the layers of (1) iteratively, and apply Proposition 1 to each partial network ending at the current

layer. All the obtained interval over-approximations of this layer's output set are then intersected to obtain a significantly tighter over-approximation, which will then be used in the computations of the next layers.

These main steps are summarized in Algorithm 1 and described below. The algorithm takes as input the neural network description (1) with an activation function Φ satisfying Assumption 1, as well as all the intervals bounding the network's uncertainties: network input x^0 and the weight matrices W^l and bias vectors b^l for each layer $l \in \{1, \dots, L\}$. For a partial network considering only layers k to l (with $k \leq l$) of (1) and denoted as $\texttt{NN}(k, l)$, we use the notations: $u(k, l)$ for the concatenated vector of all its uncertainties (the partial network's input x^{k-1}, and the elements of all W^i and b^i for $i \in \{k, \dots, l\}$); $J(k, l)$ for the derivative (or Jacobian matrix) of this partial network with respect to $u(k, l)$; and $x(k, l)$ for the output of this partial network that we want to over-approximate.

Since the Jacobian bounds are defined iteratively using products, they are initialized in line 1 as identity matrices. Then, for each layer l (lines 2 to 7), we first use Proposition 2 to compute local bounds on the activation function derivative Φ' when the pre-activation variable is the result of the affine transformation of layer l (line 3, using interval arithmetic operators): $[\underline{W^l}, \overline{W^l}] * [\underline{x^{l-1}}, \overline{x^{l-1}}] + [\underline{b^l}, \overline{b^l}]$.

Next, we consider independently each partial network covering from some previous layer $k \in \{1, \dots, l\}$ to the current layer l (lines 4 to 6). The first step in line 5 is to compute bounds on the Jacobian matrix of partial network $\texttt{NN}(k, l)$. Using the chain rule, we know that bounds on the derivative of $\texttt{NN}(k, l)$ with respect to all uncertainties in $u(k, l)$ are given by the product:

$$[\underline{J(k,l)}, \overline{J(k,l)}] = [\underline{\Phi'}, \overline{\Phi'}] * [\underline{G(k,l)}, \overline{G(k,l)}], \qquad (2)$$

where $G(k, l)$ is the derivative of $\texttt{NN}(k, l)$ without the last activation function. The bounds on $G(k, l)$ are defined as the horizontal concatenation of:

$$[\underline{W^l}, \overline{W^l}] * [\underline{J(k, l-1)}, \overline{J(k, l-1)}]$$

representing the derivative with respect to all uncertainties in $\texttt{NN}(k, l-1)$;

$$[\underline{x^{l-1}}_1, \overline{x^{l-1}}_1] * I_{n_l}, \dots, [\underline{x^{l-1}}_{n_{l-1}}, \overline{x^{l-1}}_{n_{l-1}}] * I_{n_l}$$

representing the derivative with respect to the weight matrix W^l of the last layer; and I_{n_l} representing the derivative with respect to the bias vector b^l of the last layer. Using the computed Jacobian bounds $[\underline{J(k,l)}, \overline{J(k,l)}]$ along with the known uncertainty bounds $[\underline{u(k,l)}, \overline{u(k,l)}]$, we can then apply Proposition 1 to the partial network $\texttt{NN}(k, l)$ in line 6 to obtain an interval over-approximation $[\underline{x(k,l)}, \overline{x(k,l)}]$ of the output set of $\texttt{NN}(k, l)$.

The last step of the algorithm, in line 7, is to take the intersection of the interval over-approximations obtained for each partial neural network ending at layer l. Algorithm 1 then returns the interval $[\underline{x^L}, \overline{x^L}]$ computed at the last layer L of the network.

Theorem 1. The interval $[\underline{x^L}, \overline{x^L}]$ returned by Algorithm 1 is a solution to Problem 1.

