Formal Monotony Analysis of Neural Networks with Mixed Inputs: An Asset for Certification

Guillaume Vidot, Mélanie Ducoffe, Christophe Gabreau, Iulian Ober, Ileana Ober

Formal Monotony Analysis of Neural Networks with Mixed Inputs: An Asset for Certification

Guillaume Vidot[1,2(✉)], Mélanie Ducoffe[1], Christophe Gabreau[1], Ileana Ober[2], and Iulian Ober[3]

[1] Airbus Opération S.A.S, Toulouse, France
{eric-guillaume.vidot,melanie.ducoffe,christophe.gabreau}@airbus.com
[2] University of Toulouse, Institut de Recherche en Informatique de Toulouse, Toulouse, France
{eric.vidot,ileana.ober}@irit.fr
[3] ISAE-SUPAERO, University of Toulouse, Toulouse, France
iulian.ober@isae-supaero.fr

Abstract. The use of ML technology to design safety-critical systems requires a complete understanding of the neural network's properties. Among the relevant properties in an industrial context, the verification of partial monotony may become mandatory. This paper proposes a method to evaluate the monotony property using a Mixed Integer Linear Programming (MILP) solver. Contrary to the existing literature, this monotony analysis provides a lower and upper bound of the space volume where the property does not hold, that we denote "Non-Monotonic Space Coverage". This work has several advantages: (i) our formulation of the monotony property works on discrete inputs, (ii) the iterative nature of our algorithm allows for refining the analysis as needed, and (iii) from an industrial point of view, the results of this evaluation are valuable to the aeronautical domain where it can support the certification demonstration. We applied this method to an avionic case study (braking distance estimation using a neural network) where the verification of the monotony property is of paramount interest from a safety perspective.

Keywords: Neural network verification · Monotony · Certification · Formal Methods

1 Introduction

Over the last years, neural networks have become increasingly popular and a reference method for solving a broad set of problems, such as computer vision, pattern recognition, obstacle detection, time series analysis, or natural language processing. Their usage in safety-critical embedded systems (e.g., automotive, aviation) is also becoming increasingly appealing. The aeronautical domain is known to be one of the more stringent. Indeed, products are ruled by binding regulation requirements to guarantee that the aircraft will safely operate in foreseeable operating

and environmental conditions. At the end of 2021, the European Union Aviation Safety Agency (EASA) released the first issue of a concept paper [5] to anticipate any application for AI-based products: it contains *a first set of technical objectives and organization provisions that EASA anticipates as necessary for the approval of Level 1 AI applications ('assistance to human') and guidance material which could be used to comply with those objectives.*

Among all the properties required to certify AI-based systems, robustness is of paramount importance and is a widely studied property in the machine learning and verification communities [2,14,16,18,21,23,24,27–29,31,32]. Although robustness is a critical property for classification tasks, we see the emergence of safety-related properties for regression tasks in many industries. For example, numerical models have been developed to approximate physical phenomena inherent to their systems [1]. As these models are based on physical equations, whose relevancy is asserted by scientific experts, their qualification is carried out without any issue. However, since their computational costs and running time prevent us from embedding them on board, the use of these numerical models in the aeronautical field remains mainly limited to the development and design phase of the aircraft. Thanks to the current success of deep neural networks, previous works have already investigated neural network-based surrogates for approximating numerical models [12,22]. Those surrogates come with additional safety properties linked to induced physics. One of them is the monotony which is motivated by the fact that monotonic functions are ubiquitous in most physics phenomena. For instance, one should expect that the estimate of a braking distance should be a monotonic function with respect to specific parameters such as the state of the brakes (nominal or degraded) or the state of the runway (dry or wet). Another case where monotony is relevant is in DNNs used for control. Today, state-of-the-art methods for enforcing partial monotony assume that if the property is not respected on the whole operational domain of the ML-based function, this puts at risk its certification (i.e., its compliance determination to certification requirements) and, therefore, its industrialization. We believe that this risk can be covered, especially for models that, in the future, would also be penalized for being too loose given the reference function. We propose an iterative method that measures and identifies the part of the domain for which the monotony property is violated, which can be used to demonstrate conformity to certification requirements.

2 Certification Preamble

As stated before, to certify a product, the following principle shall apply to the systems composing that product: each system performs its intended function and safely operates in foreseeable operating and environmental conditions. It means that even if the system performs a function with poor performance (this is obviously not desirable from an industrial viewpoint), it can be certifiable if the product's safety is guaranteed in the usage domain. Once that is said, a primordial principle emerges: the safety is to be considered at the system level,

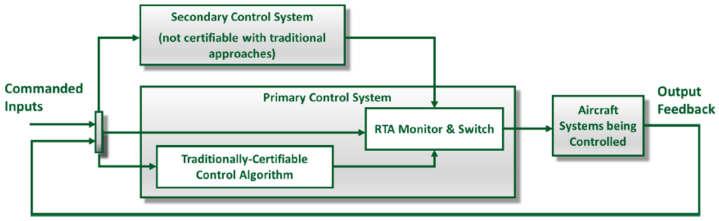

Fig. 1. Typical runtime assurance architecture proposed by NASA [20]

meaning that even if a specific algorithm is not robust in some areas of its input space, then the system can remain certifiable if one can demonstrate that any unsafe behavior is prevented or mitigated. One possible mitigation of this risk is the use of runtime assurance illustrated in Fig. 1 extracted from the NASA paper [20], which ensures the system's safety by the design of redundant control system architecture, where a certifiable function takes over the function not certifiable with traditional approaches when unsafe context is detected. This mitigation is relevant regardless of the technology used during the system development. Coming back to the specific context of ML-based development, the ability to formally define the areas of the input space where the safety properties of the model are not preserved can be a powerful asset in the compliance determination to certification requirements. The use of formal methods can save significant testing efforts while preserving the safe behavior of the function.

3 Related Work

In recent years, assessing the robustness of neural networks has been tackled with formal verification (i.e., sound algorithms demonstrating whether a property holds or not). Verifying properties on a neural network is challenging because neural networks are non-convex functions with many non-linearities, with hundreds to millions of parameters. Even if the type of architecture studied is restricted to piecewise-linear networks, it is known that this problem is already NP-hard [29]. There has been tremendous progress in the field of verification, from robustness proof of networks trained on MNIST to scaling up to CIFAR 10 and even TinyImagenet. These successes are particularly due to the collaboration of different communities that made it possible to formulate and tackle the robustness problem from different perspectives. Without being exhaustive, we can cite the methods that rely on Lipschitz-based optimization [32,33], input refinement [27] and semi-definite relaxations [21].

