# Probabilistic Symbolic Analysis of Neural Networks

Hayes Converse
*University of Texas at Austin*
hayesconverse@utexas.edu

Antonio Filieri
*Imperial College London*
a.filieri@imperial.ac.uk

Divya Gopinath
*KBR Inc., NASA Ames*
divya.gopinath@nasa.gov

Corina S. Păsăreanu
*Carnegie Mellon Univ., KBR, NASA Ames*
corina.s.pasareanu@nasa.gov

***Abstract*—Neural networks are powerful tools for automated decision-making, with applications ranging from image recognition to hiring decisions and safety-critical autonomous driving. However, due to their black-box nature and large scale, reasoning about their behavior is challenging. Statistical analysis is often used to infer probabilistic properties of a network, such as its robustness to noise and inaccurate inputs or the fairness of its decisions. While scalable, statistical methods can only provide probabilistic guarantees on the quality of their results and may underestimate the impact of low probability inputs leading to undesired behavior of the network.**

**In this paper, we investigate the use of symbolic analysis and constraint solution space quantification to precisely quantify probabilistic properties in neural networks. We collect symbolic constraints corresponding to the network's response to concrete inputs, while efficiently rejecting inputs whose responses have been seen before. We further propose a quantification procedure for the collected constraints, producing arbitrarily tight, sound interval bounds on the estimated probabilities. The proposed approach is an *anytime* algorithm, increasing in precision with more paths explored. We implemented our approach in SpaceScanner and demonstrate its potential in analyzing fairness, robustness, and sensitivity properties of neural networks.**

## I. Introduction

Neural networks are increasingly being adopted for a variety of decision-making tasks, including applications in safety-critical and mission-critical domains. Because of the potential impact of wrong or biased decisions in these contexts and the requirements of certification standards, numerous analysis techniques have been proposed to reason about properties of the behavior of a neural network [1–6].

Most recent approaches focus on the synthesis of adversarial inputs tricking the network into producing the wrong classification decision [7]. Specialized solvers can provide point-wise adversarial robustness guarantees [8], but they only serve to check whether there is an input to the network that triggers an inconsistent decision within a limited region of the input domain. Typically a counter-example to the robustness check acts as evidence that the network is not robust, but beyond this there is no further information regarding the number of such adversaries or the probability of these inputs occurring (given a realistic probability distribution over the input domain).

In addition, just like humans, decision-making procedures and neural networks can have *bias* or be overly sensitive to changes in certain input variables which should never be determinant in specific application contexts – e.g., the ethnic group or gender identity of applicants should not affect the decision on whether to hire them or not. Establishing a network's bias and sensitivity is crucial to understanding its behavior and fitness for applications. However, since it is not feasible to examine and record a program's response to every possible input, probabilistic arguments are usually made to establish a program's fairness. Although formal expressions of fairness have been formulated across the entire class of decision-making programs [9], existing precise techniques can only be applied to small programs (a few lines of code). As a result, probabilistic analysis of neural networks is often performed using statistical methods [10, 11]. However such methods may provide only probabilistic guarantees that may be insufficient to assess the reliability or lack of bias in a neural network, especially against segments of the input domain that may be under-represented within the overall population and rarely sampled during a statistical analysis campaign.

In this work, we investigate the use of symbolic analysis techniques to precisely quantify the confidence of probabilistic arguments about neural networks. Specifically, we use a form of *concolic* execution to collect symbolic constraints characterizing the inputs that would lead to the same activation pattern and decision. We then use solution space quantification techniques to estimate the probability mass concentrated around the concrete execution. This allows us to compute the *probability* of executing different paths through the network.

Our work applies to feed-forward networks with Rectified Linear Unit (ReLU) activations and extends naturally to other networks with piece-wise linear computations. As the symbolic analysis is expensive in practice, we propose several strategies to coping with scalability issues.

First, we use an *amplification* procedure to analyze the possible decision space resulting from a single observed activation pattern beyond the decision taken for the concrete input that initially triggered it. Thus, although a concrete input leads to a single decision, we analyze all the paths through the network that have the same neuron activations but lead to different decisions. Second, the observed activation patterns are stored in a compact tree data structure allowing for early rejection of inputs that bring redundant information, increasing the efficiency of the analysis. Third, we show a novel use of interval solving (RealPaver [12]) to estimate solution spaces of large sets of constraints. This goes way beyond existing exact techniques (Vinci [13]). Aggressive parallelization may be used to further reduce the computation time.

We implemented our approach in a tool named SpaceScanner and evaluated its effectiveness in two application domains, fairness and robustness analysis, and compared its results with relevant state-of-the-art tools as well as statistical analysis.

The evaluation shows that SpaceScanner can compute fairness decisions which proved infeasible for previous tools, and can provide robustness measures with precise confidence guarantees, as is desirable for the analysis of safety-critical systems. In summary, we make the following contributions:

- We define a probabilistic analysis framework for neural networks that collects relevant symbolic constraints, identifying a specific activation pattern along a concrete execution and *amplifying* it by adding constraints corresponding to all the possible decisions in the network. The framework uses solution space quantification techniques to compute the probability mass for all the inputs triggering the same activations and the feasible consequent decisions. The analysis can be parameterized with input distributions representing realistic application scenarios for the neural network.
- The framework maintains a novel tree data structure that is tailored to neural networks. The structure stores partial results of the analysis and enables the early rejection of inputs that would lead to already covered activation patterns, while keeping a small memory fingerprint.
- We propose an approximate quantification strategy producing arbitrarily tight, sound interval bounds on the estimated probability. We can thus overcome the scalability limitation of exact solution space quantification methods by exploiting a novel application of interval constraint propagation and branch-and-bound interval methods.
- We implement this framework in a prototype tool called *SpaceScanner* and describe its application to probabilistic analysis of fairness and robustness/sensitivity properties.
- We provide experimental evidence for the merits of our technique in the context of smaller networks used in previous fairness studies and a larger network, ACAS Xu, used in a safety critical domain.

## II. Background

In this section we recap background notions related to neural networks and symbolic execution.

### A. Neural Networks

Neural networks [14] are a powerful subclass of decision-making applications. In this work we focus on neural network classifiers. They take in an input, such as an image, and output a classification specific to the problem they have been trained to solve. Inputs are encoded as vectors in $\mathbb{R}^d$ representing a numerical encoding from the original input format. Networks are organized in *layers* of different types, including convolutional, activation, and pooling, each of which has a number of nodes. For the purpose of our discussion, we focus on activation layersEach node from the previous layer will output into the associated node in the activation layer, which will apply an *activation function.* Common activation functions include linear rectification (ReLU) and sigmoid. The main focus of this work is on ReLU activations, which compute an affine function over their inputs $\mathbf{x}$ of the form $\mathbf{c} \cdot \mathbf{x} + b$ (where the coefficients $c_i$ and the constant $b$ are referred to as *weights* and *bias*, respectively); if this function evaluates to a positive value, the node is *activated* and outputs that value, otherwise it outputs 0. A final logits layer produces the final decisions based on the real values computed by the network, by applying e.g., a softmax function.

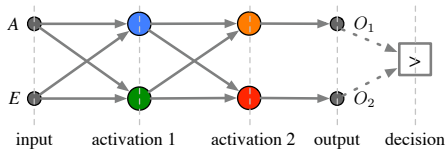



Fig. 1: A network that inputs age and education, and outputs an income decision.

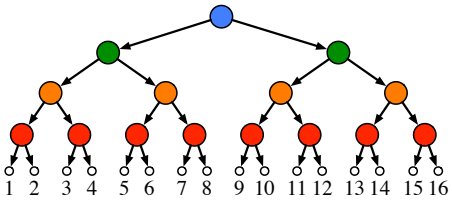



Fig. 2: Possible paths through the example network.

