Analyzing Forward Robustness of Feedforward Deep Neural Networks with LeakyReLU Activation Function through Symbolic Propagation

Giulio Masetti[1,✉] and Felicita Di Giandomenico[1]

Institute of Science and Technology "A. Faedo", 56124, Pisa, Italy[1]
✉giulio.masetti@isti.cnr.it

Abstract. FeedForward Deep Neural Networks (DNNs) robustness is a relevant property to study, since it allows to establish whether the classification performed by DNNs is vulnerable to small perturbations in the provided input, and several verification approaches have been developed to assess such robustness degree. Recently, an approach has been introduced to evaluate forward robustness, based on symbolic computations and designed for the ReLU activation function. In this paper, a generalization of such a symbolic approach for the widely adopted LeakyReLU activation function is developed. A preliminary numerical campaign, briefly discussed in the paper, shows interesting results.

1 Introduction

Deep Neural Networks are increasingly exploited in a number of AI applications, ranging from traditional speech recognition and image recognition to innovative modern sectors such as self-driving cars [8]. However, in addition to great performance capability, employment of DNNs in dependability critical applications, such as autonomous vehicles, requires the satisfaction of high degrees of accuracy/robustness. Therefore, it is paramount to analyze DNNs to verify whether their robustness degree is satisfactory for the application contexts where they are going to be employed. Among the others, abstract interpretation has been proposed in [2] for verifying DNNs. Unfortunately, as recently pointed out in [11], abstract interpretation can result too imprecisely, due to the non-linearity in DNNs.

To mitigate this problem, in [11] a novel symbolic propagation technique has been proposed with the aim of improving the precision of DNNs verification through abstract interpretation. Their idea consists in symbolically representing, for each neuron, how its value can be determined by the values of some neurons at the previous levels. They developed symbolic computations for the ReLU activation function, and proved improvements in the evaluation of DNNs, such as a better lower bound on the robustness property.

As a further contribution in this stream, which showed promising to advance on the robustness evaluation aspects, this paper offers a generalization of the symbolic computation technique to the LeakyReLU activation function case.

LeakyReLU was introduced as one of the strategies to mitigate the "dead neuron" problem that affects DNNs with the ReLU activation function: when a neuron outputs zero, it is quite unlikely that a relatively small change of the input can produce an output different from zero. It gained rather high popularity within the community. Actually, ReLU is a special case of LeakyReLU and the common features are exploited to adapt the reasoning of [11] to the one presented in this paper. A preliminary numerical campaign, briefly discussed in the paper, shows interesting results.

Given the narrowed scope of this paper, that focuses on extending the recently proposed approach in [11], we limit the review of the state of the art on DNN robustness verification through symbolic computations to [11], and rely on the positioning of the contribution already addressed there, after verification that there isn't (at the best of our knowledge) other advancements appeared in the literature in the meanwhile.

Enlarging the view on the notion of robustness, it can be observed that, although the formulation as in Definition 3 has been conceived in the context of adversarial training of DNNs, recently it has started to be enlisted under the eXplainable Artificial Intelligence (XAI) umbrella concept. In fact, it can be considered also as a prerequisite for a DNN to be "interpretable", because otherwise the interpretation has to comprise the sensitivity of the DNN to small input changes, resulting in a tedious description of too low level features of the DNN. Even further, the explanation approach itself has to be robust to small changes in the input, and many well-known approaches were proven not robust enough [9]. Robustness is a key feature of linear (regression) models, so one way to impose a certain level of robustness to DNN-based classifiers, designed having in mind interpretability, is to modify the learning process addressing locally difference boundness [7]. Here, instead of imposing "from the outside" some property to the DNN, the focus is on those activation functions for which, given any point other than zero, there exists a neighbourhood where the function is linear, in particular (Leaky)ReLU. This paper has a narrower scope than [7], where higher level features instead of pixels are considered. However, here an explicit dependency of the output from the input is provided.

Another example of training process that can take into consideration the robustness of the DNN is [4], while [5] refers to text classification under adversarial symbol substitutions. Both are originated in the context of robustness to adversarial perturbations of the input, but the reasoning is applicable in the context of XAI. Checking robustness can be formalized as solving a constrained optimization, where the constraints encode the DNN, followed by testing whether the result is less than zero. This computation can be easily integrated within the DNN training process, promoting the definition of robust neural networks. The only requirement on the activation function is that it must be monotonic so that, when propagating a box enclosing the given input through the DNN for

solving the optimization, only the intervals' bounds are relevant for the computations. Being designed to work for all the monotonic activation functions, this approach does not exploit a peculiar feature of (Leaky)ReLU: definitely activation/deactivation, as detailed in Section 3. In [4,5] all the bounds are propagated through all the layers of the DNN, instead here (following [11]) the bounds are propagated only when strictly necessary, promoting the tightness of the output layer's bounds.