So far, the verification community has mainly tackled robustness verification from adversarial robustness [26] to computing the reachable set of a network [30] despite a few other properties that are highly relevant for the industry. Among those properties, partial monotony under specific inputs appears to be a key property, especially for regression tasks. Indeed, the need for monotony appeared in various contexts such as surrogate modeling [11], economics [6],

fairness [13], or interpretability [19] and is thus a highly desirable feature in the industry. Previous works proposed to enforce the monotony property during the design; In [9], they relied on heuristics regularizers to promote monotony, whose main drawback lies in the absence of guarantees and therefore will require verification as a post-processing step. On the other side, [7] and [15] adopted hand-designed monotonic architectures, which may harden the training and not perform well in practice. Lastly, up to our knowledge, previous works mainly considered monotony under continuous inputs, while many industrial use-cases have monotony constraints on discrete inputs. One notable exception is the fairness verification in [25] that can be applied on both a discrete or a continuous input and holds similarity with monotony verification.

When it comes to continuous inputs, monotony is equivalent to verifying a property on the gradients on the whole domain. Indeed the sign of the gradient component corresponding to monotonous inputs should always be positive or negative. However, for a neural network with discrete inputs, the gradient sign condition is not necessary for the monotony to hold, even when the gradient can be computed by extending the input domain to reals. For piece-wise linear neural networks such as ReLU networks, we can base verification on the very definition of monotony (Definition 1), which can be cast as solving a mixed-integer linear programming problem. This method is complementary to the literature using the gradient condition and can verify monotony over discrete inputs.

Verifying the monotony is recognized to be more challenging than robustness since it is a global property on the whole domain rather than a local neighborhood [15]. However, we argue that applying partial monotony over the whole domain, which may affect the performance and put at risk the product's release, is a very drastic approach. Indeed, in an industrial context, it is necessary to balance quality and safety, especially as the systems will be constrained by other specifications than just monotony, such as accuracy. The solution we propose is a partitioning scheme that splits the operational domain into areas where the monotony property is respected and areas where it is (partially) violated; in the latter, the neural network's behavior could be mitigated. This possibility has been considered on a collision detection use case in [4] and studied at a higher level for the certification of a system before an ML component [17].

4 Monotony Analysis

In this section, we define the concept of *partial* monotony with respect to a set of inputs. Let $\mathcal{V}$ be a (finite) set of input features. For each feature $v \in \mathcal{V}$ we denote $\mathcal{D}(v)$ the domain in which v ranges. Hence, let $X = \times_{v \in \mathcal{V}} \mathcal{D}(v)$ be the input space, Y be the output space and $f{:}X \to Y$ be the neural network. Note that the features are generally of two types ($\mathcal{V} = \mathcal{V}_d \sqcup \mathcal{V}_c$):

- $v \in \mathcal{V}_d$ are features whose domain $\mathcal{D}(v)$ is discrete (*e.g.*, a finite set of labels or categorical values)
- $v \in \mathcal{V}_c$ are features whose domain $\mathcal{D}(v)$ is a real interval

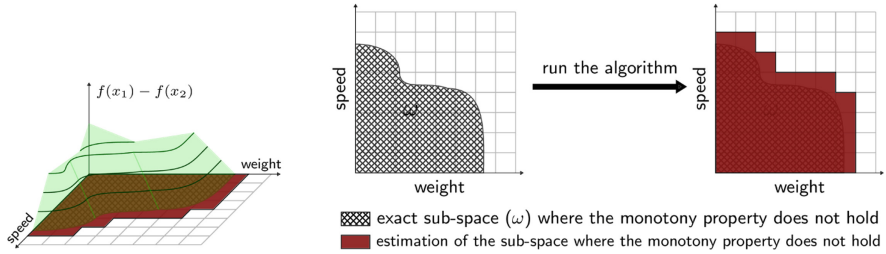

Fig. 2. Purpose of the algorithm through Example 1. In x_1 the runway is dry and in x_2 the runway is wet. The left plot represents $f(x_1) - f(x_2)$ where only the positive values are displayed (monotony property violated). The two plots on the right are the projection of these points on the plane composed of feature 1 and 2.

In this work, we are interested in monotony properties, which supposes that the set Y has an order relation denoted $\preceq$; usually, $Y \subseteq \mathbb{R}$ and $\preceq$ is one of the usual orders ($\leq$, $\geq$). The monotony property will be relative to a subset of discrete features, $\alpha \subseteq \mathcal{V}_d$ for which a partial order is defined on $\times_{v \in \alpha} \mathcal{D}(v)$, also denoted $\preceq$ without risk of confusion. For $x \in X$, let us denote $x{\downarrow_\alpha}$ the projection of x onto the dimensions in α, and $\bar{\alpha} = \mathcal{V} \backslash \alpha$.

Definition 1. *Monotony Property*
A function f is monotone with respect to an order $\preceq$ on the output domain Y and to a subset of discrete features $\alpha \subseteq \mathcal{V}_d$ endowed with a partial order defined on $\times_{v \in \alpha} \mathcal{D}(v)$ also denoted $\preceq$ (without risk of confusion) if and only if

$$\forall (x_1, x_2) \in X^2 : x_1 \downarrow_{\bar{\alpha}} = x_2 \downarrow_{\bar{\alpha}} \ \wedge x_1 \downarrow_\alpha \preceq x_2 \downarrow_\alpha \implies f(x_1) \preceq f(x_2)$$

4.1 Goal of the Analysis

Our analysis aims to identify the sub-spaces where the monotony does not hold using a MILP solver. Example 1 describes a toy example (a simplified version of the case study in Sect. 5) that we will use to explain the main concepts.

Example 1. Setup: Let f be a neural network estimating the braking distance of an aircraft based on its speed, its weight and the runway's state (dry or wet).
Property: *for the same speed and weight ($x_1 \downarrow_{\bar{\alpha}} = x_2 \downarrow_{\bar{\alpha}}$), the braking distance on a wet runway must be higher than on a dry one ($x_1 \downarrow_\alpha \preceq x_2 \downarrow_\alpha \implies f(x_1) \preceq f(x_2)$).*
Goal: Identify and quantify the input areas where the property does not hold.

