Hyperplane Input Space Cuts for Neural Network Verification

Jonathan Hjort
Department of Computer and Information Science
Linköping University
Linköping, Sweden
jonathan.hjort@liu.se

Ahmed Rezine
Department of Computer and Information Science
Linköping University
Linköping, Sweden
ahmed.rezine@liu.se

***Abstract*—To achieve tighter bounds on output neurons, previous input space Branch-and-Bound based approaches have used heuristics that determine which input dimension the problem will be split on. In this paper, we present a new technique for splitting the input space with respect to arbitrary input space hyperplanes. For ReLU Neural Networks, this allows us to guide the splitting to obtain problems with more linear neurons as tighter bounds can be obtained with off-the-shelf symbolic interval propagation techniques. Our proposed approach makes use of symbolic bounds for ambiguous ReLU neurons to construct a new base for the input space, allowing us to force a neuron to be linear in the resulting sub-problems. Effectively, this requires us to split the input space with respect to arbitrary hyperplanes, not only parallel to the axes of the input dimensions. This, combined with remembering the bounds of neurons from previous analyses, allows us to show that properties hold on neural networks having to split the problem up to two order of magnitude fewer times than traditional input space Branch-and-Bound based tools.**

***Index Terms*—Verification, Neural Networks, Safety, Abstraction Refinement**

I. Introduction

The pervasive adoption of Neural Networks (NNs), including for safety critical ones, recently resulted in a surge of interest for their safety verification and in an explosion of works on how to check their correctness. We propose an integration of existing symbolic interval propagation techniques with arbitrary hyperplane cuts of the input space. Our approach is efficient because it leverages on established symbolic interval propagation approaches without the need to encode the whole network as a Mixed Integer Linear Programming (MILP), or a relaxation of it as a Linear Programming (LP) problem. Unlike existing bound propagation approaches, our technique can cut the input space using arbitrary hyperplanes, allowing it to refine the analysis both by considering smaller input spaces and by maintaining relations between values of different input dimensions.

Broadly, current NN verification techniques can be categorized into exact methods using MILP or augmented Satisfiability Modulo Theory (SMT) formulations [4], [5], [10], or inexact abstraction based methods that trade exactness for scalability. Abstraction based methods come in different flavors. Some approaches propagate, one layer at a time, symbolic intervals or shapes to approximate reachable values [9], [12]. Other approaches use bounds on each neuron to generate linear relaxations and use them to encode the whole network leveraging on off-the-shelf LP tools [1], [11], while other techniques perform non-trivial optimizations to repeatedly tighten bounds associated to the neurons [2], [15]. Branch-and-Bound (BaB) approaches can be used to refine such approximations. Often, branches are obtained by generating subproblems where behaviors of chosen neurons are further constrained [2], [11], [13]. For networks with ReLU neurons, these constraints will typically amount to enumerating the possible states (i.e., active, inactive, ambiguous) of some targeted neurons. Such BaB approaches are quite complex as they have to encode the whole network in LP formulations (e.g., [11]) or require complex and involved optimization algorithms to propagate internal bounds (e.g., [2], [13]). On the other hand, conceptually simpler BaB solutions (henceforth, "plain" BaB) such as [3], [8], [12], partition the input space one dimension at a time to refine the analysis. Such partitioning cannot be influenced by constraints on the neurons, for instance, one cannot design a plain input space partitioning that will guarantee a given neuron state. This makes it difficult to generate input partitions that guarantee given neurons will be in chosen states. We consider ReLU networks to simplify the presentation. Given an input range, each neuron can be active, inactive, or both (henceforth ambiguous). Verification with resepct to ranges where no neuron is ambiguous amount to simple linear computations. The difficulty stems from the non linear behavior of ambiguous neurons.

We propose in this work a simple and intuitive approach to partition the input space in order to perform more informed splits that will result in branches on which symbolic interval propagation analysis will be more exact because more neurons behave linearly. Crucially, our approach can be parameterized by any symbolic interval propagation routine that is capable of giving linear symbolic upper and lower bounds (symbolic in the network input) to each neuron. Several such efficient symbolic interval propagation approaches already exist [9], [11], [12]. Our idea is to use these linear approximations to split the input space in a way that guarantees constraints on chosen neurons. Effectively, we are allowing the split to account for the desired effect. Since the introduced transformation is linear, we can run again the bound propagation on the tightest bounds so far, while (i) enforcing the desired neuron behavior, hence obtaining a more precise analysis, and (ii) collecting better

Identify applicable funding agency here. If none, delete this.

bounds for future runs. Intuitively, we relate several input space dimensions, something plain splitting is incapable of.

To isolate the effect of our approach, we did conduct thorough comparisons with an established representative for plain input space splitting [12] on the known ACAS Xu benchmark [7]. The objective of the comparison is to assess whether splitting the input space with respect to hyperplane cuts can help verification. We consistently observed that our approach results in (up to two orders of magnitude) less numbers of splits and, often, in smaller required depths.

To summarize, our contributions are:

- We introduce a simple and efficient approach to split the input space according to arbitrary hyperplanes.
- Our approach integrates symbolic interval propagators in order to (i) supply the hyperplanes and to (ii) generate tighter bounds for future rounds
- We conducted extensive experiments to compare our hyperplane input splitting approach against an established plain splitting approach.

The next Section introduces notations and Section III describes an illustrating example. Section IV depicts the splitting procedure and involved steps. We describe our experiments and results in Section V and conclude in Section VI.

