Kalpa Publications in Computing
Volume 16, 2023, Pages 71–82
Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems

Verifying Global Neural Network Specifications using Hyperproperties

David Boetius and Stefan Leue

University of Konstanz, 78457 Konstanz, Germany
{david.boetius,stefan.leue}@uni-konstanz.de

Abstract

Current approaches to neural network verification focus on specifications that target small regions around known input data points, such as local robustness. Thus, using these approaches, we can not obtain guarantees for inputs that are not close to known inputs. Yet, it is highly likely that a neural network will encounter such truly unseen inputs during its application. We study global specifications that — when satisfied — provide guarantees for all potential inputs. We introduce a hyperproperty formalism that allows for expressing global specifications such as monotonicity, Lipschitz continuity, global robustness, and dependency fairness. Our formalism enables verifying global specifications using existing neural network verification approaches by leveraging capabilities for verifying general computational graphs. Thereby, we extend the scope of guarantees that can be provided using existing methods. Recent success in verifying specific global specifications shows that attaining strong guarantees for all potential data points is feasible.

1 Introduction

Deep learning is a game changer for research, education, business and beyond [9, 11]. Yet, we remain unable to provide strong guarantees on the behaviour of neural networks. In particular, while neural network verification in principle can provide strong guarantees, current approaches almost exclusively consider *local* specifications [1, 14, 20, 25, 32, 38] that only apply to small regions around known input data points. This means that the currently widely-used specifications only sparsely cover the input space, providing no guarantees for inputs that are not close to known inputs. In contrast, *global* specifications cover the entire input space.

We propose a specification formalism for neural networks that encompasses a rich class of global specifications while enabling verification using existing verifier technology. In particular, we show how monotonicity, Lipschitz continuity, two notions of global robustness [21, 24], and dependency fairness [15, 35] can be expressed using our formalism.

As noted in [30], global specifications such as monotonicity and global robustness are hyperproperties [8]. In difference to regular properties that only consider one network execution at a time, hyperproperties relate executions for several inputs of the same neural network to each other. This allows us, for example, to express a naïve notion of global robustness stating that an arbitrary input and a second input that lies close need to receive the same classification.

A central aspect of our formalism is that we use auxiliary neural networks to define input sets and output sets. By leveraging capabilities for verifying general computational graphs [37], the auxiliary networks, together with *self-composition* [8], allow for verifying hyperproperties using existing neural network verification approaches. Here, the role of the auxiliary neural networks is to make complex hyperproperty input and output sets accessible to existing verification approaches. Concretely, we design an auxiliary neural network to generate the tuples of inputs that need to be compared to determine whether a hyperproperty is satisfied. Another auxiliary neural network detects whether the outputs a network produces for these inputs satisfy the output constraint. For the naïve notion of global robustness, this means that we derive a neural network that generates arbitrary pairs of inputs that are close to each other and another neural network that detects whether two outputs represent the same classification. Importantly, these auxiliary neural networks *exactly* capture the targeted input and output constraints using standard neural network components.

Recent success in verifying global robustness [36] and global individual fairness [35] demonstrates that verifying global specifications is feasible. Our formalism is a general framework for global specifications targeting existing verifiers [14, 22, 32, 38]. While our formalism does not alleviate the need for specialised techniques, such as the Interleaving Twin Encoding [36], it allows for

1. Comparing general-purpose verifiers with specialised verifiers for specific global specifications and
2. Applying general-purpose verifiers to global specifications for which no specialised verifier exists.

2 Preliminaries

We consider verifying whether a neural network conforms to a global specification. Neural networks are computational graphs [16]. Global specifications are formalised using hyperproperties [8, 30].

Definition 1 (Computational Graph)**.** A *computational graph* is a directed acyclic graph with computations (V, E, h), where $V = \{1, \ldots, L\}$ with $L \in \mathbb{N}$ is the set of nodes, $E \subseteq V \times V$ is the edge relation and $h = (h_1, \ldots, h_L)$ is the computations tuple. Let degin : $V \to \mathbb{N}$ denote the in-degree. The computation of node $i \in V$ is $h_i : \mathbb{R}^{m_{k_1}} \times \cdots \times \mathbb{R}^{m_{k_{\mathrm{degin}(i)}}} \to \mathbb{R}^{m_i}$, where $m_i \in \mathbb{N}$ is the output dimension of node i and $k_1, \ldots, k_{\mathrm{degin}(i)} \in \{i \mid (k, i) \in E\}$ with $k_1 \leq \cdots \leq k_{\mathrm{degin}(i)}$ are the direct predecessors of i.

Definition 2 (Neural Network)**.** A *neural network* $\mathrm{net}_\theta : \mathbb{R}^n \to \mathbb{R}^m$, $n, m \in \mathbb{N}$ is a composition of affine transformations and non-affine activation functions represented by a computational graph (V, E, h) with a source i and a single sink j, such that $h_i : \{\emptyset\} \to \mathbb{R}^n$ and $h_j : \mathbb{R}^{m_{k_1}} \times \cdots \times \mathbb{R}^{m_{k_{\mathrm{degin}(j)}}} \to \mathbb{R}^m$. The source i is the *input* of net_θ. The remaining sources of the computational graph together form the *parameters* $\boldsymbol{\theta}$ of net_θ. The sink j is the *output* of net_θ. For classification tasks, $\arg\max_{j=1}^{m} \mathrm{net}_\theta(\mathbf{x})$ is the class net_θ assigns to an input $\mathbf{x} \in \mathbb{R}^n$.