### B. Symbolic/Concolic Execution

Symbolic execution is a method of generating test inputs for a program [15]. Instead of executing a concrete input, symbolic execution keeps all inputs as symbolic variables and creates a path condition from the branches in the program. These path conditions are Boolean formulas that act as constraints on the input: if a formula is satisfiable, then there exists some subsection of the input space that takes that path through the program. An off-the-shelf solver can thus be used to provide test inputs that follow the associated path. However, as programs grow in size and complexity and their path conditions grow larger as more constraints are added, the time needed to solve these formulas increases exponentially, making purely symbolic execution infeasible for most real programs.

Concrete and symbolic execution can be combined in what is known as *concolic* execution: the program receives a concrete input and follows the path that input would normally take, but the symbolic execution engine still collects the relevant constraints on the input at each branch point. Neural networks can be symbolically (or concolically) executed like other imperative programs, and the ReLU activations can be interpreted as if-then-else instructions [16–18].

In this work we adapt the intuition of concolic execution to capture the salient aspects of the behavior of a network processing a specific input – its *activation pattern*. Similar to symbolic execution, we collect the constraints corresponding to each activation, which will define the notion of an *activation condition*. However, we do not collect the last *decision constraint*, coming from the logits layer, i.e., the layer that feeds in to the softmax decision, but instead *amplify* the collected activation constraints by appending to them different decision constraints. This allows us to find property violations in cases where the original concrete inputs would not exhibit them.

## III. Example

In this section, we present an overview of our work describing the analysis of an example network. Figure 1 shows the

architecture of a small fully-connected network that takes two inputs, a person's age and years of education, and outputs a decision on whether that person's income should be above a certain threshold. This network has two layers, each with two nodes, and thus a total of four ReLU activation nodes. The network is adapted from a fairness analysis problem proposed in [9], which is also part of our experiments. The original code of the network can be found at https://bit.ly/2QPQ9yG, where h1, h2 and o1, o2 correspond to the colored nodes in Figure 1.

Figure 2 shows the paths through this network, where each level of the tree represents one node of the network and each outgoing edge represents one of the two possible activation states, thus, each path is a sequence of activation states over all nodes. This tree's size is systematically exponential in the number of nodes in the network, which may prevent complete symbolic execution for larger networks. Some paths may not be feasible for a given input distribution, but they are difficult to identify outside of a complete analysis. Partial symbolic or concolic executions can be performed on the network's code, selecting only a subset of paths to be executed symbolically. However, the number of covered paths provides limited insights on the likelihood of the network producing a certain decision for a given input distribution.

Small networks, like the one in the example, can be analyzed with exhaustive probabilistic methods, as in [9]. The probabilistic analysis of larger networks currently relies on statistical methods, which generate samples from an input distribution and use statistical inference to estimate the probability of a network decision. This inference process continues until a target probabilistic termination criterion is satisfied or a maximum number of samples has been collected.

Consider the decision $O_2 > O_1$ (which corresponds in the original example to a "high income" decision, which we hereon refer to as `True`). Generating 100,000 random inputs according to the population distribution specified in the code of the example (function `popModel` at https://bit.ly/2QPQ9yG), we estimated the probability of the network producing `True` at $52.261\%$ (ratio between the number of times the network resulted in $O_2 > O_1$ over the total number of samples). However, collecting the symbolic path conditions covered by the same inputs (which is all of them in this small example) and quantifying the probability of their solution space we obtained that the probability of `True` should instead be $61.007\%$. The finite number of samples from the input distribution produced skewed results because it did not allow us to collect enough evidence from more rare segments of the input population, making the estimation of the probability of `True` imprecise.

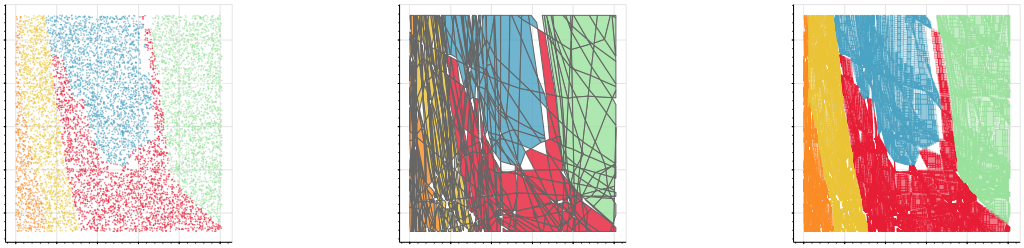
This probabilistic analysis based on solution space quantification for symbolic constraints is reminiscent of results in probabilistic symbolic execution [19] and allows us to obtain arbitrarily precise quantifications of the probability of the entire class of executions following a given execution path.

In this work, we will apply a similar principle to compute probabilistic properties of a neural network. In particular, a concrete input is executed on an instrumented network to extract constraints that characterize all the executions leading to the same activation pattern. The volume of the solution space of these constraints is quantified, producing a "weight" that will enable a precise assessment of the probability of the network drawing a target decision, even for portions of the input space that have low probability. Because the symbolic analysis of an activation pattern accounts for all the inputs triggering it, we devise a specialized strategy for the early rejection of redundant inputs, allowing for a efficient exploration procedure and reducing the number of volume computations. Finally, we propose an *amplification* technique that increases the chances of discovering rare conditions in the final decision layer of the network, allowing the construction of sound lower bounds on the probability of these decisions to be drawn.

## IV. Algorithm

**Algorithm 1:** Pseudocode for unweighted (uniform distribution) probability calculation

```
Choose stop criterion;
𝒟 ← target decision;
paths ← ∅ ;
ACs ← ∅ ;
while stop criterion false do
    Generate path;
    prefix ← empty;
    AC ← empty;
    for layer in network do
        ap_layer ← activation pattern at layer;
        if ap_layer in paths[prefix] and marked as done then
            Reject sample;
        else
            paths[prefix] ← paths[prefix] · ap_layer;
        for all pairs ap_1, ap_2 in paths[prefix] do
            if ap_1, ap_2 both marked as done and differ in exactly one place then
                Merge ap_1, ap_2;
        if paths[prefix] contains an all don't-care response then
            Remove paths[prefix], mark parent as done;
        AC ← AC∧ constraints at layer;
        prefix ← prefix · ap_layer;
    Mark paths[prefix] as done;
    DC ← constraints for 𝒟;
    ACs ← Union(ACs, AC ∧ DC) ;
volume ← 0;
for AC in ACs do
    volume ← volume+ Volume(AC)
totalVolume ← Volume (input space);
Return volume/totalVolume;
```



(a) 10k inputs (b) Corresponding conditions (c) Box approximations

Fig. 3: Inputs, conditions, and quantification.

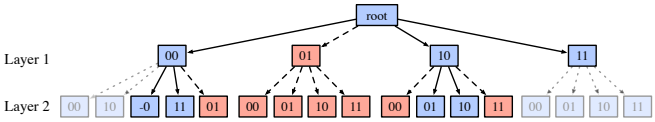



Fig. 4: Example activation tree. Pruned nodes are shaded.

The key insight of the proposed approach is the use of solution space quantification techniques to compute the probability of executing a path through the neural network. Each path corresponds to a specific activation pattern – i.e., the neurons activated when processing the input, to which we add the constraints corresponding to different decisions in the network.

An activation pattern in a ReLU network can be characterized by the conjunction of a set of linear inequalities over the input domain, one for each activated node. Each inequality represents a halfplane splitting the domain into two parts, and their conjunction defines a polytope in the input domain. We will refer to this polytope as an *activation condition* (*AC*).

The set of all activation conditions induces a partition of the input domain. Therefore, determining the activation pattern of one concrete input allows us to infer information about the entire equivalence class of inputs satisfying the corresponding activation condition. In particular, we can use the volume of the AC polytope as a measure of *how much* of the input domain would lead to the same activation. Furthermore, by adding to AC the constraints imposed by the logits layer for the desired decision, referred to as a *decision condition* (*DC*), we can use the volume computation to quantify the portion of the input space that leads to the desired final decision.