II. Preliminaries

A Neural Network (NN) $\mathtt{net}$ comes with $1 + |\mathtt{net}|$ layers $L^0, L^1, \dots L^{|\mathtt{net}|}$. Each layer L^i involves neurons $\mathtt{n}^i_1, \dots, \mathtt{n}^i_{|L^i|}$. Only internal layers have ReLU neurons. We abuse notation and use $\mathtt{n}^i_j$ (resp. $\widehat{\mathtt{n}}^i_j$) for the neuron's value after (resp. before) ReLU application if any, otherwise we let $\mathtt{n}^i_j = \widehat{\mathtt{n}}^i_j$. We use $\mathtt{n}^i$ (resp. $\widehat{\mathtt{n}}^i$) for all $\mathtt{n}^i_j$ ($\widehat{\mathtt{n}}^i_j$) values. Internal layers satisfy $\mathtt{n}^i = \mathtt{max}(0, \widehat{\mathtt{n}}^i)$ while input and output layers have $\mathtt{n}^i = \widehat{\mathtt{n}}^i$. For each neuron $\mathtt{n}^i_j$, we use $\mathbf{lo}^i_j$ and $\mathbf{up}^i_j$ (resp. $\widehat{\mathbf{lo}}^i_j$ and $\widehat{\mathbf{up}}^i_j$) to mean some concrete lower and upper bounds on $\mathtt{n}^i_j$ (resp. $\widehat{\mathtt{n}}^i_j$). We also use $\mathbf{sl}^i_j$ and $\mathbf{su}^i_j$ (resp. $\widehat{\mathbf{sl}}^i_j$ and $\widehat{\mathbf{su}}^i_j$) to mean some symbolic lower and upper bounds on $\mathtt{n}^i_j$ (resp. $\widehat{\mathtt{n}}^i_j$). Symbolic bounds are linear expressions in the inputs $\mathtt{n}^0$ of the network. We use notations such as $\mathbf{lo}^i$ and $\mathbf{su}^i$ to mean corresponding values for layer L^i. Safety properties for a neural network can be formulated using pairs of concrete bounds: A pair $(\mathbf{lo}^0, \mathbf{up}^0)$ delimiting considered inputs, and a pair $(\mathbf{lo}^{bad}, \mathbf{up}^{bad})$ delimiting bad outputs. The network is safe if no input data point p satisfying $(\mathbf{lo}^0, \mathbf{up}^0)$ can result in an output $\mathtt{net}(p)$ satisfying $(\mathbf{lo}^{bad}, \mathbf{up}^{bad})$.

III. Example

The NN in Figure 1 computes the maximum value of two inputs with the same range (i.e., $x, y \in [0, 1]$). The value of $\mathtt{n}^1_1$ always coincides with the value of non-negative input x while $\mathtt{n}^1_2$ coincides with $y - x$ if $y \geq x$ and with 0 otherwise. The network therefore outputs $\mathtt{max}(x, y)$ regardless of the values of $(x, y) \in [0, 1]^2$. Clearly, the output $\mathtt{n}^2_1$ is within $[0, 1]$.

Simple interval propagation associates $[0, 1]$ to $\widehat{\mathtt{n}}^1_1$, and $[-1, 1]$ to $\mathtt{n}^1_2$. After ReLU application, it would efficiently deduce $\mathtt{n}^2_1 \in [0, 2]$ but fails to show $\mathtt{n}^2_1 \in [0, 1]$. Symbolic interval analysis, as used by [12] and [11], would associate each $\widehat{\mathtt{n}}^i_j$

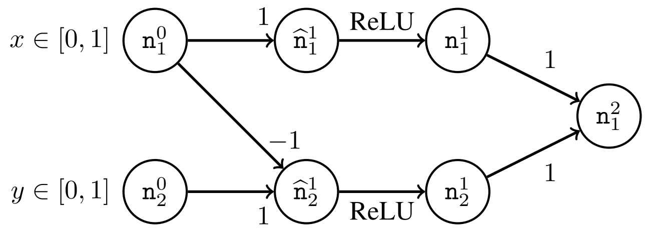

Fig. 1: Simple Neural Network computing $\mathtt{max}(x, y)$.

(resp. $\mathtt{n}^i_j$) to a symbolic interval $[\widehat{\mathbf{sl}}^i_j, \widehat{\mathbf{su}}^i_j]$ (resp. $[\mathbf{sl}^i_j, \mathbf{su}^i_j]$). For $\mathtt{n}^1_1$, such analysis gives $\widehat{\mathbf{sl}}^1_1 = \widehat{\mathbf{su}}^1_1 = x$ before ReLU application and (because $x \geq 0$) $\mathbf{sl}^1_1 = \mathbf{su}^1_1 = x$ after ReLU. For $\mathtt{n}^1_2$, we get $\widehat{\mathbf{sl}}^1_2 = \widehat{\mathbf{su}}^1_2 = y - x$ before ReLU application and (because $-1 \leq y - x \leq 1$) $\mathbf{sl}^1_2 = \frac{1}{2}(y - x)$ and $\mathbf{su}^1_1 = \frac{1}{2}(y - x + 1)$ after ReLU. Hence, we get $\frac{1}{2}(y + x) \leq \mathtt{n}^2_1 \leq \frac{1}{2}(y + x + 1)$, which, given the bounds on x and y, shows that $\mathtt{n}^2_1 \in \left[0, \frac{3}{2}\right]$.

