

# Probabilistic Verification of Neural Networks using Branch and Bound

David Boetius <sup>1</sup>, Stefan Leue <sup>1</sup>, and Tobias Sutter <sup>1</sup>

<sup>1</sup>University of Konstanz

[david.boetius](mailto:david.boetius@uni-konstanz.de), [stefan.leue](mailto:stefan.leue@uni-konstanz.de), [tobias.sutter](mailto:tobias.sutter@uni-konstanz.de)@uni-konstanz.de

31st January 2025

## Abstract

Probabilistic verification of neural networks is concerned with formally analysing the output distribution of a neural network under a probability distribution of the inputs. Examples of probabilistic verification include verifying the demographic parity fairness notion or quantifying the safety of a neural network. We present a new algorithm for the probabilistic verification of neural networks based on an algorithm for computing and iteratively refining lower and upper bounds on probabilities over the outputs of a neural network. By applying state-of-the-art bound propagation and branch and bound techniques from non-probabilistic neural network verification, our algorithm significantly outpaces existing probabilistic verification algorithms, reducing solving times for various benchmarks from the literature from tens of minutes to tens of seconds. Furthermore, our algorithm compares favourably even to dedicated algorithms for restricted subsets of probabilistic verification. We complement our empirical evaluation with a theoretical analysis, proving that our algorithm is sound and, under mildly restrictive conditions, also complete when using a suitable set of heuristics.

## 1 Introduction

As deep learning spreads through society, it becomes increasingly important to ensure the reliability of artificial neural networks, including aspects of fairness and safety. However, manually inspecting neural networks is infeasible due to their non-transparent nature. Furthermore, empirical assessments of neural networks are challenged by neural networks being fragile with respect to various types of input perturbations [14, 24, 31, 35–37, 61]. In contrast, neural network verification analyses neural networks with mathematical rigour, facilitating the faithful auditing of neural networks.

In this paper, we consider probabilistic verification of neural networks, which is concerned with proving statements about the output distribution of a neural network given a distribution of the inputs. An example of probabilistic verification is proving that a neural network net making a binary decision affecting a person (for example, hire/do not hire, credit approved/denied) satisfies the demographic parity fairness notion [8] under a probability distribution  $\mathbb{P}_{\mathbf{x}}$  of the network inputs  $\mathbf{x}$  representing the person

$$\frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is disadvantaged}]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is advantaged}]} \geq \gamma, \quad (1)$$

where ‘ $\mathbf{x}$  is disadvantaged’ could, for example, refer to a person not being male and  $\gamma \in [0, 1]$  with  $\gamma = 0.8$  being a common choice [33]. A closely related problem to probabilistic verification is

computing bounds on probabilities over a neural network. An example of this is quantifying the safety of a neural network by bounding

$$\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) \text{ is unsafe}]. \quad (2)$$

In this paper, we introduce a novel algorithm for computing bounds on probabilities such as Equation (2) using a branch and bound framework [20, 42, 51]. These bounds allow us to verify probabilistic statements like Equation (1) using bound propagation [4, 49]. Our algorithm ProbabilisticVerification (PV) is a fast and generally applicable probabilistic verification algorithm for neural networks based on massively parallel branch and bound neural network verification [75] using linear relaxations of neural networks [60, 78]. Our theoretical analysis shows that PV is sound and, under mildly restrictive conditions, complete when using suitable branching and splitting heuristics.

Our experimental evaluation reveals that PV significantly outpaces the probabilistic verification algorithms FairSquare [4] and SpaceScanner [26]. In particular, PV solves benchmark instances that FairSquare can not solve within 15 minutes in less than one minute and solves the ACAS Xu [39] probabilistic robustness case study of Converse et al. [26] in a mean runtime of 21 seconds, compared to 33 minutes for SpaceScanner.

Applying PV to #DNN verification [45], a subset of probabilistic verification, reveals that PV also compares favourably to the ProVe\_SLR algorithm [47] specialised to #DNN verification and even the  $\varepsilon$ -ProVe algorithm [46] that relaxes #DNN verification to computing a confidence interval on the solution. In contrast to  $\varepsilon$ -ProVe and similar approaches [7, 9, 68], PV computes lower and upper bounds on probabilities like Equation (2) that are guaranteed to hold with absolute certainty. Such bounds are preferable to confidence intervals in high-risk machine-learning applications.

To test the limits of PV and fuel further research in probabilistic verification, we introduce a significantly more challenging probabilistic verification benchmark: The MiniACSIncome benchmark is based on the ACSIncome dataset [27] and is concerned with verifying the demographic parity of neural networks for datasets of increasing input dimensionality. In summary, our contributions are

- the PV algorithm for the probabilistic verification of neural networks,
- a theoretical analysis of PV proving soundness and completeness,
- a thorough experimental comparison of PV with existing probabilistic verifiers for neural networks and tools dedicated to restricted subsets of probabilistic verification, and
- MiniACSIncome: a new, challenging probabilistic verification benchmark.

## 2 Related Work

This section reviews related work from non-probabilistic neural network verification and probabilistic neural network verification, as well as verification of Bayesian Neural Networks (BNNs) and non-probabilistic fairness verification.

### 2.1 Non-Probabilistic Neural Network Verification

Non-probabilistic neural network verification is concerned with proving that the outputs of a neural network satisfy a given condition for all inputs in an input set. Approaches for non-probabilistic neural network verification include Satisfiability Modulo Theories (SMT) solving [29, 39, 73], Mixed Integer Linear Programming (MILP) [5, 25, 62], and reachability analysis [6, 63, 64]. Many of these approaches can be understood as branch and bound algorithms [20]. Branch and bound [42, 51]

also powers the  $\alpha, \beta$ -CROWN [79], PyRAT [43] and Marabou [73] verifiers that lead the table in recent international neural network verifier competitions [18, 19].

A critical component of a branch and bound verification algorithm is computing bounds on the output of a neural network. Approaches for bounding neural network outputs include interval arithmetic [25, 57], dual approaches [72], linear bound propagation techniques [60, 69, 78], multi-neuron linear relaxations [52], and further optimisation-based approaches [21, 54, 75].

## 2.2 Probabilistic Neural Network Verification

Probabilistic verification algorithms can be divided into *sound* algorithms that provide valid proofs and *probably sound* algorithms that provide valid proofs only with a certain predefined probability. Fairness verification is a subset of probabilistic verification that studies problems such as Equation (1). Another subset of probabilistic verification is #DNN verification [45], corresponding to probabilistic verification under uniformly distributed inputs.

By sacrificing soundness, probably sound verification algorithms [7, 9, 45, 46] obtain efficiency. One example is  $\varepsilon$ -ProVe [46], a probably sound #DNN verification algorithm. Bastani et al. [9], Converse et al. [26] and Marzari et al. [47] compare sound and probably sound approaches for fairness verification, general probabilistic verification, and #DNN verification, respectively. We study sound probabilistic verification since certainly sound results are preferable in critical applications, such as education [32], medical applications, or autonomous driving and flight.

Concerning sound verification approaches, FairSquare [4] is a sound fairness verification algorithm that partitions the input space into disjoint hyperrectangles. Integrating the input probability distribution over these hyperrectangles then yields bounds on a target probability. To improve these bounds, FairSquare iteratively refines the input space partitioning using SMT solving. ProVe\_SLR [47] is a sound #DNN verifier based on a massively parallel branch and bound algorithm. To perform general sound probabilistic verification, Converse et al. [26] (SpaceScanner) and Borca-Tasciuc et al. [17] divide the input space into disjoint polytopes using concolic execution and reachable set verification, respectively. Morettin et al. [50] use weighted model integration [12] to obtain a general sound probabilistic verification algorithm but neither provide code nor report runtimes.

From the above approaches, FairSquare and ProVe\_SLR are most closely related to our algorithm PV. However, PV is more general than FairSquare and ProVe\_SLR, which are restricted to fairness verification and #DNN verification, respectively. Like FairSquare, PV uses iteratively refined bounds on probabilities to verify probabilistic statements like Equation (1) using bound propagation. However, while FairSquare uses expensive SMT solving for refining the input space, we use a branch and bound algorithm utilising computationally inexpensive input splitting and bound propagation techniques from non-probabilistic neural network verification for refining the input splitting. ProVe\_SLR also builds on these techniques but computes a probability exactly instead of iteratively refining bounds as PV does. These differences allow PV to significantly outpace both FairSquare and ProVe\_SLR, as we demonstrate in Section 6.

## 2.3 Verification of Bayesian Neural Networks

A related problem to probabilistic verification of neural networks is verifying Bayesian Neural Networks (BNNs) [1, 10, 13, 22, 23, 28, 70, 71]. In BNNs, the network parameters follow a probability distribution [53]. Therefore, BNN verification also requires reasoning about a probability distribution of neural network outputs. However, since neural networks typically have vastly more parameters than inputs, BNN verification treats much higher dimensional probability distributions than we consider in this paper. Our restriction to deterministic neural networks reduces the problem dimension, allowing us to provide a sound and complete yet practically scalable probabilistic verification algorithm.

## 2.4 Non-Probabilistic Fairness Verification

Besides probabilistic fairness notions, such as demographic parity [8], several approaches verify *dependency fairness* [34, 65] in a global [16, 48, 65] or local [59] setting. Dependency fairness is an individual fairness notion [30] that states that persons that only differ by their protected attribute (for example, gender or race) must be assigned to the same class. However, dependency fairness can be satisfied trivially by withholding the protected attribute from the classifier<sup>1</sup>. Since withholding the protected attribute is insufficient [56] or even harmful [41] for fairness, dependency fairness is an insufficient fairness notion.

## 3 Preliminaries and Problem Statement

Throughout this paper, we are concerned with computing (provable) lower and upper bounds.

**Definition 1** (Bounds). For  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , we call  $\ell, u \in \mathbb{R}^m$  a *lower*, respectively, *upper bound* on  $f$  for  $\mathcal{X}' \subseteq \mathbb{R}^n$  if  $\ell \leq f(\mathbf{x}) \leq u, \forall \mathbf{x} \in \mathcal{X}'$ .

**Neural Networks.** In particular, we are concerned with computing bounds on functions involving neural networks  $\text{net} : \mathcal{X} \rightarrow \mathbb{R}^m$ , where  $\mathcal{X} \subseteq \mathbb{R}^n$  is the input space of the neural network. A neural network is a composition of linear functions and a predefined set of non-linear functions, such as ReLU, Tanh, and max pooling. We assume  $\text{net}$  to be Lipschitz continuous, which is satisfied by many common architectures [58, 61]. Besides this, we refrain from defining neural networks further since our approach is not specific to any concrete architecture. We refer the interested reader to the `auto_LiRPA` library [74] that practically defines the class of neural networks to which our algorithm can be applied.

**Notation and Terminology.** We use bold letters ( $\mathbf{x}, \mathbf{y}$ ) for vectors and calligraphic letters ( $\mathcal{X}$ ) for sets. Let  $\underline{\mathbf{x}}, \bar{\mathbf{x}} \in \mathbb{R}^n$  with  $\underline{\mathbf{x}} \leq \bar{\mathbf{x}}$ . We use  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}] = \{\mathbf{x} \in \mathbb{R}^n \mid \underline{\mathbf{x}} \leq \mathbf{x} \leq \bar{\mathbf{x}}\}$  to denote the hyperrectangle with the minimal element  $\underline{\mathbf{x}}$  and the maximal element  $\bar{\mathbf{x}}$ . When we speak of *bounds* on a certain quantity in this paper, we refer to a pair of a lower and an upper bound. Throughout this paper,  $\ell$  and  $k$  denote lower bounds, while  $u$  denotes an upper bound. When convenient, we also use  $\underline{\mathbf{y}} \leq \mathbf{y} \leq \bar{\mathbf{y}}$  to denote bounds, especially on vectors. We assume that all random objects are defined on the same abstract probability space  $(\Omega, \mathcal{F}, \mathbb{P})$  and that all continuous random variables admit a probability density function.

### 3.1 Probabilistic Verification of Neural Networks

Let  $\mathcal{X} \subseteq \mathbb{R}^n$  be a (potentially unbounded) hyperrectangle and let  $v \in \mathbb{N}$ . In this paper, we are concerned with proving or disproving whether  $\text{net} : \mathcal{X} \rightarrow \mathbb{R}^m$  is feasible for the *probabilistic verification problem*

$$P : \begin{cases} f_{\text{Sat}}(p_1, \dots, p_v) \geq 0, \\ p_i = \mathbb{P}_{\mathbf{x}^{(i)}}[g_{\text{Sat}}^{(i)}(\mathbf{x}^{(i)}, \text{net}(\mathbf{x}^{(i)})) \geq 0] \quad \forall i \in \{1, \dots, v\}, \end{cases} \quad (3)$$

where  $\mathbf{x}^{(i)}, i \in \{1, \dots, v\}$ , is an  $\mathcal{X}$ -valued random variable with distribution  $\mathbb{P}_{\mathbf{x}^{(i)}}$  and  $f_{\text{Sat}} : \mathbb{R}^v \rightarrow \mathbb{R}, g_{\text{Sat}}^{(i)} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}, i \in \{1, \dots, v\}$  are *satisfaction functions*. We discuss handling non-hyperrectangular input spaces in Section 4.3. We only consider satisfaction functions that are compositions of linear functions, multiplication, division, and monotone functions, such as ReLU, Sigmoid, min and max. Throughout this paper, we assume all probabilistic verification problems to be well-defined.

<sup>1</sup>Alternatively, if dependency fairness should only hold for certain regions of the input space, the protected attribute can be set to a fixed value in these regions.

**Example 1.** We express the demographic parity fairness notion from Equation (1) as a probabilistic verification problem. Let  $\mathcal{X} \subseteq \mathbb{R}^n$  be an input space that encodes information about a person, including a categorical protected attribute, such as gender, race, or disability status that is one-hot encoded at the indices  $A \subset \{1, \dots, n\}$ . We assume a single historically advantaged category encoded at the index  $a \in A$ . Consider a neural network  $\text{net} : \mathbb{R}^n \rightarrow \mathbb{R}^2$  that acts as a binary classifier making a decision affecting a person, such as hiring or credit approval. The neural network produces a score for each class and assigns the class with the higher score to an input. We express Equation (1) as a probabilistic verification problem, that is,

$$\frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is disadvantaged}]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is advantaged}]} \geq \gamma \iff \text{net is feasible in Equation (3),}$$

where,  $f_{\text{Sat}}(p_1, p_2, p_3, p_4) = (p_1 p_4) / (p_2 p_3) - \gamma$ ,  $g_{\text{Sat}}^{(1)}(\mathbf{x}, \text{net}(\mathbf{x})) = \min(\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2, -\mathbf{x}_a)$ ,  $g_{\text{Sat}}^{(2)}(\mathbf{x}, \text{net}(\mathbf{x})) = -\mathbf{x}_a$ ,  $g_{\text{Sat}}^{(3)}(\mathbf{x}, \text{net}(\mathbf{x})) = \min(\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2, \mathbf{x}_a - 1)$ , and  $g_{\text{Sat}}^{(4)}(\mathbf{x}, \text{net}(\mathbf{x})) = \mathbf{x}_a - 1$ . Appendix A contains a detailed derivation of this equivalence.

If we consider a single probability  $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$  where  $\mathbf{x}$  is uniformly distributed, probabilistic verification corresponds to #DNN verification [45]. As Marzari et al. [45] prove, #DNN verification is #P complete, implying that probabilistic verification is #P hard. However, this does not determine which probabilistic verification problems are practically solvable.

### 3.2 Non-Probabilistic Neural Network Verification

Non-probabilistic neural network verification determines whether  $\text{net} : \mathcal{X} \rightarrow \mathbb{R}^m$  is feasible for

$$V : g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}', \quad (4)$$

where  $\mathcal{X}' \subseteq \mathcal{X}$  is a bounded hyperrectangle, and  $g_{\text{Sat}} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  is a *satisfaction function* that indicates whether the output of  $\text{net}$  is desirable ( $g_{\text{Sat}}(\cdot, \text{net}(\cdot)) \geq 0$ ) or undesirable ( $g_{\text{Sat}}(\cdot, \text{net}(\cdot)) < 0$ ). In (non-probabilistic) neural network verification,  $g_{\text{Sat}}$  can generally be considered a part of  $\text{net}$  [20, 74]. *Neural network verifiers* are algorithms for proving or disproving Equation (4). Two desirable properties of neural network verifiers are *soundness* and *completeness*.

**Definition 2** (Soundness and Completeness). A verification algorithm is *sound* if it only produces genuine counterexamples and valid proofs for Equation (4). It is *complete* if it produces a counterexample or proof for Equation (4) for any neural network in a finite amount of time.

Analogous notions of soundness and completeness also apply to probabilistic verifiers. Generally unsound but *probably sound* probabilistic verification approaches are discussed in Section 2. We introduce incomplete and complete neural network verification through two examples: interval arithmetic and branch and bound.

#### 3.2.1 Interval Arithmetic

Interval arithmetic [49] is a bound propagation technique that derives bounds on the output of a neural network from bounds on the network input. Other bound propagation approaches include DeepPoly [60] and CROWN [78]. Assume  $\underline{\mathbf{x}} \leq \mathbf{x} \leq \bar{\mathbf{x}}$  are bounds on the network input  $\mathbf{x}$  and we apply interval arithmetic to compute  $\ell \leq g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \leq u$ ,  $\forall \mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ . If the lower bound is large enough or the upper bound small enough, we can prove or disprove Equation (4) using

$$\ell \geq 0 \implies g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0, \quad \text{respectively,} \quad u < 0 \implies g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0.$$

However, if the bounds are *inconclusive*, that is  $\ell < 0 \leq u$ , we can neither prove nor disprove Equation (4). Therefore, interval arithmetic is incomplete according to Definition 2.

Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a function that we want to bound for inputs  $\mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ . Interval arithmetic and other bound propagation techniques rely on  $f = f^{(K)} \circ \dots \circ f^{(1)}$  being a composition of more fundamental functions  $f^{(k)} : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_{k+1}}$  for which we can already compute  $\ell^{(k)} \leq f^{(k)}(\mathbf{z}) \leq u^{(k)}$  given  $\underline{\mathbf{z}} \leq \mathbf{z} \leq \overline{\mathbf{z}}$ . Examples of such functions include monotone non-decreasing functions, for which  $f^{(k)}(\underline{\mathbf{z}}) \leq f^{(k)}(\mathbf{z}) \leq f^{(k)}(\overline{\mathbf{z}})$ . This *bounding rule* already allows us to bound addition, min, max, and many popular neural network activation functions like ReLU, Sigmoid, and Tanh, among others. Appendix B contains further bounding rules. Given a bounding rule  $F^{(k)} : \mathbb{R}^{n_k} \times \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_{k+1}} \times \mathbb{R}^{n_{k+1}}$  for each  $f^{(k)}$ , interval arithmetic computes bounds on  $f$  by computing  $(F^{(K)} \circ \dots \circ F^{(1)})(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ .

### 3.2.2 Branch and Bound

Bound propagation approaches, such as interval arithmetic and CROWN [78], are incomplete according to Definition 2. To obtain a complete verifier, bound propagation can be combined with *branching* to gain completeness. This algorithmic framework is called *branch and bound* [20, 42, 51]. In branch and bound, when the computed bounds are inconclusive ( $\ell < 0 \leq u$ ), the search space is split (*branching*). The idea is that splitting improves the precision of the bounds for each part of the split (each *branch*).

Algorithm 1 contains an abstract branch and bound algorithm for neural network verification. This algorithm has three subprocedures that we need to instantiate for running Algorithm 1: Select, ComputeBounds, and Split. One instantiation that yields a complete algorithm is selecting branches in a FIFO order, computing bounds using interval arithmetic and splitting the search space by bisecting the input space [67]. Xu et al. [75] present an improved complete branch and bound algorithm that selects branches in batches to utilise massively parallel hardware such as GPUs, computes bounds using a combined bound propagation and optimisation procedure, and splits the search space by splitting on hidden ReLU nodes in the network [66]. We refer to Zhang et al. [77] for a state-of-the-art neural network verifier [18, 19] based on branch and bound. In the next section, we introduce a branch and bound algorithm for the probabilistic verification of neural networks.

---

#### Algorithm 1: Branch and Bound for Non-Probabilistic Neural Network Verification

---

**Input:** Verification Problem  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0, \forall \mathbf{x} \in \mathcal{X}'$  where  $\mathcal{X}' = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$

```

1 branches  $\leftarrow \{\mathcal{X}'\}$ 
2 while branches  $\neq \emptyset$  do
3    $[\underline{\mathbf{x}}', \overline{\mathbf{x}}'] \leftarrow \text{Select}(\text{branches})$  (Select one branch)
4    $(\underline{y}, \overline{y}) \leftarrow \text{ComputeBounds}(g_{\text{Sat}}(\cdot, \text{net}(\cdot)), \underline{\mathbf{x}}, \overline{\mathbf{x}})$  (E.g. IntervalArithmetic)
5   if  $\overline{y} < 0$  then return Violated (Every  $\mathbf{x} \in [\underline{\mathbf{x}}', \overline{\mathbf{x}}']$  is a counterexample)
6   else if  $\underline{y} \geq 0$  then branches  $\leftarrow (\text{branches} \setminus \{[\underline{\mathbf{x}}', \overline{\mathbf{x}}']\})$  (Prune branch)
7   else
8     new  $\leftarrow \text{Split}([\underline{\mathbf{x}}', \overline{\mathbf{x}}'])$  (E.g. by bisection)
9     branches  $\leftarrow (\text{branches} \setminus \{[\underline{\mathbf{x}}', \overline{\mathbf{x}}']\}) \cup \text{new}$ 
10 return Satisfied (All branches pruned)

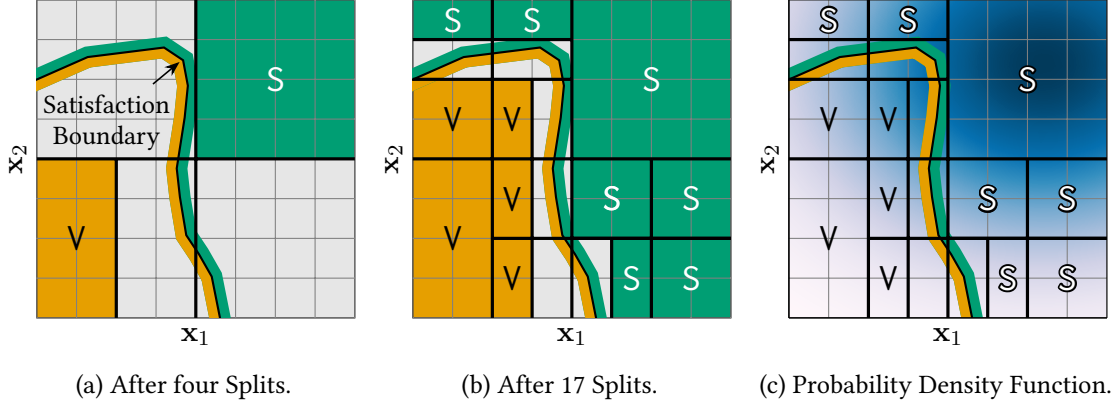
```

---

## 4 Algorithm

This section introduces ProbabilisticVerification (PV), our algorithm for probabilistic verification of neural networks as defined in Equation (3). The overall approach of PV is to iteratively refine bounds on each  $p_i$  from Equation (3) until a bound propagation approach allows us to prove or





**Figure 1: Computing Bounds on Probabilities.** This figure illustrates the steps for computing bounds on  $p = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ . Our algorithm successively splits the input space to find regions that do not intersect the satisfaction boundary  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0$  (orange/green line). Green, orange, and grey rectangles ( $\blacksquare/\blacksquare/\square$ ) denote regions for which we can prove  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0$  (satisfaction)  $\blacksquare$ ,  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0$  (violation)  $\blacksquare$ , or neither  $\square$ , respectively. By integrating the probability density  $f_{\mathbf{x}}$  in (c) (darker means higher density) over the green rectangles  $\blacksquare$ , we obtain a lower bound on  $p$ . Similarly, we can integrate over the orange rectangles  $\blacksquare$  to construct an upper bound on  $p$ . Refining the input splitting from (a) to (b) tightens the bounds on  $p$ .

disprove  $f_{\text{Sat}}(p_1, \dots, p_v) \geq 0$  (see Section 3.2.1). For computing the bounds on  $p_i$ , PV partitions the input space into hyperrectangles since this allows for computing probabilities efficiently [4]. To compute tighter bounds on  $p_i$ , PV uses a branch and bound algorithm that uses computationally inexpensive input splitting and bound propagation techniques from non-probabilistic neural network verification for refining the input splitting. Figure 1 illustrates our approach for computing and refining bounds on a probability  $p$ . The following sections first introduce the algorithm that solves Equation (3) given bounds on  $p_1, \dots, p_v$  before presenting the algorithm for computing the bounds on each  $p_i$ .

#### 4.1 Probabilistic Verification Algorithm

Algorithm 2 describes the PV algorithm. The centrepiece of PV is the procedure `ProbabilityBounds` for computing bounds  $\ell_i^{(t)} \leq p_i \leq u_i^{(t)}$  on a probability  $p_i$  from Equation (3). Given  $\ell_i^{(t)} \leq p_i \leq u_i^{(t)}$ , we apply a bound propagation technique, such as `IntervalArithmetic` as introduced in Section 3.2.1, to prove or disprove  $f_{\text{Sat}}(p_1, \dots, p_v)$ . As described in Section 3.2.1, this analysis may be inconclusive. In this case, `ProbabilityBounds` refines  $\ell_i^{(t)}, u_i^{(t)}$  to obtain  $\ell_i^{(t+1)}, u_i^{(t+1)}$  with  $\ell_i^{(t)} \leq \ell_i^{(t+1)} \leq p_i \leq u_i^{(t+1)} \leq u_i^{(t)}$ . We again apply bound propagation to  $f_{\text{Sat}}(p_1, \dots, p_v)$ , this time using  $\ell_i^{(t+1)}, u_i^{(t+1)}$ . If the result remains inconclusive, we iterate refining the bounds on each  $p_i$  until we obtain a conclusive result. PV applies `ProbabilityBounds` for each  $p_i$  in parallel, making use of several CPU cores or several GPUs. Our main contribution is the `ProbabilityBounds` algorithm for computing a converging sequence of lower and upper bounds on  $p_i$  as described in Section 4.2.

#### 4.2 Bounding Probabilities

Our `ProbabilityBounds` algorithm for deriving and refining bounds on a probability is described in detail in Algorithm 3 and illustrated in Figure 1. `ProbabilityBounds` is a massively parallel input-splitting branch and bound algorithm [20, 67, 74] that leverages a bound propagation algorithm for non-probabilistic neural network verification (`ComputeBounds`). Since we only consider a single probability in this section, we denote this probability as  $p = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ .

---

**Algorithm 2:** ProbabilisticVerification (PV)

---

**Input:** Probabilistic Verification Problem as in Equation (3), Batch Size  $N$

```
1 for  $i \in \{1, \dots, v\}$  do           (Launch  $v$  parallel instances of ProbabilityBounds)
2    $PB_i \leftarrow \text{Launch ProbabilityBounds}(p_i, N)$ 
3 for  $t \in \mathbb{N}$  do
4   for  $i \in \{1, \dots, v\}$  do Gather  $\ell_i^{(t)}, u_i^{(t)}$  from  $PB_i$ 
5    $\ell^{(t)}, u^{(t)} \leftarrow \text{ComputeBounds}(f_{\text{Sat}}, (\ell_1^{(t)}, u_1^{(t)}), \dots, (\ell_v^{(t)}, u_v^{(t)}))$ 
6   if  $\ell^{(t)} \geq 0$  then return Satisfied
7   if  $u^{(t)} < 0$  then return Violated
```

---

---

**Algorithm 3:** ProbabilityBounds

---

**Input:** Probability  $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ , Batch Size  $N$

```
1 branches  $\leftarrow \{\mathcal{X}\}$            ( $\mathcal{X}$  is the input space of net)
2  $\ell^{(0)} \leftarrow 0, u^{(0)} \leftarrow 1$ 
3 for  $t \in \mathbb{N}$  do
4   batch  $\leftarrow \text{Select}(\text{branches}, N)$            (e.g., branches  $\mathcal{B}_i$  with largest  $\mathbb{P}_{\mathbf{x}}[\mathcal{B}_i]$ )
5    $(\underline{\mathbf{y}}, \overline{\mathbf{y}}) \leftarrow \text{ComputeBounds}(g_{\text{Sat}}(\cdot, \text{net}(\cdot)), \text{batch})$ 
6    $(\text{batch}, \mathcal{X}_{\text{sat}}^{(t)}, \mathcal{X}_{\text{viol}}^{(t)}) \leftarrow \text{Prune}(\text{batch}, \underline{\mathbf{y}}, \overline{\mathbf{y}})$ 
7    $\ell^{(t)} \leftarrow \ell^{(t-1)} + \mathbb{P}_{\mathbf{x}}[\mathcal{X}_{\text{sat}}^{(t)}]$ 
8    $u^{(t)} \leftarrow u^{(t-1)} - \mathbb{P}_{\mathbf{x}}[\mathcal{X}_{\text{viol}}^{(t)}]$ 
9   yield  $\ell^{(t)}, u^{(t)}$            (Report new bounds to PV)
10  new  $\leftarrow \text{Split}(\text{batch})$            (e.g., BaBSB [20])
11  branches  $\leftarrow (\text{branches} \setminus \text{batch}) \cup \text{new}$ 
```

---

ProbabilityBounds receives  $p$  and a *batch size*  $N \in \mathbb{N}$  as input. The algorithm iteratively computes  $\ell^{(t)}, u^{(t)} \in [0, 1]$ , such that  $\ell^{(t)} \leq \ell^{(t')} \leq p \leq u^{(t')} \leq u^{(t)}, \forall t, t' \in \mathbb{N}, t' \geq t$ . The following sections describe each step of ProbabilityBounds in detail.

#### 4.2.1 Initialisation

Initially, we consider a single branch encompassing net’s entire input space  $\mathcal{X}$ . As in Section 3, we assume  $\mathcal{X}$  to be a (potentially unbounded) hyperrectangle. Section 4.3 describes how our approach can be used with non-hyperrectangular input spaces. We use the trivial bounds  $\ell^{(0)} = 0 \leq p \leq 1 = u^{(0)}$  as initial bounds on  $p$ .

#### 4.2.2 Selecting Branches

First, we select a batch of  $N \in \mathbb{N}$  branches. In the spirit of Xu et al. [75], we leverage the data parallelism of modern CPUs and GPUs to process several branches at once. In iteration  $t = 1$ , the batch only contains the branch  $\mathcal{X}$ . Which branches we select determines how fast we obtain tight bounds on  $p$ . We propose the SelectProb heuristic for selecting branches. Inspired by FairSquare [4], SelectProb selects the  $N$  branches  $\mathcal{B}_i$  with the largest  $\mathbb{P}_{\mathbf{x}}[\mathcal{B}_i]$ . This heuristic is motivated by the observation that pruning these branches would lead to the largest improvement of  $\ell^{(t)}, u^{(t)}$ .



### 4.2.3 Pruning

The next step is to prune those branches  $\mathcal{B}_j \in \text{batch}$ , for which we can determine that  $y = g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0$  is either certainly satisfied or certainly violated. For this, we first compute  $\underline{\mathbf{y}} \leq \mathbf{y} = g_{\text{Sat}}(\cdot, \text{net}(\cdot)) \leq \bar{\mathbf{y}}$  for the entire batch using a bound propagation algorithm for neural networks, such as CROWN [78], or  $\alpha$ -CROWN [75]. If  $\underline{y}_j \geq 0$  ( $\mathbf{y}_j \geq 0$  is certainly satisfied) or  $\bar{y}_j < 0$  ( $\mathbf{y}_j \geq 0$  is certainly violated), we can prune  $\mathcal{B}_j$ , meaning that we remove it from branches. We collect the branches with  $\underline{y}_j \geq 0$  in the set  $\mathcal{X}_{\text{sat}}^{(t)}$  and the branches with  $\bar{y}_j < 0$  in  $\mathcal{X}_{\text{viol}}^{(t)}$ , where  $t \in \mathbb{N}$  is the current iteration.

### 4.2.4 Updating Bounds

Let  $\hat{\mathcal{X}}_{\text{sat}}^{(t)} = \bigcup_{t'=1}^t \mathcal{X}_{\text{sat}}^{(t')}$  and  $\hat{\mathcal{X}}_{\text{viol}}^{(t)} = \bigcup_{t'=1}^t \mathcal{X}_{\text{viol}}^{(t')}$ , where  $t \in \mathbb{N}$  is the current iteration. Then,  $\ell^{(t)} = \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{sat}}^{(t)}] \leq p$ . Similarly,  $k^{(t)} = \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{viol}}^{(t)}] \leq \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0] = 1 - p$ . Therefore,  $1 - k^{(t)} = u^{(t)} \geq p$ . Practically, we only have to maintain the current bounds  $\ell^{(t)}$  and  $u^{(t)}$  instead of the sets  $\hat{\mathcal{X}}_{\text{sat}}^{(t)}$  and  $\hat{\mathcal{X}}_{\text{viol}}^{(t)}$ .

Because  $\hat{\mathcal{X}}_{\text{sat}}^{(t)}$  and  $\hat{\mathcal{X}}_{\text{viol}}^{(t)}$  are a union of disjoint hyperrectangles, exactly computing  $\mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{sat}}^{(t)}]$  and  $\mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{viol}}^{(t)}]$  is feasible for a large class of probability distributions, including most discrete and continuous univariate distributions, as well as Mixture Models and Bayesian Networks. The precise class of supported probability distributions is discussed in Section 4.3. Practically, one must also accommodate floating point errors in these computations. While we do not address this in this paper, our approach can be extended to this end.

ProbabilityBounds now reports the refined bounds  $\ell^{(t)} \leq p \leq u^{(t)}$  to PV, which applies ComputeBounds to determine whether net is feasible for the probabilistic verification problem. If this remains inconclusive, we proceed with splitting the selected branches.

### 4.2.5 Splitting

Splitting refines a branch  $\mathcal{B} = [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  by selecting a dimension  $d \in \{1, \dots, n\}$  to split. We first describe how  $d$  is split before discussing how to select  $d$ . A dimension can encode several types of variables. We consider continuous variables, such as normalised pixel values, integer variables, such as age, and dimensions containing one indicator of a one-hot encoded categorical variable like gender. The type of variable encoded in  $d$  determines how we split  $d$ .

- For continuous variables, we further differentiate whether  $\mathcal{B}$  is bounded, unbounded in one direction, or unbounded in both directions in dimension  $d$ .
  - If  $\mathcal{B}$  is bounded in dimension  $d$ , we bisect  $\mathcal{B}$  along  $d$  resulting in two new branches  $[\underline{\mathbf{x}}', \bar{\mathbf{x}}']$  and  $[\underline{\mathbf{x}}'', \bar{\mathbf{x}}'']$ . Concretely,  $\underline{\mathbf{x}}'_{d'} = \underline{\mathbf{x}}''_{d'} = \underline{\mathbf{x}}_{d'}$  and  $\bar{\mathbf{x}}'_{d'} = \bar{\mathbf{x}}''_{d'} = \bar{\mathbf{x}}_{d'}$  for all  $d' \in \{1, \dots, n\} \setminus \{d\}$  while  $\bar{\mathbf{x}}'_d = \underline{\mathbf{x}}'_d = (\underline{\mathbf{x}}_d + \bar{\mathbf{x}}_d)/2$ ,  $\underline{\mathbf{x}}''_d = \underline{\mathbf{x}}_d$ , and  $\bar{\mathbf{x}}''_d = \bar{\mathbf{x}}_d$ .
  - If  $\mathcal{B}$  is unbounded in both directions in  $d$ , we split  $d$  at zero, so that  $\bar{\mathbf{x}}'_d = \underline{\mathbf{x}}''_d = 0$ . The remaining bounds of the new branches  $[\underline{\mathbf{x}}', \bar{\mathbf{x}}']$  and  $[\underline{\mathbf{x}}'', \bar{\mathbf{x}}'']$  are as in the bounded case.
  - If  $\mathcal{B}$  is bounded from below but unbounded from above in  $d$ , that is  $-\infty < \underline{\mathbf{x}}_d < \bar{\mathbf{x}}_d = \infty$ , we split at  $\bar{\mathbf{x}}'_d = \underline{\mathbf{x}}''_d = \max(2\underline{\mathbf{x}}_d, 1)$ , all else being as above. Effectively, this split rule performs an exponential search over unbounded dimensions until the remaining unbounded branches are no longer selected by Select, for example, because they have diminishing probability in the case of SelectProb. ProbabilityBounds handles the case where  $d$  is bounded from above but unbounded from below analogously.
- For integer variables, we split  $d$  as for a continuous variable to obtain  $[\underline{\mathbf{x}}', \bar{\mathbf{x}}']$ ,  $[\underline{\mathbf{x}}'', \bar{\mathbf{x}}'']$  and round  $\bar{\mathbf{x}}'_d$  to the next smaller integer while rounding  $\underline{\mathbf{x}}''_d$  to the next larger integer.

- For a one-hot encoded categorical variable  $V$  encoded in the dimensions  $A \subseteq \{1, \dots, n\}$  with  $d \in A$ , we create one split where  $V$  is equal to the category represented by  $d$  and one where  $V$  is different from this category. Formally,  $\underline{x}'_d = \bar{x}'_d = 1$  and  $\underline{x}'_{d'} = \bar{x}'_{d'} = 0$  for  $d' \in A \setminus \{d\}$  defines  $[\underline{x}', \bar{x}']$ . For  $[\underline{x}'', \bar{x}'']$ , we set  $\underline{x}''_d = \bar{x}''_d = 0$  and leave the remaining values as they are in  $\underline{x}$  and  $\bar{x}$ . This splitting procedure eventually creates a new branch where all dimensions are set to zero. This branch has zero probability and can be discarded immediately.

In any case, we need to ensure not to select  $d$  if  $\underline{x}_d = \bar{x}_d$ . We introduce three heuristics for selecting a dimension:

- LongestEdge: This well-known heuristic [20] selects the dimension with the largest *edge length*  $\bar{x}_d - \underline{x}_d$ .
- BaBSB: We use a variant of the BaBSB heuristic of Bunel et al. [20]. The idea of BaBSB is to estimate the improvement in bounds that splitting dimension  $d$  yields by using a yet less expensive technique than ComputeBounds. Our variant of BaBSB uses IntervalArithmetic, assuming that we use CROWN or  $\alpha$ -CROWN for ComputeBounds. Let  $[\underline{x}^{(d,1)}, \bar{x}^{(d,1)}]$  and  $[\underline{x}^{(d,2)}, \bar{x}^{(d,2)}]$  be the two new branches originating from splitting dimension  $d \in \{1, \dots, n\}$  and let  $\underline{y}^{(d,1)}, \bar{y}^{(d,1)}, \underline{y}^{(d,2)}, \bar{y}^{(d,2)}$  be the bounds that IntervalArithmetic computes on  $g_{\text{Sat}}(\cdot, \text{net}(\cdot))$  for these branches. Our BaBSB selects  $d = \arg \max_{d \in \{1, \dots, n\}} \tilde{y}^{(d)}$ , where  $\tilde{y}^{(d)} = \max(\max(\underline{y}^{(d,1)}, \underline{y}^{(d,2)}), -\min(\bar{y}^{(d,1)}, \bar{y}^{(d,2)}))$ .
- BaBSB-LongestEdge-k: This heuristic alternates using BaBSB and LongestEdge, using LongestEdge for every  $k$ -th split. If we visualise branches and their descendants from splitting in a branching tree, the splits at level  $k, 2k, 3k, \dots$  use LongestEdge while the splits at all other levels use BaBSB.

While LongestEdge is more theoretically accessible, BaBSB is practically advantageous, as discussed in Appendix E. BaBSB-LongestEdge-k combines the advantages of both approaches.

### 4.3 Input Spaces and Input Distributions

This section discusses the concrete requirements that the input space  $\mathcal{X}$  and the input distributions  $\mathbb{P}_{\mathbf{x}^{(i)}}$  in Equation (3) need to satisfy for applying PV. We also discuss how some of these requirements can be mitigated, for example, for using polytopes as input spaces.

PV requires  $\mathcal{X} \subseteq \mathbb{R}^n$  to be a hyperrectangle, which can be unbounded. The dimensions of  $\mathcal{X}$  may encode discrete random variables, as discussed in Section 4.2.5. Each probability distribution  $\mathbb{P}_{\mathbf{x}^{(i)}}$  needs to allow for computing the probability of a hyperrectangle in closed form. This requirement is satisfied by a large class of probability distributions, including discrete distributions with a closed-form probability mass function and univariate continuous distributions with a closed-form cumulative density function, as well as Mixture Models and probabilistic graphical models [15], such as Bayesian Networks, of such distributions.

We now discuss how we can apply PV to multivariate normal distributions even though they do not admit a closed-form solution for the probability of a hyperrectangle. Consider a multivariate normal distribution  $\mathbb{P}_{\mathbf{z}}$  with mean  $\boldsymbol{\mu}$  and covariance  $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^\top$ . Let  $\mathbb{P}_{\mathbf{x}}$  be a standard multivariate normal distribution. Now,  $\mathbf{z} = \mathbf{A}\mathbf{x} + \boldsymbol{\mu}$  is distributed according to  $\mathbb{P}_{\mathbf{z}}$ . Therefore, by prepending the linear transformation  $\mathbf{A}\mathbf{x} + \boldsymbol{\mu}$  to net, we can apply PV to multivariate normal distributions, since the probability of a hyperrectangle under a standard multivariate normal distribution has a closed-form solution.

Lastly, we show how to apply PV to polytopal input spaces, even though they can not be used as input space directly. Let  $\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$  be a polytope and let  $\mathcal{X} \supseteq \mathcal{P}$  be a hyperrectangle

enclosing  $\mathcal{P}$ . By using  $\mathcal{X}$  as the input space and using

$$\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0 \mid \mathbf{Ax} \leq \mathbf{b}] = \frac{\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0 \wedge \mathbf{Ax} \leq \mathbf{b}]}{\mathbb{P}_{\mathbf{x}}[\mathbf{Ax} \leq \mathbf{b}]}$$

we can apply PV to polytopes as input spaces. However, before applying PV, we should check whether  $\mathbb{P}_{\mathbf{x}}[\mathbf{Ax} \leq \mathbf{b}] > 0$ , since, otherwise, the verification problem is ill-defined. We can apply ProbabilityBounds for this purpose by computing bounds on  $\mathbb{P}_{\mathbf{x}}[\mathbf{Ax} \leq \mathbf{b}]$ .

## 5 Theoretical Analysis

In this section, we prove that PV is a sound probabilistic verification algorithm when instantiated with a suitable ComputeBounds procedure. We also prove that PV is complete under mild assumptions on the probabilistic verification problem when instantiated with suitable Split, ComputeBounds, and Select procedures. Soundness and completeness are defined in Definition 2. As in Section 4.2, we omit indices and superscripts when considering only a single probability  $p = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ .

### 5.1 Soundness

First, we prove that ProbabilityBounds produces sound bounds on  $p$  when using a sound ComputeBounds procedure, such as interval arithmetic or CROWN [78]. A ComputeBounds procedure is sound if it computes valid bounds.

**Theorem 1** (Sound Bounds). *Let  $N \in \mathbb{N}$  be a batch size and assume ComputeBounds produces valid bounds. Let  $\{(\ell^{(t)}, u^{(t)})\}_{t \in \mathbb{N}}$  be the iterates of ProbabilityBounds( $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ ,  $N$ ). It holds that  $\ell^{(t)} \leq \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0] \leq u^{(t)}$  for all  $t \in \mathbb{N}$ .*

*Proof.* Let  $t \in \mathbb{N}$  and let  $\mathcal{X}_{\text{sat}}^{(t)}$  and  $\mathcal{X}_{\text{viol}}^{(t)}$  be as in Algorithm 3. ProbabilityBounds computes  $\ell^{(t)}$  as the total probability of all previously pruned satisfied branches  $\hat{\mathcal{X}}_{\text{sat}}^{(t)} = \bigcup_{t'=1}^t \mathcal{X}_{\text{sat}}^{(t')}$ . Similarly,  $u^{(t)} = 1 - \hat{k}^{(t)}$  where  $\hat{k}^{(t)}$  is the total probability of all previously pruned violated branches  $\hat{\mathcal{X}}_{\text{viol}}^{(t)} = \bigcup_{t'=1}^t \mathcal{X}_{\text{viol}}^{(t')}$ . Since we assumed that ComputeBounds produces valid bounds, Prune only prunes branches that are actually satisfied or violated. Therefore,  $\hat{\mathcal{X}}_{\text{sat}}^{(t)} \subseteq \{\mathbf{x} \in \mathcal{X} \mid g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0\}$  and  $\hat{\mathcal{X}}_{\text{viol}}^{(t)} \subseteq \{\mathbf{x} \in \mathcal{X} \mid g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0\}$ . From this, it follows directly that

$$\begin{aligned} \ell^{(t)} &= \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{sat}}^{(t)}] \leq \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0] \\ \hat{k}^{(t)} &= \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{viol}}^{(t)}] \leq \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0], \end{aligned}$$

which implies  $u^{(t)} = 1 - \hat{k}^{(t)} \geq 1 - \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0] = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ . This shows that ProbabilityBounds is sound.  $\square$

**Corollary 1** (Soundness). *PV is sound when using ComputeBounds procedures that compute valid bounds.*

*Proof.* Corollary 1 follows from Theorem 1 and the soundness of the ComputeBounds procedure applied by PV.  $\square$

## 5.2 Completeness

We study the completeness of PV. Concretely, we prove that PV instantiated with suitable Split, ComputeBounds, and Select procedures is complete under a mildly restrictive assumption on Equation (3). Below, we first state this assumption and then define what it means for the Split, ComputeBounds, and Select procedures to be suitable. Finally, we show that the SelectProb, LongestEdge, and BaBSB-LongestEdge-k heuristics introduced in Section 4.2 are suitable and prove the completeness of PV.

**Assumption 1.** Let  $v$ ,  $f_{\text{Sat}}$ ,  $g_{\text{Sat}}^{(i)}$ , and  $\mathbf{x}^{(i)}$  be as in Equation (3). Assume  $f_{\text{Sat}}(p_1, \dots, p_v) \neq 0$  and  $\forall i \in \{1, \dots, v\} : \mathbb{P}_{\mathbf{x}^{(i)}}[g_{\text{Sat}}^{(i)}(\mathbf{x}^{(i)}, \text{net}(\mathbf{x}^{(i)})) = 0] = 0$ .

To prove the completeness of PV, we first establish that ProbabilityBounds produces a sequence of lower and upper bounds that converge towards each other. Intuitively, we require Assumption 1 since converging bounds on  $f_{\text{Sat}}(p_1, \dots, p_n)$  are insufficient for proving  $f_{\text{Sat}}(p_1, \dots, p_n) \geq 0$  if  $f_{\text{Sat}}(p_1, \dots, p_n) = 0$  [4]. However, excluding  $f_{\text{Sat}}(p_1, \dots, p_v) = 0$  is only mildly restrictive as we can always tighten the constraint to  $f_{\text{Sat}}(p_1, \dots, p_n) \geq \varepsilon$  for an arbitrarily small  $\varepsilon > 0$ . The second assumption  $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) = 0] = 0$  is only mildly restrictive for similar reasons. In particular, we can tighten  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0$  to  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq \varepsilon$  for  $\varepsilon > 0$  such that  $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) = \varepsilon] = 0$ . Such a  $\varepsilon > 0$  exists because  $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) = 0] = 0$  means that the satisfaction boundary has positive volume but any neural network has only finitely many flat regions that can produce a satisfaction boundary of positive volume. Next, we define conditions on the ComputeBounds, Split, and Select procedures that ensure the completeness of PV.

**Definition 3** (Convergent Bounds). Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . We call a ComputeBounds procedure that computes  $\underline{\mathbf{y}} \leq f(\mathbf{x}) \leq \bar{\mathbf{y}}$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  *convergent* if  $\|\bar{\mathbf{y}} - \underline{\mathbf{y}}\| \rightarrow 0$  as  $\|\bar{\mathbf{x}} - \underline{\mathbf{x}}\| \rightarrow 0$  and  $\|\bar{\mathbf{y}} - \underline{\mathbf{y}}\| = 0$  if  $\|\bar{\mathbf{x}} - \underline{\mathbf{x}}\| = 0$ .

**Definition 4** (Dimension Alternation). Let  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}] \subseteq \mathbb{R}^n$ . A splitting procedure Split is *dimension-alternating* if for every  $d \in \{1, \dots, n\}$  with  $\underline{\mathbf{x}}_d \neq \bar{\mathbf{x}}_d$

$$\exists t \in \mathbb{N} : \exists [\underline{\mathbf{x}}', \bar{\mathbf{x}}'] \in \text{branches}^{(t)} : \bar{\mathbf{x}}'_d - \underline{\mathbf{x}}'_d < \bar{\mathbf{x}}_d - \underline{\mathbf{x}}_d,$$

where  $\text{branches}^{(t)} = \text{Split}(\text{branches}^{(t-1)})$  for  $t \in \mathbb{N}$  and  $\text{branches}^{(0)} = [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ .

**Definition 5** (Branch Alternation). A branch selection procedure Select is *branch-alternating* if

$$\forall t \in \mathbb{N} : \forall \mathcal{B} \in \text{branches}^{(t)} : \mathbb{P}_{\mathbf{x}}[\mathcal{B}] > 0 \implies \exists t' \geq t : \mathcal{B} \in \text{Select}(\text{branches}^{(t')}, N),$$

where  $N \in \mathbb{N}$  and  $\text{branches}^{(t)}$  is the value of the branches variable of ProbabilityBounds in iteration  $t$  where ProbabilityBounds is instantiated with Select and a ComputeBounds procedure satisfying Definition 3.

In the following, we prove SelectProb, LongestEdge, and BaBSB-LongestEdge-k as introduced in Section 4.2 are branch alternating and dimensional alternating, respectively. It is well-known that IntervalArithmetic satisfies Definition 3 [49]. We provide a proof in Appendix B.2. We also show that CROWN satisfies Definition 3.

**Proposition 1.** CROWN [78] satisfies Definition 3.

Since Proposition 1 is not directly concerned with an algorithm introduced in this paper, we defer its proof to Appendix C.

**Proposition 2.** LongestEdge satisfies Definition 4.

*Proof.* Let  $[\underline{x}, \bar{x}] \subseteq \mathbb{R}^n$ ,  $d \in \{1, \dots, n\}$  with  $\underline{x}_d \neq \bar{x}_d$ , and let branches<sup>(t)</sup> for  $t \in \mathbb{N}_0$  be as in Definition 4. We call  $\bar{x}_d - \underline{x}_d$  the *edge length* of  $d$  in  $[\underline{x}, \bar{x}]$ .

If  $\bar{x}_d - \underline{x}_d > \max_{d' \neq d} \bar{x}_{d'} - \underline{x}_{d'}$ , the dimension  $d$  is selected for splitting immediately. In the following, we not only show that  $\bar{x}_d - \underline{x}_d$  decreases when split but also that  $\bar{x}_d - \underline{x}_d \rightarrow 0$  in at least one branch when we split  $d$  repeatedly. This result is required for the second part of this proof. We differentiate several cases based on the variable encoded in  $d$ .

- *Bounded Continuous Variable.* Let  $d$  encode a continuous variable with  $\bar{x}_d - \underline{x}_d < \infty$ . As described in Section 4.2.5 we split such dimensions by bisecting  $[\underline{x}, \bar{x}]$  along  $d$ . Bisecting decreases the edge length of  $d$  in the resulting branches so that we have  $\bar{x}'_d - \underline{x}'_d < \bar{x}_d - \underline{x}_d$  for all  $[\underline{x}', \bar{x}'] \in \text{branches}^{(1)} = \text{Split}([\underline{x}, \bar{x}])$ . Furthermore, the edge length of  $d$  converges towards zero if we bisect along  $d$  repeatedly.
- *Continuous Variable Bounded from Below but Unbounded from Above.* Let  $d$  encode a continuous variable with  $-\infty < \underline{x}_d < \bar{x}_d = \infty$ . Since splitting such a dimension creates one branch where  $d$  is bounded, we have  $\bar{x}'_d - \underline{x}'_d < \bar{x}_d - \underline{x}_d$  for the bounded branch  $[\underline{x}', \bar{x}'] \in \text{branches}^{(1)}$ . Furthermore, repeatedly splitting the bounded branch along  $d$  lets the edge length of  $d$  converge towards zero, as discussed above.
- *Continuous Variable Bounded from Above but Unbounded from Below.* This case proceeds analogously to the previous case.
- *Continuous Variable Unbounded from Both Sides.* Splitting along such variables creates two branches that are bounded from one side. Therefore, after two splits, we obtain two bounded branches, such that  $\bar{x}'_d - \underline{x}'_d < \bar{x}_d - \underline{x}_d$  for two  $[\underline{x}', \bar{x}'] \in \text{branches}^{(2)} = \text{Split}(\text{branches}^{(1)})$ . Similarly, repeatedly splitting the bounded branches along  $d$  lets the edge length of  $d$  converge towards zero.
- *Integer Variable.* Let  $d$  encode an integer variable. Splitting  $d$  proceeds as for a continuous variable, except for excluding non-integer values from the new branches. This decreases the edge length of  $d$  at least as much as if we were splitting a continuous variable. Therefore, we have that  $\bar{x}'_d - \underline{x}'_d < \bar{x}_d - \underline{x}_d$  for at least one branch  $[\underline{x}', \bar{x}'] \in \text{branches}^{(1)}$ . Furthermore, the edge length of  $d$  reaches zero after finitely many splits in all bounded branches since we exclude non-integer values.
- *One-Hot Encoded Categorical Variable.* Let  $d$  contain an indicator of a one-hot encoded categorical variable. Splitting  $d$  decreases the edge length of  $d$  to zero, so that we have  $\bar{x}'_d - \underline{x}'_d = 0 < \bar{x}_d - \underline{x}_d$  for all  $[\underline{x}', \bar{x}'] \in \text{branches}^{(1)}$ .

