Boosting Multi-Neuron Convex Relaxation for Neural Network Verification*

Xuezhou Tang[0009−0005−5227−8682], Ye Zheng[0000−0003−0623−9613], and Jiaxiang Liu[0000−0002−6725−8167](✉)

Shenzhen University, Shenzhen, China
2110276137@email.szu.edu.cn
zhengyeah@foxmail.com
jiaxiang0924@gmail.com

Abstract. Formal verification of neural networks is essential for their deployment in safety-critical real-world applications, such as autonomous driving and cyber-physical controlling. Multi-neuron convex relaxation is one of the mainstream methods to improve verification precision. However, existing techniques rely on empirically selecting neuron groups before performing multi-neuron convex relaxation, which may yield redundant yet expensive convex hull computations. This paper proposes a volume approximation based approach for selecting neuron groups. We approximate the volumes of convex hulls for all group candidates, without calculating their convex hulls. The group candidates with small volumes are then selected for convex hull computation, aiming at ruling out unnecessary convex hulls with loose relaxation. We implement our approach as the neural network verification tool FAGMR, and evaluate it against state-of-the-art tools including PRIMA, α,β-CROWN, and ERAN on neural networks trained by MNIST and CIFAR-10. The experimental results demonstrate that FAGMR is more efficient than these tools, yet with the same or sometimes better verification precision.

Keywords: Neural network verification · Multi-neuron relaxation · Volume approximation

1 Introduction

The increasing adoption of neural networks in many safety-critical scenarios has underscored their safety and robustness. However, the existence of adversarial examples is revealed to be a severe threat. That is, there exist perturbed inputs (e.g. images) that are human-imperceptible but give rise to misclassification by a neural network. For example, forged traffic signs can fool certain autonomous driving systems [16]. They look almost the same for humans, yet making auto-driving systems output incorrect predictions, hence leading to unexpected behaviors.

* This work is supported by the National Natural Science Foundation of China (61836005), the Natural Science Foundation of Guangdong Province (2022A1515011458, 2022A1515010880), and the Shenzhen Science and Technology Innovation Program (JCYJ20210324094202008).

A large body of research aims to find adversarial examples based on testing (see a survey [30]). They are usually effective in falsifying robustness. Notwithstanding, the fact that these techniques discover no adversarial examples does not guarantee robustness. On the other hand, *formal verification*, which is complementary to testing, mathematically proves the robustness of a given neural network against perturbed inputs, thus providing a formal guarantee for safety-critical applications.

Formal verification of neural networks usually needs to compute the output range of a neural network given a perturbed input range. Computing an output for a single input is trivial, but computing an output region for an input region is significantly more complex. The difficulty arises from the composition of the non-linear activation functions, which leads to a highly non-linear input-output relation of the neural network. So the key challenge is to handle the enormous non-linear functions in a precise and scalable manner.

Convex relaxation methods *over-approximate* the non-linear activation functions with convex polytopes, usually represented as linear constraints. Among them, single-neuron convex relaxation based methods over-approximate each neuron separately (e.g., [20,21,29,28,33]). These methods do not capture the inter-dependencies between neurons, so they are fundamentally less precise than multi-neuron convex relaxation based methods. The latter takes multiple neurons jointly into account, designing over-approximations for groups of neurons [19,24,17]. An essential problem of multi-neuron relaxation based methods lies in convex hull computations. Typically, in the first step, these methods select groups of neurons of size k ($k \geq 2$) in the same activation layer. For each group, the $\mathbb{R}^k \times \mathbb{R}^k$ input-output relation of its activation functions is then over-approximated jointly. The over-approximation is performed by computing a convex hull of the input-output relation, represented by a set of linear constraints. It is believed that the more groups of neurons are considered and the more overlap between groups is allowed, the more precise verification results can be achieved. However, NP-hardness of convex hull computation problems limits the number of groups to be selected. For instance, adopting an exact convex hull computing algorithm, kPoly [19] partitions the neurons of an activation layer into small sets of size $n_s \leq 5$ and only selects groups of $k \leq 3$ neurons within each partition. Prima [17] proposes a polynomial-time method for approximating convex hulls, hence allowing to consider a larger number of groups in a reasonable time limit. Similarly, it partitions all neurons with respect to n_s and selects a subset of all size-k groups within each partition. But the parameter n_s in Prima is significantly larger than that in kPoly, yielding significant precision improvement. Nevertheless, these parameters and the selection of groups are decided empirically. They may not perform equally well on different verification problems, even on different activation layers in the same network. On the other hand, we observe that there may exist redundant groups, in the sense that the constraints of their convex hulls are implied by the constraints generated by other groups. Convex hull computations of these redundant groups are unnecessary and should be avoided.

In this paper, we seek to improve the efficiency of multi-neuron convex relaxation based methods by heuristically selecting neuron groups. The main idea is to evaluate the tightness of over-approximation by the volume of the convex hull. The exact calculation of volumes of (high-dimensional) convex hulls is infeasible. More importantly, it does not avoid the unnecessary convex hull computations. We propose to instead under- and over-approximate the volumes of convex hulls without the need for computing the convex hulls. Neuron groups with small estimated volumes will be selected while groups with large volumes are eliminated. In such a way, some unnecessary yet expensive convex hull computations are avoided.

For evaluation, we implement our approach as a neural network verification tool FaGMR (**Fa**st **G**rouping for **M**ulti-neuron **R**elaxation). We compare FaGMR with state-of-the-art tools Prima [17], α,β-CROWN [28] and ERAN [23] on neural networks trained by the widely-used datasets MNIST and CIFAR-10. The experimental results show that FaGMR is faster than Prima, α,β-CROWN, and ERAN, by spending on average 11.2%, 43.1%, and 15.2% less verification time respectively. Meanwhile, FaGMR successfully verifies at least the same number of verification problems as Prima and ERAN, and on average 46.7% more than α,β-CROWN.

Our contributions are summarized as follows:

- We propose a volume approximation based approach to automatically select neuron groups for multi-neuron relaxation methods. It allows to avoid unnecessary yet expensive convex hull computations, hence boosting the efficiency of multi-neuron relaxation methods.
- We implement our approach as a verification tool FaGMR and conduct an extensive evaluation, demonstrating the efficacy of our approach.

Tool Availability. To foster further research, we place FaGMR into the public domain. The source code is available at https://github.com/formes20/FaGMR.

Organization. Section 2 recalls necessary backgrounds on neural network verification and the multi-neuron relaxation method. Our approach is presented in Section 3 and evaluated in Section 4. Related work is discussed in Section 5. We conclude our presentation in Section 6.

2 Preliminaries

Notations. We reserve lowercase Latin and Greek letters $a, b, x, \ldots, \theta, \ldots$ for scalars, bold $\boldsymbol{a}$ for vectors, capitalized bold $\boldsymbol{A}$ for matrices, and capitals A, calligraphic $\mathcal{A}$ or blackboard bold $\mathbb{A}$ for sets. Similarly, scalar functions are denoted as $f : \mathbb{R}^d \to \mathbb{R}$ and vector valued functions as $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^k$. Given n elements, the number of k-combinations is represented by $\binom{n}{k}$.

2.1 Neural Network Verification

A *(feedforward) neural network* $\boldsymbol{h}(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}$ is a $|\mathcal{Y}|$-dimensional vector valued function from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. Specifically, $\boldsymbol{h}(\boldsymbol{x})$ is the interleaved composition of affine function layers $\boldsymbol{g}_i(\boldsymbol{x}) = \boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{b}_i$, with non-linear activation layers $\boldsymbol{f}_i(\boldsymbol{x})$:

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}_\ell \circ \boldsymbol{f}_\ell \circ \boldsymbol{g}_{\ell-1} \circ \cdots \circ \boldsymbol{f}_1 \circ \boldsymbol{g}_0(\boldsymbol{x})$$

where ℓ is the number of hidden layers. $\boldsymbol{f}_i(\boldsymbol{x})$ applies non-linear activation functions in an element-wise manner. If $\boldsymbol{h}(\boldsymbol{x})$ is a classification neural network, it will output the index c of its maximum output vector component, i.e. $c = \arg\max_j \boldsymbol{h}(\boldsymbol{x})_j$.

A neural network verification problem commonly needs to verify the *robustness* property. A robust neural network must satisfy the smoothness assumption [8], i.e., for any input $\boldsymbol{x}$ and a small perturbation $\boldsymbol{\delta}$, $\boldsymbol{h}(\boldsymbol{x}+\boldsymbol{\delta}) \approx \boldsymbol{h}(\boldsymbol{x})$ should hold. In the case of classification tasks, this assumption conforms to the visual capabilities of human: if $\boldsymbol{x}$ looks similar to $\boldsymbol{x}'$, they should belong to the same class.

Formally, perturbed inputs are defined by an l_p-norm ball neighborhood of $\boldsymbol{x}$:

$$\mathbb{B}_\theta^p(\boldsymbol{x}) = \{\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta} \mid \|\boldsymbol{\delta}\|_p \leq \theta\}$$

where θ is the *perturbation threshold* that bounds $\boldsymbol{\delta}$. We would like to verify that the neural network $\boldsymbol{h}(\boldsymbol{x})$ does not misclassify any perturbed input in this region.