Figure 1 contains the computational graph of a residual unit [19] as an example. This graph defines the steps necessary for computing the output of a residual unit, given an input. It also allows for computing gradients and verifying a residual unit. Assume we want to compute the outputs of a neural network for an input $\mathbf{x} \in \mathbb{R}^n$. Also, assume we have a parameter assignment $\boldsymbol{\theta}$. We assign $\mathbf{x}$ to the network input node i and the corresponding parameter

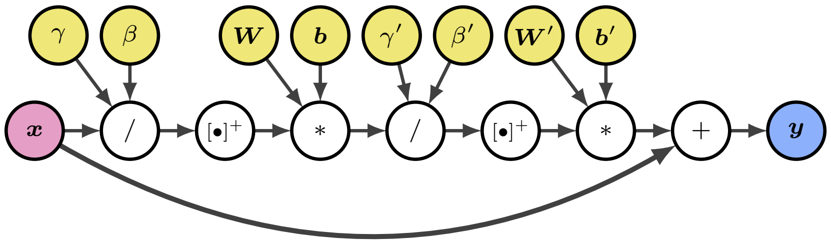

Figure 1: **The computational graph of a residual unit [19].** In this figure, $*$ denotes convolution, / denotes batch normalisation, $[\bullet]^+$ denotes ReLU, and $+$ denotes addition. We use pink nodes 🟪 for inputs, yellow 🟨 for parameters, and blue 🟦 for outputs.

values to the remaining sources. Now, computing the outputs corresponds to a forward walk over the computational graph, propagating the computation results of each node to its direct successors. Similarly, a backwards walk from sinks to sources allows for computing the gradients of the sink with respect to each source (backpropagation). Forward and backwards walks also allow for computing certified lower and upper bounds on the network output that can be used for verifying the neural network [37].

Verifying a neural network means that we want to automatically prove or disprove whether the neural network satisfies a *specification*. A specification is a set of properties.

Definition 3 (Property)**.** A *property* $\varphi = (\mathcal{X}_\varphi, \mathcal{Y}_\varphi)$ is a tuple of an *input set* $\mathcal{X}_\varphi \subseteq \mathbb{R}^n$ and an *output set* $\mathcal{Y}_\varphi \subseteq \mathbb{R}^m$, $n, m \in \mathbb{N}$. We write $\text{net}_\theta \vDash \varphi$ when a neural network $\text{net}_\theta : \mathbb{R}^n \to \mathbb{R}^m$ *satisfies* the property φ. Specifically,

$$\text{net}_\theta \vDash \varphi \Leftrightarrow \forall \mathbf{x} \in \mathcal{X}_\varphi : \text{net}_\theta(\mathbf{x}) \in \mathcal{Y}_\varphi.$$

We call an input $\mathbf{x} \in \mathcal{X}_\varphi$ for which $\text{net}_\theta(\mathbf{x}) \notin \mathcal{Y}_\varphi$ a *counterexample.*

A verifier determines whether a neural network net_θ satisfies a property φ. We require verifiers to **1.** report property satisfaction if and only if the property is indeed satisfied (*soundness*) and **2.** to terminate (*completeness*). In this paper, we only require verifiers to support bounded hyperrectangles as property input sets and the non-negative real numbers as output set. Practically, verifiers can also handle more complicated input and output sets.

For formalising global specifications, we make use of *hyperproperties.* Hyperproperties extend properties by considering multiple input variables and input-dependent output sets.

Definition 4 (Hyperproperty)**.** A *hyperproperty* $\psi = (\mathcal{X}_\psi, \mathcal{Y}_\psi)$ is a tuple of a *multi-variable input set* $\mathcal{X}_\psi \subseteq (\mathbb{R}^n)^v$ and an *input-dependent output set*

$$\mathcal{Y}_\psi \subseteq \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{v \text{ times}} \times \underbrace{\mathbb{R}^m \times \cdots \times \mathbb{R}^m}_{v \text{ times}},$$

where $n, m, v \in \mathbb{N}$. For a neural network $\text{net}_\theta : \mathbb{R}^n \to \mathbb{R}^m$, we write $\text{net}_\theta \vDash \psi$ if

$$\forall \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(v)} \in \mathcal{X}_\psi : \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(v)}, \text{net}_\theta\left(\mathbf{x}^{(1)}\right), \ldots, \text{net}_\theta\left(\mathbf{x}^{(v)}\right)\right) \in \mathcal{Y}_\psi.$$

3 Formalising Global Specifications using Hyperproperties

Global specifications allow for expressing desired behaviour for the entire input domain of a neural network while local specifications only apply to small regions around known inputs. This property of local specifications brings with it that we have a fixed reference point for each property in a local specification. We typically do not have such a fixed reference point for global specifications, since they apply to the entire input domain.

For example, a local robustness property expresses that a classifier assigns the same class to all inputs that lie within a small L_p-ball $\mathcal{B}_p(\mathbf{x})$ around a fixed input point $\mathbf{x}$. Because we have this fixed input $\mathbf{x}$ as a reference point, we know the class that should be assigned to all the inputs in $\mathcal{B}_p(\mathbf{x})$. Knowing this class allows for judging whether an input $\mathbf{x}' \in \mathcal{B}_p(\mathbf{x})$ is a counterexample to the local robustness property by executing the network once for $\mathbf{x}'$.