Overall, Definition 4 is satisfied if dimension  $d$  is selected for splitting immediately.

Now consider the case that  $d$  is not selected for splitting immediately. In this case, a different dimension  $d' \in \{1, \dots, n\}$ ,  $d' \neq d$  with  $\bar{x}_{d'} - \underline{x}_{d'} \geq \bar{x}_d - \underline{x}_d$ , is selected for splitting by LongestEdge. As we have argued above, repeatedly splitting  $d'$  lets the edge length of  $d'$  decrease towards zero in at least one branch. Therefore, we eventually obtain  $[\underline{x}', \bar{x}'] \in \text{branches}^{(t)}$  with  $\bar{x}'_{d'} - \underline{x}'_{d'} < \bar{x}_d - \underline{x}_d$ . Since this holds for all  $d'' \in \{1, \dots, n\}$ ,  $d'' \neq d$  with  $\bar{x}_{d''} - \underline{x}_{d''} \geq \bar{x}_d - \underline{x}_d$ , we eventually obtain a branch where LongestEdge splits  $d$ . Therefore, Definition 4 is also satisfied if dimension  $d$  is not selected for splitting immediately. Overall, LongestEdge satisfies Definition 4.  $\square$

**Corollary 2.** *BaBSB-LongestEdge-k satisfies Definition 4.*

**Proposition 3.** *SelectProb satisfies Definition 5.*

*Proof.* Let branches<sup>(t)</sup> be the value of the branches variable of ProbabilityBounds (Algorithm 3) in iteration  $t \in \mathbb{N}$ , where ProbabilityBounds is instantiated with SelectProb and a ComputeBounds



procedure satisfying Definition 3. Let  $N, t \in \mathbb{N}$  and  $\mathcal{B} \in \text{branches}^{(t)}$  with  $\mathbb{P}_{\mathbf{x}}[\mathcal{B}] > 0$ . Our goal is to show

$$\exists t' \geq t : \mathcal{B} \in \text{SelectProb}(\text{branches}^{(t')}, N). \quad (5)$$

If  $\mathcal{B} \in \text{SelectProb}(\text{branches}^{(t)}, N)$ , Equation (5) holds immediately. Otherwise, there are at least  $N$  branches  $\mathcal{B}'_t$  in iteration  $t$  with  $\mathbb{P}_{\mathbf{x}}[\mathcal{B}'_t] \geq \mathbb{P}_{\mathbf{x}}[\mathcal{B}]$ . We show

$$\exists t' > t : \forall \mathcal{B}'_t, \mathbb{P}_{\mathbf{x}}[\mathcal{B}'_t] \geq \mathbb{P}_{\mathbf{x}}[\mathcal{B}] : \underbrace{\forall \mathcal{B}'_{t'}, \mathcal{B}'_t \rightsquigarrow \mathcal{B}'_{t'} : \mathbb{P}_{\mathbf{x}}[\mathcal{B}'_{t'}] < \mathbb{P}_{\mathbf{x}}[\mathcal{B}]}_{(*)}, \quad (6)$$

where  $\mathcal{B}'_t \rightsquigarrow \mathcal{B}'_{t'}$  if  $\mathcal{B}'_{t'}$  is a branch in iteration  $t' \in \mathbb{N}$  that originates from splitting  $\mathcal{B}'_t$ , meaning that  $\mathcal{B}'_{t'} \subset \mathcal{B}'_t$ .

Let  $\mathcal{B}'_t$  be a branch in iteration  $t$  with  $\mathbb{P}_{\mathbf{x}}[\mathcal{B}'_t] \geq \mathbb{P}_{\mathbf{x}}[\mathcal{B}]$ . First of all, if  $\mathcal{B}'_t$  is pruned by ProbabilityBounds in iteration  $t$ , then there are no new branches originating from  $\mathcal{B}'_t$ , so that  $(*)$  holds vacuously. Otherwise, ProbabilityBounds splits  $\mathcal{B}'_t$ .

We first consider the special case where the input space only contains categorical and bounded integer variables. Dimensions encoding such variables can only be split finitely often. Therefore, splitting  $\mathcal{B}'_t$  eventually produces a finite set of branches  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  with  $\underline{\mathbf{x}} = \bar{\mathbf{x}}$ . Since we assumed ComputeBounds to satisfy Definition 3, ComputeBounds computes the bounds  $\underline{\mathbf{y}} = \bar{\mathbf{y}}$  for branches with  $\underline{\mathbf{x}} = \bar{\mathbf{x}}$ . Branches with  $\underline{\mathbf{y}} = \bar{\mathbf{y}}$  are certainly pruned by ProbabilityBounds. Therefore, if we choose  $t' > t$  large enough, Equation (6) holds with  $(*)$  holding vacuously, since all branches originating from  $\mathcal{B}'_t$  have been pruned.

Otherwise, let the input space contain at least one continuous or unbounded integer variable. We show that there is an iteration  $t' > t$  such that  $(*)$  holds for  $\mathcal{B}'_t$ . Without loss of generality, assume that the dimension selected for splitting encodes a continuous variable or an unbounded integer variable. This does not harm generality since categorical variables and bounded integer variables can only be split finitely often and, therefore, will eventually become unavailable for splitting.

First, consider splitting along a dimension  $d$  encoding a bounded continuous variable. Since we split continuous variables by bisection, the volume of all branches  $\mathcal{B}'_{t'}$  originating from  $\mathcal{B}'_t$  decreases towards zero as we split  $d$  repeatedly. As stated in Section 3, we assume that all continuous random variables admit a probability density function. This implies that the probability in all branches  $\mathcal{B}'_{t'}$  originating from splitting  $\mathcal{B}'_t$  decreases towards zero as the volume decreases towards zero.

Now, consider splitting along a dimension  $d$  encoding an unbounded variable. Without loss of generality, assume that  $\mathcal{B}'_t$  is bounded in  $d$  in at least one direction. This does not harm generality since dimensions unbounded in both directions are split into two parts, each bounded in one direction. Given this, splitting along  $d$  creates a bounded and an unbounded part. If  $d$  encodes an integer variable, the bounded part contains only finitely many discrete values. Therefore, as argued above, all branches originating from this bounded part are eventually pruned. If  $d$  encodes a continuous variable, the bounded part behaves as described above with the probability of all branches  $\mathcal{B}'_{t'}$  originating from  $\mathcal{B}'_t$  decreasing towards zero. Therefore, we only have to show that the probability remaining in the unbounded part decreases towards zero as we continue splitting to show  $(*)$ . In fact, this follows from the properties of a probability measure.

Above, we have shown that splitting repeatedly either leads to pruning the resulting branches or the probability of all resulting branches decreases towards zero. Therefore, there is a  $t'$ , such that  $(*)$  is satisfied for  $\mathcal{B}'_t$ . Since there are only finitely many branches in any iteration of ProbabilityBounds, the above implies that Equation (6) is satisfied. In turn, this directly implies that  $\mathcal{B}$  is eventually selected by ProbabilityBounds, proving Proposition 3.  $\square$

**Lemma 1** (Converging Probability Bounds). *Let  $N \in \mathbb{N}$  be a batch size. Let  $\{(\ell^{(t)}, u^{(t)})\}_{t \in \mathbb{N}}$  be the iterates of ProbabilityBounds( $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ ,  $N$ ) instantiated with a Select procedure*



satisfying Definition 4, a sound ComputeBounds procedure satisfying Definition 3 and a Split procedure satisfying Definition 5. Assume  $\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) = 0] = 0$  as in Assumption 1. Then,

$$\lim_{t \rightarrow \infty} \ell^{(t)} = \lim_{t \rightarrow \infty} u^{(t)} = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0].$$

*Proof.* We first prove that  $\lim_{t \rightarrow \infty} \ell^{(t)} = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$ . The convergence of the upper bound,  $\lim_{t \rightarrow \infty} u^{(t)} = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$  follows analogously.

Let  $\mathcal{X}_{\text{sat}}^* = \{\mathbf{x} \in \mathcal{X} \mid g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) > 0\}$ , where  $\mathcal{X}$  is the input space of  $\text{net}$ . Note that  $\mathbb{P}_{\mathbf{x}}[\mathcal{X}_{\text{sat}}^*] = \mathbb{P}_{\mathbf{x}}[\{\mathbf{x} \in \mathcal{X} \mid g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0\}]$  due to Assumption 1. Further, let  $\hat{\mathcal{X}}_{\text{sat}}^{(t)}$  be as in the proof of Theorem 1 and recall  $\ell^{(t)} = \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{sat}}^{(t)}]$ .

First, we give an argument why the limit  $\lim_{t \rightarrow \infty} \ell^{(t)}$  exists. Due to Theorem 1, the sequence  $\{\ell^{(t)}\}_{t \in \mathbb{N}}$  is bounded from above. Furthermore,  $\{\ell^{(t)}\}_{t \in \mathbb{N}}$  is non-decreasing in  $t$  since ProbabilityBounds only adds elements to  $\hat{\mathcal{X}}_{\text{sat}}^{(t)}$ . Therefore,  $\lim_{t \rightarrow \infty} \ell^{(t)}$  exists. Now, we can equivalently rewrite

$$\begin{aligned} \ell^{(t)} &\xrightarrow{t \rightarrow \infty} \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0] \\ \iff \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{sat}}^{(t)}] &\xrightarrow{t \rightarrow \infty} \mathbb{P}_{\mathbf{x}}[\mathcal{X}_{\text{sat}}^*] \\ \iff \mathbb{P}_{\mathbf{x}}[\hat{\mathcal{X}}_{\text{sat}}^{(t)}] - \mathbb{P}_{\mathbf{x}}[\mathcal{X}_{\text{sat}}^*] &\xrightarrow{t \rightarrow \infty} 0 \\ \iff \mathbb{P}_{\mathbf{x}}[\mathcal{X}_{\text{sat}}^* \setminus \hat{\mathcal{X}}_{\text{sat}}^{(t)}] &\xrightarrow{t \rightarrow \infty} 0. \end{aligned} \tag{7}$$

We now argue why (7) holds. Consider the case that the input space does not contain a continuous variable. Let  $\mathcal{B}_t$  be a bounded branch in iteration  $t \in \mathbb{N}$  for an input space containing only discrete variables. As discussed in the proof of Proposition 3, there is an iteration  $t' > t$  so that all branches  $\mathcal{B}_{t'}$  that originate from splitting  $\mathcal{B}_t$  are pruned. Due to the properties of a probability measure, the probability of the unbounded branches  $\mathcal{B}_t$  that remain decreases towards zero as  $t \rightarrow \infty$ . Therefore, (7) holds if the input space does not contain a continuous variable.

Now, assume the input space contains at least one continuous variable. Let  $\tilde{\mathcal{X}}_{\text{sat}}^{(t)} = \mathcal{X}_{\text{sat}}^* \setminus \hat{\mathcal{X}}_{\text{sat}}^{(t)}$ . With the goal of obtaining a contradiction, assume  $\lim_{t \rightarrow \infty} \mathbb{P}_{\mathbf{x}}[\tilde{\mathcal{X}}_{\text{sat}}^{(t)}] > 0$ . Since we assumed in Section 3 that every continuous probability distribution admits a density function,  $\text{vol}(\tilde{\mathcal{X}}_{\text{sat}}^{(t)}) > 0$ , where  $\text{vol}$  denotes the volume. Let  $t \in \mathbb{N}$ . Since the branches maintained by ProbabilityBounds form a partition of the input space, there is a branch  $\mathcal{B}_t$  in iteration  $t$  of ProbabilityBounds such that  $\text{vol}(\tilde{\mathcal{X}}_{\text{sat}}^{(t)} \cap \mathcal{B}_t) > 0$ .

Due to Split satisfying Definition 4 and Select satisfying Definition 5, we eventually obtain  $\mathcal{B}_{t'}$  in iteration  $t' > t$  with  $\mathcal{B}_{t'} \subseteq \tilde{\mathcal{X}}_{\text{sat}}^{(t)} \cap \mathcal{B}_t$ . Additionally, since Split satisfies Definition 4, we have that  $\|\bar{\mathbf{x}} - \underline{\mathbf{x}}\| \rightarrow 0$  for all not yet pruned branches  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  as  $t \rightarrow \infty$ . Since ComputeBounds satisfies Definition 3, we have that the bounds  $\underline{y} \leq g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \leq \bar{y}$  produced by ComputeBounds converge towards  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x}))$ . Note that  $g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) > 0$  for all  $\mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  since  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}] \subseteq \mathcal{B}_{t'} \subseteq \mathcal{X}_{\text{sat}}^*$ . This implies that there is an iteration  $t'' > t'$  in which ProbabilityBounds considers a branch  $\mathcal{B}_{t''} \subset \tilde{\mathcal{X}}_{\text{sat}}^{(t)}$  for which  $\underline{y} > 0$ , which means that ProbabilityBounds prunes  $\mathcal{B}_{t''}$ . This contradicts the construction of  $\tilde{\mathcal{X}}_{\text{sat}}^{(t)}$ . With this contradiction, we have shown  $\lim_{t \rightarrow \infty} \ell^{(t)} = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$  when the input space contains a continuous variable.

Overall, we have shown  $\lim_{t \rightarrow \infty} \ell^{(t)} = \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]$  for all possible compositions of the input space. The convergence of the upper bound  $u^{(t)}$  follows from an analogous argument on  $k^{(t)}$  as in the proof of Theorem 1 where  $u^{(t)} = 1 - k^{(t)}$ . This establishes Lemma 1.  $\square$

**Theorem 2** (Completeness). *When instantiated with ProbabilityBounds as in Lemma 1 and a ComputeBounds procedure that satisfies Definition 3, PV is complete for verification problems satisfying Assumption 1.*

*Proof.* Let  $\text{net}, f_{\text{Sat}}, g_{\text{Sat}}^{(1)}, \dots, g_{\text{Sat}}^{(v)}$  be as in Equation (3). First, consider  $f_{\text{Sat}}(p_1, \dots, p_v) > 0$ . As a consequence of Lemma 1, the bounds  $\ell \leq f_{\text{Sat}}(p_1, \dots, p_v) \leq u$  produced by ComputeBounds converge towards  $f_{\text{Sat}}(p_1, \dots, p_v)$ , since ComputeBounds satisfies Definition 3. This implies that eventually  $\ell > 0$ , meaning that PV eventually proves  $f_{\text{Sat}}(p_1, \dots, p_v) \geq 0$ .

If  $f_{\text{Sat}}(p_1, \dots, p_v) < 0$  we eventually obtain  $u < 0$  with the same argument as above. Since  $u < 0$  disproves  $f_{\text{Sat}}(p_1, \dots, p_v) \geq 0$ , PV is complete for probabilistic verification problems satisfying Assumption 1.  $\square$

Theorem 2 shows that PV is complete when using a suitable set of heuristics. Unfortunately, the BaBSB heuristic is not suitable for Theorem 2, but provides the best empirical performance when applied in PV. However, BaBSB-LongestEdge-k satisfies Definition 4 and yields comparable performance as BaBSB, as we discuss in Appendix E.2.

## 6 Experiments

We apply our algorithms to verify the demographic parity fairness notion, count the number of safety violations of a neural network controller in a safety-critical system, and quantify the robustness of a neural network. Table 1 gives an overview of the benchmarks considered in this section. For all benchmarks, ProbabilityBounds use the SelectProb and BaBSB heuristics and CROWN [78] for ComputeBounds, while PV uses IntervalArithmetic. All verification problems are defined formally in Appendix A.

As our results show, PV outpaces the probabilistic verification algorithms FairSquare [4] and SpaceScanner [26]. Additionally, we show that ProbabilityBounds compares favourably to the ProVe\_SLR [47] and  $\varepsilon$ -ProVe [46] algorithms for #DNN verification [45], which corresponds to probabilistic verification with uniformly distributed inputs.

While no code is publicly available for SpaceScanner, running ProVe\_SLR is very computationally expensive. To enable a faithful comparison, we run our experiments on less powerful hardware (HW1) compared to the hardware used by Converse et al. [26] and Marzari et al. [47] and compare the runtime of our algorithms to the runtimes reported by these authors. To obtain comparable runtimes across the experiments in this section, we run all experiments on HW1. The results of FairSquare and  $\varepsilon$ -ProVe were similarly obtained on HW1.

To test the limits of PV, we introduce a new, challenging benchmark: MiniACSIncome is based on the ACSIncome dataset [27] and consists of datasets of varying input dimensionality, probability distributions for these datasets, and neural networks of varying sizes. PV solves seven of eight instances in MiniACSIncome within an hour.

**Hardware and Implementation.** We implement PV in Python, leveraging PyTorch [55] and auto\_LiRPA [74]. Our code and benchmarks are available at <https://github.com/sen-uni-kn/probspecs>. We run all experiments on a desktop running Ubuntu 22.04 with an Intel i7-4820K CPU,

Benchmark	Input Dimension	Input Distributions	Network Size	Source
FairSquare	2–3	independent 2 Bayesian Networks	$1 \times 1, 1 \times 2$	[4]
ACAS Xu	5	uniform	$6 \times 50$	[39]
MiniACSIncome	1–8	Bayesian Network	$1 \times 10$ – $10000$ $1 \times 10$ – $10 \times 10$	Own

Table 1: **Benchmarks.** Network size is the size of the neural network given as #layers $\times$ layer size.

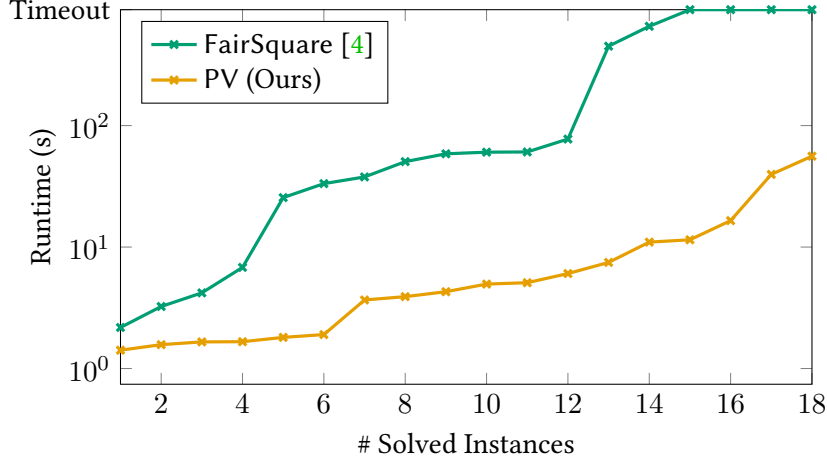


Figure 2: **FairSquare Benchmark**. The timeout for the FairSquare benchmark is 15min.

32 GB of memory, and no GPU (HW1). Appendix D.1 compares our hardware to the hardware used by Converse et al. [26] and Marzari et al. [47].

## 6.1 FairSquare Benchmark

Albarghouthi et al. [4] evaluate their FairSquare algorithm on an application derived from the Adult dataset [2]. In particular, they verify whether three small neural networks satisfy two fairness notions with respect to a person’s sex under three different distributions of the network input: a distribution of entirely independent univariate variables and two Bayesian Networks. The FairSquare benchmark assumes that all input variables except ‘sex’ are unbounded. We replicate these experiments to compare PV to FairSquare. Appendix D.2 contains more details on the FairSquare benchmark.

Figure 2 compares the runtimes of PV and FairSquare on the FairSquare benchmark. PV significantly outperforms FairSquare. In particular, PV solves four more instances than FairSquare within the timeout of 15 minutes. For the instances that both tools solve, the median runtime of PV is 4s (mean: 5s, max: 17s) compared to 44s for FairSquare (mean: 109s, max: 657s). Appendix D.2 contains the detailed results of this experiment.

## 6.2 ACAS Xu Safety

The ACAS Xu networks [39] are a suite of 45 networks, together forming a collision avoidance system for crewless aircraft. We reproduce the ACAS Xu #DNN verification [45] experiments of Marzari et al. [47]. In particular, we seek to *quantify* the number of violations (violation rate) of the ACAS Xu networks that violate the property  $\phi_2$  introduced by Katz et al. [39]. This corresponds to computing bounds on Equation (2) under a uniform distribution of  $\mathbf{x}$ .

We compare ProbabilityBounds (Algorithm 3) to the ProVe\_SLR and  $\varepsilon$ -ProVe algorithms for #DNN verification. ProVe\_SLR [47] computes the violation rate exactly, while  $\varepsilon$ -ProVe [46] computes an upper bound on the violation rate that is sound with a certain probability. In contrast, ProbabilityBounds provides sound bounds on the violation rate at any time during its execution. Therefore, we can run ProbabilityBounds only for a particular time budget or abort ProbabilityBounds when the bounds become sufficiently tight.

Table 2 compares ProbabilityBounds with ProVe\_SLR and  $\varepsilon$ -ProVe for the ACAS Xu property  $\phi_2$  [39] and the networks investigated by Marzari et al. [47]. For all three networks, ProbabilityBounds can tighten the bounds to a margin of less than 0.7% within one hour, while ProVe\_SLR requires at least four hours to compute the exact violation rate. In comparison to  $\varepsilon$ -ProVe, ProbabilityBounds

	ProbabilityBounds (Ours)			ProVe_SLR [47]		$\varepsilon$ -ProVe [46]	
	10s	1m	1h	Exact		99.9% confid.	
net	$\ell, u$	$\ell, u$	$\ell, u$	VR	Rt	$u$	Rt
$N_{4,3}$	0.17%, 2.92%	0.61%, 2.26%	1.12%, 1.75%	1.43%	8h 46m	3.61%	65s
$N_{4,9}$	0.00%, 3.31%	0.00%, 1.55%	0.08%, 0.29%	0.15%	12h 21m	0.73%	20s
$N_{5,8}$	0.90%, 4.13%	1.55%, 3.10%	1.97%, 2.57%	2.20%	4h 35m	4.52%	57s

Table 2: **Comparison of ProbabilityBounds, ProVe\_SLR, and  $\varepsilon$ -ProVe.** We run ProbabilityBounds with different time budgets (10s, 1m, 1h) and report the lower and upper bounds ( $\ell, u$ ) computed within this time budget. In contrast, ProVe\_SLR computes the exact probabilities (VR), and  $\varepsilon$ -ProVe computes a 99.9% confidence (confid.) upper bound. The runtimes (Rt) of ProVe\_SLR are taken from Marzari et al. [47].

produces tighter sound upper bounds within 10 seconds in two of three cases, while  $\varepsilon$ -ProVe requires at least 57 seconds to derive a probably sound upper bound for these cases. The extended comparison in Appendix D.3 reveals that in 13 from a total of 36 cases, ProbabilityBounds computes tighter sound bounds faster than  $\varepsilon$ -ProVe computes a *probably sound* upper bound.

### 6.3 ACAS Xu Robustness

We replicate the experiments of Converse et al. [26] who apply SpaceScanner to quantify the robustness of ACAS Xu network  $N_{1,1}$  [39] under adversarial perturbations. As Converse et al. [26], we consider 25 reference inputs — five for each class — and allow these reference inputs to be perturbed in the first two dimensions by at most 5% of the diameter of the input space in the respective dimension. To compute bounds on the output distribution, we bound the probability of each of the five ACAS Xu classes for each of the 25 inputs. Since the ACAS Xu training data is not publicly available, we sample the reference inputs randomly.

The mean runtime of ProbabilityBounds for these 125 verification problems is 21 seconds (median: 6s, maximum: 209s). Therefore, ProbabilityBounds substantially outperforms SpaceScanner, for which Converse et al. [26] report a mean runtime of 33 minutes per verification problem while running their experiments on superior hardware. Appendix D.4 contains the full results.

### 6.4 MiniACSIncome

To test the limits of PV, we introduce the MiniACSIncome benchmark. MiniACSIncome is derived from the ACSIncome dataset [27], a replacement of the Adult dataset [2] that is better suited for fair machine learning research. ACSIncome is based on US census data. The task is to predict whether a person’s yearly income exceeds \$50 000 using features such as the person’s age, sex, weekly work hours, education, etc. Our benchmark provides probabilistic verification problems of various degrees of difficulty. We apply PV to MiniACSIncome and show that it outperforms a baseline approach for solving MiniACSIncome.

**Benchmark.** To create probabilistic verification problems of increasing difficulty, we consider an increasing number of input variables from ACSIncome. The smallest instance, MiniACSIncome-1, only contains the binary ‘SEX’ variable. In contrast, the largest instance, MiniACSIncome-8, contains ‘SEX’ and seven more variables from ACSIncome, including age, education, and working hours per week. We train a neural network with a single layer of ten neurons for each MiniACSIncome- $i$ ,  $i \in \{1, \dots, 8\}$ . For MiniACSIncome-4, we additionally train deeper and wider networks to investigate the scalability of PV with respect to the network size. We fit a Bayesian Network to the MiniACSIncome-8 dataset to obtain an input distribution. We use this distribution for all

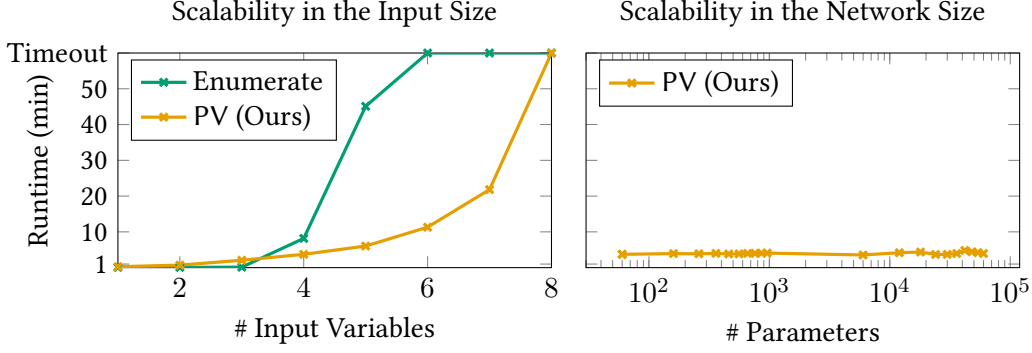


Figure 3: **MiniACSIncome Results.** The left plot depicts the runtime of PV and the baseline enumeration approach on MiniACSIncome-1 to MiniACSIncome-8 for a network of 10 neurons. The right plot depicts the runtime of PV for MiniACSIncome-4 networks of varying sizes. The timeout is one hour.

MiniACSIncome- $i$  instances by taking the variables not contained in MiniACSIncome- $i$  as latent variables. The verification problem is then to verify the demographic parity of a neural network with respect to ‘SEX’ under this input distribution. Appendix D.5 contains additional details on the MiniACSIncome benchmark.

