Neural Network Verification Using Partial Multi-Neuron Relaxation

Ido Shmuel(✉) and Guy Katz

The Hebrew University of Jerusalem, Jerusalem, Israel
{ido.shmuel,g.katz}@mail.huji.ac.il

Abstract. The increasing integration of deep neural networks in critical systems has spawned a theoretical and practical interest in formally guaranteeing safety properties about their behavior. To achieve this, contemporary verification algorithms rely on computing linear relaxations for a network's non-linear activation functions. Existing approaches for linear relaxations typically fall into one of two categories: single-neuron relaxation, in which each activation neuron is bounded in terms of its sources; and multi-neuron relaxation, in which linear bounds involving multiple activation neurons and their sources are calculated. However, existing methods might fail to balance tightness and scalability, as single-neuron bounds might not derive sufficiently tight bounds necessary for verification to complete, whereas generating multi-neuron relaxation for all activation neurons is computationally expensive. In this paper, we present a middle-ground approach featuring *partial multi-neuron relaxation*, in which we generate multi-neuron bounds for only a small, heuristically selected subset of neurons. To achieve this, we build upon existing branching heuristics for selecting neurons and for optimizing bounding hyper-planes for multi-neuron bounds. We integrated our proposed method within the Marabou verifier, and obtained favorable results in comparison to existing bound tightening methods. Our experiments showcase the potential of our technique for neural network verification.

1 Introduction

Deep neural networks [14] (DNNs) are increasingly being adopted as key components in mission-critical systems. They have been achieving unprecedented performance in diverse domains such as natural language processing [12], protein structure prediction [21,29], image recognition [40], medical analysis [6], aircraft collision avoidance [22], self-driving [3], and task scheduling [34], often greatly improving upon the results of algorithms crafted by human experts.

However, despite their immense success, the opacity of DNNs raises new concerns regarding their stability and reliability. Unlike classical, human-crafted algorithms, DNNs are perceived as black-boxes, making it troublesome to correctly reason about their decision making [5]. Moreover, DNNs are known to be susceptible to adversarial perturbations [7,13,43], where low-magnitude changes

G. Avni and C. Schilling (Eds.): SAIV 2026, LNCS 16831, pp. 122–144, 2027.
https://doi.org/10.1007/978-3-032-32357-6_6

to their inputs result in incorrect outputs. The absence of a formal proof of correctness of DNNs might raise doubts about the robustness of existing applications of DNNs and potentially slow down their adoption.

In order to address these issues, diverse strategies have been suggested to feasibly solve the problem of formal verification of DNNs [28]. Despite recent advances [25], these methods have limited scalability, in light of the fact DNN verification has been proven to be NP-complete for DNNs with piecewise-linear activation functions [23].

A major component in existing methods for DNN verification is the branch-and-bound (BaB) paradigm [4]. Due to the non-linear nature of activation functions, DNN verification often necessitates performing recursive case splitting, resulting in a large search space whose size grows exponentially as a function of the number of activation neurons. Under the BaB paradigm, prior to splitting the original verification problem to sub-problems, verifiers take advantage of bound tightening tactics to deduce upper and lower bounds on the values of the networks' neurons. This often results in a major size reduction of the search space, facilitating verification.

In order to derive bounds on a DNN's neurons, solvers leverage linear relaxations of a network's non-linear activation functions. By calculating linear over-approximations on activation neurons as a function of their sources, solvers may leverage technique featuring symbolic bound propagation [42,47] and Linear Programming (LP) [36,46] to quickly gather bounds on the network's neurons. Existing methods typically fall into one of two categories: single-neuron relaxation [42,56], in which each of the calculated linear over-approximations involve a single activation neuron; and multi-neuron relaxation, which features linear bounds consisting of multiple activation neurons [11,26,32,41]. Although the former strategy is highly scalable, it might result in insufficiently tight over-approximations, due to the convex relaxation barrier [36]. In contrast, the latter approach results in tighter bounds while incurring a higher computational toll.

In this paper, we present a novel approach to bound tightening, which seeks a better balance between scalability and tightness. We propose a general framework for performing *partial multi-neuron relaxation* (PMNR), namely, generating multi-neuron over- approximations for only a heuristically selected subset of neurons, while using tunable single-neuron bounds for the remainder of the network. Our approach can be instantiated with various heuristics for selecting neurons and hyper-planes for multi-neuron relaxations, and we demonstrate that multiple existing branching heuristics can be used for this purpose. Whereas in the BaB paradigm, branching heuristics indicate a single neuron whose bounds may be improved by first performing a case-split, in our approach we use these heuristics to identify cases where it is beneficial to infer a multi-neuron bound.

We implemented PMNR within the popular Marabou verification tool [24, 51]. We evaluated our implementation on local robustness queries with fully connected DNNs, trained on the MNIST dataset [27]; and compared it to other tightening algorithms implemented in Marabou. We discovered that the PMNR-enhanced Marabou solved 49% more queries than base Marabou, with a runtime

reduction of 17% on our benchmarks. Our experiments demonstrate the significant potential of the PMNR approach for bound tightening. Our implementation is available online [38].

The rest of the paper is organized as follows. In Sect. 2 we provide the necessary background on DNNs, their verification, and linear relaxations of activation functions. Next, in Sect. 3 we describe our general technique for verifying DNNs using partial multi-neuron relaxations, before instantiating it with specific heuristics in Sect. 4. Next, we evaluate the performance of our approach in Sect. 5. We cover related work in Sect. 6, and conclude with ideas for future research in Sect. 7.

2 Background

2.1 Deep Neural Networks and Their Verification

Deep Neural Networks. A deep neural network (DNN) $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ contains an input layer, $L-1$ hidden layers, and an output layer. The i^{th} layer of the network is a real-valued vector of size n_i. We denote the i^{th} layer as $\hat{\boldsymbol{x}}^{(i)}$ and its j^{th} neuron as $\hat{x}_j^{(i)}$. For $0 < i < L$, the DNN's hidden layers are iteratively computed given the recursion formula $\hat{\boldsymbol{x}}^{(i)} = \sigma_i(\boldsymbol{W}^{(i)}\hat{\boldsymbol{x}}^{(i-1)} + \boldsymbol{b}^{(i)})$, and the output layer is defined as $\boldsymbol{x}^{(L)} = \boldsymbol{W}^{(L)}\hat{\boldsymbol{x}}^{(L-1)} + \boldsymbol{b}^{(L)}$. For each i, $\boldsymbol{W}^{(i)}$ is a weight matrix, $\boldsymbol{b}^{(i)}$ is a bias vector, and $\sigma_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_i}$ is a (usually non-linear) activation function. Note that N's output $N(\hat{\boldsymbol{x}}^{(0)}) = \boldsymbol{x}^{(L)}$ does not contain post-activation values, which could be modeled as setting σ_L to be the identity function. We also define pre-activation values with the relation $\boldsymbol{x}^{(i)} = \boldsymbol{W}^{(i)}\hat{\boldsymbol{x}}^{(i-1)} + \boldsymbol{b}^{(i)}$. Since the activations σ_i are modeled as arbitrary multivariate functions, our formulation generalizes to several common DNN architectures (e.g. fully-connected, convolutional, residual) equipped with arbitrary activation functions.

Example Network. Consider the DNN in Fig. 1 with input $\hat{\boldsymbol{x}}^{(0)} \in [-1, 1]^2$, one hidden layer containing $\text{Abs}(x) = |x|$ (absolute value) activations with pre-activation $\boldsymbol{x}^{(1)}$ and post-activation $\hat{\boldsymbol{x}}^{(1)}$, another hidden layer containing two $\text{ReLU}(x) = \max(0, x)$ activations with pre-activation $\boldsymbol{x}^{(2)}$ and post-activation $\hat{\boldsymbol{x}}^{(2)}$, and a single output $x_0^{(3)}$. Neurons $\hat{x}_0^{(1)}, \hat{x}_1^{(2)}, \hat{x}_0^{(3)}$ have bias values of $1, 2, 26.1$ respectively, and all other neurons have a bias of 0. The piecewise-linear functions Abs and ReLU are applied element-wise for multi-dimensional inputs. The network's neurons are computed recursively:

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} 1 & 0 \\ 2 & -3 \\ 0 & 1 \end{bmatrix} \hat{\boldsymbol{x}}^{(0)} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \hat{\boldsymbol{x}}^{(1)} = \text{Abs}(\boldsymbol{x}^{(1)}).$$

$$\boldsymbol{x}^{(2)} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & -5 \end{bmatrix} \hat{\boldsymbol{x}}^{(1)} + \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \quad \hat{\boldsymbol{x}}^{(2)} = \text{ReLU}(\boldsymbol{x}^{(2)}).$$