Algorithm 1 summarizes the core routine, which aims to compute the probability of the network to give a desired decision $\mathcal{D}$. Assume for now that the inputs are distributed uniformly (we will relax this assumption later in the paper). The algorithm has two macro phases: path generation and volume-based probability quantification. The stopping criterion is a pre-defined number of inputs or a rejection threshold.

### A. Path Generation

The path generation (or exploration) can be done systematically, as typically performed in symbolic execution, by collecting and negating activation constraints. It can also be done heuristically, by collecting the activation conditions from inputs generated from a user-defined population model. We perform the latter in our experiments, for better scalability. However our approach is general and accommodates multiple exploration procedures. More advanced generation strategies, e.g., aiming to maximize the diversity of the inputs, are orthogonal to this work's contribution and left for future investigation.

In general, an exhaustive collection of all the feasible activation conditions may be infeasibile in practice due to the large size of most realistic neural networks. Our analysis provides an *anytime* approach and it gradually improves its results with more paths explored. Furthermore the analysis is able to precisely quantify how much of the input space has been covered, providing a measure of *confidence* in the results.

When the network is executed on a given input, we compute for each node at each layer the condition on the input values that leads to its observed activation status. Notably, a ReLU node is activated when an affine function of its inputs evaluates to a positive value. If activated, it produces as output the value of the affine function, otherwise, it produces as output 0. Since the output of each node in the network is a (conditional) analytical expression of its inputs, the activation condition of a node can always be expressed as an affine condition on the initial inputs of the network, resulting in a constraint of the form $\mathbf{c} \cdot \mathbf{x} + b > 0$ (where $\mathbf{x}$ is the input vector, $\mathbf{c}$ is a vector of coefficients specific to the node, and $b$ is a scalar bias value). If the node is not activated, its respective activation condition is naturally of the form $\mathbf{c} \cdot \mathbf{x} + b \leq 0$.

**Decision conditions.** After an input is executed all the way through the network, the activation condition that accompanies it is finished with a *decision condition* (DC) (lines 23-24 in the algorithm), a small set of constraints that represent the network's decision in the final layer to output desired class $\mathcal{D}$. For example, a network with $n$ nodes at its output layer representing $n$ possible decisions that selects the greatest value among its output nodes would have a decision constraint of the form $\bigwedge_{i \neq k} o_i > o_k$, for desired class $\mathcal{D} = i$. Note that given an AC, we append to it a DC corresponding to $\mathcal{D}$ even if the original input led to a different class. In this way we **amplify** the effect of the input, by searching for all possible inputs that have same activation but lead to the different class.

Note that in practice we may invoke the algorithm for multiple desired decisions (i.e., target classes), in which case multiple decision constraints are added to the same activation constraint, resulting in a set of constraints on which we apply the volume computation. Note also that with an abuse of notation we use $AC$ to also denote $AC \wedge DC$ when the context is clear (as in lines 26-27 in the pseudo-code).

**Activation tree.** The paths explored by the algorithm are organized in a tree to enable early rejection of inputs whose

activations have been already seen. For each network layer, we define the activation pattern as a bitvector, whose n-th bit represents the activation state of the n-th node in the layer (for a fixed, arbitrary order of the nodes in the layer). For an input $\mathbf{x}$, the layer's activation condition corresponds to the conjunction of the activation conditions of the layer's nodes.

To keep track of the explored activation patterns we define a tree data structure (which we call $paths$ in Algorithm 1, initialized as an empty tree in line 4), where each level $l$ after the root contains all the activation patterns observed in the $l$-th layer of the network. Figure 4 shows an example activation tree. Nodes in blue have been activated by at least one sample input. Nodes in red have not been activated by any sample; they are pictured only to demonstrate that the tree is usually sparsely populated at any time during the exploration as not all the layer activation patterns may have been covered or even be feasible. As a sample is processed, new nodes corresponding to new activation patterns are added to each level of the tree if necessary (line 14); we use $paths[prefix]$ as a shorthand for the children of the node found by following the tree down through the nodes corresponding to the current input's response at each previous layer, i.e. the set of nodes representing the responses seen thus far for inputs with the same response at layers 1 through $l-1$.

**Rejection and pruning.** An input triggering an activation pattern already covered in the activation tree does not bring any additional information and can be rejected. To allow for early rejection of uninformative inputs and keep the activation tree more compact, we introduce a pruning procedure to compress the representation of covered activation patterns.

We employ a **roll-up** mechanism to enable this early rejection of inputs. Whenever all possible activation patterns following a given activation pattern at a layer are covered, we roll-up the ensuing layers of the tree and store only the activation pattern at the given layer as a representative of the ensuing activation patterns (lines 18, 19 in Algorithm 1). Therefore, when the activation pattern of the input currently being processed matches a rolled-up activation pattern, it is rejected without further processing (lines 11, 12 in Algorithm 1).

An important aspect to note is that all neurons at the same layer of the neural network are independent of each other, hence the activation patterns at a layer do not have an ordering. This indicates that roll-ups can be performed within a layer, which is a key advantage of using a tree structured after the network (as opposed to a simpler binary version as seen in Figure 2). If for a subset of the activation pattern at a layer (activation statuses of a subset of neurons) all possible combinations of the activation statuses of the other neurons have been explored, then we can maintain or store just the partial activation pattern.

Consider the last internal level of the activation tree, i.e., having as children only leaves. Let there be two leaves with activation bitvectors differing only for the $i$-th bit. For example, the two leftmost leaves in Figure 4 are 00 and 10, which differ only for the first bit. In this situation, both activation states of node $i$ have been covered. We can then replace the two nodes differing by bit $i$ with a single node, whose bitvector contains at position $i$ a don't care symbol ($-$), indicating that both activation states for the corresponding node $i$ have been covered for that combination of activations. This merge operation is performed in lines 15-17 of our pseudocode, and can be performed at any layer of the tree provided that the nodes in question are leaves (we equivalently refer to them in the pseudocode as "marked as done"). Leaves are created either by samples being processed through every layer without rejection (line 22) or through roll-ups.

A roll-up occurs when all the children of a node have been fully explored, and the node itself becomes a leaf and can be compressed in the context of its parent. We can tell when this is the case in our tree when all children of a node have been merged into a single node with don't care symbols at every position. We check for the existence of this sort of child at line 18, rolling it up and marking the now childless parent as "done" in line 19 if appropriate. In Figure 4, the subtree below node 11 in the first layer has been rolled up, making that node a leaf and allowing any future sample with that response at layer 1 to be immediately rejected. This is to say that besides saving memory, this procedure allows for earlier rejection of inputs, as they can be discarded as soon as they reach a fully explored node, reducing the analysis time. Please note that using just a symbolic execution tree such as in Figure 2 would not enable such a rejection since the path through the tree imposes an ordering of the neurons even at the same layer.

### *B. Probabilistic analysis with volume computation*

The activation pattern discovered during the path exploration together with the appended decision condition can be seen as the representative of an equivalence class on the input values satisfying the same activation condition and leading to the same decision. Such a condition, being the conjunction of affine constraints on the input, identifies a polytope in $\mathbb{R}^d$, where $d$ is the dimension of the input vector over the Reals.

Consider Figure 3, whose x and y axes represent the values of two input dimensions from an experiment with the analysis of the ACAS Xu network (which will be detailed later). The leftmost diagram plots the analyzed inputs, each represented by a point in the input space with a color corresponding to the network decision – we pictured all the generated inputs, without rejection, for the purpose of demonstration. The polytopes corresponding to the covered conditions are plotted in Figure 3b. The volume of a polytope corresponds to the *weight* of the input – how much of the input space it accounts for.