The rest of the paper is structured as follows. Section 2 recalls some preliminaries, in particular Section 2.2 justifies the choice of the LeakyReLU activation function. Section 3 extends the work of [11] to the LeakyReLU activation function. Section 4 presents the experimental evaluation. Section 5 draws the conclusions and discusses future work.

2 Preliminaries

We work with deep feedforward neural networks (DNNs), represented as the evaluation of functions $F : \mathbb{R}^m \to \mathbb{R}^n$, where there are m input neurons and n output neurons.

As widely known, a DNN is structured as a sequence of layers: the input layer, comprising all the m input neurons, followed by n_{hl} hidden layers, and an output layer, comprising all the n output neurons, in the end. The output of a layer is the input of the next layer. To ease the notation, in this[1] paper only *fully connected* layers are considered, meaning that the output of each neuron of a layer is connected to the input of each neuron in the next layer. The DNN is then represented as the composition of transformations between layers. In particular, calling m_k the number of neurons within the k-th layer, the transformation between the $(k-1)$-th and the k-th layer is indicated as $F^{(k)} : \mathbb{R}^{m_{k-1}} \to \mathbb{R}^{m_k}$. Here $m_0 = m$ and $m_{n_{hl}+1} = n$, so that in total there are $m + m_1 + m_2 + \cdots + m_{n_l} + n$ neurons.

Typically, in each layer, a linear transformation is followed by a non-linear activation function, so the application of $F^{(k)}$ requires two intermediate steps:

$$F^{(k-\frac{1}{2})} = W^{(k)} F^{(k-1)} + b^{(k)}, \tag{1}$$

$$F^{(k)} = \text{map } f \text{ over } F^{(k-\frac{1}{2})}, \tag{2}$$

where Equation (1) is the weighted sum (plus bias) of the neurons of the $(k-1)$-th layer, Equation (2) is the application of the non-linear *activation function* $y = f(x)$ over each component of the vector $F^{(k-\frac{1}{2})}$. With a minor notation abuse, we will indicate with $F^{(k)}$ both the function and the vector

$$F^{(k)}(F^{(k-1)}(F^{(k-2)}(\cdots F^{(0)}(\texttt{in})\cdots))),$$

where `in` is the input vector. Here we focus on *classification* DNNs, i.e., the aim of the considered DNNs is to find $c(x)$, the class assigned to $x \in \mathbb{R}^m$, where $c(x) = \arg\max_i F_i(x)$.

[1] A discussion about convolutional layers and max pooling layer can be the subject of further studies.

2.1 Formal verification

A neural network is said to be robust at an input x if it does not change its predicted class on adding some small adversarial noise. Several techniques for verifying DNNs robustness have been proposed; here we focus on *abstract interpretation*. The literature on this subject is vast, but for the aims of this paper it is sufficient to recall that the central idea is: instead of working with a single point x, to work with a set of points $X \ni x$ that have a special "shape" and then to formally verify that a particular property holds.

More formally, we first define the *forward verification problem* for DNN and then *local forward robustness*.

Definition 1 (forward verification problem). *Given F, a domain $X \subseteq \mathbb{R}^m$ of the inputs and a property $C \subseteq \mathbb{R}^n$, we want to establish whether $F(X) \subseteq C$, where $F(X) = \{F(x)\ s.t.\ x \in X\}$.*

Definition 2 (local forward robustness). *Fixed a norm $||\cdot||_p$, given $\boldsymbol{in} \in \mathbb{R}^m$ and a fixed tolerance $\delta > 0$, we want to investigate the verification problem defined by $X = B(\boldsymbol{in}, \delta) = \{x \in \mathbb{R}^m\ s.t.\ ||x - \boldsymbol{in}||_p \leq \delta\}$ and $C_L = \{y \in \mathbb{R}^n\ s.t.\ \arg\max_i y_i = L\}$, where $L = \arg\max_i F_i(\boldsymbol{in})$.*

Usually δ is considered constant across all the analysis, so in the literature is often employed the δ-robustness:

Definition 3 (δ-robustness). *A given DNN is said δ-robust if, fixed a norm $||\cdot||_p$ and given $\boldsymbol{in} \in \mathbb{R}^m$, the local forward robustness is verified.*

The most commonly adopted "shapes" include boxes, zonotopes and polyhedra. In this paper we address only boxes, for which it is convenient to consider the infinity norm $||x||_\infty = \max_i |x_i|$. This eases the notation and allows us to focus on the core ideas, observing that our reasoning can be applied almost straightforwardly to polyhedra, whereas polytopes require more care. For a brief, but complete, description of abstract interpretation please refer to [11].

Being X a box, we can represent it as $[l^{(0)}, u^{(0)}]$, i.e., for each i, we keep track of the lower ($l_i^{(0)}$) and upper ($u_i^{(0)}$) bounds of x_i, and propagate these bounds through F, recording in $[l^{(k)}, u^{(k)}]$ the bounds of the box in the k-th layer.