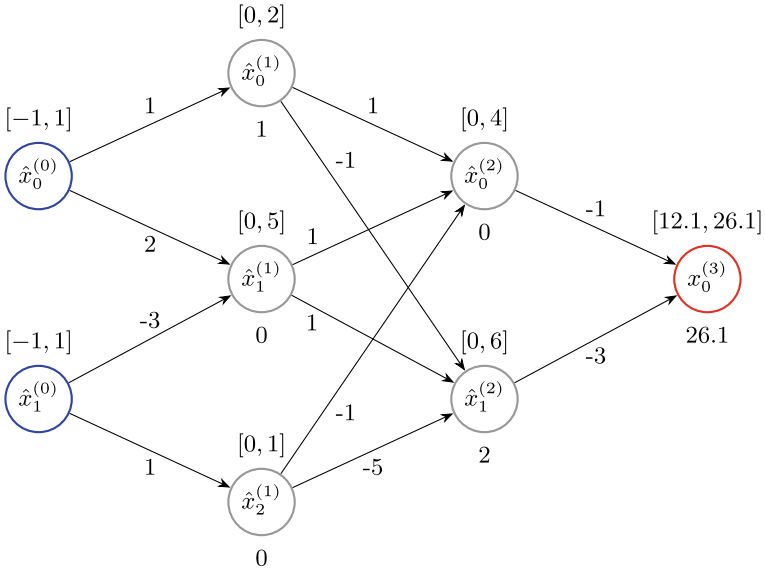

Fig. 1. An example neural network.

$$\boldsymbol{x}^{(3)} = \begin{bmatrix} -1 & -3 \end{bmatrix} \hat{\boldsymbol{x}}^{(2)} + 26.$$

Given input $\hat{\boldsymbol{x}}^{(0)} = (0.4, -0.6)^\top$, N's hidden neurons have these values:

$$\hat{\boldsymbol{x}}^{(1)} = (\text{Abs}(1.4), \text{Abs}(2.6), \text{Abs}(-0.6))^\top = (1.4, 2.6, 0.6)^\top.$$

$$\hat{\boldsymbol{x}}^{(2)} = (\text{ReLU}(3.4), \text{ReLU}(0.2))^\top = (3.4, 0.2)^\top.$$

The output of the network is $N(\hat{\boldsymbol{x}}^{(0)}) = \boldsymbol{x}^{(3)} = 22.1$.

Neural Network Verification. Neural network verification [20] is the process of soundly deciding whether a safety property holds in a neural network's output given known bounds on its inputs. Formally, a verification query is a triple $Q = (N, \mathcal{D}_{\texttt{in}}, \mathcal{D}_{\texttt{out}})$, consisting of a DNN N, an input domain $\mathcal{D}_{\texttt{in}} \subset \mathbb{R}^{n_0}$ and an output domain $\mathcal{D}_{\texttt{out}} \subset \mathbb{R}^{n_L}$, which typically represents an unsafe behavior of N. DNN verification is framed as a satisfiability problem: Q is satisfiable (SAT) if there exists $\boldsymbol{x}$ for which the predicate $\boldsymbol{x} \in \mathcal{D}_{\texttt{in}} \wedge N(\boldsymbol{x}) \in \mathcal{D}_{\texttt{out}}$ holds (i.e. N demonstrates undesirable behavior $\mathcal{D}_{\texttt{out}}$); otherwise, it is considered unsatisfiable (UNSAT). Any common verification query could be easily rewritten [9,19] into a query with a single-output DNN whose output domain is $\mathcal{D}_{\texttt{out}} = (0, \infty)$. We will therefore assume for the remainder of this paper that all queries are of this simplified form unless indicated otherwise.

As an example, consider the verification query composed of the DNN in Fig. 1, the input domain $[-1, 1]^2$ and the output domain $(0, \infty)$. The input $\hat{\boldsymbol{x}}^{(0)} = (0.4, -0.6)^\top$ satisfies it because $N(\hat{\boldsymbol{x}}^{(0)}) = 22.1 > 0$. Hence, $(0.4, -0.6)^\top$ is considered a satisfying assignment for this query, and no sound verifier would consider it unsatisfiable.

2.2 Branch and Bound (BaB)

Branch-and-bound (BaB) is a key technique, applied by various neural network verifiers [4,11,48]. It is formed by interleaving calls to a bound tightening algorithm which calculates concrete bounds $\hat{\boldsymbol{\ell}}^{(i)} \leq \hat{\boldsymbol{x}}^{(i)} \leq \hat{\boldsymbol{u}}^{(i)}, \quad \boldsymbol{\ell}^{(i)} \leq \boldsymbol{x}^{(i)} \leq \boldsymbol{u}^{(i)}$ for a DNN's neurons, and a branching method which recursively splits the verification problem into smaller, easier-to-solve sub-problems, giving rise to a search tree of an exponentially increasing size. The original problem is determined to be satisfiable if and only if at least one sub-problem is declared `SAT` by the verifier. Bound tightening is invoked post-splitting to limit the growth rate of the search tree. In order to generate sub-problems, verifiers commonly use case splitting on activation neurons [48]. Case splitting is typically applied for piecewise-linear activation neurons, which are then split into a collection of linear constraints, one for each linear segment; though it has been applied successfully for general activation functions as well [37]. Given the massive scale of branching trees in practice, verifiers heavily leverage heuristics and other techniques to prune infeasible sub-problems and limit the number of sub-problems that are generated as a result of case splitting.

2.3 Single-Neuron Relaxation

Linear Over-Approximations. Contemporary bound tightening algorithms reason about a DNN's general non-linearities by replacing them with linear over-approximations [28,42,56]. Namely, a DNN's sound single-neuron relaxation is a collection of linear bounds of the form:

$$\boldsymbol{W_\ell}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{b_\ell}^{(i)} \leq \sigma_i(\boldsymbol{x}^{(i)}) = \hat{\boldsymbol{x}}^{(i)} \leq \boldsymbol{W_u}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{b_u}^{(i)}$$

These bounds are *sound* if they hold whenever $\boldsymbol{\ell}^{(i)} \leq \boldsymbol{x}^{(i)} \leq \boldsymbol{u}^{(i)}$. The symbolic weight matrices $\boldsymbol{W_\ell}^{(i)}, \boldsymbol{W_u}^{(i)}$ and symbolic bias vectors $\boldsymbol{b_\ell}^{(i)}, \boldsymbol{b_u}^{(i)}$ are chosen as a function of $\sigma_i, \boldsymbol{\ell}^{(i)}, \boldsymbol{u}^{(i)}$ to guarantee soundness using known methods [33,42,52]. They may depend on an external, tunable parameter $\boldsymbol{\alpha}$ [52].

For example, consider again the ReLU, Abs activation functions from the network in Fig. 1, which are piecewise-linear with two linear phases. Given known bounds $x_j^{(i)} \in [\ell_j^{(i)}, u_j^{(i)}]$, sound linear bounds on $\mathrm{ReLU}(x_j^{(i)}), \mathrm{Abs}(x_j^{(i)})$ are:

$$\begin{cases} 0 \leq \mathrm{ReLU}(x_j^{(i)}) \leq 0 & \text{if } u_j^{(i)} \leq 0 \\ x_j^{(i)} \leq \mathrm{ReLU}(x_j^{(i)}) \leq x_j^{(i)} & \text{if } \ell_j^{(i)} \geq 0 \\ \alpha x_j^{(i)} \leq \mathrm{ReLU}(x_j^{(i)}) \leq \frac{u_j^{(i)}(x_j^{(i)} - \ell_j^{(i)})}{u_j^{(i)} + \ell_j^{(i)}} & \text{otherwise, for any } \alpha \in [0,1] \end{cases}$$

$$\begin{cases} -x_j^{(i)} \leq \mathrm{Abs}(x_j^{(i)}) \leq -x_j^{(i)} & \text{if } u_j^{(i)} \leq 0 \\ x_j^{(i)} \leq \mathrm{Abs}(x_j^{(i)}) \leq x_j^{(i)} & \text{if } \ell_j^{(i)} \geq 0 \\ \alpha x_j^{(i)} \leq \mathrm{Abs}(x_j^{(i)}) \leq \max(\ell_j^{(i)}, u_j^{(i)}) & \text{otherwise, for any } \alpha \in [-1,1] \end{cases}$$

In the first two cases of the inequalities above, the two activation functions are known to have a fixed phase in the domain $x_j^{(i)} \in [\ell_j^{(i)}, u_j^{(i)}]$, and their linear bounds are identical to their respective linear phases. Else, they are said to have an unfixed phase in the given domain, and they are relaxed into a pair of linear constraints. The linear relaxations in the unfixed case are illustrated in Fig. 2.