Given a distribution on the input domain, the probability for the network to produce a decision $\mathcal{D}$ can be formalized as a function of the activation conditions AC and the decision condition DC that lead to the decision $\mathcal{D}$ ($AC \wedge DC$):

$$Pr(\mathcal{D}) = \sum_{C \in AC \wedge DC} \int_{\mathbf{x}} \mathbb{1}_{(x \models C)} \cdot p(x) dx \qquad (1)$$

where the indicator function $\mathbb{1}_{(x \models C)}$ evaluates to 1 for inputs $\mathbf{x}$ satisfying condition C corresponding to the joint satisfaction of an activation condition AC and decision condition DC – lead-

ing to decision $\mathcal{D}$ – and $p(\mathbf{x})$ is the probability of generating the input $\mathbf{x}$ according to the input population distribution.

**Input distributions and probability.** Assume first that the input distribution is uniform. In this case $p(x)$ would assign the same probability to all the inputs in a polytope and the integral $\int_{\mathbf{x}} \mathbb{1}_{(x \models AC \wedge DC)} \cdot p(x)dx$ would just simplify to the ratio between the volume of the polytope and the size of the input domain $Vol(AC \wedge DC)/Vol(\mathbb{D}_x)$.

To handle non-uniform input distributions, we follow an approach similar to [19] requiring the input distribution to be discretized in an arbitrarily accurate histogram distribution. A histogram distribution is a map from a partition of the input domain to a probability value representing the cumulative probability mass of the element set of the partition. Partitions can be arbitrarily fine, allowing us to strike a trade-off between the accuracy of the discretization and the computational cost of the probabilistic analysis. Given a histogram distribution $H : s_i \rightarrow p_i$, mapping a subset of the input domain $s_i \subseteq \mathbb{D}_x$ to its probability value $p_i$ (with $\bigcup_i s_i = \mathbb{D}_x$ and $i \neq j \implies s_i \cap s_j = \emptyset$; we assume w.l.o.g. $s_i$ expressed as a boolean constraint on $\mathbf{x}$), the probability in Equation (1) can be rewritten as:

$$Pr(\mathcal{D}) = \sum_{s_i} p_i \cdot \sum_{C \in AC \wedge DC} \frac{Vol(C \wedge s_i)}{Vol(\mathbb{D}_x)} \tag{2}$$

In other words, the complexity of the computation grows linearly with the size of the support of the histogram distribution. If the sets $s_i$ are constructed as boxes (i.e., hyper-rectangles in the input domain), the computation of their probability reduces to the evaluation of the cumulative distribution function of each input of the network within each box; we adopted this discretization method for the evaluation of this work (as shown in Section VI-A, Table II). Computing an optimal discretization, where the sets $s_i$ can be arbitrary constraints, is specific to each input distribution and orthogonal to our approach.

**Volume of a polytope.** Equation (2) reduces the problem of computing the probability of a network producing a specific decision $\mathcal{D}$ for a given input distribution to the quantification of the volume of polytopes. A variety of algorithms have been studied for computing the volume of a polytope, including both exact and approximate solutions [13, 20, 21].

**Exact solutions.** To analyze networks with up to tens of distinct inputs we can use efficient off-the-shelf implementations of exact volume computation methods [13]. In particular, we adopt the implementation of Lasserre's method [22] provided by the Vinci solver [13]. This method applies directly to the representation of a polytope as a conjunction of halfplanes. In our experiments, Vinci required minutes of computation for polytopes defined by up to 200 halfplanes on 5 input variables. More efficient exact methods can be used after computing the vertices of the polytope explicitly. However, the number of vertices of a polytope may be exponential in the number of halfplanes, in the worst case, making their computation expensive for deeper networks, where the number of halfplanes grows linearly with the number of nodes.

**Sound intervals.** To analyze networks with a larger number of nodes, instead of directly computing the exact volume of the resulting polytopes, we compute an arbitrarily small interval guaranteed to contain the exact solution. For this task, we use a set of branch-and-bound and interval constraint propagation techniques implemented in Realpaver [12]. Starting from the initial input domain, Realpaver produces a set of boxes whose union is guaranteed to contain all the solutions of a given set of constraints. The set of boxes is iteratively refined to produce tighter approximations of the solution space, up to an arbitrary target accuracy (or a predefined timeout). The resulting boxes are labeled as either inner or outer boxes. Inner boxes are guaranteed to contain only solutions. Outer boxes may also contain points that do not satisfy the constraints (i.e., are outside the polytope). The volume of a box can be computed efficiently as the product of the lengths of its sides. By construction, the cumulative volume of the inner boxes is a sound lower bound of the volume of the constraining polytopes, while the total cumulative volume of inner and outer boxes is a sound upper bound of the same.

Figure 3c shows a box approximation of the polytopes in Figure 3b, where outer boxes have reduced opacity compared to the inner boxes. For this example, we allowed a maximum of 10,000 boxes for each polytope. This number can be increased to reduce the length of the interval computed via Realpaver, resulting in tighter estimated probability bounds.

In our experiments, Realpaver required approximately 12 minutes per problem for problems with over 300 constraints and 5 input variables, and a time on the order of seconds for similar problems with only 2 input variables. The number of such problems for a given experiment ranges depending on how many paths are identified during the exploration phase, but as the calls to Realpaver can be parallelized, the computation time can be reduced to a scale of tens of minutes using a larger number of cores.

**Approximate solutions.** Input spaces with higher dimensionality may challenge the scalability of our bounding method, requiring the use of statistical volume estimation methods. While state of the art statistical methods can scale on high-dimensional polytopes [21], their results are confidence intervals, which are only probabilistically correct.

Model counting can also be used when dealing with discrete input domains and has been used in the context of neural network analysis before (see related work). However, when the inputs are encoded as floating point, standard exact [23] or approximate [24] model counters are not suitable due to the non-uniform distribution of floating-point values along the Reals line – an interval with length $l$ over the reals would enclose many more floating-point values if it overlaps with the origin than if it is placed near extremes of the representable range. Finally, volume computation for polytopes can rely on more efficient solutions than more complex constraints, while the overhead for computing the volume of arbitrary activation functions may be prohibitively higher.

**Confidence measures.** When the volume of the constraints is computed exactly, the probabilistic symbolic analysis presented in this section produces as a byproduct the quantifi-

cation of the volume of the constraints computed along each explored input. The ratio between the cumulative volume of the covered constraints and the volume of the valid input domain provides a natural confidence measure for the result of the analysis, quantifying how much of the input domain has been covered. If combined with an input distribution, a straightforward adaptation of Equation (2) allows to quantify the total probability mass from the input population that has been accounted for by the analysis.

Interval analysis produces for each constraint sound lower and upper bounds. Dividing this interval by the volume of the domain provides corresponding under and upper bounds on the fraction of the domain that has been covered. For rare events, which may not have been observed at all during the path exploration therefore bringing limited statistical evidence for the corresponding executions, symbolic probabilistic analysis may produce a more accurate quantification of the probability of the corresponding executions. We will report on instances of this situation in our preliminary experimental evaluation.

### C. Limitations and Application Scope

**Complexity.** Volume computation can enhance the precision of statistical analysis, in particular for quantifying the probability of rare events. Differently from pure statistical methods, our technique does not rely on sample frequency to estimate the probability of a path, but it quantifies such probability exactly from a single occurrence. A consistent Monte Carlo estimator of the target binomial proportion (probability of violating a property or not) will eventually converge within arbitrary accuracy and confidence, given enough time. However, it is difficult to estimate a priori the number of samples required to converge. A widely used conservative bound is Chernoff-Hoeffding which requires a number of samples quadratic in the required accuracy and logarithmic in the desired statistical confidence [25]. For example, a violation probability in $O(10^{-5})$ would require up to $O(10^{-10})$ samples, neglecting the confidence's logarithmic complexity factor. Using volume computation, each non-rejected sample contributes an exact probability mass to either the satisfaction or the violation of the target property, thus shrinking the estimation interval monotonically in either case. The relevant uncertainty figure with volume computation is therefore the length of the computed estimation interval, as opposed to the variance of a statistical estimator (which can be bounded only probabilistically).