Definition 1. *Given a neural network $\boldsymbol{h}(\boldsymbol{x})$, an input $\boldsymbol{x} \in \mathcal{X}$ and a perturbation threshold θ, a* verification problem *is to give the truth value of the following statement:*

$$\arg\max_j \boldsymbol{h}(\boldsymbol{x})_j = \arg\max_j \boldsymbol{h}(\boldsymbol{x}')_j, \quad \textit{for each } \boldsymbol{x}' \in \mathbb{B}_\theta^p(\boldsymbol{x}).$$

If the statement in Definition 1 is true, we conclude that the neural network $\boldsymbol{h}$ is robust with respect to the input $\boldsymbol{x}$ against the perturbation threshold θ.

2.2 Multi-Neuron Convex Relaxation

Multi-neuron convex relaxation based methods like PRIMA [17] solve neural network verification problems (Definition 1) by encoding them as linear optimization problems. Specifically, given a neural network $\boldsymbol{h}$, an input $\boldsymbol{x}$ and a perturbation threshold θ, these methods encode the whole network $\boldsymbol{h}$ w.r.t. the (convex) region $\mathbb{B}_\theta^p(\boldsymbol{x})$ via linear constraints, define a linear optimization objective for the target statement $\arg\max_j \boldsymbol{h}(\boldsymbol{x})_j = \arg\max_j \boldsymbol{h}(\boldsymbol{x}')_j$, and finally invoke an LP solver to obtain a bound determining whether $\boldsymbol{h}$ is robust. Note that all affine function layers $\boldsymbol{g}_i$ in $\boldsymbol{h}$ can be described exactly by linear constraints, while the non-linear activation layers $\boldsymbol{f}_i$ have to be over-approximated using linear constraints in their input-output spaces. Given a non-linear activation layer, conventional single-neuron convex relaxation based methods (e.g. [20,21,29,28,33]) take each

single neuron in turn to generate neuron-wise over-approximations, accumulated as the layer-wise over-approximation. Instead, multi-neuron convex relaxation based methods select (possibly overlapping) groups of neurons, generate linear constraints for each group of neurons as group-wise over-approximations, then accumulate them as the layer-wise over-approximation. By capturing interdependencies between different neurons when obtaining group-wise over-approximations, multi-neuron relaxation methods achieve tighter over-approximations.

We now review the multi-neuron constraints leveraged by PRIMA [17] that our approach focuses on.

Let us fix some single activation layer. Assume a layer-wise input polytope $\mathcal{S}$ constraining all the inputs of the layer, and a group $G = \{v_1, v_2, \ldots, v_k\}$ of neurons of size $k \geq 2$. PRIMA starts the multi-neuron relaxation for G by projecting $\mathcal{S}$ onto the input dimensions of the group G, i.e. onto the $(x_1, x_2, \ldots, x_k)$-space, where x_i denotes the input of the neuron v_i. More precisely, PRIMA computes an octahedral over-approximation $\mathcal{P}_G$ of the projection instead of an exact projection, following the idea of KPOLY [19]. Then for the k-dimensional $\mathcal{P}_G$, PRIMA computes a $2k$-dimensional convex over-approximation $\mathcal{K}_G$ of the input-output relation of G through its novel *Split-Bound-Lift Method (SBLM)*. The linear constraints constituting the polytope $\mathcal{K}_G$ are the expected multi-neuron constraints for the group G. SBLM constructs $\mathcal{K}_G$ from $\mathcal{P}_G$ via two phases: splitting and lifting. In the following, we omit the subscript G, using $\mathcal{P}$ and $\mathcal{K}$ for clarity when causing no ambiguity.

Splitting. Assume that the activation function in the neural network is $f : \mathbb{D} \to \mathbb{R}$. The splitting phase requires a set of intervals $\mathcal{D}^j$ covering the domain $\mathbb{D}$. We will focus on the activation function $\mathrm{ReLU} : \mathbb{R} \to \mathbb{R}$ defined by $\mathrm{ReLU}(x) = \max(x, 0)$ when introducing PRIMA and our approach in the following for simplicity, and generalize our approach to activation functions Sigmoid and Tanh in Section 3.4. For the ReLU activation function, the required intervals are $\mathcal{D}^1 = (-\infty, 0]$ and $\mathcal{D}^2 = [0, +\infty)$, whose union covers the domain $\mathbb{D} = \mathbb{R}$. These intervals are instantiated w.r.t. the neuron v_i as $\mathcal{D}_i^1 = \{\boldsymbol{x} \in \mathbb{R}^k \mid x_i \leq 0\}$ and $\mathcal{D}_i^2 = \{\boldsymbol{x} \in \mathbb{R}^k \mid x_i \geq 0\}$. The splitting phase splits $\mathcal{P}$ according to the intervals $\mathcal{D}^j$ w.r.t. input variables x_i iteratively, considering one variable at a step. Without loss of generality, we (randomly) choose the splitting ordering $v_k, v_{k-1}, \ldots, v_1$. That is, the splitting phase splits w.r.t. x_k first, then x_{k-1} and so on, finally x_1. Formally, the splitting phase works as follows.

Definition 2 (Splitting). *Given a size-k group $G = \{v_1, \ldots, v_k\}$ of neurons and a polytope $\mathcal{P}$ in the $(x_1, \ldots, x_k)$-space. The* splitting *phase is performed by*

$$\begin{aligned} \mathcal{P}_{\epsilon}^{(0)} &= \mathcal{P}; \\ \mathcal{P}_{\tau,j}^{(i+1)} &= \mathcal{P}_{\tau}^{(i)} \cap \mathcal{D}_{k-i}^{j}, \quad \textit{for } 0 \leq i < k \textit{ and } j \in \{1, 2\} \end{aligned}$$

where ϵ denotes the empty sequence, and τ a sequence of length i containing 1 *and/or* 2*. The polytopes $\mathcal{P}_{\{1,2\}^k}^{(k)}$ are obtained after splitting, called* quadrants*.*

Intuitively, at the i-th ($0 \leq i < k$) step, the input variable x_{k-i} is considered. Each polytope $\mathcal{P}_\tau^{(i)}$ obtained after the previous step is split by the hyperplane $\{\boldsymbol{x} \in \mathbb{R}^k \mid x_{k-i} = 0\}$, generating two polytopes $\mathcal{P}_{\tau,1}^{(i+1)}$ and $\mathcal{P}_{\tau,2}^{(i+1)}$. Note that for each polytope $\mathcal{P}_\tau^{(i)}$ generated during splitting, the length of τ equals i. The splitting phase can be regarded as constructing a complete binary tree. The initial polytope $\mathcal{P}$ is the root. All the generated polytopes $\mathcal{P}_\tau^{(i)}$ constitute the nodes, with i being the depth and τ the path from the root. The 2^k quadrants $\mathcal{P}_{\{1,2\}^k}^{(k)}$ are the leaves.

Lifting. The lifting phase extends and combines the 2^k k-dimensional quadrants into one $2k$-dimensional polytope, during which convex hulls are computed. Similar to splitting, this phase also progresses step by step, extending polytopes by one dimension corresponding to an output variable at each step. The lifting ordering is the reverse of the splitting one: $v_1, v_2, \ldots, v_k$. That is, the quadrants in the $(x_1, \ldots, x_k)$-space are lifted to the $(x_1, \ldots, x_k, y_1)$-space first, then $(x_1, \ldots, x_k, y_1, y_2)$-space and so on, where y_i denotes the output of neuron v_i. The resulting polytope $\mathcal{K}$ after the lifting phase is in the $(x_1, \ldots, x_k, y_1, \ldots, y_k)$-space.

The lifting phase requires a set of bounds $\mathcal{B}^j$ corresponding to the intervals $\mathcal{D}^j$, bounding the output of the activation function. Each $\mathcal{B}^j$ is a pair of linear functions of the form

$$\mathcal{B}^j = (a_j^{\leq}, a_j^{\geq}),$$

where $a_j^{\leq}$ and $a_j^{\geq}$ are linear functions satisfying

$$a_j^{\leq}(x) \leq f(x) \leq a_j^{\geq}(x), \text{ for each } x \in (\mathcal{D}^j \cap [\underline{x}, \overline{x}]),$$

where $\underline{x}$ and $\overline{x}$ are concrete lower and upper bounds of x, respectively. For the ReLU activation function, the required bounds are $\mathcal{B}^1 = (0, 0)$ and $\mathcal{B}^2 = (x, x)$. They are instantiated w.r.t. the neuron v_i as $\mathcal{B}_i^1 = \{\boldsymbol{x} \in \mathbb{R}^{k+i} \mid 0 \leq y_i \leq 0\} = \{\boldsymbol{x} \in \mathbb{R}^{k+i} \mid y_i = 0\}$ and $\mathcal{B}_i^2 = \{\boldsymbol{x} \in \mathbb{R}^{k+i} \mid x_i \leq y_i \leq x_i\} = \{\boldsymbol{x} \in \mathbb{R}^{k+i} \mid y_i = x_i\}$. Note that $\mathcal{B}_i^j$ is $(k+i)$-dimensional due to the lifting ordering, precisely, being in the $(x_1, \ldots, x_k, y_1, \ldots, y_i)$-space. Formally, the lifting phase progresses as follows.

