Verifying Global Two-Safety Properties in Neural Networks with Confidence

Anagha Athavale[1,2(✉)], Ezio Bartocci[1], Maria Christakis[1], Matteo Maffei[1], Dejan Nickovic[2], and Georg Weissenbacher[1]

[1] TU Wien, Vienna, Austria
`anagha.athavale@tuwien.ac.at`
[2] AIT Austrian Institute of Technology, Vienna, Austria

Abstract. We present the first automated verification technique for confidence-based 2-safety properties, such as global robustness and global fairness, in deep neural networks (DNNs). Our approach combines self-composition to leverage existing reachability analysis techniques and a novel abstraction of the softmax function, which is amenable to automated verification. We characterize and prove the soundness of our static analysis technique. Furthermore, we implement it on top of Marabou, a safety analysis tool for neural networks, conducting a performance evaluation on several publicly available benchmarks for DNN verification.

1 Introduction

Deep neural networks (DNNs) [1,2] encountered tremendous success in the recent past due to their ability to infer highly nonlinear relations from data, learn accurate predictive models, and make smart decisions with little or no human intervention. Despite their success, the correctness of neural networks remains a major concern due to their complexity and lack of transparency. This is especially the case for safety- and security-critical applications where errors and biases can have serious undesired consequences, such as in medical diagnosis [3], self-driving cars [4], or financial systems [5]. New techniques that can prove mathematical guarantees concerning the behavior of neural networks are the need of the hour [6]. An effective approach to address this issue is the use of automatic verification techniques, which can either formally prove that the network adheres to a specified property or return a concrete input (witness) demonstrating a violation of the property [7].

Robustness and *fairness* are two important properties of neural networks. Robustness refers to the neural network's ability to make accurate predictions even in the presence of input perturbations. In particular, a robust neural network is able to produce accurate results without being overly sensitive to small changes in the input. Fairness, on the other hand, refers to a neural network's ability to make unbiased and equitable predictions, particularly in cases where the input data may contain sensitive attributes such as gender, race, or age. A neural network that is not fair may produce biased results that discriminate against certain groups, which can have serious ethical and social implications.

Local Robustness and Fairness provide the dominant perspective in verification and adversarial testing of DNNs. Local robustness [8–11] intuitively requires the DNN to have the following property with respect to a given input x – it has to make the same prediction for the input x as for all the points in the vicinity of x. Local fairness [8,12,13] is defined in a similar way, with the distance metric used for the inputs being the main difference. Both properties can be formalized as safety properties. This has led to the design of a variety of SMT-based techniques [10,14,15], which encode the neural networks and the property to be verified as an SMT solving problem in order to enable automated verification. Other works approach the verification problem using static analysis [16–19] which over-approximates DNN executions, thereby compromising precision for higher scalability. Alternative verification techniques include mixed-integer programming [20–22] and modified simplex algorithms [9,23].

Figure 1 (a) illustrates the properties of local robustness. It shows the classification of an input $\vec{x}$ that includes two (continuous) features x_1 and x_2. The pair of purple points is not a counterexample to robustness, as both inputs lie within the same class. The green and blue points, however, represent counterexamples to local robustness, as they fall on different sides of the decision boundaries.

The above example shows two major limitations of local properties. First, there are always inputs arbitrarily close to the decision boundary, which then constitute counterexamples to local robustness. Second, local robustness is defined only for a specific input. Consequently, it does not provide any guarantees for any other input. It follows that the robustness of the entire neural network cannot be assessed with local robustness only.

Global Robustness and Fairness. The limitation of the local definition for robustness and fairness indicates the need for a global property that evaluates the expected input/output relation over *all pairs of inputs.*

We first observe that global robustness and fairness of DNNs are *hyperproperties*, i.e. properties that relate multiple executions of the model. Khedr et al. [24] and Biswas et al. [25] recently introduced the first verification techniques for hyperproperties in DNNs. These works assume that the inputs contain categorical variables. Based on this strong assumption, these two approaches partition the input space based on categorical features to avoid comparing inputs close to decision boundaries, which would lead to a non-satisfiable property. This is illustrated in Fig. 1 (b). Here, we assume that $\vec{x}$ includes a categorical feature x_3 in addition to the continuous features x_1 and x_2. The left (right) part of Fig. 1 (b) depicts classes and inputs in the partition based on the categorical feature x_3 with value v (v'). Consequently, only pairs of inputs belonging to the *same* partition are compared. Inputs belonging to two different partitions (e.g. green points in Fig. 1 (b)) deviate in at least one categorical feature and can hence be assumed to violate the premise that these inputs are "close". According to this approach, a classification in a secure network can only change with different categorical values. Any two points that lie in the same partition but belong to different classes (e.g. the pair of blue points in Fig. 1 (b)) are considered counterexamples to the global property. This leads to a strong limitation
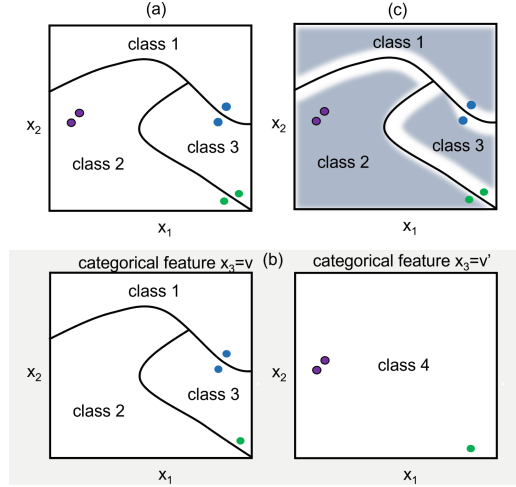
Fig. 1. (a) Local, (b) partitioned-global and (c) our confidence-based robustness. x_1 and x_2 denote continuous input points, while x_3 denotes a categorical input in the partitioned-global approach (b). The shades of gray in (c) depict the level of confidence of the neural network with respect to the given inputs – dark gray denotes high while white denotes low confidence level. The neural network is robust to the pair of purple points in all three cases (a), (b) and (c). The neural network is not robust for the pair of blue points in the case of local and partitioned-global (b) robustness, but is robust according to our definition (c). Finally, the neural network is not robust for the pair of green points according to both the local and our confidence-based global robustness (a) and (c), but is robust with respect to the partitioned-global robustness (b). The global partitioning method does not catch the counterexample, because the two green points are in separate partitions. (Color figure online)

that does not admit two classes to result from continuous inputs only, as typically required for robustness. As a result, the two approaches [24,25] address only verification of global fairness.

Our Contributions. Inspired by the work of Chen et al. [26] on properties of rule-based security classifiers, we adopt a *confidence-based* view on global robustness and fairness for DNN. The idea is to compare all input pairs which are (1) sufficiently close and (2) for which at least one of them yields a high confidence classification. This intuitive definition expects robust and fair DNNs to generate outputs with low confidence near the decision boundary.

We therefore propose *confidence-based 2-safety property*, the first definition that unifies global robustness and fairness for DNNs. Our definition highlights the hyperproperty nature of global properties and uses the confidence in the DNN as a first class citizen.