However, this increased precision relies on heavier integration and constraint propagation methods than pure Monte Carlo sampling. More precisely, their complexity deteriorates exponentially with the cardinality of the input space – which affects the cost of each individual path volume quantification – while it is less sensitive to the size of hidden layers – which may increase the maximum number of independent paths to be quantified, in parallel.

**Application scope.** These peculiar complexity trade-offs make our technique particularly valuable for networks used in critical systems (e.g., adaptive industrial controllers) or decision support systems (e.g., financial or hiring advice), which require precise guarantees for certification and usually have up to tens of inputs (sensor readings or subject attributes) and up to hundreds of densely-connected hidden nodes. Input distributions for these systems are typically inferred from dynamical models of physical phenomena or historical records, allowing tailored probabilistic analyses quantifying the probability of property violations in operational conditions (as opposed to techniques that synthesize a single adversarial input, but cannot quantify the likelihood of it occurring in reality, for example). Conversely, while theoretically applicable, our technique would not be useful, e.g., to analyze a convolutional image recognition network, where the input distribution cannot be easily formalized, and input cardinality would be prohibitively large.

**Activation functions.** We propose two quantification methods: exact and approximate. Exact quantification relies on volume computation for polytopes, and therefore yields exact results for ReLU or any piecewise linear activations. Approximate quantification relies on interval constraint propagation and branch-and-bound numerical methods. Hence, it does not require linearity and can analyze nonlinear activation functions (including, e.g., polynomial, sigmoid, or exponential). Our implementation uses RealPaver and it can therefore be applied to activation functions defined using any nonlinear function supported by the tool [12] (alternative ICP methods can replace RealPaver supporting different constraints and with different performance, e.g., IbexLib [26]). In this work we focus on ReLU activations, which can be analyzed with both methods.

## V. Applications to Fairness and Robustness

In this section, we describe two selected applications (cf. Section IV-C): fairness and robustness analysis, which will be the focus of the experiments presented in the next section.

### A. Fairness Analysis

Fairness analysis is a natural application for our technique as it can be expressed in probabilistic terms and tailored to specific input distributions (e.g., representing the customers of a certain company). We handle the challenge of conditioning the analysis on latent variables of the population model which may not be themselves input to the analyzed network.

For this analysis, we refer to the formalization of fairness defined in [9]. For instance, for the example network from Section III, fairness may be defined with respect to a latent variable of the input distribution defining the gender of an individual, which appears in the generative model of the population (see `popModel()` at https://bit.ly/2QPQ9yG) and correlates with the age and education attributes used by the network to draw its decision. Formally, we aim to check that:

$$\frac{Pr(\textit{high income}|\textit{female})}{Pr(\textit{high income}|\textit{male})} > 1 - \epsilon \tag{3}$$

Here $\epsilon$ is set according to the application (it is set to 0.15 in [9]). In other words, the network is said to be fair if the probability of a high-income decision for a female individual divided by the probability of a high-income decision for a male individual is greater than than 0.85.

TABLE I: Fairness analysis results.

| Statistical | | | | |
|---|---|---|---|---|
| Network | Pr(T) | Pr(T\|M) | Pr(T\|F) | Report |
| 3-in 2-h | 42.619% | 44.404% | 39.111% | *None* |
| 2-in 2-h | 52.261% | 52.658% | 52.753% | Fair |
| 2-in 1-h | 55.071% | 54.712% | 54.988% | Fair |

| SpaceScanner | | | | | |
|---|---|---|---|---|---|
| Network | Pr(T) (U) | Pr(T) | Pr(T\|M) | Pr(T\|F) | Report |
| 3-in 2-h | 55.415% | 52.175% | 55.254% | 53.110% | Fair |
| 2-in 2-h | 61.007% | 65.919% | 66.181% | 66.418% | Fair |
| 2-in 1-h | 52.233% | 58.483% | 58.314% | 58.140% | Fair |

### B. Robustness Analysis

For robustness analysis, our framework can be employed to provide precise estimates on the probability distribution of the output labels within the proximity of an input $x_0$. Assuming the network produced a decision $\mathcal{D}_0$ for the input, a large probability $Pr(Decision = \mathcal{D}_0)$ of obtaining the same decision over all the inputs $\mathbf{x}_i$ within a given distance from $\mathbf{x}_0$ indicates that the network is *robust* within that region. If on the other hand, the probability is small, it indicates that the network is *sensitive* to perturbations in that region. The analysis can be thus used to quantify how robust and how sensitive an analysis is wrt small perturbations of the inputs. These perturbations may be due, for example, to sensor failures or inaccuracies, as in the case of the controller analyzed in Section VI or can be adversarial (created by a malicious user). Notably, the goal of the analysis presented in this paper is not to exclude the existence of one input $\mathbf{x}_i$ that may lead to a decision different from $\mathcal{D}_0$, but to assess the probability of any such inputs being produced under a specific input distribution, which may represent, for example, a probabilistic profile of the physical environment around a sensor.

## VI. Experiments

We implemented the proposed framework in a prototype tool called SpaceScanner, in Python 2.7.15. All experiments were run on a default Google Colaboratory runtime unless otherwise specified.

### A. Fairness Analysis

We conducted our experiments on fairness using the same networks analyzed by Albarghouthi et. al to test their tool, FairSquare [9]. These are small, fully-connected networks with a single hidden layer that are meant to determine an employee's salary as "high" or "low" based on a set of their traits; our running example is one of them. The population model for these networks changes the distribution of these traits with the person's gender; when uncontrolled, male employees appear twice as often as female ones.

Two of the three networks evaluated with FairSquare take the employee's age and years of education as input variables; the third network also considers the person's capital gain in addition to these two inputs. These networks have up to two nodes in their single hidden layer and were analyzed by FairSquare using a 900-second timeout, which was able to classify both of the two-input networks as fair. FairSquare however timed out on the three-input network [9].

TABLE II: Discretized joint distribution of age and education scores for general population.

| age $n$ edu | $\leq$ -15.93 | (-15.93, -0.27] | (-0.27, 15.38] | $\geq$ 15.38 |
|---|---|---|---|---|
| $\leq$ -8.98 | $\approx 0$ | | | |
| (-8.98, -5.67] | $\approx 0$ | 0.01 | 0.01 | $\approx 0$ |
| (-5.67, -2.35] | | 1.81 | 0.94 | |
| (-2.35, 0.96] | | 25.02 | 12.67 | |
| (0.96, 4.27] | | 34.44 | 17.59 | |
| (4.27, 7.59] | | 5.04 | 2.57 | |
| (7.59, 10.90] | | 0.07 | 0.04 | |
| $\geq$ 10.90 | $\approx 0$ | | | |

TABLE III: Discretized joint distribution of age and education scores for Females / Males sub-populations.

| age $n$ edu | $\leq$ -15.93 | (-15.93, -0.27] | (-0.27, 15.38] | $\geq$ 15.38 |
|---|---|---|---|---|
| $\leq$ -8.98 | $\approx 0$ | | | |
| (-8.98, -5.67] | $\approx 0$ | 0.01 | 0.01 | $\approx 0$ |
| (-5.67, -2.35] | | 1.67 / 1.82 | 0.88 / 0.96 | |
| (-2.35, 0.96] | | 24.81 / 24.79 | 12.67 / 12.67 | |
| (0.96, 4.27] | | 35.12 / 34.12 | 17.46 / 17.63 | |
| (4.27, 7.59] | | 4.84 / 5.19 | 2.42 / 2.68 | |
| (7.59, 10.90] | | 0.07 / 0.08 | 0.04 / 0.04 | |
| $\geq$ 10.90 | $\approx 0$ | | | |

Using our framework, we explored *all* the paths through the network that had a "True" decision and used Vinci to get the volume of each. These volumes were considered both on their own and with the additional factor of different input distributions for male vs. female, as described below. This analysis was performed for each of the networks, outputting a decision for each in under 900 seconds.