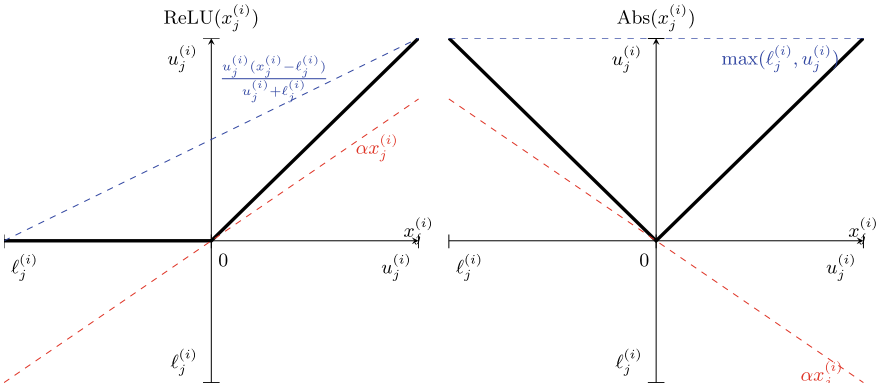

Fig. 2. An illustration of linear bounds on the ReLU and Abs functions over $x \in [\ell_j^{(i)}, u_j^{(i)}]$ where $\ell_j^{(i)} < 0 < u_j^{(i)}$. $\alpha x_j^{(i)}$ is a sound linear lower bound on ReLU($x_j^{(i)}$) for $\alpha \in [0, 1]$, and it is a sound linear lower bound on Abs($x_j^{(i)}$) for $\alpha \in [-1, 1]$.

Symbolic Bound Tightening (SBT). Symbolic Bound Tightening is a common, light-weight tightening method requiring linear over-approximations on a DNN's non-linear activations. In this method, linear bounds of every neuron in the network as a function the previous layer's neurons are computed iteratively using back-substitution and concretization. Two noteworthy variations of SBT are Symbolic Intervals [47] and DeepPoly [42].

Linear Programming (LP). Alternative approaches to bound tightening reduce the DNN verification query Q to a linear program. The existence a of positive solution to the following LP establishes the satisfiability of Q:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{x} \in \mathcal{D}_{\text{in}}} \quad & N(\boldsymbol{x}) = \hat{\boldsymbol{x}}^{(L)} \\ s.t. \quad & \boldsymbol{x}^{(i)} = \boldsymbol{W}^{(i)} \hat{\boldsymbol{x}}^{(i-1)} + \boldsymbol{b}^{(i)} \\ & \hat{\boldsymbol{x}}^{(i)} \geq \boldsymbol{W_\ell}^{(i)} \boldsymbol{x}^{(i)} + \boldsymbol{b_\ell}^{(i)} \\ & \hat{\boldsymbol{x}}^{(i)} \leq \boldsymbol{W_u}^{(i)} \boldsymbol{x}^{(i)} + \boldsymbol{b_u}^{(i)} \end{aligned}$$

This LP is typically dispatched by an LP solver or duality-based strategies [8, 36]. Alternately, the problem might be transformed into a Mixed-Integer LP (MILP) instance, and then solved by invoking a MILP solver [45].

2.4 Multi-neuron Relaxation

Although single-neuron relaxation allows for scalable bound tightening, its precision is inherently limited by the convex relaxation barrier, even for simple ReLU activations [36]. A multi-neuron bound is a linear bound involving multiple activations and their associated pre-activation values. Formally, given activation neurons $\texttt{Neurons} = \{\hat{x}_{j_1}^{(i_1)}, \ldots \hat{x}_{j_d}^{(i_d)}\}$, multi-neuron linear bounds are a bounding polyhedron of the form $\sum_{k=1}^{d}(\boldsymbol{c}^{(k)}\hat{x}_{j_k}^{(i_k)} - \boldsymbol{C}^{(k)}\boldsymbol{x}^{(i_k)}) \leq \boldsymbol{d}$. The technique of multi-neuron relaxation bypasses the convex barrier by incorporating multi-neuron linear bounds. Some existing verification tools apply this technique, and are able to learn stronger bounds by calculating multi-neuron bounds for all neurons [11,32,41].

3 Partial Multi-neuron Relaxation Paradigm

While the technique of Multi-Neuron relaxation produces tighter bounds compared to Single-Neuron Tightening, it incurs a significant runtime overhead. Current methods [32,41] either require calculating multi-neuron bounds for every activation neuron in a given DNN, solving MILPs [55], or having already performed branching [58]. The first two might not scale well for larger networks, and the third does not apply for initial (pre-branching) bound tightening.

In order to achieve more accurate bounds compared to Single-Neuron Tightening, while avoiding the higher cost of Multi-Neuron Tightening, we propose to extend single-neuron relaxation by heuristically selecting a small subset of neurons and generating multi-neuron bounds only for these neurons. This allows to circumvent the convex relaxation barrier without needing to calculate multi-neuron bounds for all activation neurons.

Though it is difficult to know *a priori* which subset of neurons will yield the tightest bounds, we argue that existing branching heuristics might be suitable for the task of neuron selection. The motivation is that these heuristics are already designed to identify neurons for which Single-Neuron Relaxation fails to produce sufficiently tight bounds, so that branching can be performed on them. Here, instead of performing branching, we propose to tighten these neurons' bounds by incorporating them in multi-neuron bound calculation.

In this section, we introduce the concept of lemmas, and outline our Partial Multi-Neuron Relaxation (PMNR) bound tightening framework—the pseudo-code of which appears in Algorithm 1. Next, we describe each step in greater detail and prove soundness properties.

Derived Lemmas. For the purpose of defining soundness, we introduce the following definitions pertaining to lemmas inferred from a verification query or from other lemmas.

Definition 1. *The set of constraints C on $\hat{\boldsymbol{x}}^{(i)}, \boldsymbol{x}^{(i)}$ is a lemma derived from verification query Q if $\hat{\boldsymbol{x}}^{(0)} \in \mathcal{D}_{\texttt{in}} \wedge N(\hat{\boldsymbol{x}}^{(0)}) \in \mathcal{D}_{\texttt{out}}$ implies $\hat{\boldsymbol{x}}^{(i)}, \boldsymbol{x}^{(i)}$ satisfy all the constraints in C, and we denote $Q \Rightarrow C$.*

Definition 2. *For two sets C_1, C_2 of constraints on $\hat{\boldsymbol{x}}^{(i)}, \boldsymbol{x}^{(i)}$, C_2 is a lemma derived from C_1 if any $\hat{\boldsymbol{x}}^{(i)}, \boldsymbol{x}^{(i)}$ which satisfy C_1's constraints also satisfy C_2's, and we denote $C_1 \Rightarrow C_2$.*

As a corollary of Definition 1, if $Q \Rightarrow C$ and for no choice of $\hat{\boldsymbol{x}}^{(0)} \in \mathbb{R}^{n_0}$ do $\hat{\boldsymbol{x}}^{(i)}, \boldsymbol{x}^{(i)}$ satisfy C's constraints, Q is necessarily unsatisfiable.

Single-Neuron Relaxation. Our framework uses as a backend a single-neuron relaxation-based bound tightening method. Any number of existing methods can be plugged in for this purpose, and we invoke them through a call to the abstract SingleNeuronTightening method—which returns concrete bounds `ConcreteB`, single-neuron relaxation `SingleB` and a optimizable parameter $\boldsymbol{\alpha}_{\text{INITIAL}}$. Here, `SingleB` is a set of linear over-approximations (as in Subsect. 2.3) which depend on an optimizable parameter $\boldsymbol{\alpha}$, and $\texttt{SingleB}(\boldsymbol{\alpha})$ is the resulting linear over-approximations by substituting a value of $\boldsymbol{\alpha}$. Our framework requires `ConcreteB` and $\texttt{SingleB}(\boldsymbol{\alpha})$ returned by SingleNeuronTightening are lemmas learned from Q for all values of $\boldsymbol{\alpha}$.

Partial Multi-neuron Relaxation. In the case `ConcreteB` does not contain any constraint which is a contradiction (which we denote by $\perp$), we proceed to calculating multi-neuron bounds for a heuristically selected subset of neurons. The process is repeated until StopCondition becomes true or `ConcreteB` contains a contradiction $\perp$.

First, at line 6 of Algorithm 1, the optimizable parameter selection heuristic PickAlphas outputs optimizable parameters $\boldsymbol{\alpha}_{\text{SELECT}}, \boldsymbol{\alpha}_{\text{GENER}}, \boldsymbol{\alpha}_{\text{FINAL}}$ to be used in the next stages of the PMNR paradigm. Then, at line 7, SelectNeurons outputs a set of activation neurons $\texttt{Neurons} = \{\hat{x}_{j_1}^{(i_1)}, \ldots \hat{x}_{j_d}^{(i_d)}\}$. Afterwards, at line 8, GeneratePMNR returns a set of multi-neuron bounds `MultiB` which is a polyhedron $\sum_{k=1}^{d}(\boldsymbol{c}^{(k)}\hat{x}_{j_k}^{(i_k)} - \boldsymbol{C}^{(k)}\boldsymbol{x}^{(i_k)}) \leq \boldsymbol{d}$, Finally, at line 9, PostTighten returns updated bounds $\texttt{ConcreteB}'$. For soundness, our framework requires that `MultiB` is a lemma learned from $\texttt{ConcreteB} \cup \texttt{SingleB}(\boldsymbol{\alpha}_{\text{GENER}})$, $\texttt{ConcreteB}'$ is a lemma learned from $\texttt{ConcreteB} \cup \texttt{SingleB}(\boldsymbol{\alpha}_{\text{FINAL}}) \cup \texttt{MultiB}$, and it holds true that $\texttt{ConcreteB}' \Rightarrow \texttt{ConcreteB}$ (i.e. $\texttt{ConcreteB}'$ is stronger than `ConcreteB`).