**Baseline Approach.** Since all variables in ACSIncome are discrete, we can verify the demographic parity of a neural network by enumerating all discrete values in the input space of MiniACSIncome- $i$ . However, as expected, the number of discrete values scales exponentially with the number of input variables. While MiniACSIncome-1 contains only two values, MiniACSIncome-4 already contains 38 000 values, and MiniACSIncome-8 contains 2.4 billion values. As we demonstrate next, PV can explore this space more efficiently than the baseline enumeration approach.

**Results.** Figure 3 (left) displays the runtime of PV and the baseline enumeration approach for MiniACSIncome-1 to MiniACSIncome-8. While enumeration is faster than PV for input spaces of up to three variables, where the network can be evaluated for all discrete values in one batch, enumeration falls behind PV as soon as this becomes infeasible. PV can solve MiniACSIncome for up to seven input variables in less than 30 minutes, only exceeding the timeout of one hour for MiniACSIncome-8.

**Effect of Network Size.** Figure 3 (right) displays the runtime of PV for MiniACSIncome-4 networks of various sizes. The studied networks include wide single-layer networks of up to 10 000 neurons and deep networks of up to 10 layers of 10 neurons. As the figure shows, PV is largely unaffected by the size of the MiniACSIncome-4 networks. This is unexpected since the network size indirectly determines the performance of PV through the complexity of the decision boundary. However, larger networks need not necessarily have a more complex decision boundary, and large networks do not provide a performance benefit for MiniACSIncome-4, as discussed in Appendix D.5. Thoroughly exploring the impacts of network size requires more intricate datasets for which larger networks actually provide a benefit.

## 7 Conclusion and Future Work

As Section 6 shows, our PV algorithm for the probabilistic verification of neural networks significantly outpaces existing algorithms for probabilistic verification. We obtain this speedup by applying a massively parallel branch and bound algorithm based on bound propagation algorithms for neural networks. Our MiniACSIncome benchmark provides a challenging testbed for future probabilistic verification algorithms.

A promising direction for the probabilistic verification of neural networks is *neuron branching* [66], which fuels the current state-of-the-art non-probabilistic neural network verifiers. However, using neuron branching for probabilistic verification is challenging because it creates non-hyperrectangular regions in the input space that are expensive to integrate over. Alternative directions include new branching and splitting heuristics and using learning to select branches and split dimensions [44].

## References

- [1] Steven Adams, Andrea Patane, Morteza Lahijanian and Luca Laurenti. ‘BNN-DP: Robustness Certification of Bayesian Neural Networks via Dynamic Programming’. In: *ICML*. Vol. 202. Proceedings of Machine Learning Research. 2023. URL: <https://proceedings.mlr.press/v202/adams23a.html>.
- [2] *Adult*. UCI Machine Learning Repository. 1996. URL: <https://archive.ics.uci.edu/ml/datasets/Adult/>.
- [3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta and Masanori Koyama. ‘Optuna: A Next-generation Hyperparameter Optimization Framework’. In: *KDD*. 2019. DOI: [10.1145/3292500.3330701](https://doi.org/10.1145/3292500.3330701).
- [4] Aws Albarghouthi, Loris D’Antoni, Samuel Drews and Aditya V. Nori. ‘FairSquare: Probabilistic Verification of Program Fairness’. In: *Proc. ACM Program. Lang.* 1.OOPSLA (2017). DOI: [10.1145/3133904](https://doi.org/10.1145/3133904).
- [5] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja and Juan Pablo Vielma. ‘Strong mixed-integer programming formulations for trained neural networks’. In: *Math. Program.* 183.1 (2020). DOI: [10.1007/s10107-020-01474-5](https://doi.org/10.1007/s10107-020-01474-5).
- [6] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs and Taylor T. Johnson. ‘Improved Geometric Path Enumeration for Verifying ReLU Neural Networks’. In: *CAV (1)*. Vol. 12224. Lecture Notes in Computer Science. 2020. DOI: [10.1007/978-3-030-53288-8\\_4](https://doi.org/10.1007/978-3-030-53288-8_4).
- [7] Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S. Meel and Prateek Saxena. ‘Quantitative Verification of Neural Networks and Its Security Applications’. In: *CCS*. 2019. DOI: [10.1145/3319535.3354245](https://doi.org/10.1145/3319535.3354245).
- [8] Solon Barocas, Moritz Hardt and Arvind Narayanan. *Fairness and Machine Learning*. MIT Press, 2023. URL: <https://fairmlbook.org/>.
- [9] Osbert Bastani, Xin Zhang and Armando Solar-Lezama. ‘Probabilistic Verification of Fairness Properties via Concentration’. In: *Proc. ACM Program. Lang.* 3.OOPSLA (2019). DOI: [10.1145/3360544](https://doi.org/10.1145/3360544).
- [10] Ben Batten, Mehran Hosseini and Alessio Lomuscio. ‘Tight Verification of Probabilistic Robustness in Bayesian Neural Networks’. In: *AISTATS*. Vol. 238. Proceedings of Machine Learning Research. 2024. URL: <https://proceedings.mlr.press/v238/batten24a.html>.
- [11] Fabian Bauer-Marquart, David Boetius, Stefan Leue and Christian Schilling. ‘SpecRepair: Counter-Example Guided Safety Repair of Deep Neural Networks’. In: *SPIN*. Vol. 13255. Lecture Notes in Computer Science. 2022. DOI: [10.1007/978-3-031-15077-7\\_5](https://doi.org/10.1007/978-3-031-15077-7_5).
- [12] Vaishak Belle, Andrea Passerini and Guy Van den Broeck. ‘Probabilistic Inference in Hybrid Domains by Weighted Model Integration’. In: *IJCAI*. 2015. URL: <http://ijcai.org/Abstract/15/392>.



- [13] Leonard Berrada, Sumanth Dathathri, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Jonathan Uesato, Sven Gowal and M. Pawan Kumar. ‘Make Sure You’re Unsure: A Framework for Verifying Probabilistic Specifications’. In: *NeurIPS*. 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/5c5bc7df3d37b2a7ea29e1b47b2bd4ab-Abstract.html>.
- [14] Adel Bibi, Modar Alfadly and Bernard Ghanem. ‘Analytic Expressions for Probabilistic Moments of PL-DNN With Gaussian Input’. In: *CVPR*. 2018. DOI: [10.1109/CVPR.2018.00948](https://doi.org/10.1109/CVPR.2018.00948).
- [15] Christopher M. Bishop. *Pattern recognition and machine learning*. 5th ed. Information science and statistics. Springer, 2007. URL: <https://link.springer.com/book/10.1007/978-0-387-45528-0>.
- [16] Sumon Biswas and Hridesh Rajan. ‘Fairify: Fairness Verification of Neural Networks’. In: *ICSE*. 2023. DOI: [10.1109/ICSE48619.2023.00134](https://doi.org/10.1109/ICSE48619.2023.00134).
- [17] Georgian Borca-Tasciuc, Xingzhi Guo, Stanley Bak and Steven Skiena. ‘Provable Fairness for Neural Network Models Using Formal Verification’. In: *EWAF*. Vol. 3442. CEUR Workshop Proceedings. 2023. URL: <https://ceur-ws.org/Vol-3442/paper-04.pdf>.
- [18] Christopher Brix, Stanley Bak, Taylor T. Johnson and Haoze Wu. ‘The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results’. In: *CoRR* abs/2412.19985 (2024). DOI: [10.48550/ARXIV.2412.19985](https://doi.org/10.48550/ARXIV.2412.19985).
- [19] Christopher Brix, Stanley Bak, Changliu Liu and Taylor T. Johnson. ‘The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results’. In: *CoRR* abs/2312.16760 (2023). DOI: [10.48550/ARXIV.2312.16760](https://doi.org/10.48550/ARXIV.2312.16760).
- [20] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli and M. Pawan Kumar. ‘Branch and Bound for Piecewise Linear Neural Network Verification’. In: *J. Mach. Learn. Res.* 21 (2020). URL: <http://jmlr.org/papers/v21/19-468.html>.
- [21] Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr and M. Pawan Kumar. ‘Lagrangian Decomposition for Neural Network Verification’. In: *UAI*. Vol. 124. Proceedings of Machine Learning Research. 2020. URL: <http://proceedings.mlr.press/v124/bunel20a.html>.
- [22] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane and Matthew Wicker. ‘Statistical Guarantees for the Robustness of Bayesian Neural Networks’. In: *IJCAI*. 2019. URL: <https://doi.org/10.24963/ijcai.2019/789>.
- [23] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti and Andrea Patane. ‘Robustness Guarantees for Bayesian Inference with Gaussian Processes’. In: *AAAI*. 2019. DOI: [10.1609/AAAI.V33I01.33017759](https://doi.org/10.1609/AAAI.V33I01.33017759).
- [24] Nicholas Carlini and David A. Wagner. ‘Audio Adversarial Examples: Targeted Attacks on Speech-to-Text’. In: *IEEE Symposium on Security and Privacy Workshops*. 2018. DOI: [10.1109/SPW.2018.00009](https://doi.org/10.1109/SPW.2018.00009).
- [25] Chih-Hong Cheng, Georg Nührenberg and Harald Ruess. ‘Maximum Resilience of Artificial Neural Networks’. In: *ATVA*. Vol. 10482. Lecture Notes in Computer Science. 2017. DOI: [10.1007/978-3-319-68167-2\\_18](https://doi.org/10.1007/978-3-319-68167-2_18).
- [26] Hayes Converse, Antonio Filieri, Divya Gopinath and Corina S. Pasareanu. ‘Probabilistic Symbolic Analysis of Neural Networks’. In: *ISSRE*. 2020. DOI: [10.1109/ISSRE5003.2020.00023](https://doi.org/10.1109/ISSRE5003.2020.00023).
- [27] Frances Ding, Moritz Hardt, John Miller and Ludwig Schmidt. ‘Retiring Adult: New Datasets for Fair Machine Learning’. In: *NeurIPS*. 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/32e54441e6382a7fbacbbbf3c450059-Abstract.html>.
- [28] Alice Doherty, Matthew Wicker, Luca Laurenti and Andrea Patane. ‘Individual Fairness in Bayesian Neural Networks’. In: *CoRR* abs/2304.10828 (2023). DOI: [10.48550/ARXIV.2304.10828](https://doi.org/10.48550/ARXIV.2304.10828).

- [29] Hai Duong, Linhan Li, ThanhVu Nguyen and Matthew Dwyer. ‘A DPLL(T) Framework for Verifying Deep Neural Networks’. In: *CoRR* (2023). URL: <https://arxiv.org/abs/2307.10266>.
- [30] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold and Richard S. Zemel. ‘Fairness Through Awareness’. In: *ITCS*. 2012. DOI: [10.1145/2090236.2090255](https://doi.org/10.1145/2090236.2090255).
- [31] Javid Ebrahimi, Anyi Rao, Daniel Lowd and Dejing Dou. ‘HotFlip: White-Box Adversarial Examples for Text Classification’. In: *ACL* (2). 2018. DOI: [10.18653/V1/P18-2006](https://doi.org/10.18653/V1/P18-2006).
- [32] European Parliament. *EU AI Act: first regulation on artificial intelligence*. 2023. URL: <https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence> (visited on 19/05/2024).
- [33] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger and Suresh Venkatasubramanian. ‘Certifying and Removing Disparate Impact’. In: *KDD*. 2015. DOI: [10.1145/2783258.2783311](https://doi.org/10.1145/2783258.2783311).
- [34] Sainyam Galhotra, Yuriy Brun and Alexandra Meliou. ‘Fairness Testing: Testing Software for Discrimination’. In: *ESEC/SIGSOFT FSE*. 2017. DOI: [10.1145/3106237.3106277](https://doi.org/10.1145/3106237.3106277).
- [35] Dan Hendrycks and Thomas G. Dietterich. ‘Benchmarking Neural Network Robustness to Common Corruptions and Perturbations’. In: *ICLR*. 2019. URL: <https://openreview.net/forum?id=HJz6tiCqYm>.
- [36] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt and Dawn Song. ‘Natural Adversarial Examples’. In: *CVPR*. 2021. DOI: [10.1109/CVPR46437.2021.01501](https://doi.org/10.1109/CVPR46437.2021.01501).
- [37] Hossein Hosseini, Baicen Xiao and Radha Poovendran. ‘Google’s Cloud Vision API is Not Robust to Noise’. In: *ICMLA*. 2017. DOI: [10.1109/ICMLA.2017.0-172](https://doi.org/10.1109/ICMLA.2017.0-172).
- [38] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian and Mykel J. Kochenderfer. ‘Towards Proving the Adversarial Robustness of Deep Neural Networks’. In: *FVAV@iFM*. Vol. 257. EPTCS. 2017. DOI: [10.4204/EPTCS.257.3](https://doi.org/10.4204/EPTCS.257.3).
- [39] Guy Katz, Clark W. Barrett, David L. Dill, Kyle D. Julian and Mykel J. Kochenderfer. ‘Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks’. In: *CAV* (1). Vol. 10426. Lecture Notes in Computer Science. 2017. DOI: [10.1007/978-3-319-63387-9\\_5](https://doi.org/10.1007/978-3-319-63387-9_5).
- [40] Diederik P. Kingma and Jimmy Ba. ‘Adam: A Method for Stochastic Optimization’. In: *ICLR*. 2015. DOI: [10.48550/ARXIV.1412.6980](https://doi.org/10.48550/ARXIV.1412.6980).
- [41] Matt J. Kusner, Joshua R. Loftus, Chris Russell and Ricardo Silva. ‘Counterfactual Fairness’. In: *NIPS*. 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html>.
- [42] Ailsa H. Land and Alison G. Doig. ‘An Automatic Method for Solving Discrete Programming Problems’. In: *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. First published in *Econometrica* 28 (1960). 2010. DOI: [10.1007/978-3-540-68279-0\\_5](https://doi.org/10.1007/978-3-540-68279-0_5).
- [43] Augustin Lemesle, Julien Lehmann and Tristan Le Gall. ‘Neural Network Verification with PyRAT’. In: *CoRR* abs/2410.23903 (2024). DOI: [10.48550/ARXIV.2410.23903](https://doi.org/10.48550/ARXIV.2410.23903).
- [44] Jingyue Lu and M. Pawan Kumar. ‘Neural Network Branching for Neural Network Verification’. In: *ICLR*. 2020. URL: <https://openreview.net/forum?id=B1evfa4tPB>.
- [45] Luca Marzari, Davide Corsi, Ferdinando Cicalese and Alessandro Farinelli. ‘The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks’. In: *IJCAI*. 2023. DOI: [10.24963/IJCAI.2023/25](https://doi.org/10.24963/IJCAI.2023/25).
- [46] Luca Marzari, Davide Corsi, Enrico Marchesini, Alessandro Farinelli and Ferdinando Cicalese. ‘Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees’. In: *AAAI*. 2024. DOI: [10.1609/AAAI.V38I19.30134](https://doi.org/10.1609/AAAI.V38I19.30134).

- [47] Luca Marzari, Gabriele Roncolato and Alessandro Farinelli. ‘Scaling #DNN-Verification Tools with Efficient Bound Propagation and Parallel Computing’. In: *AIRO@AI\*IA*. Vol. 3686. CEUR Workshop Proceedings. 2023. URL: <https://ceur-ws.org/Vol-3686/paper2.pdf>.
- [48] Kiarash Mohammadi, Aishwarya Sivaraman and Golnoosh Farnadi. ‘FETA: Fairness Enforced Verifying, Training, and Predicting Algorithms for Neural Networks’. In: *EAAMO*. 2023. DOI: [10.1145/3617694.3623243](https://doi.org/10.1145/3617694.3623243).
- [49] Ramon E. Moore, R. Baker Kearfott and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009. DOI: [10.1137/1.9780898717716](https://doi.org/10.1137/1.9780898717716).
- [50] Paolo Morettin, Andrea Passerini and Roberto Sebastiani. ‘A Unified Framework for Probabilistic Verification of AI Systems via Weighted Model Integration’. In: *CoRR* abs/2402.04892 (2024). DOI: [10.48550/ARXIV.2402.04892](https://doi.org/10.48550/ARXIV.2402.04892).
- [51] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe and Edward C. Sewell. ‘Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning’. In: *Discret. Optim.* 19 (2016). DOI: [10.1016/j.disopt.2016.01.005](https://doi.org/10.1016/j.disopt.2016.01.005).
- [52] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel and Martin T. Vechev. ‘PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations’. In: *Proc. ACM Program. Lang.* 6.POPL (2022). URL: <https://doi.org/10.1145/3498704>.
- [53] Radford M. Neal. *Bayesian learning for neural networks*. Vol. 118. Lecture Notes in Statistics. Springer Science & Business Media, 1996. DOI: <https://doi.org/10.1007/978-1-4612-0745-0>.
- [54] Alessandro De Palma, Harkirat S. Behl, Rudy Bunel, Philip H. S. Torr and M. Pawan Kumar. ‘Scaling the Convex Barrier with Active Sets’. In: *ICLR*. 2021. URL: <https://openreview.net/forum?id=uQfOy7LrlTR>.
- [55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala. ‘PyTorch: An Imperative Style, High-Performance Deep Learning Library’. In: *NeurIPS*. 2019. URL: <https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>.
- [56] Dino Pedreschi, Salvatore Ruggieri and Franco Turini. ‘Discrimination-aware data mining’. In: *KDD*. 2008. DOI: [10.1145/1401890.1401959](https://doi.org/10.1145/1401890.1401959).
- [57] Luca Pulina and Armando Tacchella. ‘An Abstraction-Refinement Approach to Verification of Artificial Neural Networks’. In: *CAV*. Vol. 6174. Lecture Notes in Computer Science. 2010. DOI: [10.1007/978-3-642-14295-6\\_24](https://doi.org/10.1007/978-3-642-14295-6_24).
- [58] Wenjie Ruan, Xiaowei Huang and Marta Kwiatkowska. ‘Reachability Analysis of Deep Neural Networks with Provable Guarantees’. In: *IJCAI*. 2018. DOI: [10.24963/ijcai.2018/368](https://doi.org/10.24963/ijcai.2018/368).
- [59] Anian Ruoss, Mislav Balunovic, Marc Fischer and Martin T. Vechev. ‘Learning Certified Individually Fair Representations’. In: *NeurIPS*. 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/55d491cf951b1b920900684d71419282-Abstract.html>.
- [60] Gagandeep Singh, Timon Gehr, Markus Püschel and Martin T. Vechev. ‘An Abstract Domain for Certifying Neural Networks’. In: *Proc. ACM Program. Lang.* 3.POPL (2019). DOI: [10.1145/3290354](https://doi.org/10.1145/3290354).
- [61] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow and Rob Fergus. ‘Intriguing properties of neural networks’. In: *ICLR (Poster)*. 2014. DOI: [10.48550/ARXIV.1312.6199](https://doi.org/10.48550/ARXIV.1312.6199).
- [62] Vincent Tjeng, Kai Y. Xiao and Russ Tedrake. ‘Evaluating Robustness of Neural Networks with Mixed Integer Programming’. In: *ICLR (Poster)*. 2019. URL: <https://openreview.net/forum?id=HyGIIdRqtm>.

- [63] Hoang-Dung Tran, Stanley Bak, Weiming Xiang and Taylor T. Johnson. ‘Verification of Deep Convolutional Neural Networks Using ImageStars’. In: *CAV (1)*. Vol. 12224. Lecture Notes in Computer Science. 2020. doi: [10.1007/978-3-030-53288-8\\_2](https://doi.org/10.1007/978-3-030-53288-8_2).
- [64] Hoang-Dung Tran, Xiaodong Yang, Diego Manzananas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak and Taylor T. Johnson. ‘NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems’. In: *CAV (1)*. Vol. 12224. Lecture Notes in Computer Science. 2020. doi: [10.1007/978-3-030-53288-8\\_1](https://doi.org/10.1007/978-3-030-53288-8_1).
- [65] Caterina Urban, Maria Christakis, Valentin Wüstholtz and Fuyuan Zhang. ‘Perfectly Parallel Fairness Certification of Neural Networks’. In: *Proc. ACM Program. Lang.* 4.OOPSLA (2020). doi: [10.1145/3428253](https://doi.org/10.1145/3428253).
- [66] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang and Suman Jana. ‘Efficient Formal Safety Analysis of Neural Networks’. In: *NeurIPS*. 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html>.
- [67] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang and Suman Jana. ‘Formal Security Analysis of Neural Networks using Symbolic Intervals’. In: *USENIX Security Symposium*. 2018. URL: <https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi>.
- [68] Tsui-Wei Weng, Pin-Yu Chen, Lam M. Nguyen, Mark S. Squillante, Akhilan Boopathy, Ivan V. Oseledets and Luca Daniel. ‘PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach’. In: *ICML*. Vol. 97. Proceedings of Machine Learning Research. 2019. URL: <http://proceedings.mlr.press/v97/weng19a.html>.
- [69] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning and Inderjit S. Dhillon. ‘Towards Fast Computation of Certified Robustness for ReLU Networks’. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. 2018. URL: <http://proceedings.mlr.press/v80/weng18a.html>.
- [70] Matthew Wicker, Luca Laurenti, Andrea Patane and Marta Kwiatkowska. ‘Probabilistic Safety for Bayesian Neural Networks’. In: *UAI*. Vol. 124. Proceedings of Machine Learning Research. 2020. URL: <http://proceedings.mlr.press/v124/wicker20a.html>.
- [71] Matthew Wicker, Andrea Patane, Luca Laurenti and Marta Kwiatkowska. ‘Adversarial Robustness Certification for Bayesian Neural Networks’. In: *FM*. Vol. 14933. Lecture Notes in Computer Science. 2024. doi: [10.1007/978-3-031-71162-6\\_1](https://doi.org/10.1007/978-3-031-71162-6_1).
- [72] Eric Wong and J. Zico Kolter. ‘Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope’. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. 2018. URL: <http://proceedings.mlr.press/v80/wong18a.html>.
- [73] Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz and Clark Barrett. ‘Marabou 2.0: A Versatile Formal Analyzer of Neural Networks’. In: *CoRR* abs/2401.14461 (2024). doi: [10.48550/arXiv.2401.14461](https://doi.org/10.48550/arXiv.2401.14461).
- [74] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kaillkhura, Xue Lin and Cho-Jui Hsieh. ‘Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond’. In: *NeurIPS*. 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/0cbc5671ae26f67871cb914d81ef8fc1-Abstract.html>.
- [75] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin and Cho-Jui Hsieh. ‘Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers’. In: *ICLR*. 2021. URL: <https://openreview.net/forum?id=nVZtXBI6LNn>.

- [76] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane S. Boning and Cho-Jui Hsieh. ‘Towards Stable and Efficient Training of Verifiably Robust Neural Networks’. In: *ICLR*. 2020. URL: <https://openreview.net/forum?id=Skxuk1rFwB>.
- [77] 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: *NeurIPS*. 2022. URL: <https://openreview.net/forum?id=5haAJAcofjc>.
- [78] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh and Luca Daniel. ‘Efficient Neural Network Robustness Certification with General Activation Functions’. In: *NeurIPS*. 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html>.
- [79] Duo Zhou, Christopher Brix, Grani A. Hanasusanto and Huan Zhang. ‘Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes’. In: *NeurIPS*. 2024. URL: <https://openreview.net/forum?id=FwhM1Zpyft>.

## A Probabilistic Verification Problems

This section contains the formal definitions of all probabilistic verification problems in this paper. First of all, we provide a detailed derivation of Example 1

**Example** (Derivation of Example 1). Let  $\mathcal{X} \subseteq \mathbb{R}^n$ ,  $A \subset \{1, \dots, n\}$ ,  $a \in A$ ,  $\text{net} : \mathbb{R}^n \rightarrow \mathbb{R}^2$  and  $\gamma$  be as in Example 1. We rewrite Equation (1) as

$$\begin{aligned}
& \frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is disadvantaged}]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is advantaged}]} \geq \gamma \\
\iff & \frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \mid \mathbf{x}_a \leq 0]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \mid \mathbf{x}_a \geq 1]} \geq \gamma \\
\iff & \frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \wedge \mathbf{x}_a \leq 0] / \mathbb{P}_{\mathbf{x}}[\mathbf{x}_a \leq 0]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \wedge \mathbf{x}_a \geq 1] / \mathbb{P}_{\mathbf{x}}[\mathbf{x}_a \geq 1]} \geq \gamma \\
\iff & \frac{\mathbb{P}_{\mathbf{x}}[\min(\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2, -\mathbf{x}_a) \geq 0] / \mathbb{P}_{\mathbf{x}}[-\mathbf{x}_a \geq 0]}{\mathbb{P}_{\mathbf{x}}[\min(\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2, \mathbf{x}_a - 1) \geq 0] / \mathbb{P}_{\mathbf{x}}[\mathbf{x}_a - 1 \geq 0]} \geq \gamma \\
\iff & \frac{\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}^{(1)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0] / \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}^{(2)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]}{\mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}^{(3)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0] / \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}^{(4)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0]} - \gamma \geq 0 \\
\iff & f_{\text{Sat}}\left(\mathbb{P}_{\mathbf{x}}\left[g_{\text{Sat}}^{(1)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0\right], \dots, \mathbb{P}_{\mathbf{x}}\left[g_{\text{Sat}}^{(4)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0\right]\right) \geq 0
\end{aligned}$$

where  $f_{\text{Sat}}, g_{\text{Sat}}^{(1)}, g_{\text{Sat}}^{(2)}, g_{\text{Sat}}^{(3)}$ , and  $g_{\text{Sat}}^{(4)}$  are as in Example 1.

