OSIP: Tightened Bound Propagation for the Verification of ReLU Neural Networks

Vahid Hashemi[1], Panagiotis Kouvaros[2(✉)], and Alessio Lomuscio[2]

[1] Audi AG, Ingolstadt, Germany
[2] Imperial College London, London, UK
p.kouvaros@imperial.ac.uk

Abstract. Abstraction-based methods for the verification of ReLU-based neural networks suffer from rapid degradation in their effectiveness as the neural network's depth increases. We propose OSIP, an abstraction method based on symbolic interval propagation in which the choice of the ReLU relaxation at each node is determined via optimisation. We present an implementation of OSIP on top of Venus, a publicly available toolkit for complete verification of neural networks. In the experiments reported, OSIP calculated bounds that were tighter than the state-of-the-art on ReLU networks from the first competition for neural network verification. As a case study we apply the method for the verification of VGG16, a deep, high-dimensional, 300,000 node-strong model used for object classification in autonomous vehicles against local robustness properties. We demonstrate that OSIP could verify the correctness of the model against perturbations that are larger than what can be analysed with the present state-of-the-art.

1 Introduction

Methods based on machine-learning (ML) are increasingly being deployed in AI-based, safety-critical applications, including autonomous vehicles. Rather than being directly programmed by software engineers, these modules often take the form of neural networks (NN) synthesised directly from data. A notable example of these are the class of ML-based object detectors and classifiers presently used in autonomous vehicles. These are typically deep (i.e., multi-layered) networks often comprising hundreds of thousands of neurons that can automatically detect and classify objects of interest in an image, whether these are vehicles, humans, fixed and moving obstacles, etc. While the performance of neural classifiers is high, their error rates are often in the region of 1–2%, hence still too high to be deployed safely on their own. It is also known that neural models are particularly fragile against out of sample data, i.e., while the nominal performance may be high on data with the same statistical distribution of the training set, this may not be the case with out-of-distribution inputs. To mitigate these problems the area of verification of neural networks has grown to propose methods to verify the correctness of classifiers.

R. Calinescu and C. S. Păsăreanu (Eds.): SEFM 2021, LNCS 13085, pp. 463–480, 2021.
https://doi.org/10.1007/978-3-030-92124-8_26

Due to their importance in applications, particular emphasis to date has been devoted to methods addressing the verification of feed-forward (i.e., non-recurrent) neural networks based on Rectified-Linear Units (ReLU). A key specification that is analysed in this context is *l*ocal robustness. Simply stated, local robustness refers to whether or not the network alters its output in the presence of small changes to the input. This can be useful to analyse the network's stability in the presence of input noise, or its susceptibility to adversarial attacks [12].

Related work. Methods in formal verification of ReLU-based neural networks can be partitioned into *complete*, or exact, and *incomplete*, or approximate. Complete methods [2,4,6,9,11,13,17–19,21,21,22,29,31,32] can theoretically solve a verification query, such as local robustness, both with a positive and a negative answer. However, they also suffer from scalability issues and may not be able to resolve queries when the network or the perturbation range is large.

In contrast with this, incomplete methods [3,7,8,10,23,24,26,28,33,34,36,37,37] rely on various abstraction methods to overapproximate the computation of a neural network. Because of this overapproximation, incomplete methods can only certify that a network is compliant against a specification, but not that it is not. While leading incomplete approaches can solve some verification problems that cannot be established via complete approaches, they still fall short of being capable of addressing large models of industrial sizes. This is because the abstraction methods are often too coarse thereby inhibiting the evaluation of significant perturbations on the input. This is particularly evident in *symbolic interval propagation* (SIP)-based methods [26,28,33–35,37] whereby the ReLU function is linearly approximated and the bounds for the nodes are computed via backward passes of variable substitutions through the network. Whilst this often achieves state-of-the-art scalability, it trades off precision by inducing coarser overapproximations as compared to other methods.

Contribution. In this paper we aim to make a contribution in this direction. We propose an incomplete, SIP-based method that improves the precision. Differently from related methods, where the relaxation for a ReLU node is heuristically chosen to induce the minimum local, i.e., at neuron-level, overapproximation area [26,37] or the minimum local maximum error [32], our method jointly determines via optimisation the relaxations to be conducted for all the nodes in a layer, thereby accounting for intra-layer dependencies to improve precision. Additionally, it provides a simple, yet effective treatment for max-pooling layers towards further improving precision. This enables solving verification queries that could not previously be determined by the state-of-the-art. We experimentally evaluate the method proposed by benchmarking on the ReLU networks from the first competition for neural network verification. Additionally we report the results obtained when analysing VGG16, an image classifier consisting of over 300,000 nodes. The results show that our method produces tighter bounds and is able to solve verification queries that cannot be solved by present methods.

The rest of the paper is organised as follows. In Sect. 2 we fix the notation and present key concepts used throughout the paper. Section 3 reports `OSIP`, the SIP method here developed. Section 4 reports details of the resulting

implementation and reports experimental results against ReLU networks from the first competition for neural network verification and VGG16. We conclude in Sect. 5.

2 Preliminaries

Feed-Forward Neural Networks. A *feed-forward neural network* (FFNN) is a vector-valued function $\boldsymbol{f}\colon \mathbb{R}^{s_0} \to \mathbb{R}^{s_L}$ that composes a sequence of $L \geq 1$ layers $\boldsymbol{f}_1 : \mathbb{R}^{s_0} \to \mathbb{R}^{s_1}, \ldots, f_L : \mathbb{R}^{s_{L-1}} \to \mathbb{R}^{s_L}$. Each layer $\boldsymbol{f}_i$, $i \in \{1, \ldots, L-1\}$, is said to be a *hidden layer*; the last layer $\boldsymbol{f}_L$ of the network is said to be the *output layer*. Each element of each layer $\boldsymbol{f}_i$ is said to be a *neuron*, or a *node*. We use $n_{i,q}$ to refer to the q-th node of layer i. Each layer $\boldsymbol{f}_i$, $i \in \{1, \ldots, L\}$, implements one of the following functions for input $\boldsymbol{x}_{i-1}$:

1. an affine transformation $f_i(\boldsymbol{x}_{i-1}) = W^{(i)}\boldsymbol{x}_{i-1} + b_i$, for a weight matrix $W^{(i)} \in \mathbb{R}^{n_i \times n_{i-1}}$ and a bias vector $b \in \mathbb{R}^{n_i}$.
2. a ReLU activation function $f_i(\boldsymbol{x}_{i-1}) = \max(\boldsymbol{x}_{i-1}, 0)$, where the maximum function is applied element-wise.
3. a max-pool function which collapses rectangular neighbourhoods of its input into the maximal value within each neighbourhood.

Note that for ease of presentation we separate affine transformations from the ReLU activation function, where we consider each as a different layer, as opposed to their standard treatment whereby their composition defines a layer. Also, we hereafter assume that the bias vector for all the layers is the zero vector. There is no technical difficulty to extend the discussion to non-zero bias vectors.