If we now look at global robustness, we find that it does not suffice to consider a single execution of a network to check for specification violations. As the inputs now are arbitrary inputs from the entire input domain, we can not determine whether robustness is violated by looking only at the output for one input $\mathbf{x}^{(1)}$. Instead, we need to find another input $\mathbf{x}^{(2)} \in \mathcal{B}_p\big(\mathbf{x}^{(1)}\big)$ such that the classes that a network assigns to $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ do not match. Only in pair, these inputs form a counterexample. The necessity to compare outputs for multiple inputs requires us to adopt hyperproperties for formalising global specifications.

If we look more closely at our example of global robustness, we find that requiring the points in all L_p-balls to have the same output forces the network to produce the same output for all inputs. This means that we also have to consider more complicated output sets for global specifications. In this case, we either need to allow small changes in class scores (Example 2) or devise special rules for points close to the decision boundary (Example 3). Furthermore, if we express global robustness as Lipschitz continuity [7] (Example 4), our output set needs to be *input-dependent*. This means that it does not suffice to only compare network outputs with network outputs to determine whether a specification is violated. Instead, we also need to take the inputs that lead to the observed outputs into account.

For the reasons outlined above, we consider hyperproperties with multi-variable input sets and input-dependent output sets as in Definition 4 for formalising global specifications. To leverage existing neural network verification approaches for verifying these hyperproperties, we express the multi-variable input set and the input-dependent output set using auxiliary neural networks.

Definition 5 (Neural-Network-Defined Hyperproperty)**.** Let $n, m, v, w \in \mathbb{N}$. A *Neural-Network-Defined Hyperproperty (NNDH)* is a hyperproperty $\psi = (\mathcal{X}_\psi, \mathcal{Y}_\psi)$, where

$$\mathcal{X}_\psi = \{\mathrm{net}_{\mathrm{In}}(\mathbf{w}) \mid \mathbf{w} \in \mathcal{W}\}$$
$$\mathcal{Y}_\psi = \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(v)}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(v)} \,\middle|\, \mathrm{net}_{\mathrm{Sat}}\left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(v)}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(v)}\right) \geq 0\right\},$$

where $\mathcal{W} \subset \mathbb{R}^w$ is a bounded hyperrectangle and $\mathrm{net}_{\mathrm{In}}$ and $\mathrm{net}_{\mathrm{Sat}}$ are neural networks with

$$\mathrm{net}_{\mathrm{In}} : \mathbb{R}^w \to (\mathbb{R}^n)^v \qquad \mathrm{net}_{\mathrm{Sat}} : \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{v \text{ times}} \times \underbrace{\mathbb{R}^m \times \cdots \times \mathbb{R}^m}_{v \text{ times}} \to \mathbb{R}.$$

We can think of the neural network $\mathrm{net}_{\mathrm{In}}$ as generating the multi-variable input set from a single-variable hyperrectangular input space. The neural network $\mathrm{net}_{\mathrm{Sat}}$ serves as a *satisfaction function* [4] for the output set. A satisfaction function is non-negative if and only if an output — or, in this case, a tuple of inputs and outputs — lies within the output set of a property or hyperproperty.

It is central to Definition 5 that $\mathrm{net}_{\mathrm{In}}$ and $\mathrm{net}_{\mathrm{Sat}}$ do not *approximate* our desired input and output set, but express them *exactly*. Usually, we train neural networks to approximate a potentially unknown relationship between inputs and outputs. The neural networks $\mathrm{net}_{\mathrm{In}}$ and $\mathrm{net}_{\mathrm{Sat}}$, however, are not trained but carefully constructed to generate our desired input and output set. As such, these auxiliary neural networks are relatively simple structures in this paper. Their main purpose is to make hyperproperties accessible for existing neural network verification approaches.

We now provide several concrete examples of NNDHs including concrete $\mathrm{net}_{\mathrm{In}}$ and $\mathrm{net}_{\mathrm{Sat}}$ networks. We formalise global monotonicity, two notions of global robustness [21, 24], Lipschitz continuity, and dependency fairness [15, 35] as NNDHs. Afterwards, we show how NNDHs can be verified using existing neural network verifiers that can handle general computational graphs.

In the following, let $\mathcal{X} \subset \mathbb{R}^n$ be the bounded hyperrectangular input domain of the neural network under consideration. This domain is determined by the target application. In the case of image classification, for example, $\mathcal{X}$ would be the (normalised) pixel space.

Example 1 (Global Monotonicity)**.** Monotonicity is a desired behaviour of a neural network in applications from medicine to aviation [33]. Here, we formalise that the output $j \in \{1, \dots, m\}$ may *not increase* when input $i \in \{1, \dots, n\}$ increases. Non-decreasing monotonicity can be formalised analogously. We formalise global monotonicity as a hyperproperty $\psi_M = (\mathcal{X}_{\psi_M}, \mathcal{Y}_{\psi_M})$, where the input set $\mathcal{X}_{\psi_M} \subseteq \mathcal{X} \times \mathcal{X}$ and output set $\mathcal{Y}_{\psi_M} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$ are

$$\begin{aligned}\mathcal{X}_{\psi_M} &= \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \;\middle|\; \mathbf{x}^{(2)}_i \geq \mathbf{x}^{(1)}_i \right\}\\ \mathcal{Y}_{\psi_M} &= \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)} \;\middle|\; \mathbf{y}^{(2)}_j \leq \mathbf{y}^{(1)}_j \right\}.\end{aligned}$$