The input distributions are described in the FairSquare code repository (function `popModel` at https://bit.ly/2QPQ9yG). To quantify the probability of satisfying each of the collected conditions given the input distribution, we first generated a histogram, that is, we divided the input domain of each input in disjoint subintervals (10 in our case). The cartesian product of the subintervals of each input variable defines a grid partition of the input domain in boxes. The probability of each such box can be computed independently and is reported in Table II (for the sake of readability, we only report the probability of the boxes that are larger than $0.01$ and round the values in the table to two decimal digits; the actual grid would be 10x10).

We note that the Male/Female information is not a direct input to the analyzed networks, yet the fairness analysis requires computing probabilities conditioned on this latent information. Generating a histogram entirely from samples of sub-populations defined by independent variables allows us to condition the probabilities on those variables. For instance, Table III shows the discretized distributions for the Female and Male sub-populations that we generated for our experiments.

We performed the same evaluation using a purely statistical approach with 100,000 samples. Table I shows the results from the pure statistical approach and our technique, as well as FairSquare's reported decision for the network, i.e., whether it satisfied the condition in Section V-A. A report of "None" indicates that the analysis of the network timed out or otherwise did not output a decision. For each of these categories, Pr(T) is the probability of a high-income decision using the general population model and Pr(T|M) and Pr(T|F) are the probabilities of that decision for the Male and Female sub-populations, respectively. Pr(T) (U) is the result of applying

our technique considering a uniform distribution over the input space, i.e. without using a probability histogram. In all cases of our technique, we achieve complete path coverage, thereby assigning precise probabilities and outcomes to the entire input space, thus there is no uncertainty in the results. The statistical method, by contrast, did not converge to a single correct numerical figure. Rather, using 100,000 samples, it has a 99%-confidence interval of approximately $\pm 0.0041$ ($z$-interval assuming the Binomial estimator's convergence to Gaussian), with small variations for the different probabilities estimated in Table I. SpaceScanner's Report shows whether or not the probabilities satisfied the condition in Section V-A.

As can be observed from Table I, SpaceScanner compares favorably to FairSquare as it can handle one network which FairSquare can not. For the other two networks, our technique is able to obtain similar results to FairSquare and finds the outcome to be fair with respect to gender. The pure statistical analysis yields results faster than our approach. However, although the decision with respect to fairness is the same, it can be observed that the estimated probabilities are less precise.

### *B. Robustness Analysis*

We applied our technique to perform robustness and sensitivity analysis for the **ACAS Xu** network. ACAS Xu is a safety-critical collision avoidance system for unmanned aircraft control [27]. It receives sensor information regarding the drone (the *ownship*) and any nearby intruder drones, and then issues horizontal turning advisories aimed at preventing collisions. The input sensor data includes: (1) Range: distance between ownship and intruder; (2) $\theta$: angle to intruder relative to ownship heading direction; (3) $\psi$: heading angle of intruder relative to ownship heading direction; (4) $v_{\text{own}}$: speed of ownship; and (5) $v_{\text{int}}$: speed of intruder;

The five possible output advisories are: (0) Clear-of-Conflict (COC), (1) Weak Left, (2) Weak Right, (3) Strong Left, and (4) Strong Right. Each advisory is assigned a score, with the lowest score corresponding to the best action. The FAA is exploring an implementation of ACAS Xu that uses 45 deep neural networks. We selected the first network for our analysis.

TABLE IV: ACAS Xu: input rejection rates.

| Label | Input # | Unique paths | Rejection rate (%) | Label | Input # | Unique paths | Rejection rate (%) |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 23 | 99.77 | 2 | 3 | 901 | 90.99 |
| 0 | 1 | 56 | 99.44 | 2 | 4 | 927 | 91.73 |
| 0 | 2 | 404 | 95.96 | 3 | 0 | 471 | 95.29 |
| 0 | 3 | 4 | 99.96 | 3 | 1 | 1843 | 81.57 |
| 0 | 4 | 16 | 99.84 | 3 | 2 | 841 | 91.59 |
| 1 | 0 | 1474 | 85.26 | 3 | 3 | 1050 | 89.50 |
| 1 | 1 | 921 | 90.79 | 3 | 4 | 971 | 90.29 |
| 1 | 2 | 357 | 96.43 | 4 | 0 | 699 | 93.01 |
| 1 | 3 | 1073 | 89.27 | 4 | 1 | 1147 | 88.53 |
| 1 | 4 | 253 | 97.47 | 4 | 2 | 2120 | 78.80 |
| 2 | 0 | 1672 | 82.28 | 4 | 3 | 1764 | 82.36 |
| 2 | 1 | 1533 | 84.67 | 4 | 4 | 914 | 90.86 |
| 2 | 2 | 712 | 92.88 | | | | |

The ACAS Xu network has been the subject of adversarial robustness analysis in the past [8, 28]. We employ our analysis to not only determine adversarial robustness with high confidence (when the desired label has a 100% probability), but also provide insight into the sensitivity of the network at that point with respect to other labels (when there is a non-zero distribution of probabilities across labels). Specifically, given an input $x$ and an $\epsilon$, we generate inputs from within the region defined by $||x - x'||_\infty \leq \epsilon$. We then apply our probabilistic analysis to determine the probability distribution across all the labels. We only applied the $\epsilon$ perturbation to the first two dimensions; range and $\theta$. This is based on information from the domain experts, that inputs can be considered truly adversarial only if they differ in those two dimensions.

We took a set of five distinct inputs for each label (for a total of 25 inputs), relaxed the first two dimensions by 5% ($\epsilon$ = 0.05) in both directions and generated 10,000 inputs from within that region. With this input set, we applied our analysis to determine the distribution of probabilities. The results of input rejection can be seen in Table IV. The high rejection rate indicates that most of the inputs followed the same paths through the code and our technique is able to effectively prune the search space, processing only the inputs that lead to the discovery of new paths (average 885 unique paths). This sampling and rejection phase took about 30 minutes per input[1].

We then computed the volume represented by each pairing of path and decision condition as per our amplification procedure. We used Realpaver on a AMD EPYC 7401P 24-Core Processor, limited to use up to 46 threads; memory consumption never exceeded 4Gb in our experiments. The runs took an average of 2.68 minutes per label, bringing the total execution time of the technique to around 33 minutes.

Table V gives the overall results on the 25 base inputs. For each input, we display the probabilities of classification to all five labels within the 5% perturbation region. We display the probabilities obtained based on both the inner boxes (under-approximation) and the inner+outer boxes (over-approximation) in the "P" columns under each label. The confidence value is calculated as the sum of the probabilities obtained from the inner-boxes for each of the labels. This corresponds to the total amount of the covered input space covered (see last column "Confidence P" in the table). We observe that for inputs belonging to Label 0, for 4 out of 5 inputs, the probability for Label 0 is very high (average 99.2% from the inner-boxes) and the probabilities are 0 for the other labels. This indicates that the network is highly robust around these inputs with high confidence. For one of the inputs, there is a distribution of the probabilities where there are approximately 21% and 22% chances of finding adversaries mis-classified to Labels 1 and 2, respectively. This gives an indication of the vulnerability of the network. We also obtain a quantification of this sensitivity and an identification of the labels that the network may mis-classify to. Thus our approach is able to provide a more detailed analysis, specifically a quantification of the robustness and sensitivity of the network with high confidence as compared to existing approaches which focus on proving the robustness of the network but cannot provide any more information when such a guarantee cannot be obtained.