If we plot $f(x_1) - f(x_2)$ versus the speed and the weight, the Definition 1 holds if and only if all the values are negative. The 3D plot in Fig. 2 shows a sketch of this example when the monotony property partially holds, *i.e.* $f(x_1) - f(x_2)$ is partially positive. To ease the visualization we only draw the positive values. The crosshatched area in the 2D plots are projections of the positive values of the curve on the plane representing the speed and the weight features and

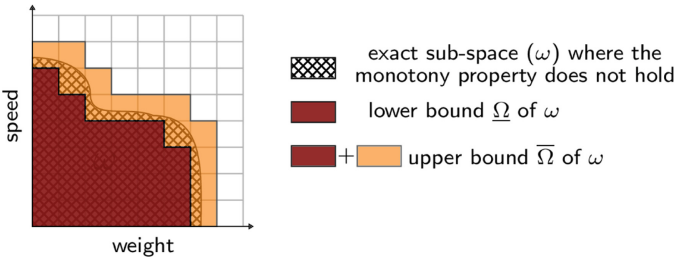

Fig. 3. Based on Fig. 2: representation of $\underline{\Omega}$ and $\overline{\Omega}$ considering Example 1 (Color figure online)

models the area where the monotony property is not respected, namely the Non-Monotonic Space Coverage denoted as ω. The rightmost 2D plot shows what we expect from our analysis on Example 1: identifying and estimating ω. To estimate ω, we partition the space (grid in Fig. 2) and then the monotony property is checked on each sub-spaces. The dark red area represents the identified sub-space where monotony issues occur, i.e., an over-approximation of ω. In addition, our approach provides a lower and upper bound of the size of ω relative to the whole input domain, respectively denoted as $\underline{\Omega}$ and $\overline{\Omega}$ (See Fig. 3).

Our approach can distinguish the sub-spaces where the monotony property does not hold (dark red area in Fig. 3) from the ones where it partially holds (orange area in Fig. 3). Hence, the lower bound is the dark red area, while the upper bound is the dark red and orange areas. The benefit of having a lower and upper bound, instead of just an overestimation, is to be able to assess whether our estimation is precise: large gaps between the upper and lower bound may reveal that our bounds are not representative of ω. The iterative nature of our approach overcomes this problem: we refine our space, which leads to a finer grid for the Fig. 3 and run again the MILP solver where the property partially holds to have a most accurate estimation of ω.

4.2 MILP Formulation

Neural Network Encoding. Let $f : X \rightarrow Y$ be a neural network composed of n layers with ReLU activations. The layer 0 corresponds to the input layer while the layer n to the output one. We use the MILP formulation proposed by [3], which uses the big-M method [8] to encode the ReLU activation. By convention, the notations in bold denote the MILP variables, and those not in bold denote constants. For $1 \leq i \leq (n-1)$, let C^i be the conjunction of constraints for the layer i:

$$C^i \triangleq \quad \hat{\mathbf{x}}^{\mathbf{i}} = W^i \mathbf{x}^{\mathbf{i-1}} + b^i \tag{1}$$

$$\wedge\, \mathbf{x}^{\mathbf{i}} \leq \hat{\mathbf{x}}^{\mathbf{i}} + M^i(1 - \mathbf{a}^{\mathbf{i}}) \qquad \wedge \qquad \mathbf{x}^{\mathbf{i}} \geq \hat{\mathbf{x}}^{\mathbf{i}} \tag{2}$$

$$\wedge\, \mathbf{x}^{\mathbf{i}} \leq M^i \cdot \mathbf{a}^{\mathbf{i}} \qquad \wedge \qquad \mathbf{x}^{\mathbf{i}} \geq 0 \tag{3}$$

$$\wedge\, \mathbf{a}^{\mathbf{i}} \in \{0,1\}^{|\mathbf{x}^{\mathbf{i}}|} \tag{4}$$

where $\hat{\mathbf{x}}^{\mathbf{i}}$ and $\mathbf{x}^{\mathbf{i}}$ are the vector of neuron values at the layer i before and after the ReLU activation respectively. M^i is a large valid upper bound s.t. $-M^i \leq \hat{\mathbf{x}}^{\mathbf{i}}$ and $\mathbf{x}^{\mathbf{i}} \leq M^i$ [3]. W_i and b_i are, respectively, the weights and bias at the layer i, and $\mathbf{a}^{\mathbf{i}}$ is a binary vector that represents whether the neurons are activated or not. The Eq. (1) is the constraint for the affine transformation and the Eqs. (2)–(4) are the constraints encoding the ReLU activation. For the output layer n, there is no ReLU activation, then we have:

$$C^n \triangleq \quad \hat{\mathbf{x}}^{\mathbf{n}} = W^n \mathbf{x}^{\mathbf{n-1}} + b^n \tag{5}$$

It remains to encode the constraints of the input layer, which enforce the lower and upper bounds of the domain of the input features. Our analysis relies on a partition of the input space X, thus the encoding of the input layer will depend on it: let $\mathcal{P}$ be a partition of X, $p \in \mathcal{P}$ be a subset of X represented by a set of linear constraints (also denoted p). Hence, the neural network f is encoded as the conjunction of the constraints defined for each layer and p which is constraining the input layer:

$$C^f(p) \triangleq p \wedge \left(\bigwedge_{i=1}^{n} C^i \right) \wedge C^n \tag{6}$$

Monotony Property Encoding. Following Definition 1, we must encode f twice in MILP: C_1^f and C_2^f. Similarly to the encoding of the input space's constraints, we encode the monotony property regarding the partition $\mathcal{P}$. So, let $p_i, p_j \in \mathcal{P}^2$ be two sub-spaces of X such that $\exists x_1, x_2 \in p_i \times p_j$, $x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}} \wedge x_1 \prec x_2$. Then, we have:

$$C^{mon}(p_i, p_j) \triangleq \left(\mathbf{x_1^0}{\downarrow_{\bar{\alpha}}} = \mathbf{x_2^0}{\downarrow_{\bar{\alpha}}} \wedge \mathbf{x_1^0}{\downarrow_{\alpha}} \preceq \mathbf{x_2^0}{\downarrow_{\alpha}} \right) \wedge \left(C_1^f(p_i) \wedge C_2^f(p_j) \right) \wedge \left(\hat{\mathbf{x}}_1^{\mathbf{n}} \leq \hat{\mathbf{x}}_2^{\mathbf{n}} \right) \tag{7}$$

$$C^{\neg mon}(p_i, p_j) \triangleq \left(\mathbf{x_1^0}{\downarrow_{\bar{\alpha}}} = \mathbf{x_2^0}{\downarrow_{\bar{\alpha}}} \wedge \mathbf{x_1^0}{\downarrow_{\alpha}} \preceq \mathbf{x_2^0}{\downarrow_{\alpha}} \right) \wedge \left(C_1^f(p_i) \wedge C_2^f(p_j) \right) \wedge \left(\hat{\mathbf{x}}_1^{\mathbf{n}} \geq \hat{\mathbf{x}}_2^{\mathbf{n}} + \epsilon \right) \tag{8}$$

