Boosting Multi-Neuron Convex Relaxation for Neural Network Verification

No Author Given

No Institute Given

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 with state-of-the-art tools on neural networks trained by MNIST and CIFAR-10. The experimental results demonstrate that FAGMR is more efficient than the state-of-the-art works, yet with better precision in most of the cases.

Keywords: Neuron network verification · Multi-neuron relaxation · Volume approximation.

1 Introduction

The increasing adoption of neural networks in many safety-critical scenarios have 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 of a neural network. For example, forged traffic signs can fool certain autonomous driving systems [17]. They look almost the same as humans, yet making auto-driving systems output incorrect predictions, hence leading to unexpected behaviors.

A large body of research aims to find adversarial examples based on testing (see a survey [31]). 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, and 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 the 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 *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 [21,22,30,29,33]. These methods do not capture the interdependencies 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 [20,25,18]. 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 their 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, the NP-hardness of the convex hull computation problem limits the number of groups to be selected. For instance, adopting an exact convex hull computing algorithm, kPoly [20] 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 [18] 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 concerning 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 although they perform well in specific cases. They may not perform equally well on different verification problems, even on different activation layers in the same neural 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 computation. 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 [18], $\alpha\beta$CROWN [29] and ERAN [24] on neural networks trained by the widely-used datasets MNIST and CIFAR-10. The experimental results show that FaGMR is faster than Prima, $\alpha\beta$CROWN, and ERAN, by spending on average 18.8%, 38.5%, and 23.1% less verification time respectively. Meanwhile, FaGMR successfully verifies much the same numbers of verification problems as Prima and ERAN, and 30.1% more than $\alpha\beta$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.

Organization. Section 2 gives an overview of our approach. Section 3 recalls necessary backgrounds on neural network verification and the multi-neuron relaxation method. Our approach is detailed in Section 4 and evaluated in Section 5. Related works are discussed in Section 6. We conclude our presentation in Section 7.

2 Overview

Figure 1 illustrates a simple workflow of FaGMR. To accurately verify the property φ of a neural network $\boldsymbol{h}$, FaGMR employs multi-neuron relaxations in each intermediate layer to obtain more precise constraints. To reduce the extensive computation of convex hull involved in multi-neuron relaxations, FaGMR pick out the groups with low-dimensional computation. Then inputs these chosen groups to the existing implementation for their respective multi-neuron relaxation via high-dimensional computation. The intersection of all the multi-neuron relaxations from selected groups in one layer provides the constraints for the following layers.

Notations. We reserve lowercase Latin and Greek letters $a, b, x, \ldots, \lambda, \ldots$ for scalars, bold $\boldsymbol{a}$ for vectors or coordinate points, capitalized bold $\boldsymbol{A}$ for matrices, and calligraphic $\mathcal{A}$ or blackboard bold $\mathbb{A}$ for sets. Similarly, scalar functions are denoted as $f : \mathbb{R}^d \to \mathbb{R}$. Given n elements, the number of k-combinations is denoted by $\binom{n}{k}$. We call a set of neurons a group. Mark $\{0, 1, ..., n-1\}$ as $[\![n]\!]$.

2.1 Neural Network Verifier

The property φ discussed in this paper is defined as $\varphi := \arg\max_j \boldsymbol{h}(\boldsymbol{x})_j = \arg\max_j \boldsymbol{h}(\boldsymbol{x}')_j$, where $\boldsymbol{x}' \in \mathbb{B}$ and $\boldsymbol{x}$ is the input of neural network $\boldsymbol{h}$. Most state-of-the-art methods aim to verify φ by computing the convex approximation of the output $\boldsymbol{h}(\boldsymbol{x})$ within the input domain $\mathbb{B}$. If the verifier can prove φ holds for any $\boldsymbol{x}'$ within $\mathbb{B}$, it is regarded as successful verification, otherwise, the verifier fails. However, various verifiers present different trade-offs between approximation time and precision, and are sensitive to hyperparameters settings.

FAGMR excels in computing tight convex approximations while automatically balancing time and precision for $\boldsymbol{h}(\boldsymbol{x})$.

2.2 Illustrative Example

We consider 2-neuron relaxations with multiple neurons connected in a fully connected layer, denoted as $x_i, x_j, ..., x_k$, as shown in Figure 1 and y_i implies the output of activation function for x_i . We introduce the group s, such as $s = \{x_i, x_j\}$. Each layer produces a set of constraints from each group, constraining the output of the next ReLU layer until the final layer. Then, the verifier examines if φ is satisfied. In addition, we use ReLU as the activation function in this example, but FAGMR can also work with other activation functions.

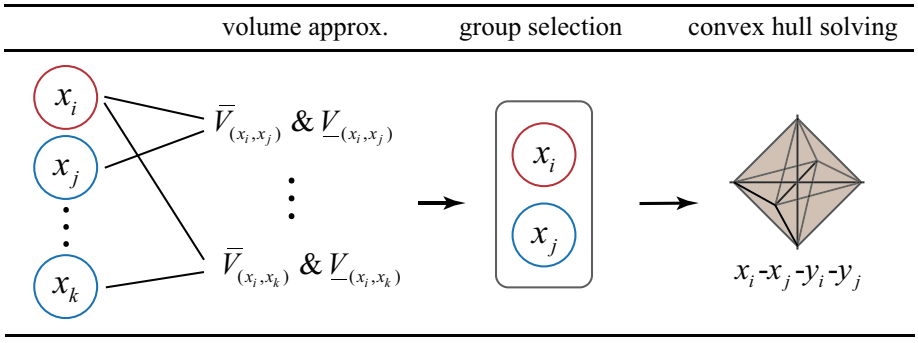

Fig. 1. Illustration of FAGMR finding a partner neuron for x_i and send them to compute the four-dimensional convex hull

Multi-neuron Relaxation The verifiers using k-neuron relaxation to compute bounds, like KPOLY and PRIMA, should construct a k-dimensional polyhedron $\mathcal{P}$

at first. Then use split, extend, bound, and lift operations, which is called as Split-Bound-Lift Method(SBLM) in PRIMA, to obtain a $2k$-dimensional polyhedron $\mathcal{C}$ which is regarded as the output constraints in KPOLY. $\mathcal{C}$ is well used to bound the outputs from the subsequent activation layers.

Formation of $\mathcal{P}$. Generally speaking, any method guaranteeing the input polytope $\mathcal{P}$ is a convex polytope for multi-neuron relaxation is feasible. We give one of the examples in this section, the bound of x_i and x_j are $[-1, 1]$. Next, apply addition and subtraction between these inequalities mutually, an example of doing addition is $\{x_i + x_j \leq \max([-1, 1] + [-1, 1]) = 2\}$, as shown in Figure. 2(a). Then, add bounds of x_i and x_j to the constructed constraints, and the input k-dimensional polyhedron $\mathcal{P}$ can be generated as the blue part shown in Figure. 2(b).