It is well-known that, just considering the propagation of a box through $F^{(k-\frac{1}{2})}$, the first computational step of the k-th neuron as defined in Equation (1), can lead to precision loss due to the very nature of interval arithmetic (see Example 1). Thus, to mitigate this issue we follow [11] in resorting to a symbolic computation approach. In addition, also the second computational step, defined in Equation (2), can lead to precision loss (see Example 2) and then the activation function has to be selected with great care.

2.2 Activation function

In the literature, the activation function at the basis of neurons computation $f : \mathbb{R} \to \mathbb{R}$ is usually chosen among the following: sigmoid(x), tanh(x), ReLU(x),

LeakyReLU(x), eLU(x). Different characteristics of f guide not only the choice of the most appropriate[2] learning algorithm, taking into account the problem at hand and its context of application, but also the kind of robustness analysis we can perform. A summary of the properties for the above mentioned activation functions is reported in Table 1.

	sigmoid	tanh	ReLU	LeakyReLU	eLU
bounded	✓	✓			
invertible	✓	✓		✓	✓
differentiable	✓	✓			
differentiable for $x \neq 0$	✓	✓	✓	✓	✓
test + operation			✓	✓	✓
test + linear			✓	✓	

Table 1. Classification of activation functions.

For the purpose of our study that focuses on symbolic expressions, it is required the property to test "is x greater than zero?", that is a "definitely behavior", as better explained in Section 3. This property is shown by ReLU(x), LeakyReLU(x), eLU(x). However, for the last activation function, the transformation following the test is not linear, which is a necessary condition to apply symbolic computation, as we are interested in this work. Therefore, the only two activation functions that are amenable to be treated through symbolic computation are ReLU(x), LeakyReLU(x), whose equations are as follows:

$$\text{ReLU}(x) = \max\{0, x\} \tag{3}$$

$$\text{LeakyReLU}(x) = \max\{0, x\} + \alpha \cdot \min\{0, x\} \tag{4}$$

Since ReLU's robustness through symbolic computation has been already analyzed, in this paper we focus on Equation (4). Notice that we can also train the parameter α of LeakyReLU, instead of considering it as a constant, obtaining the so called PReLU. However, this has no impact on the reasoning of this paper, so in the following we refer to LeakyReLU only.

3 Symbolic Forward Propagation for DNNs

As already said, we adapt the symbolic computation approach developed in [11] for the ReLU activation function to the case where the activation function is LeakyReLU. Therefore, as in [11], we consider vectors of expressions $E^{(k-\frac{1}{2})}$

[2] A large body of knowledge is available on this subject, here we just mention [3] as a general overview and [10] for experimental comparisons among ReLU and some of its variants in the context of image recognition (convolutional DNNs are considered, as in [11]).

and $E^{(k)}$ together with the functions' $\min, \max : \text{Expr} \to \mathbb{R}$ for evaluating the minimum and maximum value of an expression, given that the symbols it comprises belong to known intervals.

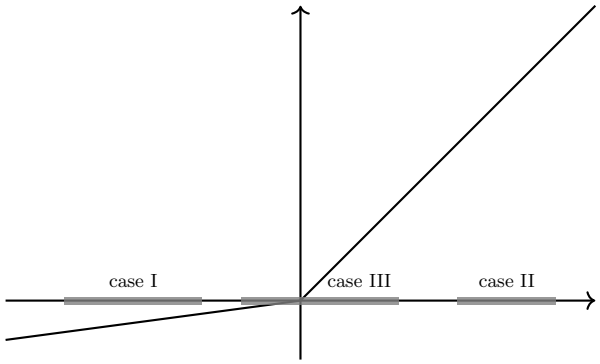

Fig. 1. Relative positions of $[l_i^{(k)}, u_i^{(k)}]$ with respect to 0, considering the LeakyReLU, i.e., Equation (4). Case I results in a definitely-deactivated neuron, Case II results in a definitely-activated neuron, Case III cannot be resolved in the k-th layer.

Only three cases are possible, according to the relative position of $\min(E_i^{(k-\frac{1}{2})})$ and $\max(E_i^{(k-\frac{1}{2})})$ with respect to 0, as depicted in Figure 1: case I, where the behavior is "definitely-deactivated", meaning that $E_i^{(k-\frac{1}{2})}$ is always less than 0 because $\max(E_i^{(k-\frac{1}{2})}) < 0$; case II, where the behavior is "definitely-activated", meaning that $E_i^{(k-\frac{1}{2})}$ is always greater than 0 because $\min(E_i^{(k-\frac{1}{2})}) \geq 0$; and case III, where $E_i^{(k)}$ can be either negative or positive, since $\min(E_i^{(k-\frac{1}{2})}) < 0$ and $\max(E_i^{(k-\frac{1}{2})}) > 0$. For $i = 1, \ldots, m_k$, the expression $E_i^{(k-\frac{1}{2})}$ comprises a linear combination of the expressions $E_j^{(k-1)}$, and we can decide how to define $E_i^{(k)}$ taking into account the properties of the activation function.