Soundness. It is straightforward to prove by induction that the returned concrete bounds from Pmnr are sound and are no less precise than those inferred by SingleNeuronTightening if Pmnr's soundness requirements hold. Formally:

Theorem 1. *If* Pmnr*'s requirements hold and* $\texttt{ConcreteB}_{single}, \texttt{ConcreteB}_{\text{PMNR}}$ *are the concrete bounds yielded by* SingleNeuronTightening *and Algorithm 1 respectively, then* $Q \Rightarrow \texttt{ConcreteB}_{\text{PMNR}}$ *and* $\texttt{ConcreteB}_{\text{PMNR}} \Rightarrow \texttt{ConcreteB}_{single}$.

4 Instantiating PMNR

In Sect. 3, we described our approach for performing partial multi-neuron tightening and proved soundness properties. Here, we list specific heuristics we used to instantiate the PMNR paradigm in our experiments. Our suggested heuristics

Algorithm 1. PMNR($N, \mathcal{D}_{\texttt{in}}, \mathcal{D}_{\texttt{out}}$)

while ¬StopCondition() **do**
  ConcreteB, SingleB, $\boldsymbol{\alpha}_{\text{INITIAL}} \leftarrow$ SingleNeuronTightening($N, \mathcal{D}_{\texttt{in}}, \mathcal{D}_{\texttt{out}}$)
  if $\bot \in$ ConcreteB **then**
    break
  end if
  $\boldsymbol{\alpha}_{\text{SELECT}}, \boldsymbol{\alpha}_{\text{GENER}}, \boldsymbol{\alpha}_{\text{FINAL}} \leftarrow$ PickAlphas(N, ConcreteB, SingleB, $\boldsymbol{\alpha}_{\text{INITIAL}}$)
  Neurons $\leftarrow$ SelectNeurons(N, ConcreteB, SingleB, $\boldsymbol{\alpha}_{\text{SELECT}}$)
  MultiB $\leftarrow$ GeneratePMNR(N, ConcreteB, SingleB, $\boldsymbol{\alpha}_{\text{GENER}}$, Neurons)
  ConcreteB $\leftarrow$ PostTighten($N, \mathcal{D}_{\texttt{in}}, \mathcal{D}_{\texttt{out}}$, ConcreteB, SingleB, $\boldsymbol{\alpha}_{\text{FINAL}}$, MultiB)
end while
if $\bot \in$ ConcreteB **then**
  break
end if
return ConcreteB

build on contemporary branching heuristics and bound tightening algorithms that tolerate general non-linearities, and are therefore generally applicable to multiple kinds of DNNs and activations.

4.1 Neuron Selection

Neuron Selection with Symbolic Expressions (NSSE). The novel NSSE heuristic presented here, designed for neuron selection from DNNs with arbitrary activation functions, is reworked from the Bound Propagation with Shortcuts (BBPS) branching heuristic [37]. BBPS, which supports arbitrary non-linearities, estimates the lower bound on a DNN's output neuron for all potential branchings and selects the neuron which is projected to yield the maximal improvement. One major alteration between BBPS and NSSE is that BBPS assigns a score for every pair of neuron and possible branch, while NSSE assigns a score for every unfixed-phase neuron.

Calculating BBPS Scores. To calculate the BBPS score for the k^{th} phase of neuron $x_j^{(i)}$, the single-neuron relaxation-based tightening method is augmented to calculate linear lower over-approximations on $\boldsymbol{x}^{(L)}$ in terms of $\hat{x}_j^{(i)}$, as well as linear bounds on $\hat{x}_j^{(i)}$ in terms of $\boldsymbol{x}^{(i)}$, which are sound when $x_j^{(i)}$ is in its k^{th} phase. By substituting the concrete bounds on $\boldsymbol{x}^{(i)}$ in these linear over-approximations, a concrete linear bound for the output layer is calculated. This lower bound is defined to be the BBPS score of k^{th} phase of neuron $x_j^{(i)}$, and it serves as a cheap approximation on the post-branching lower bounds on the output. The BBPS heuristic prioritizes neurons with the highest score, in an attempt to prove the output domain $(0, \infty)$ is satisfiable.

Calculating NSSE Scores. To calculate the NSSE score of a neuron $x_j^{(i)}$, we derive linear upper and lower bounds on the output neuron $\boldsymbol{x}^{(L)}$ as a function

of a chosen source neuron $x_{j'}^{(i)}$ which are sound when $x_j^{(i)}$ is in its k^{th} phase, in a similar fashion to the calculation routine of the BBPS heuristic. Then, we separately aggregate the linear upper and lower over-approximations over all branches, producing two symbolic expressions of $x_{j'}^{(i)}$. and the post-concretization average range of these symbolic expressions is $x_j^{(i)}$'s NSSE score. Our heuristic picks the d highest-score unfixed-phase neurons from the layer ℓ with highest score-sum layer, unlike BBPS which picks the highest-score neuron and branch.

4.2 PMNR Generation

We will break down the novel *Bounding hyper-planes via Splitting and Optimization* (BHSO) paradigm for generating bounding hyper-planes, described in Algorithm 2, and apply it to PMNR generation. BHSO features elements from the Branch-and-Bound paradigm [4,37] and preimage over-approximation [26], and applies them to the problem of inferring bounding hyper-planes.

Initial Hyper-Planes. Though BHSO applies to general bounding hyper-planes $\sum_{i=0}^{L-1} \hat{\boldsymbol{C}}^{(i)} \hat{\boldsymbol{x}}^{(i)} + \sum_{i=1}^{L} \boldsymbol{C}^{(i)} \boldsymbol{x}^{(i)} \leq \boldsymbol{d}$, we focused in our evaluation on inferring multi-neuron bounds of the form (similarly to [41]):

$$\sum_{k\in[d]} \boldsymbol{\epsilon}^{(k)} \left(\hat{x}_{j_k}^{(\ell)} - \boldsymbol{W_u}_{j_k:}^{(\ell)} \boldsymbol{x}^{(\ell)} \right) \leq \boldsymbol{d}, \quad \sum_{k\in[d]} \boldsymbol{\epsilon}^{(k)} \left(\hat{x}_{j_k}^{(\ell)} - \boldsymbol{W_\ell}_{j_k:}^{(\ell)} \boldsymbol{x}^{(\ell)} \right) \leq \boldsymbol{d}.$$

To avoid re-optimizing `ConcreteB` or `SingleB` with OptimizePMNR, we limited ourselves to vectors $\boldsymbol{\epsilon} \in \{-1, 0, 1\}^d$ with more than one non-zero entry. The number of such vectors equals $3^d - 2d - 1$, which grows exponentially with d: For d values of $2, 3, 4$, the quantity of bounding hyper-planes to be generated would be 4, 20 and 72 respectively. We limit ourselves to $d \in \{2, 3\}$ selected neurons in order to ensure Pmnr remains computationally affordable within our experiments.

Optimizing Hyper-Planes. BHSO employs OptimizePMNR to refine the bias of all hyper-planes defined above. In the case of ReLU networks, the INVPROP algorithm [26] might be used to instantiate it. To support general networks, we employ a generalized version of [26, Theorem 2, Appendix C], which applies to general activations σ_i and input domains $\mathcal{D}_{\mathtt{in}}$ and allows to optimize current hyper-planes depending on previous ones. See Appendix B of the full version of this paper [39] for details on the generalized theorem, and Appendix C for proof. We utilize this theorem to optimize multi-neuron bounds sequentially given `ConcreteB`, `SingleB` and previously optimized multi-neuron bounds `MultiB` with PGD.

Optimizing Further with General Branching. BHSO incorporates branching in order to learn more precise bounds from OptimizePMNR. Like NSSE, it assumes all chosen neurons could be partitioned into several branches, for instance, via ReLU splitting or GenBaB [37]. OptimizePMNR operates several

times per hyper-plane, with each run superseding a pre-activation value's bounds with those of the current branch combinations, and substituting the neurons' linear over-approximations with the corresponding, more accurate ones. The hyper-plane's new bias $\boldsymbol{d}^{(k)}$ is the weakest bound among all branch combinations, thereby ensuring the soundness of BHSO is not impaired by the existence of infeasible branch combinations (for which, the dual problem solved using either INVPROP or Theorem 2 in the full version of this paper [39], is unbounded).