We briefly illustrate the intuition behind our confidence-based 2-safety property definition with focus on robustness in Fig. 1 (c), in which the input space is colored into shades of gray and where every gray value corresponds to a confidence of the network. Darker shades of gray represent higher levels of confidence for the given classification. Our definition captures two reasonable assumptions: (1) continuous inputs can also trigger changes in classification, and (2) the confidence of the neural network at a decision boundary must be relatively low. In essence, our definition requires that for any input with high-confidence, all its ϵ-neighbour inputs yield the same class (e.g. the two purple points in Fig. 1 (c)). This notion discards inputs near the decision boundaries as counterexamples, as long as they result in outputs with low confidence (e.g. the two blue points in Fig. 1 (c)). Systems satisfying the 2-safety properties hence guarantee that input points classified with a high confidence are immune to adversarial perturbation attacks. In Fig. 1 (c), the pair of green inputs witness the violation of the confidence-based 2-safety property – the two points lie in different classes and one of them has an output with high confidence.

This confidence-based view makes a conceptual change to the definition of global properties, as it requires relating not only inputs, but also confidence values to the outputs. This conceptual change poses a significant challenge to the verification problem because checking a confidence-based property on a DNN requires reasoning about its *softmax* layer, which is not supported by the state-of-the-art DNN verification tools [23,27–31]. To solve this problem, we develop the first verification method that supports DNNs with softmax, in which we use a linearized over-approximation of the softmax function. We then combine it with self-composition [32] in order to verify confidence-based 2-safety properties. We formally prove the soundness of our analysis technique, characterizing, in particular, the error bounds of our softmax over-approximation.

We demonstrate our approach[1]in Marabou [23], a state-of-the-art analysis tool for local robustness based on a modified simplex algorithm, which we extend to support global robustness and global fairness. We show that by combining our method with binary search, we can go beyond verification and synthesize the minimum confidence for which the DNN is globally robust or fair. We finally conduct a performance evaluation on four neural networks trained with publicly available datasets to demonstrate the effectiveness of our approach in identifying counterexamples and proving global robustness and fairness properties.

2   Background

2.1   Feed-Forward Neural Networks

In feed-forward neural networks data flows uni-directionally, which means there are no back edges. An input layer receives the inputs that move via one or multiple hidden layers to the output layer [33]. A layer has multiple neurons, each connected to the neurons in the next layer using a set of weights. Each layer also

---

[1] https://github.com/anaghaathavale/Global_2Safety_with_Confidence.git.

has an associated bias. Weight and bias selection is crucial to the performance of a neural network and is performed during the training phase. Outputs are calculated by processing the inputs using weights and biases, followed by applying the activation functions and then propagating the processed inputs through the network [34].

Formally, a feed-forward neural network $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is modeled as a directed acyclic graph $G = (V, E)$ that consists of a set (finite) of nodes V and a set of edges $E \subseteq V \times V$.[2] The nodes V are partitioned into l layers V^i with $1 \leq i \leq l$, where V^1 and V^l represent the input and output layers, and $V^2, \ldots, V^{l-1}$ represent the hidden layers, respectively. We use $v_{i,j}$ to denote node j in layer i. The edges E connect nodes in V^{i-1} with their successor nodes in V^i (for $1 < i \leq l$).

Each node $v_{i,j}$ has an input and an output, where the latter is derived from the former by means of an activation function. We use $\text{in}(v_{i,j})$ and $\text{out}(v_{i,j})$ to denote the input and output value of node $v_{i,j}$, respectively. The output is determined by

$$\text{out}(v_{i,j}) = a_{i,j}(\text{in}(v_{i,j})), \tag{1}$$

where $a_{i,j}$ is the activation function. The input to node $v_{i,j}$ in layer V^i is determined by the outputs of its predecessors $v_{i-1,1}, \ldots, v_{i-1,k}$ in V^{i-1} and weights associated with the edges $(v_{i-1,k}, v_{i,j}) \in E$ for $1 \leq k \leq |V^{i-1}|$:

$$\text{in}(v_{i,j}) = \sum_{k=1}^{|V^{i-1}|} \text{weight}((v_{i-1,k}, v_{i,j})) \cdot \text{out}(v_{i-1,k})$$

The values of the nodes in the input layer V^1 are determined by the input $\vec{x}$ to $f(\vec{x})$, i.e.,

$$(\text{in}(v_{1,1}), \ldots, \text{in}(v_{1,m})) = \vec{x}.$$

The output of the final layer V^l is then computed by propagating the inputs according to the activation functions (see Eq. 1 above). Consequently, a graph G with $|V^1| = m$ input and $|V^l| = n$ output nodes induces a function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ whose semantics is determined by the activation functions.

In this paper, we concentrate on the Rectified Linear Unit (ReLU) activation function, which is frequently applied to the hidden layers of deep neural networks. For a (scalar) input value x, ReLU returns the maximum of 0 and x, i.e.

$$\text{ReLU}(x) = \max(0, x).$$

In neural networks that are used as classifiers and map an input $\vec{x}$ to one of m labels in a set of classes C, the final layer typically employs a softmax function to ensure that the output represents normalized probabilities corresponding to each of the n classes. Mathematically,

$$\text{softmax}(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2}$$

---

[2] We follow the formalization of [35] and omit the biases for simplicity.

where $\vec{z}$ represents the values $\text{out}(v_{l-1,i})$ for $1 \leq i \leq n$ and $n = |V^{l-1}|$, and z_i is the ith element in $\vec{z}$. This induces a function $y : \mathbb{R}^n \rightarrow [0,1]^n$ mapping every output of V^{l-1} to a confidence score in the range $[0,1]$. Consequently, $f(\vec{x})$ outputs a probability distribution over the possible labels in C, where each component of the output vector represents the probability of input $\vec{x}$ belonging to the corresponding class. We use $\text{conf}(f(\vec{x}))$ to refer to the highest probability value in the softmax layer of $f(\vec{x})$ and call it the *confidence*, i.e.,

$$\text{conf}(f(\vec{x})) = \max(\text{out}(v_{l,1}), \ldots, \text{out}(v_{l,n})) \tag{3}$$

Finally, a function class : $\mathbb{R}^m \rightarrow C$ then maps the output of f to the class C corresponding to the highest probability in $f(\vec{x})$:

$$\text{class}(f(\vec{x})) = \arg\max_{1 \leq i \leq n}(\text{out}(v_{l,i})) \tag{4}$$

2.2   Hyperproperties

Hyperproperties [36] are a class of properties that capture relationships between multiple execution traces. This is in contrast to traditional properties, which are evaluated over individual traces.

To define traces in the context of feed-forward neural networks, we extend our notation out to layers as follows:

$$\text{out}(V^i) = (\text{out}(v_{i,1}), \ldots, \text{out}(v_{i,k}))$$

where $k = |V^i|$. Let $\text{in}(V^i)$ be defined similarly. The corresponding trace π for $f(\vec{x})$ is then formally defined as

$$\pi = \text{in}(V^1), \text{out}(V^1), \ldots, \text{in}(V^l), \text{out}(V^l)$$

where $\text{in}(V^1) = \vec{x}$.