Definition 3 (Lifting). *Given a size-k group $G = \{v_1, \ldots, v_k\}$ of neurons and 2^k quadrants $\mathcal{P}_{\{1,2\}^k}^{(k)}$ obtained by splitting. The* lifting *phase is performed by*

$$\mathcal{K}_{\{1,2\}^k}^{(0)} = \mathcal{P}_{\{1,2\}^k}^{(k)};$$
$$\mathcal{K}_\tau^{(i+1)} = \mathrm{conv}\Big((\mathrm{e}_{y_{i+1}}(\mathcal{K}_{\tau,1}^{(i)}) \cap \mathcal{B}_{i+1}^1) \cup (\mathrm{e}_{y_{i+1}}(\mathcal{K}_{\tau,2}^{(i)}) \cap \mathcal{B}_{i+1}^2) \Big), \quad \textit{for } 0 \leq i < k$$

where $\mathrm{conv}(\cdot)$ *denotes the convex hull, and* $\mathrm{e}_y(\mathcal{A}) = \mathcal{A} \times \mathbb{R}$ *extends* $\mathcal{A}$ *by the dimension* y *for any set* $\mathcal{A}$*. The polytope* $\mathcal{K} = \mathcal{K}_\epsilon^{(k)}$ *is the output of the lifting phase, as well as the SBLM.*

PRIMA computes the convex hull $\text{conv}(\cdot)$ via its novel *Partial Double Description Method (PDDM)*, which we do not detail in this paper. Intuitively, the lifting phase progresses as per the binary tree constructed during the splitting phase, from the leaves to the root. At the i-th ($0 \leq i < k$) step, any two children $\mathcal{K}_{\tau,1}^{(i)}$ and $\mathcal{K}_{\tau,2}^{(i)}$ of the same parent are extended by a new dimension y_{i+1}, and bounded by $\mathcal{B}^1$ and $\mathcal{B}^2$, respectively. The convex hull $\mathcal{K}_{\tau}^{(i+1)}$ of their union is put at the position of their parent. All quadrants are combined along the tree until only one polytope $\mathcal{K}$ is obtained at the root.

3 Volume Approximation Based Grouping

Before generating multi-neuron constraints for a given group of neurons as presented in Section 2.2, a key problem is that which and how many groups we should select to generate constraints. For an activation layer with n neurons, given the group size k, it is certainly ideal to consider all $\binom{n}{k}$ possible size-k groups. Nevertheless, computing multi-neuron constraints for each of them is too expensive due to the high complexity of convex hull computation, even by using SBLM. Empirically selecting groups by pre-defined parameters, like in KPOLY and PRIMA, is an efficient and effective solution in some scenarios. But we observe that sometimes these grouping strategies may lead to *redundant* groups, in the sense that the generated multi-neuron constraints for them are implied by the constraints for other groups.

Example 1. Assume that three neurons v_1, v_2, and v_3 are in the same activation layer, the group size $k = 2$. There are three possible groups $G_1 = \{v_1, v_2\}$, $G_2 = \{v_1, v_3\}$ and $G_3 = \{v_2, v_3\}$. Assume that the octahedral over-approximations for them are respectively

$$\begin{aligned}
\mathcal{P}_{G_1} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_1 + x_2 \leq 3,\ x_1 - x_2 \leq 3,\ -x_1 + x_2 \leq 3,\ -x_1 - x_2 \leq 3,\\
& -1 \leq x_1 \leq 1,\ -2 \leq x_2 \leq 2\ \},\\
\mathcal{P}_{G_2} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_1 + x_3 \leq 2,\ x_1 - x_3 \leq 2,\ -x_1 + x_3 \leq 2,\ -x_1 - x_3 \leq 2,\\
& -1 \leq x_1 \leq 1,\ -1 \leq x_3 \leq 1\ \},\\
\mathcal{P}_{G_3} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_2 + x_3 \leq 3,\ x_2 - x_3 \leq 3,\ -x_2 + x_3 \leq 3,\ -x_2 - x_3 \leq 3,\\
& -2 \leq x_2 \leq 2,\ -1 \leq x_3 \leq 1\ \}.
\end{aligned}$$

The multi-neuron constraints generated by PRIMA are as follows, where $\mathcal{C}_{G_i}$ denotes the constraints of the output polytope $\mathcal{K}_{G_i}$ by SBLM:

$$\begin{aligned}
\mathcal{C}_{G_1} = \{\ & 0.5x_1 - y_1 \leq 0.5,\ 0.5x_2 - y_2 \leq 1,\ -x_2 + y_2 \leq 0.36,\\
& -x_1 + y_1 \leq 0.18,\ y_1 \leq 1,\ y_2 \leq 1\ \},\\
\mathcal{C}_{G_2} = \{\ & 0.5x_1 - y_1 \leq 0.5,\ 0.5x_3 - y_3 \leq 1,\ -x_3 + y_3 \leq 0.18,\\
& -x_1 + y_1 \leq 0.18,\ y_1 \leq 1,\ y_3 \leq 1\ \},\\
\mathcal{C}_{G_3} = \{\ & 0.5x_3 - y_3 \leq 0.5,\ 0.5x_2 - y_2 \leq 1,\ -x_2 + y_2 \leq 0.36,\\
& -x_3 + y_3 \leq 0.18,\ y_2 \leq 1,\ y_3 \leq 1\ \}.
\end{aligned}$$

One can verify that $\mathcal{C}_{G_1}$ is implied by $\mathcal{C}_{G_2} \cup \mathcal{C}_{G_3}$. It means that the constraints in $\mathcal{C}_{G_1}$ are useless when accumulated into the layer-wise over-approximation which already includes $\mathcal{C}_{G_2}$ and $\mathcal{C}_{G_3}$. Therefore, $\mathcal{C}_{G_1}$ and thus the corresponding group G_1 are redundant.

The multi-neuron constraint generation, especially the convex hull computation therein, for the redundant groups like G_1 in Example 1 is unnecessary and should be avoided for efficiency. However, convex hull computation in high-dimension is unavoidable if we would like to identify these redundant groups precisely. Adopting the idea in [21], which considers in the single-neuron case an over-approximation with smaller area as a tighter over-approximation, we expect redundant groups to generate looser over-approximations (i.e. the polytopes $\mathcal{K}_G$) with larger (high-dimensional) volumes. Instead of exact volumes, our approach computes the approximations of the volumes in order to avoid expensive calculation in high-dimension for both convex hulls and exact volumes. The groups with larger approximated volumes are then discarded since they are more likely redundant. As a result, our approach produces neuron groups that may include fewer redundant ones, while their multi-neuron constraints preserve precision.

3.1 Overview of Our Approach

Our approach follows the multi-neuron relaxation framework. The key idea in our approach is to fast compute the volume approximations of all possible size-k neuron groups *before* selecting some of them to generate multi-neuron constraints.

Each neuron in the activation layer is processed iteratively. For each neuron, our approach comprises the following three steps: volume approximation, group selection, and convex hull computation. Figure 1 shows the workflow to select neuron groups involving the neuron v_i, when $k = 2$.

1. *Volume approximation.* Computing the precise volume of a $2k$-dimensional convex hull generated for a size-k neuron group is time-consuming. Therefore, we leverage the approximation methods detailed in Section 3.2 to under- and over-approximate the volume of the convex hull for each group candidate, in order to decide whether it will be selected for the $2k$-dimensional convex hull generation. Our volume approximation methods are based on Betke and Henk [4]. We improve their method for the over-approximation by utilizing the volume of the octahedron $\mathcal{P}$ to make the approximation tighter. Our over-approximation is hence obtained via k-dimensional calculation of the exact volume of $\mathcal{P}$ that can be done by off-the-shelf volume tools. It may reduce some $2k$-dimensional computations of unnecessary convex hulls. In the case where $k = 2$ and v_i is considered, Figure 1 demonstrates that the under- and over-approximations of the volumes of all group candidates (of size 2) containing v_i are calculated.
2. *Neuron group selection.* Once under- and over-approximated volumes have been computed for all the group candidates, "better" candidates are selected. Our selecting strategy prefers those groups with smaller volumes. It will

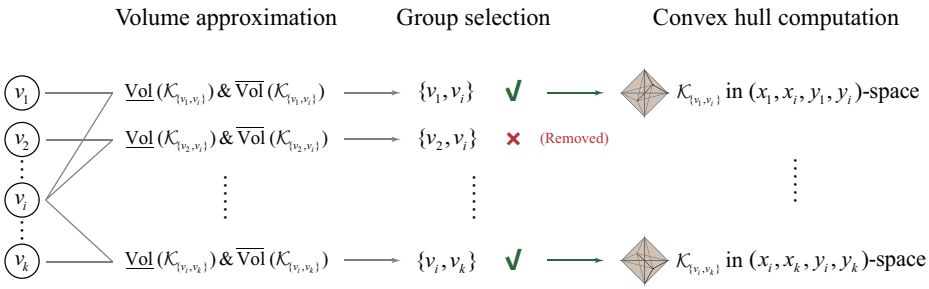

Fig. 1. Workflow of Our Approach to Select Groups for v_i ($k = 2$)

select the groups with very small over-approximated volumes, and eliminate the groups with very large under-approximated volumes. The details can be found in Section 3.3. In Figure 1, we can see that some group candidates (e.g. $\{v_2, v_i\}$) are weeded out by our selecting strategy. Only some groups (e.g. $\{v_1, v_i\}$ and $\{v_i, v_k\}$) are passed to the following convex hull solving step.

3. *Convex hull computation.* After the groups are decided, their $2k$-dimensional convex hulls are computed. The computation can be performed by existing techniques. Specifically, our implementation leverages SBLM and PDDM in PRIMA due to their efficiency and exactness. This step obtains the convex hull, thus the multi-neuron constraints, for each selected group. For instance, the convex hull $\mathcal{K}_{\{v_1,v_i\}}$ in the (x_1, x_i, y_1, y_i)-space for the selected group $\{v_1, v_i\}$ in Figure 1.