To generate these sets using neural networks to obtain an NNDH, we define

$$\mathcal{W}_M = \left\{ \mathbf{x}^{(1)}_1, \dots, \mathbf{x}^{(1)}_n, \mathbf{x}^{(2)}_1, \dots, \mathbf{x}^{(2)}_n \;\middle|\; \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in \mathcal{X} \right\},$$

$$\begin{aligned}\mathrm{net}_{\mathrm{In}_M}\left(\mathbf{x}^{(1)}_1, \dots, \mathbf{x}^{(1)}_n, \mathbf{x}^{(2)}_1, \dots, \mathbf{x}^{(2)}_n\right) &= \left(\mathbf{x}'^{(1)}, \mathbf{x}'^{(2)}\right),\\ \text{where } \mathbf{x}'^{(1)} &= \left(\mathbf{x}^{(1)}_1, \dots, \min\left(\mathbf{x}^{(1)}_i, \mathbf{x}^{(2)}_i\right), \dots, \mathbf{x}^{(1)}_n\right)\\ \mathbf{x}'^{(2)} &= \left(\mathbf{x}^{(2)}_1, \dots, \max\left(\mathbf{x}^{(1)}_i, \mathbf{x}^{(2)}_i\right), \dots, \mathbf{x}^{(2)}_n\right),\end{aligned}$$

and

$$\mathrm{net}_{\mathrm{Sat}_M}\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}\right) = \mathbf{y}^{(1)}_j - \mathbf{y}^{(2)}_j.$$

The function $\mathrm{net}_{\mathrm{Sat}_M}$ is a neural network with a single affine layer. Concerning $\mathrm{net}_{\mathrm{In}_M}$, we can compute max either using a maxpooling layer or by leveraging $\forall a, b \in \mathbb{R} : \max(a, b) = [a - b]^+ + b$ where $[\bullet]^+ = \max(\bullet, 0)$ is ReLU. Furthermore, since $\forall a, b \in \mathbb{R} : \min(a, b) = -\max(-a, -b)$, we can also compute min in a neural network. Therefore, $\mathcal{W}_M$, $\mathrm{net}_{\mathrm{In}_M}$ and $\mathrm{net}_{\mathrm{Sat}_M}$ together form an NNDH having $\mathcal{X}_{\psi_M}$ as its input set and $\mathcal{Y}_{\psi_M}$ as its output set.

Example 2 (Global L_∞ Robustness following [21])**.** Neural networks are susceptible to adversarial attacks where slightly modifying the input allows an attacker to control the output produced by a neural network [34]. This is a safety concern, for example, for traffic sign recognition [12] and biometric authentication using face recognition [31]. In this example, we express L_∞ global robustness according to [21] as an NNDH. This specification limits how much the output of a neural network may change for inputs that lie within an L_∞-ball of a certain size. Let $\delta, \varepsilon \in \mathbb{R}_{>0}$ be the radius of the L_∞-ball and the permitted magnitude of change, respectively. Let

$$\begin{aligned} \mathcal{W}_R &= \{\mathbf{x}_1, \ldots, \mathbf{x}_n, \boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_n \mid \mathbf{x} \in \mathcal{X}, \boldsymbol{\tau} \in [-\delta, \delta]^n\} \\ \mathrm{net}_{\mathrm{In}_R}(\mathbf{x}_1, \ldots, \mathbf{x}_n, \boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_n) &= (\mathbf{x}, \mathrm{project}_{\mathcal{X}}(\mathbf{x} + \boldsymbol{\tau})) \\ \mathrm{net}_{\mathrm{Sat}_{R1}}\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}\right) &= \varepsilon - \left\| \mathbf{y}^{(1)} - \mathbf{y}^{(2)} \right\|_\infty = \varepsilon - \max_{j=1}^{m} \left| \mathbf{y}^{(1)}_j - \mathbf{y}^{(2)}_j \right|, \end{aligned}$$

where $\mathrm{project}_{\mathcal{X}}$ computes the projection into the hyperrectangle $\mathcal{X}$. Projecting a point $\mathbf{x}$ into a hyperrectangle corresponds to computing the minimum between each coordinate and the lower boundary of the hyperrectangle and the maximum between each coordinate and the upper boundary of the hyperrectangle. As we show in Example 1, we can compute minima and maxima in a neural network. Similarly, $\mathrm{net}_{\mathrm{Sat}_{R1}}$ computes a maximum and absolute values, which we can compute by leveraging $\forall a \in \mathbb{R} : |a| = \max(a, -a)$. Overall, $\mathcal{W}_R$, $\mathrm{net}_{\mathrm{In}_R}$, and $\mathrm{net}_{\mathrm{Sat}_{R1}}$ define an NNDH $\psi_{R1} = (\mathcal{X}_{\psi_R}, \mathcal{Y}_{\psi_{R1}})$, where $\mathcal{X}_{\psi_R} \subset \mathcal{X} \times \mathcal{X}$ and $\mathcal{Y}_{\psi_{R1}} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$, with

$$\begin{aligned} \mathcal{X}_{\psi_R} &= \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \;\middle|\; \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(2)} \right\|_\infty \leq \delta \right\} \\ \mathcal{Y}_{\psi_{R1}} &= \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)} \;\middle|\; \left\| \mathbf{y}^{(1)} - \mathbf{y}^{(2)} \right\|_\infty \leq \varepsilon \right\}. \end{aligned}$$

This captures that a network is globally robust as defined in [21].