To achieve tighter bounds on $\mathtt{n}^2_1$ previous plain input space splitting tools use heuristics to choose one input dimension, split it in half and analyze the resulting subproblems. However, such methods have difficulties terminating at all on NNs as the one given in Fig. 1. Consider the n-th split of dimension x such that $x \in \left[1 - \frac{1}{2^n}, 1\right]$ and let $y \in [a, 1]$ with $0 \leq a < 1$. Using symbolic interval analysis, we have $x \leq \widehat{\mathtt{n}}^1_1 \leq x$ and $x \leq \mathtt{n}^1_1 \leq x$. For $\widehat{\mathtt{n}}^1_2$, we find that $y - x \leq \widehat{\mathtt{n}}^1_2 \leq y - x$. Here, $\mathtt{n}^1_2$ is still considered an ambiguous neuron since the lower and upper bounds of $y - x$ are $a - 1 < 0$ and $\frac{1}{2^n}$ given the new bounds of x. Therefore, [12] and [11] would over-approximate $\mathtt{n}^1_2$ such that $\frac{(y-x)}{(1-a)2^n+1} \leq \mathtt{n}^1_2 \leq \frac{y-x+(1-a)}{(1-a)2^n+1}$. At the output neuron $\mathtt{n}^2_1$, because of the symbolic bounds of $\mathtt{n}^1_1$ and $\mathtt{n}^1_2$, we now see that its upper bound is $\frac{(1-a)2^n x+y+1}{(1-a)2^n+1}$. This, given the bounds of x and y, is bounded by $\frac{(1-a)2^n+2}{(1-a)2^n+1} > 1$. This shows two things; we still cannot conclude that $\mathtt{n}^2_1 \in [0, 1]$, and $\mathtt{n}^1_2$ is still an ambiguous neuron even after n splits.

We want to use the intuition that tighter bounds are obtained by symbolic propagation methods when more activations are linear. Indeed, enumerating all activation states would result in linear NN. Therefore, when we split the input space, we want to ensure that a neuron that was ambiguous becomes linear in the resulting subproblems. In our example we found that $\widehat{\mathtt{n}}^1_2$ was ambiguous through symbolic interval analysis. We also found that it is bounded by the linear equation $y - x$ such that $y - x \leq \widehat{\mathtt{n}}^1_2 \leq y - x$. Therefore, when we split the problem, i.e., the input space, we want to split it such that the subproblems satisfy, in one case, $y - x \geq 0$ (making the neuron active), and in another case $y - x \leq 0$ (making the neuron inactive). Instead of adding such conditions to expensive LP encodings of the whole network with limited possibilities to strengthen internal bounds for future branches (e.g., as in [11]), or of using complex bound optimizations that do not relate to the input space of the problem (as in [2], [13]), we propose to use the intuitive idea of adopting a change of basis in the input space. For the first dimension x' in the new basis we use the symbolic bounds on $\widehat{\mathtt{n}}^1_2$, and let x' be proportional to $(y - x)$.

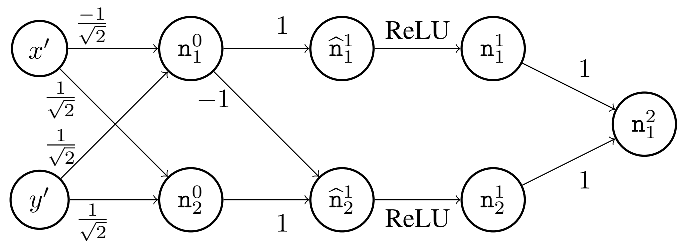

Fig. 2: NN of Fig. 1 appended with a linear transformation to express (in)activation of $\widehat{\mathtt{n}}_2^1$.

For presentation simplicity we use an orthonormal basis and let y' be proportional to $(y+x)$ obtaining $B = \frac{1}{\sqrt{2}}\begin{pmatrix}-1 & 1\\ 1 & 1\end{pmatrix}$. The new basis can be viewed as an extra linear layer before the original input in the NN. To find the bounds for the new input variables x' and y' we invoke an off-the-shelf LP-solver on a problem encoding the relation between original and new input bounds together with the hyperplanes collected in the current branch so far (none currently). Such LP problems only involve input variables and branching constraints on them (unlike encodings of entire network relaxations in addition to the collected branching constraints). We get $y' \in \left[0, \sqrt{2}\right]$, and $x' \in \left[0, \frac{\sqrt{2}}{2}\right]$ and $x' \in \left[\frac{-\sqrt{2}}{2}, 0\right]$ for the active and inactive subproblems, respectively. We are effectively analyzing the NN in Fig. 2 in the active and inactive subproblems.

Now, consider the active subproblem, i.e., $x' \in \left[0, \frac{\sqrt{2}}{2}\right]$ and $y' \in \left[0, \sqrt{2}\right]$. For $\widehat{\mathtt{n}}_1^1$, we find through symbolic interval analysis that its' upper and lower bounds coincide with $\frac{y'-x'}{\sqrt{2}}$. Our previous analysis, which still holds in the subproblems it produced, showed that $\widehat{\mathtt{n}}_1^1$ was linear (active). Therefore, it can still safely be considered active. We then have that $\mathtt{n}_1^1$ has the same bounds as $\mathtt{n}_1^1$, i.e., $\frac{y'-x'}{\sqrt{2}}$. For $\widehat{\mathtt{n}}_2^1$, we find that it is bounded by $\frac{2x'}{\sqrt{2}}$. In our current analysis $\widehat{\mathtt{n}}_2^1$ is active because we chose a basis and a constraint that allow us to express $\widehat{\mathbf{sl}}_2^1 = y - x \geq 0$. Observe that the new basis and the associated lower and upper bounds may give us the possibility to obtain tighter bounds (and even linearity) for more neurons. We now have that $\mathtt{n}_2^1$ is bounded by $\frac{2x'}{\sqrt{2}}$. Finally, at $\mathtt{n}_1^2$ we get, due to the bounds on $\mathtt{n}_1^1$ and $\mathtt{n}_2^1$, that it is bounded by $\frac{x'+y'}{\sqrt{2}}$. Given the constraints on x' and y', this is sometimes enough to conclude. However, here, we have $\mathtt{n}_1^2 \in \left[0, \frac{3}{2}\right]$ and we still have not concluded! Yet, under the constraints of the current analysis we have tightened the bounds and found that $\widehat{\mathtt{n}}_2^1$ is indeed active. This, combined with $\widehat{\mathtt{n}}_1^1$ being active (from the previous analysis), makes it possible for us to (efficiently) re-run the symbolic interval analysis in the old basis with these newly found facts. We now find that $\mathtt{n}_1^2$ is bounded by only y and we can conclude that $\mathtt{n}_1^2 \in [0, 1]$. The inactive subproblem is solved similarly.