As an illustration, here are the main steps that the BHSO paradigm might perform to deduce a hyper-plane involving the ReLU neurons $\hat{x}_0^{(2)}, \hat{x}_1^{(2)}$ from the example network in Fig. 1. An initial hyper-plane would be directly derived via OptimizePMNR, leveraging the neurons' unfixed-phase linear bounds. To tighten it further, OptimizePMNR will be called four additional times, per each combination of the neurons being in their active $x_0^{(2)} \leq \hat{x}_0^{(2)} \leq x_0^{(2)}$, $x_1^{(2)} \leq \hat{x}_1^{(2)} \leq x_0^{(2)}$ or inactive phase $0 \leq \hat{x}_0^{(2)} \leq 0$, $0 \leq \hat{x}_1^{(2)} \leq 0$, while superseding the neurons' unfixed-phase linear bounds with precise per-branch bounds. Ultimately, BHSO will choose the loosest bound found.

Infeasible Branches Detection. It is possible to identify some branch combinations which are infeasible by calculating an upper bound for the bounding hyper-planes (e.g. with simple concretization) and comparing it to the bias resulted by invoking OptimizePMNR. These findings might be integrated with the next steps of the larger Branch-and-Bound paradigm, though we have not explored this direction yet.

4.3 Other Heuristics

In this subsection we describe other heuristics employed in our evaluation.

Single-Neuron Tightening and Stopping Criteria. We instantiated the method SingleNeuronTightening with DeepPoly [42], which rapidly gathers concrete bounds by propagating single-neuron bounds across a DNN via concretization and back-substitution. If the $\bot$ becomes an element of `ConcreteB` during at any point during Algorithm 1, then Pmnr terminates as the verification query Q is proven to be unsatisfiable. The stopping criterion StopCondition, which controls the execution of the main loop of Pmnr, holds when any of these conditions is met: (i) the main loop of Algorithm 1 has completed n iterations, where n is a user-defined budget parameter; or (ii) the operation of PostTighten at line 9 of Algorithm 1 has not resulted in any revision to `ConcreteB`.

Optimizable Parameters. Among the algorithm four tunable parameters, the first three $\boldsymbol{\alpha}_{\text{INITIAL}} = \boldsymbol{\alpha}_{\text{SELECT}} = \boldsymbol{\alpha}_{\text{GENER}}$ are chosen as detailed at [46], whilst $\boldsymbol{\alpha}_{\text{FINAL}}$ is defined as such: Linear over-approximations of N's output layer in terms of its input layer are produced through Symbolic Intervals [47], following which local optimization [57, Section 4.2, Appendix F.] is applied to select $\boldsymbol{\alpha}_{\text{FINAL}}$ which minimizes the volume of the input-space polytope created by them.

Algorithm 2. GeneratePMNR(N, ConcreteB, SingleB, $\boldsymbol{\alpha}_{\text{GENER}}$, Neurons)

```
BranchPoints, BranchB ← Branches(N, ConcreteB, SingleB, α_SELECT)
InfeasibleBranches ← ∅
MultiB ← ∅
m, C^(i), Ĉ^(i), d^(i), dFeasible^(i) ←
    initialPMNR(N, ConcreteB, SingleB, α_GENER, C^(i), Ĉ^(i))
for k ∈ 1, ... m do
    dOptimized^(k) ←
    OptimizePMNR(N, ConcreteB, SingleB, α_GENER, C^(i), Ĉ^(i), d^(i), MultiB)
    dBranch^(k) ← −∞
    for branchCombination in BranchCombinations(Neurons, BranchB) do
        dBranch^(k) ←
        OptimizePMNR(N, ConcreteB, SingleB, α_GENER, C^(i), Ĉ^(i), d^(i), MultiB)
        if dBranch^(k) > dFeasible^(k) then
            InfeasibleBranches ← InfeasibleBranches ∪ {branchCombination}
        end if
        dOptimized^(k) ← min(dOptimized^(k), dBranch^(k))
    end for
    d^(k) ← max(d^(k), dOptimized^(k))
    MultiB ← MultiB ∪ {Σ_{i=0}^{L−1} Ĉ^(i)_{:k} x̂^(i) + Σ_{i=1}^{L} C^(i)_{:k} x^(i) ≤ d^(k)}
end for
return MultiB
```

Final Tightening. To further tighten ConcreteB given multi-neuron bounds, PostTighten capitalizes on a modified version of the LP-based Forward- Backward Abstract Interpretation [50] framework. It consists of a forward pass, during which only the subset of linear bounds from $\mathcal{D}_{\text{in}} \cup \mathcal{D}_{\text{out}} \cup$ ConcreteB $\cup$ SingleB $\cup$ MultiB containing neurons from current or preceding layers are counted among the constraints of the LPs solved, as well as a backward pass, in which only those containing neurons from current or subsequent layers are included.

4.4 Heuristics For PMNR-ALL

For the purpose of fairly comparing PMNR instantiated with the heuristics described in previous subsections to existing Multi Neuron Relaxation approaches, we introduce another instantiation of the PMNR paradigm called PmnrAll.

It generates multi-neuron bounds for nearly all activation neurons from all layers using the same heuristics in Subsects. 4.2 and 4.3, although it differs from our main instantiation of Pmnr in regard to neuron selection. While Pmnr chooses d neurons from a single layer with the NSSE heuristic, PmnrAll selects a set containing all groups of d consecutive unfixed-phase activation neurons from all layers. Following neuron selection, both instantiations produce hyper-planes involving each group of d neurons separately, via BHSO.

Pmnr constitutes a middle-ground between SingleNeuronTightening and PmnrAll in regard to performance and tightness. Pmnr improves on

SingleNeuronTightening by producing hyper-planes only involving d heuristically selected neurons, whereas PmnrAll does so by generating multi-neuron bounds for nearly all neurons. Notably, in PmnrAll the number of hyper-planes to be calculated scales linearly in the size of the DNN, while in Pmnr it would be constant. Combined with the fact that the computational resources necessary to generate a single hyper-plane also grows with the DNN's size, it follows that the bounds discovered by PmnrAll are likely to be stronger than these deduced by Pmnr, though they are more computationally expensive to obtain.

4.5 Running Example

Here is a demonstration of Pmnr on an example query Q, featuring the network depicted in Fig. 1.

DeepPoly Fails to Verify Q. Consider once more the neural network N from Fig. 1, and domains $\mathcal{D}_{\texttt{in}} = [-1, 1]^2$ and $\mathcal{D}_{\texttt{out}} = (-\infty, 0)$. Executing DeepPoly (as the instantiation of SingleNeuronTightening) yields the bounds depicted in Appendix D of the full version of this paper [39]. DeepPoly does not manage to prove that the verification query $Q = (N, \mathcal{D}_{\texttt{in}}, \mathcal{D}_{\texttt{out}})$ is `UNSAT`, because the computed output layer's bounds $x_0^{(3)} \in [-0.15, 40.1]$ are not adequately strong. Thus, we progress to the ensuing stages of the PMNR paradigm.

Running Example Heuristics. For the running example, we employed simplified tunable parameters and neuron selection heuristics to demonstrate Pmnr. First, $\text{ReLU}(x_j^{(i)}) \geq x_j^{(i)}$ and $\text{Abs}(x_j^{(i)}) \geq 0$ are the linear lower bounds of our choice for unfixed-phase ReLU neurons and Abs neurons, respectively. Moreover, rather than computing NSSE scores for all unfixed-phase activation neurons, we use the span of each neuron's concrete bounds as its score: i.e., $\texttt{score}_j^{(i)} = u_j^{(i)} - \ell_j^{(i)}$ is the score of neuron $\hat{x}_j^{(i)}$.

Multi-neuron Relaxation Solves Q. We first demonstrate how Q is solved with PmnrAll, a Multi-Neuron Relaxation-based approach; and then proceed to show that PMNR likewise solves it while requiring less computational effort.

PMNR-ALL selects all unfixed-phase neurons in N: $\hat{x}_1^{(1)}, \hat{x}_2^{(1)}, \hat{x}_0^{(2)}, \hat{x}_1^{(2)}$. It applies multi-neuron bound generation with BHSO, and, given the linear over-approximations discovered by DeepPoly, it produces 12 multi-neuron bounds of the form $\boldsymbol{\epsilon}^{(1)}\hat{x}_0^{(1)} + \boldsymbol{\epsilon}^{(2)}\hat{x}_1^{(1)} \leq t$, $\boldsymbol{\epsilon}^{(1)}(\hat{x}_0^{(2)} - x_0^{(2)}) + \boldsymbol{\epsilon}^{(2)}(\hat{x}_1^{(2)} - x_1^{(2)}) \leq t$ and $\boldsymbol{\epsilon}^{(1)}(\hat{x}_0^{(2)} - \frac{7}{8}x_0^{(2)}) + \boldsymbol{\epsilon}^{(2)}(\hat{x}_1^{(2)} - \frac{7}{12}x_1^{(2)}) \leq t$ for $\boldsymbol{\epsilon} \in \{-1, 1\}^2$ (see Appendix D of the full version of this paper [39] for the resulting hyper-planes).