Note that each execution is entirely determined by the input value $\vec{x}$ (assuming that the function f implemented by the network is deterministic). Quantifying over traces π of $f(\vec{x})$ hence corresponds to quantifying over the corresponding inputs $\vec{x}$. A traditional safety property would then quantify over the inputs $\vec{x}$, e.g.

$$\forall \vec{x} \, . \, \text{conf}(f(\vec{x})) \geq \kappa,$$

stating that the confidence of each classification of the network should be larger than a threshold κ. Another example of a traditional safety property is local robustness, given in Definition 1 in Subsect. 2.4.

A hyperproperty, on the other hand, refers to, and quantifies over, more than one trace. An example would be

$$\forall \vec{x}, \vec{x}' \, . \, \frac{|f(\vec{x})_i - f(\vec{x}')_i|}{||\vec{x} - \vec{x}'||} \leq K_i, 1 \leq i \leq n \tag{5}$$

where $f(\vec{x})_i$ denotes out$(v_{l,i})$. Equation 5 states that K_i sets the maximum limit of the Lipschitz constant for $f(\vec{x})_i$. A hyperproperty central to this paper is global robustness, defined in Definition 2 Subsect. 2.4.

Hyperproperties are used to capture important properties that involve multiple inputs, such as robustness and fairness. By verifying hyperproperties of neural networks, we can ensure that they behave correctly across all possible input traces.

2.3   Relational Verification and Self-composition

Hyperproperties are verified by means of so-called relational verification techniques: the idea is to verify if k program executions satisfy a given property [37], expressing invariants on inputs and outputs of such executions. Several security properties (e.g., information flow) can be expressed by relating two executions of the same program differing in the inputs: such properties are called 2-safety properties. Global robustness in neural networks can also be seen as a 2-safety property [8].

2-safety properties can be verified in a generic way by self-composition [37]: the idea is to compose the program with itself and to relate the two executions. In the context of neural networks, the self-composition of a network f is readily defined as a function over

$$f(\vec{x}) \times f(\vec{x}') = \lambda(\vec{x}, \vec{x}') \, . \, (f(\vec{x}), f(\vec{x}')) \tag{6}$$

where $(\vec{x}, \vec{x}')$ denotes the concatenation of the vectors $\vec{x}$ and $\vec{x'}$ and $\lambda \vec{x} \, . \, f(\vec{x})$ denotes the lambda term that binds $\vec{x}$ in $f(\vec{x})$. The underlying graph $G = (V, E)$ is simply duplicated, i.e., we obtain a graph $G \times G' = (V \cup V', E \cup E')$ where V' and E' are primed copies of V and E.

A counterexample to a universal 2-safety property over the self-composition of f comprises of a pair of traces of f witnessing the property violation.

2.4   Robustness and Fairness

Robustness in neural networks refers to the ability of a model to perform consistently in the presence of small perturbations of the input data. The common approach to address robustness in neural networks is to define it as a *local robustness* [38] property. For an input $\vec{x}$, a neural network is locally robust if it yields the same classification for $\vec{x}$ and all inputs $\vec{x}'$ within distance ϵ from $\vec{x}$ [39]:

Definition 1 (Local Robustness). *A model f is locally ϵ-robust at point $\vec{x}$ if*

$$\forall \vec{x}' \, . \, ||\vec{x} - \vec{x}'|| \leq \epsilon \rightarrow \text{class}(f(\vec{x})) = \text{class}(f(\vec{x}'))$$

Local robustness, therefore, is defined only for inputs within a distance ϵ of a specific $\vec{x}$ and, thus, does not provide global guarantees. Here $||\cdot||$ represents the distance metric used over the input space. Intuitively, global robustness tackles this problem by requiring that the local robustness property must hold for *every*

input within the input space [8]. Definition 2 gives the general definition of global robustness used in [26, 40]. It essentially states that all input points in a small neighborhood ϵ, are mapped to the same class.

Definition 2 (Global robustness). *A model f is globally ϵ-robust if*

$$\forall \vec{x}, \vec{x}' . \, ||\vec{x} - \vec{x}'|| \leq \epsilon \rightarrow \text{class}(f(\vec{x})) = \text{class}(f(\vec{x}'))$$

Clearly, global robustness as formalized in Definition 2 makes sense only for selected distance metrics, which in particular avoid comparing inputs close to the decision borders. For instance, [40] addresses this by introducing an additional class $\bot$ to which class$(f(\vec{x}))$ evaluates whenever the difference between the highest and second-highest probability falls below a certain threshold (determined by the Lipschitz constants of f). The global robustness requirement is then relaxed at these points.

Definition 3 (Global fairness) *A model is said to be globally fair if:*

$$\forall \vec{x} = (x_s, \vec{x_n}), \vec{x'} = (x'_s, \vec{x'_n}).$$
$$||\vec{x_n} - \vec{x'_n}|| \leq \epsilon \, \wedge (x_s \neq x'_s) \rightarrow \text{class}(f(\vec{x})) = \text{class}(f(\vec{x'}))$$

where x_s and x_n are the sensitive and non-sensitive attributes of $\vec{x}$, respectively.

[24, 25] address a similar problem, which arises in the context of fairness, by partitioning the input space based on categorical features. In general, if the input to a decision-making neural network comprises of certain sensitive attributes, say age or gender, the network is said to be fair if the sensitive attributes do not influence its decisions [8]. Definition 3 gives the general definition of global fairness used in [24, 25].

Ensuring fairness in neural networks is important because these models are increasingly being used in decision-making processes that can have significant impacts on peoples' lives. For example, a hiring algorithm that discriminates against certain groups of job applicants based on their race or gender could perpetuate existing biases and inequalities in the workplace [41].

3   Confidence Based Global Verification of Feed-Forward Neural Networks

We now formalize confidence-based 2-safety property, the first definition that unifies global robustness and fairness for DNNs in Definition 4. It is a hyperproperty that takes the confidence of the decision into account when checking for the safety of the network. Before we give the actual definition, we introduce additional notation. Given an input $\vec{x} = (x_1, \ldots, x_n)$, we assume that its every component x_i is either a *categorical* or *real* value. We then define the distance $d(x_i, x'_i)$ as $|x_i - x'_i|$ when x_i is real-valued. We use instead the following distance:

$$d(x_i, x'_i) = \begin{cases} 0, & \text{if } x_i = x'_i \\ 1, & \text{otherwise} \end{cases}$$

when x_i is a categorical value. We define $cond(\vec{x}, \vec{x}', \vec{\epsilon})$ as a (generic) Boolean *condition* that relates inputs $\vec{x}$ and $\vec{x}'$ to a tolerance vector $\vec{\epsilon}$.

Definition 4 (Confidence-based global 2-safety). *A model f is said to be globally 2-safe for confidence $\kappa > 0$ and tolerance $\vec{\epsilon}$ iff*

$$\forall \vec{x}, \vec{x}' \, . \, \mathrm{cond}(\vec{x}, \vec{x}', \vec{\epsilon}) \, \wedge \, \mathrm{conf}(f(\vec{x})) > \kappa \implies \mathrm{class}(f(\vec{x})) = \mathrm{class}(f(\vec{x}'))$$

Next, we instantiate the above 2-safety property for confidence-based global robustness and fairness.