Formation of $\mathcal{C}$. Assume that we start with splitting from x_i dimension. Specifically, we split polyhedron $\mathcal{P}$ into two parts by halfspaces $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_i \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \mid x_i \leq 0\}$. The resulting parts are $\{\boldsymbol{x} \in \mathbb{R}^2 | \boldsymbol{x} \in \mathcal{P} \wedge x_i \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 | \boldsymbol{x} \in \mathcal{P} \wedge x_i \leq 0\}$. Next, we extend these two parts to $(k+1)$ dimension by adding a new dimension y_i to $\boldsymbol{x}$. The extension can be regarded as a track of the parallel motion along y_i axis by $\mathcal{P}$. The results by extending are presented as $\{\boldsymbol{x} \in \mathbb{R}^3 | \boldsymbol{x} \in \mathcal{P} \wedge x_i \geq 0 \wedge y_i \in \mathbb{R}\}$ and $\{\boldsymbol{x} \in \mathbb{R}^3 | \boldsymbol{x} \in \mathcal{P} \wedge x_i \leq 0 \wedge y_i \in \mathbb{R}\}$. Note that $\mathcal{P}$ is put into the 3-dimensional space. We then use halfspace depend-

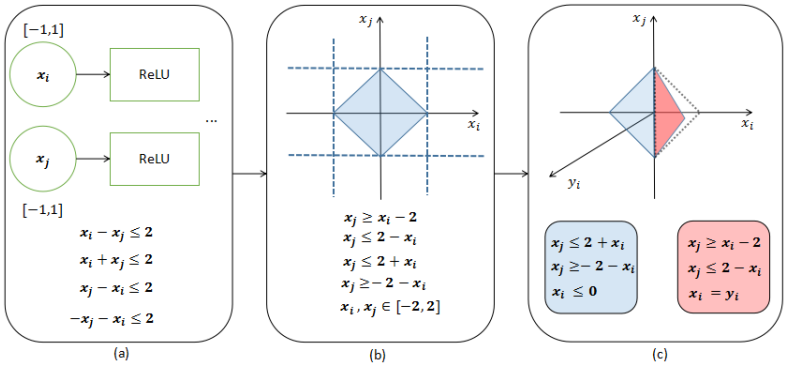

Fig. 2. The formation process for $\mathcal{C}_s$ and $\mathcal{P}_s$ in ReLU network.

.

ing on activation functions to bound the extensions. For the extension with $\{x_i \geq 0\}$, we use halfspace $\{y_i = x_i\}$ to bound it. For the one with $\{x_i \leq 0\}$, use $\{y_i = 0\}$ for binding. $\{\boldsymbol{x} \in \mathbb{R}^3 | \boldsymbol{x} \in \mathcal{P} \wedge x_i \geq 0 \wedge y_i = x_i\}$(the red part in Figure. 2(c)) and $\{\boldsymbol{x} \in \mathbb{R}^3 | \boldsymbol{x} \in \mathcal{P} \wedge x_i \leq 0 \wedge y_i = 0\}$(the blue one in Figure. 2(c)) are the resulting parts. Finally, we compute the convex hull of the two parts and get a 3-dimensional convex polyhedron, this step is called a lift. Deal with x_j

dimension in the same way based on the generated 3-dimensional polyhedron, then generate a 4-dimensional polyhedron $\mathcal{C}$. All of the $\mathcal{C}$ from different groups in one layer are taken as an intersection as the constraints for the following layers.

kPoly and Prima directly get $2k$-dimenisonal polyhedra respectively in each split space of $\mathcal{P}_s$ through the split, extend, bound, and lift operations, then convex them together to compute $\mathcal{C}_s$. Meanwhile, for the convenience of the following proof, the process described above uses split, lift, and bound operations simultaneously for each split space of $\mathcal{P}_s$, adding dimension from k to $2k$ step by step. They are equivalent to each other and the proof is given in Theorem. 2.

In Prima, a state-of-the-art depended on kPoly selects groups based on hyperparameters and uses the PDDM method to simplify the constraints, however, the parameters may not always be suitable for different networks and may sacrifice precision. FaGMR aims at automatically selecting some of the groups before computing high-dimensional constraints in PDDM for different networks to save time while maintaining precision.

Motivation The problem with the classical grouping strategy lies in that it produces redundant groups for computing the convex hulls, i.e., a multi-neuron relaxation from a group may contain the intersection of the relaxations from other two or more groups.

Assume that there are three groups: $s_0 = \{x_i, x_j\}$, $s_1 = \{x_i, x_k\}$ and $s_2 = \{x_j, x_k\}$. We observe a situation that the $\mathcal{C}_0$ may contain the intersection of the $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, then the s_0 is regarded as the redundant group. For example, The output constraints $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ can be expressed as:

$$
\begin{array}{lll}
\mathcal{C}_0 = \{x \in \mathbb{R}^6 \mid & \mathcal{C}_1 = \{x \in \mathbb{R}^6 \mid & \mathcal{C}_2 = \{x \in \mathbb{R}^6 \mid \\
-x_i + x_j - y_i - y_j \leq 0 & -x_k - y_k \leq -2 & x_j - x_k - y_j - y_k \leq 0 \\
x_i - x_j + y_i - y_j \leq 2 & x_k + y_k \leq 2 & x_j - x_k + y_j - y_k \leq 2 \\
x_i + x_j - y_i - y_j \leq 4 & x_i - y_i \leq -1 & x_j + x_k - y_j - y_k \leq 4 \\
x_i + y_i + y_j \leq 2 & x_i + y_i \leq 1 & x_j + y_j + y_k \leq 2 \\
-x_i + x_j - y_j \leq 4 & x_i - x_k - y_k \leq 0 & -x_j + x_k - y_k \leq 4 \\
x_i + x_j - y_j \leq 0 & -x_i + x_k + y_k \leq 0 & x_j + x_k - y_k \leq 0 \\
y_j \leq 1 & x_i - y_i + y_k \leq -2 & y_k \leq 1 \\
y_i \leq 1 & x_i - y_i - y_k \leq 2 & y_j \leq 1 \\
x_i - y_i \leq 1 \} & x_i + x_k + y_i + y_k \leq 0.5 \} & x_j - y_j \leq 1 \}
\end{array}
$$

where $-1 \leq x_i, x_k \leq 1, -2 \leq x_j \leq 2$. One can verify that $\mathcal{C}_0$ contains the intersection of $\mathcal{C}_1$ and $\mathcal{C}_2$. We find that for groups containing one neuron x_i, the redundancy maybe the one with a bigger volume of its $\mathcal{C}$. Additionally, considering the intervals, which can reflect the volume to some extent, maybe a feasible way to select groups, but this may overlook the information in the $\mathcal{P}$ and $\mathcal{C}$.