Our ability to both remember the bounds of the neurons from previous analysis, and to force the state of a neuron after we split allowed us to conclude and show $\mathtt{n}_1^2 \in [0, 1]$ while only splitting the problem once, into two subproblems.

IV. Hyperplane Input Space Cuts

We start with a description of the involved steps before we state soundness of the approach.

Procedure "check" recursively strengthens bounds until it shows safety or it returns an adversarial example.

```
Input:
  • $\widehat{\mathbf{lo}}, \widehat{\mathbf{up}}$ initial bounds (prior to ReLU if any)
  • $\Phi$ collected linear inequalities defining half input spaces
  • $\mathbf{B}$ basis for expressing current cut
  • $(\mathbf{lo}^c, \mathbf{up}^c)$ cut bounds expressed in $\mathbf{B}$
Output: safe, adv(p)
1  $(\widehat{\mathbf{lo}'}, \widehat{\mathbf{up}'}), (\widehat{\mathbf{sl}}, \widehat{\mathbf{su}}) :=$ sP $\left((\mathbf{lo}^c, \mathbf{up}^c) \cdot (\widehat{\mathbf{lo}}, \widehat{\mathbf{up}}), \mathbf{B} \cdot \mathtt{net}\right)$
2  if $(\widehat{\mathbf{lo}'}^{|\mathtt{net}|}, \widehat{\mathbf{up}'}^{|\mathtt{net}|}) \sqcap (\mathbf{lo}^{bad}, \mathbf{up}^{bad}) = \bot$ then return safe
3  $p :=$ choosePoint $\left((\widehat{\mathbf{lo}'}^0, \widehat{\mathbf{up}'}^0), \mathbf{B}, (\mathbf{lo}^c, \mathbf{up}^c)\right)$
4  if $\mathbf{lo}^{bad} \leq \mathbf{net}(p) \leq \mathbf{up}^{bad}$ then return adv$(p)$
5  Choose ambiguous $\mathtt{n}_j^i$ with $\widehat{\mathbf{lo}'}_j^i < 0 < \widehat{\mathbf{up}'}_j^i$
6  $\mathbf{cut}^+ :=$ deriveCut $\left((\widehat{\mathbf{lo}'}^0, \widehat{\mathbf{up}'}^0), 0 \leq \widehat{\mathbf{sl}}_j^i, \Phi\right)$
7  if isFeasible $\left(\mathbf{cut}^+\right)$ then
8      $\widehat{\mathbf{lo}''}, \widehat{\mathbf{up}''} := \widehat{\mathbf{lo}'}, \widehat{\mathbf{up}'}$
9      $\widehat{\mathbf{lo}''}_j^i := 0$ /* mark $n_j^i$ active */
10     $\Phi^+ := \Phi \cup \left\{0 \leq \widehat{\mathbf{sl}}_j^i\right\}$
11     $((\mathbf{lo}^+, \mathbf{up}^+), \mathbf{B}^+) :=$ getCut $\left(\mathbf{cut}^+\right)$
12     if check $\left((\widehat{\mathbf{lo}''}, \widehat{\mathbf{up}''}), \Phi^+, \mathbf{B}^+, (\mathbf{lo}^+, \mathbf{up}^+)\right)$ = adv$(p)$ then
13         return adv$(p)$
14 $\mathbf{cut}^- :=$ deriveCut $\left((\widehat{\mathbf{lo}'}^0, \widehat{\mathbf{up}'}^0), \widehat{\mathbf{su}}_j^i \leq 0, \Phi\right)$
15 if isFeasible $\left(\mathbf{cut}^-\right)$ then
16     $\widehat{\mathbf{lo}''}, \widehat{\mathbf{up}''} := \widehat{\mathbf{lo}'}, \widehat{\mathbf{up}'}$
17     $\widehat{\mathbf{lo}''}_j^i := \widehat{\mathbf{up}''}_j^i := 0$ /* mark $n_j^i$ inactive */
18     $\Phi^- := \Phi \cup \left\{\widehat{\mathbf{su}}_j^i \leq 0\right\}$
19     $((\mathbf{lo}^-, \mathbf{up}^-), \mathbf{B}^-) :=$ getCut $\left(\mathbf{cut}^-\right)$
20     if check $\left((\widehat{\mathbf{lo}''}, \widehat{\mathbf{up}''}), \Phi^-, \mathbf{B}^-, (\mathbf{lo}^-, \mathbf{up}^-)\right)$ = adv$(p)$ then
21         return adv$(p)$
22 $\mathbf{cut}^\pm :=$ deriveCut $\left((\widehat{\mathbf{lo}'}^0, \widehat{\mathbf{up}'}^0), \widehat{\mathbf{sl}}_j^i \leq 0 \leq \widehat{\mathbf{su}}_j^i, \Phi\right)$
23 if isFeasible $\left(\mathbf{cut}^\pm\right)$ then
24     $\widehat{\mathbf{lo}''}, \widehat{\mathbf{up}''} := \widehat{\mathbf{lo}'}, \widehat{\mathbf{up}'}$
25     $\Phi^\pm := \Phi \cup \left\{\widehat{\mathbf{sl}}_j^i \leq 0 \leq \widehat{\mathbf{su}}_j^i\right\}$
26     $((\mathbf{lo}^\pm, \mathbf{up}^\pm), \mathbf{B}^\pm) :=$ getCut $\left(\mathbf{cut}^\pm\right)$
27     if check $\left((\widehat{\mathbf{lo}''}, \widehat{\mathbf{up}''}), \Phi^\pm, \mathbf{B}^\pm, (\mathbf{lo}^\pm, \mathbf{up}^\pm)\right)$ = adv$(p)$ then
28         return adv$(p)$
29 return safe
```

Our approach (see Proc. "check") consists of four main steps: 1) symbolic propagation, line 1, which tightens the current bounds $(\mathbf{lo}, \mathbf{up})$ and provides symbolic lower and upper bounds before ReLU application of each neuron; 2) check if resulting bounds are sufficient to return `safe` or to find an adversarial example, lines 2-4; 3) pick an ambiguous neuron, line 5, whose symbolic bounds will be used to determine hyperplanes for input space cuts; and 4) define resulting subproblems, lines 6-28, which consist of creating a new basis for the input space, finding corresponding input regions, and recursively calling the procedure. Finally, at line 29, our procedure returns `safe` if no subproblem has returned an adversarial example.