For *confidence-based global robustness*, cond is defined as:

$$\mathrm{cond}(\vec{x}, \vec{x}', \vec{\epsilon}) = \bigwedge_{i \in [1,n]} d(x_i, x_i') \leq \epsilon_i$$

For *confidence-based global fairness*, $\vec{x}$ can be split into sensitive $\vec{x_s}$ and non-sensitive $\vec{x_n}$ attributes. For confidence-based global fairness,

$$\mathrm{cond}(\vec{x}, \vec{x}', \vec{\epsilon}) = \bigwedge_{x_i \in \vec{x_s}} d(x_i, x_i') > 0 \, \wedge \bigwedge_{x_i \in \vec{x_n}} d(x_i, x_i') \leq \epsilon_i$$

where for any categorical $x_i \in \vec{x_n}$, its associated tolerance threshold $\epsilon_i = 0.5$.

Intuitively, confidence-based global fairness ensures that for any data instance, x classified with high confidence κ, no other data instance, x', that only differs with x in the value of the sensitive attribute (e.g. age, gender, ethnicity) shall be classified to a different class.

As defined in Sect. 2, $f(x)$ represents the feed-forward neural network, which maps inputs to classes with corresponding confidence scores. By introducing the threshold κ, our definition effectively ignores classification mismatches that arise from decisions with low confidence. The rationale is as follows:

– Different classifications close to decision boundaries need to be allowed, as safety can otherwise only be satisfied by degenerate neural networks that map all inputs to a single label.
– On the other hand, input points classified with a high confidence should be immune to adversarial perturbations and also uphold fairness.

3.1   Encoding 2-Safety Properties as Product Neural Network

In this section, we reduce checking of the 2-safety hyperproperty in Definition 4 to a safety property over a single trace. Given a neural network f (as defined earlier in Sect. 2), the product neural network is formed by composing a copy of the original neural network with itself. Checking 2-safety then reduces to checking an ordinary safety property for the self-composed neural network that consists of two copies of the original neural network, each with its own copy of the variables.

Table 1. Marabou's piecewise linear constraints

Equation	::=	Sum $\Diamond$ Constant with $\Diamond \in \{\leq, \geq, =\}$
		and Sum ::= Sum + Constant $\cdot$ Variable $\mid$ Constant $\cdot$ Variable
MaxConstraint	::=	Variable = $\max$(VariableList)
		and VariableList ::= VariableList, Variable $\mid$ Variable
AbsConstraint	::=	Variable = $\mid$Variable$\mid$
ReLUConstraint	::=	Variable = ReLU(Variable)
Disjunction	::=	(Disjunction $\vee$ Equation) $\mid$ Equation

The product neural network is now treated as the model to be verified. A product network allows the reduction of a hyperproperty to a trace property, thereby reducing the problem of hyperproperty verification to a standard verification problem, which can be solved using an existing standard verification technique.

Product Neural Network. We encode $f(\vec{x})$ using piecewise linear constraints (see Table 1). Each node $v_{i,j}$ is represented by two variables $\mathsf{in}_{i,j}$ and $\mathsf{out}_{i,j}$ representing its input and output, respectively. Inputs and outputs are related by the following constraints:

$$\mathsf{in}_{i,j} = \sum_{k=1}^{|V^{i-1}|} w_{i,j}^{i-1,k} \cdot \mathsf{out}_{(i-1,)k} \quad \wedge \quad \mathsf{out}_{i,j} = a_{i,j}(\mathsf{in}_{i,j})$$

where $w_{i,j}^{i-1,k}$ is the weight associated with the edge $(v_{i-1,k}, v_{i,j})$ and $a_{i,j}$ is the activation function of node $v_{i,j}$. To encode the self-composition, we duplicate all variables and constraints by introducing primed counterparts $\mathsf{in}'_{i,j}$ and $\mathsf{out}'_{i,j}$ for $\mathsf{in}_{i,j}$ and $\mathsf{out}_{i,j}$.

Transfer Functions and Operators. ReLUs can be readily encoded using $\mathsf{out}_{i,j} = \mathrm{ReLU}(\mathsf{in}_{i,j})$. There is, however, no direct way to encode softmax using the constraints in Table 1, hence we defer the discussion to Subsect. 3.2.

The conf operator can be implemented using the max constraint (cf. Eq. 3). The operator class as well as the implication, on the other hand, are not necessarily supported by state-of-the-art static analysis tools for DNNs. For instance, they are not supported by Marabou [23], on which we base our implementation. For reference, Table 1 illustrates the linear constraints supported by Marabou. We thus introduce an encoding, which we detail below.

First, checking the validity of the implication in Definition 4 can be reduced to checking the unsatisfiability of

$$\mathrm{cond}(\vec{x}, \vec{x'}, \vec{\epsilon}) \wedge \mathrm{conf}(f(\vec{x})) > \kappa \wedge \mathrm{class}(f(\vec{x})) \neq \mathrm{class}(f(\vec{x'})) \tag{7}$$

However, the grammar in Table 1 provides no means to encode disequality or class (which returns the *index* of the largest element of a vector). To implement

disequality, we perform a case split over all $n = |V^l|$ labels by instantiating the encoding of the entire network over $\mathsf{out}_{l,i}$ and $\mathsf{out}'_{l,i}$ for $1 \leq i \leq n$. To implement this in Marabou, we execute a separate query for every case.

To handle the operator class, we can encode the disequality $\mathrm{class}(f(\vec{x})) \neq \mathrm{class}(f(\vec{x}'))$ as:

$$\ldots \wedge \overbrace{\max(\mathsf{out}_{l,1}, \ldots, \mathsf{out}_{l,n})}^{\mathrm{conf}(f(\vec{x}))} > \kappa \;\wedge\; (\max(\mathsf{out}_{l,1}, \ldots, \mathsf{out}_{l,n}) - \mathsf{out}_{l,i} = 0) \;\wedge$$
$$(\max(\mathsf{out}'_{l,1}, \ldots, \mathsf{out}'_{l,n}) - \mathsf{out}'_{l,i} \neq 0) \quad (8)$$

The constraint $(\max(\mathsf{out}_{l,1}, \ldots, \mathsf{out}_{l,n}) - \mathsf{out}_{l,i} = 0)$ ensures that $\mathsf{out}_{l,i}$ corresponds to the largest element in $f(\vec{x})$ (and hence that $\mathrm{class}(f(\vec{x}) = i)$. Consequently, if $(\max(\mathsf{out}'_{l,1}, \ldots, \mathsf{out}'_{l,n}) - \mathsf{out}'_{l,i} \neq 0)$, then we can conclude that $\mathrm{class}(f(\vec{x}')) \neq i$ and hence the safety constraint is violated.

Since, Marabou does not support the disequality operator, we check whether $(\max(\mathsf{out}'_{l,1}, \ldots, \mathsf{out}'_{l,n}) - \mathsf{out}'_{l,i} < 0)$ and $(\max(\mathsf{out}'_{l,1}, \ldots, \mathsf{out}'_{l,n}) - \mathsf{out}'_{l,i} > 0)$, and if both constraints are not satisfied, we know that $(\max(\mathsf{out}'_{l,1}, \ldots, \mathsf{out}'_{l,n}) - \mathsf{out}'_{l,i} \neq 0)$.

While the above transformation is equivalence preserving, the encoding of softmax requires an approximation, described in the following subsection.