However, it is unavoidable for high-dimensional calculation for picking out such redundancy precisely, which will spend a lot of time. Because such groups

always have a large value of volume, we turn to compute the approximation of volume through the lower dimension to discard redundant groups as much as possible, the groups discarded are also called worse groups in the following.

FaGMR can efficiently identify the worse groups before solving the convex hull problems. As a result, it produces neuron groups that have fewer redundant groups while considering the multi-neuron relaxation to preserve precision.

FaGMR Algorithm The key idea behind FaGMR is *fast computing the volume approximations of all possible k-neuron groups.* If the volume approximation for the group $\{x_i, x_j\}$ is smaller than the ones for any other groups', such as $\{x_i, x_k\}$, then we choose $\{x_i, x_j\}$ as the selected group. Figure. 1 shows the workflow of FaGMR finding a partner for neuron x_i. It has the following three steps:

Volume approximation. Computing the precise volume of a $(2k)$-dimensional convex hull formed by a k-neuron group can be time-consuming (i.e., the volume of the convex hull x_i-x_j-y_i-y_j in Figure. 1). Therefore, we compute the approximation method in Section 4.1 to under and over-approximate the volumes of convex hulls for each group, in order to select groups which will be computed a $(2k)$-dimensional convex hull. The volume approximation method used in FaGMR is based on Betke et al. [5], and we improve the over-approximation by using the volume of $\mathcal{P}$ to make the approximation tighter. Our approximation is computed on k-dimension(particularly in computation for the volume of $\mathcal{P}$ by existing volume tools) and is used to reduce some of the $(2k)$-dimensional computations.

Neurons grouping selection. Once all under and over-approximated volumes have been computed for the groups, select the groups for each neuron, like x_i, whose over-approximation volumes are smaller than most of the under-approximation volumes of other groups. Our strategy prefers selecting the groups which avoid the phenomenon in our motivation while maintaining groups as many as possible to guarantee precision and saving time as possibly as possible. More details can be found in Section 4.2.

Convex hull solving. Once the groups of each neuron are determined, they are sent to derive their exact $(2k)$-dimensional convex hull, like $\{x_i, x_j\}$ in Figure. 1. We use the solver in Prima [18] to accomplish this task. It provides worst-case time-complexity convex hull computation and guarantees exactness, making it a suitable choice for our work. Using the intersection of these computed convex hulls, the bounds of y_i and y_j can be obtained by a linear programming(LP) solver. With these bounds for the neurons in a layer, FaGMR can then determine the groups for the subsequent layer.
In Section 3, we introduce the formation process for multi-neuron convex relaxation, which is used in Section 4.1, where we present formulas for under and over-approximation.

3 Preliminaries

This section reviews some notions needed for understanding our approach.

3.1 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 $\boldsymbol{f}_i(\boldsymbol{x})$:

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

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. $c = \arg\max_j \boldsymbol{h}(\boldsymbol{x})_j$.

Neural network verification problem commonly needs to verify the *robustness* property. A robust neural network must satisfy the smoothness assumption [11], 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 a p-norm ball neighborhood of $\boldsymbol{x}$:

$$\mathbb{B}^p_\varepsilon = \{\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta} \mid \|\boldsymbol{\delta}\|_p \leq \varepsilon\}$$

where ε 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 ε, a* verification problem *is to give the true value of the following statement:*

$$\arg\max_j \boldsymbol{h}(\boldsymbol{x})_j = \arg\max_j \boldsymbol{h}(\boldsymbol{x}')_j,$$
$$\textit{for each } \boldsymbol{x}' \in \mathbb{B}^p_\varepsilon(\boldsymbol{x}).$$

If the above statement is true, then we can conclude that the neural network $\boldsymbol{h}$ is robust with respect to the input $\boldsymbol{x}$ against the perturbation threshold ε.

3.2 Formation for multi-neuron relaxation

The formation from kPoly is utilized in both of Prima and FaGMR to generate output constraints $\mathcal{C}_s$. Formally, for a k-dimensional input polytope $\mathcal{P}_s = \left\{\boldsymbol{x} \in \mathbb{R}^k \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\right\}$ generated by a set of k neurons $s \subseteq \mathcal{S}$, $\mathcal{S}$ is the set of all neurons, the input polytope $\mathcal{P}_s$ will generate a higher-dimensional polyhedron $\mathcal{C}_s \subseteq \mathbb{R}^{2k}$. In this section, we introduce the formation and definition for $\mathcal{C}_s$.

The input polytope $\mathcal{P}_s$ can be constructed by any method which uses the relaxation among neurons in group s to guarantee $\mathcal{P}_s$ is a convex polytope, with linear inequations for the convenience of LP solver, like the process in Prima.

Formation of multi-neuron relaxation $\mathcal{C}_s$ Among the formation of $\mathcal{C}_s$, we have four main steps: split, extend, bound, and lift. We start by splitting space from x_i dimension. Specifically, we split $\mathcal{P}_s$ into two parts with halfspaces $t_i^+ = \{\boldsymbol{x} \in \mathbb{R}^k \mid x_i \geq c\}$ and $t_i^- = \{\boldsymbol{x} \in \mathbb{R}^k \mid x_i \leq c\}$(start from $\forall x_i \in s$ is valid, and c is a constant [18]). The resulting parts are $\mathcal{P}_s \cap t_i^{\{+,-\}}$ respectively. Next, we extend $\mathcal{P}_s \cap t_i^{\{+,-\}} \subseteq \mathbb{R}^k$ to $(k+1)$ dimension by adding a new dimension to $\boldsymbol{x}$, marked as y_i, and defining the extension as $e(\mathcal{P}_s^{(i)} \cap t_i^{\{+,-\}}) \subseteq \mathbb{R}^{k+1}$. Then we use halfspaces $\mathcal{B}_i^{\{+,-\}} = \left\{\boldsymbol{x} \in \mathbb{R}^{k+1} | y_i = u_i^{\{+,-\}}(x_i) \cup y_i = l_i^{\{+,-\}}(x_i), \forall \boldsymbol{x} \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+,-\}})\right\}$ to bound the extensions. For ReLU, $u_i^{\{+,-\}}(\cdot) = l_i^{\{+,-\}}(\cdot)$. The resulting parts are defined as $e(\mathcal{P}_s^{(i)} \cap t_i^+) \cap \mathcal{B}_i^+$ and $e(\mathcal{P}_s^{(i)} \cap t_i^-) \cap \mathcal{B}_i^-$, when the activation function is $f(\cdot)$, $u_i^{\{+,-\}}(\cdot)$ and $l_i^{\{+,-\}}(\cdot)$ are four linear functions satisfying $l_i^{\{+,-\}}(x_i) \leq f(x_i) \leq u_i^{\{+,-\}}(x_i), \forall x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+,-\}})$. Finally, compute the convex hull of the two parts and get a $(k+1)$-dimensional convex polyhedron, which is the lift step. Deal with other dimensions in $\mathcal{P}_s$ in the same way until generating a $(2k)$-dimensional polyhedron $\mathcal{C}_s$, shown by Definition. 2.