The MILP solver may output either `SAT`, `UNSAT` or `TIMEOUT`. For (7) and (8), `TIMEOUT` means that the time limit is reached. C^{mon} checks whether the neural network f is monotonic:

- `SAT`: there is an assignment for $\mathbf{x_1^0}, \mathbf{x_2^0} \in p_i \times p_j$ which respects the monotony.
- `UNSAT`: the monotony is violated on the entire sub-space $p_i \times p_j$.

$C^{\neg mon}$ checks whether the neural network is not monotonic:

- `SAT`: there is an assignment for $\mathbf{x_1^0}, \mathbf{x_2^0} \in p_i \times p_j$ which violates the monotony.
- `UNSAT`: the monotony is respected on the complete sub-space $p_i \times p_j$.

To avoid having `SAT` for $C^{\neg mon}$ when $\hat{\mathbf{x}}_1^{\mathbf{n}} = \hat{\mathbf{x}}_2^{\mathbf{n}}$, we introduce the ϵ term (Eq. 8).

To determine for each sub-space $p_i \times p_j$ whether the monotony property holds, partially holds, or does not hold (see Fig. 3), we must solve successively $C^{\neg \text{mon}}$ and C^{mon} (see Sect. 4.3 for more detail).

Algorithm 1. Monotony analysis refinement

Require: T: the number of iteration of the procedure

1: $P_1 \leftarrow \{(p_i, p_j) \in \mathcal{P}^2 \mid \exists (x_1, x_2) \in p_i \times p_j,\ x_1 \prec x_2 \text{ and } x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}}\}$
2: $\underline{\Omega_0} \leftarrow 0$ and $\widehat{P}_0 \leftarrow P_1$
3: $\Omega \leftarrow [\,]$, $\mathbb{P}^{\neg\text{mon}} \leftarrow \emptyset$, and $\mathbb{P}^{\text{partially mon}} \leftarrow \emptyset$
4: **for** t from 1 to T **do**
5: $\quad \widehat{P}_t \leftarrow \widehat{P}_{t-1} \wedge P_t$
6: $\quad (P_t^{\neg\text{mon}}, P_t^{\text{partially mon}}) \leftarrow \mathbf{F}(\widehat{P}_t)$
7: $\quad \underline{\Omega_t} \leftarrow \underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|}$ and $\overline{\Omega_t} \leftarrow \underline{\Omega_t} + \frac{|P_t^{\text{partially mon}}|}{|P_t|}$ {See Fig. 4}
8: $\quad \Omega \leftarrow \Omega + (\underline{\Omega_t}, \overline{\Omega_t})$
9: $\quad \mathbb{P}^{\neg\text{mon}} \leftarrow \mathbb{P}^{\neg\text{mon}} \cup P_t^{\neg\text{mon}}$ and $\mathbb{P}^{\text{partially mon}} \leftarrow P_t^{\text{partially mon}}$
10: $\quad \widehat{P}_t \leftarrow P_t^{\text{partially mon}}$
11: $\quad P_{t+1} \leftarrow \texttt{partition}(P_t)$
12: **end for**
13: **return** Ω, $\mathbb{P}_{\neg\text{mon}}$ and $\mathbb{P}_{\text{partially mon}}$

4.3 Verification Procedure

As explained in Sect. 4.1, our verification procedure implies the partition of the space and the verification of each sub-space. In Algorithm 1 the monotony property is iteratively analyzed regarding a partition while refining this partition in the zone of interest to sharpen the analysis. Algorithm 2 details the verification run at each iteration.

Algorithm 1. The monotony is defined on the space X^2; however, we define earlier the partition $\mathcal{P}$ of X. Hence to verify the monotony on the complete space, *i.e.* X^2, we need to go through all the sub-spaces *i.e.* $p_i \times p_j$, $\forall (p_i, p_j) \in \mathcal{P}^2$. However, it may happen that the monotony does not apply to the sub-space (p_i, p_j) because there are no comparable elements within the sub-space: P_1, in Line 1, contains all and only the (p_i, p_j) including comparable elements. We denote the elements of P_1 and more generally, P_t, "monotony scenario".

We propose an iterative procedure where at each iteration we use, in Line 6, $\mathbf{F}(\cdot)$ (see Algorithm 2) to retrieve $P_t^{\neg\text{mon}}$ and $P_t^{\text{partially mon}}$. Then, we compute in Line 7 the metrics $\underline{\Omega_t}$ and $\overline{\Omega_t}$ for the iteration t, which respectively lower-bounds and upper-bounds ω; ω is the exact ratio of the space where f is not monotonic, which corresponds to the ratio of monotony scenarios where f is not monotonic. In Line 11, we refine the partition of the space for the next iteration: `partition` is the function that takes the current partition of the space and returns a finer partition; we suppose that all elements in the partition have the same size. Note that P_t gets finer and finer through the iterations: the more we refine, the more elements P_t will have. We highlight that in Line 5, the operator $\wedge$ applies the intersection between each subset of $\widehat{P}_{t-1}$ and P_t where P_t is a finer partition of the space than $\widehat{P}_{t-1}$. It allows to get the elements of interest ($\widehat{P}_{t-1}$) in the right level of details (P_t). For the first iteration, we run $\mathbf{F}(\cdot)$ on all the elements (initialization of $\widehat{P}_0$ to P). However, we only need to refine the sub-spaces where the monotony property is partially respected for the other iterations. Finally,

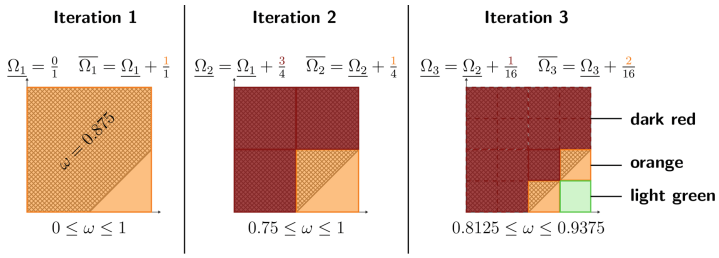

Fig. 4. Run of Algorithm 1 on Example 1 with the detailed computation of $\underline{\Omega_t}$ and $\overline{\Omega_t}$. The crosshatched area represents the sub-space the algorithm strives to estimate. (Color figure online)

the algorithm returns the lower and upper bounds of each iteration and all the sub-spaces where the monotony property does not hold or partially holds.