1) Symbolic propagation: We write $\mathbf{B} \cdot \mathtt{net}$ to mean the network obtained by prepending a fully connected linear layer (with $|L^0|$ inputs) $\mathbf{B}$ before layer L^0, and write $(\mathbf{lo}^c, \mathbf{up}^c) \cdot$

Procedure symbolicReLU tighten and generates concrete and symbolic intervals.

```
Input:
  • $(\widehat{\mathbf{lo}}^i_j, \widehat{\mathbf{up}}^i_j)$ assumed concrete bounds at $\widehat{\mathtt{n}}^i_j$
  • $(\widehat{\mathbf{sl}}^i_j, \widehat{\mathbf{su}}^i_j)$ assumed symbolic bounds at $\widehat{\mathtt{n}}^i_j$
  • $(\mathbf{lo}^c, \mathbf{up}^c)$ concrete bounds for symbolic inputs
Output:
  • $(\widehat{\mathbf{lo}'}^i_j, \widehat{\mathbf{up}'}^i_j)$ tightened concrete bounds at $\widehat{\mathtt{n}}^i_j$
  • $(\mathbf{sl}^i_j, \mathbf{su}^i_j)$ symbolic bounds at $\mathtt{n}^i_j$ (i.e., after ReLU)
1  /* Concrete bounds for symbolic upper bound $\widehat{\mathbf{su}}^i_j$ */
   $\widehat{\mathbf{lsu}}^i_j, \widehat{\mathbf{usu}}^i_j := \mathtt{min}_{(\mathbf{lo}^c,\mathbf{up}^c)}(\widehat{\mathbf{su}}^i_j), \mathtt{max}_{(\mathbf{lo}^c,\mathbf{up}^c)}(\widehat{\mathbf{su}}^i_j)$
2  /* Concrete bounds for symbolic lower bound $\widehat{\mathbf{sl}}^i_j$ */
   $\widehat{\mathbf{lsl}}^i_j, \widehat{\mathbf{usl}}^i_j := \mathtt{min}_{(\mathbf{lo}^c,\mathbf{up}^c)}(\widehat{\mathbf{sl}}^i_j), \mathtt{max}_{(\mathbf{lo}^c,\mathbf{up}^c)}(\widehat{\mathbf{sl}}^i_j)$
3  /* Tighten concrete bounds of $\widehat{\mathtt{n}}^i_j$ accordingly */
   $\widehat{\mathbf{lo}'}^i_j, \widehat{\mathbf{up}'}^i_j := \mathtt{max}(\widehat{\mathbf{lo}}^i_j, \widehat{\mathbf{lsl}}^i_j), \mathtt{min}(\widehat{\mathbf{up}}^i_j, \widehat{\mathbf{usu}}^i_j)$
4  if $\widehat{\mathbf{up}'}^i_j \leq 0$ then return $(0,0), (0,0)$ /* inactive */
5  if $\widehat{\mathbf{lo}'}^i_j \geq 0$ then return $(\widehat{\mathbf{lo}'}^i_j, \widehat{\mathbf{up}'}^i_j), (\widehat{\mathbf{sl}}^i_j, \widehat{\mathbf{su}}^i_j)$ /*active*/
6  if $\widehat{\mathbf{lsu}}^i_j \geq 0$ then $\mathbf{su}^i_j := \widehat{\mathbf{su}}^i_j$
7  else if $\widehat{\mathbf{lsu}}^i_j \geq \widehat{\mathbf{lo}'}^i_j$ then $\mathbf{su}^i_j := \frac{\widehat{\mathbf{up}'}^i_j}{\widehat{\mathbf{up}'}^i_j - \widehat{\mathbf{lsu}}^i_j}\left(\widehat{\mathbf{su}}^i_j - \widehat{\mathbf{lsu}}^i_j\right)$
8  else if $\widehat{\mathbf{lsu}}^i_j \leq \widehat{\mathbf{lo}'}^i_j$ then $\mathbf{su}^i_j := \frac{\widehat{\mathbf{up}'}^i_j}{\widehat{\mathbf{up}'}^i_j - \widehat{\mathbf{lo}'}^i_j}\left(\widehat{\mathbf{su}}^i_j - \widehat{\mathbf{lo}'}^i_j\right)$
9  if $\widehat{\mathbf{usl}}^i_j \leq 0$ then $\mathbf{sl}^i_j := 0$
10 else if $\widehat{\mathbf{usl}}^i_j \geq \widehat{\mathbf{up}'}^i_j$ then $\mathbf{sl}^i_j := \frac{\widehat{\mathbf{up}'}^i_j}{\widehat{\mathbf{up}'}^i_j - \widehat{\mathbf{lo}'}^i_j}\widehat{\mathbf{sl}}^i_j$
11 else if $\widehat{\mathbf{usl}}^i_j \leq \widehat{\mathbf{up}'}^i_j$ then $\mathbf{sl}^i_j := \frac{\widehat{\mathbf{usl}}^i_j}{\widehat{\mathbf{usl}}^i_j - \widehat{\mathbf{lo}'}^i_j}\widehat{\mathbf{sl}}^i_j$
12 return $(\widehat{\mathbf{lo}'}^i_j, \widehat{\mathbf{up}'}^i_j), (\mathbf{sl}^i_j, \mathbf{su}^i_j)$
```