Definition 2. *For k selected inputs $x_0, x_1, ..., x_{k-1}$, their multi-neuron relaxation formed from Section 3.2 is defined as $\mathcal{C}_s$ where $s = \{x_0, x_1, ..., x_{k-1}\}$. $conv(\cdot)$ implies the convex hull for the set of points. There has an equation according to the process in Section 3.2:*

$$\begin{aligned} \mathcal{P}_s^{(i+1)} = conv(&e(\mathcal{P}_s^{(i)} \cap t_i^+) \cap \mathcal{B}_i^+ \\ &\cup e(\mathcal{P}_s^{(i)} \cap t_i^-) \cap \mathcal{B}_i^-), \forall i \in [\![k]\!] \end{aligned} \tag{1}$$

where:

$$\mathcal{P}_s^{(0)} = \{\boldsymbol{x} \in \mathbb{R}^k | \boldsymbol{x} \in \mathcal{P}_s\}; \mathcal{P}_s^k = \mathcal{C}_s$$

$$t_i^+ = \{\boldsymbol{x} \in \mathbb{R}^{k+i} | x_i \geq c\}; t_i^- = \{\boldsymbol{x} \in \mathbb{R}^{k+i} | x_i \leq c\}$$

$$\mathcal{B}_i^{\{+,-\}} = \left\{\boldsymbol{x} \in \mathbb{R}^{k+1} | y_i = u_i^{\{+,-\}}(x_i), y_i = l_i^{\{+,-\}}(x_i), \forall \boldsymbol{x} \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+,-\}})\right\}$$

$$e(\mathcal{D}) : \mathbb{R}^i \leftarrow \mathbb{R}^i \times \mathbb{R}, \mathcal{D} = \{\boldsymbol{x} | \boldsymbol{x} \in \mathbb{R}^i\}$$

$t_i^{\{+,-\}}$ means the split space, $\mathcal{B}_i^{\{+,-\}}$ is the added bounds according to $t_i^{\{+,-\}}$, $e(\mathcal{D})$ is the function for adding a dimension to the set of points $\mathcal{D}$, $u_i^{\{+,-\}}(\cdot)$ and $l_i^{\{+,-\}}(\cdot)$ satisfy $l_i^{\{+,-\}}(x_i) \leq f(x_i) \leq u_i^{\{+,-\}}(x_i), \forall x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+,-\}})$. The definition is equal to the description from PRIMA and the proof can be found in Theorem. 2.

4 Volume Approximation Based Grouping

In this section, we provide our improved neurons grouping strategy for k-neuron($k \in Z^+, k \geq 2$) convex relaxation. Assume that we are dealing with n_p neurons in the p-th layer of a fully connected layer, these n_p value of neurons marked as $x_0, x_1, ..., x_{n_p-1}$, and a set $\mathcal{S} = \{x_0, x_1, ..., x_{n_p-1}\}$. The formalization and soundness proof for FAGMR will be provided in the following.

4.1 Volumes Approximation of FaGMR

In this section, we focus on how to approximate to guide the neuron grouping in multi-neuron convex relaxation methods while maintaining its soundness. This can reduce the $2k$-dimensional calculation to a k-dimensional one.

Under-approximation The most time-consuming procedure step in multi-neuron relaxation is the computation of convex hull , which is an NP-hard problem as in traditional methods. Rather than computing the convex hull for each k-neuron group in traditional ways, we compute a volume approximation of them. The volume approximation is then used to guide grouping.

For a convex polyhedron $\mathcal{K} \in \mathbb{R}^d$, its over and under-approximations of volume, denoted as $\overline{V}(\mathcal{K})$ and $\underline{V}(\mathcal{K})$ respectively, have the following relation [9]:

$$\frac{\overline{V}(\mathcal{K})}{\underline{V}(\mathcal{K})} \geq \left(\frac{cd}{\log d}\right)^d$$

where exists a constant c. One such algorithm from Betke et al. can obtain $\overline{V}(\mathcal{K}) = \prod_{i=0}^{d-1}(u_i - l_i)$ and $\underline{V}(\mathcal{K}) = \prod_{i=0}^{d-1}(u_i - l_i)/d!$, $[l_i, u_i]$ implies the range of the i-th dimension in $\mathcal{K}$. We use $\underline{V}(\mathcal{K})$ as under-approximation in FaGMR, and generate tighter approximation based on the idea presented by Betke et al. to guide well for grouping strategy while keeping soundness.

Over-approximation We present the formulas of the under and over-approximation used in FaGMR and provide the proof about the over-one based on the definition in Section. 3.2.

Theorem 1. *(Approximation formation). $VoL(\mathcal{C}_s)$ can be bounded by $\overline{V}_s$ and $\underline{V}_s$ as follows:*

$$\overline{V}_s = VoL(\mathcal{P}_s) \cdot \prod_{i \in [\![k]\!]} (\overline{f}_i - \underline{f}_i) \tag{2}$$

$$\underline{V}_s = \frac{1}{(2k)!} \cdot \prod_{i \in [\![k]\!]} (u_i - l_i) \cdot (\overline{f}_i - \underline{f}_i) \tag{3}$$

where $\overline{f}_i = \max(u_i^+(x_i), u_i^-(x_i))$, $\underline{f}_i = \min(l_i^+(x_i), l_i^-(x_i))$, $\forall x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+,-\}})$.