Proof. Proposition 1 is guaranteed in Meyer et al. (2021) to return an interval over-approximation for the output

Input: L-layer network (1) with activation function Φ satisfying Assumption 1, uncertainties $[\underline{x^0}, \overline{x^0}] \subseteq \mathbb{R}^{n_0}$, $[\underline{W^l}, \overline{W^l}] \subseteq \mathbb{R}^{n_l \times n_{l-1}}$ and $[\underline{b^l}, \overline{b^l}] \subseteq \mathbb{R}^{n_l}$ for all $l \in \{1, \dots, L\}$

1 $\forall k, l \in \{0, \dots, L\},\ \underline{J(k,l)} \leftarrow I, \overline{J(k,l)} \leftarrow I$

2 **for** $l \in \{1, \dots, L\}$ **do**

3 $\quad [\underline{\Phi'}, \overline{\Phi'}] \leftarrow \texttt{Prop2}(\Phi', [\underline{W^l}, \overline{W^l}] * [\underline{x^{l-1}}, \overline{x^{l-1}}] + [\underline{b^l}, \overline{b^l}])$

4 $\quad$ **for** $k \in \{1, \dots, l\}$ **do**

5 $\quad\quad$ Compute $[\underline{J(k,l)}, \overline{J(k,l)}]$ as in (2)

6 $\quad\quad [\underline{x(k,l)}, \overline{x(k,l)}] \leftarrow$ $\texttt{Prop1}(\texttt{NN}(k,l), [\underline{u(k,l)}, \overline{u(k,l)}], [\underline{J(k,l)}, \overline{J(k,l)}])$

7 $\quad [\underline{x^l}, \overline{x^l}] \leftarrow [\underline{x(1,l)}, \overline{x(1,l)}] \cap \dots \cap [\underline{x(l,l)}, \overline{x(l,l)}]$

Output: Over-approximation $[\underline{x^L}, \overline{x^L}]$ of the network output

Algorithm 1: Mixed-monotonicity reachability analysis of an uncertain feedforward neural network.

set of the considered system. The theorem statement can then be proved by induction. For layer 1, Algorithm 1 computes $[\underline{x^1}, \overline{x^1}] = [\underline{x(1,1)}, \overline{x(1,1)}]$, which is indeed an over-approximation of the output set of layer 1 under all uncertainties on x^0, W^1 and b^1. Next, assuming that intervals $[\underline{x^1}, \overline{x^1}]$, ..., $[\underline{x^{l-1}}, \overline{x^{l-1}}]$ over-approximate the output sets of layers 1 to $l-1$ respectively, then according to Proposition 1 each interval $[\underline{x(k,l)}, \overline{x(k,l)}]$ over-approximates the output set of layer l. If the true output set of layer l is included in each of these intervals, it is then included in their intersection $[\underline{x^l}, \overline{x^l}]$. ■

The proof of Theorem 1 thus primarily relies on the fact that the intersection operator preserves the soundness of over-approximations. Another benefit of the use of intersections comes into play in the fact that Algorithm 1 proposes an exhaustive exploration of all possible decomposition of (1) into partial networks. Indeed, this ensures that the final result of Algorithm 1 is at least as tight as the one that would be obtained from using Proposition 1 on any specific decomposition into partial networks. And in practice, the result of Algorithm 1 is often strictly tighter than any other result from specific decomposition, due to the fact that the intersection of multiple intervals is strictly smaller than each individual interval in most cases (except in the case where one is included in all others, in which case the intersection is equal to this smallest interval).

4. ERROR-BASED SYMBOLIC INTERVAL PROPAGATION

Although the primary contribution of this paper is the mixed-monotonicity approach presented in Section 3, numerically evaluating its performances is of great importance as for any computational method on neural networks. Since we were not able to find any relevant and comparable approaches in the literature, we developed a second method solving Problem 1, to be able to provide numerical comparisons between them in Section 5.

The approach proposed in this section relies on symbolic interval propagation, which has been used in several neural network verification tools such as ReluVal (Wang et al.,

2018b), Neurify (Wang et al., 2018a) and VeriNet (Henriksen and Lomuscio, 2020). The core idea is to create bounding functions depending linearly in the network input, and propagating these linear functions iteratively through the layers of the network (i.e. through both the layer's affine transformation, and the linear relaxations of the nonlinear activation functions (Salman et al., 2019)). This ensures that the dependency to the network's input is preserved during the propagation of these bounding functions through the network, which results in significantly tighter reachable set over-approximations compared to naive interval bound propagation approaches (see e.g. Xiang et al. (2020)) where this input dependency is lost at each layer.