Applying PostTighten results in these concrete bounds: $\hat{\ell}_0^{(2)} \leftarrow 0$, $\hat{\ell}_1^{(2)} \leftarrow 0$, $\hat{\ell}_0^{(3)} \leftarrow 0.1$, $\hat{u}_0^{(3)} \leftarrow 26.1$. Augmenting DeepPoly with partial multi-neuron relaxation yielded the tightened output layer's bounds $x_0^{(3)} \in [0.1, 26.1]$, which, when intersected with the output domain $\mathcal{D}_{\texttt{out}} = (-\infty, 0)$, results in empty concrete bounds $Q \Rightarrow \bot$. The stronger relaxations learned by PMNR-ALL thus allowed proving that Q is `UNSAT`.

PMNR Solves Q More Quickly. The unfixed-phase neurons of N are $\hat{x}_1^{(1)}, \hat{x}_2^{(1)}, \hat{x}_0^{(2)}, \hat{x}_1^{(2)}$, with associated scores $\texttt{score}_1^{(1)} = 10$, $\texttt{score}_2^{(1)} = 2$, $\texttt{score}_0^{(2)} = 8$, $\texttt{score}_1^{(2)} = 12$. Because $\hat{\boldsymbol{x}}^{(2)}$ is the layer with greatest neuron score sum, the $d = 2$ highest-score neurons from it (namely, $\hat{x}_0^{(2)}$ and $\hat{x}_1^{(2)}$) are chosen per our BHSO paradigm. PMNR produces eight multi-neuron bounds of the form $\boldsymbol{\epsilon}^{(1)}(\hat{x}_0^{(2)} - x_0^{(2)}) + \boldsymbol{\epsilon}^{(2)}(\hat{x}_1^{(2)} - x_1^{(2)}) \leq t, \boldsymbol{\epsilon}^{(1)}(\hat{x}_0^{(2)} - \frac{7}{8}x_0^{(2)}) + \boldsymbol{\epsilon}^{(2)}(\hat{x}_1^{(2)} - \frac{7}{12}x_1^{(2)}) \leq t$ for $\boldsymbol{\epsilon} \in \{-1, 1\}^2$, which are listed in Appendix D of the full version of this paper [39].

Applying PostTighten yields the same bounds computed by PMNR-ALL, proving Q is UNSAT. Thus, despite having calculated fewer hyper-planes via BHSO than PMNR-ALL, PMNR manages to solve Q. The ratio between the number of hyper-planes produced by the Multi-Neuron Relaxation-based PMNR-ALL versus PMNR scales linearly with N's size, implying that PMNR's advantage over PMNR-ALL should become even clearer for larger DNNs.

5 Experiments and Evaluation

5.1 Implementation

For our evaluation, we implemented the PMNR paradigm with the heuristics defined in Subsects. 4.1–4.3 within the SMT-based Marabou verification tool [24], and compared it to the DeepPoly [42] Symbolic Bound Tightening framework. In addition, we implemented PMNR-ALL defined in Subsect. 4.4 as a representative multi-neuron relaxation method, and compared it to our approach. See Appendices E.1 and E.2 of the full version of this paper [39] for a measurement of the impact of our proposed NSSE heuristic and proposed stopping criteria on PMNR's performance; and Appendix E.3 for a comparison of PMNR to the Single-Neuron Relaxation-based bound tightening method, Forward-Backward Abstract Interpretation [50].

We evaluated our approach on ℓ_∞-local robustness queries with fully-connected FFNNs, trained on the MNIST dataset [27], including both piecewise-linear (PL) and non-piecewise-linear (NPL) activations. Local robustness verification pertains to the robustness of a DNN to small perturbations around a given input. Formally, given a DNN N, an input x_0 and positive reals ε, δ, ℓ_∞-local robustness queries have $\mathcal{D}_{\texttt{in}} = \{x : \|x - x_0\|_\infty \leq \varepsilon\}$ and $\mathcal{D}_{\texttt{out}} = \{x : \|N(x) - N(x_0)\|_\infty \leq \delta\}$.

Marabou's SMT Solver is interleaved with calls to bound tightening procedures (by default, DeepPoly) [51]. In our implementation, the initial bound tightening algorithm is replaced by our implementation of PMNR or PMNR-ALL, while DeepPoly is called for subsequent bound tightening. We have opted to employ DeepPoly for subsequent tightening since invoking PMNR repeatedly would incur an intolerable computational toll. This experimental setup guarantees the only difference between base Marabou, PMNR-enhanced and PMNR-ALL-enhanced Marabou results from augmenting the first run of DeepPoly with PMNR or PMNR-ALL, with the aim of discovering stronger bounds. We evaluated Marabou with DeepPoly, Pmnr and PmnrAll as the initial bound tightening method on 344 local robustness queries overall, including 272 queries for PL

networks and 72 queries for NPL networks. The architectures of all the FFNNs in our evaluation are specified in Appendix A of the full version of this paper [39].

The external LP solver Gurobi [15] served to assist the PMNR paradigm's final bound tightening method PostTighten. For our experiments in this section, we set $n = 10$ for the stopping condition for PMNR and PMNR-ALL. All experiments were conducted on dual-core machines with 4GB of memory, running Debian 12, with a timeout of $T = 14100$ seconds (235 min).

5.2 FFNNs with Piecewise-Linear Activations

For our benchmarks on DNNs with piecewise-linear activations, we experimented on fully-connected networks featuring the ReLU, LeakyReLU, Max (Max Pool) [42] and Sign [1] activations.

We trained the ReluSignMax and LeakyRelu 14×28 FFNNs ourselves using the PyTorch library [44], and verified local robustness for the first image in the MNIST test set with arbitrarily selected ε values of $0.02, 0.04, 0.06, 0.08$, thereby producing 36 queries per DNN. As for the remaining two FFNNs, we used the two benchmarks denoted as $\texttt{MNIST}_1, \texttt{MNIST}_2$ in [50], which consist of verifying local robustness around the first 100 MNIST test images with $\varepsilon = 0.02$.

The results of our experiments on PL networks is summarized at Table 1. The "Verified" column represents the number of robustness queries which Marabou verified as either satisfiable or unsatisfiable within the time frame of 14100 seconds, and the "Time" column represents the average time required to verify a query (in seconds), computed over the queries that the solver successfully verified.

These results highlight the superiority of our approach over both Single-Neuron Relaxation and Multi-Neuron Relaxation: thanks to the tighter hyperplanes discovered by PMNR, PMNR-enhanced Marabou has verified 245 queries out of 272, a 40% improvement over base Marabou which verified only 156 queries, and a 612% improvement over PMNR-ALL which verified merely 40 queries within the time limit due to its higher runtime overhead.

Table 1. Comparing Pmnr to DeepPoly and PmnrAll on piecewise-linear networks.

Model	Queries	Pmnr		DeepPoly		PmnrAll	
		Solved ↑	Time ↓	Solved ↑	Time ↓	Solved ↑	Time ↓
LeakyRelu5 × 100	100	**100**	**67**	97	129	40	5003
LeakyRelu8 × 100	100	**98**	**294**	33	501	0	∞
LeakyRelu14 × 28	36	**29**	1321	26	**16**	0	∞
ReluSignMax	36	**18**	**6138**	**18**	6742	0	∞
Total	272	**245**	**752**	174	866	40	5003

5.3 FFNNs with Non-Piecewise-Linear Activations

For evaluation on non-piecewise-linear activations, we used the PyTorch library to train MNIST classifiers containing the Sigmoid [42], Bilinear (Multiplication) [37] and Softmax [49] activations. We executed Marabou on 36 local robustness queries per network, characterized in the same way as the LeakyRelu 14×28 benchmarks. Summary statistics for our experiments on NPL networks are visible at Table 2. Notably, PMNR-enhanced Marabou verified 56 queries out of 72, an 100% increase over base Marabou (28 verified) and a 3% increase over PMNR-ALL-enhanced Marabou (54 verified). Overall, out of the 344 robustness queries tested, PMNR-enhanced Marabou successfully solved 301 queries, an 49% improvement over base Marabou which solved 202 queries; and a 220% improvement over PMNR-ALL-enhanced Marabou which solved 94. Employing PMNR within Marabou resulted in an average time requirement for verification of 619 s, which is 17% faster than DeepPoly (751 s) and 86% faster than PMNR-ALL (4622 s).

Table 2. Comparing Pmnr to DeepPoly and PmnrAll on NPL FFNNs.

Model	Queries	Pmnr		DeepPoly		PmnrAll	
		Solved ↑	Time ↓	Solved ↑	Time ↓	Solved ↑	Time ↓
ReluBilinearSoftmax	36	29	54	10	**46**	**36**	6485
LeakyReluSigmoid	36	**27**	22	18	**19**	18	50
Total	72	**56**	38	28	**28**	54	4340

Figure 4 compares the runtime of PMNR-enhanced Marabou to base Marabou and PMNR-ALL-enhanced Marabou for both classes of benchmarks, while Fig. 3 displays the cumulative runtime of Marabou for every model. Both figures show that, for all models beside ReluBilinearSoftmax, PMNR-enhanced Marabou solved all queries more rapidly compared to PMNR-ALL-enhanced Marabou—due to the latter's higher associated computational complexity. Further, the figures show that for all models beside LeakyReluSigmoid there exist several dozens of easier queries, which are solved quickly by both methods and for which base Marabou runs faster than PMNR-enhanced Marabou—since the tighter concrete bounds discovered by augmenting DeepPoly with PMNR cause an unnecessary overhead. Nonetheless, the tighter bounds revealed by PMNR assist Marabou in solving the remaining instances that base Marabou would take longer to verify, or fail to verify within the timeout. This is readily apparent in Fig. 3, as the graph of the cumulative number of instances PMNR-enhanced Marabou solved "catches up" to the graph of base Marabou and surpasses it.