Verification Problem. Given a FFNN, the verification problem is to answer positively or negatively as to whether the output of the network for every input within a linearly definable set of inputs[1] is contained within a linearly definable set of outputs. Formally, we have:

Definition 1. *Verification problem. Given a FFNN $\boldsymbol{f}$, a linearly definable set of inputs $\mathcal{X} \subset \mathbb{R}^{s_0}$ and a linearly definable set of outputs $\mathcal{Y} \subset \mathbb{R}^{s_L}$, the verification problem is to establish whether*

$$\forall \boldsymbol{x} \in \mathcal{X}\colon \boldsymbol{f}(\boldsymbol{x}) \in \mathcal{Y}.$$

One of the most well-studied instantiations of the verification problem is the *local adversarial robustness* problem. The problem concerns answering whether all images within a norm-ball of a given input image are classified equivalently by the network $\boldsymbol{f}$ [1,2,8,15,17,31]. Formally, the local adversarial robustness problem is derived from the verification problem by setting

$$\mathcal{X} = \{\boldsymbol{x}' \mid x - \epsilon \leq x' \leq x + \epsilon\}$$
$$\mathcal{Y} = \{\boldsymbol{y} \mid \forall i \neq c\colon \boldsymbol{f}(\boldsymbol{x'})_i < \boldsymbol{f}(\boldsymbol{x'})_c\}\,,$$

[1] A linearly definable set is a set that can be expressed as a finite set of affine constraints over real-valued variables.

Algorithm 1. Verification via over-approximation.

```
1: procedure VERIFY(f, 𝒳, 𝒴)
2:     Input: network f, set of inputs 𝒳, set of outputs 𝒴
3:     Output: yes/unknown
4:     compute R̂ such that R̂ ⊇ {f(x) | x ∈ 𝒳}
5:     if R̂ ⊆ 𝒴 then
6:         return yes
7:     else
8:         return unknown
```

for a given image $\boldsymbol{x}$ with class label c and perturbation radius $\epsilon \geq 0$. In this paper, we focus on verification problems whereby the set of inputs $\mathcal{X}$ is defined by a lower and an upper bound for each element of the input $\boldsymbol{x}_0$ to the network, i.e., $\mathcal{X} = \{\boldsymbol{x}_0 \mid l_i \leq \boldsymbol{x}_{0,i} \leq u_i\}$, where l_i, $u_i \in \mathbb{R}$.

3 OSIP: Tightened Bound Propagation

In this section we present OSIP (*optimised* SIP), a novel tight symbolic interval propagation method for the verification of feed-forward neural networks.

Given a network and lower and upper bounds of its inputs, OSIP estimates lower and upper bounds of the network's output nodes. OSIP can then potentially use these bounds to determine the satisfaction of the verification property in question. OSIP is incomplete in that the bounds may be overestimated to such a degree that solving the verification problem is not possible.

The key novel element of OSIP consists in a novel treatment of the ReLU function whereby the linear approximation of the function is selected via optimisation. As it will be clear in the next section, this results in a method that in experiments calculates the tightest overestimation when compared to leading methods. The size of the approximation is essential in incomplete methods. Intuitively, the smaller the uncertainty, the more likely it is that a verification query can be solved.

Overview. OSIP is an instance of verification algorithms operating by overapproximating the network computation (see Algorithm 1 for a high level description of this class of algorithms). The method computes an overapproximation $\hat{\mathcal{R}}$ of the reachable output set $\mathcal{R} = \{\boldsymbol{f}(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}\}$ for a given network $\boldsymbol{f}$ and set of inputs $\mathcal{X}$ concerning a verification problem $\forall \boldsymbol{x} \in \mathcal{X}\colon \boldsymbol{f}(\boldsymbol{x}) \in \mathcal{Y}$. This over-approximation is obtained from a layer-by-layer application of the layers' functions to the input set $\mathcal{X}$, where the ReLU function is linearly relaxed.

Depending of the tightness of the overapproximation, Algorithm 1 may or may not be able to solve the verification problem. In particular, if $\hat{\mathcal{R}} \subseteq \mathcal{Y}$, then the algorithm outputs yes, i.e., the verification property is satisfied. For

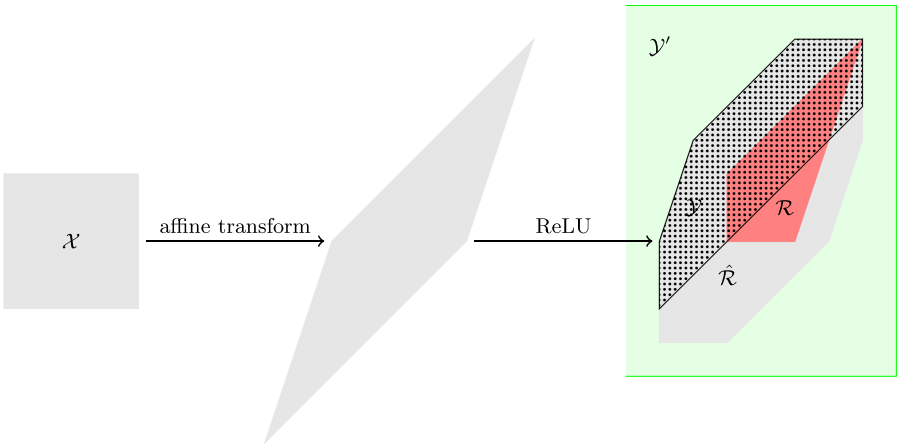

Fig. 1. Over-approximation of a network's bounds. The network has a two-dimensional input and two layers: an affine transformation layer and a ReLU layer. The verification problem $\forall \boldsymbol{x} \in \mathcal{X} \colon \boldsymbol{f}(\boldsymbol{x}) \in \mathcal{Y}'$ is satisfied whereas the verification problem $\forall \boldsymbol{x} \in \mathcal{X} \colon \boldsymbol{f}(\boldsymbol{x}) \in \mathcal{Y}$ cannot be solved.

instance, if analysing a local adversarial problem and $\hat{\mathcal{R}} \subseteq \mathcal{Y}$, then all images whose network output is within $\hat{\mathcal{R}}$ are classified equally to the image given as input to the problem. Since these images form a superset of the set $\mathcal{X}$ of images within the norm-ball of the image in question, the verification problem is satisfied.

Otherwise, if $\hat{\mathcal{R}} \not\subseteq \mathcal{Y}$, then Algorithm1 outputs `unknown`, i.e., the verification problem cannot be solved. For instance, if analysing a local adversarial problem and $\hat{\mathcal{R}} \not\subseteq \mathcal{Y}$, then any image whose network output is within $\hat{\mathcal{R}} \backslash \mathcal{Y}$, i.e., any image that potentially falsifies the verification problem, may or may not lie within the norm-ball of the image in question; consequently, it cannot be used to falsify the verification problem. Figure 1 gives a graphical illustration of these two possible outcomes of Algorithm 1.