Example 3 (Global L_∞ Robustness following [24])**.** We also present an alternative definition of global robustness using an extra class representing non-robustness at an input point [24]. This definition may be more desirable in some applications, as it still permits non-robustness for noise-only *rubbish class* inputs [17] that lie off the data manifold. Let $\delta \in \mathbb{R}_{>0}$ be as in Example 2. Assume the classifier network we are studying produces an additional output $\perp = m + 1$ that shall indicate non-robustness. We reuse $\mathcal{X}_{\psi_R}$ from Example 2 and define $\psi_{R2} = (\mathcal{X}_{\psi_R}, \mathcal{Y}_{\psi_{R2}})$, where $\mathcal{Y}_{\psi_{R2}} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^{m+1} \times \mathbb{R}^{m+1}$ and, concretely,

$$\mathcal{Y}_{\psi_{R2}} = \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(1)} \;\middle|\; \mathit{NR}\left(\mathbf{y}^{(1)}\right) \vee \mathit{NR}\left(\mathbf{y}^{(2)}\right) \vee \mathit{Same}\left(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}\right) \right\},$$

where

$$\begin{aligned} \mathit{NR}(\mathbf{y}) &= \bigwedge_{j=1}^{m} \mathbf{y}^{(k)}_\perp \geq \mathbf{y}^{(k)}_j \\ \mathit{Same}\left(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}\right) &= \bigvee_{j_1=1}^{m} \bigwedge_{k=1}^{2} \bigwedge_{j_2=1}^{m} \mathbf{y}^{(k)}_{j_1} \geq \mathbf{y}^{(k)}_{j_2}. \end{aligned}$$

Intuitively, *NR* captures that the extra class $\perp$ is assigned to an input, while *Same* captures that the same class is assigned to $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$[1]. To construct a neural network $\mathrm{net}_{\mathrm{Sat}_{R2}}$ that

[1] Strictly speaking, *Same* only requires that there is an intersection between the largest elements of $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$. This comes into play when the assigned class is ambiguous due to an output having several largest elements.

serves as a satisfaction function for ψ_{R2}, we note that for an arbitrary vector $\mathbf{u} \in \mathbb{R}^u$, $u \in \mathbb{N}$

$$\bigvee_{a \in \mathcal{A}} \bigwedge_{b \in B(a)} \mathbf{u}_{k_1(a,b)} \geq \mathbf{u}_{k_2(a,b)} \tag{1}$$

$$\Leftrightarrow \left(\max_{a \in \mathcal{A}} \min_{b \in B(a)} \mathbf{u}_{k_1(a,b)} - \mathbf{u}_{k_2(a,b)} \right) \geq 0, \tag{2}$$

where $\mathcal{A}$ and $\mathcal{B}$ are finite sets, $B : \mathcal{A} \to 2^{\mathcal{B}}$, and $k_1, k_2 : \mathcal{A} \times \mathcal{B} \to \mathbb{N}$. As we can transform any formula in propositional logic into Disjunctive Normal Form, we can bring the formula defining $\mathcal{Y}_{\psi_{R2}}$ into the form of Equation (1). Therefore, since we can compute min and max using a neural network (Example 1), we can define a neural network $\mathrm{net}_{\mathrm{Sat}_{R2}}$ serving as a satisfaction function for ψ_{R2}. Together with $\mathcal{W}$ and $\mathrm{net}_{\mathrm{In}_R}$ from Example 2, $\mathrm{net}_{\mathrm{Sat}_{R2}}$ defines an NNDH with the same input and output set as ψ_{R2}.

Example 4 (Lipschitz Continuity)**.** The Lipschitz continuity of a neural network is linked not only to robustness [34] but also to fairness [10], generalisation [3], and explainability [13]. While many neural network architectures are always Lipschitz continuous [7, 34, 29], it is the magnitude of the Lipschitz constant that matters [7]. Let $K \in \mathbb{R}_{\geq 0}$ be the desired global Lipschitz constant. Define $\mathcal{W}_C = \left\{ \mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_n^{(1)}, \mathbf{x}_1^{(2)}, \ldots, \mathbf{x}_n^{(2)} \;\middle|\; \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in \mathcal{X} \right\}$ and

$$\mathrm{net}_{\mathrm{In}_C}\left(\mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_n^{(1)}, \mathbf{x}_1^{(2)}, \ldots, \mathbf{x}_n^{(2)}\right) = \left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}\right)$$
$$\mathrm{net}_{\mathrm{Sat}_C}\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}\right) = K \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(2)} \right\|_\infty - \left\| \mathbf{y}^{(1)} - \mathbf{y}^{(2)} \right\|_\infty .$$

First, $\mathrm{net}_{\mathrm{In}_C}$ is an identity function and, thus, a trivial neural network. Then, by computing $\| \bullet \|_\infty$ as in Example 2 in a neural network, we obtain an NNDH $\psi_C = (\mathcal{X}_{\psi_C}, \mathcal{Y}_{\psi_C})$ with

$$\mathcal{X}_{\psi_C} = \mathcal{X} \times \mathcal{X}$$
$$\mathcal{Y}_{\psi_C} = \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)} \;\middle|\; \left\| \mathbf{y}^{(1)} - \mathbf{y}^{(2)} \right\|_\infty \leq K \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(2)} \right\|_\infty \right\},$$

which corresponds to Lipschitz continuity with Lipschitz constant K.