3.2 Volume Approximation

Given a group G of neurons, we detail in this section how to approximate the volume of the generated convex hull $\mathcal{K}_G$ (presented in Section 2.2) *without* actually computing the convex hull. Following the notations in Section 2.2, we denote the neuron group of size k as $G = \{v_1, \ldots, v_k\}$, the octahedral over-approximation input to SBLM as $\mathcal{P}$, and the convex hull output by SBLM as $\mathcal{K}$.

For an arbitrary (convex) d-dimensional polytope $\mathcal{Q} \in \mathbb{R}^d$, we denote its (high-dimensional) volume as $\mathrm{Vol}(\mathcal{Q})$, the under- and over-approximations of the volume as $\underline{\mathrm{Vol}}(\mathcal{Q})$ and $\overline{\mathrm{Vol}}(\mathcal{Q})$, respectively. That is, $\underline{\mathrm{Vol}}(\mathcal{Q}) \leq \mathrm{Vol}(\mathcal{Q}) \leq \overline{\mathrm{Vol}}(\mathcal{Q})$. Betke and Henk gave in [4] an algorithm to calculate the under- and over-approximations as follows.

Lemma 1 ([4]). *Given an arbitrary (convex) polytope $\mathcal{Q} \in \mathbb{R}^d$, the under- and over-approximations of the volume of $\mathcal{Q}$ can be calculated by*

$$\underline{\mathrm{Vol}}(\mathcal{Q}) = \frac{1}{d!} \cdot \prod_{i=1}^{d} (u_i - l_i)$$

$$\overline{\mathrm{Vol}}(\mathcal{Q}) = \prod_{i=1}^{d}(u_i - l_i)$$

where $[l_i, u_i]$ *is the range of the* i*-th dimension of* $\mathcal{Q}$*.*

We adopt the under-approximation given by Lemma 1 in our approach to calculate the under-approximated volume $\underline{\mathrm{Vol}}(\mathcal{K})$ of the convex hull $\mathcal{K}$.

On the other hand, for the over-approximated volume, the one given by Lemma 1 intuitively over-approximates the polytope via the high-dimensional box represented by the Cartesian product $\prod_{i=1}^{d}[l_i, u_i]$. The result is generic, but not tight enough for our purpose, since the convex hull $\mathcal{K}$ we consider is obtained by specific construction and thus has specific characteristics. Generally speaking, we over-approximate in our approach the convex hull $\mathcal{K}$ with a $2k$-dimensional *prism* [22].

A d-dimensional prism is geometrically formed by parallel segments of the same length drawn from all the points of a $(d-1)$-dimensional polytope called *base*. The volume of a prism $\mathcal{Q}$ with the base $\mathcal{S}$ can be computed by $\mathrm{Vol}(\mathcal{Q}) = \mathrm{Vol}(\mathcal{S}) \cdot h$, where h is the *height* of the prism. We have the following property about the over-approximations and the volumes of the generated convex hulls during the lifting phase (Definition 3).

Lemma 2. *Given a size-k group* $G = \{v_1, \ldots, v_k\}$ *of neurons and the polytope* $\mathcal{P}$ *input to SBLM, for any* $0 \leq i \leq k$*, each polytope* $\mathcal{K}_\tau^{(i)}$ *generated during the lifting phase satisfies:*

$$\mathcal{K}_\tau^{(i)} \subseteq \mathcal{Q}^{(i)},$$

$$\mathrm{Vol}(\mathcal{Q}^{(i)}) = \mathrm{Vol}(\mathcal{P}) \cdot \prod_{j=1}^{i}(\overline{y_j} - \underline{y_j}), \text{ and}$$

$$\mathrm{Vol}(\mathcal{K}_\tau^{(i)}) \leq \mathrm{Vol}(\mathcal{P}) \cdot \prod_{j=1}^{i}(\overline{y_j} - \underline{y_j}),$$

where $\mathcal{Q}^{(i)} = \mathcal{P} \times \prod_{j=1}^{i}[\ \underline{y_j}, \overline{y_j}\ ]$*,* $\underline{y_j}$ *and* $\overline{y_j}$ *denote the lower and upper bounds of the* y_j*-dimension of* $\mathcal{K}_\tau^{(i)}$*, respectively.*

Proof. We prove by induction on i.

- For the base case $i = 0$, $\mathcal{K}_\tau^{(0)} = \mathcal{P}_\tau^{(k)} \subseteq \mathcal{P} = \mathcal{Q}^{(0)}$ by Definitions 2 and 3. It trivially holds that $\mathrm{Vol}(\mathcal{Q}^{(0)}) = \mathrm{Vol}(\mathcal{P})$. The third statement $\mathrm{Vol}(\mathcal{K}_\tau^{(0)}) \leq \mathrm{Vol}(\mathcal{P})$ follows from the first two.
- For the induction step, assuming the three statements hold when $i = m$, we prove that they hold as well when $i = m + 1$. By Definition 3,

$$\mathcal{K}_\tau^{(m+1)} = \mathrm{conv}\Big((\mathrm{e}_{y_{m+1}}(\mathcal{K}_{\tau,1}^{(m)}) \cap \mathcal{B}_{m+1}^{1}) \cup (\mathrm{e}_{y_{m+1}}(\mathcal{K}_{\tau,2}^{(m)}) \cap \mathcal{B}_{m+1}^{2})\Big).$$

And by induction hypothesis, we have both $\mathcal{K}_{\tau,1}^{(m)} \subseteq \mathcal{Q}^{(m)}$ and $\mathcal{K}_{\tau,2}^{(m)} \subseteq \mathcal{Q}^{(m)}$. Hence,

$$\mathcal{K}_{\tau}^{(m+1)} \subseteq \mathrm{conv}\Big((\mathrm{e}_{y_{m+1}}(\mathcal{Q}^{(m)}) \cap \mathcal{B}_{m+1}^{1}) \cup (\mathrm{e}_{y_{m+1}}(\mathcal{Q}^{(m)}) \cap \mathcal{B}_{m+1}^{2})\Big).$$

Recall that for any i, $\mathcal{B}_i^1 = \{\boldsymbol{x} \in \mathbb{R}^{k+i} \mid y_i = 0\}$ and $\mathcal{B}_i^2 = \{\boldsymbol{x} \in \mathbb{R}^{k+i} \mid y_i = x_i\}$, so both $\mathcal{B}_{m+1}^1$ and $\mathcal{B}_{m+1}^2$ can be over-approximated by $\{\boldsymbol{x} \in \mathbb{R}^{k+m+1} \mid \underline{y_{m+1}} \leq y_{m+1} \leq \overline{y_{m+1}}\}$. Therefore, we further get

$$\begin{aligned}
\mathcal{K}_{\tau}^{(m+1)} &\subseteq \mathrm{conv}\Big((\mathrm{e}_{y_{m+1}}(\mathcal{Q}^{(m)}) \cap \{\boldsymbol{x} \in \mathbb{R}^{k+m+1} \mid \underline{y_{m+1}} \leq y_{m+1} \leq \overline{y_{m+1}}\}) \\
&\qquad \cup (\mathrm{e}_{y_{m+1}}(\mathcal{Q}^{(m)}) \cap \{\boldsymbol{x} \in \mathbb{R}^{k+m+1} \mid \underline{y_{m+1}} \leq y_{m+1} \leq \overline{y_{m+1}}\})\Big) \\
&= \mathrm{conv}\Big((\mathrm{e}_{y_{m+1}}(\mathcal{Q}^{(m)}) \cap \{\boldsymbol{x} \in \mathbb{R}^{k+m+1} \mid \underline{y_{m+1}} \leq y_{m+1} \leq \overline{y_{m+1}}\})\Big) \\
&= \mathrm{conv}\Big(\mathcal{Q}^{(m)} \times [\ \underline{y_{m+1}}, \overline{y_{m+1}}\ ]\Big).
\end{aligned}$$

Since $\mathcal{Q}^{(m)} = \mathcal{P} \times \prod_{j=1}^{m} [\ \underline{y_j}, \overline{y_j}\ ]$ by definition, $\mathcal{Q}^{(m)} \times [\ \underline{y_{m+1}}, \overline{y_{m+1}}\ ]$ is convex. So $\mathrm{conv}\Big(\mathcal{Q}^{(m)} \times [\ \underline{y_{m+1}}, \overline{y_{m+1}}\ ]\Big) = \mathcal{Q}^{(m)} \times [\ \underline{y_{m+1}}, \overline{y_{m+1}}\ ]$. Then,

$$\mathcal{K}_{\tau}^{(m+1)} \subseteq \mathcal{Q}^{(m)} \times [\ \underline{y_{m+1}}, \overline{y_{m+1}}\ ] = \mathcal{P} \times \prod_{j=1}^{m+1} [\ \underline{y_j}, \overline{y_j}\ ] = \mathcal{Q}^{(m+1)}.$$

The first statement is proved.
For the second statement, because $\mathcal{Q}^{(m+1)} = \mathcal{Q}^{(m)} \times [\ \underline{y_{m+1}}, \overline{y_{m+1}}\ ]$, it is a prism formed by the base $\mathcal{Q}^{(m)}$ and parallel segments along the y_{m+1}-axis, and the height is $\overline{y_{m+1}} - \underline{y_{m+1}}$. Thus its volume $\mathrm{Vol}(\mathcal{Q}^{(m+1)}) = \mathrm{Vol}(\mathcal{Q}^{(m)}) \cdot (\overline{y_{m+1}} - \underline{y_{m+1}})$. By induction hypothesis, $\mathrm{Vol}(\mathcal{Q}^{(m)}) = \mathrm{Vol}(\mathcal{P}) \cdot \prod_{j=1}^{m} (\overline{y_j} - \underline{y_j})$, it then follows that

$$\mathrm{Vol}(\mathcal{Q}^{(m+1)}) = \mathrm{Vol}(\mathcal{Q}^{(m)}) \cdot (\overline{y_{m+1}} - \underline{y_{m+1}}) = \mathrm{Vol}(\mathcal{P}) \cdot \prod_{j=1}^{m+1} (\overline{y_j} - \underline{y_j}).$$

Finally, the third statement follows from the first two, which concludes our proof. □

Lemma 2 states that each convex hull generated during the lifting phase in SBLM can be over-approximated by a prism. It moreover tells that the volume of the prism can be calculated through the volume of the input polytope $\mathcal{P}$ and the bounds of all y_i-dimensions. Now we have our main theorem for the volume approximation.