Detailed Description. In line with previous symbolic interval propagation methods [26,31,32,37], `OSIP` analyses a given network in a layer-by-layer fashion, where for each node $n_{i,j}$, it constructs the following:

- a (symbolic) linear constraint $\boldsymbol{\beta}^{\geq}_{i,j}$ of its lower bound, built from variables expressing the inputs to the node,
- a similarly defined linear constraint $\boldsymbol{\beta}^{\leq}_{i,j}$ of its upper bound;
- a concrete (i.e., numeric) lower bound $\boldsymbol{l}_{i,j}$,
- a concrete upper bound $\boldsymbol{u}_{i,j}$.

This results in the derivation of concrete lower and upper bounds for the output nodes of the network, which can potentially be used to determine the satisfaction of the verification problem as per Algorithm 1.

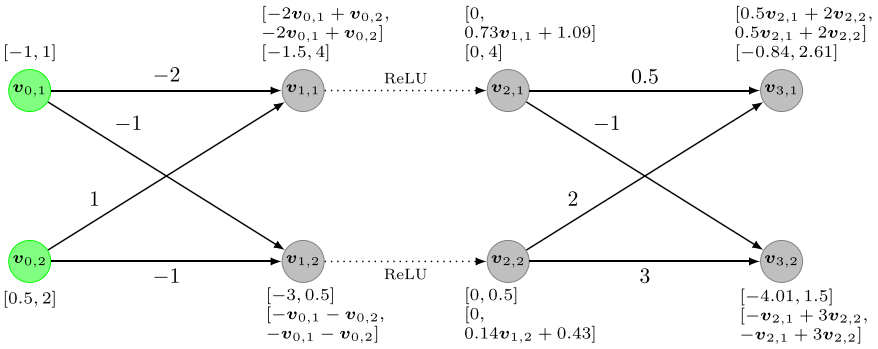

Fig. 2. A feed-forward neural network. The inputs are depicted in green colour. Concrete ranges next to a node indicate the concrete lower and upper bounds of the node as obtained via `OSIP`. Symbolic ranges next to a node indicate the lower and upper bound constraints of the node as obtained via `OSIP`.

Following the presentation from [26] we now describe the computation of the constraints and bounds for each of the types of layers constituting the networks considered in this paper. The novel elements of `OSIP` consist in the treatment of the ReLU and max-pool layers. In the following we use symbolic variables $\boldsymbol{v}_{i,j}$ for the value of each node $n_{i,j}$; we use these variables to build the bound constraints. To exemplify each computation we use the network from Fig. 2. We begin with the constraints and bounds of the input to the network. We then propagate these through the network to derive the constraints and bounds of the output of the network.

Input. The constraints and bounds for the input to the network are instantiated from the bounds of the input prescribed by the verification problem:

$$\boldsymbol{\beta}^{\geq}_{0,j} = \boldsymbol{l}_j; \quad \boldsymbol{\beta}^{\leq}_{0,j} = \boldsymbol{u}_j; \quad \boldsymbol{l}_{0,j} = \boldsymbol{l}_j; \quad \boldsymbol{u}_{0,j} = \boldsymbol{u}_j.$$

Affine Transformation Layer. Given the vector $\boldsymbol{v}_i = \left[\boldsymbol{v}_{i,1}, \boldsymbol{v}_{i,2}, \ldots, \boldsymbol{v}_{i,s_i}\right]^T$ of layer i's variables, the lower and upper bound constraints of an affine transformation layer $\boldsymbol{f}_{i+1}$ are defined by:

$$\boldsymbol{\beta}^{\geq}_{i+1,j} = \boldsymbol{\beta}^{\leq}_{i+1,j} = W^{(i+1)}_{j,:}\boldsymbol{v}_i.$$

In other words, the lower and upper bound constraints for affine transformation layers are identical and instantiated to the network function for the node in question.

Example 1. Consider the network from Fig. 2. Given the vector of input variables $\left[\boldsymbol{v}_{0,1}\ \boldsymbol{v}_{0,2}\right]^T$, the lower and upper bound constraints of node $n_{1,1}$ are

$$\boldsymbol{\beta}^{\geq}_{1,1} = \boldsymbol{\beta}^{\leq}_{1,1} = W^{(1)}_{1,:}\left[\boldsymbol{v}_{0,1}\ \boldsymbol{v}_{0,2}\right]^T = \left[-2\ 1\right]\left[\boldsymbol{v}_{0,1}\ \boldsymbol{v}_{0,2}\right]^T = -2\boldsymbol{v}_{0,1} + \boldsymbol{v}_{0,2}.$$

The concrete bounds of the layer are obtained by replacing the variables in the nodes' constraints with their associated lower or upper bound constraints, depending on the signs of the variables. In particular, to compute the lower bound (upper bound, respectively) of a node, we replace the variables within its lower bound constraint (upper bound constraint, respectively) with the lower bound constraints (upper bound constraints, respectively) of the nodes of the previous layer if the sign of the variables is positive; otherwise, if it is negative, then we use the upper bound constraints (lower bound constraints, respectively) of said nodes. We continue by replacing the newly introduced variables with their corresponding constraints, and so on, until the constraints depend only on the input variables whereby we can compute the concrete bounds.

Formally, the derivation of the concrete bounds is defined as follows. We begin by replacing the $v_{i,j}$ variables in $\boldsymbol{\beta}^{\geq}_{i+1}$ and $\boldsymbol{\beta}^{\leq}_{i+1}$ with their corresponding bound constraints:

$$\boldsymbol{\beta}^{\geq}_{i+1,j} = \left(W^{(i+1)-}_{j,:}\boldsymbol{\beta}^{\leq}_{i} + W^{(i+1)+}_{j,:}\boldsymbol{\beta}^{\geq}_{i}\right)\boldsymbol{v}_{i-1},$$
$$\boldsymbol{\beta}^{\leq}_{i+1,j} = \left(W^{(i+1)-}_{j,:}\boldsymbol{\beta}^{\geq}_{i} + W^{(i+1)+}_{j,:}\boldsymbol{\beta}^{\leq}_{i}\right)\boldsymbol{v}_{i-1},$$

where:

- $W^{(i+1)-}$ and $W^{(i+1)+}$ are obtained from $\min(W^{(i+1)}, 0)$ and $\max(W^{(i+1)}, 0)$ with the element-wise application of the min and max functions;
- $W^{(i+1)x}_{j,:}\beta^{y}_{i}$, $x \in \{-,+\}$, $y \in \{\leq,\geq\}$, denotes (with slight abuse of notation) the multiplication of $W^{(i+1)x}_{j,:}$ with the matrix of the coefficients of the constraints β^{y}_{i} over $\boldsymbol{v}_{i-1}$.

We then repeat this back-substitution step until all layers have been processed and the upper and lower bounds for the node in question are instantiated by numerical values.