As shown in Equation (3) with respect to Equation (4), ReLU is a special case of LeakyReLU, where $\alpha = 0$. Therefore, the novelty of our approach consists in adapting the previous reasoning, valid for $\alpha = 0$, to the LeakyReLU, where $\alpha > 0$, that is changing the treatment of case I and case III. LeakyReLU is linear when evaluated on $\mathbb{R}^+$ (case II) or $\mathbb{R}^-$ (case I), so we can define $E_i^{(k)}$ always to be linear in the symbol it comprises: if we know for sure that $E_i^{(k-\frac{1}{2})}$ is either in $\mathbb{R}^-$ or $\mathbb{R}^+$ then we can define $E_i^{(k)}$ as $E_i^{(k-\frac{1}{2})}$ times a constant. Otherwise (case III) we can introduce a fresh symbol s_i^k, discard the expression $E_i^{(k-\frac{1}{2})}$ and define $E_i^{(k)} = s_i^k$, so to have again a linear expression in the symbols. More specifically, the action that corresponds to Equation (1) is $E_i^{(k-\frac{1}{2})} = \sum_j W_{ij}^{(k)} E_j^{(k-1)} - b_i^{(k)}$ followed by a symbolic simplification. Observing that $E_i^{(k)}$ is linear in $E_i^{(k-\frac{1}{2})}$ or

s_i^k, $E_i^{(k-\frac{1}{2})}$ is linear in $E_j^{(k-1)}$, and that $E_i^{(0)} = s_i^{(0)}$, we can prove by induction that $E_i^{(k)}$ is linear in the symbols. This fact does not only promote the symbolic computations, but also guarantees that the gradient of $E_i^{(k)}$, with respect to the symbols it comprises, is a constant. Thus, min and max can be easily evaluated.

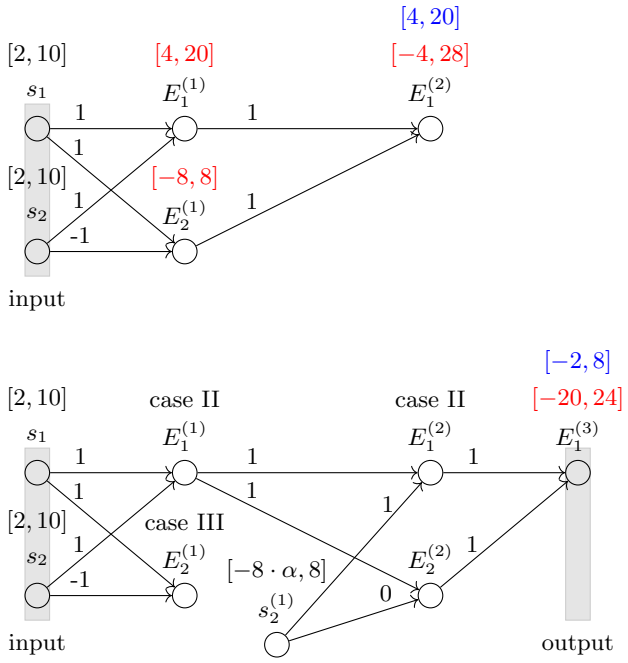

Fig. 2. Representation of Example 1 (top) and Example 2 (bottom). In red are reported the intervals obtained only through interval arithmetic, in blue the intervals obtained exploiting symbolic computations. The second neuron of the first hidden layer in Example 2, being of case III, is replaced by a new neuron with a fresh symbol. Only the relevant neurons and arcs are depicted.

Evaluating the bounds for each neuron in each layer, just using the interval arithmetic as in [2], can lead to a loss of precision, as illustrated in Example 1, because at increasing of the number of hidden layers the chances of over-approximate the bounds increase due to the very nature of interval arithmetic. Thus, the bounds are actually evaluated only when we are not sure of the sign of $E_i^{(k-\frac{1}{2})}$, i.e., when we introduce a fresh symbol. This gives us also a metric to evaluate how often we compute the bounds because it is sufficient to count the number of new symbols (indicated as Δ_{count}) that are introduced.

The pseudocode of the described procedure is listed in Algorithm 1. The benefits of evaluating the bounds for each neuron in each layer through sym-

bolic computations instead of interval arithmetic are essentially related to the occurrence of *cancellation* [1] and are illustrated through Example 1. Notice that the type of symbolic computation we are interested in is also known as "affine arithmetic".