Proof. The structure of $\mathcal{C}_s$ is unique, allowing us to derive a tighter over-approximation than the one described in Betke et al.
Regarding the over-approximation, we have the following:

$$
\begin{aligned}
&\because \mathcal{B}_i^+ \subseteq \left\{ \boldsymbol{x} \in \mathbb{R}^{k+i+1} | y_i \leq \overline{f}_i, x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^+) \right\}; \\
&\quad \mathcal{B}_i^- \subseteq \left\{ \boldsymbol{x} \in \mathbb{R}^{k+i+1} | y_i \geq \underline{f}_i, x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^-) \right\}; \\
&\quad \mathcal{P}_s^{(i+1)} = conv(e(\mathcal{P}_s^{(i)} \cap t_i^+) \cap \mathcal{B}_i^+ \cup e(\mathcal{P}_s^{(i)} \cap t_i^-) \cap \mathcal{B}_i^-), \forall i \in [\![k]\!] \\
&\therefore P_s^{(i+1)} \subseteq conv(e(P_s^{(i)}) \cap \left\{ \boldsymbol{x} \in \mathbb{R}^{k+i+1} | y_i \leq \overline{f}_i \right\} \\
&\quad \cup e(P_s^{(i)}) \cap \left\{ \boldsymbol{x} \in \mathbb{R}^{k+i+1} | y_i \geq \underline{f}_i \right\}, \forall i \in [\![k]\!])
\end{aligned}
$$

The right-hand side of the above equation can be regarded as the extension of $\mathcal{P}_s^{(i)}$ along the y_i axis. The extension is intersected by the planes $y_i = \overline{f}_i$ and $y_i = \underline{f}_i$, and the content of the intersected extension is the over-approximation for $\mathcal{P}_s^{(i+1)}$. Based on the definition of a prism [23], a k-dimensional prism is formed by the parallel motion of a $(k-1)$-dimensional polytope, the base content S of the prism is the volume of the $(k-1)$-dimensional polytope and its height h equals to the distance of the parallel motion. Because the $\mathcal{P}_s^i$ corresponds to the $(k-1)$-dimensional polytope, and the extension of $\mathcal{P}_s^i$ along the y_i axis, which is cutten by planes $y_i = \overline{f}_i$ and $y_i = \underline{f}_i$, corresponds the distance of parallel motion between planes $y_i = \overline{f}_i$ and $y_i = \underline{f}_i$. Therefore, the over-approximation of $VoL(P_s^{(i+1)})$ is a volume of prism with a base content of $VoL(\mathcal{P}_s^{(i)})$ and a height of $\overline{f}_i - \underline{f}_i$.

$$
\begin{aligned}
&\therefore VoL(P_s^{(i+1)}) \leq VoL(P_s^{(i)}) \cdot (\overline{f}_i - \underline{f}_i) \\
&\therefore VoL(\mathcal{C}_s) \leq VoL(\mathcal{P}_s) \cdot \prod_{i \in [\![k]\!]} (\overline{f}_i - \underline{f}_i)
\end{aligned}
$$

Because $VoL(\mathcal{P}_s)$ is k-dimensional, it is easier to compute an actual value which is also over-approximated in Betke et al., so $\overline{V}_s$ in FaGMR is tighter.

4.2 Detailed algorithm of FaGMR

We now present the detailed workflow of FaGMR in Algorithm 1. It takes as input a set $\mathcal{S}$ that needs to be grouped, the pre-computed lower bounds $\mathcal{L}$ and upper bounds $\mathcal{U}$ of these neurons, and outputs the group $\mathcal{G}$ which will be sent to PDDM, the instrument in Prima.

For x_i, FaGMR collects all groups containing x_i (lines 2 to 4). Line 6 calculates the halfspace-representation of $\mathcal{P}_s$ [10]. Once the volume approximations of all possible groups for x_i are calculated(line 7), FaGMR chooses the groups that have smaller over-approximations as $\mathcal{G}_i$ by Algorithm 2(line 8), the $VoL(\mathcal{P}_s)$ with low dimension can be easily computed by Qhull [4].

Specifically, we compute all the volume $\overline{V}_s$ and $\underline{V}_s$ for s containing x_i using equation 2 and 3. Traverse $\overline{V}_s$ and $\underline{V}_s$ for all $s \subseteq \mathcal{S}$, if there is a group $s \notin \mathcal{G}_i$ that satisfies $\overline{V}_s \leq \max_{j \in \mathcal{G}_i}(\underline{V}_j)$, let $\mathcal{G}_i = \{s\}$. If $\underline{V}_s \geq \min_{j \in \mathcal{G}_i}(\overline{V}_j)$, ignore this

Algorithm 1 FAGMR Grouping Strategy for a Layer

1: **Input:** Input set $\mathcal{S}$ need to be grouped of the layer, pre-computed bounds $\mathcal{L}$ and $\mathcal{U}$ of the layer.
2: $\mathcal{G} \leftarrow \emptyset$
3: **for** x_i in $\mathcal{S}$ **do**
4: **for** s containing x_i in $\mathcal{S}$ **do**
5: **if** $\overline{V}_s = \varnothing$ **then**
6: Create region: $\mathcal{P}_s \leftarrow$ GET_REGION(s, $\mathcal{L}$,$\mathcal{U}$)
7: Volume approxiamtion: $\overline{V}_s \leftarrow$ Equation 2, $\underline{V}_s \leftarrow$ Equation 3
8: Updating $\mathcal{G}_i$: $\mathcal{G}_i \leftarrow$ UPDATE($\mathcal{G}_i$,$\overline{V}_s$, $\underline{V}_s$)
9: **end if**
10: **if** $\mathcal{G}_i$ **then**
11: Add to final group: $\mathcal{G} = \mathcal{G} \cup \mathcal{G}_i$
12: **end if**
13: **end for**
14: **end for**
15: Return $\mathcal{G}$

s, otherwise, add s to set $\mathcal{G}_i$. After that, continue to traverse $\overline{V}_s$ and $\underline{V}_s$ to make the set $\mathcal{G}_i$ as the set of groups selected for neuron i. Finally, adds $\mathcal{G}_i$ to $\mathcal{G}$ or do nothing if no group is found(line 10 to 11).

Algorithm 2 a strategy for selecting groups UPDATE

1: **Input:** group set $\mathcal{G}_i$ and the approximations $\overline{V}_s$ and $\underline{V}_s$.
2: **if** $\mathcal{G}_i = \emptyset$ **then**
3: $\mathcal{G}_i = \{s\}$
4: **else if** $\overline{V}_s \leq \max_{j \in \mathcal{G}_i}(\underline{V}_j)$ **then**
5: $\mathcal{G}_i = \{s\}$
6: **else if** $\underline{V}_s \geq \min_{j \in \mathcal{G}_i}(\overline{V}_j)$ **then**
7: pass
8: **else**
9: Add s to $\mathcal{G}_i$
10: **end if**
11: Return $\mathcal{G}_i$

The resulting set $\mathcal{G}$ for a layer is then passed to the PDDM [18] to obtain their $(2k)$-dimensional convex hulls. From the intersection of these convex hulls, $\mathcal{L}$ and $\mathcal{U}$ for each neuron member's lower and upper bounds in the next layer, respectively, can be obtained using an LP solver. Then Algorithm 1 can be recursively applied to find groups for the next layer.