$(\widehat{\mathbf{lo}}, \widehat{\mathbf{up}})$ to account for its concrete bounds (on the new layer and on the neurons before ReLU applications if any). The weights $\mathbf{B}$ correspond to a change of basis. This step has two main tasks: finding linear (in neurons of prepended input layer) over-approximations before ReLU applications (given by symbolic lower $\widehat{\mathbf{sl}}$ and upper $\widehat{\mathbf{su}}$ bounds), and tightening the concrete bounds $(\widehat{\mathbf{lo}}, \widehat{\mathbf{up}})$. We say the call $(\widehat{\mathbf{lo}}, \widehat{\mathbf{up}}), (\widehat{\mathbf{sl}}, \widehat{\mathbf{su}}) := \mathtt{sP}\left((\mathbf{lo}^c, \mathbf{up}^c) \cdot (\widehat{\mathbf{lo}}, \widehat{\mathbf{up}}), \mathbf{B} \cdot \mathtt{net}\right)$ in line 1 is *sound* if running $\mathbf{B} \cdot \mathtt{net}$ from any input x satisfying $(\mathbf{lo}^c, \mathbf{up}^c) \cdot (\widehat{\mathbf{lo}}, \widehat{\mathbf{up}})$ will result in values $\widehat{\mathtt{n}}^i_j$ satisfying $\widehat{\mathbf{lo}'}^i_j \leq \widehat{\mathtt{n}}^i_j \leq \widehat{\mathbf{up}'}^i_j)$ and $\widehat{\mathbf{sl}}^i_j(x) \leq \widehat{\mathtt{n}}^i_j \leq \widehat{\mathbf{su}}^i_j(x))$. We adapt the symbolic propagation of [11] to account for concrete bounds $(\widehat{\mathbf{lo}}, \widehat{\mathbf{up}})$ obtained from runs with other bases. This is depicted in Proc. symbolicReLU. For instance, lines 1-2 use incoming symbolic lower and upper bounds to compute concrete bounds valid on the input region delimited by $(\mathbf{lo}^c, \mathbf{up}^c)$ in $\mathbf{B}$. These are then used to tighten the previously known concrete bounds in line 3. Such bounds could have been established by previous calls with different basis and cuts but corresponding to regions including the current one. Obtained concrete bounds are then used to tighten the symbolic intervals obtained after the ReLU in lines 4-11. Finally, symbolic intervals generated after the ReLUs are linearly combined according to the network weights (and their signs) before reaching the next ReLU application, resulting in a sound symbolic propagation. We always follow a symbolic analysis with a prepended $\mathbf{B}$ by one without as this might tighten the bounds even more as shown in the running example. The returned symbolic bounds used for later steps are expressed in the old basis to avoid duplicating the number of variables.

2) Direct checks: Once we have the tightened bounds on the output neurons $(\widehat{\mathbf{lo}'}^{|\mathtt{net}|}, \widehat{\mathbf{up}'}^{|\mathtt{net}|})$, we check whether it overlaps with $(\mathbf{lo}^{bad}, \mathbf{up}^{bad})$. If it does *not*, then we can return `safe`. If it does, then we choose a point within the bounds of the current subproblem and check if it produces a violation of the property we are checking. There are different viable ways of choosing this point; a simple and efficient one is to choose the middle of the current input bounds.

3) Choosing hyperplanes for cuts: The third step selects a neuron that will be used to determine the hyperplane that cuts the input space. We require the neuron, here $\widehat{\mathtt{n}}^i_j$, is in an ambiguous state, i.e., $\widehat{\mathbf{lo}'}^i_j < 0 < \widehat{\mathbf{up}'}^i_j$. If there are no ambiguous neurons, then the network is linear and it can be cheaply analysed with symbolic interval propagation. As with the previous step, there are different viable ways of selecting the neuron here. One could simply pick the first one encountered in the NN, or even pick one at random. The heuristic we chose works backwards and assigns a score to each neuron based on its bounds and its weights. The hybrid neuron with the highest score is picked to determine the hyperplane of our cut in the input space.

4) Recursive calls: The fourth step creates at most three subproblems. We describe how the first one is created, the remaining two are similar. Given bounds on input neurons, $(\widehat{\mathbf{lo}'}_0, \widehat{\mathbf{up}'}_0)$, and previously collected constraints Φ on the input space, we further constrain the input by adding the linear inequality $0 \leq \widehat{\mathbf{sl}}$ and check if it has a feasible solution, otherwise, it means $\widehat{\mathbf{sl}}$ cannot be forced to be positive in the considered region. If feasible, we find a new basis $\mathbf{B}^+$, and corresponding bounds $(\mathbf{lo}^+, \mathbf{up}^+)$. The idea is to have one new dimension x' that is proportional to $\widehat{\mathbf{sl}}^i_j$ to express $x' \geq 0$. We have to use an LP-solver to check whether a cut is feasible and to find new input bounds. However, since we are constraining and encoding input space constraints, the size of generated LP queries is linear in the size of Φ and in $|L^0|$, which is typically much smaller than the size of the networks. Recall $\widehat{\mathbf{sl}}$ is a lower symbolic bound for $\widehat{\mathtt{n}}^i_j$, so the neuron is active in the subproblem. In addition, we hope the new basis and bounds will allow us to exhibit tighter bounds and more linearity for other neurons. A cut $((\mathbf{lo}^+, \mathbf{up}^+), \mathbf{B}^+)$ obtained with $\mathtt{deriveCut}\left((\widehat{\mathbf{lo}'}_0, \widehat{\mathbf{up}'}_0), 0 \leq \widehat{\mathbf{sl}}^i_j, \Phi\right)$ is sound if the image of $(\mathbf{lo}^+, \mathbf{up}^+)$ by $\mathbf{B}^c$ includes all input points satisfying $(\mathbf{lo}'_0, \mathbf{up}'_0)$ and $0 \leq \mathbf{sl}'^i_j$ and Φ. This is easily obtained using an LP solver to obtain bounds.