Example 2. Consider the network from Fig. 2. We execute the computation of the concrete lower and upper bounds of node $n_{3,1}$ whose lower and upper bound constraints are

$$\boldsymbol{\beta}^{\geq}_{3,1} = 0.5 \cdot \boldsymbol{v}_{2,1} + 2 \cdot \boldsymbol{v}_{2,2} \leq \boldsymbol{v}_{3,1} \leq 0.5 \cdot \boldsymbol{v}_{2,1} + 2 \cdot \boldsymbol{v}_{2,2} = \boldsymbol{\beta}^{\leq}_{3,1}.$$

We begin by replacing the variables $\boldsymbol{v}_{2,1}$, $\boldsymbol{v}_{2,2}$ in $\boldsymbol{\beta}^{\geq}_{3,1}$ with the lower bound constraints of nodes $n_{2,1}$, $n_{2,2}$ and by replacing said variables in $\boldsymbol{\beta}^{\leq}_{3,1}$ with the upper bound constraints of said nodes:

$$0 \leq \boldsymbol{v}_{3,1} \leq 0.37 \cdot \boldsymbol{v}_{1,1} + 0.28 \cdot \boldsymbol{v}_{1,2} + 1.41.$$

Next, we perform similar replacements to the newly introduces variables:

$$0 \leq \boldsymbol{v}_{3,1} \leq -1.02 \cdot \boldsymbol{v}_{0,1} + 0.09 \cdot \boldsymbol{v}_{0,2} + 1.41$$

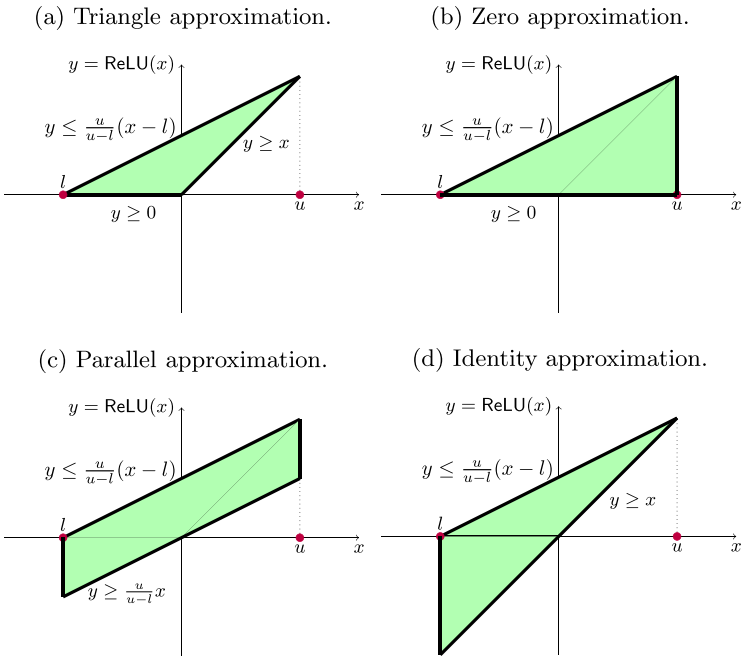

Fig. 3. Convex approximations of the ReLU function $\mathsf{ReLU}(x) = \max(x, 0)$.

From above, the concrete lower bound of $\boldsymbol{v}_{3,1}$ equals 0. To obtain its concrete upper bound, we replace $\boldsymbol{v}_{0,1}$ with the lower bound of its associated input and $\boldsymbol{v}_{0,2}$ with the upper bound of its associated input. This gives $0 \leq \boldsymbol{v}_{3,1} \leq 2.61$.

Remark 1. Observe that the concrete bounds for a node can alternatively be directly computed by replacing the variables in the bound constraints of the node with the concrete bounds of the nodes associated with the variables. While this is more efficient than the back-substituting the variables, it is known to lead to looser bounds [24].

ReLU layer. The derivation of the bound constraints for a ReLU node requires a convex approximation of the ReLU function $\mathsf{ReLU}(x) = \max(x, 0)$. The optimal convex approximation of the function is the triangle approximation [9]. The approximation bounds the function from above with $\mathsf{ReLU}(x) \leq \frac{u}{u-l}(x - l)$, where l and u are the lower and upper bounds of x, and from below with $\mathsf{ReLU}(x) \geq 0$, $\mathsf{ReLU}(x) \geq x$ (Fig. 3a). Though optimal, the approximation is problematic in that it uses two lower bound constraints, thereby leading to an exponential blow-up of the number of constraints required for the overall analysis [26]. To circumvent this, the ReLU function is instead typically bounded from below with $\mathsf{ReLU}(x) \geq \lambda x$, $\lambda \in [0, 1]$. Commonly used approximations are the *parallel approximation* with $\lambda = \frac{u}{u-l}$ (Fig. 3c) [31,33], the *zero approximation* with $\lambda = 0$ (Fig. 3b) [26,37] and the *identity approximation* with $\lambda = 1$

(Fig. 3d) [26,37]. State-of-the-art methods select a node's approximation either on the basis of the minimum maximum distance from the ReLU function [32] (i.e., always select the parallel approximation) or in terms of the smallest over-approximation area induced [26,37] (i.e., the select the identity approximation if $|l| < u$, otherwise the zero approximation). However, as we experimentally show in the next section, since these heuristic rules operate at a local, neuron level and do not account for intra-layer neuron dependencies, their comparative performance (in terms of the tightness of the derived output bounds) varies with different verification problems.

Differently from these works, we now propose a method that efficiently determines the ReLU approximation for each node via optimisation. The key idea is to jointly optimise the approximation slopes of a layer to bring about the tightest bounds in the subsequent layer. By jointly optimising the slopes we account for the nodes' intra-layer dependencies and the influence thereof in the bounds of the subsequent layer. As we experimentally show in the next section, this results in a method that consistently outperforms the leading methods.

We focus on optimising the slopes to maximise the concrete lower bounds of layer $i+1$; the case of minimising the upper bounds is analogous. We begin with setting

$$\boldsymbol{\beta}^{\geq}_{i+1,j} = \boldsymbol{\lambda}^{\geq}_{i+1,j}\boldsymbol{v}_{i,j}, \quad \boldsymbol{\beta}^{\leq}_{i+1,j} = \frac{\boldsymbol{u}_{i,j}}{\boldsymbol{u}_{i,j} - \boldsymbol{l}_{i,j}}(\boldsymbol{v}_{i,j} - \boldsymbol{l}_{i,j}),$$

where $\boldsymbol{\lambda}^{\geq}_{i+1,j}$ is an optimisation variable for the slope of node $n_{i+1,j}$ and $\boldsymbol{v}_{i,j}$ is the variable representing the input to said node. Now, recall that the bound constraints $\boldsymbol{\beta}^{\geq}_{i+2}$, $\boldsymbol{\beta}^{\leq}_{i+2}$ of the subsequent layer are constraints over the variables associated with the nodes at layer $i+1$. By performing a single back-substitution step we obtain the constraints $W^{(i+2)-}\boldsymbol{\beta}^{\leq}_{i+1} + W^{(i+2)+}\boldsymbol{\beta}^{\geq}_{i+1}$ which are over the variables associated with the nodes at layer i. These constraints can be used to compute the concrete lower bounds of layer $i+2$ by concretising their variables as follows: $\boldsymbol{l}_{i+2} = K^{+}\boldsymbol{l}_i + K^{-}\boldsymbol{u}_i$, where $K = W^{(i+2)-}\beta^{\leq}_{i+1} + W^{(i+2)+}\beta^{\geq}_{i+1}$. Our aim is to derive the approximations which maximise the sum of these bounds. Formally, our aim is to solve the following optimisation problem:

$$\begin{aligned} \max_{\boldsymbol{\lambda}^{\geq}_{i+1}} \quad & \sum_j K^{+}_{j,:}\boldsymbol{l}_{i,j} + K^{-}_{j,:}\boldsymbol{u}_{i,j} \\ \text{subject to} \quad & K = W^{(i+2)-}\beta^{\leq}_{i+1} + W^{(i+2)+}\beta^{\geq}_{i+1}, \\ & \boldsymbol{\beta}^{\geq}_{i+1,j} = \boldsymbol{\lambda}^{\geq}_{i+1,j}\boldsymbol{v}_{i,j}, \quad \boldsymbol{\beta}^{\leq}_{i+1,j} = \frac{\boldsymbol{u}_{i,j}}{\boldsymbol{u}_{i,j} - \boldsymbol{l}_{i,j}}(\boldsymbol{v}_{i,j} - \boldsymbol{l}_{i,j}), \\ & \boldsymbol{\lambda}^{\geq}_{i+1,j} \in [0,1]. \end{aligned} \tag{1}$$

The solution to this optimisation problem determines the slopes to be used in the lower bound constraints $\boldsymbol{\beta}^{\geq}_{i+1} = \boldsymbol{\lambda}^{\geq}_{i}\boldsymbol{v}_i$ when computing the concrete lower bound of a node in a subsequent layer using back-substitution. The analogous optimisation problem (which minimises the bounds in the subsequent layer) determines the slopes $\boldsymbol{\lambda}^{\leq}_{i}$ to be used when computing the upper bound of a node in a subsequent layer.

This concludes the derivation of the bound constrains for a ReLU node. The concrete bounds of each ReLU node $n_{i,j}$ are

$$\boldsymbol{l}_{i+1,j} = \min(\boldsymbol{\lambda}^{\geq}_{i+1,j} \cdot \boldsymbol{l}_{i,j}, \boldsymbol{\lambda}^{\leq}_{i+1,j} \cdot \boldsymbol{l}_{i,j}), \quad \boldsymbol{u}_{i+1,j} = \boldsymbol{u}_{i,j}.$$

Remark 2. Note that the optimisation problem 1 is non-convex and therefore hard to solve for large layers. Still, as we experimentally show in the next section, instead of jointly optimising the slopes of layer $i+1$ to tighten all concrete bounds in layer $i+2$, it is sufficient to consider the bounds of only a small number of nodes to efficiently and consistently (i.e., for all the networks and radii considered in our experiments) outperform the state-of-the-art. We hereafter refer to this number of nodes as the *number of optimised nodes parameter*. The selection of the nodes to be considered in the optimisation problem is carried out on the basis of the looseness of the nodes' bounds: the nodes having the looser bounds are the ones to be selected. The bounds are computed by concretising the nodes' bound constraints with the concrete bounds from layer $i+1$, i.e., $W^{(i+2)-}\boldsymbol{u}_{i+1}+W^{(i+2)+}\boldsymbol{l}_{i+1}$ for the lower bounds and $W^{(i+2)+}\boldsymbol{u}_{i+1}+W^{(i+2)-}\boldsymbol{l}_{i+1}$ for the upper bounds (note that as the bounds $\boldsymbol{l}_{i+1}$ depend on the approximation slopes of layer $i+1$, which have not been determined yet, we here use the slopes from the smallest overapproximation area heuristic).

Example 3. Consider the network from Fig. 2. We execute the computation of the upper and lower bound constraints of node $n_{2,1}$. The concrete lower and upper bounds of the input $\boldsymbol{v}_{1,1}$ to the node are -1.5 and 4, respectively. We therefore have that the upper bound constraint equals

$$\boldsymbol{\beta}^{\leq}_{2,1} = \frac{4}{4-(-1.5)}(\boldsymbol{v}_{1,1} - (-1.5)) = 0.73\boldsymbol{v}_{1,1} + 1.09.$$

We compute two lower bound constraints for the node: one constraint to be used in the back-substitution process for the computation of a concrete lower bound of a node in a subsequent layer; the other to be used for the computation of a concrete upper bound. Consider the former constraint. The constraint has the form $\boldsymbol{\beta}^{\geq}_{2,1} = \boldsymbol{\lambda}^{\geq}_{2,1} \cdot \boldsymbol{v}_{1,1}$. To determine $\boldsymbol{\lambda}^{\geq}_{2,1}$, we maximise the sum of the lower concrete bounds of nodes $n_{3,1}$ and $n_{3,2}$:

$$\begin{aligned}
&\max_{\boldsymbol{\lambda}^{\geq}_{2,1}, \boldsymbol{\lambda}^{\geq}_{2,2}} \boldsymbol{l}_{3,1} + \boldsymbol{l}_{3,2} \\
&= \max_{\boldsymbol{\lambda}^{\geq}_{2,1}, \boldsymbol{\lambda}^{\geq}_{2,2}} 0.5 \cdot \boldsymbol{v}_{2,1} + 2 \cdot \boldsymbol{v}_{2,2} - \boldsymbol{v}_{2,1} + 3 \cdot \boldsymbol{v}_{2,2} \\
&= \max_{\boldsymbol{\lambda}^{\geq}_{2,1}, \boldsymbol{\lambda}^{\geq}_{2,2}} 0.5 \cdot \boldsymbol{\lambda}^{\geq}_{2,1} \cdot \boldsymbol{v}_{1,1} + 2 \cdot \boldsymbol{\lambda}^{\geq}_{2,2} \cdot \boldsymbol{v}_{1,2} - 0.73 \cdot \boldsymbol{v}_{1,1} - 1.09 + 3 \cdot \boldsymbol{\lambda}^{\geq}_{2,2} \cdot \boldsymbol{v}_{1,2} \\
&= \max_{\boldsymbol{\lambda}^{\geq}_{2,1}, \boldsymbol{\lambda}^{\geq}_{2,2}} -0.75\boldsymbol{\lambda}^{\geq}_{2,1} - 15\boldsymbol{\lambda}^{\geq}_{2,2} - 4.01.
\end{aligned}$$

Since $0 \leq \boldsymbol{\lambda}^{\geq}_{2,1} \leq 1$, $0 \leq \boldsymbol{\lambda}^{\geq}_{2,2} \leq 1$, it follows that $\boldsymbol{\lambda}^{\geq}_{2,1} = 0$. Analogously we can determine the slope $\boldsymbol{\lambda}^{\leq}_{2,1}$ of the constraint $\boldsymbol{\beta}^{\geq}_{2,1} = \boldsymbol{\lambda}^{\leq}_{2,1} \cdot \boldsymbol{v}_{1,1}$ associated with

the computation of concrete upper bounds by minimising the sum of the upper concrete bounds of nodes $n_{3,1}$ and $n_{3,2}$: $\boldsymbol{\lambda}^{\leq}_{2,1} = 0$.