Example 5 (Dependency Fairness)**.** Machine learning applications from automated hiring [6] to image classification [28] bear the danger of producing unfair machine-learning models. However, in some applications, ensuring fairness may be legally required [27]. One fairness requirement that we may pose is that "similar individuals are treated similarly" [10]. *Dependency fairness* [15, 35] is a fairness criterion based on this idea[2]. Assume the first dimension of the input space is a categorical sensitive attribute with $A \in \mathbb{N}$ disjoint values. We consider two inputs to be *similar* if they are equal except for the sensitive attribute. Dependency fairness specifies that all similar inputs are assigned to the same class. Let $\psi_F = (\mathcal{X}_{\psi_F}, \mathcal{Y}_{\psi_F})$,

[2]We believe dependency fairness is an overly simplistic fairness criterion as it can be trivially satisfied by withholding sensitive attributes from the neural network, which is known to be insufficient for real-world fairness [2]. However, we still think that dependency fairness is suitable as an example for experimenting with verifying global specifications.

with $\mathcal{X}_{\psi_F} \subset \mathcal{X}^A$, $\mathcal{Y}_{\psi_F} \subset (\mathbb{R}^n)^A \times (\mathbb{R}^m)^A$, where

$$\mathcal{X}_{\psi_F} = \left\{ \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(A)} \;\middle|\; \begin{array}{l} \forall k \in \{1, \dots, A\} : \\ \quad \left(\mathbf{x}^{(k)}_1 = k \wedge \forall i \in \{2, \dots, n\} : \mathbf{x}^{(1)}_i = \mathbf{x}^{(k)}_i \right) \end{array} \right\}$$

$$\mathcal{Y}_{\psi_F} = \left\{ \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(A)}, \mathbf{y}^{(1)}, \dots, \mathbf{y}^{(A)} \;\middle|\; \bigvee_{j_1=1}^{m} \bigwedge_{k=1}^{A} \bigwedge_{j_2=1}^{m} \mathbf{y}^{(k)}_{j_1} \geq \mathbf{y}^{(k)}_{j_2} \right\}.$$

We can construct a neural network satisfaction function $\mathrm{net}_{\mathrm{Sat}_F}$ for this property analogously to Example 3. The input set $\mathcal{X}_{\psi_F}$ consists of tuples of similar inputs which contain each value of the sensitive attribute in a fixed order. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be the diagonal matrix with $0, 1, \dots, 1$ on its diagonal. Let $\mathrm{assign} : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ be an affine function with $\mathrm{assign}(k, \mathbf{x}) = \mathbf{A}\mathbf{x} + (k, 0, \dots, 0)^T$. Define $\mathcal{W} = \mathcal{X}$ and $\mathrm{net}_{\mathrm{In}_F}(\mathbf{x}) = (\mathrm{assign}(1, \mathbf{x}), \dots, \mathrm{assign}(A, \mathbf{x}))$. Since assign is affine, $\mathrm{net}_{\mathrm{In}_F}$ is a neural network. Overall, $\mathcal{W}$, $\mathrm{net}_{\mathrm{In}_F}$, and $\mathrm{net}_{\mathrm{Sat}_F}$ define an NNDH with the same input and output set as ψ_F.

These examples demonstrate that Definition 5 is an expressive specification formalism, despite restricting input and output sets to be defined by neural networks. It remains to show that we can indeed verify NNDHs using existing neural network verification approaches. This builds upon the ability to verify general computational graphs. In [37], the Linear Relaxation-based Perturbation Analysis (LiRPA) framework is extended to general computational graphs. LiRPA underlies verifiers such as α,β-CROWN [38] and ERAN [32], and is used in Marabou [22] and MN-BaB [14], among others. Among these verifiers, α,β-CROWN already supports verifying general computational graphs.

The central idea in verifying an NNDH ψ is to compose the network to verify net_θ with itself and the networks $\mathrm{net}_{\mathrm{In}_\psi}$ and $\mathrm{net}_{\mathrm{Sat}_\psi}$ that define the input and output set of ψ.

Theorem 1 (NNDH Verification)**.** *Let $\psi = (\mathcal{X}_\psi, \mathcal{Y}_\psi)$ with $\mathcal{W} \subseteq \mathbb{R}^w$, $\mathrm{net}_{\mathrm{In}} : \mathbb{R}^w \to (\mathbb{R}^n)^v$ and $\mathrm{net}_{\mathrm{Sat}} : (\mathbb{R}^m)^v \to \mathbb{R}$ be an NNDH. Let $\mathrm{net}_\theta : \mathbb{R}^n \to \mathbb{R}^m$ be a neural network. Define $\mathrm{net}'_\theta : \mathbb{R}^w \to \mathbb{R}$ as*

$$\mathrm{net}'_\theta(\mathbf{w}) = \mathrm{net}_{\mathrm{Sat}}\left(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(v)}, \mathrm{net}_\theta\left(\mathbf{x}^{(1)}\right), \dots, \mathrm{net}_\theta\left(\mathbf{x}^{(v)}\right) \right)$$
$$\text{where } \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(v)} = \mathrm{net}_{\mathrm{In}}(\mathbf{w}).$$

Further, let $\varphi = (\mathcal{W}, \mathbb{R}_{\geq 0})$. It holds that $\mathrm{net}'_\theta \vDash \varphi \Leftrightarrow \mathrm{net}_\theta \vDash \psi$.

Proof. Theorem 1 follows from applying Definitions 3 and 5. □

Figure 2 visualises net'_θ from Theorem 1. We construct a new computational graph by generating several inputs using $\mathrm{net}_{\mathrm{In}}$ and feeding each input to a separate copy of net_θ. Finally, $\mathrm{net}_{\mathrm{Sat}}$ takes the generated inputs and the output of each copy of net_θ and computes the satisfaction function value. Considering several copies of the same artefact is known as *self-composition* [8]. As Theorem 1 shows, verifying an NNDH ψ corresponds to verifying a property φ of the new computational graph net'_θ. Overall, net'_θ has a more complicated graph structure than net_θ, but it only contains computations that also appear in net_θ, $\mathrm{net}_{\mathrm{In}}$ or $\mathrm{net}_{\mathrm{Sat}}$. Therefore, ψ can be verified using verifiers that can verify net_θ, $\mathrm{net}_{\mathrm{In}}$ and $\mathrm{net}_{\mathrm{Sat}}$ and support general computational graphs.