### A.1 Parity of Qualified Persons

The following probabilistic verification problem concerns verifying the parity of qualified persons, a variant of demographic parity that only considers the subpopulation of persons qualified for, for example, hiring [4]. Let  $\mathcal{X} \subseteq \mathbb{R}^n$ ,  $A \subset \{1, \dots, n\}$ ,  $a \in A$ , and  $\text{net} : \mathbb{R}^n \rightarrow \mathbb{R}^2$  be as in Example 1. Additionally, let  $q \in \{1, \dots, n\} \setminus A$  and  $\hat{q} \in \mathbb{R}$ , such that persons with  $\mathbf{x}_q \geq \hat{q}$  are considered to be qualified. In their extended set of experiments, Albarghouthi et al. [4] consider a  $q$  that encodes age and  $\hat{q} = 18$  so that only persons who are at least 18 years old are considered to be qualified. The parity of qualified persons fairness notion is

$$\begin{aligned}
& \frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is disadvantaged} \wedge \mathbf{x} \text{ is qualified}]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x}) = \text{yes} \mid \mathbf{x} \text{ is advantaged} \wedge \mathbf{x} \text{ is qualified}]} \geq \gamma \\
\iff & \frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \mid \mathbf{x}_a \leq 0 \wedge \mathbf{x}_q \geq \hat{q}]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \mid \mathbf{x}_a \geq 1 \wedge \mathbf{x}_q \geq \hat{q}]} \geq \gamma \\
\iff & \frac{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \mid \min(-\mathbf{x}_a, \mathbf{x}_q - \hat{q}) \geq 0]}{\mathbb{P}_{\mathbf{x}}[\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2 \geq 0 \mid \min(\mathbf{x}_a - 1, \mathbf{x}_q - \hat{q}) \geq 0]} \geq \gamma \\
\iff & f_{\text{Sat}}\left(\mathbb{P}_{\mathbf{x}}\left[g_{\text{Sat}}^{(1)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0\right], \dots, \mathbb{P}_{\mathbf{x}}\left[g_{\text{Sat}}^{(4)}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0\right]\right) \geq 0
\end{aligned}$$

where  $\gamma \in [0, 1]$ ,  $f_{\text{Sat}}(p_1, p_2, p_3, p_4) = (p_1 p_4) / (p_2 p_3) - \gamma$ ,  $g_{\text{Sat}}^{(1)}(\mathbf{x}, \text{net}(\mathbf{x})) = \min(\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2, -\mathbf{x}_a, \mathbf{x}_q - \hat{q})$ ,  $g_{\text{Sat}}^{(2)}(\mathbf{x}, \text{net}(\mathbf{x})) = \min(-\mathbf{x}_a, \mathbf{x}_q - \hat{q})$ ,  $g_{\text{Sat}}^{(3)}(\mathbf{x}, \text{net}(\mathbf{x})) = \min(\text{net}(\mathbf{x})_1 - \text{net}(\mathbf{x})_2, \mathbf{x}_a - 1, \mathbf{x}_q - \hat{q})$ , and  $g_{\text{Sat}}^{(4)}(\mathbf{x}, \text{net}(\mathbf{x})) = \min(\mathbf{x}_a - 1, \mathbf{x}_q - \hat{q})$ .

### A.2 ACAS Xu Safety

Next, we consider Equation (2) for an ACAS Xu network, where to be safe means satisfying property  $\phi_2$  of Katz et al. [39]. For quantifying the number of violations, we first define what



it means for an ACAS Xu neural network  $\text{net} : \mathbb{R}^5 \rightarrow \mathbb{R}^5$  to violate  $\phi_2$ . Using the satisfaction functions of Bauer-Marquart et al. [11], violating  $\phi_2$  means

$$g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) = \max_{i=1}^5 \text{net}(\mathbf{x})_i - \text{net}(\mathbf{x})_1 < 0 \quad \forall \mathbf{x} \in \mathcal{X}_{\phi_2} \cap \mathcal{X}, \quad (8)$$

where  $\mathcal{X}$  is the bounded hyperrectangular input space of  $\text{net}$  and

$$\mathcal{X}_{\phi_2} = [55947.961, \infty] \times \mathbb{R}^2 \times [1145, \infty] \times [-\infty, 60].$$

We refer to Katz et al. [39] for an interpretation of  $\phi_2$  in the application context. Quantifying the number of violations with respect to  $\phi_2$  corresponds to computing

$$\ell \leq \mathbb{P}_{\mathbf{x}}[g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) < 0] = \mathbb{P}_{\mathbf{x}}[-g_{\text{Sat}}(\mathbf{x}, \text{net}(\mathbf{x})) \geq 0] \leq u,$$

where  $g_{\text{Sat}}$  is as in Equation (8) and  $\mathbf{x}$  is uniformly distributed on  $\mathcal{X}_{\phi_2} \cap \mathcal{X}$  with all points outside  $\mathcal{X}_{\phi_2} \cap \mathcal{X}$  having zero probability.

### A.3 ACAS Xu Robustness

For the ACAS Xu robustness experiment in Section 6.3, we solve five probabilistic verification problems for each reference input  $\mathbf{x}$  — one for each of the five classes. Our goal is to bound the probability of  $\text{net}$  classifying an input  $\mathbf{x}'$  as class  $i \in \{1, \dots, 5\}$ , where  $\mathbf{x}'$  is close to the reference input  $\mathbf{x}$  in the first two dimensions and identical to  $\mathbf{x}$  in the remaining dimensions.

Let  $\text{net}$  be the ACAS Xu network  $N_{1,1}$  of Katz et al. [39] with input space  $\mathcal{X} = [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ . Let  $\mathbf{x}$  be a reference input. Note that the ACAS Xu networks assign the class with the *minimal* score to an input instead of using the maximal score. For bounding the probability of obtaining class  $i \in \{1, \dots, 5\}$  for inputs close to  $\mathbf{x}$ , we compute bounds on

$$\mathbb{P}_{\mathbf{x}'}[g_{\text{Sat}}(\mathbf{x}', \text{net}(\mathbf{x}')) \geq 0],$$

where  $g_{\text{Sat}}(\mathbf{x}', \text{net}(\mathbf{x}')) = \min_{j=1}^5 \text{net}(\mathbf{x}')_j - \text{net}(\mathbf{x}')_i$  and  $\mathbf{x}'$  is uniformly distributed on the set

$$\mathcal{X} \cap ([\mathbf{x}_{1:2} - 0.05 \cdot \mathbf{w}_{1:2}, \mathbf{x}_{1:2} + 0.05 \cdot \mathbf{w}_{1:2}] \times \{\mathbf{x}_{3:5}\}),$$

where  $\mathbf{w} = \bar{\mathbf{x}} - \underline{\mathbf{x}}$  and  $\mathbf{z}_{i:j}$  is the vector containing the elements  $i, \dots, j$  of a vector  $\mathbf{z}$ .

## B Interval Arithmetic

This section introduces additional interval arithmetic bounding rules for linear functions, multiplication, and division, complementing the interval arithmetic bounding rules for monotone functions in Section 3.2.1. Furthermore, we provide Theorems 5.1 and 6.1 Moore et al. [49] for reference and provide a proof that IntervalArithmetic satisfies Definition 3. These results provide the foundation for the theoretical analysis of CROWN in Appendix C.

### B.1 Bounding Rules

Let  $f^{(k)}$  be as in Section 3.2.1. First, consider the multiplication of two scalars, that is,  $f^{(k)}(z, w) = zw$  where  $\underline{z} \leq z \leq \bar{z}$  and  $\underline{w} \leq w \leq \bar{w}$ . We have

$$\min(\underline{z}\underline{w}, \underline{z}\bar{w}, \bar{z}\underline{w}, \bar{z}\bar{w}) \leq z_1 z_2 \leq \max(\underline{z}\underline{w}, \underline{z}\bar{w}, \bar{z}\underline{w}, \bar{z}\bar{w}).$$

For the element-wise multiplication of vectors, we apply the above rule to each element separately. Multiplication of several arguments can be rewritten as several multiplications of two arguments.

Now, consider computing bounds of the reciprocal  $f^{(k)}(z) = \frac{1}{z}$  with  $\underline{z} \leq z \leq \bar{z}$ . We differentiate the following cases

$$\begin{aligned} \frac{1}{\bar{z}} &\leq \frac{1}{z} & \text{if } 0 \notin (\underline{z}, \bar{z}] & \quad \frac{1}{z} &\leq \frac{1}{\underline{z}} & \text{if } 0 \notin [\underline{z}, \bar{z}) \\ -\infty &\leq \frac{1}{z} & \text{if } 0 \in (\underline{z}, \bar{z}] & \quad \frac{1}{z} &\leq \infty & \text{if } 0 \in [\underline{z}, \bar{z}). \end{aligned}$$

Using bounds on the reciprocal, we can compute bounds on a division by rewriting division as multiplication by the reciprocal. Lastly, for an affine function  $f^{(k)}(\mathbf{z}) = \mathbf{W}\mathbf{z} + \mathbf{b}$  where  $\underline{\mathbf{z}} \leq \mathbf{z} \leq \bar{\mathbf{z}}$ , we have

$$[\mathbf{W}]^+ \underline{\mathbf{z}} + [\mathbf{W}]^- \bar{\mathbf{z}} + \mathbf{b} \leq \mathbf{W}\mathbf{z} + \mathbf{b} \leq [\mathbf{W}]^+ \bar{\mathbf{z}} + [\mathbf{W}]^- \underline{\mathbf{z}} + \mathbf{b}, \quad (9)$$

where  $[\mathbf{W}]_{i,j}^+ = \max(0, \mathbf{W}_{i,j})$  and  $[\mathbf{W}]_{i,j}^- = \min(0, \mathbf{W}_{i,j})$ .

## B.2 Theoretical Properties

We include Theorems 5.1 and 6.1 of Moore et al. [49] and relevant definitions for reference. Let  $\mathbb{H}^n = \{[\underline{\mathbf{x}}, \bar{\mathbf{x}}] \mid \underline{\mathbf{x}}, \bar{\mathbf{x}} \in \mathbb{R}^n, \underline{\mathbf{x}} \leq \bar{\mathbf{x}}\}$  be the set of hyperrectangles in  $\mathbb{R}^n$ . Let  $w : 2^{\mathbb{R}^n} \rightarrow \mathbb{R}_{\geq 0}$  with

$$w(\mathcal{X}) = \max_{i \in \{1, \dots, n\}} (\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x}_i - \min_{\mathbf{x} \in \mathcal{X}} \mathbf{x}_i) = \max_{\mathbf{x}, \mathbf{x}' \in \mathcal{X}} \|\mathbf{x} - \mathbf{x}'\|_\infty$$

be the *width* of the set  $\mathcal{X} \subseteq \mathbb{R}^n$ . We denote the *image* of a hyperrectangle  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  under  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  as  $f([\underline{\mathbf{x}}, \bar{\mathbf{x}}]) = \{f(\mathbf{x}) \mid \mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]\}$ .

### B.2.1 Definitions

Theorem 5.1 of Moore et al. [49] applies to inclusion isotonic interval extensions as defined below.

**Definition 6.** Let  $F : \mathbb{H}^n \rightarrow \mathbb{H}^m$  and  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ .

- $F$  is an *interval extensions* of  $f$  if  $\forall \mathbf{x} \in \mathbb{R}^n : F([\underline{\mathbf{x}}, \mathbf{x}]) = [f(\mathbf{x}), f(\mathbf{x})]$ .
- $F$  is *inclusion isotonic* if  $\forall [\underline{\mathbf{x}}, \bar{\mathbf{x}}], [\underline{\mathbf{x}'}, \bar{\mathbf{x}'}] \in \mathbb{H}^n, [\underline{\mathbf{x}'}, \bar{\mathbf{x}'}] \subseteq [\underline{\mathbf{x}}, \bar{\mathbf{x}}] : F([\underline{\mathbf{x}'}, \bar{\mathbf{x}'}]) \subseteq F([\underline{\mathbf{x}}, \bar{\mathbf{x}}])$ .

Theorem 6.1 of Moore et al. [49] requires an inclusion isotonic *Lipschitz* interval extension.

**Definition 7.** Let  $F : \mathbb{H}^n \rightarrow \mathbb{H}^m$  be an interval extension of  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ .  $F$  is *Lipschitz* if there exists an  $L \in \mathbb{R}$  such that  $\forall [\underline{\mathbf{x}}, \bar{\mathbf{x}}] \in \mathbb{H}^n : w(F([\underline{\mathbf{x}}, \bar{\mathbf{x}}])) \leq Lw([\underline{\mathbf{x}}, \bar{\mathbf{x}}])$ .

IntervalArithmetic, as introduced in Section 3.2.1, corresponds to *natural interval extensions* in Moore et al. [49]. As Moore et al. [49] show, natural interval extensions — and, therefore, IntervalArithmetic — satisfy Definitions 6 and 7.

### B.2.2 Theorems and Propositions

Theorem 5.1 of Moore et al. [49] is known as the fundamental theorem of interval analysis. It proves that IntervalArithmetic is sound. We also provide a proof for the well-known property that IntervalArithmetic satisfies Definition 3. This result is closely related to Theorem 6.1 of Moore et al. [49], which we also include for reference.

**Theorem 3** (Theorem 5.1 of Moore et al. [49]). *If  $F : \mathbb{H}^n \rightarrow \mathbb{H}^m$  is an inclusion isotonic interval extension of  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , we have  $f([\underline{\mathbf{x}}, \bar{\mathbf{x}}]) \subseteq F([\underline{\mathbf{x}}, \bar{\mathbf{x}}])$  for every  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}] \in \mathbb{H}^n$ .*

**Theorem 4** (Theorem 6.1 of Moore et al. [49]). Let  $F : \mathbb{H}^n \rightarrow \mathbb{H}^m$  be an inclusion isotonic Lipschitz interval extension of  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . Let  $\mathcal{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \in \mathbb{H}^n$ . We define the  $M$ -step uniform subdivision of  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$  with  $M \in \mathbb{N}$  as

$$\mathcal{X}_{i,j} = \left[ \underline{\mathbf{x}}_i + (j-1) \frac{w([\underline{\mathbf{x}}_i, \overline{\mathbf{x}}_i])}{M}, \underline{\mathbf{x}}_i + j \frac{w([\underline{\mathbf{x}}_i, \overline{\mathbf{x}}_i])}{M} \right], \quad j \in \{1, \dots, M\}.$$

Further, let

$$F^{(M)}([\underline{\mathbf{x}}, \overline{\mathbf{x}}]) = \bigcup_{j_i=1}^M F(\mathcal{X}_{1,j_1} \times \dots \times \mathcal{X}_{n,j_n}).$$

It holds that

$$w(F^{(M)}([\underline{\mathbf{x}}, \overline{\mathbf{x}}])) - w(f([\underline{\mathbf{x}}, \overline{\mathbf{x}}])) \leq 2L \frac{w(\mathcal{X})}{M},$$

where  $L$  is the Lipschitz constant of  $F$ .

**Proposition 4.** Every Lipschitz interval extension satisfies Definition 3.

*Proof.* Let  $F : \mathbb{H}^n \rightarrow \mathbb{H}^m$  be a Lipschitz interval extension with Lipschitz constant  $L$ . We write  $[\underline{\mathbf{y}}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]}, \overline{\mathbf{y}}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]}] = F([\underline{\mathbf{x}}, \overline{\mathbf{x}}])$  for  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}] \in \mathbb{H}^n$ . Using that  $F$  is Lipschitz, we obtain

$$\lim_{\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\| \rightarrow 0} \|\overline{\mathbf{y}}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]} - \underline{\mathbf{y}}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]}\| = \lim_{\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\| \rightarrow 0} w(F([\underline{\mathbf{x}}, \overline{\mathbf{x}}])) \leq \lim_{\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\| \rightarrow 0} Lw([\underline{\mathbf{x}}, \overline{\mathbf{x}}]) = 0.$$

Further, since  $F$  is an interval extension, we have  $\|\overline{\mathbf{y}}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]} - \underline{\mathbf{y}}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]}\| = 0$  for any  $\mathbf{x} \in \mathbb{R}^n$ . This proves that  $F$  satisfies Definition 3.  $\square$

**Corollary 3.** IntervalArithmetic satisfies Definition 3.

## C Theoretical Analysis of CROWN

In this section, we prove that CROWN [78] satisfies Definition 3 (Proposition 1). In fact, we provide a slightly stronger result showing that CROWN possesses the properties of IntervalArithmetic that follow from Theorem 5.1 and 6.1 of Moore et al. [49]. Concretely, we prove that the bounds computed by CROWN are always contained in the bounds computed by an inclusion isotonic Lipschitz interval extension, such that Proposition 4 and Theorems 3 and 4 apply. Since the bounds computed by this interval extension converge and they contain the bounds computed by CROWN, the bounds computed by CROWN also need to converge, establishing that CROWN satisfies Definition 3. We first revisit CROWN, discuss why CROWN itself is not inclusion isotonic, and finally provide the inclusion isotonic Lipschitz interval extension that is guaranteed to compute looser bounds than CROWN.

In the following, we restrict our attention to ReLU-activated fully-connected neural networks since these are the most relevant for this paper. However, similar arguments to the arguments we make below can also be made for other architectures and activation functions.

### C.1 CROWN

We revisit CROWN for ReLU-activated fully-connected neural networks. Applying CROWN for other activation functions and architectures is described by Zhang et al. [78] and Xu et al. [74]. Let  $\text{net} : \mathcal{X} \rightarrow \mathbb{R}^m$  be a ReLU-activated fully-connected neural network with  $K \in \mathbb{N}$  layers, that is,  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$ , where  $f^{(k)} : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_{k+1}}$  is either an affine function or ReLU. We

---

**Algorithm 4:** CROWN

---

**Input:** Neural Network  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$ , Input Bounds  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ , Layer Bounds  $[\underline{\mathbf{z}}^{(1)}, \overline{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(K-1)}, \overline{\mathbf{z}}^{(K-1)}]$

- 1  $\underline{\mathbf{A}}^{(K)} \leftarrow \mathbf{I}, \overline{\mathbf{A}}^{(K)} \leftarrow \mathbf{I}$  ( $\mathbf{I} \in \mathbb{R}^{m \times m}$  is the identity matrix)
- 2  $\underline{\mathbf{d}}^{(K)} \leftarrow \mathbf{0}, \overline{\mathbf{d}}^{(K)} \leftarrow \mathbf{0}$  ( $\mathbf{0} \in \mathbb{R}^m$  is the all-zero vector)
- 3  $\underline{\mathbf{z}}^{(0)} \leftarrow \underline{\mathbf{x}}, \overline{\mathbf{z}}^{(0)} \leftarrow \overline{\mathbf{x}}$
- 4 **for**  $k \in \{K, \dots, 1\}$  **do**
- 5      $(\underline{\mathbf{A}}^{(k-1)}, \overline{\mathbf{A}}^{(k-1)}, \underline{\Delta}, \overline{\Delta}) \leftarrow \text{CROWNRule}(f^{(k)}, \underline{\mathbf{A}}^{(k)}, \overline{\mathbf{A}}^{(k)}, \underline{\mathbf{z}}^{(k-1)}, \overline{\mathbf{z}}^{(k-1)})$
- 6      $\underline{\mathbf{d}}^{(k-1)} \leftarrow \underline{\mathbf{d}}^{(k)} + \underline{\Delta}$
- 7      $\overline{\mathbf{d}}^{(k-1)} \leftarrow \overline{\mathbf{d}}^{(k)} + \overline{\Delta}$
- 8 **return**  $\underline{\mathbf{A}}^{(0)}, \overline{\mathbf{A}}^{(0)}, \underline{\mathbf{d}}^{(0)}, \overline{\mathbf{d}}^{(0)}$

---



---

**Algorithm 5:** CROWN with CROWN Layer Bounds

---

**Input:** Neural Network  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$ , Input Bounds  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$

- 1 **for**  $k \in \{1, \dots, K\}$  **do** (Below, CROWN refers to Algorithm 4)
- 2      $[\underline{\mathbf{z}}^{(k)}, \overline{\mathbf{z}}^{(k)}] \leftarrow \text{Concretise}(\text{CROWN}(f^{(k)}, [\underline{\mathbf{x}}, \overline{\mathbf{x}}], [\underline{\mathbf{z}}^{(1)}, \overline{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(k-1)}, \overline{\mathbf{z}}^{(k-1)}]))$
- 3 **return**  $\text{CROWN}(\text{net}, [\underline{\mathbf{x}}, \overline{\mathbf{x}}], [\underline{\mathbf{z}}^{(1)}, \overline{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(K-1)}, \overline{\mathbf{z}}^{(K-1)}])$

---

use  $f^{(:,k)} = f^{(k)} \circ \dots \circ f^{(1)}$  and  $f^{(k,:)} = f^{(K)} \circ \dots \circ f^{(k)}$  to denote partial evaluation of net. Further, let  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathcal{X}$ . CROWN computes a linear lower bound and a linear upper bound on net

$$\underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{d}} \leq \text{net}(\mathbf{x}) \leq \overline{\mathbf{A}}\mathbf{x} + \overline{\mathbf{d}}, \quad \forall \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}].$$

These linear bounds can be *concretised* into constant bounds as

$$\underline{\mathbf{y}}_{\text{CROWN}} = [\underline{\mathbf{A}}]^+ \underline{\mathbf{x}} + [\underline{\mathbf{A}}]^- \overline{\mathbf{x}} + \underline{\mathbf{d}} \leq \text{net}(\mathbf{x}) \leq [\overline{\mathbf{A}}]^+ \overline{\mathbf{x}} + [\overline{\mathbf{A}}]^- \underline{\mathbf{x}} + \overline{\mathbf{d}} = \overline{\mathbf{y}}_{\text{CROWN}},$$

where  $\underline{\mathbf{x}}, \overline{\mathbf{x}}$  are bounds on the input,  $\mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ ,  $[\mathbf{A}]_{i,j}^+ = \max(0, \mathbf{A}_{i,j})$ , and  $[\mathbf{A}]_{i,j}^- = \min(0, \mathbf{A}_{i,j})$ .

CROWN computes the linear bounds by a backwards walk over the network net, starting from  $f^{(K)}$  and propagating linear bounds backwards until reaching  $f^{(1)}$ . Algorithm 4 defines CROWN for feed-forward neural networks, such as net. The algorithm iteratively computes linear bounds

$$\underline{\mathbf{A}}^{(k)} \mathbf{z}^{(k)} + \underline{\mathbf{d}}^{(k)} \leq f^{(k+1,:)}(\mathbf{z}^{(k)}) \leq \overline{\mathbf{A}}^{(k)} \mathbf{z}^{(k)} + \overline{\mathbf{d}}^{(k)}, \quad \forall \mathbf{z}^{(k)} \in [\underline{\mathbf{z}}^{(k)}, \overline{\mathbf{z}}^{(k)}],$$

where  $[\underline{\mathbf{z}}^{(k)}, \overline{\mathbf{z}}^{(k)}]$  are bounds on  $f^{(:,k)}(\mathbf{x})$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$  that are computed before invoking CROWN, for example, using IntervalArithmetic [76]. The bounds  $\underline{\mathbf{z}}^{(k)}, \overline{\mathbf{z}}^{(k)}$  are sometimes known as pre-activation bounds. Alternatively to using IntervalArithmetic, we can compute  $\underline{\mathbf{z}}^{(k)}, \overline{\mathbf{z}}^{(k)}$  by invoking Algorithm 4 iteratively for  $f^{(:,1)}, \dots, f^{(:,K)}$  and use the concretised bounds on  $f^{(:,1)}, \dots, f^{(:,k-1)}$  as the layer bounds for the  $k$ -th invocation of Algorithm 4 [78]. Algorithm 5 describes this approach in more detail. Applying Algorithm 4 with the layer bounds computed by IntervalArithmetic is known as CROWN-IBP [76], while using Algorithm 5 is referred to as just CROWN. To avoid confusion, in this section, we use CROWN to only refer to Algorithm 4.

Similarly to IntervalArithmetic, Algorithm 4 uses CROWNRules to propagate linear bounds through more fundamental functions, such as affine layers or ReLU layers. Specifically, the CROWNRule for an affine function  $f^{(k)}(\mathbf{z}) = \mathbf{W}\mathbf{z} + \mathbf{b}$  computes

$$\underline{\mathbf{A}}^{(k-1)} = \underline{\mathbf{A}}^{(k)} \mathbf{W}, \quad \overline{\mathbf{A}}^{(k-1)} = \overline{\mathbf{A}}^{(k)} \mathbf{W}, \quad \underline{\Delta}^{(k-1)} = \underline{\mathbf{A}}^{(k)} \mathbf{b}, \quad \overline{\Delta}^{(k-1)} = \overline{\mathbf{A}}^{(k)} \mathbf{b}. \quad (10)$$

The CROWNRule for a ReLU layer  $f^{(k)}(\mathbf{z}) = [\mathbf{z}]^+$  is

$$\begin{aligned}\underline{\mathbf{A}}^{(k-1)} &= [\underline{\mathbf{A}}^{(k)}]^+ \text{diag}(\underline{\boldsymbol{\alpha}}) + [\underline{\mathbf{A}}^{(k)}]^- \text{diag}(\overline{\boldsymbol{\alpha}}), & \underline{\Delta}^{(k-1)} &= [\underline{\mathbf{A}}^{(k)}]^- \overline{\boldsymbol{\beta}}, \\ \overline{\mathbf{A}}^{(k-1)} &= [\overline{\mathbf{A}}^{(k)}]^+ \text{diag}(\overline{\boldsymbol{\alpha}}) + [\overline{\mathbf{A}}^{(k)}]^- \text{diag}(\underline{\boldsymbol{\alpha}}), & \overline{\Delta}^{(k-1)} &= [\overline{\mathbf{A}}^{(k)}]^+ \overline{\boldsymbol{\beta}},\end{aligned}\tag{11}$$

where  $\text{diag}(\boldsymbol{\alpha})$  is a diagonal matrix with the vector  $\boldsymbol{\alpha}$  on its diagonal and  $\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\beta}} \in \mathbb{R}^{n_k}$  have

$$\begin{aligned}\overline{\boldsymbol{\alpha}}_i &= \begin{cases} 0 & \overline{\mathbf{z}}_i^{(k-1)} \leq 0 \\ 1 & \underline{\mathbf{z}}_i^{(k-1)} \geq 0 \\ \frac{\overline{\mathbf{z}}_i^{(k-1)}}{\overline{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}} & \text{otherwise,} \end{cases} & \overline{\boldsymbol{\beta}}_i &= \begin{cases} 0 & 0 \notin (\underline{\mathbf{z}}_i^{(k-1)}, \overline{\mathbf{z}}_i^{(k-1)}) \\ -\underline{\mathbf{z}}_i^{(k-1)} \overline{\boldsymbol{\alpha}}_i & \text{otherwise,} \end{cases} \\ \underline{\boldsymbol{\alpha}}_i &= \begin{cases} 0 & \overline{\mathbf{z}}_i^{(k-1)} \leq 0 \\ 1 & \underline{\mathbf{z}}_i^{(k-1)} \geq 0 \\ 0 & \underline{\mathbf{z}}_i^{(k-1)} < 0 < \overline{\mathbf{z}}_i^{(k-1)} \text{ and } |\underline{\mathbf{z}}_i^{(k-1)}| \geq |\overline{\mathbf{z}}_i^{(k-1)}| \\ 1 & \text{otherwise,} \end{cases}\end{aligned}$$

for every  $i \in \{1, \dots, n_k\}$ . Instead of the last two cases for  $\underline{\boldsymbol{\alpha}}_i$ , we can also optimise each  $\underline{\boldsymbol{\alpha}}_i$  using gradient descent to improve the linear bounds on the network [75], as long as we maintain  $\underline{\boldsymbol{\alpha}}_i \in [0, 1]$ .