In this paper, we are interested in a particular variation called ESIP (Error-based Symbolic Interval Propagation), which was first introduced (but not published) in the implementation of the tool Neurify (Wang et al., 2018a) and first published in the paper of the tool VeriNet (Henriksen and Lomuscio, 2020). Unlike the classical approach designing and propagating two linear functions (for the lower and upper bounds, respectively), ESIP relies on a single linear function alongside an error matrix. The symbolic equation represents the behavior of the network if the nonlinear activation function of each node is replaced by its lower linear relaxation. The error matrix accounts for deviations from this symbolic equation induced by nodes operating at the upper linear relaxation of their activation function. This particular approach is chosen here because it was shown in Meyer (2022) to be very computationally efficient, with a low and constant computation time per node in the network, while other methods had a computation time per node that grew with the size of the network.

Compared to the ESIP implementation in Henriksen and Lomuscio (2020) in the case where the neural network only has uncertainties on its input, we propose here an extension of this approach to also handle uncertainties on the weight matrices and bias vectors and thus to be able to solve Problem 1. This extension is summarized in Algorithm 2 and detailed below. Due to space limitations and since this new approach is primarily introduced for comparison with our main contribution in Section 3, the formal proof that Algorithm 2 solves Problem 1 is left for future work. We refer the reader to Henriksen and Lomuscio (2020) for more theoretical details in the case of uncertainties only on the network input x^0.

In line 1, we initialize the uncertainty vector u^0 to the input x^0, the symbolic equation S^0 to the identity function, and the error $[\underline{E}, \overline{E}]$ to and empty interval matrix. Then, for each layer l we first propagate these variables through the affine transformation $x \to W^l * x + b^l$, by appending all elements of W^l and b^l at the end of the previous uncertainty vector u^{l-1} (line 3), updating the symbolic equation with respect to the new uncertainties W^l and b^l introduced in this layer (line 4), and multiplying the error by the bounds on W^l (line 5).

Next, propagating through the layer's activation function is done individually for each node i of the layer (line 6). We first compute concrete bounds of the pre-activation variable (line 7) by evaluating the symbolic equation S_i^l on the current uncertainty bounds $[\underline{u^l}, \overline{u^l}]$, and then adding all negative error terms to the lower bound and all positive errors to the upper bound. These pre-activation bounds can then be used in line 8 to compute a linear relaxation of the activation function, i.e. two linear functions $\underline{r}$ and $\overline{r}$ such that $\underline{r}(x) \leq \Phi(x) \leq \overline{r}(x)$ for all $x \in [\underline{x^l}_i, \overline{x^l}_i]$. More details on how to compute such linear relaxations can be found e.g. in Xu et al. (2021) for ReLU functions and in Henriksen and Lomuscio (2020) for sigmoid-shaped functions. We then propagate the symbolic equation through the lower relaxation $\underline{r}$ (line 9) and compute the maximal error between the relaxation bounds over the pre-activation range $[\underline{x^l}_i, \overline{x^l}_i]$ (line 10). These new error terms are appended at the end of both bounds of the error (line 11).

For the final layer L, the propagation of the symbolic equation and error through the activation function (lines 8-11) can be skipped since the interval over-approximation of the output set can be simply computed by propagating the pre-activation bounds (from line 7) through the activation function. Note that in line 12, we obtain these bounds by applying Φ directly to the lower and upper bounds because this class of approaches relying on linear relaxations are currently limited in the literature (either in their theory or their implementation) to monotone increasing activation functions (Wang et al., 2018a; Henriksen and Lomuscio, 2020; Zhang et al., 2018).

Input: L-layer network (1), uncertainties $[\underline{x^0}, \overline{x^0}] \subseteq \mathbb{R}^{n_0}$, $[\underline{W^l}, \overline{W^l}] \subseteq \mathbb{R}^{n_l \times n_{l-1}}$ and $[\underline{b^l}, \overline{b^l}] \subseteq \mathbb{R}^{n_l}$ for all $l \in \{1, \dots, L\}$

1 $u^0 \leftarrow x^0$, $S^0(u^0) \leftarrow x^0$, $\underline{E} \leftarrow []$, $\overline{E} \leftarrow []$