4.3 Equivalency proof

Before proving our description of $\mathcal{C}$ equals to the one in PRIMA. We give the description of $\mathcal{C}$ in PRIMA, using the symbols in Definition. 2 through convexity

preserving [15]:

$$\mathcal{C}_s = conv\big(\bigcup_t (e^k(\mathcal{P}_s^{(0)}) \cap \bigcap_{i\in s} t_i^{\{+,-\}} \cap \bigcap_{i\in s} \mathcal{B}_i^{\{+,-\}})\big) \tag{4}$$

where t means any combination of $t_i^{\{+\}}$ or $t_i^{\{-\}}$, $\forall i \in s$, $e^k(\cdot)$ means do k times extend operations on the set.

Theorem 2. *(Equivalency proof). The Definition. 2 is equivalent to Equation. 4.*

Proof. Use the convexity preserving and the definition of the $e(\cdot)$, $t/t_i^{\{+,-\}}$ means the combinations without $t_i^{\{+,-\}}$, we can get following equations:

$$\begin{aligned}
\mathcal{C}_s &= conv\big(e^k(\mathcal{P}_s^{(0)}) \cap \bigcup_t (\bigcap_{i\in s} t_i^{\{+,-\}} \cap \bigcap_{i\in s} \mathcal{B}_i^{\{+,-\}})\big) \\
&= conv\big((e^k(\mathcal{P}_s^{(0)}) \cap \bigcup_{t/t_0^-} (t_0^+ \cap \mathcal{B}_0^+ \cap \bigcap_{i\in s/0} t_i^{\{+,-\}} \cap \bigcap_{i\in s/0} \mathcal{B}_i^{\{+,-\}})) \\
&\cup (e^k(\mathcal{P}_s^{(0)}) \cap \bigcup_{t/t_0^+} (t_0^- \cap \mathcal{B}_0^- \cap \bigcap_{i\in s/0} t_i^{\{+,-\}} \cap \bigcap_{i\in s/0} \mathcal{B}_i^{\{+,-\}}))\big) \\
&= conv\big(e^{k-1}(e(\mathcal{P}_s^{(0)}) \cap (t_0^+ \cap \mathcal{B}_0^+ \cup t_0^- \cap \mathcal{B}_0^-)) \cap \bigcup_{t/t_0^{\{+,-\}}} (\bigcap_{i\in s/0} t_i^{\{+,-\}} \cap \bigcap_{i\in s/0} \mathcal{B}_i^{\{+,-\}})\big) \\
&= conv\big(e^{k-1}(\mathcal{P}_s^{(1)}) \cap \bigcup_{t/t_0^{\{+,-\}}} (\bigcap_{i\in s/0} t_i^{\{+,-\}} \cap \bigcap_{i\in s/0} \mathcal{B}_i^{\{+,-\}}\big) \\
&= ... \\
&= conv\big(e(\mathcal{P}_s^{(k-1)} \cap t_{k-1}^+) \cap \mathcal{B}_{k-1}^+ \cup e(\mathcal{P}_s^{(k-1)} \cap t_{k-1}^-) \cap \mathcal{B}_{k-1}^-\big) = \mathcal{P}_s^{(k)}
\end{aligned}$$

The left of the equation is equivalent to the right, so the theorem is proven.

4.4 Generalizaion

FaGMR supports fast grouping with common activation functions. According to Algorithm 1, FaGMR avoids $2k$-dimensional calculation by approximating volume with a k-dimensional polyhedron $\mathcal{P}_s$. Therefore, the necessary inputs for FaGMR are the bounds for each $u_i^{\{+,-\}}(x_i)$ and $l_i^{\{+,-\}}(x_i)$. The problem is turned to finding the bounds of $u_i^{\{+,-\}}(x_i)$ and $l_i^{\{+,-\}}(x_i)$ under different activation functions.

For the Sigmoid and Tanh functions, which are given by $\sigma(x) = \frac{e^x}{e^x+1}$ and $tanh(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$, respectively, FaGMR uses the bounds in Singh et al [21]:

$$f(x) \leq f(u_d) + (x - u_d) \begin{cases} (f(u_d) - f(l_d))/(u_d - l_d) & ,u_d \leq 0, \\ \min(f'(u_d), f'(l_d)) & ,else. \end{cases}$$

$$f(x) \geq f(l_d) + (x - l_d) \begin{cases} (f(u_d) - f(l_d))/(u_d - l_d) & ,l_d \geq 0, \\ \min(f'(u_d), f'(l_d)) & ,else. \end{cases}$$

and $l \leq x \leq u$, the x is called an unfixed neuron. $f(\cdot)$ denotes the Sigmoid or Tanh function. For example, on Tanh function with $c \geq 0$ and $l \leq 0 \leq u$, the interval of x is splited as $[l, c]$ and $[c, u]$. $u^+(x) = f(u) + (x - u)\min(f'(u), f'(c))$, $l^+(x) = f(c) + (x - c)(f(u) - f(c))/(u - c)$ according to $[l_d, u_d] = [c, u]$ and $c, u \geq 0$. Similarly we can get the formations of $u^-(x)$ and $l^-(x)$. Then, get $\overline{f}$ and $\underline{f}$ of x. Note that there is no unfixed neuron in Sigmoid networks($l \geq 0$) so we consider all neurons in one layer for selecting groups.

5 Experiments

In this section, we evaluate the efficiency and effectiveness of FaGMR on common benchmarks. Specifically, we compare FaGMR with the state-of-the-art multi-neuron relaxation verifier Prima in Section 5.2 and analyze FaGMR's efficiency advantage in Section 5.3. Section 5.4 compares FaGMR with two other state-of-the-art verifiers from Verification of Neural Networks Competition (VNN-COMP) 2021: [3]: $\alpha\beta$CROWN [29]and ERAN [24]. They are representatives of other mainstream verification techniques besides multi-neuron relaxation. Section 5.5 shows the evaluation results of FaGMR on Sigmoid and Tanh neural networks.

5.1 Experiments Configuration

Neural Networks. We conduct experiments on two commonly used datasets: MNIST and CIFAR-10. Especially, besides the fully-connected and convolutional network structures, our evaluations also include a large ResNet network. The Table.1 shows the details about all the ReLU networks used in our experiments. These networks can be downloaded at ERAN homepage [24], and they are benchmarks of VNN-COMP 2021 [3].