6 Related Work

Bound Tightening for DNN Verification. The problem of DNN verification has been thoroughly studied in recent years, bringing about various approaches to tackle this problem, including BaB-based [4,11,48] and SMT-based techniques [23,24,51], abstraction-refinement [9], techniques featuring LP [8], MILP solvers [45,54], Lipschitz bounds [43] and other approaches [28].

Our work focuses on bound tightening, which is a key element in many DNN verification techniques. Since symbolic bound tightening methods were first introduced [42,46,47], various techniques have been devised to improve upon them—for instance, by forward-backward abstract interpretation [50], derivation of over-approximations on the inputs with dual optimization [26] or with optimizable parameters [52,57], reducing errors in symbolic bound propagation [54], spurious region-guided refinement [53], inferring multi-neuron bounds from MILP coefficients [55] or from prior branching performed [58], and producing multi-neuron bounds for all neurons [32,41].

Building on this large body of existing work, our Partial Multi-Neuron Relaxation (PMNR) approach heuristically selects neurons and generates multi-neuron constraints only involving them and their source neurons. Our framework is general, and is novel in the sense that it allows selecting neurons arbitrarily, without needing to perform actual branching, calculating MILP coefficients, or generating multi-neuron bounds for all neurons.

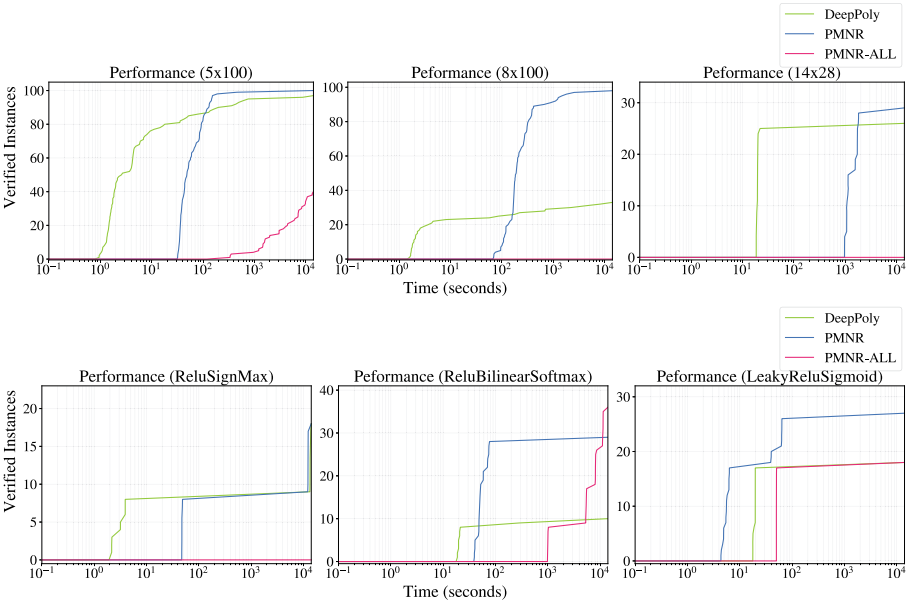

Fig. 3. Cumulative instances verified vs. time (seconds, log scale) for LeakyReLU FFNNs (Top) and other activations FFNNs (bottom).

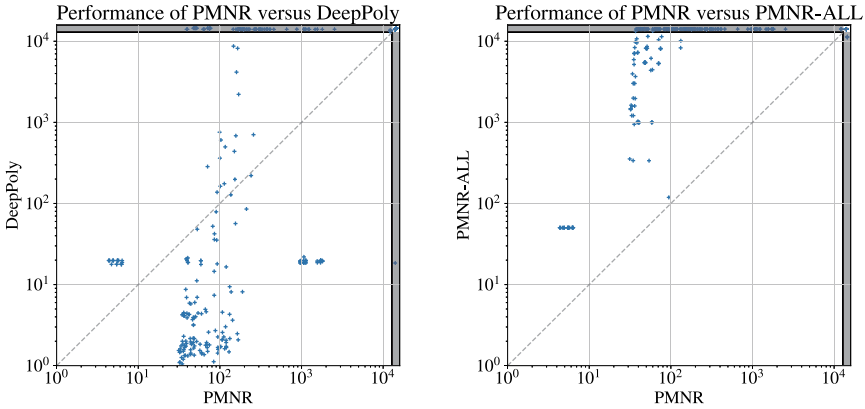

Fig. 4. Comparing the runtimes (seconds, logscale) of PMNR-enhanced Marabou to base Marabou (Left) and to PMNR-ALL-enhanced Marabou (right).

Dual Optimization of Multi-neuron Bounds. The INVPROP algorithm [26] receives concrete bounds and single-neuron bounds for a ReLU DNN, and uses them to learn bounding hyper-planes using dual linear optimization. INVPROP can be used for generating multi-neuron bounds for ReLU networks, and it inspired our generalized dual optimization method (see Appendices B and C of the full version of this paper [39])—which played a key part in our heuristic for generating multi-neuron bounds for general activations, as detailed in Subsect. 4.2. We further integrated the GenBaB branch-and-bound method for arbitrary non-linearities [37] in our BHSO framework to further optimize hyper-planes. Finally, the dual optimization technique β-CROWN [48], which encodes split-neuron constraints using optimizable parameters, has been successfully applied to obtain multi-neuron bounds for ReLU networks [55]. Integrating this technique with Generalized INVPROP while supporting arbitrary activation functions remains an intriguing direction for future work.

Other Verification-Related Tasks. The Marabou verifier and other verification tools have been successfully applied to a wide array of tasks, including verification of binarized [1], quantized [16] and recurrent [20] neural networks, verification of aerospace controllers [31] and reinforcement-learning systems [30], proof production and minimization [10,17,18], ensemble selection [2] and multi-layer modification [35]. Our proposed approach for extending bound tightening could benefit these tasks.

7 Conclusion and Future Work

We presented our approach to augment existing Single Neuron Relaxation-based bound tightening methods by learning multi-neuron bounds featuring only a heuristically selected subset of all neurons. We achieved this by designing new heuristics for neuron selection (NSSE) and multi-neuron bound generation by altering contemporary branching heuristics and bound tightening algorithms. Implementing PMNR in Marabou has successfully resulted in the derivation of tighter bounds at the cost of a runtime overhead for simpler queries, and we seek to explore directions to improve it further. Some of our directions for future work include: Evaluating our approach over non-MNIST benchmarks and comparing it to, or combining it with, other related approaches, including PRIMA [32], k-ReLU [41] or β-CROWN [48]; adding support for GPU-based parallelization; improving our NSSE and BHSO methods, by integrating the resulting multi-neuron bounds and BHSO-inferred infeasible branch combinations with search-based techniques; combining our approach with automatic inferring of single-neuron linear over-approximations [33] to support arbitrary black-box activations, without expert-designed linear bounds.

Acknowledgments. This research was partially supported by a grant from the Israeli Science Foundation (grant number 558/24). In addition, this work was partially funded by the European Union (ERC, VeriDeL, 101112713). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Amir, G., Wu, H., Barrett, C., Katz, G.: An SMT-based approach for verifying binarized neural networks. In: Proceedings of 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 203–222 (2021). https://doi.org/10.1007/978-3-030-72013-1_11
2. Amir, G., Zelazny, T., Katz, G., Schapira, M.: Verification-aided deep ensemble selection. In: Proceedings of 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 27–37 (2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_8
3. Bojarski, M., et al.: End to End Learning for Self-Driving Cars (2016), technical Report. https://arxiv.org/abs/1604.07316
4. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and Bound for Piecewise Linear Neural Network Verification (2025), technical Report. https://arxiv.org/abs/1909.06588
5. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. J. Artifi. Intell. Res. **70**, 245–317 (2021). https://doi.org/10.1613/jair.1.12228
6. Cao, H., et al.: Swin-Unet: Unet-like pure transformer for medical image segmentation. In: Proc. European Conference on Computer Vision (ECCV), pp. 205–218 (2023). https://doi.org/10.1007/978-3-031-25066-8_9

7. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: Proc. IEEE Symposium on Security and Privacy (S&P), pp. 39–57 (2017). https://doi.org/10.1109/SP.2017.49
8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Proc. 15th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 269–286 (2017). https://doi.org/10.1007/978-3-319-68167-2_19
9. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Proc. 32nd International Conference on Computer Aided Verification (CAV), pp. 43–65 (2020). https://doi.org/10.1007/978-3-030-53288-8_3
10. Elboher, Y.Y., Isac, O., Katz, G., Ladner, T., Wu, H.: Abstraction-based proof production in formal verification of neural networks. In: Proc. 8th International Symposium on AI Verification (SAIV), pp. 203–220 (2025). https://doi.org/10.1007/978-3-031-99991-8_10
11. Ferrari, C., Muller, M.N., Jovanovic, N., Vechev, M.: Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound (2022), technical Report. https://arxiv.org/abs/2205.00263
12. Gemini Team Google: Gemini: A Family of Highly Capable Multimodal Models (2025), technical Report. https://arxiv.org/abs/2312.11805
13. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proc. International Conference on Learning Representations (ICLR) (2015). https://doi.org/10.48550/arXiv.1412.6572
14. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). https://www.deeplearningbook.org
15. Gurobi Optimizer Reference Manual: Gurobi Optimization, LLC (2026). https://www.gurobi.com
16. Huang, P., et al.: Towards efficient verification of quantized neural networks. In: Proc. AAAI Conference on Artificial Intelligence, pp. 21152–21160 (2024). https://doi.org/10.1609/aaai.v38i19.30108
17. Isac, O., Barrett, C., Zhang, M., Katz, G.: Neural network verification with proof production. In: Proc. 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 38–48 (2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_9
18. Isac, O., Refaeli, I., Wu, H., Barrett, C., Katz, G.: Proof minimization in neural network verification. In: Proc. 27th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI), pp. 99–124 (2026). https://doi.org/10.1007/978-3-032-15700-3_6
19. Isac, O., Zohar, Y., Barrett, C., Katz, G.: DNN verification, reachability, and the exponential function problem. In: Proc. 34th International Conference on Concurrency Theory (CONCUR), pp. 26:1–26:18 (2023). https://doi.org/10.4230/LIPIcs.CONCUR.2023.26
20. Jacoby, Y., Barrett, C., Katz, G.: Verifying recurrent neural networks using invariant inference. In: Proc. 18th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 57–74 (2020). https://doi.org/10.1007/978-3-030-59152-6_3
21. John, J., et al.: Highly accurate protein structure prediction with alphafold. Nature **596**, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2
22. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: Proc. 35th IEEE/AIAA Digital

Avionics Systems Conference (DASC), pp. 1–10. (2016). https://doi.org/10.1109/DASC.2016.7778091
23. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proc. 29th International Conference on Computer Aided Verification (CAV), pp. 97–117 (2017). https://doi.org/10.1007/978-3-319-63387-9_5
24. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Proc. 31st International Conference on Computer Aided Verification (CAV), pp. 443–452 (2019). https://doi.org/10.1007/978-3-030-25540-4_26
25. Kaulen, K., et al.: The 6th International Verification of Neural Networks Competition (VNN-COMP 2025): Summary and Results (2025), technical Report. https://arxiv.org/abs/2512.19007
26. Kotha, S., Brix, C., Kolter, J.Z., Dvijotham, K., Zhang, H.: Provably bounding neural network preimages. In: Proc. 37th Conference on Neural Information Processing Systems (NeurIPS), pp. 80270–80290. (2023)
27. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. of the IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
28. Li, L., Xie, T., Li, B.: SoK: certified robustness for deep neural networks. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 1289–1310 (2023). https://doi.org/10.1109/SP46215.2023.10179303
29. Lin, Z., et al.: Evolutionary-scale prediction of atomic-level protein structure with a language model. Science **379**(6637), 1123–1130 (2023). https://doi.org/10.1126/science.ade2574
30. Mandal, U., et al: Formally verifying deep reinforcement learning controllers with lyapunov barrier certificates. In: Proc. 24th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD), pp. 95–106 (2024). https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_15
31. Mandal, U., et al.: Safe and reliable training of learning-based aerospace controllers. In: 43rd AIAA DATC/IEEE Digital Avionics Systems Conference (DASC), pp. 1–10 (2024). https://doi.org/10.1109/dasc62030.2024.10749499
32. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: general and precise neural network certification via scalable convex hull approximations. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022). https://doi.org/10.1145/3498704
33. Paulsen, B., Wang, C.: LinSyn: synthesizing tight linear bounds for arbitrary neural network activation functions. In: Proc. 28th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 357–376 (2022). https://doi.org/10.1007/978-3-030-99524-9_19
34. Peng, Y., Bao, Y., Chen, Y., Wu, C., Meng, C., Lin, W.: DL2: A Deep Learning-driven Scheduler for Deep Learning Clusters (2019), technical Report. https://arxiv.org/abs/1909.06040
35. Refaeli, I., Katz, G.: minimal multi-layer modifications of deep neural networks. In: 5th International Workshop on Software Verification and Formal Methods for ML-Enables Autonomous Systems (FoMLAS), pp. 46–66 (2022). https://doi.org/10.1007/978-3-031-21222-2_4
36. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Proc. 33rd Conference on Neural Information Processing Systems (NeurIPS), pp. 9835–9846 (2019)

37. Shi, Z., Jin, Q., Kolter, Z., Jana, S., Hsieh, C., Zhang, H.: Neural Network Verification with Branch-and-Bound for General Nonlinearities (2025), technical Report. https://arxiv.org/abs/2405.21063
38. Shmuel, I., Katz, G.: Neural Network Verification using Partial Multi-Neuron Relaxation (Code) (2026). https://github.com/ido-shm-uel/PMNR-Code
39. Shmuel, I., Katz, G.: Neural Network Verification using Partial Multi-Neuron Relaxation (Full Version) (2026), technical Report. https://arxiv.org/abs/2605.30155
40. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition (2014), technical Report. https://arxiv.org/abs/1409.1556
41. Singh, G., Ganvir, R., Puschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Proc. 33rd Conf. on Neural Information Processing Systems (NeurIPS), pp. 15098–15109 (2019)
42. Singh, G., Gehr, T., Puschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) (2019). https://doi.org/10.1145/3290354
43. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing Properties of Neural Networks (2013), technical Report. https://arxiv.org/abs/1312.6199
44. The PyTorch Team: PyTorch 2: faster machine learning through dynamic python bytecode transformation and graph compilation. In: Proc. of the 29th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 929–947. (2024). https://doi.org/10.1145/3620665.3640366
45. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating Robustness of Neural Networks with Mixed Integer Programming (2017), technical Report. https://arxiv.org/abs/1711.07356
46. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Proc. 32nd Conference on Neural Information Processing Systems (NeurIPS), pp. 6369–6379. (2018)
47. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proc. 27th USENIX Security Symposium, pp. 1599–1614 (2018)
48. Wang, S., et al.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Proc. 35th Conference on Neural Information Processing Systems (NeurIPS), pp. 29909–29921 (2021)
49. Wei, D., Wu, H., Wu, M., Chen, P., Barrett, C., Farchi, E.: Convex Bounds on the Softmax Function with Applications to Robustness Verification. In: Proc. 26th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 6853–6878 (2023)
50. Wu, H., Barrett, C., Sharif, M., Narodytska, N., Singh, G.: Scalable verification of GNN-based job schedulers. Proc. ACM Program. Lang. **6**(OOPSLA2), 1036–1065 (2022). https://doi.org/10.1145/3563325
51. Wu, H., et al.: Marabou 2.0: a versatile formal analyzer of neural networks. In: Proc. 36th International Conference on Computer Aided Verification (CAV), pp. 249–264 (2024). https://doi.org/10.1007/978-3-031-65630-9_13
52. Xu, K., et al.: Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers (2020), technical Report. https://arxiv.org/abs/2011.13824
53. Yang, P., et al.: Improving neural network verification through spurious region guided refinement. In: Proc. 27th International Conference on Tools and Algo-

rithms for the Construction and Analysis of Systems (TACAS), pp. 389–408 (2021). https://doi.org/10.1007/978-3-030-72016-2_21
54. Zelazny, T., Wu, H., Barrett, C., Katz, G.: On optimizing back-substitution methods for neural network verification. In: Proc. 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD). pp. 17–26 (2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_7
55. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C., Kolter, J.Z.: General cutting planes for bound-propagation-based neural network verification. In: Proc. 36th Conference on Neural Information Processing Systems (NeurIPS), pp. 1656–1670 (2022)
56. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Proc. 32nd Conference on Neural Information Processing Systems (NeurIPS), pp. 4944–4953 (2018)
57. Zhang, X., Wang, B., Kwiatkowska, M.: Provable Preimage Under-Approximation for Neural Networks (Full Version) (2024), technical Report. https://arxiv.org/abs/2305.03686
58. Zhou, D., Brix, C., Hanasusanto, G.A., Zhang, H.: Scalable neural network verification with branch-and-bound inferred cutting planes. In: Proc. 38nd Conference on Neural Information Processing Systems (NeurIPS), pp. 29324–29353 (2024)