2 **for** $l \in \{1, \dots, L\}$ **do**

`/* Affine transformation */`

3 $u^l \leftarrow [u^{l-1}; W^l(:); b^l]$

4 $S^l(u^l) \leftarrow W^l * S^{l-1}(u^{l-1}) + b^l$

5 $[\underline{E}, \overline{E}] \leftarrow [\underline{W^l}, \overline{W^l}] * [\underline{E}, \overline{E}]$

6 **for** $i \in \{1, \dots, n_l\}$ **do**

`/* Pre-activation bounds */`

7 $[\underline{x^l}_i, \overline{x^l}_i] \leftarrow S_i^l([\underline{u^l}, \overline{u^l}]) + [\sum_{j|\underline{E}(i,j)<0} \underline{E}(i,j), \sum_{j|\overline{E}(i,j)>0} \overline{E}(i,j)]$

`/* Activation function */`

8 find $\underline{r}, \overline{r} \mid \underline{r}(x) \leq \Phi(x) \leq \overline{r}(x), \forall x \in [\underline{x^l}_i, \overline{x^l}_i]$

9 $S_i^l(u^l) \leftarrow \underline{r}(S_i^l(u^l))$

10 $e_i \leftarrow \max(\overline{r}(\underline{x^l}_i) - \underline{r}(\underline{x^l}_i), \overline{r}(\overline{x^l}_i) - \underline{r}(\overline{x^l}_i))$

11 $\underline{E} \leftarrow [\underline{E}, diag(e)]$, $\overline{E} \leftarrow [\overline{E}, diag(e)]$

12 $[\underline{x^L}, \overline{x^L}] \leftarrow [\Phi(\underline{x^L}), \Phi(\overline{x^L})]$

Output: Over-approximation $[\underline{x^L}, \overline{x^L}]$ of the network output

Algorithm 2: ESIP reachability analysis of an uncertain feedforward neural network.

Algorithm 2 is very similar to the ESIP approach described in Henriksen and Lomuscio (2020) in the particular case without weight and bias uncertainty. The main differences are that in our Algorithm 2, the dimension of uncertainty vector u^l grows at each layer, and the error needs to be described as an interval matrix (due to the product with uncertain weight W^l in line 5) instead of a simple matrix in Henriksen and Lomuscio (2020).

In terms of implementation of the algorithm however, there is a much more significant difference with Henriksen and Lomuscio (2020): the symbolic equation which was a linear function in the network intput x^0 in Henriksen and Lomuscio (2020) is now a multi-linear function in the uncertainty vector u^l. This introduces a significantly higher complexity in terms of implementation and memory usage. Indeed, in Henriksen and Lomuscio (2020) the symbolic equation of layer l could be simply defined as an $n_l \times (n_0 + 1)$ matrix, where for each of the n_l output nodes of this layer we only need to store n_0 values for the factors multiplying the terms of x^0, and the final value for the constant term of the linear equation. On the other hand, for the multi-linear function S^l in Algorithm 2, we would similarly need to store one factor for each multi-linear term appearing in the equation. The creation of such huge matrices thus limits the application of this ESIP approach to very shallow and narrow neural networks, as illustrated in Example 1 below.

Example 1. The initial symbolic equation S^0 has dimensions $n_{S^0} = n_0 \times (n_0 + 1)$. Next if S^{l-1} is stored as an $n_{l-1} \times n_{S^{l-1}}$ matrix, then the affine transformation at layer l ($S^l(u^l) = W^l * S^{l-1}(u^{l-1}) + b^l$), implies that the new symbolic equation S^l has $n_l * n_{l-1} * n_{S^{l-1}}$ columns for the multi-linear and linear terms in $W^l * S^{l-1}(u^{l-1})$, followed by n_l columns for the linear terms in b^l, and a final column for the constant of the symbolic equation (which becomes a non-zero value only after its propagation through the activation function). Therefore, S^l is stored as an $n_l \times n_{S^l}$ matrix, with $n_{S^l} = n_l * n_{l-1} * n_{S^{l-1}} + n_l + 1$.

For simplicity, assume that all layers have the same width: $\exists n \in \mathbb{N} \mid \forall l \in \{0, \dots, L\},\ n_l = n$. Then we can prove by induction that the width of the matrix for S^l is $n_{S^l} = \sum_{i=0}^{2l+1} n^i$. We can thus conclude that the final symbolic equation S^L is of dimensions:

$$n \times \left(\frac{1 - n^{2L+2}}{1 - n} \right).$$

Therefore, the complexity of Algorithm 2 is exponential in the depth L of the network, and polynomial in its width n.