Example 1. Consider the DNN, depicted on top of Figure 2, with two two-dimensional hidden layers and identity activation function defined by (the second neuron of the second layer is not relevant for the example):

$$\begin{aligned} &E_1^{(1)} = s_1 + s_2,\ E_2^{(1)} = s_1 - s_2, \\ &\quad \text{input layer: } s_1, s_2 \in [2, 10], \\ &E_1^{(2)} = E_1^{(1)} + E_2^{(1)} = s_1 + s_2 + s_1 - s_2 = 2s_1, \end{aligned}$$

where evaluating the expressions in layer 1 produces $[l_1^{(1)}, u_1^{(1)}] = [4, 20]$ and $[l_2^{(1)}, u_2^{(1)}] = [-8, 8]$, and then in layer 2 we have $[l_1^{(2)}, u_1^{(2)}] = [-4, 28]$, whereas simplifying $E^{(2)}$ and evaluating it only in layer 2 we have $[l_1^{(2)}, u_1^{(2)}] = [4, 20]$, a smaller interval. △

Adopting LeakyReLU ensures that $E_i^{(k)}$ is always linear in the symbols it comprises, in particular: $\alpha \cdot E_i^{(k-\frac{1}{2})}$, if in case I; $E_i^{(k-\frac{1}{2})}$, if in case II; introduce a new symbol s_i^k, set $E_i^{(k)} = s_i^k$, $l_i^{(k)} = \alpha \cdot \min(E_i^{(k-\frac{1}{2})})$ and $u_i^{(k)} = \max(E_i^{(k-\frac{1}{2})})$, if in case III.

To appreciate the impact of the activation function, in Example 2 the identity activation function of Example 1 is replaced by the LeakyReLU.

Example 2. Consider the same DNN of Example 1 where we replace the linear activation function with LeakyReLU, setting $\alpha = 0.5$, as depicted at the bottom of Figure 2. Calling the symbols of input layer $s_1^0, s_2^0 \in [2, 10]$, we have

$$E_1^{(1)} = s_1^0 + s_2^0, \quad E_2^{(1)} = s_1^0 - s_2^0.$$

So $\min(E_1^{(1)}) \geq 0$, and then the first neuron of the first layer is in case II, whereas $\min(E_2^{(1)}) = -8$ and $\max(E_2^{(1)}) = 8$, so the second neuron of the first layer is in case III. Thus, the fresh symbol s_2^1 is introduced, the expression $E_2^{(1)}$ is set equal to s_2^1 and the bounds of s_2^1 are set to $[-8 \cdot \alpha, 8]$. The first neuron of the second layer is

$$E_1^{(2)} = E_1^{(1)} + E_2^{(1)} = s_1^0 + s_2^0 + s_2^1.$$

Thus, $\min(E_1^{(2)}) = 0$, and then the first neuron in the second layer is in case II. Define now the second neuron of the second layer as $E_2^{(2)} = E_1^{(1)}$ and the first neuron of the output layer as $E_1^{(3)} = E_1^{(2)} - E_2^{(2)}$. Taking into account the fact that the first neuron in the second layer is in case II we obtain $[-2, 8]$ as bounds for $E_1^{(3)}$, whereas just propagating the bounds from layer 1 to layer 3 produces $[-20, 24]$ as bounds for $E_1^{(3)}$. △

The combined effects illustrated in Examples 1 and 2 show that it is possible to obtain sharper bounds with symbolic computations than just propagating the box through the layers.

4 Experimental Evaluation

The experimental evaluation presented in the following aims at three major objectives: i) demonstrate the feasibility of the symbolic computation approach, described through Algorithm 1, to assess the robustness of the LeakyReLU activation function; ii) analyze the most computationally intensive operations performed by Algorithm 1, as well as evaluate Δ_{count}, that is the number of new symbols s_i^k introduced during its execution. This last corresponds to the number of times case III is encountered, which is the most problematic case inducing precision loss in the computation; and iii) compare the robustness of ReLU and LeakyReLu.

To this purpose, starting from available educational code[3], we have implemented[4] Algorithm 1 in MATLAB. As dataset, we adopted the MNIST[5] dataset of handwritten digits, among the most popular ones used in similar studies (including [11]). Two DNNs have been developed and trained. The first consists of $n_{\text{hl}} = 2$ layers, where the first layer is populated by 80 neurons, while the second one has 60 neurons. Since both LeakyReLU and ReLU are considered in the experiments, we denote with $N^R_{80,60}$ the one where the neurons in the hidden layers work with the ReLU activation function, and with $N^L_{80,60}$ the one where neurons work with the LeakyReLU activation function. The LeakyReLU's parameter has been chosen equal to $\alpha = 0.01$. The activation function of the output layer of both $N^R_{80,60}$ and $N^L_{80,60}$ is the identity function $f(x) = x$. The second DNN consists of $n_{\text{hl}} = 4$ hidden layers with 128 neurons each. Similarly, two variants $N^R_{4\times128}$ and $N^L_{4\times128}$ are exercised, distinguished by the adoption in the hidden layers of ReLU and LeakyReLU as activation functions, respectively. Again, the output layer's activation function is the identity function.