Theorem 1 (Volume approximation). *Given a size-k group $G = \{v_1, \ldots, v_k\}$ of neurons and the polytope $\mathcal{P}$ input to SBLM, the volume $\mathrm{Vol}(\mathcal{K})$ of the convex hull $\mathcal{K}$ output by SBLM is bounded by $\underline{\mathrm{Vol}}(\mathcal{K})$ and $\overline{\mathrm{Vol}}(\mathcal{K})$ as follows:*

$$\underline{\mathrm{Vol}}(\mathcal{K}) = \frac{1}{(2k)!} \cdot \prod_{i=1}^{k} \left((\overline{x_i} - \underline{x_i}) \cdot (\overline{y_i} - \underline{y_i})\right),$$

$$\overline{\mathrm{Vol}}(\mathcal{K}) = \mathrm{Vol}(\mathcal{P}) \cdot \prod_{i=1}^{k} (\overline{y_i} - \underline{y_i}),$$

where $\underline{x_i}$ and $\overline{x_i}$ (resp., $\underline{y_i}$ and $\overline{y_i}$) denote the lower and upper bounds of the x_i-dimension (resp., y_i-dimension) of $\mathcal{K}$, respectively.

Proof. The lower bound is derived by applying Lemma 1, while the upper bound by Definition 3 and Lemma 2. □

In Theorem 1, the bounds of each x_i can be computed by $\underline{x_i} = \min_{\boldsymbol{x} \in \mathcal{P}} x_i$ and $\overline{x_i} = \max_{\boldsymbol{x} \in \mathcal{P}} x_i$. Recall that the bounding linear functions required by the lifting phase for ReLU are $\mathcal{B}^1 = (a_1^{\leq}, a_1^{\geq}) = (0, 0)$ and $\mathcal{B}^2 = (a_2^{\leq}, a_2^{\geq}) = (x, x)$. The bounds of each y_i can be computed by $\underline{y_i} = \min_{\boldsymbol{x} \in \mathcal{P}} (a_1^{\leq}(x_i), a_2^{\leq}(x_i)) = \min_{\boldsymbol{x} \in \mathcal{P}} (0, x_i) = 0$ and $\overline{y_i} = \max_{\boldsymbol{x} \in \mathcal{P}} (a_1^{\geq}(x_i), a_2^{\geq}(x_i)) = \max_{\boldsymbol{x} \in \mathcal{P}} (0, x_i) = \overline{x_i}$. Theorem 1 indicates that the under- and over-approximations of the volume $\mathrm{Vol}(\mathcal{K})$ can be calculated via the volume of the easily generated, low-dimensional $\mathcal{P}$ as well as the bounds of all dimensions, without actually computing the computationally expensive, high-dimensional $\mathcal{K}$.

3.3 Detailed Algorithm

The details of our approach are presented in Algorithm 1. It takes as input the group size k, the set V of all neurons in the considered activation layer, and the pre-computed lower bounds $\mathcal{L}$ and upper bounds $\mathcal{U}$ of all neurons in the neural network. The algorithm outputs the set $\mathcal{G}$ of groups that will be sent for convex hull computation to generate multi-neuron constraints.

The algorithm first filters out all the *fixed* neurons (line 2), whose inputs have lower bounds greater than 0 or upper bounds less than 0. The filtering is performed by the function FILTER($\cdot$). All *unfixed* neurons constitute the set V'. Any group with size k consisting of the neurons in V' is a group candidate. All $\binom{|V'|}{k}$ group candidates are generated (line 3). For each neuron v_i in V', our algorithm iteratively selects groups $\mathcal{G}_i$ whose convex hulls are needed (lines 5–14). Each iteration starts with collecting group candidates containing v_i (line 5). Then for each group candidate G, the approximated volumes of its corresponding convex hull $\mathcal{K}_G$ are computed (lines 9–10), during which the octahedral over-approximation $\mathcal{P}_G$ is calculated by the function GETOCTAAPPR($\cdot$) provided in PRIMA. With the under- and over-approximations of $\mathrm{Vol}(\mathcal{K}_G)$, the algorithm decides whether the candidate G should be selected (line 12), according to the selecting strategy implemented in SELECT($\cdot$) that is detailed in Algorithm 2. Finally, all groups selected iteratively are accumulated together (line 14).

Algorithm 2 shows our selecting strategy given the current set $\mathcal{G}_i$ of selected groups and approximated volumes. When no group is selected (i.e. $\mathcal{G}_i = \emptyset$), the group candidate G is selected (lines 1–2). If the over-approximated volume is small enough, all selected groups can be abandoned and only G will be selected (lines 3–4). Correspondingly, if the under-approximated volume is large enough, G should be dropped (lines 5–6). Otherwise, G is added (lines 7–8).

Algorithm 1 Volume Approximation based Grouping (for a Single Layer)

Input: the target neural network $\boldsymbol{h}$, group size k, the set V of all neurons in the layer, pre-computed bounds $\mathcal{L}$ and $\mathcal{U}$ of all neurons in $\boldsymbol{h}$
Output: the set $\mathcal{G}$ of groups for convex hull computation

$\mathcal{G} \leftarrow \emptyset$
Select only unfixed neurons: $V' \leftarrow \text{FILTER}(V, \mathcal{L}, \mathcal{U})$
$\mathcal{GC} \leftarrow \text{GENCANDIDATE}(V', k)$
for all $v_i \in V'$ **do**
  Get group candidates involving v_i: $\mathcal{GC}_i \leftarrow \text{COLLECT}(v_i, \mathcal{GC})$
  Initialize selected groups involving v_i: $\mathcal{G}_i \leftarrow \emptyset$
  for all $G \in \mathcal{GC}_i$ **do**
    if $\underline{\text{Vol}}(\mathcal{K}_G)$ and $\overline{\text{Vol}}(\mathcal{K}_G)$ are not available **then**
      Compute octahedral over-approximation: $\mathcal{P}_G \leftarrow \text{GETOCTAAPPR}(G, \mathcal{L}, \mathcal{U}, \boldsymbol{h})$
      Volume approximation: calculate $\underline{\text{Vol}}(\mathcal{K}_G)$ and $\overline{\text{Vol}}(\mathcal{K}_G)$ by Theorem 1
    end if
    Update $\mathcal{G}_i$: $\mathcal{G}_i \leftarrow \text{SELECT}(\mathcal{G}_i, G, \underline{\text{Vol}}(\mathcal{K}_G), \overline{\text{Vol}}(\mathcal{K}_G))$
  end for
  $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{G}_i$
end for
return $\mathcal{G}$

Algorithm 2 Selecting Strategy $\text{SELECT}(\cdot)$

Input: the set $\mathcal{G}_i$ of currently selected groups for neuron v_i, group candidate G, approximated volumes $\underline{\text{Vol}}(\mathcal{K}_G)$ and $\overline{\text{Vol}}(\mathcal{K}_G)$
Output: the updated set $\mathcal{G}_i'$ of selected groups for neuron v_i

if $\mathcal{G}_i = \emptyset$ **then**
  $\mathcal{G}_i' \leftarrow \{G\}$
else if $\overline{\text{Vol}}(\mathcal{K}_G) \leq \max_{G' \in \mathcal{G}_i}(\underline{\text{Vol}}(\mathcal{K}_{G'}))$ **then**
  $\mathcal{G}_i' \leftarrow \{G\}$
else if $\underline{\text{Vol}}(\mathcal{K}_G) \geq \min_{G' \in \mathcal{G}_i}(\overline{\text{Vol}}(\mathcal{K}_{G'}))$ **then**
  $\mathcal{G}_i' \leftarrow \mathcal{G}_i$
else
  $\mathcal{G}_i' \leftarrow \mathcal{G}_i \cup \{G\}$
end if
return $\mathcal{G}_i'$

Intuitively, our selecting strategy attempts to select the groups with very small over-approximated volumes, as well as to eliminate the candidates with very large under-approximated volumes. It conservatively keeps the candidates otherwise, in order to prevent losing precision.

We give a detailed example to illustrate our algorithms.