3.2   Softmax Approximation

Softmax in Terms of Max and Sig. We can approximate softmax using a max operator and a sigmoid function as follows. Consider $\mathrm{softmax}(\vec{z})_i$ (cf. Eq. 2), for $i = 1$,

$$\mathrm{softmax}(\vec{z})_1 = \frac{1}{1 + (e^{z_2} + \cdots + e^{z_n})e^{-z_1}} \quad (9)$$

$$= \frac{1}{1 + e^{log(e^{z_2} + \cdots + e^{z_n})}e^{-z_1}} = \frac{1}{1 + e^{(-z_1 + log(e^{z_2} + \cdots + e^{z_n}))}} \quad (10)$$

We can now generalize the result from (9) for i

$$\mathrm{softmax}(\vec{z})_i = \frac{1}{1 + e^{(-z_i + log(\sum_{j \neq i}^n e^{z_j}))}} = \mathrm{Sig}(z_i - \mathop{\mathrm{LSE}}_{\substack{1 \\ j \neq i}}^{n}(z_j)) \quad (11)$$

where LSE (the *log-sum-exp*) is:

$$\mathop{\mathrm{LSE}}_{\substack{1 \\ j \neq i}}^{n}(z_j) = log\left(\sum_{j=1, j \neq i}^{n} e^z_j\right) \text{ and } \mathrm{Sig}(x) = \frac{1}{1 + e^{-x}}$$

We know from [42] that LSE is bounded:

$$max_1^n(z_i) \leq \mathop{\mathrm{LSE}}_1^n(z_i) \leq max_1^n(z_i) + log(n) \quad (12)$$

with $max_1^n(z_i) = max(z_1, \cdots, z_n)$, in particlular, when $z_1 = \cdots = z_n$, we have:

$$\underset{1}{\overset{n}{\text{LSE}}}(z_i) = max_1^n(z_i) + log(n) \tag{13}$$

Then softmax has as lower bound:

$$\text{softmax}(\vec{z})_\text{i} \geq \text{Sig}(z_\text{i} - \underset{j\neq i}{max_1^n}(z_j) + log(n-1)) \tag{14}$$

and as upper bound:

$$\text{softmax}(\vec{z})_\text{i} \leq \text{Sig}(z_\text{i} - \underset{j\neq i}{max_1^n}(z_j)) \tag{15}$$

When $n = 2$, the softmax is equivalent to the sigmoid:

$$\text{softmax}(\vec{z})_1 = \text{Sig}(z_1 - z_2) \text{ and softmax}(\vec{z})_2 = \text{Sig}(z_2 - z_1) \tag{16}$$

Now that we know how to approximate a softmax using a sigmoid and max, we need to find a piece-wise linear approximation of sigmoid since sigmoid is also a non-linear exponential function.

Piece-Wise Approximation of Sigmoid. We approximate sigmoid as a piece-wise linear function using the Remez exchange algorithm [43]. The Remez algorithm is an iterative algorithm that finds simpler approximations to functions. It aims to minimize the maximum absolute difference between the approximated polynomial and the actual function. The algorithm takes a maximum acceptable error δ and generates l linear segments to approximate the sigmoid function such that the error is less than δ. We use the Remez algorithm to approximate the sigmoid in the interval $[\text{Sig}^{-1}(\delta), \text{Sig}^{-1}(1 - \delta)]$, where Sig^{-1} is the inverse function of the sigmoid. The inverse of sigmoid is the logit function i.e., $\text{Sig}^{-1}(y) = logit(y) = log(y)/(1 - y)$. For example, if the user sets δ to 0.0006, then the input domain for the algorithm lies in $[-7.423034723582278, 7.423034723582278]$. Thus, the approximated sigmoid is:

$$|\widehat{\text{Sig}}(x) - \text{Sig}(x)| \leq \delta$$

We approximate softmax with its lower bound:

$$\widehat{\text{softmax}}(\vec{z})_\text{i} = \widehat{\text{Sig}}(z_\text{i} - \underset{j\neq i}{max_1^n}(z_j) + log(n-1)) - \delta \tag{17}$$

and the upper bound for the softmax is:

$$\widehat{\text{softmax}}(\vec{z})_\text{i} \leq \text{softmax}(\vec{z})_\text{i} \leq \widehat{\text{Sig}}(z_\text{i} - \underset{j\neq i}{max_1^n}(z_j)) + \delta \tag{18}$$

Theorem 1. *Let* softmax *and* $\widehat{\text{softmax}}$ *compute the real and linearly approximated softmax (with precision δ), respectively for the last layer of $n \geq 2$ neurons $\vec{z}$ of a neural network and $z_i = max(z_1, \cdots, z_n)$. Then, we have the following result:*

$$\forall \vec{z}. \ \text{softmax}(\vec{z})_i - \widehat{\text{softmax}}(\vec{z})_i \leq \frac{n-2}{(\sqrt{n-1}+1)^2} + 2\delta$$

Proof. We refer to [44]. □

Theorem 2. *(Class consistency) Let f and $\hat{f}$ denote the real and the approximated (with precision δ) neural networks with $n \geq 2$ outputs, respectively. Then:*

$$\text{conf}(\hat{f}(\vec{x})) > \frac{1}{2} \implies \text{class}(\hat{f}(\vec{x})) = \text{class}(f(\vec{x}))$$

Proof. We refer to [44]. □

Soundness. For the confidence-based 2-safety property discussed before, our analysis provides a soundness guarantee. This means that whenever the analysis reports that the property specified in Definition 4 holds, then the property also holds true in the concrete execution.

Theorem 3. *(Soundness) Let f and $\hat{f}$ be the original neural network and over-approximated neural network, respectively. Let $b_{n,\delta}$ be the error bound of the approximated softmax ($b_{n,\delta} = \frac{n-2}{(\sqrt{n-1}+1)^2} + 2\delta$ (see Theorem 1)). Then we have the following soundness guarantee: Whenever the approximated neural network is 2-safe for $\text{conf}(\hat{f}(\vec{x})) > (\kappa - b_{n,\delta})$, the real neural network is 2-safe for $\text{conf}(f(\vec{x})) > \kappa$, given $\text{conf}(\hat{f}(\vec{x})) > \frac{1}{2}$. Formally:*

$$\begin{pmatrix} \forall \vec{x}, \vec{x'}. \ \text{cond}(\vec{x}, \vec{x'}, \vec{\epsilon}) \wedge \text{conf}(\hat{f}(\vec{x})) > (\kappa - b_{n,\delta}) \\ \implies \text{class}(\hat{f}(\vec{x})) = \text{class}(\hat{f}(\vec{x'})) \end{pmatrix} \implies$$

$$\begin{pmatrix} \forall \vec{x}, \vec{x'}. \ \text{cond}(\vec{x}, \vec{x'}, \vec{\epsilon}) \wedge \text{conf}(f(\vec{x})) > \kappa \\ \implies \text{class}(f(\vec{x})) = \text{class}(f(\vec{x'})) \end{pmatrix}, \ with \ \text{conf}(\hat{f}(\vec{x})) > \frac{1}{2}$$

Proof. We refer to [44]. □

4   Implementation

For the implementation of our technique, we use the state-of-the-art neural network verification tool Marabou [23] as our solver. In this section, we describe how we encode the confidence-based 2-safety property in Marabou. Note that such an encoding can be expressed in a similar way for virtually any off-the-shelf neural network verifier.