Considering networks with different numbers of hidden layers allows to investigate how the evaluation of δ-robustness (Definition 3) is impacted by both how deep the DNNs are and by the behavior of the activation function.

4.1 Analysis results

The accuracy of $N^R_{80,60}$ is 96.65%, whereas the accuracy of $N^L_{80,60}$ is 96.55%. We adopted two DNNs with almost the same accuracy, since we are interested in understanding whether LeakyReLU offers advantages from the robustness evaluation point of view over ReLU.

[3] `https://it.mathworks.com/matlabcentral/fileexchange/73010-mnist-neural-network-training-and-testing`

[4] `https://github.com/106ohm/DeepNeuralNetworkForwardRobustness`

[5] `http://yann.lecun.com/exdb/mnist`

Algorithm 1 Forward propagation of boxes for LeakyReLU

Require: a Forward Neural Network
Require: lower and upper bonds $l^{(0)}, u^{(0)} \in \mathbb{R}^m$ of `in`
Require: $\alpha > 0$
Ensure: lower and upper bonds $l^{(\text{out})}, u^{(\text{out})} \in \mathbb{R}^n$

```
for i = 1, ..., m do
    E_i^(0) ← s_i^(0)                          ▷ a fresh symbol for each entry of the input vector
end for
for k = 1, ..., n_hl do
    E^(k-1/2) ← W^(k) · E^(k-1) + b^(k)        ▷ symbolic computations
    simplify(E^(k-1/2))                        ▷ symbolic simplifications
    for i = 1, ..., m_k do
        if max(E_i^(k-1/2)) ≤ 0 then           ▷ i-th neuron is definitely-deactivated: case I
            E_i^(k) ← α · E_i^(k-1/2)
            continue
        else if min(E_i^(k-1/2)) ≥ 0 then      ▷ i-th neuron is definitely-activated: case II
            E^(k) ← E^(k-1/2)
            continue
        else                                   ▷ case III
            l_i^(k) ← α · min(E_i^(k-1/2))
            u_i^(k) ← max(E_i^(k-1/2))
            E_i^(k) ← s_i^(k)                  ▷ introduce a fresh symbol
        end if
    end for
end for
E^(out) ← W^(n_hl+1) · E^(n_hl) + b^(n_hl+1)
for i = 1, ..., m_{n_hl+1} do
    l_i^(out) ← min(E^(out))
    u_i^(out) ← max(E^(out))
end for
```

Focusing on the 37-th image of the MNIST test suit, both $N^R_{80,60}$ and $N^L_{80,60}$ correctly classify it as a "seven", as can be seen from the *out* columns of Table 2.

Setting $\delta = 0.01$, we propagate the box of width δ through both $N^R_{80,60}$ and $N^L_{80,60}$ obtaining the lower and upper bounds reported in Table 2. On one hand, both $N^R_{80,60}$ and $N^L_{80,60}$ are δ-robust, when tested on the 37-th image, and several information can be extracted from Table 2. Examining the upper bounds, particularly relevant for ReLU, of $N^R_{80,60}$ we can notice that digit "two" presents a quite large upper bound (0.846 for $N^R_{80,60}$ and 1.006 for $N^L_{80,60}$), and also other digits have upper bounds close to 0.4. This lead us thinking that we can gain insights about how $N^R_{80,60}$ judges the similarities among digits 7 and 2, and other digits, looking at upper bounds.

	$N^R_{80,60}$			$N^L_{80,60}$		
digit	lower	out	upper	lower	out	upper
0	-0.052	-0.002	0.327	-0.001	$-4.2 \cdot 10^{-5}$	0.048
1	-0.061	-0.004	0.024	-0.001	$-2.9 \cdot 10^{-6}$	0.026
2	-0.083	0.183	**0.846**	-0.005	0.070	**1.006**
3	-0.665	-0.027	0.380	-0.003	0.032	0.453
4	-0.137	-0.001	0.065	-0.002	$-1.6 \cdot 10^{-4}$	0.036
5	-0.275	-0.014	0.093	-0.002	$-2.2 \cdot 10^{-4}$	0.075
6	-0.086	-0.001	0.027	-0.001	$-7.9 \cdot 10^{-5}$	0.158
7	**0.278**	**0.997**	**1.699**	**0.722**	**1.026**	**1.047**
8	-0.351	-0.020	0.425	-0.001	0.004	0.047
9	-0.925	-0.121	0.462	-0.002	$-6.1 \cdot 10^{-5}$	0.046

Table 2. $N^R_{80,60}$ and $N^L_{80,60}$: lower bound, *out* layer and upper bound taking as *input* the 37-th image of the MNIST test suit.

On the other hand, the bounds of $N^L_{80,60}$ are much tighter than those of $N^R_{80,60}$, demonstrating that, for the 37-th image, $N^L_{80,60}$ is much more robust than $N^R_{80,60}$.