The verification procedure is *sound* if it never returns safe when the property does not hold.

Theorem 1: The procedure depicted in Proc. "check" is sound if symbolic propagations and cut derivations are sound.

Observe that returned adversarial examples are genuine by construction. Termination however is not guaranteed in general as it depends on the quality of the symbolic interval analysis and on the heuristic used to choose hyperplanes for splitting.

V. Experiments

Implementation and setup: Our contributions are implemented in a tool we call *DCut*[1]. It is implemented on top of ReluVal[2] [12] using the linear relaxation of Neurify [11]. DCut uses OpenBLAS version 0.3.21 for fast linear algebra operations and Gurobi version 11.0.2 for LP problems. DCut heavily uses parallelization to solve subproblems. We compare DCut against ReluVal [12], an established representative for plain input space splitting. Experiments were run on an AMD Ryzen Threadripper PRO 3955WX with 16 cores (32 threads) and 64GB memory running Ubuntu 22.04.5.

ACAS Xu Benchmark: The ACAS Xu system [7] is a well-known benchmark within NN verification. For our experiments, we are using the same ACAS Xu NNs and properties as used by ReluVal. In short, the ACAS Xu benchmark consists of a set of properties and a set of NNs. A subset of the NNs is associated with each property. The NNs have five input neurons, six hidden layers with 50 ReLU neurons each, and five output neurons. The description of the properties are found in [5] and [12]. In the following sections, we refer to a property-NN pair as an instance.

Measurements: We measure (i) the number of subproblems analyzed, i.e., how many times we had to split a problem to show robustness; (ii) subproblems' average depths were `safe` was returned; and (iii) the depth distribution of subproblems where `safe` was returned. Here, depth refers to how many times the original input space has been split to obtain the subproblem. Continuing, subproblems where `safe` was returned will be referred to as *verified subproblems*. Finally, we will discuss execution time.

Results: We discuss results for the instances involving properties 1 and 3 of the ACAS Xu benchmark. For these, we collected 44 and 39 data points, respectively. These results are representative of those involving instances of the remaining properties. Due to space constraints they are not included.

a) Number of subproblems: Our results show that DCut managed to generate fewer subproblems than ReluVal in 93.53% of the instances across all properties. Each point in Fig. 3 corresponds to a problem instance; points above the diagonal line show that ReluVal had to analyze more subproblems than our method. Specifically, in 86.36% and 100% of the instances for properties 1 and 3, respectively, our method analyzed (often much) fewer subproblems.

b) Average depth of verified subproblems: DCut achieved a lower average depth of verified subproblems in 45.82% of the instances across all properties. Fig. 4 shows the average depth of the verified subproblems of properties 1 and 3, respectively. Specifically, DCut had a lower average depth of verified subproblems in 22.72% and 87.17% of the instances for properties 1 and 3, respectively. This is interesting, because it means that even though our method had to analyze fewer subproblems, the subproblems we do analyze go deeper than ReluVal's. One explanation for this is that our method, because of the way we cut the input space, can conclude on some subproblems earlier, and the ones that are left for us to analyze are the ones where the Proc. "check" had to go deep and cut the input space many times. This obviously affects the average depth of the verified subproblems. Another explanation for this can be our heuristic for choosing neurons to define the cuts. It is possible that it results in poor progress; investigating alternative heuristics is a possible direction of future work.

c) Depth distribution of subproblems: Detailed results of depth distributions for specific instances show two things; there are instances where verified subproblems' depths are significantly lower for DCut, and DCut's distribution of verified subproblems' depths is smoother. Fig. 6 shows one instance where DCut's verified subproblems' depths are significantly lower. This indicates that our method can indeed find hyperplane cuts in the input space and tighter bounds that result in converging in a fewer number of steps. In both Fig. 5 and Fig. 6 the depth distribution of ReluVal's verified subproblems shows that there are certain depths where ReluVal can conclude on many subproblems while there are depths where it can only conclude on a few. DCut's smoother distribution shows each cut in the input space (resulting in subproblems with higher depth) consistently contributes to verifying subproblems. ReluVal's "spiky" behavior indicates it cuts subproblems even when it does not contribute in verification.

d) Execution time: In only 11.59% of the instances across all properties had DCut shorter execution times. This is easily explained by the fact that our method makes use of more expensive technology, namely LP-solving. This however allowed DCut to cut the input space in such a way that it becomes possible for us to verify properties in a fewer number of steps.

VI. Conclusion and Future Work

As mentioned in Section IV, there are different viable heuristics that can be used to choose the neuron that will be used to define the hyperplane cutting the input space. One direction for future work is to investigate different heuristics for this choice. A limitation of our method is that we can only assure that the neuron we choose to base our cuts on can be made linear. It would be beneficial if we could make this choice and assure that several neurons are made linear by only one cut in the input space.

Something that is noticeable from Section V is that DCut seems to perform better on some properties, i.e., it performs better on property 3 than it does on property 1. Since DCut is relatively expensive to use, due to the need for LP-computations of input bounds, it would be beneficial to characterize what type of properties our method performs better at. This could inform the use of our method such that cheaper methods could be used to verify many instances, but switch to our effective, but slightly expensive, method when the property to be checked has certain characteristics. We suspect this is the case when correctness needs to account for the strong correlation between a subset of input dimensions.