We also performed pure statistical analysis based on the

[1]The rejection sampling process can be parallelized across any number of inputs, reducing the execution time to the order of tens of minutes.

TABLE V: ACAS Xu results.

(10,000 Inputs; P: min, max percentages / SP: percentage ± confidence interval, ± 0 when only one label has been sampled)

| Label | Input # | Label 0 | | Label 1 | | Label 2 | | Label 3 | | Label 4 | | Confidence |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P | SP | P | SP | P | SP | P | SP | P | SP | P |
| 0 | 0 | 98.58, 100 | 100 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 98.58 |
| 0 | 1 | 99.13, 100 | 100 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 99.13 |
| 0 | 2 | 52.94, 63.70 | 53.27 ± 0.98 | 21.64, 22.05 | 22.5 ± 0.82 | 22.78, 23.29 | 23.5 ± 0.83 | 0, 0 | 0 ± 0 | 0.69, 0.70 | 0.73 ± 0.17 | 98.04 |
| 0 | 3 | 99.77, 100 | 100 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 99.77 |
| 0 | 4 | 99.46, 100 | 100 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 0, 0 | 0 ± 0 | 99.46 |
| 1 | 0 | 24.94, 25.26 | 25.15 ± 0.85 | 23.68, 24.06 | 24.97 ± 0.85 | 26.70, 27.09 | 27.92 ± 0.88 | 5.43, 5.60 | 6.55 ± 0.48 | 13.14, 13.54 | 15.41 ± 0.71 | 93.90 |
| 1 | 1 | 0, 0 | 0 ± 0 | 66.01, 67.39 | 68.31 ± 0.91 | 3.21, 3.28 | 3.24 ± 0.35 | 22.61, 23.58 | 24.69 ± 0.85 | 3.24, 3.41 | 3.76 ± 0.37 | 95.07 |
| 1 | 2 | 23.15, 23.62 | 22.97 ± 0.82 | 66.33, 68.81 | 69.09 ± 0.91 | 0.19, 0.23 | 0.33 ± 0.11 | 3.57, 4.10 | 4.31 ± 0.40 | 2.70, 3.05 | 3.3 ± 0.35 | 95.94 |
| 1 | 3 | 34.46, 34.97 | 34.9 ± 0.93 | 36.54, 37.53 | 37.9 ± 0.95 | 0, 0 | 0 ± 0 | 23.84, 25.02 | 27.2 ± 0.87 | *1.03e-03, 1.04e-03* | *0 ± 0* | 94.84 |
| 1 | 4 | 50.63, 51.62 | 51.16 ± 0.98 | 14.31, 14.77 | 15.18 ± 0.70 | 23.40, 24.39 | 24.12 ± 0.84 | 8.99, 9.14 | 9.54 ± 0.58 | 0, 0 | 0 ± 0 | 97.32 |
| 2 | 0 | 9.17, 9.28 | 9.18 ± 0.57 | 0, 0 | 0 ± 0 | 42.31, 42.89 | 43.67 ± 0.97 | 0, 0 | 0 ± 0 | 41.80, 42.68 | 47.15 ± 0.98 | 93.28 |
| 2 | 1 | 14.52, 14.67 | 14.5 ± 0.69 | 0, 0 | 0 ± 0 | 39.39, 39.87 | 40.36 ± 0.96 | 0, 0 | 0 ± 0 | 40.66, 41.46 | 45.14 ± 0.98 | 94.57 |
| 2 | 2 | 21.71, 21.95 | 21 ± 0.80 | 0, 0 | 0 ± 0 | 67.28, 68.08 | 70.15 ± 0.90 | 0, 0 | 0 ± 0 | 8.99, 9.07 | 8.85 ± 0.56 | 97.97 |
| 2 | 3 | 4.83, 4.88 | 4.6 ± 0.41 | 0, 0 | 0 ± 0 | 55.38, 56.75 | 57.97 ± 0.97 | 0, 0 | 0 ± 0 | 34.92, 36.57 | 37.43 ± 0.95 | 95.13 |
| 2 | 4 | 41.05, 41.77 | 41.55 ± 0.97 | 0, 0 | 0 ± 0 | 36.17, 37.43 | 39.08 ± 0.96 | 0, 0 | 0 ± 0 | 18.23, 19.46 | 19.37 ± 0.77 | 95.48 |
| 3 | 0 | 0, 0 | 0 ± 0 | 74.73, 77.56 | 77.65 ± 0.82 | 0.20, 0.25 | 0.23 ± 0.09 | 17.38, 19.16 | 19.6 ± 0.78 | 2.07, 2.35 | 2.52 ± 0.31 | 94.39 |
| 3 | 1 | 2.13, 2.15 | 2.15 ± 0.28 | 31.18, 31.66 | 32.4 ± 0.92 | 0, 0 | 0 ± 0 | 57.07, 57.85 | 62.56 ± 0.95 | 2.17, 2.20 | 2.89 ± 0.33 | 92.55 |
| 3 | 2 | 0, 0 | 0 ± 0 | 35.90, 37.05 | 37.82 ± 0.95 | 0.21, 0.23 | 0.31 ± 0.11 | 51.90, 53.14 | 53.84 ± 0.98 | 7.41, 7.78 | 8.03 ± 0.53 | 95.42 |
| 3 | 3 | 32.42, 33.02 | 33.22 ± 0.92 | 5.56, 5.71 | 5.33 ± 0.44 | 0, 0 | 0 ± 0 | 56.37, 57.66 | 60.42 ± 0.96 | 0.79, 0.84 | 1.03 ± 0.20 | 95.13 |
| 3 | 4 | 8.30, 8.41 | 8.6 ± 0.55 | 24.35, 24.92 | 25.25 ± 0.85 | 0, 0 | 0 ± 0 | 51.37, 52.52 | 53.6 ± 0.98 | 11.36, 11.80 | 12.55 ± 0.65 | 95.38 |
| 4 | 0 | 0, 0 | 0 ± 0 | 28.93, 29.33 | 29.76 ± 0.90 | 34.10, 34.55 | 34.3 ± 0.93 | 10.64, 10.91 | 11.39 ± 0.62 | 23.24, 23.69 | 34.55 ± 0.84 | 96.90 |
| 4 | 1 | 0.99, 1.00 | 1.03 ± 0.20 | 0, 0 | 0 ± 0 | 38.59, 39.00 | 39.68 ± 0.96 | 8.16, 8.30 | 9.43 ± 0.57 | 48.18, 48.85 | 49.86 ± 0.98 | 95.92 |
| 4 | 2 | *3.46e-7, 3.51e-7* | *0 ± 0* | 0, 0 | 0 ± 0 | 20.17, 20.42 | 21.78 ± 0.81 | 18.55, 19.07 | 21.06 ± 0.80 | 52.23, 53.02 | 57.16 ± 0.97 | 90.95 |
| 4 | 3 | 5.77e-2, 5.81e-2 | 0.03 ± 0.03 | 0, 0 | 0 ± 0 | 19.62, 19.87 | 20.86 ± 0.80 | 4.41, 4.48 | 5.05 ± 0.43 | 68.76, 69.73 | 74.06 ± 0.86 | 92.84 |
| 4 | 4 | 24.23, 24.67 | 24.5 ± 0.84 | 0.83, 0.89 | 1.04 ± 0.20 | 0.16, 0.16 | 17.14 ± 0.74 | 0, 0 | 0 ± 0 | 53.64, 55.58 | 57.23 ± 0.97 | 94.58 |

same 10,000 input set to determine the probabilities for the different labels for 25 inputs belonging to each of the five labels. This analysis took times on the order of hundreds of seconds before reaching a high rejection rate (i.e., sampling few new paths). Table V shows the results in the "SP" columns. Each probability is shown in terms of its statistical confidence interval. The probabilities for all inputs overall match with those obtained from our analysis; however, since the probabilities we obtain are based on precise volume calculation, they are more dependable than pure statistical guarantees. As the table shows, the probabilities can vary within the interval.