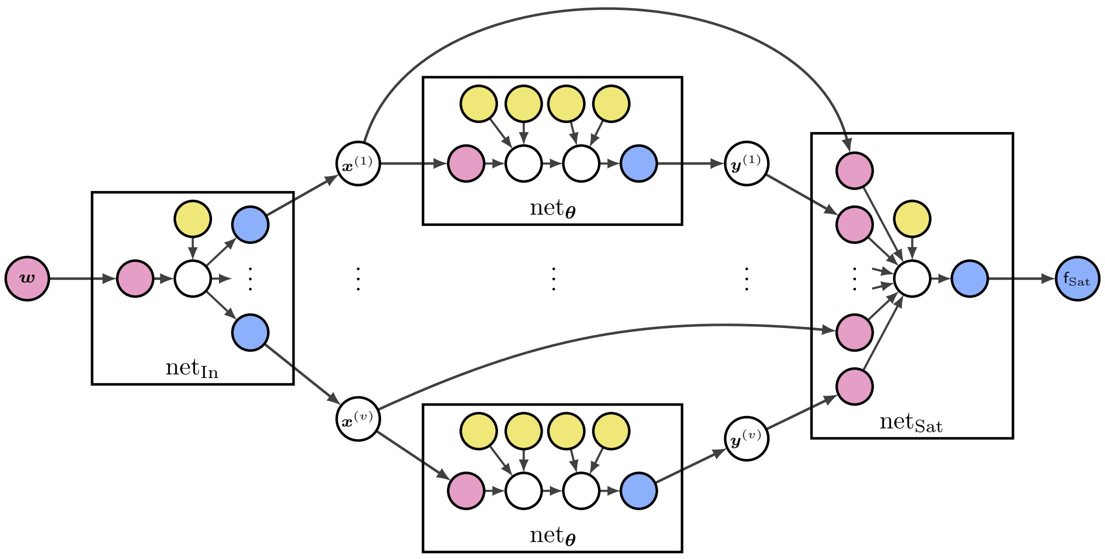

Figure 2: **Computational Graph for Verifying NNDHs.** Verifying an NNDH (Definition 5) reduces to verifying an input-output property of the computational graph in this figure. The boxes enclose sub-graphs of the computational graph. The contents of each box are placeholders. Pink nodes ■ represent inputs, yellow nodes ■ represent parameters, and blue nodes ■ represent outputs. The input and output nodes in each sub-graph are repetitions of their direct predecessors or direct successors outside of the subgraph. The inputs of net_{Sat} were rearranged for better legibility.

4 Related Work

Using self-composition for verifying specific global specifications was explored previously [21, 23]. We use self-composition for verifying a range of global specifications. Improved encodings of self-composition [36] and approaches from differential verification of neural networks [26] are interesting directions for improving the verification of NNDHs.

In 2017, verifying global robustness was found to be infeasible using the then-available verifiers [21]. Recent approaches to global robustness [36] and global fairness specifications [35] have demonstrated that verifying global specifications is feasible today. The reason behind this could be that practically, neural networks appear not to realise their full combinatorial potential [18], in a way that allows for efficient branch-and-bound verification [35].

5 Conclusion

We present a versatile formalism for expressing global specifications while maintaining compatibility with existing verification approaches. Evaluating this approach empirically remains future work. A promising verifier for this approach is α,β-CROWN [38], as it already supports verifying arbitrary computational graphs. An interesting direction is comparing our generally applicable approach with approaches specialised to individual global specifications, such as, global robustness [36], dependency fairness [35] and Lipschitz continuity [5].

References

[1] Stanley Bak, Changliu Liu, and Taylor T. Johnson. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. *CoRR*, abs/2109.00498, 2021.

[2] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning.* fairmlbook.org, 2019.

[3] Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *NIPS*, pages 6240–6249, 2017.

[4] Fabian Bauer-Marquart, David Boetius, Stefan Leue, and Christian Schilling. SpecRepair: Counter-Example Guided Safety Repair of Deep Neural Networks. In Owolabi Legunsen and Grigore Rosu, editors, *SPIN*, volume 13255 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2022.

[5] Aritra Bhowmick, Meenakshi D'Souza, and G. Srinivasa Raghavan. LipBaB: Computing Exact Lipschitz Constant of ReLU Networks. In Igor Farkas, Paolo Masulli, Sebastian Otte, and Stefan Wermter, editors, *ICANN (4)*, volume 12894 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2021.

[6] Miranda Bogen and Aaron Rieke. Help wanted: An Examination of Hiring Algorithms, Equity, and Bias. Report, Upturn, 2018.

[7] Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier. Parseval Networks: Improving Robustness to Adversarial Examples. In Doina Precup and Yee Whye Teh, editors, *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR, 2017.

[8] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *CSF*, pages 51–65. IEEE Computer Society, 2008.