Marabou [23]. Marabou is a simplex-based linear programming neural network verification and analysis tool. Marabou is capable to address queries about network's properties, such as local robustness, by encoding them into constraint

satisfaction problems. It supports fully-connected feed-forward neural networks. A network can be encoded as a set of linear constraints representing the weighted sum of the neurons' outputs feeding the next neuron's input, and a set of non-linear constraints defining the activation functions. A verification query to Marabou comprises a neural network along with a property that needs to be verified. This property is defined as "linear and nonlinear constraints on the network's inputs and outputs" [23]. In Marabou, network's neurons are treated as variables. As a result, the verification problem involves identifying a variable assignment that satisfies all the constraints at the same time, or establishing that such an assignment does not exist. The tool uses a variant of the *Simplex* algorithm at its core to make the variable assignment satisfy the linear constraints. During the execution, the tool adjusts the variable assignment to either fix a linear or a non-linear constraint violation. Although the technique implemented in Marabou is sound and complete, the tool can work only with piece-wise linear activation functions (including ReLU function and the max function) to guarantee termination. Additionally, an essential aspect of Marabou's verification approach is deduction – specifically, deriving more precise lower and upper bounds for each variable. The tool leverages these bounds to relax piece-wise linear constraints into linear ones by considering one of its segments.

The original network g is a function of the following: input parameters, neurons, neuron connection weights, layer biases, ReLU activation functions and output classes. To make Marabou amenable for verification of 2-safety properties, we need a product neural network. This means that the execution is tracked over two copies of the original network, g and g' (cf. Subsect. 2.3). Let X_i denote the set of input variables to g and let X_i' be a set of primed copies of the variables in X_i. As a result, we obtain a self-composition $g(X_i) \times g'(X_i')$ of g over the input variables $X_i \cup X_i'$.

Next, we extend the output layer with softmax function in order to extract the confidence scores with which output classes are predicted.

Linearized Sigmoid. We explain our linearized sigmoid function in this subsection. This function is used to implement an approximated piece-wise linear sigmoid function. Let the outputs of the last inner layer $l-1$ be represented by z_i for $1 \leq i \leq n$, where n is the number of output classes. In Marabou, we first encode the linear piece-wise sigmoid function which we obtain by setting the maximum acceptable error to 0.005. This provides us with a piece-wise linear approximated sigmoid with 35 segments of the form $q_j = \{m_j \cdot z_i + c_j, \mid LB \leq z_i \leq UB\}$, where z_i is the variable representing the output node whose confidence we want to find. We encode each segment as an equation in and represent it using a variable q_j. Next, we need to select the applicable segment corresponding to the value of z_i. Unfortunately, Marabou does not provide a conditional construct. So, we deploy the min and max functions to emulate if-then-else.

First, we split the sigmoid into two convex pieces S_1 and S_2. Figure 2 illustrates this step using a simplified approximation of sigmoid with 4 linear segments q_1, q_2, q_3, and q_4. The resulting value of S_1 can now be expressed as $S_1 = \min(\max(0, q_1, q_2), 0.5)$. Similarly, $S_1 = \max(\min(0.5, q_3, q_4), 1)$. The values 0 and

1 are the minimum and maximum values of the sigmoid function and 0.5 is the value of sigmoid at the splitting point. Second, we combine the convex segments by adding them:

$$S = \min(\max(0, q_1, q_2), 0.5) + \max(\min(0.5, q_3, q_4), 1) - 0.5$$

Note that we have 35 segments instead of four used in our simplified explanation.

Linearized Softmax. The next step consists in implementing the softmax function using the output of the sigmoid function and the max function (see Eq. 17). To this end, we find the maximum of all output nodes excluding the current one, and subtract that maximum from the current output value. Finally, we apply the linearized sigmoid (cf. Sect. 4), to obtain the result of softmax.

We repeat the above steps for all output nodes to obtain the softmax values corresponding to all output classes. Finally, we find the maximum value of these softmax outputs, which represents the confidence.
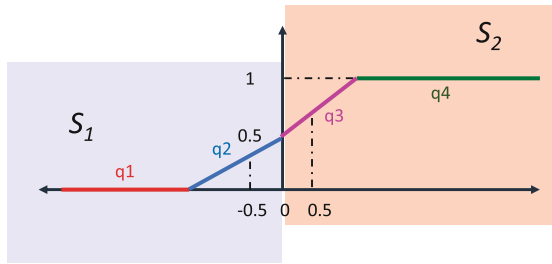
Fig. 2. (Simplified) Approximation of sigmoid with 4 linear segments

5   Experimental Evaluation

For our evaluation, we used four publicly available benchmark datasets to evaluate our technique. We pre-process the datasets to remove null entries, select relevant categorical attributes, and hot-encode them. For each dataset, we train a fully connected feed-forward neural network with up to 50 neurons and ReLU activation functions.

German Credit: The German Credit Risk dataset [45] describes individuals requesting credit from a bank and classified, based on their characteristics, in two categories ("good" or "bad") of credit risk. The dataset comprises 1000 entries.

Adult: The Adult dataset, also referred to as the "Census Income" dataset, is used to estimate whether a person's income surpasses $50,000 per year based on census information [46].

COMPAS: COMPAS ("Correctional Offender Management Profiling for Alternative Sanctions") is a widely-used commercial algorithm that is utilized by judges and parole officers to assess the probability of criminal defendants committing future crimes, also known as recidivism [47].

Law School: The Law School Admissions Council (LSAC) provides a dataset called Law School Admissions, which includes information on approximately 27,000 law students from 1991 to 1997. The dataset tracks the students' progress through law school, graduation, and bar exams. It uses two types of academic scores (LSAT and GPA) to predict their likelihood of passing the bar exam [48].

We use TensorFlow for training neural networks and the NN verifier Marabou [23] whose implementation is publicly available. The accuracies for the deployed models are as follows: German Credit: 0.71; COMPAS: 0.74; Law: 0.94; Adult: 0.77. In our experiments, adding more layers or nodes per layer did not result in an increased accuracy. We run all our experiments using a single AMD EPYC 7713 64-Core Processor, Ubuntu 22.04 LTS Operating System with 32 GB RAM.
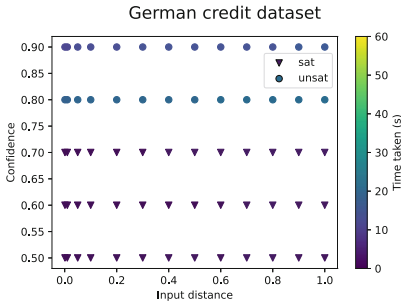
Fig. 3. Input distance vs. confidence for German credit dataset
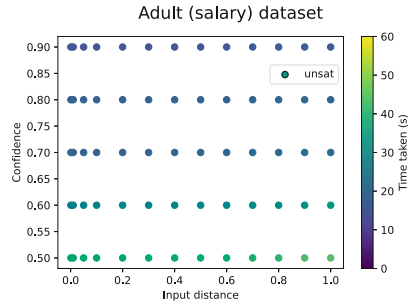
Fig. 4. Input distance vs. confidence for adult dataset
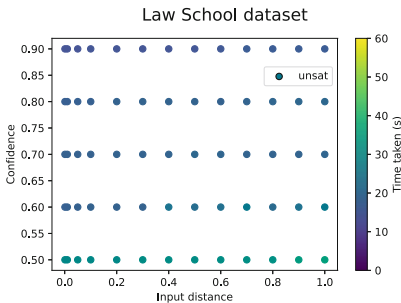
Fig. 5. Input distance vs. confidence for law school dataset
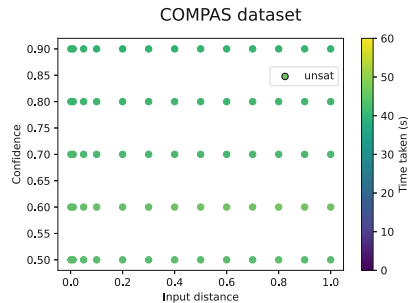