Concerning the evaluation of Δ_{count}, that is the number of fresh symbols added when in case III. For $N^R_{80,60}$ at the end of the output layer there are in total 813 symbols, meaning that the new symbols are 29, since the computation started with $28{\cdot}28 = 784$ symbols. Instead, for $N^L_{80,60}$ we $\Delta_{\text{count}} = 816{-}784 = 31$. The better bounds shown by LeakyReLU are obtained despite the slightly higher value of Δ_{count} with respect to ReLU.

	$N^R_{4\times128}$			$N^L_{4\times128}$		
digit	lower	out	upper	lower	out	upper
0	-12.251	0.025	11.638	-0.233	0.003	21.637
1	-12.195	-0.039	13.346	-0.240	-0.001	25.460
2	-15.166	0.014	17.023	-0.294	0.017	24.191
3	-15.546	0.008	**17.404**	-0.159	0.046	17.880
4	-12.302	-0.006	12.466	-0.124	-0.006	24.270
5	-15.717	-0.028	14.635	-0.272	0.041	22.845
6	-13.579	0.007	15.936	-0.224	0.014	28.455
7	-21.041	0.024	13.046	-0.164	0.024	**34.517**
8	-13.160	0.052	15.740	-0.248	-0.034	29.613
9	-11.768	-0.014	15.475	-0.259	-0.003	25.208

Table 3. $N^R_{4\times128}$ and $N^L_{4\times128}$: lower bound, *out* layer and upper bound taking as *input* the 37-th image of the MNIST test suit.

At increasing of n_{hl} the chances of hitting a case III increase, and this is expected to produce a domino effect, leading to a higher number of fresh symbols, and then negatively affecting the tightness of the bounds. In order to study this phenomenon we train and test $N^R_{4\times128}$ and $N^L_{4\times128}$, the former with an accuracy of 97.78% and the latter of 97.60%, and then we propagate the box of width δ through them to obtain upper and lower bounds, reported in Table 3. $N^R_{4\times128}$ results *not* δ-robust because the digit with the greatest upper bound is "three" instead of "seven". Instead, $N^L_{4\times128}$ *is* δ-robust. This shows that, even though the bounds of $N^R_{4\times128}$ are tighter than those of $N^L_{4\times128}$, $N^L_{4\times128}$ is more resilient to slightly changes in the input. In addition, the lower bounds of $N^L_{4\times128}$ are much smaller than those of $N^R_{4\times128}$, so demonstrating that LeakyReLU maintains more information than ReLU. For $N^R_{4\times128}$, Δ_{count} results to be equal to $1020 - 784 = 236$, an increment of about 30%, whereas for $N^L_{4\times128}$ is $\Delta_{\text{count}} = 1064 - 784 = 280$. So, as before, $N^L_{4\times128}$ has slightly more fresh symbols than $N^R_{4\times128}$.

Focusing on a single neuron, though, we can observe that the amount of new symbols that affect its behavior are quite limited. For instance, the 67-th neuron of the fourth layer of $N^R_{4\times128}$ comprises: the 784 symbols defined within the input layer, 18 symbols defined in layer 1, 28 in layer 2 and 63 in layer 3. In total, this accounts to a 14% increment in the number of symbols.

4.2 Considerations on computational performance

Although the current implementation of Algorithm 1 is still preliminary, and does not exploit the inherent parallelism, we can observe a couple of interesting behaviors.

First, even though the overall computation of the bounds can be quite slow (it can take about two hours and a half for $N^R_{80,60}$ and $N^L_{80,60}$, and about five days for $N^R_{4\times128}$ and $N^L_{4\times128}$ on a low-performing laptop[6]) a clear pattern emerges: the computation time is linear in the number of symbols. In fact, in MATLAB the performance of all the expressions' manipulations is highly sensitive to the number of symbols they comprise and then, if Δ_{count} is small compared with the number of symbols within the input layer, then the computation time is linear in n_{hl}. In particular, for neurons in case I and II the expressions are just manipulated through the application of linear functions and the MATLAB function *isAlways*.

Second, when a fresh symbol is needed we also need to compute the bounds of the symbol, and this is the single most expensive atomic computation the algorithm needs to perform. There are several ways to accomplish this task; at the moment we employ the MATLAB *funmincon* function exploiting the fact, pointed out in Section 3, that the grading of an expression with respect to the symbols it comprises is a constant. Thus, if the vast majority of the neurons is in case I and II, the longer computations needed for case III are not

[6] CPU Intel(R) Core(TM) i3-6100U at 2.30GHz, 4Gb RAM at 2133MHz, running Ubuntu 20.04, MATLAB 2019b.

prevalent. Otherwise, as for $N^R_{4\times128}$ and $N^L_{4\times128}$, the computations related to case III overwhelm those related to the other cases.