[9] Yogesh K. Dwivedi, Nir Kshetri, Laurie Hughes, Emma Louise Slade, Anand Jeyaraj, Arpan Kumar Kar, Abdullah M. Baabdullah, Alex Koohang, Vishnupriya Raghavan, Manju Ahuja, Hanaa Albanna, Mousa Ahmad Albashrawi, Adil S. Al-Busaidi, Janarthanan Balakrishnan, Yves Barlette, Sriparna Basu, Indranil Bose, Laurence Brooks, Dimitrios Buhalis, Lemuria Carter, Soumyadeb Chowdhury, Tom Crick, Scott W. Cunningham, Gareth H. Davies, Robert M. Davison, Rahul Dé, Denis Dennehy, Yanqing Duan, Rameshwar Dubey, Rohita Dwivedi, John S. Edwards, Carlos Flavián, Robin Gauld, Varun Grover, Mei-Chih Hu, Marijn Janssen, Paul Jones, Iris Junglas, Sangeeta Khorana, Sascha Kraus, Kai R. Larsen, Paul Latreille, Sven Laumer, F. Tegwen Malik, Abbas Mardani, Marcello Mariani, Sunil Mithas, Emmanuel Mogaji, Jeretta Horn Nord, Siobhan O'Connor, Fevzi Okumus, Margherita Pagani, Neeraj Pandey, Savvas Papagiannidis, Ilias O. Pappas, Nishith Pathak, Jan Pries-Heje, Ramakrishnan Raman, Nripendra P. Rana, Sven-Volker Rehm, Samuel Ribeiro-Navarrete, Alexander Richter, Frantz Rowe, Suprateek Sarker, Bernd Carsten Stahl, Manoj Kumar Tiwari, Wil van der Aalst, Viswanath Venkatesh, Giampaolo Viglia, Michael Wade, Paul Walton, Jochen Wirtz, and Ryan Wright. "So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational ai for research, practice and policy. *Int. J. Inf. Manag.*, 71:102642, 2023.

[10] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. In Shafi Goldwasser, editor, *ITCS*, pages 214–226. ACM, 2012.

[11] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. GPTs are GPTs: An early look at the labor market impact potential of large language models. *CoRR*, abs/2303.10130, 2023.

[12] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634. Computer Vision Foundation / IEEE Computer Society, 2018.

[13] Thomas Fel, David Vigouroux, Rémi Cadène, and Thomas Serre. How good is your explanation? algorithmic stability measures to assess the quality of explanations for deep neural networks. In

WACV, pages 1565–1575. IEEE, 2022.

[14] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *ICLR*. OpenReview.net, 2022.

[15] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: testing software for discrimination. In Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman, editors, *ESEC/SIGSOFT FSE*, pages 498–510. ACM, 2017.

[16] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning.* Adaptive computation and machine learning. MIT Press, 2016.

[17] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In Yoshua Bengio and Yann LeCun, editors, *ICLR (Poster)*, 2015.

[18] Boris Hanin and David Rolnick. Deep ReLU Networks Have Surprisingly Few Activation Patterns. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS*, pages 359–368, 2019.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV (4)*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016.

[20] Patrick Henriksen and Alessio Lomuscio. DEEPSPLIT: An Efficient Splitting Method for Neural Network Verification via Indirect Effect Analysis. In Zhi-Hua Zhou, editor, *IJCAI*, pages 2549–2555. ijcai.org, 2021.

[21] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Towards Proving the Adversarial Robustness of Deep Neural Networks. In Lukas Bulwahn, Maryam Kamali, and Sven Linker, editors, *FVAV@iFM*, volume 257 of *EPTCS*, pages 19–26, 2017.

[22] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

[23] Haitham Khedr and Yasser Shoukry. CertiFair: A Framework for Certified Global Fairness of Neural Networks. *CoRR*, abs/2205.09927, 2022.

[24] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-Robust Neural Networks. In Marina Meila and Tong Zhang, editors, *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 6212–6222. PMLR, 2021.

[25] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. *CoRR*, abs/2212.10376, 2022.

[26] Brandon Paulsen, Jingbo Wang, and Chao Wang. ReluDiff: differential verification of deep neural networks. In Gregg Rothermel and Doo-Hwan Bae, editors, *ICSE*, pages 714–726. ACM, 2020.

[27] Dino Pedreschi, Salvatore Ruggieri, and Franco Turini. Discrimination-aware data mining. In Ying Li, Bing Liu, and Sunita Sarawagi, editors, *KDD*, pages 560–568. ACM, 2008.

[28] Vinay Uday Prabhu and Abeba Birhane. Large image datasets: A pyrrhic win for computer vision? *CoRR*, abs/2006.16923, 2020.

[29] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability Analysis of Deep Neural Networks with Provable Guarantees. In *IJCAI*, pages 2651–2659. ijcai.org, 2018.

[30] Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. Formal Specification for Deep Neural Networks. In Shuvendu K. Lahiri and Chao Wang, editors, *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2018.

[31] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In Edgar R. Weippl, Stefan Katzen-

beisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *CCS*, pages 1528–1540. ACM, 2016.

[32] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An Abstract Domain for Certifying Neural Networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019.

[33] Aishwarya Sivaraman, Golnoosh Farnadi, Todd D. Millstein, and Guy Van den Broeck. Counterexample-Guided Learning of Monotonic Neural Networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020.

[34] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *ICLR (Poster)*, 2014.

[35] Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. Perfectly parallel fairness certification of neural networks. *Proc. ACM Program. Lang.*, 4(OOPSLA):185:1–185:30, 2020.

[36] Zhilu Wang, Chao Huang, and Qi Zhu. Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding. *CoRR*, abs/2203.14141, 2022.

[37] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020.

[38] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *NeurIPS*, 2022.