Notably, our analysis has the ability to find non-zero probabilities for labels (indicating potential adversaries) for which the statistical analysis counterpart finds a probability of zeros (highlighted in italics in Table V). While these probabilities tend to be small, false robustness guarantees can have serious implications for safety-critical applications. The identification of regions where the network is vulnerable and where it is guaranteed to be robust could be used to assess the risk of adversarial behaviors within a region or for a label, and possibly to refine the training set with more inputs from regions/labels with higher misclassification probability.

## VII. Related Work

The closest to our work is the quantitative verification method for neural networks from [29]. That work focuses on binarized neural networks (BNNs) which can be translated efficiently into Conjunctive Normal Form (CNF) formulas. SpaceScanner can analyze a broader class of networks, with piecewise linear activations for exact quantification and more general classes for approximate quantification.

We discussed different model counting and volume computation techniques in Section IV. We propose a novel use of interval analysis (using RealPaver) to compute under- and over-approximations of the exact volume for a set of constraints. Our method for volume estimation is not limited to linear constraints or to neural networks, and can be generalized to other applications that require volume computations.

FairSquare [9] is a general probabilistic framework for fairness and bias analysis designed to be applicable to a wide range of programs. It implements a custom symbolic-volume-computation algorithm that exploits SMT solvers to compute weighted volumes over formulas in real arithmetic. However, experimental evidence indicates that FairSquare has limited scalability. Our technique is specialized to neural networks and leverages interval solving for volume estimation, making it applicable to much larger networks.

VeriFair is a recently proposed approach which performs probabilistic verification of fairness properties by leveraging sampling and concentration inequalities [30]. VeriFair scales better than FairSquare but it is a pure statistical approach that treats the neural network as a black-box. Also related is PROVEN [10], a framework to probabilistically verify neural networks with statistical guarantees. It uses existing robustness verification tools to derive probabilistic certificates wrt robustness of neural networks with lp balls surrounding an input. The approach from [11] has a similar goal but uses a combination of abstract interpretation and importance sampling to estimate the local robustness of a neural network. We focus on providing rigorous guarantees instead of statistical confidence estimates.

## VIII. Conclusion

We presented a probabilistic analysis framework for neural networks, which introduces a variation of concolic execution tailored to the topology of the network and uses solution space quantification over collected constraints, corresponding to neuron activations.Our evaluation shows that this framework provides useful results for fairness and robustness analysis, in the context of medium sized networks. In future work we plan to extend our framework to support more diverse activation functions and perform a broader experimental campaign to further assess its potential.

## References


[1] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV*, 2017.

[2] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *CAV*, 2017.

[3] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018, pp. 3–18. [Online]. Available: https://doi.org/10.1109/SP.2018.00058

[4] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018.

[5] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings*, 2018.

[6] M. Fischetti and J. Jo, "Deep neural networks as 0-1 mixed integer linear programs: A feasibility study," *CoRR*, vol. abs/1712.06174, 2017.

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013, technical Report. http://arxiv.org/abs/1312.6199.

[8] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, 2017, pp. 97–117.

[9] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori, "Fairsquare: probabilistic verification of program fairness," *PACMPL*, vol. 1, no. OOPSLA, pp. 80:1–80:30, 2017. [Online]. Available: https://doi.org/10.1145/3133904

[10] L. Weng, P. Chen, L. M. Nguyen, M. S. Squillante, A. Boopathy, I. V. Oseledets, and L. Daniel, "PROVEN: verifying robustness of neural networks with a probabilistic approach," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 6727–6736. [Online]. Available: http://proceedings.mlr.press/v97/weng19a.html

[11] R. Mangal, A. V. Nori, and A. Orso, "Robustness of neural networks: a probabilistic and practical approach," in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019*, 2019, pp. 93–96. [Online]. Available: https://doi.org/10.1109/ICSE-NIER.2019.00032

[12] L. Granvilliers and F. Benhamou, "Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques," *ACM Transactions on Mathematical Software (TOMS)*, vol. 32, no. 1, pp. 138–156, 2006.

[13] B. Büeler, A. Enge, and K. Fukuda, *Exact Volume Computation for Polytopes: A Practical Study*. Basel: Birkhäuser Basel, 2000, pp. 131–154. [Online]. Available: https://doi.org/10.1007/978-3-0348-8438-9_6

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," *ACM Comput. Surv.*, vol. 51, no. 3, May 2018. [Online]. Available: https://doi.org/10.1145/3182657

[16] D. Gopinath, K. Wang, M. Zhang, C. S. Pasareanu, and S. Khurshid, "Symbolic execution for deep neural networks," *CoRR*, vol. abs/1807.10439, 2018. [Online]. Available: http://arxiv.org/abs/1807.10439

[17] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 109–119. [Online]. Available: https://doi.org/10.1145/3238147.3238172

[18] A. Agarwal, P. Lohia, S. Nagar, K. Dey, and D. Saha, "Automated test generation to detect individual discrimination in ai models," 2018.

[19] M. Borges, A. Filieri, M. d'Amorim, C. S. Păsăreanu, and W. Visser, "Compositional solution space quantification for probabilistic software analysis," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '14. New York, NY, USA: ACM, 2014, pp. 123–132. [Online]. Available: http://doi.acm.org/10.1145/2594291.2594329

[20] P. Gritzmann and V. Klee, "On the complexity of some basic problems in computational convexity: I. containment problems," *Discrete Mathematics*, vol. 136, no. 1, pp. 129 – 174, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0012365X9400111U

[21] I. Z. Emiris and V. Fisikopoulos, "Practical polytope volume approximation," *ACM Trans. Math. Softw.*, vol. 44, no. 4, pp. 38:1–38:21, Jun. 2018. [Online]. Available: http://doi.acm.org/10.1145/3194656

[22] J. B. Lasserre, "An analytical expression and an algorithm for the volume of a convex polyhedron inrn," *Journal of Optimization Theory and Applications*, vol. 39, no. 3, pp. 363–377, Mar 1983. [Online]. Available: https://doi.org/10.1007/BF00934543

[23] S. Sharma, S. Roy, M. Soos, and K. S. Meel, "Ganak: A scalable probabilistic exact model counter," in *Proceedings of the Twenty-Eighth International*

*Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 1169–1176. [Online]. Available: https://doi.org/10.24963/ijcai.2019/163

[24] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter," in *Principles and Practice of Constraint Programming*, C. Schulte, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 200–216.

[25] W. Hoeffding, *Probability Inequalities for sums of Bounded Random Variables*. Springer, 1994, pp. 409–426.

[26] I. Araya, G. Trombettoni, B. Neveu, and G. Chabert, "Upper bounding in inner regions for global optimization under inequality constraints," *Journal of Global Optimization*, vol. 60, no. 2, pp. 145–164, Oct 2014. [Online]. Available: https://doi.org/10.1007/s10898-014-0145-7

[27] K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *Proc. 35th Digital Avionics System Conf. (DASC)*, 2016, pp. 1–10.

[28] D. Gopinath, G. Katz, C. S. Pasareanu, and C. Barrett, "Deepsafe: A data-driven approach for checking adversarial robustness in neural networks," 2017, https://arxiv.org/abs/1710.00486.

[29] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena, "Quantitative verification of neural networks and its security applications," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, 2019, pp. 1249–1264.

[30] O. Bastani, X. Zhang, and A. Solar-Lezama, "Probabilistic verification of fairness properties via concentration," *PACMPL*, vol. 3, no. OOPSLA, pp. 118:1–118:27, 2019. [Online]. Available: https://doi.org/10.1145/3360544