In Matlab, matrices cannot contain more than $2^{48} - 1 \approx 2.8 * 10^{14}$ elements. To illustrate the high memory usage of this approach, we show in Table 1 the maximum width n of an L-layer network for the symbolic equation S^L to remain within this Matlab constraint.

Depth L	1	2	3	4	5	6
Max allowed width n	4095	255	63	27	15	10

Table 1. Maximum width n of the network for the symbolic equation S^L to be storable in Matlab.

Note however that this constraint on the matrix dimension is never actually reached in practice, since we would first reach another limitation related to the actual weight of this matrix compared to the available RAM. Taking the same conditions that will be considered in the numerical example of Section 5 with a network of depth $L = 3$ and width $n = 20$, the matrix storing the symbolic equation would weigh up to 215 GB (counting 8 bytes per element in the matrix). This is significantly higher than the available RAM on most computers, which will result in a crash of Matlab when attempting to create such matrix. ▲

5. NUMERICAL EXAMPLES

In this section, we provide a numerical comparison of Algorithms 1 and 2 on a set of randomly generated neural networks with various dimensions and activation functions, and we highlight the better performances of the mixed-monotonicity approach from Section 3 with respect to most criteria (generality, tightness, computation time, memory usage). Both algorithms are implemented in Matlab 2021b and run on a laptop with 1.80GHz processor and 16GB of RAM.

In our first numerical experiment, we consider neural networks as in (1) with increasing depth L and a fixed uniform width n for all input, hidden and output layers (i.e. $n_l = n$ for all $l \in \{0, \dots, L\}$). Since we already know from Example 1 that the ESIP approach will struggle in terms of complexity and memory usage, we focus this comparison on narrow networks with $n = 20$ nodes per layer. All uncertainty variables (input, weight matrices, bias vectors) are assigned randomly generated bounds within $[-1, 1]$. The simulation are run a total of $N = 10$ times, each with different random uncertainty bounds, and the obtained results in terms of width of the interval over-approximation, computation time and memory usage are averaged over this number of runs. Since the original ESIP implementation in VeriNet (Henriksen and Lomuscio, 2020) is limited to piecewise-affine or sigmoid-shaped activation functions, we focus this first comparison on the most popular activation function: Rectified Linear Unit (ReLU), which is the piecewise-affine function $\Phi(x) = \max(0, x)$.

In Table 2 are summarized the obtained results for both Algorithm 1 using mixed-monotonicity and Algorithm 2 using the ESIP approach. In terms of the width of the computed interval over-approximations, we first notice that both algorithms return identical results for shallow networks ($L = 1$), but that the mixed-monotonicity approach always generates tighter intervals for networks with hidden layers. In terms of complexity, the superiority of the mixed-monotonicity approach is striking, since the computation times are on average 12 times faster than ESIP with one layer, and up to 7000 times faster with two layers. Similarly the memory usage is on average 1.4 times smaller than ESIP with one layer, and 176 times smaller with two layers. As predicted in Example 1, as soon as we add a third layer, the ESIP approach attempts to create a matrix much larger than the available 16 GB of RAM (even despite the use of sparse matrices), which causes Matlab to crash. On the other hand, we can see that the mixed-monotonicity approach from Algorithm 1 still behaves well in terms of complexity (time and memory) for deeper networks, although the conservativeness of the over-approximation naturally increases with the size of the network.

The second set of numerical experiments is run only by the mixed-monotonicity approach in Algorithm 1 and focuses on its performances while dealing with the main two limitations of the ESIP approach that we could not explore in the comparison of Table 2: larger networks and

		Mixed-monotonicity	ESIP
$L = 1$	**Width**	21.2	21.2
	Time (s)	0.067	0.79
	Memory (MB)	0.16	0.22
$L = 2$	**Width**	201	263
	Time (s)	0.25	368
	Memory (MB)	0.46	81
$L = 3$	**Width**	2144	–
	Time (s)	0.64	–
	Memory (MB)	0.90	> 16000
$L = 4$	**Width**	21796	–
	Time (s)	1.4	–
	Memory (MB)	1.5	–
$L = 5$	**Width**	284625	–
	Time (s)	2.5	–
	Memory (MB)	2.2	–
$L = 6$	**Width**	2866970	–
	Time (s)	4.2	–
	Memory (MB)	3.1	–
$L = 10$	**Width**	44421135944	–
	Time (s)	18	–
	Memory (MB)	7.9	–