In line with the previous paragraph, integrating our method as part of larger framework is another direction of future

[1]https://github.com/hycut/dcut_date26

[2]https://github.com/tcwangshiqi-columbia/ReluVal

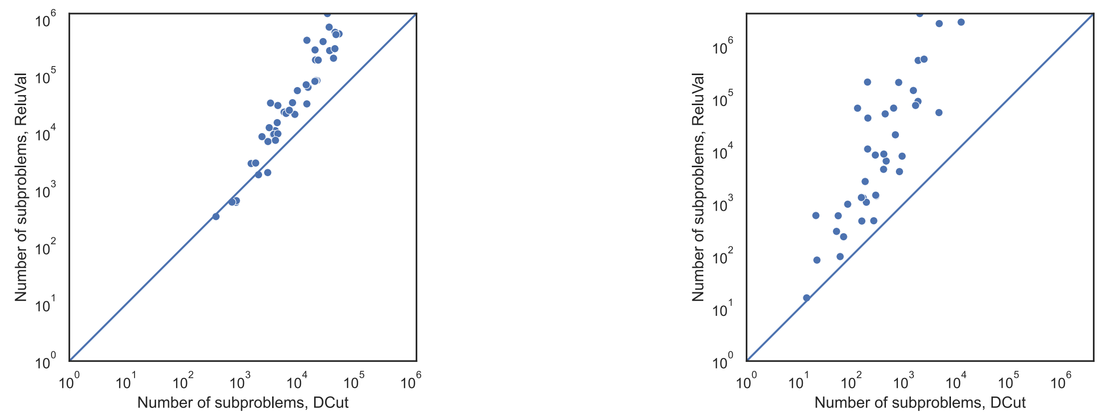

Fig. 3: Analyzed subproblems to conclude `safe` on 44 instances of property 1 (left) and 39 instances of property 3 (right).

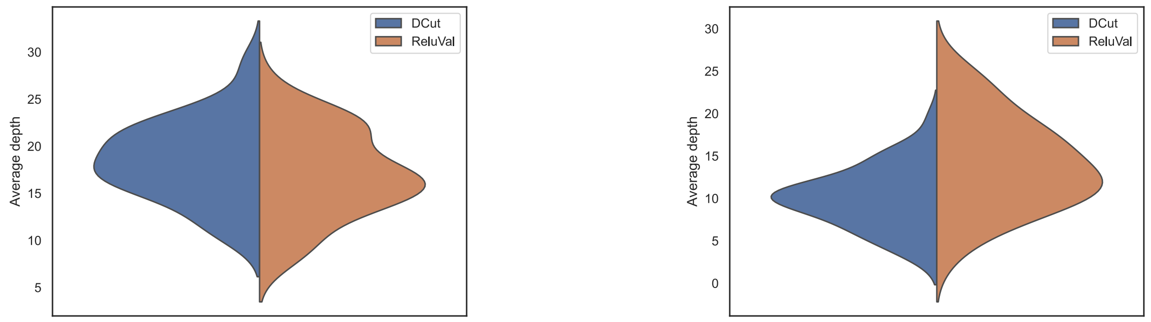

Fig. 4: Average depth of verified subproblems across 44 instances of property 1 (left) and 39 instances of property 3 (right).

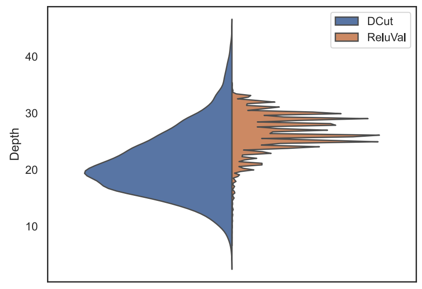

Fig. 5: Depth of 17532 and 621592 verified subproblems for DCut and ReluVal, respectively, on on an instance of property 1.

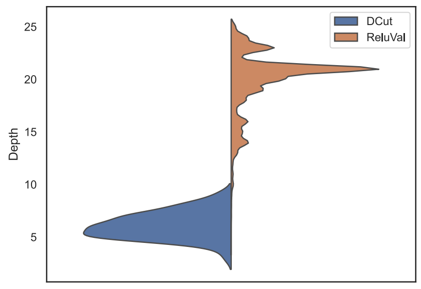

Fig. 6: Depth of 64 and 33000 verified subproblems for DCut and ReluVal, respectively, on on an instance of property 3.

work. There are several well-known and well-engineered tools in the NN verification area, e.g., α, β-CROWN [13], [15] and Marabou [6], [14]. Investigating whether our method can be integrated with such tools is interesting since, e.g., α, β-CROWN does optimize the symbolic approximation of neurons in order to achieve tighter bounds [15]. We believe our method could benefit from such approximations.

To conclude, we have presented a new method for cutting the input space with arbitrary hyperplanes, and implemented it as DCut. Our method uses symbolic bounds of neurons to find such hyperplanes, and to find better bounds used by following symbolic the analyses. Experiments on the ACAS Xu benchmark show that (i) using our method less subproblems needed to be checked; (ii) the average depth of the subproblems is lower; and (iii) on many instances the depth of the verified subproblems is significantly lower, compared to an established plain splitting approach. The method shows promising results, and future work includes investigating how it can be integrated into existing verification frameworks.

REFERENCES

[1] Anahita Baninajjar, Kamran Hosseini, Ahmed Rezine, and Amir Aminifar. Safedeep: A scalable robustness verification framework for deep neural networks. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[2] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

[3] Serge Durand, Augustin Lemesle, Zakaria Chihani, Caterina Urban, and François Terrier. Reciph: Relational coefficients for input partitioning heuristic. In *1st Workshop on Formal Verification of Machine Learning (WFVML)*, 2022.

[4] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International symposium on automated technology for verification and analysis*, pages 269–286. Springer, 2017.

[5] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[6] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International conference on computer aided verification*, pages 443–452. Springer, 2019.

[7] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[8] Brandon Paulsen, Jingbo Wang, and Chao Wang. Reludiff: differential verification of deep neural networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 714–726, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[10] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

[11] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Advances in neural information processing systems*, 31, 2018.

[12] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1599–1614, 2018.

[13] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. Beta-crown: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

[14] Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, et al. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification*, pages 249–264. Springer, 2024.

[15] 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. *arXiv preprint arXiv:2011.13824*, 2020.