In Fig. 4 we run Algorithm 1 on Example 1[1]: α contains the runway's state, we partition X on α and we have a unique (p_i, p_j) in P_1; in p_i the runway is dry and in p_j wet. Then, the two axes represent features of $\bar{\alpha}$ (speed and weight) and the squares, the partition of the space. The crosshatched surface is the exact sub-space where the monotony property does not hold. The orange squares means that the monotony property partially holds, the dark red squares means it does not hold, and the light green squares means it holds. Through the iteration, we refine the partition (smaller squares) while running the verification only for the smaller squares (in solid lines) coming from a bigger orange square (in Line 5; $\widehat{P}_{t-1}$ is the orange square of iteration 2 and P_t is the small squares of iteration 3).

Algorithm 2. The verification function $\mathbf{F}(P)$ aims to analyze the monotony of f regarding P a subset of $\mathcal{P}^2$ which gathers the sub-spaces where the monotony property must be checked. Intuitively, the partition $\mathcal{P}$ and thus P can be seen as the level of details of the monotony analysis. Indeed, a finer partition $\mathcal{P}$ results in smaller sub-spaces in P; hence a more detailed analysis.

Then, from Lines 4 to 12, we identify in which sub-spaces $p_i \times p_j$ the neural network f partially respects or does not respect the monotony property and sort them in $P^{\text{partially mon}}$ and $P^{\neg\text{mon}}$. In Lines 4 and 5, `solve(·)` refers to any off-the-shelf MILP solver taking as input a MILP problem. Table 1 shows the interpretation of the monotony of f within the sub-space regarding every truth values of the conditions of Lines 4 and 5. Note that we arrive in Line 11 when the condition of Line 4 is False, and we jump to the next sub-space (or monotony scenario) because the monotony property holds for the current sub-space $p_i \times p_j$. Finally, we return the two sets gathering the sub-spaces where the monotony property does not hold and where it partially holds.

[1] Note that we simplify the crosshatched area's shape in order to know the omega value for the explanation.

Algorithm 2. $\mathbf{F}(P) \longrightarrow$ Monotony analysis of $P \subseteq \mathcal{P}^2$

Require: $P \subseteq \mathcal{P}^2$ gathers the sub-spaces that need to be verified.
1: $P^{\neg \text{mon}} \leftarrow \emptyset$
2: $P^{\text{partially mon}} \leftarrow \emptyset$
3: **for all** $(p_i, p_j) \in P$ **do**
4: **if** `solve`$(C^{\neg mon}(p_i, p_j))$ is `SAT` **then**
5: **if** `solve`$(C^{mon}(p_i, p_j))$ is `SAT` **then**
6: $P^{\text{partially mon}} \leftarrow P^{\text{partially mon}} \cup \{(p_i, p_j)\}$
7: **else**
8: $P^{\neg \text{mon}} \leftarrow P^{\neg \text{mon}} \cup \{(p_i, p_j)\}$
9: **end if**
10: **else**
11: Continue to the next (p_i, p_j) {Monotonic on the whole domain $p_i \times p_j$}
12: **end if**
13: **end for**
14: **return** $P^{\neg \text{mon}}$ and $P^{\text{partially mon}}$ $\{\{P^{\neg \text{mon}} \cup P^{\text{partially mon}}\} \subseteq P\}$

Table 1. State of the monotony property regarding the condition of Lines 4 and 5

Case	Line 4	$C^{\neg mon}$	Line 5	C^{mon}	Monotony property on $p_i \times p_j$
1	True	`SAT`	True	`SAT`	partially holds
2	True	`SAT`	False	`UNSAT`	does not hold
3	False	`UNSAT`	-	-	holds

Non-monotonic Space Coverage. $\underline{\Omega_t}$ and $\overline{\Omega_t}$ are defined as the ratio of sub-spaces (monotony scenarios) where f has monotony issue over the total number sub-spaces in P_t (contains all the monotony scenarios):

Definition 2. *Lower bound and upper bound of* ω

$$\underline{\Omega_t} = \underline{\Omega_{t-1}} + \frac{|P_t^{\neg mon}|}{|P_t|} \quad (9) \qquad \overline{\Omega_t} = \underline{\Omega_t} + \frac{\left|P_t^{partially\ mon}\right|}{|P_t|} \quad (10)$$

On the one hand, $\underline{\Omega_t}$ takes into account only the sub-spaces where the monotony property holds not; hence, it lower-bounds ω. On the other hand, $\overline{\Omega_t}$ considers the sub-spaces where the monotony property holds not and partially holds; hence, it upper-bounds ω. Figure 4 details the computation of these metrics along with the iteration: at each iteration, the lower bound $\underline{\Omega_t}$ is represented by all the dark red squares and the upper bound $\overline{\Omega_t}$ by all the dark red and orange squares.

Example 2. Computation of $\underline{\Omega_t}$ *and* $\overline{\Omega_t}$ *considering Example 1.*

Iteration 1. We consider the entire space. Hence, we only have one sub-space where we assess the monotony property ($|P_t| = 1$). There is no dark red square, *i.e.* sub-space where the monotony property does not hold, which means that $|P_1^{\neg \text{mon}}| = 0$, then $\underline{\Omega_1} = 0$. We have one orange square: in this sub-space, the monotony property partially holds, then $|P_1^{\text{partially mon}}| = 1$ and $\overline{\Omega_1} = 1$.

Iteration 2. We partition the space in 4 smaller sub-spaces ($|P_t| = 4$) and run again the verification on each sub-space. We proceed similarly as previously for the computation of $\underline{\Omega_2}$ and $\overline{\Omega_2}$. We have 3 dark red squares ($|P_2^{\neg\text{mon}}| = 3$) and 1 orange square ($|P_2^{\text{partially mon}}| = 1$): $\underline{\Omega_2} = \underline{\Omega_1} + \frac{3}{4} = 0.75$ and $\overline{\Omega_2} = \underline{\Omega_2} + \frac{1}{4} = 1$.