Max-pool Layers. We provide a novel treatment of max-pool layers (in the context of SIP) as follows. To derive the bound constraints of a max-pool node, we express the max-pool function as a sequence of affine transformations and ReLU layers, whose constraints can be computed as above. We begin with expressing the multivariate maximum function as a composition of maximum functions of two variables:

$$\max(v_1, v_2, \ldots, v_{n-1}, v_n) = \max(\ldots \max(\max(\max(v_1, v_2), v_3), v_4) \ldots, v_n)$$

Then we use that $max(v_1, v_2) = max(v_1 - v_2, 0) + v_2$ to obtain

$$\begin{aligned}
&\max(v_1, v_2, \ldots, v_{n-1}, v_n) \\
&= \max(\ldots \max(\max(\max(v_1 - v_2, 0) + v_2 - v_3, 0) + v_3 - v_4, 0) + v_4 \\
&\ldots - v_n, 0) + v_n \\
&= \mathsf{ReLU}(\ldots \mathsf{ReLU}(\mathsf{ReLU}(\mathsf{ReLU}(v_1 - v_2) + v_2 - v_3) + v_3 - v_4) + v_4 \ldots - v_n) + v_n,
\end{aligned}$$

which is a sequence of affine and ReLU transformations.

Note that this symbolic treatment of max-pools differs from [26], where the upper bound constraints are *concretised* to equal the concrete upper bounds, thereby potentially leading to bigger overapproximations. Also note that our symbolic treatment comes at the cost of computing bound constraints for the affine transformation and ReLU layers that compose the max-pool one.

Summary. Having concluded the description of the various approximations, Algorithm 2 summarises the overall algorithm that computes $\hat{\mathcal{R}}$ from Algorithm 1. Algorithm 2 and Algorithm 1 can therefore be combined to solve a verification query.

4 Implementation and Evaluation

In this section we evaluate OSIP, the verification procedure introduced in the previous section, and present comparisons with different approximations of the ReLU function and with Eran [26], a state-of-the-art SIP-based tool. OSIP is implemented in Python 3.7 on top of Venus, a MILP-based, complete tool with several optimisations including dependency analysis [4,19]. The experiments were carried out on an Intel Core i9-10920X (12 cores) equipped with 128GB RAM, running Linux kernel 5.4.

Comparison with Different ReLU Aapproximations. We compare OSIP with the zero [26,37], identity [26,37] and parallel [31,33] approximations. We also compare OSIP with the *Min_Area* [26,37] heuristic which selects the approximation with the smallest over-approximation area for each ReLU node. The comparisons are drawn with respect to the tightness of the bounds of the output nodes, which is a key aspect to (i) determine the ability of a method to

Algorithm 2. OSIP

procedure APPROXIMATION OF OUTPUT BOUNDS($\boldsymbol{f}, \boldsymbol{l}, \boldsymbol{u}$)
  Input: network $\boldsymbol{f}$, vectors of input lower and upper bounds $\boldsymbol{l}$ and $\boldsymbol{u}$
  Output: vectors of output lower and upper bounds
  $\boldsymbol{f'} \leftarrow$ replace each max-pool in $\boldsymbol{f}$ as a composition of affine and ReLU transformations
  $\boldsymbol{\beta}_0^{\geq,l}, \boldsymbol{\beta}_0^{\geq,u} \leftarrow \boldsymbol{l}$, $\boldsymbol{l}_0 \leftarrow \boldsymbol{l}$
  $\boldsymbol{\beta}_0^{\leq} \leftarrow \boldsymbol{u}$, $\boldsymbol{u}_0 \leftarrow \boldsymbol{u}$
  for each layer $\boldsymbol{f}_i'$ in $\boldsymbol{f'}$ **do**
    if $\boldsymbol{f}_i'$ is an affine transformation layer **then**
      $\boldsymbol{\beta}_i^{\geq,l}, \boldsymbol{\beta}_i^{\geq,u} \leftarrow \boldsymbol{W}^{(i)}\boldsymbol{v}_{i-1}$, $\boldsymbol{\beta}_i^{\leq} \leftarrow \boldsymbol{W}^{(i)}\boldsymbol{v}_{i-1}$
      $\boldsymbol{l}_i \leftarrow \boldsymbol{W}^{(i)}$, $\boldsymbol{u}_i \leftarrow \boldsymbol{W}^{(i)}$
      for $j \leftarrow i$ to 1 **do**
        $\boldsymbol{l}_i \leftarrow \boldsymbol{l}_i^{-}\boldsymbol{\beta}_{i-1}^{\leq} + \boldsymbol{l}_i^{+}\boldsymbol{\beta}_{i-1}^{\geq,l}$, $\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i^{-}\boldsymbol{\beta}_{i-1}^{\geq,u} + \boldsymbol{u}_i^{+}\boldsymbol{\beta}_{i-1}^{\leq}$
    else if $\boldsymbol{f}_i'$ is a ReLU layer **then**
      for each neuron j in the layer **do**
        $\boldsymbol{\beta}_{i,j}^{\leq} \leftarrow \frac{\boldsymbol{u}_{i,j}}{\boldsymbol{u}_{i,j}-\boldsymbol{l}_{i,j}}(\boldsymbol{v}_{i,j} - \boldsymbol{l}_{i,j})$
        $\boldsymbol{\lambda}_{i,j}^{\geq} \leftarrow$ solution to optimisation problem 1
        $\boldsymbol{\lambda}_{i,j}^{\leq} \leftarrow$ solution to the analogous minimisation problem of 1
        $\boldsymbol{\beta}_{i,j}^{\geq,l} \leftarrow \boldsymbol{\lambda}_{i,j}^{\geq} \cdot \boldsymbol{v}_{i,j}$ $\boldsymbol{\beta}_{i,j}^{\geq,u} \leftarrow \boldsymbol{\lambda}_{i,j}^{\leq} \cdot \boldsymbol{v}_{i,j}$
        $\boldsymbol{u}_{i,j} \leftarrow \boldsymbol{u}_{i-1,j}$
        $\boldsymbol{l}_{i,j} \leftarrow \min(\boldsymbol{\lambda}_{i,j}^{\geq} \cdot \boldsymbol{l}_{i-1,j}, \boldsymbol{\lambda}_{i,j}^{\leq} \cdot \boldsymbol{l}_{i-1,j})$
  return $\boldsymbol{l}_L, \boldsymbol{u}_L$

resolve a verification query (as discussed in Sect. 3) and (ii) formulate strong mixed integer linear programming encodings towards improved scalability in complete verification [29]. We consider the following benchmarks for fully connected ReLU FFNNs from the first competition for neural network verification (VNN-COMP) [30]:

- *ACASXU* [16] is a collection of 45 ReLU FFNNs which were developed as part of an airborne collision avoidance system to advise horizontal steering decisions for unmanned aircraft. Each network has 5 inputs, 300 ReLU nodes arranged in 6 layers with 50 neurons each, and 5 outputs. We verify the networks against the safety specifications from [17]. These include four properties that are checked on all of the 45 networks and 6 properties that are checked on a single network. Overall this results in a total of 186 verification problems.
- *MNIST* [20] is a dataset comprising images of hand-written digits 0–9, each formatted as a $28 \times 28 \times 1$-pixel grayscale image.
  We use three fully connected ReLU FFNNs trained on the dataset: FC2, FC4 and FC6. The networks comprise 2, 4 and 6 layers, respectively. Each layer of each of the networks has 256 ReLU nodes. We verify the networks against the local adversarial robustness property w.r.t 25 correctly classified images and perturbation radii of 0.02 and 0.05. This results in a total of 150 verification problems.