### CROWN is not Inclusion-Isotonic.

We demonstrate that CROWN is not inclusion-isotonic according to Definition 6 using an example. Consider a single ReLU neuron  $[x]^+$  with input bounds  $[\underline{x}, \overline{x}] = [-3, 2]$ . The CROWN bounds are  $0 \leq [x]^+ \leq \frac{2}{5}x + \frac{6}{5}$  so that the concretised lower bound  $\underline{y}_{\text{CROWN}} = 0$ . However, the CROWN bounds for  $[\underline{x}', \overline{x}'] = [-1, 2] \subset [\underline{x}, \overline{x}]$  are  $x \leq [x]^+ \leq \frac{2}{3}x + \frac{2}{3}$  which yields the concretised lower bound  $\underline{y}'_{\text{CROWN}} = -1$ . Since  $\underline{y}'_{\text{CROWN}} < \underline{y}_{\text{CROWN}}$  although  $[\underline{x}', \overline{x}'] \subset [\underline{x}, \overline{x}]$ , CROWN is not inclusion-isotonic.

## C.2 CROWN Interval Extension

In this section, we introduce the CROWN Interval Extension (CIE). CIE is a theoretical device that we use to prove that the width of the concretised CROWN bounds linearly converges to zero as the width of the input bounds converges to zero, analogously to Theorem 4. Concretely, CIE is an inclusion-isotonic Lipschitz interval extension according to Definition 6 and Definition 7 that is guaranteed to contain the concretised CROWN bounds. Since Theorem 4 applies to CIE and the concretised CROWN bounds are always contained in the CIE bounds, the convergence properties of Theorem 4 also apply to CROWN. In the following, we first define CIE, show that it is an inclusion-isotonic Lipschitz interval extension, show that the bounds computed by IntervalArithmetic are contained in the bounds computed by CIE, and finally prove that the concretised CROWN bounds are contained in the bounds computed by CIE.

**CIE Algorithm.** To define CIE, we define a bounding rule for ReLU. We use the same bounding rule for affine functions as IntervalArithmetic, as introduced in Appendix B.1. Similarly to IntervalArithmetic, CIE then computes bounds on a ReLU-activated feed-forward neural network  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$  by computing  $(F_{\text{CIE}}^{(K)} \circ \dots \circ F_{\text{CIE}}^{(1)})(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ , where  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathcal{X}$  and  $F_{\text{CIE}}^{(k)} : \mathbb{R}^{n_k} \times \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_k} \times \mathbb{R}^{n_k}$  is the CIE bounding rule for  $f^{(k)}$ . The CIE bounding rule for ReLU layers  $f^{(k)}(\mathbf{v}) = [\mathbf{v}]^+$  is  $F^{(k)}([\underline{\mathbf{v}}, \overline{\mathbf{v}}]) = [\underline{\mathbf{w}}, \overline{\mathbf{w}}]$ , where

$$\underline{\mathbf{w}}_i = \begin{cases} 0 & \overline{\mathbf{v}}_i \leq 0 \\ \underline{\mathbf{v}}_i & \text{otherwise,} \end{cases} \quad \overline{\mathbf{w}}_i = \begin{cases} 0 & \overline{\mathbf{v}}_i \leq 0 \\ \overline{\mathbf{v}}_i & \text{otherwise,} \end{cases}\tag{12}$$

where  $i \in \{1, \dots, n_k\}$ . Comparing this bounding rule to the CROWNRule for ReLU, we can see that the bounds computed by CIE always contain the concretised CROWN bounds for a ReLU neuron. The remainder of this section aims to show that this is also the case for a complete neural network.

**Proposition 5.** *Let  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$  be a ReLU-activated fully-connected neural network. The CIE interval function  $F_{\text{CIE}}^{(K)} \circ \dots \circ F_{\text{CIE}}^{(1)}$  is a inclusion-isotonic Lipschitz interval extension of  $\text{net}$ .*

*Proof.* We first show that the CIE bounding rule for ReLU is an inclusion-isotonic Lipschitz interval extension of ReLU. Let  $F^{(k)}$  be the CIE bounding rule for the ReLU layer  $f^{(k)} : \mathbb{R}^{(n_k)} \rightarrow \mathbb{R}^{(n_k)}$  according to Equation (12).

- *Interval extension.* Let  $\mathbf{v} \in \mathbb{R}^{n_k}$  and  $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] = F^{(k)}([\mathbf{v}, \mathbf{v}])$ . Let  $i \in \{1, \dots, n_k\}$ . If  $\mathbf{v}_i \leq 0$ , we have  $\underline{\mathbf{w}}_i = \overline{\mathbf{w}}_i = 0 = [\mathbf{v}]_i^+$ . Otherwise, if  $\mathbf{v}_i > 0$ , we have  $\underline{\mathbf{w}}_i = \overline{\mathbf{w}}_i = \mathbf{v}_i = [\mathbf{v}]_i^+$ .
- *Inclusion-isotonic.* Let  $[\mathbf{v}', \overline{\mathbf{v}}'] \subseteq [\underline{\mathbf{v}}, \overline{\mathbf{v}}] \subseteq \mathbb{R}^{n_k}$ ,  $[\underline{\mathbf{w}}', \overline{\mathbf{w}}'] = F^{(k)}([\mathbf{v}', \overline{\mathbf{v}}'])$  and  $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] = F^{(k)}([\underline{\mathbf{v}}, \overline{\mathbf{v}}])$ . Let  $i \in \{1, \dots, n_k\}$ . If  $\overline{\mathbf{v}}_i \leq 0$ , we have  $[\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i] = [0, 0] = [\underline{\mathbf{w}}'_i, \overline{\mathbf{w}}'_i]$  since  $\overline{\mathbf{v}}'_i \leq \overline{\mathbf{v}}_i$ . On the other hand, if  $\overline{\mathbf{v}}_i > 0$ , we have  $[\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i] = [\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i]$ . If  $\overline{\mathbf{v}}'_i \leq 0$ ,  $[\underline{\mathbf{w}}'_i, \overline{\mathbf{w}}'_i] = [0, 0] \subseteq [\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i]$  since  $\underline{\mathbf{v}}_i \leq \overline{\mathbf{v}}'_i < \overline{\mathbf{v}}_i$ . Otherwise,  $[\underline{\mathbf{w}}'_i, \overline{\mathbf{w}}'_i] = [\underline{\mathbf{v}}'_i, \overline{\mathbf{v}}'_i] \subseteq [\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i]$ . Therefore,  $F^{(k)}$  is inclusion-isotonic.
- *Lipschitz.* Let  $[\underline{\mathbf{v}}, \overline{\mathbf{v}}] \subseteq \mathbb{R}^{n_k}$  and  $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] = F^{(k)}([\underline{\mathbf{v}}, \overline{\mathbf{v}}])$ . For all  $i \in \{1, \dots, n_k\}$  with  $\overline{\mathbf{w}}_i \leq 0$  we have  $w([\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i]) = 0$  and, therefore, trivially  $w([\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i]) \leq w([\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i])$ . For  $i \in \{1, \dots, n_k\}$  with  $\overline{\mathbf{w}}_i > 0$ , we have  $[\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i] = [\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i]$ . Overall,  $w([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \max_{i \in \{1, \dots, n_k\}} \overline{\mathbf{w}}_i - \underline{\mathbf{w}}_i \leq \max_{i \in \{1, \dots, n_k\}} \overline{\mathbf{v}}_i - \underline{\mathbf{v}}_i = w([\underline{\mathbf{v}}, \overline{\mathbf{v}}])$ . Therefore,  $F^{(k)}$  is Lipschitz with Lipschitz constant 1.

For affine functions, CIE uses the same bounding rule as IntervalArithmetic that is an inclusion-isotonic Lipschitz interval extension [49]. Lemma 6.3 of Moore et al. [49] now gives us that the composition  $F_{\text{CIE}}^{(K)} \circ \dots \circ F_{\text{CIE}}^{(1)}$  of inclusion-isotonic Lipschitz interval extensions is also inclusion isotonic and Lipschitz. Clearly, a composition of interval extensions is also an interval extension.  $\square$

**Proposition 6.** *Let  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$  be a ReLU-activated fully-connected neural network. Let  $[\underline{\mathbf{y}}, \overline{\mathbf{y}}]$  and  $[\underline{\mathbf{w}}, \overline{\mathbf{w}}]$  be the bounds on  $\text{net}(\mathbf{x})$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$  computed by IntervalArithmetic and CIE, respectively. It holds that  $[\underline{\mathbf{y}}, \overline{\mathbf{y}}] \subseteq [\underline{\mathbf{w}}, \overline{\mathbf{w}}]$ .*

*Proof.* Let  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$  and  $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$  be as in Proposition 6. We prove Proposition 6 by finite induction over  $k \in \{0, \dots, K\}$ . Let  $[\underline{\mathbf{y}}^{(k)}, \overline{\mathbf{y}}^{(k)}]$  and  $[\underline{\mathbf{v}}^{(k)}, \overline{\mathbf{v}}^{(k)}]$  be the bounds that IntervalArithmetic and CIE respectively compute on  $f^{(k)}(\mathbf{x})$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ .

*Induction start.* For  $k = 0$ , we consider the identity function  $\text{net}(\mathbf{x}) = \mathbf{x}$ , for which both CIE and IntervalArithmetic compute  $[\underline{\mathbf{y}}^{(0)}, \overline{\mathbf{y}}^{(0)}] = [\underline{\mathbf{v}}^{(0)}, \overline{\mathbf{v}}^{(0)}] = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ .

*Induction step.* Now, let  $k \in [K]$  and assume  $[\underline{\mathbf{y}}^{(k-1)}, \overline{\mathbf{y}}^{(k-1)}] \subseteq [\underline{\mathbf{v}}^{(k-1)}, \overline{\mathbf{v}}^{(k-1)}]$ . We show  $[\underline{\mathbf{y}}^{(k)}, \overline{\mathbf{y}}^{(k)}] \subseteq [\underline{\mathbf{v}}^{(k)}, \overline{\mathbf{v}}^{(k)}]$ . We differentiate two cases:

- If  $f^{(k)}$  is an affine function, both IntervalArithmetic and CIE use the IntervalArithmetic bounding rule for affine functions from Appendix B.1 to compute  $[\underline{\mathbf{y}}^{(k)}, \overline{\mathbf{y}}^{(k)}]$  and  $[\underline{\mathbf{v}}^{(k)}, \overline{\mathbf{v}}^{(k)}]$ . Since this bounding rule is inclusion-isotonic [49], we have  $[\underline{\mathbf{y}}^{(k)}, \overline{\mathbf{y}}^{(k)}] \subseteq [\underline{\mathbf{v}}^{(k)}, \overline{\mathbf{v}}^{(k)}]$ .
- Otherwise,  $f^{(k)}$  is ReLU. The IntervalArithmetic bounding rule for ReLU computes  $[\underline{\mathbf{y}}^{(k)}, \overline{\mathbf{y}}^{(k)}] = [\underline{\mathbf{y}}^{(k-1)}]^+, [\overline{\mathbf{y}}^{(k-1)}]^+$ . Since the CIE rule for ReLU is an inclusion-isotonic interval extension (Proposition 5), it follows that  $[\underline{\mathbf{y}}^{(k-1)}]^+, [\overline{\mathbf{y}}^{(k-1)}]^+ \subseteq [\underline{\mathbf{v}}^{(k)}, \overline{\mathbf{v}}^{(k)}]$ .



Overall, this shows that  $[\underline{\mathbf{y}}, \bar{\mathbf{y}}] = [\underline{\mathbf{y}}^{(K)}, \bar{\mathbf{y}}^{(K)}] \subseteq [\underline{\mathbf{v}}^{(K)}, \bar{\mathbf{v}}^{(K)}] = [\underline{\mathbf{v}}, \bar{\mathbf{v}}]$ .  $\square$

**Proposition 7.** Let  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$  be a ReLU-activated fully-connected neural network, let  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}] \subseteq \mathbb{R}^n$  and let  $[\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$  be the CIE bounds on  $f^{(k)}(\mathbf{x})$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ ,  $\forall k \in \{1, \dots, K\}$ . Further, let  $[\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}]$  also be bounds on  $f^{(k)}(\mathbf{x})$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  with  $[\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}] \subseteq [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$ ,  $\forall k \in \{1, \dots, K\}$ . It holds that

$$\underline{\mathbf{v}}^{(K)} \leq \underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{d}} \quad \text{and} \quad \bar{\mathbf{A}}\mathbf{x} + \bar{\mathbf{d}} \leq \bar{\mathbf{v}}^{(K)}, \quad \forall \mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}],$$

where  $\underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{d}}$  and  $\bar{\mathbf{A}}\mathbf{x} + \bar{\mathbf{d}}$  are the bounds on  $\text{net}(\mathbf{x})$  for  $\mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  computed by CROWN (Algorithm 4) using the layer bounds  $[\underline{\mathbf{z}}^{(1)}, \bar{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(K-1)}, \bar{\mathbf{z}}^{(K-1)}]$ .

*Proof.* Let  $\text{net} = f^{(K)} \circ \dots \circ f^{(1)}$ ,  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ ,  $[\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}]$ , and  $[\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$  for all  $k \in \{1, \dots, K\}$  be as in Proposition 7. Let  $\underline{\mathbf{A}}^{(k)}, \bar{\mathbf{A}}^{(k)}, \underline{\mathbf{d}}^{(k)}, \bar{\mathbf{d}}^{(k)}$  for all  $k \in \{0, \dots, K\}$  be as in Algorithm 4 for the inputs  $\text{net}$ ,  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  and  $[\underline{\mathbf{z}}^{(1)}, \bar{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(K)}, \bar{\mathbf{z}}^{(K)}]$ . Further, let  $[\underline{\mathbf{z}}^{(0)}, \bar{\mathbf{z}}^{(0)}] = [\underline{\mathbf{v}}^{(0)}, \bar{\mathbf{v}}^{(0)}] = [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ . In the following, we prove

$$\underline{\mathbf{A}}^{(k)}\mathbf{z}^{(k)} + \underline{\mathbf{d}}^{(k)} \geq \underline{\mathbf{v}}^{(K)} \quad \text{and} \quad \bar{\mathbf{A}}^{(k)}\mathbf{z}^{(k)} + \bar{\mathbf{d}}^{(k)} \leq \bar{\mathbf{v}}^{(K)}, \quad \forall \mathbf{z}^{(k)} \in [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}], \quad (13)$$

for every  $k \in \{0, \dots, K\}$ . Note that  $[\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}] \subseteq [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$  by assumption in Proposition 7. Proving Equation (13) also proves Proposition 7 when  $k = 0$ . We proceed by finite induction over  $k$  from  $K$  to 0, following the backwards walk performed by Algorithm 4.

*Induction start.* We consider  $k = K$ . In this case,  $\underline{\mathbf{A}}^{(K)} = \bar{\mathbf{A}}^{(K)} = \mathbf{I}$  and  $\underline{\mathbf{d}}^{(K)} = \bar{\mathbf{d}}^{(K)} = \mathbf{0}$ , where  $\mathbf{I} \in \mathbb{R}^{n_K \times n_K}$  is the identity matrix and  $\mathbf{0} \in \mathbb{R}^{n_K}$  is the all-zero vector. Therefore, we have

$$\begin{aligned} \underline{\mathbf{A}}^{(K)}\mathbf{z}^{(K)} + \underline{\mathbf{d}}^{(K)} &= \mathbf{I}\mathbf{z}^{(K)} + \mathbf{0} \geq \underline{\mathbf{v}}^{(K)} \\ \bar{\mathbf{A}}^{(K)}\mathbf{z}^{(K)} + \bar{\mathbf{d}}^{(K)} &= \mathbf{I}\mathbf{z}^{(K)} + \mathbf{0} \leq \bar{\mathbf{v}}^{(K)}, \end{aligned}$$

for  $\mathbf{z}^{(K)} \in [\underline{\mathbf{v}}^{(K)}, \bar{\mathbf{v}}^{(K)}]$ . This proves Equation (13) for  $k = K$ .

*Induction step.* Let  $k \in \{1, \dots, K\}$  and assume Equation (13) holds for  $k$ . We show that it also holds for  $k - 1$ . We differentiate two cases, based on whether  $f^{(k)}$  is a affine function or ReLU.

- *Affine function.* If  $f^{(k)}(\mathbf{z}) = \mathbf{W}\mathbf{z} + \mathbf{b}$ , CROWNRULE computes the linear bounds on  $f^{(k)}$  as in Equation (10), so that we obtain

$$\underline{\mathbf{A}}^{(k-1)}\mathbf{z} + \underline{\mathbf{d}}^{(k-1)} = \underline{\mathbf{A}}^{(k)}\mathbf{W}\mathbf{z} + \underline{\mathbf{A}}^{(k)}\mathbf{b} + \underline{\mathbf{d}}^{(k)} = \underline{\mathbf{A}}^{(k)}(\mathbf{W}\mathbf{z} + \mathbf{b}) + \underline{\mathbf{d}}^{(k)} \geq \underline{\mathbf{v}}^{(K)},$$

where  $\mathbf{z} \in [\underline{\mathbf{v}}^{(k-1)}, \bar{\mathbf{v}}^{(k-1)}]$ . The final inequality is due to the induction assumption that Equation (13) holds for  $k$ , which we can apply since  $\mathbf{W}\mathbf{z} + \mathbf{b} \in [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$ , since CIE computes  $[\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$  from  $[\underline{\mathbf{v}}^{(k-1)}, \bar{\mathbf{v}}^{(k-1)}]$  using Equation (9). Similarly, we obtain

$$\bar{\mathbf{A}}^{(k-1)}\mathbf{z} + \bar{\mathbf{d}}^{(k-1)} = \bar{\mathbf{A}}^{(k)}\mathbf{W}\mathbf{z} + \bar{\mathbf{A}}^{(k)}\mathbf{b} + \bar{\mathbf{d}}^{(k)} = \bar{\mathbf{A}}^{(k)}(\mathbf{W}\mathbf{z} + \mathbf{b}) + \bar{\mathbf{d}}^{(k)} \leq \bar{\mathbf{v}}^{(K)},$$

again by using the induction assumption.

- *ReLU.* Let  $\mathbf{z} \in [\underline{\mathbf{v}}^{(k-1)}, \bar{\mathbf{v}}^{(k-1)}]$  and  $i \in \{1, \dots, n_k\}$ . CROWN computes the linear bounds on  $f^{(k)}$  according to Equation (11), which yields

$$\begin{aligned} \underline{\mathbf{A}}_{:,i}^{(k-1)}\mathbf{z}_i + \underline{\mathbf{d}}_i^{(k-1)} &= \left( \left[ \underline{\mathbf{A}}_{:,i}^{(k)} \right]^+ \underline{\alpha}_i + \left[ \underline{\mathbf{A}}_{:,i}^{(k)} \right]^- \bar{\alpha}_i \right) \mathbf{z}_i + \left[ \underline{\mathbf{A}}_{:,i}^{(k)} \right]^- \bar{\beta}_i + \underline{\mathbf{d}}_i^{(k)} \\ \bar{\mathbf{A}}_{:,i}^{(k-1)}\mathbf{z}_i + \bar{\mathbf{d}}_i^{(k-1)} &= \left( \left[ \bar{\mathbf{A}}_{:,i}^{(k)} \right]^+ \bar{\alpha}_i + \left[ \bar{\mathbf{A}}_{:,i}^{(k)} \right]^- \underline{\alpha}_i \right) \mathbf{z}_i + \left[ \bar{\mathbf{A}}_{:,i}^{(k)} \right]^+ \bar{\beta}_i + \bar{\mathbf{d}}_i^{(k)}, \end{aligned}$$

where  $\mathbf{A}_{:,i}$  denotes the  $i$ -th column of matrix  $\mathbf{A}$ . We now differentiate three cases based on the signs of  $\underline{\mathbf{z}}_i^{(k-1)}$  and  $\bar{\mathbf{z}}_i^{(k-1)}$ .

(i) Consider  $\bar{\mathbf{z}}_i^{(k-1)} \leq 0$ . We have  $\underline{\alpha}_i = \bar{\alpha}_i = \bar{\beta}_i = 0$ . Therefore,

$$\begin{aligned}\underline{\mathbf{A}}_{:,i}^{(k-1)} \mathbf{z}_i + \underline{\mathbf{d}}_i^{(k-1)} &= \underline{\mathbf{d}}_i^{(k)} \geq \underline{\mathbf{v}}^{(K)} \\ \overline{\mathbf{A}}_{:,i}^{(k-1)} \mathbf{z}_i + \overline{\mathbf{d}}_i^{(k-1)} &= \overline{\mathbf{d}}_i^{(k)} \leq \overline{\mathbf{v}}^{(K)},\end{aligned}$$

where both inequalities are due to the induction assumption with  $\mathbf{z}_i^{(k)} = 0$  in Equation (13). We can apply the induction assumption since  $0 = [\bar{\mathbf{z}}_i^{(k-1)}]^+ \in [\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}]$  and, therefore,  $0 \in [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$  since  $[\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}] \subseteq [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$ .

(ii) Consider  $\bar{\mathbf{z}}_i^{(k-1)} > 0$  and  $\underline{\mathbf{z}}_i^{(k-1)} \geq 0$ . First, we find that  $[\underline{\mathbf{v}}_i^{(k)}, \bar{\mathbf{v}}_i^{(k)}] = [\underline{\mathbf{v}}_i^{(k-1)}, \bar{\mathbf{v}}_i^{(k-1)}]$  according to Equation (12) since  $\bar{\mathbf{v}}_i^{(k-1)} \geq \bar{\mathbf{z}}_i^{(k-1)} > 0$ . Regarding CROWN, we have  $\underline{\alpha}_i = \bar{\alpha}_i = 1$  and  $\bar{\beta}_i = 0$ . Therefore,

$$\begin{aligned}\underline{\mathbf{A}}_{:,i}^{(k-1)} \mathbf{z}_i + \underline{\mathbf{d}}_i^{(k-1)} &= \left( [\underline{\mathbf{A}}_{:,i}^{(k)}]^+ + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \right) \mathbf{z}_i + \underline{\mathbf{d}}_i^{(k)} = \underline{\mathbf{A}}_{:,i}^{(k)} \mathbf{z}_i + \underline{\mathbf{d}}_i^{(k)} \geq \underline{\mathbf{v}}^{(K)} \\ \overline{\mathbf{A}}_{:,i}^{(k-1)} \mathbf{z}_i + \overline{\mathbf{d}}_i^{(k-1)} &= \left( [\overline{\mathbf{A}}_{:,i}^{(k)}]^+ + [\overline{\mathbf{A}}_{:,i}^{(k)}]^- \right) \mathbf{z}_i + \overline{\mathbf{d}}_i^{(k)} = \overline{\mathbf{A}}_{:,i}^{(k)} \mathbf{z}_i + \overline{\mathbf{d}}_i^{(k)} \leq \overline{\mathbf{v}}^{(K)},\end{aligned}$$

where the inequalities are due to the induction assumption that we can apply since  $\mathbf{z}_i \in [\underline{\mathbf{v}}_i^{(k-1)}, \bar{\mathbf{v}}_i^{(k-1)}] = [\underline{\mathbf{v}}_i^{(k)}, \bar{\mathbf{v}}_i^{(k)}]$ .

(iii) Finally, consider  $\underline{\mathbf{z}}_i^{(k-1)} < 0 < \bar{\mathbf{z}}_i^{(k-1)}$ . With the same argument as in the previous case, we have  $[\underline{\mathbf{v}}_i^{(k)}, \bar{\mathbf{v}}_i^{(k)}] = [\underline{\mathbf{v}}_i^{(k-1)}, \bar{\mathbf{v}}_i^{(k-1)}]$ . We have

$$\bar{\alpha}_i = \frac{\bar{\mathbf{z}}_i^{(k-1)}}{\bar{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}}, \quad \bar{\beta}_i = -\underline{\mathbf{z}}_i^{(k-1)} \bar{\alpha}_i = \frac{-\underline{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{z}}_i^{(k-1)}}{\bar{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}}$$

and  $\underline{\alpha}_i \in [0, 1]$ . Now,

$$\begin{aligned}\underline{\mathbf{A}}_{:,i}^{(k-1)} \mathbf{z}_i + \underline{\mathbf{d}}_i^{(k-1)} &= \left( [\underline{\mathbf{A}}_{:,i}^{(k)}]^+ \underline{\alpha}_i + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \bar{\alpha}_i \right) \mathbf{z}_i + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \bar{\beta}_i + \underline{\mathbf{d}}_i^{(k)}\end{aligned}\tag{14a}$$

$$\geq [\underline{\mathbf{A}}_{:,i}^{(k)}]^+ \underline{\alpha}_i \underline{\mathbf{v}}_i^{(k-1)} + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \bar{\alpha}_i \bar{\mathbf{v}}_i^{(k-1)} + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \bar{\beta}_i + \underline{\mathbf{d}}_i^{(k)}\tag{14b}$$

$$\geq [\underline{\mathbf{A}}_{:,i}^{(k)}]^+ \underline{\mathbf{v}}_i^{(k-1)} + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \frac{\bar{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{v}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{z}}_i^{(k-1)}}{\bar{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}} + \underline{\mathbf{d}}_i^{(k)}\tag{14c}$$

$$\geq [\underline{\mathbf{A}}_{:,i}^{(k)}]^+ \underline{\mathbf{v}}_i^{(k-1)} + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \frac{\bar{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{v}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{v}}_i^{(k-1)}}{\bar{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}} + \underline{\mathbf{d}}_i^{(k)}\tag{14d}$$

$$\begin{aligned}&\geq [\underline{\mathbf{A}}_{:,i}^{(k)}]^+ \underline{\mathbf{v}}_i^{(k-1)} + [\underline{\mathbf{A}}_{:,i}^{(k)}]^- \bar{\mathbf{v}}_i^{(k-1)} + \underline{\mathbf{d}}_i^{(k)} \\ &\geq \underline{\mathbf{A}}_{:,i}^{(k)} \left( \mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} \geq 0} \underline{\mathbf{v}}_i^{(k-1)} + \mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} < 0} \bar{\mathbf{v}}_i^{(k-1)} \right) + \underline{\mathbf{d}}_i^{(k)} \\ &\geq \underline{\mathbf{v}}^{(K)},\end{aligned}\tag{14e}$$

where  $\mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} \geq 0} \in \{0, 1\}^{n_k}$  with

$$\left( \mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} \geq 0} \right)_j = \begin{cases} 1 & \underline{\mathbf{A}}_{j,i}^{(k)} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

for  $j \in \{1, \dots, n_k\}$  and  $\mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} < 0}$  is defined equivalently. Above, the inequality in Equation (14b) is by choosing  $\mathbf{z}_i \in [\underline{\mathbf{v}}_i^{(k-1)}, \bar{\mathbf{v}}_i^{(k-1)}]$  as the minimiser of Equation (14a). The

inequality in Equation (14c) is by choosing  $\underline{\alpha}_i = 1$  which minimises Equation (14b) since  $\underline{\mathbf{v}}_i^{(k-1)} \leq \underline{\mathbf{z}}_i^{(k-1)} < 0$ . To justify the inequality in Equation (14d) we note that  $-\underline{\mathbf{z}}_i^{(k-1)}[\underline{\mathbf{A}}_{:,i}^{(k)}]^- \leq 0$  since  $\underline{\mathbf{z}}_i^{(k-1)} < 0$  and, further  $\bar{\mathbf{v}}_i^{(k-1)} \geq \bar{\mathbf{z}}_i^{(k-1)}$ . Finally, Equation (14e) follows from the induction assumption that we can apply since

$$\left( \mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} \geq 0} \underline{\mathbf{v}}_i^{(k-1)} + \mathbf{1}_{\underline{\mathbf{A}}_{:,i}^{(k)} < 0} \bar{\mathbf{v}}_i^{(k-1)} \right) \in [\underline{\mathbf{v}}^{(k-1)}, \bar{\mathbf{v}}^{(k-1)}].$$

For the upper bound, we apply similar steps to obtain

$$\begin{aligned} \bar{\mathbf{A}}_{:,i}^{(k-1)} \mathbf{z}_i + \bar{\mathbf{d}}_i^{(k-1)} &= \left( [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \bar{\alpha}_i + [\bar{\mathbf{A}}_{:,i}^{(k)}]^- \underline{\alpha}_i \right) \mathbf{z}_i + [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \bar{\beta}_i + \bar{\mathbf{d}}_i^{(k)} \\ &\leq [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \bar{\alpha}_i \bar{\mathbf{v}}_i^{(k-1)} + [\bar{\mathbf{A}}_{:,i}^{(k)}]^- \underline{\alpha}_i \underline{\mathbf{v}}_i^{(k-1)} + [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \bar{\beta}_i + \bar{\mathbf{d}}_i^{(k)} \\ &\leq [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \frac{\bar{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{v}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{z}}_i^{(k-1)}}{\bar{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}} + [\bar{\mathbf{A}}_{:,i}^{(k)}]^- \underline{\mathbf{v}}_i^{(k-1)} + \bar{\mathbf{d}}_i^{(k)} \\ &\leq [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \frac{\bar{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{v}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)} \bar{\mathbf{v}}_i^{(k-1)}}{\bar{\mathbf{z}}_i^{(k-1)} - \underline{\mathbf{z}}_i^{(k-1)}} + [\bar{\mathbf{A}}_{:,i}^{(k)}]^- \underline{\mathbf{v}}_i^{(k-1)} + \bar{\mathbf{d}}_i^{(k)} \\ &\leq [\bar{\mathbf{A}}_{:,i}^{(k)}]^+ \bar{\mathbf{v}}_i^{(k-1)} + [\bar{\mathbf{A}}_{:,i}^{(k)}]^- \underline{\mathbf{v}}_i^{(k-1)} + \bar{\mathbf{d}}_i^{(k)} \\ &\leq \bar{\mathbf{A}}_{:,i}^{(k)} \left( \mathbf{1}_{\bar{\mathbf{A}}_{:,i}^{(k)} \geq 0} \bar{\mathbf{v}}_i^{(k-1)} + \mathbf{1}_{\bar{\mathbf{A}}_{:,i}^{(k)} < 0} \underline{\mathbf{v}}_i^{(k-1)} \right) + \bar{\mathbf{d}}_i^{(k)} \\ &\leq \bar{\mathbf{v}}^{(K)}. \end{aligned}$$

By this induction, we have shown Equation (13). Proposition 7 now follows for  $k = 0$ , where  $\underline{\mathbf{A}}^{(0)} = \underline{\mathbf{A}}$ ,  $\bar{\mathbf{A}}^{(0)} = \bar{\mathbf{A}}$ ,  $\underline{\mathbf{d}}^{(0)} = \underline{\mathbf{d}}$ ,  $\bar{\mathbf{d}}^{(0)} = \bar{\mathbf{d}}$  and  $[\underline{\mathbf{v}}^{(0)}, \bar{\mathbf{v}}^{(0)}] = [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$ .  $\square$

Proposition 7 requires layer bounds  $[\underline{\mathbf{z}}^{(1)}, \bar{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(K-1)}, \bar{\mathbf{z}}^{(K-1)}]$  which may not be looser than the CIE bounds. As we have shown in Proposition 6, this applies for IntervalArithmetic. A consequence of Proposition 7 is that this also applies to Algorithm 5.