Iteration 3. We refine the partition of the previous step, and we end up with 16 sub-spaces. However, we only run the verification on the sub-spaces coming from an orange square (Lines 5 of Algorithm 1), *i.e.* , a sub-spaces where f is partially monotonic. We have $\underline{\Omega_3} = \frac{3}{4} + \frac{1}{16} = 0.8125$ and $\overline{\Omega_3} = \frac{3}{4} + \frac{1}{16} + \frac{2}{16} = 0.9375$.

The Proposition 1 shows that the lower and upper bounds are tighter over the iterations: the more iterations we run, the closer to ω we are.

Proposition 1. *For any $t \geq 1$, we have*

$$\underline{\Omega_{t-1}} \leq \underline{\Omega_t} \qquad (11) \quad \textit{and} \quad \overline{\Omega_t} \leq \overline{\Omega_{t-1}} \qquad (12)$$

Proof. For Eq. 11, from the facts that

$$\underline{\Omega_0} = 0 \quad \text{and} \quad \frac{|P_t^{\neg\text{mon}}|}{|P_t|} \geq 0 \quad \text{and} \quad \underline{\Omega_t} = \underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|},$$

we can deduce $\underline{\Omega_{t-1}} \leq \underline{\Omega_t}$.

Then, to prove Eq. 12, we need first to state some invariant: for the computation of $\overline{\Omega_t}$ (Algorithm 1, Line 7) we have,

$$\left(P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right) \subseteq \widehat{P}_t \qquad (13)$$

Based on Eq. (13), we have:

$$\begin{aligned}
\left|P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right| &\leq \left|\widehat{P}_t\right| \\
&\leq \left|P_{t-1}^{\text{partially mon}}\right| * \frac{|P_t|}{|P_{t-1}|} && \text{By construction of } \widehat{P}_t \text{ which is a finer partition of } P_{t-1}^{\text{partially mon}} \\
\underline{\Omega_{t-1}} + \frac{\left|P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right|}{|P_t|} &\leq \underline{\Omega_{t-1}} + \frac{\left|P_{t-1}^{\text{partially mon}}\right|}{|P_{t-1}|} && \text{We divide both side of the inequality by } \left|\widehat{P}_t\right| \text{ and add } \underline{\Omega_{t-1}} \\
\underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} &\leq \overline{\Omega_{t-1}} && P_t^{\text{partially mon}} \cap P_t^{\neg\text{mon}} = \emptyset \\
\underline{\Omega_t} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} &\leq \overline{\Omega_{t-1}} \\
\overline{\Omega_t} &\leq \overline{\Omega_{t-1}}
\end{aligned}$$

5 Case Study: Braking Distance Estimation

5.1 Description of the Case Study

Our case study comes from the aeronautical industry. It is an R&D project consisting in training a neural network to estimate the braking distance of an

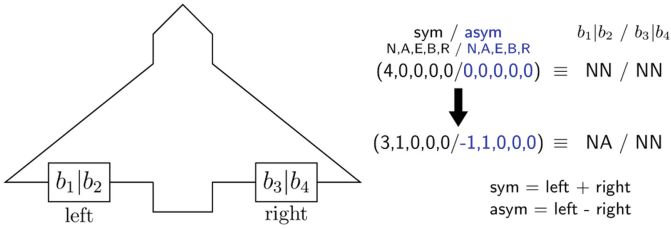

Fig. 5. Representation of the position of the four brakes on an aircraft denoted by $b_i \in \{N, A, E, B, R\}$. For example, we have NN-NN when all the brakes are in the normal state. Then if the state of one of the left brakes becomes *Altered*, we have NA-NN. Note that NA-NN≡AN-NN due to the choice of the representation of the brakes.

aircraft based on physical information. The R&D team provides us with a trained feedforward neural network composed of 2 layers (30 and 29 neurons on the first and second layers, respectively) and ReLU activation functions. There are 15 input features, including 13 discrete and 2 continuous. Among the discrete features, ten describe the state of the brakes. The aircraft has 4 brakes, and each brake has 5 possible modes: Normal (N), Altered (A), Emergency (E), Burst (B), and Released (R). Then, the network has 2 features for each mode: *(i)* the total number of brakes in a given mode (referred to as "symmetric") and *(ii)* the difference between the number of brakes on the left and right side of a given mode (referred to as "asymmetric"). From this information, we can find back the states of the pairs of brakes on the left and right sides of the aircraft (see Fig. 5), although the state of each individual brake cannot be retrieved. For clarity and since we have the equivalence between both notations, we will describe the state of the brakes using the form "b_1b_2-b_3b_4".

To show how to handle simultaneously several input features within the monotony property, we focus on the one involving the state of the brakes. We can textually express the monotony property as follows:

When the brakes' state deteriorates, the braking distance should increase.

To perform the monotony analysis, we need to define what *deteriorates* means formally. Relying on the system expert's knowledge, the following partial order applies to the different modes of the brake:

$$N \prec_b A \prec_b E \prec_b B \prec_b R \tag{14}$$

where $b_i \prec_b b_j$ means that the state b_j is more deteriorated than the state b_i. We can easily extend the partial order $\preceq_b$ on one brake to the state of an aircraft's brakes composed of 4 brakes. Let $S_1 = (b_1, b_2, b_3, b_4)$ and $S_2 = (b'_1, b'_2, b'_3, b'_4)$ be two states of an aircraft's brakes, we have

$$S_1 \prec S_2 \iff \forall b_i, b'_i \in S_1 \times S_2,\ b_i \preceq_b b'_i \text{ and } \exists b_i, b'_i \in S_1 \times S_2\ b_i \prec_b b'_i \tag{15}$$

It means that S_2 is deteriorated compared to S_1 if and only if for all brakes in S_2, the brake's mode in S_2 is at most as good as its counterpart in S_1 and there exists a brake in S_2 whose mode is strictly worse than its counterpart in S_1.

5.2 Experimentation

Setup. Let $\mathcal{V}$ be the set of 15 input features described above, $X = \times_{v \in \mathcal{V}} \mathcal{D}(v)$ be the input space, $Y = \mathbb{R}^+$ be the output space, and $f : X \mapsto Y$ be the neural network estimating the braking distance of an aircraft. We consider the monotony property as formulated in Definition 1. As stated earlier, we are dealing with the monotony with respect to the brakes' space. Hence, $\alpha \subseteq \mathcal{V}$ is made up of the ten features describing the state of the brakes and the partial order $\preceq$ on $\times_{v \in \alpha} \mathcal{D}(v)$ is as defined in Eq. (15). We take advantage of the discrete nature of the brakes' features: a natural partition $\mathcal{P}$ is to enumerate all the possible values for the ten brakes features. We have $|\mathcal{P}| = 225$.