Robustness Property. Same as the most works, we consider the l_∞-norm perturbation on a correct-classified input from the test set of the datasets above. In this setting, given a perturbation threshold ε, the input region is an ε-ball containing all the possible perturbed images (infinite in number). A verifier need to verify the neural network does not misclassify any perturbed input in this region. Generally, there are hundreds of verification tasks (we mark the number "total" in experiments) for a network. If a verifier verifies more of them than others, then it is better in precision, because all methods guarantee their answers. In principle, the perturbations ε are selected as the same as they are in experiments of Prima, except all of the verifiers failed to verify the benchmarks due to memory constraints, although the perturbation thresholds ε were the same as those used in the experiments of Prima.

Parameters of FaGMR. FaGMR chooses $k = 3$ for 3-neuron relaxation, which is claimed as the optimal option in Prima. In each activation layer, FaGMR finds all the 3-neuron groups.

Table 1. Neural Networks used in the experiments.

Dataset	Model	Type	Units	Layers
MNIST	5 × 100	FC	510	5
	6 × 200	FC	1010	6
	ConvBig	Conv	48064	6
CIFAR-10	ConvSmall	Conv	4852	3
	ResNet2b	Residual	11364	13

Machine and Software. All experiments are conducted on a 12-core 2.20GHz Intel Xeon Sliver 4212 CPU platform with 64 GB main memory. FaGMR is implemented in Python 3.7 and it uses Gurobi 9.5.1 [1] for LP problem solving.

5.2 Comparison of Verification Precision and Runtime

We first compare the verification precision and runtime of FaGMR with Prima, the state-of-the-art multi-neuron relaxation tool for neural network verification. Instead of using all the possible k-neuron groups in a layer, Prima defines a parameter n_s to divide the neurons of a layer into sets of size n_s, and for each set it chooses its $C_{n_s}^k$ k-groups. The other parameter s is the limited overlapping between arbitrary two groups, which means the maximum counts of identical neurons in two groups is s($s = 1$ is claimed as a default option in Prima). we use the default value $n_s = 70$ in Prima. We call Prima with this grouping heuristic Prima-para, and the strategy using all the possible k-neuron groups Prima-all. Table.2 shows the comparison results of FaGMR with Prima-all and Prima-para.

For a given network with total correct-classified inputs and a perturbation threshold ε, we compare three tools w.r.t. their average number of groups (g), average runtime in verification (t), and verified number (v). The best data of these three metrics is in bold in Table.2. We notice FaGMR out-performs Prima-all and Prima-para in runtime, and both the runtime and verification precision in some cases. Specifically, on MNIST 6×200 with $\varepsilon = 0.015$, FaGMR is 16.0% faster than Prima-para with one more verified, and it is 46.0% faster than Prima-all while having the same verification precision. On CIFAR-10 ResNet2b, FaGMR is 51.1% faster than Prima-para. Besides, it is worth noting that in the complex network MNIST ConvBig, both Prima-all and Prima-para is out of memory, whereas FaGMR returns results in the same memory limitation.

When comparing Prima-para with Prima-all, we can find the heuristic of Prima-para can boost the efficiency of Prima-all to some extent. It is possible to find a better balance between precision and runtime by changing the hyperparameters in Prima-para. However, the optimal hyperparameters depend on the size of a layer in different networks, which need to know in prior and need some fine-tuning. FaGMR avoids this limitation by only selecting necessary

Table 2. Comparison of FAGMR with PRIMA-all and PRIMA-para in average number of groups g, average runtime t (in seconds), and verification precision v on five ReLU networks. "Total" is the number of correct-classified inputs that verifiers need to consider, and "ε" is the perturbation threshold. Symbol "-" means results are not available due to out-of-memory issues.

Network	Total	ϵ	PRIMA-all			PRIMA-para			FAGMR		
			g	t	v	g	t	v	g	t	v
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
MNIST 6×200	972	0.015	1.6K	106.7	959	1.5K	68.6	958	**877.0**	**57.6**	**959**
MNIST ConvBig	929	0.03	-	-	-	-	-	-	**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
CIFAR-10 ResNet2b	161	1/255	-	-	-	7.0K	407.5	158	**1.1K**	**199.4**	158

groups. Through our grouping strategy, FAGMR can reduce the verification time by 24.7% on average compared with PRIMA-para, and with minimal precision loss compared to PRIMA-all. This experiment confirms the effectiveness and efficiency of FAGMR.

Statistics of Runtime The essential reason for the efficiency advantage of FAGMR is that it forms fewer groups, which causes the LP solver to deal with lesser constraints. The second reason is that we abandon groups through an approximation strategy based on volume approximation before sending it to PDDM for computing high-dimensional convex hulls. We quantitatively analyze the influence of fewer groups by splitting the runtime into three detailed parts: 1) time of generating groups (T_1); 2) time to gain constraints by PDDM (T_2); and 3) time by the LP solver (T_3). We observe that the sum of T_1 and T_2 in FAGMR is less than PRIMA-all and PRIMA-para in all cases. For instance, in MNIST 5×100 when $\varepsilon = 0.015$, $T_1 + T_2$ of FAGMR is 70% and 53.4% faster than PRIMA-all and PRIMA-para respectively. The results imply FAGMR reduces some redundant groups whose constraints need to be considered by PDDM in PRIMA-all or PRIMA-para.

Additionally, if we assume that bounded polyhedra have at most n_v vertices and n_a constraints, the worst-case time complexity of once PDDM is about $O(n_v \cdot n_a^4 + 2 \cdot n_a^2 \cdot log(n_a))$, and that the one of abandoning a group containing x_i through Qhull is about $O(n_i^2 \cdot k^3)$, where n_i is the maximum count of vertexes of $\mathcal{P}_s$ among s containing x_i. For instance, in detail of ConvSmall(CIFAR-10) with $\varepsilon = 2/255$, the sum of $T_1 + T_2$ of FAGMR is almost identical to that of PRIMA-para or PRIMA-all, although the grouping strategy in FAGMR succeeds

Table 3. The detailed runtime (in seconds) statistics of FAGMR vs. PRIMA-all and PRIMA-para. T_1 is the time of generating groups; T_2 is the time to gain constraints by PDDM; and T_3 the time by the LP solver.

Network	ϵ	PRIMA-all			PRIMA-para			FAGMR		
		T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3
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**

in inducing T_2 and T_3, it increases T_1 for the demand for selecting groups. The reason is that there is no obvious gap between $O(n_v \cdot n_a^4 + 2 \cdot n_a^2 \cdot log(n_a))$ and $O(n_i^2 \cdot k^3 \cdot n_p)$ in this case, where n_p is the count of neurons in p-th layer. According to experiments, this situation occurs when there are only a few unfix-neurons in the case, and the hyperparameters in PRIMA-para almost lose efficacy. As a result, the more unfix-neurons are, the more efficient FAGMR is compared to PRIMA-para and PRIMA-all while maintaining precision.