**Corollary 4.** Let net,  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  and  $[\underline{\mathbf{v}}^{(K)}, \bar{\mathbf{v}}^{(K)}]$  be as in Proposition 7. It holds that

$$\underline{\mathbf{v}}^{(K)} \leq \underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{d}} \quad \text{and} \quad \bar{\mathbf{A}}\mathbf{x} + \bar{\mathbf{d}} \leq \bar{\mathbf{v}}^{(K)}, \quad \forall \mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}],$$

where  $\underline{\mathbf{A}}\mathbf{x} + \underline{\mathbf{d}}$  and  $\bar{\mathbf{A}}\mathbf{x} + \bar{\mathbf{d}}$  are the bounds on net( $\mathbf{x}$ ) for  $\mathbf{x} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  computed by Algorithm 5.

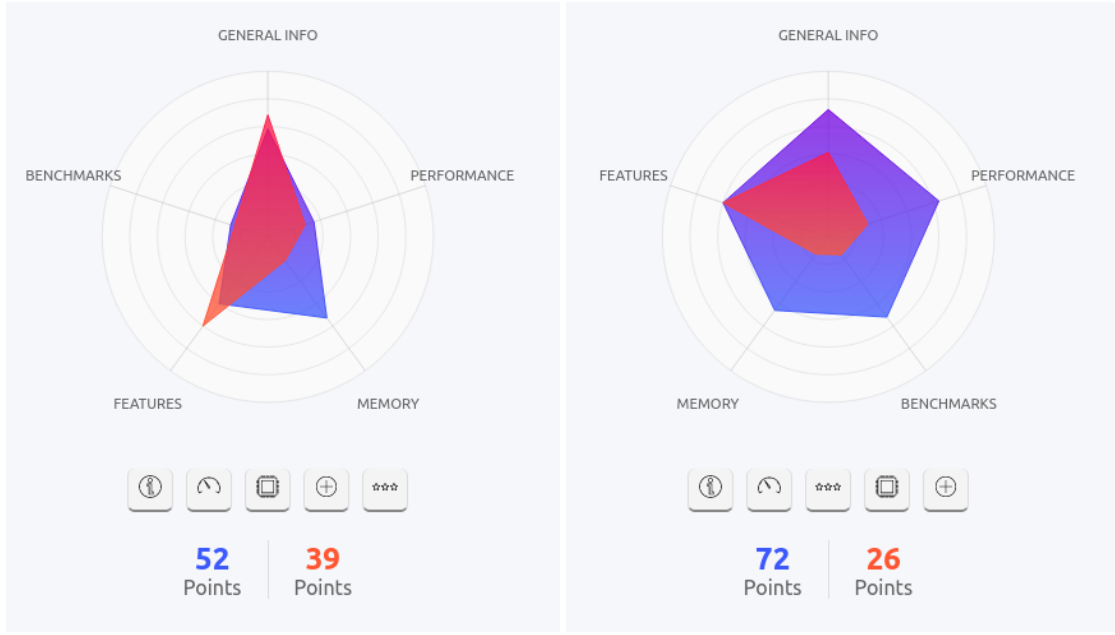
*Proof.* Corollary 4 follows from Proposition 7 by an inductive argument over  $k \in \{1, \dots, K\}$ . Let net =  $f^{(K)} \circ \dots \circ f^{(1)}$ ,  $[\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  and  $[\underline{\mathbf{v}}^{(1)}, \bar{\mathbf{v}}^{(1)}], \dots, [\underline{\mathbf{v}}^{(K)}, \bar{\mathbf{v}}^{(K)}]$  be as in Proposition 7.

*Induction start.* Applying Algorithm 4 to  $f^{(:1)} = f^{(1)}$  does not require layer bounds, so that Proposition 7 holds for the (concretised) CROWN bounds on  $f^{(:1)}$ .

*Induction step.* Let  $k \in \{2, \dots, K\}$ . Assume the concretised CROWN bounds  $[\underline{\mathbf{z}}^{(1)}, \bar{\mathbf{z}}^{(1)}], \dots, [\underline{\mathbf{z}}^{(k-1)}, \bar{\mathbf{z}}^{(k-1)}]$  on  $f^{(:1)}, \dots, f^{(:k-1)}$  satisfy Proposition 7, that is,  $[\underline{\mathbf{z}}^{(k)}, \bar{\mathbf{z}}^{(k)}] \subseteq [\underline{\mathbf{v}}^{(k)}, \bar{\mathbf{v}}^{(k)}]$  for every  $k \in \{1, \dots, k-1\}$ . Since we only require layer bounds on  $f^{(:1)}, \dots, f^{(:k-1)}$  to apply CROWN to  $f^{(:k)}$ , Proposition 7 applies to the (concretised) CROWN bounds we obtain for  $f^{(:k)}$ .  $\square$

We are now able to proof Proposition 1 as a corollary of Proposition 4 and Proposition 7.

*Proof of Proposition 1.* Since CIE is a Lipschitz interval extension according to Proposition 5, Proposition 4 applies. Therefore, CIE satisfies Definition 3. Now, due to Proposition 7, CROWN also satisfies Definition 3.  $\square$



(a) HW1 (red) vs. Converse et al. [26] (blue)

(b) HW1 (red) vs. Marzari et al. [47] (blue)

Figure 4: **Hardware Comparison from [versus.com](https://versus.com).** The figures are taken from <https://versus.com/en/amd-epyc-7401p-vs-intel-core-i7-6700> and <https://versus.com/en/intel-core-i5-13600kf-vs-intel-core-i7-4820k>, respectively. Accessed on the 17th of May 2024.

## D Experiments

This section contains additional details on the experiments from Section 6 and additional experimental results.

### D.1 Hardware

We run all our experiments on a workstation running Ubuntu 22.04 with an Intel i7-4820K CPU, 32 GB of memory and no GPU (HW1). This CPU model is ten years old (introduced in late 2013) and has four cores and eight threads.

In comparison, Converse et al. [26] use an AMD EPYC 7401P CPU for their ACAS Xu robustness experiments, limiting their tool to use 46 threads and at most 4GB of memory. AMD EPYC 7401P was introduced in mid-2017, targeting servers. Marzari et al. [47] use a Ubuntu 22.04 workstation with an Intel i5-13600KF CPU and an Nvidia GeForce RTX 4070 Ti GPU. This CPU was introduced in end-2022 and has 14 cores with 20 threads.

Figure 4 contains the CPU comparison from [versus.com](https://versus.com). As the figure shows, our HW1 is less performant when considering both the theoretical performance according to the hardware specification and the actual performance on a series of computational benchmarks. Therefore, HW1 is inferior in computation power to the hardware used by Converse et al. [26] and Marzari et al. [47].

### D.2 FairSquare Benchmark

We provide additional details on the FairSquare benchmark and present additional results.

#### D.2.1 Extended Description of the Benchmark

The input space in the FairSquare benchmark consists of three unbounded continuous variables for the age, the years of education (‘edu’), and the yearly gain in capital of a person (‘capital gain’),

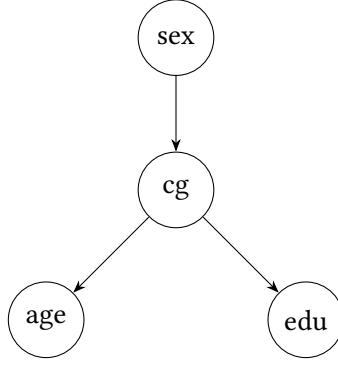


Figure 5: **FairSquare Bayes Net 1.** The network structure of the Bayesian Network from the FairSquare benchmark. In this figure, ‘cg’ denotes the capital gain variable and ‘edu’ denotes the number of years of education.

respectively, as well as an additional discrete protected variable indicating a person’s (assumed binary) sex. The neural networks use ‘age’ and ‘edu’ or ‘age’, ‘edu’ and ‘capital gain’, depending on the network architecture, to predict whether a person has a high salary (higher than \$50 000). The three input distributions used by the FairSquare benchmark are

1. the combination of three independent normal distributions for the continuous variables and a Bernoulli distribution for the person’s sex,
2. a Bayesian Network with the structure in Figure 5 introducing correlations between the variables that are similarly distributed as for the independent input distribution (‘Bayes Net 1’), and
3. the same Bayesian Network augmented with a constraint that the years of education may not exceed a person’s age (‘Bayes Net 2’).

The integrity constraint is implemented by Albarghouthi et al. [4] as a post-processing of the samples from the Bayesian Network. In particular, a person’s years of education are set to the person’s age if the sampled years of education are larger than the sampled age. We introduce a pre-processing layer before the neural network we want to verify to implement this integrity constraint in our setting. This pre-processing layer computes  $\text{edu}' = \min(\text{edu}, \text{age})$  and leaves the remaining inputs unchanged.

Besides experiments on the demographic parity fairness notion, the extended FairSquare benchmarks also include experiments on the parity of qualified persons fairness notion as defined in Appendix A.1. For both fairness notions, the FairSquare benchmark uses a fairness threshold of  $\gamma = 0.85$ .

## D.2.2 Extended Results

Table 3 contains the comparison of PV and FairSquare on the extended set of FairSquare benchmarks from Albarghouthi et al. [4]. The runtime from this table is visualised in Figure 2.

The input distribution of the FairSquare benchmark gives a positive probability to unrealistic values, such as negative values of age. We additionally consider a version of the benchmark where the input distributions are clipped to realistic ranges. Table 4 contains the results of running PV on these *clipped* input distributions. Using realistic input distributions reveals that some FairSquare networks are, in fact, unfair under realistic conditions. Faithfully comparing these results with FairSquare is not feasible since FairSquare does not implement truncated normal distributions, which we use for the clipped input distributions.

Benchmark Instance		Demographic Parity			Parity of Qualified Persons		
		Runtime (s)			Runtime (s)		
		PV	FairSquare	Fair?	PV	FairSquare	Fair?
net	$\mathbb{P}_x$						
NN <sub>2,1</sub>	independent	2	2	✓	2	3	✓
NN <sub>2,1</sub>	Bayes Net 1	4	76	✓	8	38	✓
NN <sub>2,1</sub>	Bayes Net 2	5	51	✓	12	26	✓
NN <sub>2,2</sub>	independent	2	4	✓	2	7	✓
NN <sub>2,2</sub>	Bayes Net 1	4	33	✓	11	60	✓
NN <sub>2,2</sub>	Bayes Net 2	6	59	✓	17	61	✓
NN <sub>3,2</sub>	independent	2	451	✓	1	657	✓
NN <sub>3,2</sub>	Bayes Net 1	4	TO	✓	40	TO	✓
NN <sub>3,2</sub>	Bayes Net 2	5	TO	✓	56	TO	✓

Table 3: **FairSquare Benchmark.** The first two columns indicate the neural network net that is verified and the input probability distribution  $\mathbb{P}_x$ , respectively. We verify two fairness notions: demographic parity and parity of qualified persons. For each fairness notion, the first two columns contain the runtime of PV and FairSquare in seconds. Here, ‘TO’ indices timeout (900s). The last column contains the result of the verification with PV. The results in this table are for the non-clipped input distributions that include unrealistic values, such as negative values of a person’s age.

Benchmark Instance		Demographic Parity		Parity of Qualified Persons	
		PV		PV	
		Runtime	Fair?	Runtime	Fair?
net	$\mathbb{P}_x$				
NN <sub>2,1</sub>	independent clipped	1	✓	1	✓
NN <sub>2,1</sub>	Bayes Net 1 clipped	1	✓	1	✓
NN <sub>2,1</sub>	Bayes Net 2 clipped	1	✓	2	✓
NN <sub>2,2</sub>	independent clipped	1	✓	1	✓
NN <sub>2,2</sub>	Bayes Net 1 clipped	1	✓	2	✓
NN <sub>2,2</sub>	Bayes Net 2 clipped	2	✓	3	✓
NN <sub>3,2</sub>	independent clipped	1	✓	1	✓
NN <sub>3,2</sub>	Bayes Net 1 clipped	2	✗	2	✗
NN <sub>3,2</sub>	Bayes Net 2 clipped	3	✗	TO	?

Table 4: **FairSquare Benchmark for Realistic Input Distributions.** This table contains the results for running PV on the FairSquare benchmarks with clipped input distributions that rule out unrealistic values, such as negative values of a person’s age. The first two columns indicate the neural network net that is verified and the input probability distribution  $\mathbb{P}_x$ , respectively. The following columns contain the runtime of PV in seconds and the verification result for the demographic parity and the parity of qualified persons fairness notions. Here, ‘TO’ indices timeout (900s).



Regarding the runtime, using realistic input distributions accelerates PV, except for one benchmark instance (NN<sub>3,2</sub>, Bayes Net 2 clipped, Parity of Qualified Persons). For brevity, we call this benchmark instance ‘NN<sub>3,2</sub>BN2cQual’ in the following. For NN<sub>3,2</sub>BN2cQual, PV can not verify or disprove fairness in 15 minutes under a realistic input distribution, compared to 56 seconds under an unrealistic input distribution. Similarly, PV also exceeds the timeout on NN<sub>3,2</sub>BN2cQual when using the LongestEdge or BaBSB-LongestEdge-10 heuristics instead of BaBSB.

Studying the progress of PV on NN<sub>3,2</sub>BN2cQual more closely reveals that ProbabilityBounds converges very slowly for the probabilities involved in NN<sub>3,2</sub>BN2cQual. This indicates that NN<sub>3,2</sub> has an intricate decision boundary in a region that has substantial probability mass under Bayes Net 2 clipped, while this region has relatively lower probability mass in Bayes Net 2 because the input space is larger.

### D.3 ACAS Xu Safety

We provide a comparison of ProbabilityBounds with  $\varepsilon$ -ProVe [46] on all 36 ACAS Xu networks to which property  $\phi_2$  of Katz et al. [39] applies. These results are contained in Table 5, revealing that the networks in Table 2 are outliers. However, ProbabilityBounds still computes tighter sound bounds faster than  $\varepsilon$ -ProVe computes probably sound bounds for 13 of the 36 networks.

Table 6 contains additional results for running ProbabilityBounds with a finer grid of time budgets on the networks from Table 2 and two additional challenging instances from Katz et al. [39].

### D.4 ACAS Xu Robustness

Table 7 contains the bounds computed by ProbabilityBounds for each reference input and each output class: COC (Clear-of-Conflict), WL (steer Weak Left), WR (steer Weak Right), SL (steer Strong Left), and SR (steer Strong Right). The table reveals that the ACAS Xu network  $N_{1,1}$  could tend to classify inputs as Clear-of-Conflict (COC), regardless of the class assigned to the reference input. This insight does not agree with the insight drawn by Converse et al. [26]. However, both results are based on a tiny sample of the input space of only 25 points. Obtaining valid results requires considering a significantly larger sample of the input space or quantifying global robustness [38].

### D.5 MiniACSIncome

We provide additional details on the MiniACSIncome benchmark, including how we construct the dataset and train the networks.

#### D.5.1 Dataset

The MiniACSIncome benchmarks are built by sampling about 100 000 entries from the ACS PUMPS 1-Year horizon data for all states of the USA for the year 2018 using the folktabs Python package [27]. In line with ACSIncome [27], we only sample individuals older than 16 years, with a yearly income of at least \$100, reported working hours per week of at least 1, and a ‘PWGTP’ (more details in Ding et al. [27]) of at least 1. In total, our dataset contains 102 621 samples.

**Variable Order.** For obtaining the benchmark MiniACSIncome- $i$ ,  $i \in \{1, \dots, 8\}$ , we select  $i$  input variables in the following order: ‘SEX’, ‘COW’, ‘SCHL’, ‘WKHP’, ‘MAR’, ‘RAC1P’, ‘RELP’, ‘AGEP’. We choose ‘SEX’ as the first variable so that we can verify the fairness with respect to ‘SEX’ on every benchmark instance. The order of the remaining variables is chosen based on each variable’s expected predictive value and the number of discrete values. In particular, we select ‘COW’ (class of work), ‘SCHL’ (level of education), and ‘WKHP’ (work hours per week) first, as we consider these variables to be more predictive than ‘MAR’ (marital status), ‘RAC1P’ (races of a person), ‘RELP’

$\phi$	net	ProbabilityBounds			$\varepsilon$ -ProVe	
		10s $\ell, u$	30s $\ell, u$	60s $\ell, u$	99.9% confid. $u$	Rt
$\varphi_2$	$N_{2,1}$	0.03%, 3.20%	0.09%, 2.52% $\leftarrow$	0.17%, 2.07%	2.58%	64s
	$N_{2,2}$	0.00%, 3.54%	0.09%, 2.92%	0.25%, 2.62%	2.49% $\leftarrow$	44s
	$N_{2,3}$	0.52%, 3.45% $\leftarrow$	0.87%, 2.93%	1.00%, 2.68%	3.58%	55s
	$N_{2,4}$	0.00%, 2.65% $\leftarrow$	0.03%, 2.13%	0.04%, 2.06%	2.78%	66s
	$N_{2,5}$	0.03%, 4.22%	0.12%, 3.69%	0.35%, 3.13%	3.06% $\leftarrow$	47s
	$N_{2,6}$	0.01%, 3.65%	0.17%, 2.86%	0.31%, 2.49%	2.32% $\leftarrow$	45s
	$N_{2,7}$	0.30%, 5.54%	0.85%, 4.84%	1.27%, 4.31%	4.04% $\leftarrow$	47s
	$N_{2,8}$	0.98%, 3.66%	1.24%, 2.99% $\leftarrow$	1.39%, 2.77%	3.28%	53s
	$N_{2,9}$	0.01%, 2.85%	0.08%, 1.63%	0.10%, 1.36%	0.58% $\leftarrow$	11s
	$N_{3,1}$	0.23%, 4.25%	0.49%, 3.36%	0.88%, 2.74% $\leftarrow$	2.84%	40s
	$N_{3,2}$	0.00%, 3.57%	0.00%, 1.44%	0.00%, 1.15%	0.00% $\leftarrow$	1s
	$N_{3,3}$	0.00%, 2.88%	0.00%, 1.99%	0.00%, 0.84%	0.00% $\leftarrow$	1s
	$N_{3,4}$	0.00%, 3.26%	0.10%, 1.95%	0.13%, 1.53%	1.36% $\leftarrow$	31s
	$N_{3,5}$	0.33%, 2.84%	0.52%, 2.31% $\leftarrow$	0.63%, 2.01%	2.67%	42s
	$N_{3,6}$	0.00%, 6.07%	0.00%, 5.41%	0.02%, 5.18%	2.69% $\leftarrow$	32s
	$N_{3,7}$	0.00%, 5.76%	0.00%, 4.56%	0.00%, 3.32%	0.91% $\leftarrow$	30s
	$N_{3,8}$	0.15%, 5.39%	0.21%, 4.45%	0.32%, 3.46%	2.79% $\leftarrow$	61s
	$N_{3,9}$	0.29%, 4.80%	1.06%, 4.00%	1.39%, 3.72%	3.52% $\leftarrow$	32s
	$N_{4,1}$	0.00%, 2.93%	0.01%, 1.94%	0.04%, 1.56%	1.16% $\leftarrow$	26s
	$N_{4,2}$	0.00%, 2.64%	0.00%, 1.67%	0.00%, 1.37%	0.00% $\leftarrow$	1s
	$N_{4,3}$	0.17%, 2.92%	0.37%, 2.62%	0.61%, 2.26% $\leftarrow$	2.43%	41s
	$N_{4,4}$	0.05%, 2.47%	0.17%, 2.22%	0.27%, 1.98%	1.94% $\leftarrow$	40s
	$N_{4,5}$	0.00%, 4.55%	0.15%, 3.46%	0.50%, 3.01%	2.85% $\leftarrow$	41s
	$N_{4,6}$	0.32%, 4.54%	0.92%, 3.48%	1.16%, 3.27%	3.22% $\leftarrow$	39s
	$N_{4,7}$	0.09%, 4.17%	0.30%, 3.34%	0.56%, 2.90%	2.58% $\leftarrow$	30s
	$N_{4,8}$	0.18%, 3.83%	0.56%, 3.20% $\leftarrow$	0.86%, 2.83%	3.51%	52s
	$N_{4,9}$	0.00%, 3.31%	0.00%, 1.95%	0.00%, 1.55%	0.39% $\leftarrow$	11s
	$N_{5,1}$	0.03%, 2.59%	0.31%, 2.10% $\leftarrow$	0.42%, 1.99%	2.14%	35s
	$N_{5,2}$	0.18%, 2.64% $\leftarrow$	0.47%, 2.03%	0.52%, 1.92%	3.22%	67s
	$N_{5,3}$	0.00%, 1.72%	0.00%, 0.72%	0.00%, 0.54%	0.00% $\leftarrow$	1s
	$N_{5,4}$	0.01%, 2.95%	0.14%, 2.33%	0.23%, 2.08% $\leftarrow$	2.31%	54s
	$N_{5,5}$	0.74%, 3.28%	1.32%, 2.68% $\leftarrow$	1.37%, 2.61%	2.77%	33s
	$N_{5,6}$	0.20%, 4.41% $\leftarrow$	0.79%, 3.18%	0.98%, 3.02%	4.82%	74s
	$N_{5,7}$	1.20%, 4.61%	1.73%, 4.03% $\leftarrow$	1.99%, 3.75%	4.27%	45s
	$N_{5,8}$	0.90%, 4.13%	1.33%, 3.36% $\leftarrow$	1.55%, 3.10%	3.38%	42s
	$N_{5,9}$	0.65%, 3.72%	1.26%, 3.04% $\leftarrow$	1.42%, 2.89%	3.06%	36s
# $\leftarrow$		4	9	3	20	

Table 5: **Comparison of ProbabilityBounds and  $\varepsilon$ -ProVe for ACAS Xu Safety.** A ProbabilityBounds column is marked with an arrow ( $\leftarrow$ ) if ProbabilityBounds computes a tighter upper bound than  $\varepsilon$ -ProVe within a certain time budget. If this is not the case for any time budget, the  $\varepsilon$ -ProVe entry is marked with an arrow ( $\leftarrow$ ). In total, ProbabilityBounds computes tighter upper bounds within 60s in 16 cases. In 13 cases, it even computes a tighter bound faster than  $\varepsilon$ -ProVe.

		Timeout					
		10s		30s		1m	
$\phi$	net	$\ell, u$	$u - \ell$	$\ell, u$	$u - \ell$	$\ell, u$	$u - \ell$
$\phi_2$	$N_{4,3}$	0.17%, 2.92%	2.74%	0.37%, 2.62%	2.25%	0.61%, 2.26%	1.65%
	$N_{4,9}$	0.00%, 3.31%	3.31%	0.00%, 1.95%	1.95%	0.00%, 1.55%	1.55%
	$N_{5,8}$	0.90%, 4.13%	3.23%	1.33%, 3.36%	2.03%	1.55%, 3.10%	1.55%
$\phi_7$	$N_{1,9}$	0.00%, 98.71%	98.71%	0.00%, 94.02%	94.02%	0.00%, 87.29%	87.29%
$\phi_8$	$N_{2,9}$	0.00%, 75.92%	75.92%	0.00%, 65.08%	65.08%	0.00%, 57.82%	57.82%
		10m		1h			
$\phi$	net	$\ell, u$	$u - \ell$	$\ell, u$	$u - \ell$		
$\phi_2$	$N_{4,3}$	0.96%, 1.92%	0.97%	1.12%, 1.75%	0.62%		
	$N_{4,9}$	0.03%, 0.51%	0.48%	0.08%, 0.29%	0.21%		
	$N_{5,8}$	1.90%, 2.66%	0.76%	1.97%, 2.57%	0.59%		
$\phi_7$	$N_{1,9}$	0.00%, 51.66%	51.66%	0.00%, 30.45%	30.45%		
$\phi_8$	$N_{2,9}$	0.00%, 34.30%	34.30%	0.00%, 15.23%	15.23%		

Table 6: **Extended Probability Bounds Results for ACAS Xu Safety.**

(relationship), and ‘AGEP’ (age of a person). The variables are ordered by their number of discrete values within these groups of expected predictive value. For example, ‘COW’ has nine categories, while ‘WKHP’ has 99 possible integer values.