Monotony Analysis. We run Algorithm 1 for 5 iterations with the setup described above. The algorithm is explained in Sect. 4.3. Here we only focus on the partitions used for the analysis and the refinement strategy which are specific to the case study. Additionally, we will see how to capitalize on the data available at each iteration to perform some space exploration. P_1 is setup using $\mathcal{P}$, and $\preceq$; it represents the brakes' sub-space. Then our partition's refining strategy is to start with the remaining discrete features (second iteration) and then consider the continuous features (the last three iterations). For the discrete features, the partitioning consists in enumerating the possible values, while for the continuous features, it consists of a uniform partition (finer through the iterations). To illustrate the impact of the refinement on the level of details of the analysis, we detail the total number of sub-spaces in each partition P_t: $|P_1| = 10800$, $|P_2| = 259200$, $|P_3| = 6480000$, $|P_4| = 25920000$ and $|P_5| = 103680000$.

Based on the partition and the outcomes of $\mathbf{F}(\cdot)$, the algorithm yields at each iteration the metrics $\underline{\Omega_t}$ and $\overline{\Omega_t}$. Nonetheless, for our case study, we put in place visualization means (see Fig. 6). However, the relevant visualizations helping space exploration are case-dependent, so we do not propose any generic way to do it. Firstly, it might be relevant to visualize the sub-space composed of the features on which the partial order $\preceq$ is defined, *i.e.* α corresponding to the brakes' space. It is modeled as a graph where the nodes are the brakes' states (the elements of $\mathcal{P}$), and the edges are the transitions between the states modeled by the partial order $\prec$ (the elements of P_1) and with the outcomes of the first iteration, we can highlight (dashed line in Fig. 6) the transitions which violate the monotony property (in Fig. 6, we plot only a sub-graph as an example). Then, to include the information of the formal verification of the following iteration in the space visualization, we plot some features versus the difference of distances $f(x_1) - f(x_2)$ and visualize in which sub-spaces monotony issue occurs. These visualizations are helpful for exploration purposes after the analysis for the expert of the system (e.g., if the expert can identify some place of interest within the space and wants to know what happens there).

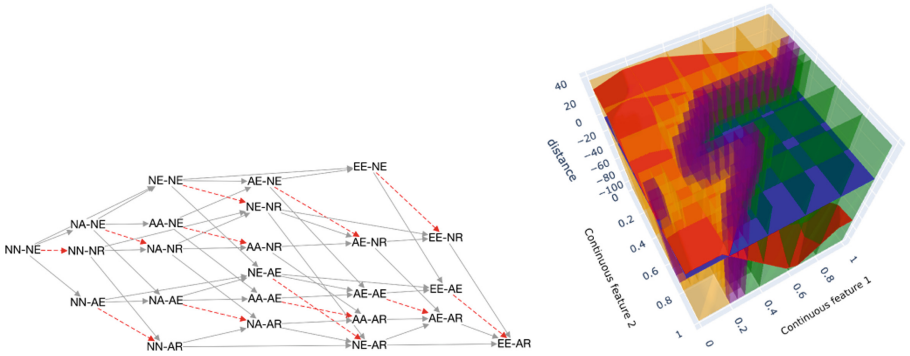

Fig. 6. Example of visualization of features in α (left) and $\bar{\alpha}$ (right).

Table 2. Values of $\underline{\Omega}$ and $\overline{\Omega}$ bounding the percentage of the space where the monotony property is violated.

Metrics	It.1	It.2	It.3	It.4	It.5
$\overline{\Omega_t}$	11.57%	4.11%	1.95%	1.72%	1.61%
$\underline{\Omega_t}$	0.03%	0.45%	1.12%	1.29%	1.39%

Metrics: Non-monotonic Space Coverage. At each iteration, we compute $\underline{\Omega_t}$ and $\overline{\Omega_t}$, which bound the ratio of the space where f violates the monotony property (*i.e.* ω). The results are summarised in Table 2. In Fig. 7, we can clearly see the convergence of $\underline{\Omega_t}$ and $\overline{\Omega_t}$. At the first iteration, we can explain the notable gap between $\underline{\Omega_1}$ and $\overline{\Omega_1}$ by the coarse partitioning of the space. That is why, $\overline{\Omega_1}$ is large (numerous sub-spaces where the monotony property is partially respected) and $\underline{\Omega_1}$ small (only few sub-spaces where the monotony property does not hold). We can notice a significant drop of $\overline{\Omega_t}$ compared to the rise of $\underline{\Omega_t}$: there are more sub-spaces where the monotony property holds than not. Eventually the algorithm yields a narrow gap between the bounds; we obtain at the fifth iteration: $1.39\% \leq \omega \leq 1.61\%$. The stopping criterion of the algorithm may depends on various things such that the system's requirements (e.g. bounds precision, max value to not cross for $\underline{\Omega}$ or min value to reach for $\overline{\Omega}$).

Through these five steps, we analyze the monotony of f considering finer and finer partition of the space; we obtain: *(i)* metrics bounding the percentage of the space where the neural network is non-monotonic and *(ii)* the identification of the sub-spaces where the monotony issue occurs thanks to the formal verification on each elements of the partitions.

We run our experiments on MacBook Pro 8 core 2,3 GHz Intel Core i9 with 32 Gb of RAM. The MILP solver used is Gurobi 9 [10] and our monotony analysis took less than 10 h.