5 From forward to backward propagation: initial thoughts

Inspired by [6], we realize that the symbolic computation techniques described in this paper can also impact other areas of Explainable Neural Networks research, in particular those focused on determining which characteristics of a given DNN primarily influence a specific result. The idea is to symbolically propagate back through the DNN a box enclosing a given output, to determine the box enclosing those inputs that are mapped to the selected output. Not being able to classify this kind of reasoning according to already established nomenclature, such as the one discussed in [9], we can refer to it as "backward behavior". So far, this kind of analysis couldn't be considered feasible mainly because of two obstacles related to the activation function: on one side, among the most widely adopted activation functions, those that are invertible (a necessary condition for backward behavior investigations) do not fit well with exact symbolic computations; and, on the other side, the one that is perfect for symbolic computation, i.e. ReLU, is not invertible. Now, the generalization of the work in [11] to the LeakyReLU, that is invertible (Table 1), paves the way to new possibility of explaining the behavior of DNNs.

However, although the premises are encouraging, the research is still at an initial stage. The main challenge about backward behavior of DNNs seems to be related to the inversion of the symbolic expression manipulation that corresponds to Equation (1). DNNs are often characterized by having hidden layers with different numbers of neurons, so $W^{(k)}$ is rectangular, and it can also happen that $W^{(k)}$ is square but not full rank. Maintaining a symbolic approach, but trading on the abstract interpretation's shape, we could resort to use the Moore-Penrose pseudoinverse, that is the best approximant for $||\cdot||_2$, in particular considering that almost always $W^{(k)}$ is full rank. Alternatively, maintaining $||\cdot||_\infty$ but trading on symbolic computations we can define an optimization problem to find one of the $F^{(k-1)}$ such that $W^{(k)}F^{(k-1)} + b^{(k)} = F^{(k+\frac{1}{2})}$ given $F^{(k+\frac{1}{2})}$. Both solutions are not satisfactory: the first, according to our preliminary experiments, produce boxes that are too large to have some meaning; the second because a single point cannot be representative of an interval, so the entire concept of abstract interpretation is not maintained.

It is expected that from these initial thoughts new ideas will be triggered, to possibly find the way to advance in this research direction.

6 Conclusions and future work

This paper presented an approach to assess forward robustness of DNNs, where the adopted activation function is LeakyReLU. The approach exploits symbolic computation, as recently proposed in [11] for the case of the ReLU activation

function. A preliminary campaign of experiments has demonstrated the feasibility of the approach and the better robustness obtained by DNNs employing LeakyReLU with respect to those employing ReLU. This is mainly due to the dying problem affecting ReLU, that is actually alleviated by LeakyReLu, thus confirming the value of developing our symbolic computation technique for an activation function that has good popularity within the community.

From the initial investigations presented in this paper, several research lines are foreseen as interesting advancements. The most natural ones are to apply the approach to deeper DNNs and to extend the abstract elements to shapes other than boxes. In particular, zonotopes and polyhedra are planned to be investigated. Another direction is testing Algorithm 1 on real-world scenarios to understand whether there are significant limitations to the application of the approach in concrete contexts. Of interest for a niche within the community, we can try to extend the concept of definitely-activated and definitely-deactivated to those complex activation functions defined after ReLU, for which several authors have proposed different ways to trade between holomorphicity and troubles related to the Liouville's theorem.

Of course, extending investigations on the backward behavior as discussed in Section 5 is a further study where we see interesting potentialities, although indirectly connected to the forward robustness this paper focuses on.

Acknowledgements. We would like to thank Prof. Laura Semini, University of Pisa, and Prof. Alessandro Fantechi, University of Florence, for the initial fruitful discussion on the subject of the paper, and the reviewers for their constructive suggestions.

References

1. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: VI Brazilian Symposium on Computer Graphics and Image Processing
2. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18 (2018)
3. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
4. Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., Kohli, P.: On the effectiveness of interval bound propagation for training verifiably robust models. arXiv preprint arXiv:1810.12715 (2018)
5. Huang, P.S., Stanforth, R., Welbl, J., Dyer, C., Yogatama, D., Gowal, S., Dvijotham, K., Kohli, P.: Achieving verified robustness to symbol substitutions via interval bound propagation. arXiv preprint arXiv:1909.01492 (2019)
6. Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W., Müller, K.: Unmasking clever hans predictors and assessing what machines really learn. CoRR abs/1902.10178 (2019)
7. Melis, D.A., Jaakkola, T.: Towards robust interpretability with self-explaining neural networks. In: Advances in Neural Information Processing Systems. pp. 7775–7784 (2018)

8. Sze, V., Chen, Y., Yang, T., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE 105(12), 2295–2329 (2017)
9. Vilone, G., Longo, L.: Explainable artificial intelligence: a systematic review (2020)
10. Xu, B., Wang, N., Chen, T., Li, M.: Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv e-prints (2015)
11. Yang, P., Liu, J., Li, J., Chen, L., Huang, X.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: Proceedings of the 26th International Symposium on Static Analysis. pp. 296–319 (2019)