Table 2. Average width $\|\overline{x^L} - \underline{x^L}\|_2$ of the interval over-approximation, computation time (in seconds) and memory usage (in megabytes) for both algorithms over 10 ReLU networks.

more uncommon activation functions. Indeed, the ESIP approach is not only limited by its complexity, but as mentioned in Section 4, Algorithm 2 and its original version in VeriNet (Henriksen and Lomuscio, 2020) also cannot handle non-monotone activation functions. Here, we thus consider neural networks with the Sigmoid Linear Unit (SiLU) activation function, which is the non-monotone function $\Phi(x) = x/(1+e^{-x})$ introduced in Ramachandran et al. (2017). This SiLU activation function satisfies Assumption 1 (with global arg min and arg max of its derivative defined as $\underline{z} = -2.3994$ and $\overline{z} = 2.3994$, respectively), and it is thus natively handled by the mixed-monotonicity approach in Algorithm 1.

Tables 3 and 4 report the average (over $N = 10$ randomly generated uncertainty bounds as in the previous test) computation time and memory usage, respectively, when the depth L of the neural network goes from 1 to 10 and its width n from 20 to 100. Although both these quantities naturally increase with the size (L or n) of the network, we can observe that Algorithm 1 could solve Problem 1 on all neural networks of up to 10 layers and 100 neurons per layer, in less than an hour and using less than 1 GB of RAM. This is a significant advantage compared to Algorithm 2 which took over 6 minutes per network for a 2-layer 20-width network, and over hundreds of GB for a 3-layer network. After plotting the obtained results from Tables 3 and 4 using log and various n-th roots to identify the growth rates, we have identified that the mixed-monotonicity approach in Algorithm 1 has a polynomial complexity in $O(n^3 * L^3)$ for the computation time and $O(n^3 * L^2)$ for the memory.

6. CONCLUSIONS

In this paper, we consider the reachability analysis problem for neural networks with uncertainties not only on their inputs, but also on all their weight matrices and bias vectors. To the best of our knowledge, this problem has not yet been addressed in the literature. We propose two approaches to tackle this problem. The first one and our main contribution relies on a repeated call of mixed-monotonicity reachability analysis on each partial network within the main neural network. The second approach, primarily provided to offer some elements of comparison with our first approach in this yet unexplored topic, extends to uncertain networks the ESIP (Error-based Symbolic Interval Propagation) approach from Henriksen and Lomuscio (2020). In both the theoretical sections and the numerical simulations, we highlight the superiority of the mixed-monotonicity approach with respect to all criteria relevant to solving the considered problem. Indeed, the algorithm is widely applicable to networks with any Lipschitz-continuous activation function, while the ESIP approach is limited to piecewise-affine and sigmoid-shaped functions. In terms of computation time and memory usage, the mixed-monotonicity approach has only a polynomial complexity in both the depth and width of the network, while the ESIP complexity is exponential in the network's depth which limits it to only shallow networks. Finally, on all networks where the ESIP algorithm could be run, the mixed-monotonicity approach returns tighter interval over-approximation of the output set (or equal to ESIP in the case of a single-layer network).

While most verification tools in the neural network literature currently focus on verifying a single pre-trained neural network, the work presented in this paper instead analyzes a whole family of neural networks for any weight matrices and bias vectors in their respective bounds. This opens the door to new topics such as safe training (finding the subset of weight and bias values such that the resulting networks satisfy a given property) or network repair (finding the minimal changes to apply to a given network in order to

	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
$L = 1$	0.068	0.40	1.4	3.6	7.7
$L = 2$	0.24	2.0	7.2	18	38
$L = 3$	0.66	5.6	21	51	108
$L = 4$	1.4	12	43	108	225
$L = 5$	2.6	21	76	191	402
$L = 6$	4.1	34	124	312	665
$L = 7$	6.2	54	191	486	1028
$L = 8$	9.0	77	276	705	1499
$L = 9$	13	109	389	990	2109
$L = 10$	17	146	526	1333	2844

Table 3. Average computation time (in seconds) of Algorithm 1 over 10 SiLU networks.