We additionally use two convolutional networks: Conv1 and Conv3. The architecture of the networks includes two layers. The first layer has 32 filters of size 5×5, a padding of 2 and strides of 2. The second layer has 64 filters of size of 4×4, a padding of 2 and strides of 1. Conv3 has the same architecture with Conv1 but for 128 filters in the second layer. We verify the networks against the local adversarial robustness property w.r.t 100 correctly classified images. We use a perturbation radius of 0.1 for Conv1 and 0.3 for Conv3.

OSIP was run with the number of optimised nodes parameter set to 4 for all fully connected networks and to 200 for all convolutional networks.

Table 1 reports the experimental results obtained. We observe that the zero and Min_Area approximations always outperform the identity and parallel approximations. However, the comparative performance of the zero and Min_Area approximations varies with the networks and perturbation radii. For instance, the zero approximation outperforms the Min_Area one on FC2 for the 0.05 radius, whereas the Min_Area approximation outperforms the zero one on the same network for the 0.02 radius. In contrast, OSIP consistently outperforms all of the approximations on all of the networks and radii, often exhibiting less than half of the bound interval of either the zero or the Min_Area approximation.

We additionally observe that OSIP is more effective for fully connected networks than it is for convolutional ones. We conjecture that this is because the nodes in a convolutional layer are only connected to a small subset of nodes in the previous layer thereby exhibiting less sensitivity to intra-layer dependencies between the nodes.

Lastly we note that OSIP needs only small values for the number of optimised nodes parameter to outperform all of the approximations. Indeed, as we can observe from Fig. 4, which shows the average bound interval of the output nodes computed by OSIP on FC6 as a function of the parameter, said interval initially decreases rapidly before having a more gradual decrement with larger values of the parameter.

Comparison on VGG16. We now proceed to evaluate OSIP on a variant of the VGG16 model [25] that forms a key component of the Multi-View 3D Detector (MV3D) [5], a high-accuracy 3D object detection network for autonomous driving. The model we produced was trained on the GTSRB dataset [27].

The model comprises the sequence of layers $c(32,3,3)$, $c(32,3,3)$, $p(2,2)$, $c(64,3,3)$, $c(64,3,3)$, $p(2,2)$, $c(128,3,3)$, $c(128,3,3)$, $c(128,3,3)$, $p(2,2)$, $c(128,3,3)$, $c(128,3,3)$, $c(128,3,3)$, 43, where $c(\alpha,\beta,\gamma)$ denotes a convolutional layer with output channel α, kernel width β and kernel height γ (padding and strides equal 1 for all convolutional layers), $p(\alpha,\beta)$ denotes a pooling layer with pooling width α and pooling height β, and 43 denotes a fully connected layer of 43 nodes. In total the network has 290304 ReLUs. We chose to run this experiment to evaluate the performance of OSIP on large perception systems that are closer in size to industrial applications.

Table 1. Experimental results comparing the ReLU approximations from Sect. 3. The *ver* columns report the number of images that were verified, the *time* column reports the average times, and the *range* column reports the average range of the bounds of the output nodes. Highlighted cells denote the approximation that generated the tightest bounds. The zero, identity, parallel and Min_Area approximations have equal average times.

Model	Radius	OSIP			Zero			Identity		Parallel		Min Area	
		ver	time	range	ver	time	range	ver	range	ver	range	ver	range
FC2	0.02	15	12.24	0.21	13	0.06	0.44	1	15.18	13	0.58	14	0.29
	0.05	0	15.89	2.98	0	0.05	3.33	0	52.30	0	12.53	0	6.61
FC4	0.02	21	10.4	0.32	17	0.10	0.81	3	1.15K	17	4.31	21	0.99
	0.05	0	24.76	182.3	0	0.10	310.6	0	20K	0	16.6K	0	251.87
FC6	0.02	17	14.89	159.71	8	0.15	243.53	0	245K	15	1.5K	18	161.21
	0.05	0	29.96	13.2K	0	0.15	13.2K	0	4.29e6	0	19.3K	0	14.7K
Conv1	0.1	88	11.76	10.83	72	5.27	16.05	16	32.60	81	13.80	89	11.62
Conv3	0.3	5	16.65	13.66	0	7.43	28.12	0	62.10	0	40.89	4	14.15
ACASXU	-	3	0.69	899.73	3	0.00	1.02K	0	38K	2	4.6K	10	1.9K

We verified the local robustness of the VGG16 for perturbation radiuses of 0.001, 0.0015, and 0.002 against 10 correctly classified images from the GTSRB dataset. We compare OSIP with Eran [26], a tool for the verification of feed-forward neural networks whose DeepPoly domain (which is SIP with the Min_Area approximation) presents the state-of-the art in bound propagation-based methods. We refer to [30] for more tools and details.

OSIP was run with the number of optimised nodes parameter set to 200. Eran was run using the DeepPoly domain. Each verification problem was run with a timeout of two days. Table 2 reports the experimental results obtained. We observe that OSIP provides tighter bounds than Eran, ranging from three times tighter bounds for the smallest perturbation radius to progressively tighter bounds for the larger perturbation radiuses. As previously discussed, this directly impacts the number of verification queries that each tool is able to solve. Both tools were able to resolve some of the queries for the smallest perturbation radius (with OSIP resolving 9 of them and Eran 5 of them). In our experiments only OSIP was able to resolve some of the queries (5 of them) for the intermediate radius; none of the tools was able to resolve any query for the largest perturbation radius. As far as we are aware these are the first documented successful verification results for large and complex perception systems such as VGG16 and for perturbation radiuses of 10^{-3}. We note that additional experiments on the *refinepoly* domain of Eran, which is DeepPoly enhanced with optimisation-based bound tightening methods, were not concluded after the timeout of two days. Also, experiments with NNV [14], a set-based verification tool for neural networks, were not concluded because of memory errors.