Example 2. Assume that there are five neurons in the ReLU activation layer. That is, $V = \{v_1, v_2, v_3, v_4, v_5\}$. Let the group size $k = 2$, and the pre-computed bounds be $x_1 \in [-1, 1]$, $x_2 \in [-2, 2]$, $x_3 \in [-1, 1]$, $x_4 \in [-3, 3]$, and $x_5 \in [-\frac{1}{2}, \frac{1}{2}]$. All neurons are unfixed, so $V' = V = \{v_1, v_2, v_3, v_4, v_5\}$. All $\binom{5}{2} = 10$ group

candidates are

$$\begin{aligned}\mathcal{GC} = \{&\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\},\\ &\{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}\}.\end{aligned}$$

We start with $v_1 \in V'$. Collecting group candidates in $\mathcal{GC}$ that contain v_1, we have $\mathcal{GC}_1 = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\}\}$. $\mathcal{G}_1$ is then initialized as $\mathcal{G}_1 = \emptyset$. For all $G \in \mathcal{GC}_1$, the under- and over-approximated volumes are calculated:

– By function getOctaAppr$(\cdot)$ in Prima, the octahedral over-approximations are

$$\begin{aligned}\mathcal{P}_{\{v_1,v_2\}} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_1 + x_2 \le 3,\ -x_1 + x_2 \le 1,\ x_1 - x_2 \le 1,\\ & -x_1 - x_2 \le 3,\ -1 \le x_1 \le 1,\ -2 \le x_2 \le 2\ \},\\ \mathcal{P}_{\{v_1,v_3\}} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_1 + x_3 \le 2,\ -x_1 + x_3 \le \frac{1}{2},\ x_1 - x_3 \le \frac{1}{2},\\ & -x_1 - x_3 \le 2,\ -1 \le x_1 \le 1,\ -1 \le x_3 \le 1\ \},\\ \mathcal{P}_{\{v_1,v_4\}} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_1 + x_4 \le 4,\ -x_1 + x_4 \le 2,\ x_1 - x_4 \le 2,\\ & -x_1 - x_4 \le 4,\ -1 \le x_1 \le 1,\ -3 \le x_4 \le 3\ \},\\ \mathcal{P}_{\{v_1,v_5\}} = \{\ \boldsymbol{x} \in \mathbb{R}^2 \mid\ & x_1 + x_5 \le \frac{3}{2},\ -x_1 + x_5 \le \frac{1}{2},\ x_1 - x_5 \le \frac{1}{2},\\ & -x_1 - x_5 \le \frac{3}{2},\ -1 \le x_1 \le 1,\ -\frac{1}{2} \le x_5 \le \frac{1}{2}\ \}.\end{aligned}$$

– By Theorem 1, the approximated volumes are

$$\begin{aligned}\underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_2\}}) &= \frac{2}{3}, & \overline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_2\}}) &= 4,\\ \underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_3\}}) &= \frac{1}{6}, & \overline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_3\}}) &= 1,\\ \underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_4\}}) &= \frac{3}{2}, & \overline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_4\}}) &= 9,\\ \underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_5\}}) &= \frac{1}{24}, & \overline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_5\}}) &= \frac{1}{2}.\end{aligned}$$

Note that we invoke Qhull [3] to compute $\mathrm{Vol}(\mathcal{P})$ in Theorem 1.

From $\mathcal{G}_1 = \emptyset$, Algorithm 2 updates $\mathcal{G}_1$ for each $G \in \mathcal{GC}_1$ (line 12 in Algorithm 1):

– When $G = \{v_1, v_2\}$, since $\underline{\mathcal{G}_1} = \emptyset$, it is updated to $\mathcal{G}_1 = \{G\} = \{\{v_1, v_2\}\}$.
– When $G = \{v_1, v_3\}$, since $\overline{\mathrm{Vol}}(\mathcal{K}_G) = 1 \not\le \frac{2}{3} = \underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_2\}}) = \max_{G' \in \mathcal{G}_1}(\underline{\mathrm{Vol}}(\mathcal{K}_{G'}))$, and $\underline{\mathrm{Vol}}(\mathcal{K}_G) = \frac{1}{6} \not\ge 4 = \overline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_2\}}) = \min_{G' \in \mathcal{G}_1}(\overline{\mathrm{Vol}}(\mathcal{K}_{G'}))$, $\mathcal{G}_1$ is updated to $\{\{v_1, v_2\}\} \cup \{G\} = \{\{v_1, v_2\}, \{v_1, v_3\}\}$.
– When $G = \{v_1, v_4\}$, since $\overline{\mathrm{Vol}}(\mathcal{K}_G) = 9 \not\le \frac{2}{3} = \underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_2\}}) = \max_{G' \in \mathcal{G}_1}(\underline{\mathrm{Vol}}(\mathcal{K}_{G'}))$, and $\underline{\mathrm{Vol}}(\mathcal{K}_G) = \frac{3}{2} \ge 1 = \overline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_3\}}) = \min_{G' \in \mathcal{G}_1}(\overline{\mathrm{Vol}}(\mathcal{K}_{G'}))$, then G is dropped and $\mathcal{G}_1$ remains $\{\{v_1, v_2\}, \{v_1, v_3\}\}$.
– When $G = \{v_1, v_5\}$, since $\overline{\mathrm{Vol}}(\mathcal{K}_G) = \frac{1}{2} \le \frac{2}{3} = \underline{\mathrm{Vol}}(\mathcal{K}_{\{v_1,v_2\}}) = \max_{G' \in \mathcal{G}_1}(\underline{\mathrm{Vol}}(\mathcal{K}_{G'}))$, $\mathcal{G}_1$ is updated to $\{G\} = \{\{v_1, v_5\}\}$.

Finally, $\mathcal{G}_1 = \{\{v_1, v_5\}\}$.

Then $v_2 \in V'$ is considered. $\mathcal{GC}_2 = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}\}$. The process is similar. The obtained set of selected groups containing v_2 is $\mathcal{G}_2 = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_5\}\}$.

Similarly, we get $\mathcal{G}_3 = \{\{v_3, v_5\}\}$, $\mathcal{G}_4 = \{\{v_4, v_5\}\}$, and $\mathcal{G}_5 = \{\{v_1, v_5\}, \{v_2, v_5\}, \{v_3, v_5\}, \{v_4, v_5\}\}$ for the neurons v_3, v_4 and v_5, respectively.

Lastly, the set of selected groups for the whole activation layer is $\mathcal{G} = \bigcup_{i=1}^{5} \mathcal{G}_i = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_5\}, \{v_2, v_5\}, \{v_3, v_5\}, \{v_4, v_5\}\}$. As a result, our approach eliminates $|\mathcal{GC}| - |\mathcal{G}| = 10 - 6 = 4$ group candidates.

3.4 Generalization

Our approach is presented until now for the ReLU activation function. However, note that the ReLU function only plays a role when instantiating the intervals $\mathcal{D}^j$ and the bounds $\mathcal{B}^j$ required by SBLM (Section 2.2). The characteristics of the ReLU function are not necessary for our approach. Our approach can be generalized, as SBLM in [17], to the Sigmoid function $\sigma(x) = \frac{e^x}{e^x+1}$ and the Tanh function $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ by instantiating $\mathcal{D}^j$ and $\mathcal{B}^j$ differently.

Following [17], assuming $x \in [\underline{x}, \overline{x}]$, the intervals are instantiated as $\mathcal{D}^1 = (-\infty, c]$ and $\mathcal{D}^2 = [c, +\infty)$, where the constant c is chosen to minimize the area of the abstraction of a single neuron in the input-output plane. And the single-neuron abstraction of the activation function f (Sigmoid or Tanh) is defined by the bounds $\mathcal{B}^j = (a_j^{\leq}, a_j^{\geq})$ instantiated using the bounds from [21] as:

$$f(x) \leq a_j^{\geq}(x) = f(u_j) + (x - u_j) \cdot \begin{cases} \frac{f(u_j) - f(l_j)}{u_j - l_j}, & \text{if } u_j \leq 0, \\ \min(f'(u_j), f'(l_j)), & \text{otherwise,} \end{cases}$$

$$f(x) \geq a_j^{\leq}(x) = f(l_j) + (x - l_j) \cdot \begin{cases} \frac{f(u_j) - f(l_j)}{u_j - l_j}, & \text{if } l_j \geq 0, \\ \min(f'(u_j), f'(l_j)), & \text{otherwise,} \end{cases}$$

where l_j and u_j denote the lower and upper bounds of $\mathcal{D}^j \cap [\underline{x}, \overline{x}]$, respectively, and f' for the derivative of f. Using the above instantiation, our approach is applicable to the neural networks with Sigmoid and Tanh activation functions.

4 Experiments

We implement our approach as a neural network verifier FAGMR (**Fa**st **G**rouping for **M**ulti-neuron **R**elaxation) based on PRIMA, by replacing its grouping strategy with our volume approximation based approach. In this section, we evaluate the efficiency and effectiveness of FAGMR on common benchmarks. Specifically, we compare FAGMR against our baseline PRIMA in Section 4.2. It is further compared in Section 4.3 with two other state-of-the-art verifiers from the Verification of Neural Networks Competition (VNN-COMP) 2021 [2]: α,β-CROWN [28] and ERAN [23]. They are representatives of mainstream verification techniques. Section 4.4 shows the results of FAGMR on Sigmoid and Tanh neural networks.

Table 1. Neural networks used in the experiments. "Type" is the network structure: fully-connected (FC), convolutional (Conv), or Residual. "Training" is the used robust training technique: non-robust (NOR), PGD [14], or DiffAI [15].

Dataset	Model	Training	Type	Neurons	Layers	Activation
MNIST	5×100	NOR	FC	510	5	ReLU
	9×100	NOR	FC	810	9	ReLU
	6×200	NOR	FC	1010	6	ReLU
	6×500	NOR	FC	3000	6	Sigmoid
	6×500	NOR	FC	3000	6	Tanh
	ConvMed	PGD	Conv	3604	3	ReLU
	ConvMed	PGD	Conv	3604	3	Sigmoid
	ConvMed	PGD	Conv	3604	3	Tanh
	ConvBig	DiffAI	Conv	48064	6	ReLU
CIFAR-10	ConvSmall	DiffAI	Conv	3604	3	ReLU
	ConvMed	PGD	Conv	5703	3	ReLU
	ResNet2b	DiffAI	Residual	11364	13	ReLU

4.1 Experiment Configurations

Neural Networks. The evaluation is conducted on two widely-used datasets: MNIST and CIFAR-10. In particular, besides fully-connected and convolutional network structures, our evaluation includes a large ResNet network. Table 1 shows the details about all the networks used in our experiments. They are benchmarks from VNN-COMP 2021 [2] and can be downloaded at [23].