	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
$L = 1$	0.16	1.1	3.7	8.6	17
$L = 2$	0.46	3.3	11	26	50
$L = 3$	0.89	6.6	22	51	99
$L = 4$	1.5	11	36	85	164
$L = 5$	2.2	16	54	127	246
$L = 6$	3.0	23	76	178	345
$L = 7$	4.1	31	101	237	459
$L = 8$	5.2	39	130	304	590
$L = 9$	6.5	49	162	380	737
$L = 10$	7.9	60	198	464	901

Table 4. Average memory usage (in megabytes) of Algorithm 1 over 10 SiLU networks.

make it satisfy a given property), which will be the main focus of our future work.

REFERENCES

Bak, S., Liu, C., and Johnson, T. (2021). The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. *arXiv preprint arXiv:2109.00498*.

Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., and Misener, R. (2020). Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3291–3299.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., and Kohli, P. (2018). A dual approach to scalable verification of deep networks. In *UAI*, 550–559.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., and Kohli, P. (2018). On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*.

Henriksen, P. and Lomuscio, A. (2020). Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, 2513–2520. IOS Press.

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

Kim, E., Gopinath, D., Pasareanu, C., and Seshia, S.A. (2020). A programmatic and semantic approach to explaining and debugging neural network based object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11128–11137.

Liu, C., Arnon, T., Lazarus, C., Barrett, C., and Kochenderfer, M.J. (2021). Algorithms for verifying deep neural networks. *Foundation and Trend in Optimization*, 4(3-4), 244–404.

Majd, K., Zhou, S., Amor, H.B., Fainekos, G., and Sankaranarayanan, S. (2021). Local repair of neural networks using optimization. *arXiv preprint arXiv:2109.14041*.

Meyer, P.J. (2022). Reachability analysis of neural networks using mixed monotonicity. *IEEE Control Systems Letters*, 6, 3068–3073.

Meyer, P.J., Devonport, A., and Arcak, M. (2021). *Interval Reachability Analysis: Bounding Trajectories of Uncertain Systems with Boxes for Control and Verification*. Springer.

Mirman, M., Gehr, T., and Vechev, M. (2018). Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, 3578–3586. PMLR.

Müller, M.N. (2022). ERAN: ETH robustness analyzer for neural networks. URL `https://github.com/eth-sri/eran`.

Raghunathan, A., Steinhardt, J., and Liang, P. (2018). Certified defenses against adversarial examples. In *International Conference on Learning Representations*.

Ramachandran, P., Zoph, B., and Le, Q.V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Salman, H., Yang, G., Zhang, H., Hsieh, C.J., and Zhang, P. (2019). A convex relaxation barrier to tight robustness verification of neural networks. *arXiv preprint arXiv:1902.08722*.

Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., and Johnson, T.T. (2020). NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, 3–17. Springer.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018a). Efficient formal safety analysis of neural networks. In *32nd Conference on Neural Information Processing Systems*.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018b). Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium*.

Xiang, W., Musau, P., Wild, A.A., Lopez, D.M., Hamilton, N., Yang, X., Rosenfeld, J., and Johnson, T.T. (2018). Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989*.

Xiang, W., Tran, H.D., Yang, X., and Johnson, T.T. (2020). Reachable set estimation for neural network control systems: A simulation-guided approach. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5), 1821–1830.

Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., and Hsieh, C.J. (2021). Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*.

Yang, X., Yamaguchi, T., Tran, H.D., Hoxha, B., Johnson, T.T., and Prokhorov, D. (2022). Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 221–236. Springer.

Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., and Daniel, L. (2018). Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31, 4939–4948.

Zhu, M., Min, W., Wang, Q., Zou, S., and Chen, X. (2021). PFLU and FPFLU: Two novel non-monotonic activation functions in convolutional neural networks. *Neurocomputing*, 429, 110–117.

Zuo, Z., Wang, Z., Chen, Y., and Wang, Y. (2014). A non-ellipsoidal reachable set estimation for uncertain neural networks with time-varying delay. *Communications in Nonlinear Science and Numerical Simulation*, 19(4), 1097–1106.