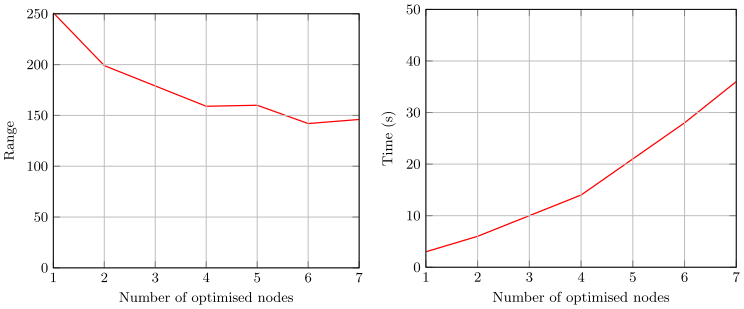

Fig. 4. Average bound interval of the output nodes and average runtime of OSIP as a function of the number of optimised nodes for the FC6 network and the 0.02 perturbation radius.

Table 2. Experimental results obtained on VGG16. The *ver* columns report the number of images that were verified, the *time* column reports the average times, and the *range* column reports the average range of the bounds of the output nodes.

Radius	**OSIP**			**Eran** (Deeppoly)		
	ver (#)	time (s)	range	ver (#)	time (s)	range
0.0010	9	66811	0.0179	5	9605	0.0460
0.0015	7	66889	0.0623	0	9718	0.4921
0.0020	0	66642	0.2915	0	10040	18.5823

The bound tightness exhibited by OSIP comes at the cost of the tool being slower (approximately 6.5 times) than Eran. This is mainly to be attributed to the handling of the max-pooling layers where additional layers are introduced to the analysis chain of OSIP.

5 Conclusions

In this paper we analysed the problem of obtaining tight bounds for the verification of feed-forward neural networks. As we observed, present state-of-the-art incomplete tools may often be unable to determine the result of a verification query for large and deep models, such as those in object classifiers, due to the compounding errors in the bound estimations.

We presented OSIP, a novel verification method based on symbolic interval propagation. OSIP provides tighter approximations than the present SoA approximations of a single univariate ReLU function in most commonly accepted benchmarks. This is obtained by determining the choice of the ReLU approximation at each node via optimisation.

We additionally benchmarked OSIP against Eran, a state-of-the-art symbolic interval propagation tool. To assess their performance in a setting close to industrial applications we carried out experiments on a variant VGG16, the largest

component of the MV3D object detector and classifier for autonomous vehicles. This is a convolutional neural network consisting of approximately 300,000 ReLU nodes. In our benchmarks `OSIP` obtained bounds that were at times two orders of magnitude smaller than Eran.

Tighter bounds are directly linked to an increased ability to reduce the number of unknowns in the verification queries. This was confirmed in our experiments in which we documented cases in which `OSIP` was the only method capable of solving the verification query. In summary, to the best of our knowledge, at present `OSIP` constitutes the most performing tool for the verification of VGG16.

Acknowledgements. The authors acknowledge support from the Audi Verifiable AI project and by BMWi under the KARLI project (grant 19A21031C).

References

1. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Math. Progr. **183**(1), 3–39 (2020). https://doi.org/10.1007/s10107-020-01474-5
2. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A.V., Criminisi, A.: Measuring neural net robustness with constraints. In: Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS16), pp. 2613–2621 (2016)
3. Battern, B., Kouvaros, P., Lomuscio, A., Zheng, Y.: Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In: International Joint Conference on Artificial Intelligence (IJCAI21), pp. 2184–2190. ijcai.org (2021)
4. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of neural networks via dependency analysis. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI20). AAAI Press (2020)
5. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3D object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1907–1915 (2017)
6. Cheng, C.-H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 251–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_18
7. Dathathri, S., et al.: Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. In: NeurIPS20 (2020)
8. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI. vol. 1, p. 2 (2018)
9. Ehlers, R.: In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
10. Fazlyab, M., Morari, M., Pappas, G.J.: Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming (2019). arXiv preprint arXiv:1903.01287
11. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**(3), 296–309 (2018). https://doi.org/10.1007/s10601-018-9285-6

12. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). arXiv preprint arXiv:1412.6572
13. Henriksen, P., Lomuscio, A.: DEEPSPLIT: an efficient splitting method for neural network verification via indirect effect analysis. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21), pp. 2549–2555. ijcai.org (2021)
14. Tran, H.-D., et al.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 3–17. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_1
15. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
16. Julian, K., Lopez, J., Brush, J., Owen, M., Kochenderfer, M.: Policy compression for aircraft collision avoidance systems. In: DASC16, pp. 1–10 (2016)
17. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
18. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Proceedings of the 31st International Conference on Computer Aided Verification (CAV19), pp. 443–452 (2019)
19. Kouvaros, P., Lomuscio, A.: Towards scalable complete verification of relu neural networks via dependency-based branching. In: International Joint Conference on Artificial Intelligence (IJCAI21), pp. 2643–2650. ijcai.org (2021)
20. LeCun, Y., Cortes, C., Burges, C.J.: The MNIST database of handwritten digits (1998)
21. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. CoRR abs/1706.07351 (2017)
22. Henriksen, P., Lomuscio, A.: Efficient neural network verification via adaptive refinement and adversarial search. In: ECAI20 (2020)
23. Raghunathan, A., Steinhardt, J., Liang, P.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Advances in Neural Information Processing Systems 31, pp. 10877–10887. Curran Associates, Inc. (2018)
24. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Advances in Neural Information Processing Systems 32, pp. 9835–9846. Curran Associates, Inc. (2019)
25. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint arXiv:1409.1556
26. Singh, G., Gehr, T., Püschel, M., Vechev, P.: An abstract domain for certifying neural networks. In: Proceedings of the ACM on Programming Languages 3(POPL), 41 (2019)
27. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The german traffic sign recognition benchmark: a multi-class classification competition. In: The 2011 International Joint Conference on Neural Networks, pp. 1453–1460. IEEE (2011)
28. Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., Patel, K., Vielma, J.: The convex relaxation barrier, revisited: tightened single-neuron relaxations for neural network verification. In: NeurIPS20 (2020)
29. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proceedings of the 7th International Conference on Learning Representations (ICLR19) (2019)

30. VNN-COMP: Vefication of neural networks competition (2020). https://sites.google.com/view/vnn20/vnncomp
31. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS18), pp. 6369–6379 (2018)
32. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proceedings of the 27th USENIX Security Symposium, (USENIX18), pp. 1599–1614 (2018)
33. Weng, T., et al.: Towards fast computation of certified robustness for relu networks (2018). arXiv preprint arXiv:1804.09699
34. Wong, E., Kolter, J.: Provable defenses against adversarial examples via the convex outer adversarial polytope (2017). arXiv preprint arXiv:1711.00851
35. Wong, E., Schmidt, F., Metzen, J., Kolter, J.: Scaling provable adversarial defenses. In: Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS18) (2018)
36. Xiang, W., Tran, H., Johnson, T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5777–5783 (2018)
37. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS2018), pp. 4944–4953. Curran Associates, Inc. (2018)