Robustness Property. As most works do, we consider the l_∞-norm perturbation on correctly classified inputs from the datasets. All verifiers are asked to answer verification problems as defined by Definition 1. More successfully verified problems mean better precision. In general, the perturbation thresholds θ are chosen as the same as those in [17].

Parameters of FAGMR. FAGMR chooses $k = 3$, which is claimed as the optimal option in PRIMA. That is, FAGMR considers size-3 neuron groups for convex relaxation.

Machine and Software. All experiments are conducted on a 12-core 2.20GHz Intel Xeon Sliver 4212 platform with 64GB memory. FAGMR is implemented in Python 3.7 and uses Gurobi 9.5.1 [10] for LP solving. QHULL [3] is employed to compute Vol($\mathcal{P}$) in Theorem 1.

4.2 Comparison of Verification Precision and Runtime

We first compare the verification precision and runtime of FAGMR against PRIMA, the state-of-the-art multi-neuron relaxation tool as well as the baseline

Table 2. Comparison of FaGMR against Prima. "Total" is the number of verification problems, θ for the perturbation threshold, "#group" for average number of selected groups, "time" for average runtime (in seconds), and "#verified" for the number of successfully verified problems. The best data are highlighted.

	Network	Total	θ	Prima-all			Prima-para			FaGMR		
				#group	time	#verified	#group	time	#verified	#group	time	#verified
MNIST	5×100	979	0.015	11.0	43.9	974	10.4	38.8	974	**8.7**	**33.5**	**974**
			0.026	45.6	52.8	970	37.4	47.0	970	**25.5**	**40.3**	**970**
	9×100	947	0.026	1.5k	96.8	891 (20)*	338.0	61.6	911	**138.8**	**57.6**	**911**
	6×200	972	0.015	1.6k	106.7	959	1.5k	68.6	958	**877.0**	**57.6**	**959**
	ConvMed	983	0.026	324.2	99.7	973 (2)	207.5	60.1	975	**108.4**	**54.8**	**975**
	ConvBig	929	0.03	366.4	114.7	922 (4)	259.3	**89.6**	924 (2)	**125.5**	96.9	**926**
CIFAR-10	ConvSmall	471	4/255	3.0k	49.0	452	2.3k	34.6	452	**1.4k**	**24.2**	**452**
	ConvMed	312	2/255	15.4k	649.5	288 (7)	2.4k	251.6	295	**1.8k**	**223.6**	**295**
	ResNet2b	161	1/255	-	-	-	7.0k	407.5	158	**1.1k**	**199.4**	**158**

* in parentheses is the number of out-of-memory cases.
\- results omitted due to lack of availability (over 50% out-of-memory cases).

that FaGMR is built on. For thorough evaluation, the comparison includes the following two variants of Prima discussed in [17]:

- Prima-all: It selects all possible k-neuron groups in an activation layer.
- Prima-para: It defines parameters n_s and s, partitions the neurons of an activation layer into sets of size n_s, and selects for each set a subset of all size-k groups that pairwise overlap by at most s. It is the default in Prima.

We use the suggested $k = 3$ for all three verifiers FaGMR, Prima-all, and Prima-para. For Prima-para, the default configuration ($n_s = 70$, $s = 1$) in its implementation is used. The three verifiers are compared on 8 different networks of various sizes. The results are shown in Table 2.

In Table 2, FaGMR successfully verifies the most problems in all cases, and outperforms Prima-all and Prima-para w.r.t. runtime in most cases. Particularly, observe that for the MNIST 6×200 network with $\theta = 0.015$, FaGMR verifies one more problem than Prima-para while being 16.0% ($\approx (68.6 - 57.6)/68.6$) faster, and it is 46.0% faster than Prima-all yet achieving the same verification precision. For the large network CIFAR-10 ResNet2b, FaGMR successfully verifies the same number of problems as Prima-para, but is much (51.1%) faster. On some complicated benchmarks, the MNIST ConvBig with $\theta = 0.03$ for instance, both Prima-all and Prima-para run out of memory in several problems, but FaGMR does not, because it selects a much less number of groups for convex relaxation. All the benchmarks demonstrate that FaGMR effectively reduces the number of neuron groups. As a result, FaGMR reduces the verification time by 11.2% on average compared to Prima-para, but with negligible precision loss compared to Prima-all. The comparison confirms the effectiveness and efficiency of FaGMR.

Details of Runtime. The efficiency improvement of FaGMR comes from selecting less number of neuron groups, which decreases the time cost for both

Table 3. Detailed runtime of FAGMR and PRIMA (in seconds). T_{gr} is the time for neuron grouping, T_{cg} the time for multi-neuron constraint generation, T_{lp} the time for LP solving.

Network	θ	PRIMA-all			PRIMA-para			FAGMR		
		T_{gr}	T_{cg}	T_{lp}	T_{gr}	T_{cg}	T_{lp}	T_{gr}	T_{cg}	T_{lp}
MNIST 5×100	0.015	0.4	7.7k	34.8k	0.4	5.0k	32.6k	132	**2.2k**	**30.0k**
	0.026	0.4	9.0k	41.1k	0.4	6.4k	39.0k	350	**2.8k**	**35.8k**
CIFAR-10 ConvSmall	2/255	1.4	3.0k	2.6k	1.4	3.0k	2.6k	2.7	**2.1k**	**2.3k**
	4/255	2.3	15.0k	6.8k	2.3	9.9k	5.1k	9.8	**4.5k**	**3.2k**

constraint generation and solving. We quantitatively analyze the influence of group reduction by dividing the runtime into three parts:

- T_{gr}: time for grouping;
- T_{cg}: time for constraint generation;
- T_{lp}: time for LP solving.

Table 3 shows the detailed runtime for some benchmarks. All the time is the total time spent on all verification problems. We can see that FAGMR spends more time for neuron grouping (T_{gr}) than both PRIMA-all and PRIMA-para, due to the volume approximation and comparison. However, the grouping time is negligible compared to the time for multi-neuron constraint generation (T_{cg}) and LP solving (T_{lp}). The increased grouping time of FAGMR brings significant efficiency improvements in both constraint generation and LP solving. For instance, in the MNIST 5×100 network with $\theta = 0.015$, $T_{gr} + T_{cg}$ for FAGMR is 70.0% ($\approx (7.7 - 2.332)/7.7$) and 53.4% less than PRIMA-all and PRIMA-para, respectively. And the total runtime ($T_{gr} + T_{cg} + T_{lp}$) of FAGMR is 24.0% and 14.0% faster than PRIMA-all and PRIMA-para, respectively.

Notice that for the CIFAR-10 ConvSmall network with $\theta = 2/255$ in Table 3, PRIMA-all and PRIMA-para have the same T_{gr}, T_{cg} and T_{lp}. It is because there are only a few number of unfixed neurons that need to be grouped in such verification problems with the small perturbation threshold, making the statically chosen grouping parameters of PRIMA-para ineffective. Our dynamic grouping strategy in FAGMR however avoids this to some extent, yielding respectively 30.0% and 11.5% efficiency improvement in T_{cg} and T_{lp}.

4.3 Comparison with Other State-of-the-Art Verifiers

We have shown the effectiveness and efficiency of FAGMR compared to its baseline PRIMA, the state-of-the-art multi-neuron relaxation verifier. In this section, FAGMR is further compared against α,β-CROWN [28] and ERAN [23], two outstanding verifiers in VNN-COMP 2021 [2]. For α,β-CROWN, we use its default configuration, under which it performs complete verification using Branch

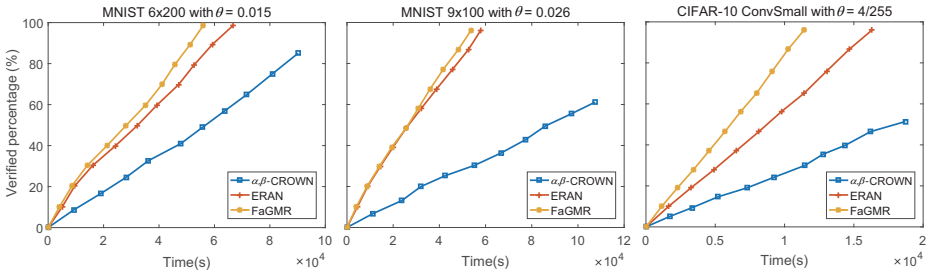

Fig. 2. Verified Percentage of FaGMR, α,β-CROWN and ERAN w.r.t. Runtime. Three benchmarks consist of 972, 947, and 471 verification problems, respectively.

and Bound (BaB). For ERAN, we use its multi-neuron relaxation technique with default $n_s = 70$ and $s = 1$. The detailed settings are as follows.

- α,β-CROWN: BaB verifier with 20 β-CROWN iterations; CPU mode; time-out for each task being 300 seconds; MILP refinement enabled; PGD attack disabled. The configuration files for different networks can be found at its repository[1] with names `network_name.yaml`.
- ERAN: incomplete verifier with `refinepoly` domain; $n_s = 70$ and $s = 1$ for multi-neuron relaxation; CPU mode; MILP disabled. The configuration file is available here.[2]

Figure 2 shows the comparison results on three neural networks. The y-axes stand for the verified percentage and the x-axes for the runtime. The figure illustrates that the verified percentage increases as the time passes. The fact that a trend line is steeper indicates that its corresponding verifier is more efficient.