Table 8 contains the number of input dimensions and the total number of discrete values in each MiniACSIncome- $i$  input space. The input space contains more dimensions than input variables due to the one-hot encoding of all categorical variables.

### D.5.2 Input Distribution

The Bayesian Network input distribution of MiniACSIncome has the network structure depicted in Figure 6. For using ‘AGEP’ as a parent node, we summarised the age groups 17–34, 35–59, and 60–95. This means that the conditional probability table of ‘MAR’ does not have 78 entries for ‘AGEP’, but three, corresponding to the ranges 17–34, 35–59, and 60–95.

To fit the Bayesian Network, we walk the network from sources to sinks and fit each conditional distribution to match the empirical distribution of the data subset that matches the current condition. For example, for fitting the conditional distribution of ‘SCHL’ given ‘SEX=1’, ‘RAC1P=1’, we select the samples in the dataset having ‘SEX=1’ and ‘RAC1P=1’ and fit the conditional distribution of ‘SCHL’ to match the empirical distribution of these samples. We use categorical distributions for all categorical variables and ‘WKHP’. For ‘AGEP’, we fit a mixture model of four truncated normal distributions that we discretise to integer values.

Figure 7 depict the marginal distributions of 10 000 samples from the fitted Bayesian Network and the empirical marginal distributions of the MiniACSIncome-8 dataset. Figure 8 contains the correlation matrix of the same sample compared to the correlation matrix of MiniACSIncome-8. The Bayesian Network approximates the empirical marginal distribution and the correlation structure of MiniACSIncome-8 reasonably well.

Note that for real-world fairness verification, the input distribution should not be fitted to the same dataset on which the neural network to verify is trained on. Instead, the input distribution needs to be carefully constructed by domain experts. For fairness audits, the input distribution could also be designed adversarially by a fairness auditing entity.

Reference Input			Target Label											
			COC		WL		WR		SL		SR			
Label	Input #	$\ell, u$	Rt	$\ell, u$	Rt	$\ell, u$	Rt	$\ell, u$	Rt	$\ell, u$	Rt	$\ell, u$	Rt	
COC	1	100%, 100%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	
COC	2	99.9%, 100%	1s	0.0%, 0.0%	1s	0.0%, 0.1%	1s	0.0%, 0.0%	1s	0.0%, 0.0%	1s	0.0%, 0.0%	1s	
COC	3	91.7%, 91.8%	3s	1.3%, 1.4%	6s	1.9%, 2.0%	4s	2.1%, 2.2%	12s	2.8%, 2.8%	4s	2.8%, 2.8%	4s	
COC	4	100%, 100%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	
COC	5	100%, 100%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	0.0%, 0.0%	0s	
WL	1	89.9%, 90.0%	2s	1.6%, 1.7%	39s	4.7%, 4.8%	2s	3.6%, 3.7%	39s	0.0%, 0.0%	1s	0.0%, 0.0%	1s	
WL	2	92.9%, 93.0%	3s	2.3%, 2.4%	2s	3.0%, 3.1%	3s	1.4%, 1.5%	3s	0.2%, 0.3%	1s	0.2%, 0.3%	1s	
WL	3	90.4%, 90.5%	3s	1.1%, 1.2%	8s	3.6%, 3.7%	4s	3.8%, 3.9%	11s	0.8%, 0.9%	2s	0.8%, 0.9%	2s	
WL	4	83.6%, 83.7%	18s	5.4%, 5.5%	53s	2.6%, 2.7%	51s	4.6%, 4.7%	57s	3.5%, 3.6%	60s	3.5%, 3.6%	60s	
WL	5	96.8%, 96.9%	2s	2.9%, 3.0%	2s	0.0%, 0.1%	1s	0.0%, 0.0%	1s	0.1%, 0.2%	1s	0.1%, 0.2%	1s	
WR	1	62.2%, 62.3%	17s	0.0%, 0.0%	1s	3.4%, 3.5%	19s	15.8%, 15.9%	9s	18.5%, 18.6%	12s	18.5%, 18.6%	12s	
WR	2	96.6%, 96.7%	1s	1.5%, 1.6%	1s	1.7%, 1.8%	2s	0.0%, 0.1%	1s	0.0%, 0.1%	1s	0.0%, 0.1%	1s	
WR	3	94.8%, 94.9%	3s	0.0%, 0.1%	28s	1.9%, 2.0%	209s	0.0%, 0.1%	5s	3.2%, 3.3%	122s	3.2%, 3.3%	122s	
WR	4	94.3%, 94.4%	2s	1.8%, 1.9%	2s	3.4%, 3.5%	2s	0.3%, 0.4%	2s	0.0%, 0.1%	1s	0.0%, 0.1%	1s	
WR	5	91.1%, 91.1%	4s	0.8%, 0.9%	7s	4.2%, 4.3%	9s	3.0%, 3.1%	12s	0.7%, 0.8%	4s	0.7%, 0.8%	4s	
SL	1	81.1%, 81.2%	22s	2.3%, 2.4%	77s	4.8%, 4.9%	149s	4.8%, 4.9%	66s	6.7%, 6.8%	137s	6.7%, 6.8%	137s	
SL	2	93.8%, 93.9%	3s	1.1%, 1.2%	25s	3.7%, 3.8%	44s	0.7%, 0.8%	11s	0.5%, 0.6%	17s	0.5%, 0.6%	17s	
SL	3	83.3%, 83.4%	9s	2.8%, 2.9%	14s	6.5%, 6.6%	40s	2.1%, 2.2%	12s	5.0%, 5.1%	28s	5.0%, 5.1%	28s	
SL	4	81.8%, 81.9%	17s	3.3%, 3.4%	92s	5.2%, 5.3%	136s	5.6%, 5.7%	82s	4.0%, 4.1%	56s	4.0%, 4.1%	56s	
SL	5	84.9%, 85.0%	14s	1.7%, 1.8%	24s	5.4%, 5.5%	40s	3.0%, 3.1%	25s	4.8%, 4.9%	39s	4.8%, 4.9%	39s	
SR	1	91.0%, 91.0%	5s	0.0%, 0.1%	3s	5.2%, 5.3%	63s	0.0%, 0.1%	2s	3.6%, 3.7%	36s	3.6%, 3.7%	36s	
SR	2	87.0%, 87.1%	7s	0.9%, 1.0%	5s	5.7%, 5.8%	69s	1.2%, 1.3%	6s	5.0%, 5.1%	86s	5.0%, 5.1%	86s	
SR	3	88.4%, 88.5%	3s	2.5%, 2.6%	9s	4.7%, 4.8%	9s	2.6%, 2.7%	14s	1.5%, 1.6%	9s	1.5%, 1.6%	9s	
SR	4	93.0%, 93.1%	6s	0.7%, 0.8%	60s	3.7%, 3.8%	118s	1.0%, 1.1%	30s	1.4%, 1.5%	41s	1.4%, 1.5%	41s	
SR	5	79.3%, 79.4%	6s	0.2%, 0.3%	2s	9.1%, 9.2%	10s	3.7%, 3.8%	5s	7.4%, 7.5%	6s	7.4%, 7.5%	6s	

Table 7: **ACAS Xu Robustness.** We report the lower and upper bounds ( $\ell, u$ ) ProbabilityBounds computes for the probability that a perturbed input is classified as the target label and the runtime in seconds (Rt) of ProbabilityBounds for computing these bounds. For each combination of unperturbed label, input, and target label, we run ProbabilityBounds until the difference between the lower and the upper bound is at most 0.1%.

	#Input Variables							
	1	2	3	4	5	6	7	8
#Input Dimensions	2	10	34	35	40	49	67	68
#Discrete Values	2	16	382	38K	190K	1.7M	31M	2B
PV Runtime	16s	41s	121s	219s	367s	675s	1314s	TO
10-Neuron Network Fair?	×	×	×	×	×	×	×	?

Table 8: MiniACSIncome Details and PV Verification Results.

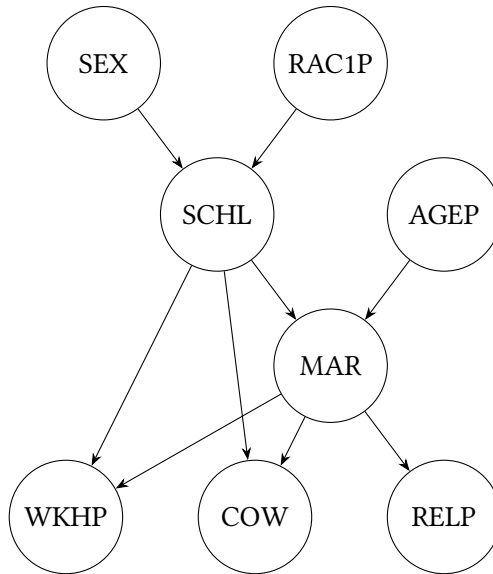


Figure 6: MiniACSIncome Bayesian Network Structure.

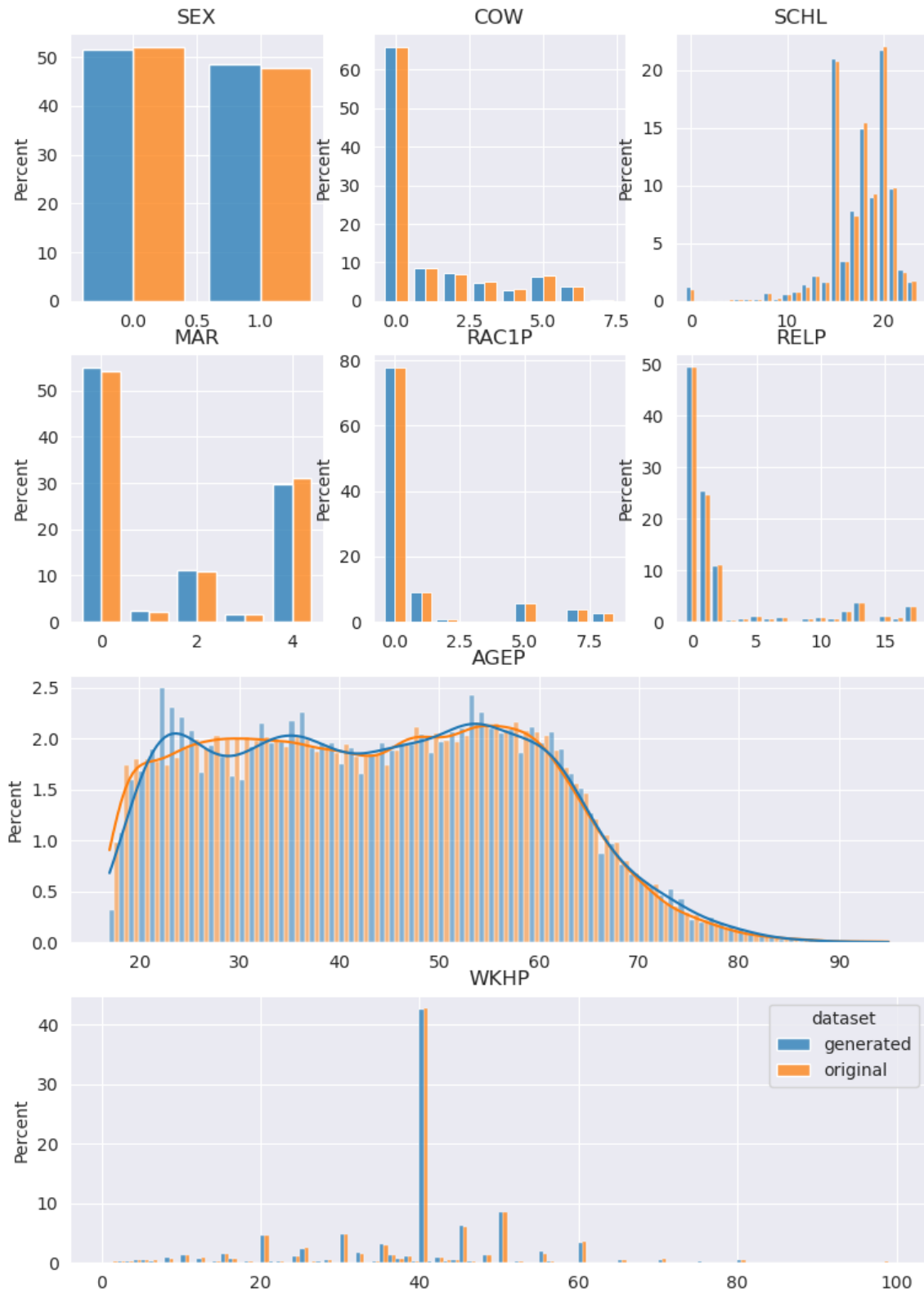


Figure 7: **MiniACSIIncome Bayesian Network — Marginal Distributions.** The ‘generated’ data is sampled from the Bayesian Network, while the ‘original’ data is the MiniACSIIncome-8 dataset.



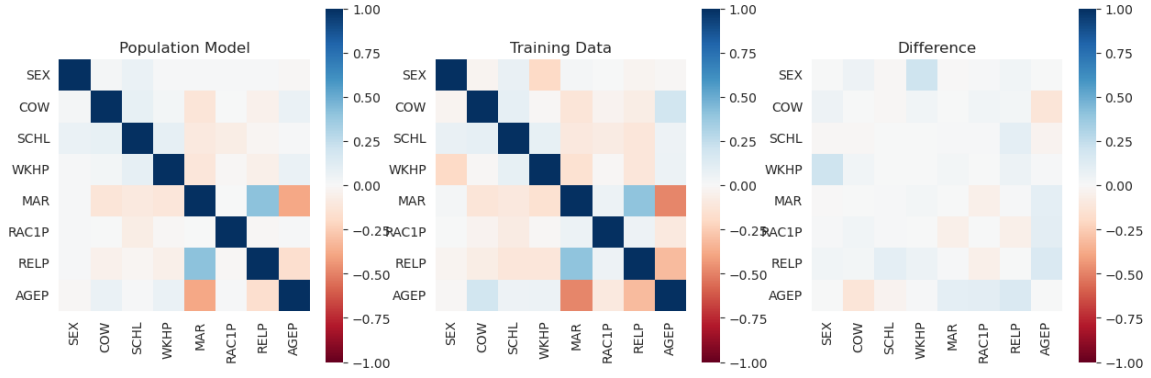


Figure 8: **MiniACSIncome Bayesian Network — Correlation Matrix.** ‘Population Model’ denotes the fitted Bayesian Network, while ‘Training Data’ stands for the full MiniACSIncome-8 dataset.

### D.5.3 Training

All MiniACSIncome neural networks are trained on a 56%/14%/30% split of the MiniACSIncome- $i$  dataset into training, validation, and testing data. This split is identical for all MiniACSIncome- $i$  datasets. All networks are trained using: the Adam optimiser [40], cross entropy as loss function, no  $L_2$  regularisation (weight decay),  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\varepsilon = 10^{-8}$  as suggested by Kingma and Ba [40], and a learning rate decay by 0.1 after 2000 and 4000 iterations. The learning rate and the number of training epochs are contained in Table 9. We perform five random restarts for each network and select the network with the lowest cross-entropy on the validation data.

Table 10 contains the accuracy, precision, and recall for the overall test set, the persons with female sex in the test set, and the persons with male sex in the test set. Additionally, the table contains whether a network satisfies the demographic parity fairness notion according to PV.

We used Optuna [3] for an initial exploration of the hyperparameter space but did not apply automatic hyperparameter optimisation to obtain the final training hyperparameters.

Dataset	Architecture	Learning Rate	# Epochs
MiniACSIIncome-1	$1 \times 10$	0.0001	1
MiniACSIIncome-2	$1 \times 10$	0.001	2
MiniACSIIncome-3	$1 \times 10$	0.001	3
MiniACSIIncome-4	$1 \times 10$	0.001	4
MiniACSIIncome-5	$1 \times 10$	0.001	5
MiniACSIIncome-6	$1 \times 10$	0.001	3
MiniACSIIncome-7	$1 \times 10$	0.001	3
MiniACSIIncome-8	$1 \times 10$	0.001	3
MiniACSIIncome-4	$1 \times 1000$	0.001	2
MiniACSIIncome-4	$1 \times 2000$	0.001	2
MiniACSIIncome-4	$1 \times 3000$	0.001	2
MiniACSIIncome-4	$1 \times 4000$	0.001	2
MiniACSIIncome-4	$1 \times 5000$	0.001	2
MiniACSIIncome-4	$1 \times 6000$	0.001	2
MiniACSIIncome-4	$1 \times 7000$	0.001	2
MiniACSIIncome-4	$1 \times 8000$	0.001	2
MiniACSIIncome-4	$1 \times 9000$	0.001	2
MiniACSIIncome-4	$1 \times 10\ 000$	0.001	2
MiniACSIIncome-4	$2 \times 10$	0.001	2
MiniACSIIncome-4	$3 \times 10$	0.001	2
MiniACSIIncome-4	$4 \times 10$	0.001	2
MiniACSIIncome-4	$5 \times 10$	0.001	2
MiniACSIIncome-4	$6 \times 10$	0.001	2
MiniACSIIncome-4	$7 \times 10$	0.001	2
MiniACSIIncome-4	$8 \times 10$	0.001	2
MiniACSIIncome-4	$9 \times 10$	0.001	2
MiniACSIIncome-4	$10 \times 10$	0.001	2

Table 9: **MiniACSIIncome Training Hyperparameters.**

Dataset	Net	Overall			Female			Male			Fair?
		A	P	R	A	P	R	A	P	R	
mACSI-1	1×10	57%	44%	63%	72%	–	0%	44%	44%	100%	×
mACSI-2	1×10	65%	55%	14%	71%	–	0%	58%	55%	23%	×
mACSI-3	1×10	73%	67%	48%	76%	65%	35%	69%	68%	55%	×
mACSI-4	1×10	75%	68%	60%	78%	66%	48%	72%	69%	66%	×
mACSI-5	1×10	76%	69%	63%	79%	65%	53%	74%	71%	69%	×
mACSI-6	1×10	77%	69%	63%	79%	65%	55%	74%	72%	68%	×
mACSI-7	1×10	77%	71%	64%	79%	67%	50%	76%	72%	72%	×
mACSI-8	1×10	78%	70%	68%	80%	67%	56%	76%	71%	75%	?
mACSI-4	1×1000	75%	71%	54%	79%	70%	44%	72%	71%	61%	×
mACSI-4	1×2000	75%	65%	67%	78%	61%	60%	72%	67%	71%	✓
mACSI-4	1×3000	74%	63%	71%	77%	58%	67%	72%	66%	73%	✓
mACSI-4	1×4000	75%	71%	55%	79%	69%	44%	72%	71%	61%	×
mACSI-4	1×5000	75%	69%	58%	78%	71%	39%	73%	68%	69%	×
mACSI-4	1×6000	75%	69%	59%	79%	69%	44%	72%	68%	67%	×
mACSI-4	1×7000	75%	66%	64%	78%	62%	55%	72%	68%	68%	×
mACSI-4	1×8000	75%	66%	64%	78%	63%	55%	72%	68%	69%	×
mACSI-4	1×9000	75%	70%	53%	78%	66%	47%	71%	72%	56%	×
mACSI-4	1×10 000	75%	67%	63%	79%	69%	45%	72%	66%	73%	×
mACSI-4	2×10	75%	67%	59%	78%	65%	48%	72%	69%	65%	×
mACSI-4	3×10	75%	68%	58%	79%	67%	47%	72%	69%	65%	×
mACSI-4	4×10	75%	68%	58%	78%	66%	46%	72%	69%	66%	×
mACSI-4	5×10	75%	68%	59%	78%	67%	47%	72%	69%	66%	×
mACSI-4	6×10	75%	67%	61%	78%	67%	46%	72%	67%	70%	×
mACSI-4	7×10	75%	65%	66%	78%	63%	52%	72%	65%	74%	×
mACSI-4	7×10	75%	67%	59%	78%	65%	46%	72%	68%	67%	×
mACSI-4	9×10	75%	67%	59%	78%	65%	47%	72%	68%	67%	×
mACSI-4	19×10	75%	67%	61%	78%	64%	49%	72%	68%	68%	×

Table 10: **MiniACSIIncome Neural Networks.** The abbreviation ‘mACSI-*i*’ stands for MiniACSIIncome-*i*. We report the accuracy (A), precision (P), and recall (R) of each trained network (Net) for the whole test dataset (Overall), the persons with ‘SEX=2’ in the test set (Female), and the persons with ‘SEX=1’ in the test set (Male). Additionally, we report whether the network satisfies the demographic parity fairness notion according to PV (Fair?).

## E Heuristics for ProbBounds

This section contains motivation and additional implementation details for our BaBSB Split heuristic that we introduce in Section 4.2. Additionally, we experimentally compare the LongestEdge, BaBSB, and BaBSB-LongestEdge-k heuristics, as well as IntervalArithmetic and CROWN to justify our decision to use BaBSB and CROWN in Section 6.

### E.1 BaBSB

Our BaBSB split selection heuristic is a variation of the BaBSB heuristic for non-probabilistic neural network verification of Bunel et al. [20]. One difference is that Bunel et al. [20] use the method of Wong and Kolter [72] for estimating the improvement in bounds, while we use IntervalArithmetic. Another difference is that while Bunel et al. [20] are mainly interested in lower bounds, we are equally interested in lower and upper bounds. Therefore, we select the dimension  $d$  that yields the largest lower bound or smallest upper bound in any of the new branches, while Bunel et al. [20] select the dimension  $d$  with the largest lower bound among the smaller lower bound for the two branches originating from splitting  $d$ . We found this variant to be the most successful for our application. Bunel et al. [20] discuss further variants.

**Implementation.** We round all bounds to four decimal places to mitigate floating point issues. If several dimensions yield equal improvements in bounds, we randomly select one of these dimensions. Without this random tie-breaking, we might split a single dimension repeatedly if the IntervalArithmetic bounds are very loose. We use a separate pseudo-random number generator with a fixed seed for this tie-breaking so that BaBSB remains entirely deterministic.

### E.2 Experiments

We experimentally compare the LongestEdge, BaBSB and BaBSB-LongestEdge-k heuristics to justify our selection in Section 6. Concretely, we study BaBSB-LongestEdge-10. Additionally, we compare IntervalArithmetic to CROWN when used as ComputeBounds procedure of ProbabilityBounds. To perform the comparison, we run the different variants of PV on the FairSquare benchmark, as described in Section 6.1.

Figure 9 compares the LongestEdge, BaBSB and BaBSB-LongestEdge-10 heuristics. It contains the runtime of PV when using SelectProb, CROWN and either LongestEdge, BaBSB, or BaBSB-LongestEdge-10 as Split heuristic. While using LongestEdge is faster for four easy-to-solve benchmark instances, using it only allows solving eight instances from the FairSquare benchmark, while using BaBSB allows solving all 18 instances. Using BaBSB-LongestEdge-10 also allows us to solve all benchmark instances while requiring slightly more time per benchmark instance than using BaBSB.

Figure 10 contains the runtime of PV when using SelectProb, BaBSB and either IntervalArithmetic or CROWN as ComputeBounds procedure in ProbabilityBounds. As the figure reveals, using CROWN allows PV to terminate faster on all benchmark instances from the FairSquare benchmark. Furthermore, using IntervalArithmetic only allows PV to solve 14 benchmark instances, while CROWN enables PV to solve all 18 benchmark instances.

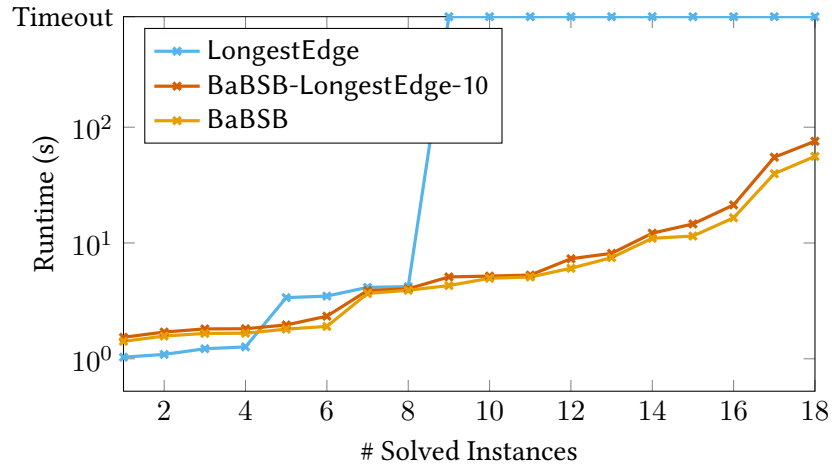


Figure 9: **Split Heuristic Comparison on the FairSquare Benchmark.** The timeout for the FairSquare benchmark is 15min.

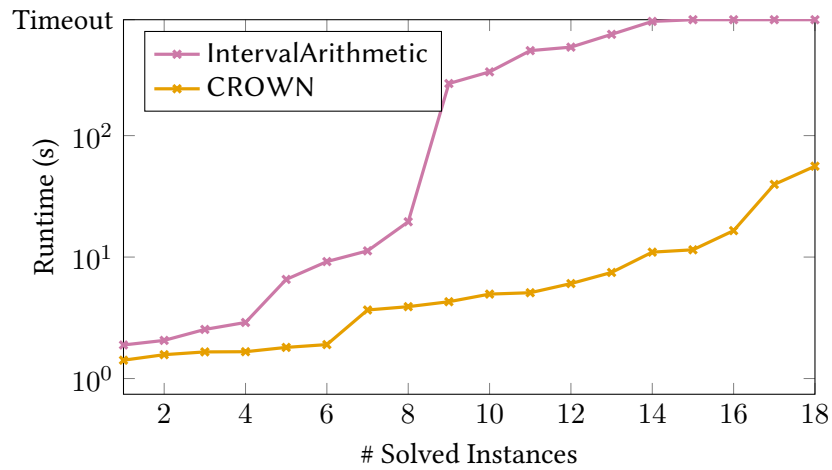


Figure 10: **ComputeBounds Procedure Comparison on the FairSquare Benchmark.** The timeout for the FairSquare benchmark is 15min.