Fig. 6. Input distance vs. confidence for COMPAS dataset

First, we present our confidence-based global robustness results. We evaluate our implementation on the neural networks trained with the benchmark datasets

for various combinations of input distance and confidence values. We aim to find proofs for globally robust neural networks. The plots in Figs. 3, 4, 5 and 6 show our experimental results as scatter plots. Markers denoting 'sat' correspond to the query resulting in a counter-example. A counter-example here means that for the input distance and confidence values in that query, the inputs are classified into different output classes. The 'unsat' markers stand for the query being proved (i.e. the model is robust), which means that for the corresponding input distance and confidence threshold, the inputs are classified to the same output class and the model is globally robust. The color bar on the right side denotes the time taken in seconds to run each query; the time scale goes from deep purple to blue, green and yellow as the time taken increases from 0 to 60 sec. The plot in Fig. 3 depicts the effect of varying input distance and confidence on the German credit benchmark. We ran our query with the confidence-based global robustness property for input distance, ϵ and confidence κ values ranging from 0.001 to 1.0 and 0.5 to 0.9, respectively. Observe that for κ values below 0.7, the model is sat i.e. we find counter-examples. However, for confidence values above 0.75, even for larger input distances, the queries result in unsat and a proof that the model is robust above a confidence threshold of 0.75.

The plots in Figs. 4, 5 and 6 show the results for neural networks trained with Adult, Law School, and COMPAS datasets. As can be observed from the scatter plots, these models are robust. For confidence values above 0.5, they are 2-safe and we are successfully able to prove this rather fast in 50 s or less.

Table 2. Global fairness on German credit/COMPAS datasets for various criteria

Dataset	Sensitive attribute	Confidence threshold	Result	Time taken
German credit	Gender	0.5	unsat	10.232 s
German credit	Age	0.5	unsat	11.478 s
COMPAS	Gender	0.5	sat	7.423 s
COMPAS	Ethnicity	0.5	sat	18.293 s
COMPAS	Ethnicity	0.99	sat	25.846 s
COMPAS	Ethnicity	0.999	unsat	171 min 15 s

Next, we present the results for confidence-based global fairness verification, which are shown in Table 2. Each row in the table depicts the verification result for a NN along with the sensitive attribute and confidence threshold considered. If the result is 'unsat', it means that the query is proved (i.e. the model is fair). In other words, for the corresponding sensitive attribute and confidence value constraints, the inputs are classified to the same output class and the model is globally fair. On the other hand, 'sat' corresponds to the query resulting in a counter-example. A counter-example here means that for the corresponding sensitive attribute and confidence threshold in the query, the inputs are classified into different output classes.

The German credit model is proved to be globally fair for confidence values above 0.5 for sensitive attributes Gender and Age. Running our query with the confidence-based global fairness property for the COMPAS model, with Gender as the sensitive attribute gives counter-examples for all confidence values. Additionally, when Ethnicity is considered as the sensitive attribute while verifying the COMPAS model, we find counter-examples for lower confidence values. However, the model is proved to be globally fair for confidence values above 0.999.

We combined our method with binary search, to synthesize the minimum confidence for which the DNN is globally robust or fair. We perform the binary search, starting with confidence 0.5. If the model is unsat, we are done. Else, we check for confidence $mid = (0.5 + 1)/2$, and continue in this way till we find the minimum confidence accurate to the nearest 0.05. For instance, binary search combined with our method, on German credit gave us 0.75 (in 45 s) to be the minimum confidence for which the DNN is globally robust.

Our experimental results on 2-safety properties (with regard to both, global robustness and global fairness), clearly point out that taking confidence along with input distance into account is crucial when verifying neural networks.

5.1   Discussion

Soundness. Our proof of soundness guarantees that if our approach yields that a model is robust or fair for a given confidence and input distance, the model is indeed safe. In case of the German Credit model, for instance, the model is indeed globally robust for all input distances when the confidence is at least 0.75. Moreover, we can use binary search to find the minimal confidence value above which a model is robust. Our approach guarantees soundness and when a model is found to be safe, it is indeed safe. However, if a counterexample is found, it may be a false positive. False positives (or spurious counterexamples) may in general stem either from overapproximations of the underlying reachability analysis tool or from our own softmax approximation. In our implementation, the former are not present since Marabou is complete (i.e., it does not have false positives), whereas our softmax approximation yields a confidence error that depends on the number of DNN-outputs, as formalized in Theorem 1 and quantified in its proof. For DNNs with two outputs, such as German Credit, Adult, and Law School, there is no error, whereas for three outputs (COMPAS) the error is $\sim$ 0.171. Hence, if we want to certify a three-output DNN for confidence x, we run our analysis for confidence $x - 0.171$: if no attack is found, we can certify the network for confidence x (soundness), otherwise we know the counterexample violates the 2-safety property (completeness of Marabou) for confidence in between x and $x - 0.171$ (the possible imprecision is due to our softmax overapproximation). We report counterexamples in this paper on German Credit and on COMPAS: the former are true positives (2-output DNN), whereas the latter are counterexamples for confidence in between 0.999 (the confidence we can certify the network for) and 0.828. We ran the network on the counterexample reported by Marabou and found the real confidence to be 0.969. Hence, the

network is for sure fair for a confidence higher than 0.999, unfair for confidence levels lower than 0.969, while we cannot decide it for the confidence levels in the interval between 0.969 and 0.999. This means that on our datasets, our analysis is very accurate.

Threats to Validity. Presuming a high level of confidence as a precondition can make low-confidence networks vacuously safe. However, an accurate but low-confidence network is not desirable in the first place. This concept is known in the literature as miscalibration. The Expected Calibration Error is defined as weighted average over the absolute difference between confidence and accuracy. In scenarios where accurate confidence measures are crucial, the goal is to reduce the Maximum Calibration Error [49] that is the maximum discrepancy between confidence and accuracy.

This is orthogonal to our work and there is an entire field of research [50,51] aiming at minimizing such calibration errors.

6   Conclusion

We introduce the first automated method to verify 2-safety properties such as global robustness and global fairness in DNNs based on the confidence level. To handle the nonlinear softmax function computing the confidence, we approximate it with a piece-wise linear function for which we can bound the approximation error. We then compute the self-composition of the DNN with the approximated softmax and we show how to leverage existing tools such as Marabou to verify 2-safety properties. We prove that our analysis on the approximated network is sound with respect to the original one when the value of confidence is greater than 0.5 in the approximated one. We successfully evaluate our approach on four different DNNs, proving global robustness and global fairness in some cases while finding counterexamples in others.