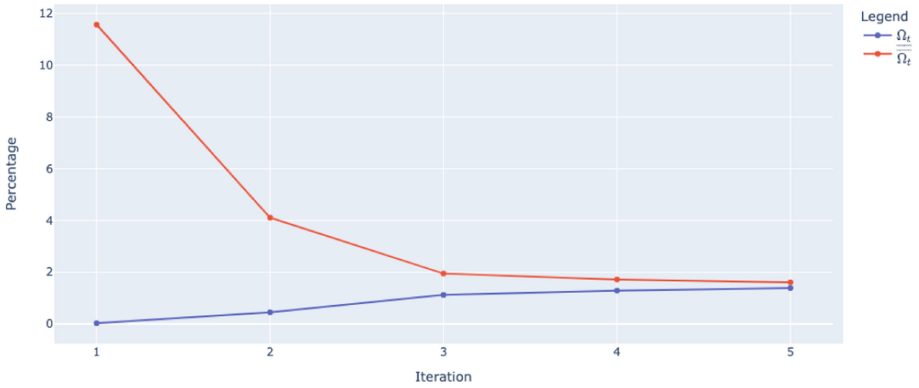

Fig. 7. Evolution of $\underline{\Omega_t}$ and $\overline{\Omega_t}$

6 Conclusion

This work develops an iterative method to assess the monotony of a neural network using a MILP solver. The monotony property defined is suited for discrete features. This iterative method allows for lower and upper bounding the space where the neural network does not hold the property and formally identifies these areas. This is a step further in the demonstration that neural networks can preserve important function properties and therefore in the capability to embed the ML technology in an aeronautical safety-critical system.

We applied this method on an aeronautical case study that consists in estimating the braking distance of an aircraft using a neural network mixing discrete and continuous inputs. We managed to quantify the percentage of the space where the neural network does not preserve the monotony property and to identify formally each sub-space where it occurs. In addition, we showed that we can capitalize on the available data to visualize the sub-spaces for helping the braking function's experts in processing the results of the algorithm.

Note that this work leaves room for some optimizations, such as using tighter big-M values in Eq. 2–3, or using asymmetric bounds, computed by incomplete methods such as [23,31]. To the best of our knowledge, the scalability of complete method remains a challenge in the verification community and is mainly used with "shallow" neural networks. Thus, this method is mainly useful for small to medium networks used as surrogates or for control.

As an extension of this work, we plan to estimate the integral under the curve of $f(x_1) - f(x_2)$ in the sub-spaces where the monotony is violated by leveraging our definition of the monotony property. This would give a key indicator on the level of violation of the monotony property that could support the performance of the training phase. Another perspective would be to extend this work to continuous features by using the formulation of the monotony based on the gradient.

References

1. Biannic, J., Hardier, G., Roos, C., Seren, C., Verdier, L.: Surrogate models for aircraft flight control: some off-line and embedded applications. Aerospace Lab (12), 1 (2016)
2. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: IEEE SP, pp. 39–57. IEEE Computer Society (2017)
3. Cheng, C.-H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 251–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_18
4. Damour, M., et al.: Towards certification of a reduced footprint ACAS-Xu system: a hybrid ML-based solution. In: Habli, I., Sujan, M., Bitsch, F. (eds.) SAFECOMP 2021. LNCS, vol. 12852, pp. 34–48. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-83903-1_3
5. EASA: Concept paper first usable guidance for level 1 machine learning applications (2021). https://www.easa.europa.eu/downloads/134357/en
6. Feelders, A.J.: Prior knowledge in economic applications of data mining. In: Zighed, D.A., Komorowski, J., Żytkow, J. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 395–400. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45372-5_42
7. Gauffriau, A., Malgouyres, F., Ducoffe, M.: Overestimation learning with guarantees. arXiv preprint arXiv:2101.11717 (2021)
8. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. Optim. Eng. **3**(3), 227–252 (2002)
9. Gupta, A., Shukla, N., Marla, L., Kolbeinsson, A., Yellepeddi, K.: How to incorporate monotonicity in deep networks while preserving flexibility? arXiv preprint arXiv:1909.10662 (2019)
10. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022). https://www.gurobi.com
11. Hao, J., Ye, W., Jia, L., Wang, G., Allen, J.: Building surrogate models for engineering problems by integrating limited simulation data and monotonic engineering knowledge. Adv. Eng. Inform. **49**, 101342 (2021)
12. Jian, Z.D., Chang, H.J., Hsu, T.S., Wang, D.W.: Learning from simulated world - surrogates construction with deep neural network. In: SIMULTECH 2017: Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. SCITEPRESS (2017)
13. Karpf, J.: Inductive modelling in law: example based expert systems in administrative law. In: Proceedings of the 3rd International Conference on Artificial Intelligence and Law, pp. 297–306 (1991)
14. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
15. Liu, X., Han, X., Zhang, N., Liu, Q.: Certified monotonic neural networks. Adv. Neural. Inf. Process. Syst. **33**, 15427–15438 (2020)
16. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: ICLR. OpenReview.net (2018)
17. Mamalet, F., et al.: White paper machine learning in certified systems. IRT Saint Exupéry - ANITI (2021)
18. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: general and precise neural network certification via scalable convex hull approximations. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022)

19. Nguyen, A.P., Martínez, M.R.: Mononet: towards interpretable models by learning monotonic features. arXiv preprint arXiv:1909.13611 (2019)
20. Peterson, E., DeVore, M., Cooper, J., Carr, G.: Run time assurance as an alternate concept to contemporary development assurance processes. NASA/CR-2020-220586 (2020)
21. Raghunathan, A., Steinhardt, J., Liang, P.S.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Advances in Neural Information Processing Systems, pp. 10877–10887 (2018)
22. Sudakov, O., Koroteev, D., Belozerov, B., Burnaev, E.: Artificial neural network surrogate modeling of oil reservoir: a case study. In: Lu, H., Tang, H., Wang, Z. (eds.) ISNN 2019. LNCS, vol. 11555, pp. 232–241. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22808-8_24
23. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019)
24. Tsuzuku, Y., Sato, I., Sugiyama, M.: Lipschitz-margin training: scalable certification of perturbation invariance for deep neural networks. In: NeurIPS, pp. 6542–6551 (2018)
25. Urban, C., Christakis, M., Wüstholz, V., Zhang, F.: Perfectly parallel fairness certification of neural networks. Proc. ACM Program. Lang. **4**(OOPSLA), 1–30 (2020)
26. Urban, C., Miné, A.: A review of formal methods applied to machine learning. arXiv preprint arXiv:2104.02466 (2021)
27. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium (USENIX Security 2018), Baltimore, MD, pp. 1599–1614. USENIX Association, August 2018
28. Wang, S., et al.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems (2021)
29. Weng, T.W., et al.: Towards fast computation of certified robustness for ReLU networks. arXiv preprint arXiv:1804.09699 (2018)
30. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5777–5783 (2018)
31. Xu, K., et al.: Automatic perturbation analysis for scalable certified robustness and beyond. In: NeurIPS (2020)
32. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems, pp. 4939–4948 (2018)
33. Zhang, H., Zhang, P., Hsieh, C.J.: Recurjac: an efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 5757–5764 (2019)