Figure 2 demonstrates that α,β-CROWN eventually verifies the least verification problems among the three tools on each network. It is mostly because α,β-CROWN utilizes single-neuron relaxation techniques that inherently lead to lower precision than multi-neuron relaxation leveraged by both FaGMR and ERAN. FaGMR achieves similar verification precision to ERAN on these three benchmarks. Precisely, it successfully verifies one more problem than ERAN on the MNIST 6×200 network when $\theta = 0.015$, and exactly the same number of problems on the other two networks. For efficiency, we can easily see that FaGMR outperforms the other two tools on all three networks.

The experimental results show that FaGMR is also competitive among the state-of-the-art verifiers using different techniques.

4.4 Comparison on Sigmoid and Tanh Neural Networks

FaGMR is applicable to the neural networks with activation functions Sigmoid and Tanh as presented in Section 3.4. In this section, we compare FaGMR against

[1] https://github.com/Verified-Intelligence/alpha-beta-CROWN/tree/main/complete_verifier/exp_configs/

[2] https://github.com/formes20/FaGMR/blob/main/code/tf_verify/config.py

Table 4. Comparison of FAGMR against PRIMA-para on Sigmoid and Tanh networks. "Total" is the number of verification problems, θ for the perturbation threshold, "#group" for average number of selected groups, "time" for average runtime (in seconds), and "#verified" for the number of successfully verified problems. The best data are highlighted.

	Network	Total	Activation	θ	PRIMA-para			FAGMR		
					#group	time	#verified	#group	time	#verified
MNIST	6×500	95	Sigmoid	0.012	6.2k	640.9	95	**3.8k**	**570.0**	**95**
	ConvMed	99	Sigmoid	0.014	6.3k	255.6	99	**5.9k**	**252.5**	**99**
	6×500	99	Tanh	0.005	406.6	211.8	97 (1)*	**184.8**	**194.9**	**98**
	ConvMed	98	Tanh	0.005	1.2k	220.4	98	**446.3**	**191.8**	**98**

* in parentheses is the number of out-of-memory cases.

PRIMA-para on such networks. PRIMA-all is excluded due to out-of-memory issues on these activation functions. Specifically, we choose two MNIST networks 6×500 and ConvMed with Sigmoid and Tanh activation functions (see Table 1), then conduct the comparison experiments similar to Section 4.2.

Similarly, Table 4 shows that FAGMR outperforms PRIMA-para on these networks with activation functions Sigmoid and Tanh. Specifically, by reducing the number of selected groups, FAGMR is on average 8.3% faster than PRIMA-para. In particular, it successfully verifies one more problem than PRIMA-para on the 6×500 Tanh network when $\theta = 0.005$ due to the latter's out-of-memory issue, while obtaining the same precision on the other three benchmarks.

5 Related Work

According to the completeness of verification results, neural network verification methods can be categorized into complete methods and incomplete ones.

Complete verification. Complete verification methods can describe the exact behavior of a neural network w.r.t. an input region. They can be further classified into: (i) SAT/SMT based methods [11,12,7], which encode the verification problem into an SAT/SMT query; (ii) Mixed-integer linear programming (MILP) based methods [7,1,5], which encode the verification problem into an MILP problem and then invoke MILP solvers; (iii) Branch-and-Bound based methods, which split non-linear activation functions into linear pieces [28,25,6,27], or split the input region to be small enough so that the neural network behaves linearly on each input sub-region [26]. Complete methods are usually limited in scalability because of the computational complexity.

Incomplete verification. Incomplete verification methods often relax non-linear activations such as ReLU to speed up verification. Most incomplete methods attempt to develop efficient and precise over-approximations for a given activation function (e.g. [20,21,29,18,28,33], see a survey in [13]). FAGMR belongs

to this category. It applies convex relaxation to over-approximate non-linear activation functions, hence is incomplete but is faster and more scalable than complete methods. Among incomplete verification methods, single-neuron convex relaxation based methods consider each neuron separately and over-approximate its activation function. They are very efficient (e.g., [20,21,29,18,28,33,32,31,9]), but their verification precision is proved to be limited by a convex relaxation barrier [18]. Instead, multi-neuron convex relaxation methods [19,24,17] suggest over-approximating multiple neurons jointly for higher precision, hence breaking down the barrier faced by single-neuron relaxation. Our work follows this promising direction and improves the efficiency of existing techniques.

6 Conclusion

We have presented a fast and effective neuron grouping strategy for multi-neuron convex relaxation methods. The key idea is to compute the volume approximations of all possible neuron group candidates and then select better groups according to the approximated volumes. The evaluation shows that our approach effectively reduces verification time and achieves competitive verification precision compared to the state-of-the-art tools.

Acknowledgements. We thank all the anonymous reviewers and Gagandeep Singh for their invaluable comments and suggestions.

References

1. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Math. Program. **183**(1), 3–39 (2020), https://doi.org/10.1007/s10107-020-01474-5
2. Bak, S., Liu, C., Johnson, T.T.: The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. CoRR **abs/2109.00498** (2021), https://arxiv.org/abs/2109.00498
3. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: Qhull: Quickhull algorithm for computing the convex hull. Astrophysics Source Code Library pp. ascl–1304 (2013)
4. Betke, U., Henk, M.: Approximating the volume of convex bodies. Discrete & Computational Geometry **10**, 15–21 (1993)
5. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of ReLU-based neural networks via dependency analysis. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. pp. 3291–3299. AAAI Press (2020), https://ojs.aaai.org/index.php/AAAI/article/view/5729
6. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. J. Mach. Learn. Res. **21**, 42:1–42:39 (2020), http://jmlr.org/papers/v21/19-468.html

7. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Kumar, K.N. (eds.) Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10482, pp. 269–286. Springer (2017), https://doi.org/10.1007/978-3-319-68167-2_19
8. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016), http://www.deeplearningbook.org/
9. Goubault, E., Palumby, S., Putot, S., Rustenholz, L., Sankaranarayanan, S.: Static analysis of ReLU neural networks with tropical polyhedra. In: Dragoi, C., Mukherjee, S., Namjoshi, K.S. (eds.) Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12913, pp. 166–190. Springer (2021). https://doi.org/10.1007/978-3-030-88806-0_8
10. Gurobi Optimization: Gurobi Optimizer, http://www.gurobi.com
11. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 97–117. Springer (2017), https://doi.org/10.1007/978-3-319-63387-9_5
12. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The Marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 443–452. Springer (2019), https://doi.org/10.1007/978-3-030-25540-4_26
13. Liu, C., Arnon, T., Lazarus, C., Strong, C.A., Barrett, C.W., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. Found. Trends Optim. **4**(3-4), 244–404 (2021), https://doi.org/10.1561/2400000035
14. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. CoRR **abs/1706.06083** (2017), http://arxiv.org/abs/1706.06083
15. Mirman, M., Gehr, T., Vechev, M.T.: Differentiable abstract interpretation for provably robust neural networks. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 3575–3583. PMLR (2018), http://proceedings.mlr.press/v80/mirman18b.html
16. Morgulis, N., Kreines, A., Mendelowitz, S., Weisglass, Y.: Fooling a real car with adversarial traffic signs. CoRR **abs/1907.00374** (2019), http://arxiv.org/abs/1907.00374
17. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.T.: 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
18. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information

Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 9832–9842 (2019)
19. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 15072–15083 (2019), https://proceedings.neurips.cc/paper/2019/hash/0a9fdbb17feb6ccb7ec405cfb85222c4-Abstract.html
20. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 10825–10836 (2018), https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html
21. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 41:1–41:30 (2019), https://doi.org/10.1145/3290354
22. Sommerville, D.M.: Introduction to the Geometry of N Dimensions. Courier Dover Publications (2020)
23. SRI Lab: ETH robustness analyzer for neural networks (ERAN) (2022), https://github.com/eth-sri/eran
24. Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., Patel, K., Vielma, J.P.: The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/f6c2a0c4b566bc99d596e58638e342b0-Abstract.html
25. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 6369–6379 (2018), https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html
26. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Enck, W., Felt, A.P. (eds.) 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. pp. 1599–1614. USENIX Association (2018), https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi
27. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 29909–29921 (2021), https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html

28. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021), https://openreview.net/forum?id=nVZtXBI6LNn
29. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 4944–4953 (2018)
30. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. IEEE Trans. Software Eng. **48**(2), 1–36 (2022)
31. Zhao, Z., Zhang, Y., Chen, G., Song, F., Chen, T., Liu, J.: CLEVEREST: accelerating CEGAR-based neural network verification via adversarial attacks. In: Singh, G., Urban, C. (eds.) Static Analysis - 29th International Symposium, SAS 2022, Auckland, New Zealand, December 5-7, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13790, pp. 449–473. Springer (2022). https://doi.org/10.1007/978-3-031-22308-2_20
32. Zheng, Y., Liu, J., Shi, X.: MpBP: verifying robustness of neural networks with multi-path bound propagation. In: Roychoudhury, A., Cadar, C., Kim, M. (eds.) Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022. pp. 1692–1696. ACM (2022). https://doi.org/10.1145/3540250.3558924
33. Zheng, Y., Shi, X., Liu, J.: Multi-path back-propagation method for neural network verification (in Chinese). Ruan Jian Xue Bao/Journal of Software **33**(7), 2464–2481 (2022), http://www.jos.org.cn/1000-9825/6585.htm