While we improve over recent verifiers for global properties that are limited to binary classifiers [25], a limitation of our current approach is that we can only handle DNNs with few (two to five) outputs, since the approximation error increases with the number of outputs. We plan to overcome this limitation in future work by devising more accurate abstractions of softmax.

To improve scalability, we will investigate how to refine our approach by integrating pruning strategies, such as those developed in [25], which we intend to refine to fit our static analysis framework.

We also plan to explore more sophisticated and effective verification techniques for 2-safety properties, possibly tailored to specific DNN structures.

Finally, we plan to complement our verification approach with testing techniques to further explore the generated counterexamples.

References

1. Gurney, K.: An Introduction to Neural Networks. CRC Press, Boca Raton (1997)
2. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive Computation and Machine Learning. MIT Press, Cambridge (2016)
3. Amato, F., López, A., Peña-Méndez, E.M., Vaňhara, P., Hampl, A., Havel, J.: Artificial neural networks in medical diagnosis (2013)
4. Rao, Q., Frtunikj, J.: Deep learning for self-driving cars: chances and challenges. In: Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems, pp. 35–38 (2018)
5. Duan, J.: Financial system modeling using deep neural networks (DNNs) for effective risk assessment and prediction. J. Franklin Inst. **356**(8), 4716–4731 (2019)
6. Tran, H., Xiang, W., Johnson, T.T.: Verification approaches for learning-enabled autonomous cyber-physical systems. IEEE Des. Test **39**(1), 24–34 (2022)
7. Bjesse, P.: What is formal verification?. ACM SIGDA Newsl. **35**(24), 1–es (2005)
8. Seshia, S.A., Desai, A., Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Shivakumar, S., Vazquez-Chanlatte, M., Yue, X.: Formal specification for deep neural networks. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 20–34. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_2
9. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
10. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
11. Gopinath, D., Katz, G., Păsăreanu, C.S., Barrett, C.: DeepSafe: a data-driven approach for assessing robustness of neural networks. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 3–19. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_1
12. Urban, C., Christakis, M., Wüstholz, V., Zhang, F.: Perfectly parallel fairness certification of neural networks. Proc. ACM Program. Lang. **4**(OOPSLA), 1–30 (2020)
13. Xie, X., Zhang, F., Hu, X., Ma, L.: Deepgemini: verifying dependency fairness for deep neural network. Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, pp. 15251–15259 (2023)
14. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Commun. **25**(2), 117–135 (2012)

15. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: towards higher precision and faster verification. In: Chang, B.-Y.E. (ed.) SAS 2019. LNCS, vol. 11822, pp. 296–319. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_15

16. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 243–257. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24

17. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019)

18. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE (2018)

19. Baninajjar, A., Hosseini, K., Rezine, A., Aminifar, A.: Safedeep: a scalable robustness verification framework for deep neural networks. In: ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5 (2023)

20. 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

21. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. arXiv preprint arXiv:1711.07356 (2017)

22. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) NFM 2018. LNCS, vol. 10811, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77935-5_9

23. 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

24. Khedr, H., Shoukry, Y.: Certifair: a framework for certified global fairness of neural networks. arXiv preprint arXiv:2205.09927 (2022)

25. Biswas, S., Rajan, H.: Fairify: fairness verification of neural networks. In: 45th IEEE/ACM International Conference on Software Engineering. ICSE 2023, Melbourne, Australia, 14–20 May 2023, pp. 1546–1558. IEEE (2023)

26. Chen, Y., Wang, S., Qin, Y., Liao, X., Jana, S., Wagner, D.: Learning security classifiers with verified global robustness properties. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 477–494 (2021)

27. 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, vol. 34, pp. 29909–29921 (2021)

28. Xu, K., et al.: Fast and complete: enabling complete neural network verification with rapid and massively parallel incomplete verifiers. arXiv preprint arXiv:2011.13824 (2020)

29. 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, vol. 31 (2018)

30. Ferrari, C., Muller, M.N., Jovanovic, N., Vechev, M.: Complete verification via multi-neuron relaxation guided branch-and-bound. arXiv preprint arXiv:2205.00263 (2022)

31. Bak, S., Tran, H.-D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying ReLU neural networks. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 66–96. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_4

32. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. Math. Struct. Comput. Sci. **21**(6), 1207–1252 (2011)

33. Huang, G.-B.: Learning capability and storage capacity of two-hidden-layer feed-forward networks. IEEE Trans. Neural Netw. **14**(2), 274–281 (2003)

34. Sharma, S., Sharma, S., Athaiya, A.: Activation functions in neural networks. Towards Data Sci. **6**(12), 310–316 (2017)

35. Albarghouthi, A., et al.: Introduction to neural network verification. Found. Trends® Program. Lang. **7**(1–2), 1–157 (2021)

36. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010)

37. Barthe, Gilles, Crespo, Juan Manuel, Kunz, César.: Relational verification using product programs. In: Butler, Michael, Schulte, Wolfram (eds.) FM 2011. LNCS, vol. 6664, pp. 200–214. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21437-0_17

38. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. In: Advances in Neural Information Processing Systems, vol. 29 (2016)

39. Mangal, R., Nori, A.V., Orso, A.: Robustness of neural networks: a probabilistic and practical approach. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 93–96. IEEE (2019)

40. Leino, K., Wang, Z., Fredrikson, M.: Globally-robust neural networks. In: International Conference on Machine Learning, pp. 6212–6222. PMLR (2021)

41. Binns, R.: Fairness in machine learning: lessons from political philosophy. In: Conference on Fairness, Accountability and Transparency, pp. 149–159. PMLR (2018)

42. Pant, Y.V., Abbas, H., Mangharam, R.: Smooth operator: control using the smooth robustness of temporal logic. In: 2017 IEEE Conference on Control Technology and Applications (CCTA), pp. 1235–1240. IEEE (2017)

43. Remez, E.Y.: Sur la détermination des polynômes d'approximation de degré donnée (1934)

44. Athavale, A., Bartocci, E., Christakis, M., Maffei, M., Nickovic, D., Weissenbacher, G.: Verifying global two-safety properties in neural networks with confidence. arXiv preprint arXiv:2405.14400 (2024)

45. Hofmann, H., German credit dataset. UCI Machine 2023 Repository. University of California, School of Information and Computer Science, Irvine (1994). http://archive.ics.uci.edu/ml

46. Dua, D., Graff, C.: UCI machine 2023 repository (2017)

47. Larson, J. (2017). https://github.com/propublica/compas-analysis

48. Wightman, L.F.: LSAC national longitudinal bar passage study. LSAC research report series (1998)

49. Naeini, M.P., Cooper, G., Hauskrecht, M.: Obtaining well calibrated probabilities using Bayesian binning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29 (2015)

50. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International Conference on Machine Learning, pp. 1321–1330. PMLR (2017)
51. Ao, S., Rueger, S., Siddharthan, A.: Two sides of miscalibration: identifying over and under-confidence prediction for network calibration. In: Uncertainty in Artificial Intelligence, pp. 77–87. PMLR (2023)