Both the experiments and complexity analysis show that abandoning groups through an approximation strategy has the potential to be faster than using parameters.

5.3 Comparison of Robustness Radius

In the previous section, we compare the verification ability of FAGMR and PRIMA under a perturbation threshold. However, another important question is: *how well do they perform on a given input?* The most common metric for this is the robustness radius, which measures the perturbation threshold at which a neural network misclassifies an input. Since verification methods provide a "safe" answer, a higher robustness radius indicates higher precision.

In this experiment, we select the first 10 verified images from the MNIST test set and compute their robustness radius on the MNIST 5×100 network using PRIMA-all, PRIMA-para, and FAGMR. The method of calculating robustness radius is dichotomy [13] and the iteration is 15. Figure. 3 presents the results. The blue bar in Figure. 3 means the robust radius obtained by PRIMA-all, the green one implies FAGMR, and the yellow implies PRIMA-para. In this experiment, PRIMA-all cost 4.3Ks on average to get 0.0024 total improvement than PRIMA-para. FAGMR spends 3.9Ks on average, 7.1% faster with 0.0021 improvement than PRIMA-para, particularly in images 7. Figure. 3 means discarding some groups can save time but also influence the verification precision, and we can also find out that FAGMR efficiently fills the precision gap between PRIMA-para and PRIMA-all.

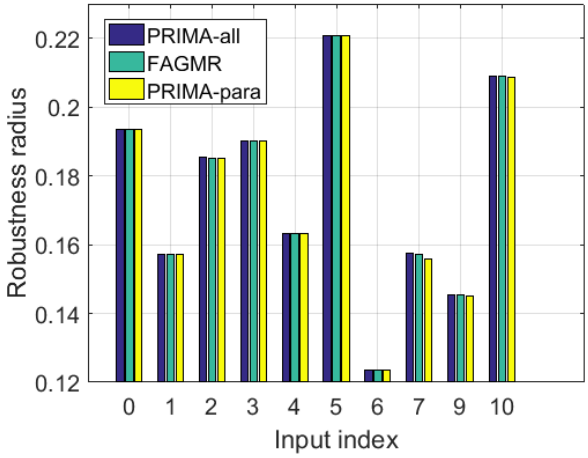

Fig. 3. Verified robustness radius by PRIMA-all, PRIMA-para, and FAGMR.

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

We have shown FAGMR outperforms the state-of-the-art multi-neuron relaxation verifier PRIMA. Besides this, it is necessary to know the performance position of FAGMR among other kinds of state-of-the-art verifiers. Figure 4 shows the

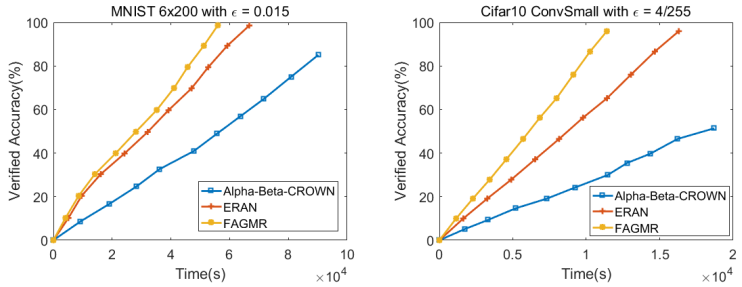

Fig. 4. Comparisons on MNIST 6×200 with $\varepsilon = 0.015$, and ConvSmall(CIFAR-10) with $\varepsilon = 4/255$ respectively.

comparison of runtime and precision of FAGMR with other state-of-the-art verifiers, including $\alpha\beta$CROWN [29] and ERAN, which are both outstanding verifiers in VNN-COMP 2021 [3]. $\alpha\beta$CROWN is an efficient verifier that uses symbol propagation methods [30] represented as matrices, and its relaxation method is single-neuron based. ERAN is also an efficient verification toolbox

Table 4. Average groups number g, average runtime t(s), and verification precision v on Sigmoid or Tanh networks.

Network	Total	Activation	ϵ	PRIMA-para			FAGMR		
				g	t	v	g	t	v
MNIST 6×500	95	Sigmoid	0.012	6.2K	640.9	95	**3.8K**	**570.0**	95
MNIST ConvMed	99	Sigmoid	0.014	6.3K	255.6	99	**5.9K**	**252.5**	99
MNIST 6×500	99	Tanh	0.005	-	-	-	**184.8**	**194.9**	**98**
MNIST ConvMed	98	Tanh	0.005	1.2K	220.4	98	**446.3**	**191.8**	98

with several verifiers that rely on LP or MILP solvers, or symbolic propagation solvers, and one of its relaxation methods is PRIMA.

We use their default configurations and perform comparisons on the 6×200 MNIST network and ConvSmall(CIFAR-10). The results show that $\alpha\beta$CROWN has lower precision and longer runtime than both ERAN and FAGMR. This can be attributed to $\alpha\beta$CROWN's use of single-neuron relations that lead to lower precision and the fact that it uses symbolic propagation with MILP and BaB strategy, which requires more iterations and runtime.

Regarding the comparison between ERAN and FAGMR, we observed that FAGMR runs 23.1% faster than ERAN on average with one more verified on the two benchmarks with small turbulences. Therefore, FAGMR can be considered as one of the competitive network verifiers.

5.5 Comparison on Tanh and Sigmoid Neural Networks

As mentioned in Section 4.4, FAGMR can also be applied to other activations such as Singmoid and Tanh. In this experiment, we compare the performance of PRIMA-para and FAGMR on the non-piecewise-linear activation functions, Tanh and Sigmoid, using the MNIST dataset. We evaluate 100 images on MNIST 6×500(3000 neurons) and ConvMed(5704 neurons), both datasets can be obtained from the ERAN homepage [24]. However, due to out-of-memory issues, PRIMA-all is excluded from this experiment. FAGMR is configured similarly to PRIMA-para, except we have omitted parameters. In Table 4, we observe that FAGMR is 8.4% faster than PRIMA-para on average with the same number of verified. These results demonstrate that FAGMR is a general method that performs better than the current state-of-the-art to some extent.

6 Related Work

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

Complete Verification. Complete verification methods can describe the exact behavior of a neural network on an input region. They can be further classified into: (1) SAT/SMT based methods [13,14,8], which encodes the verification problem into an SAT/SMT query; (2) Mixed-integer linear programming (MILP) based methods [8,2,6], which encodes the verification into a MILP problem and is solved by specific solvers. (3) Branch-and-Bound based methods, which split non-linear activation functions into linear pieces [29,26,7,28], or split the input region to be small enough so that the neural network behaves linearly on each input sub-region [27]. Complete methods are 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. [21,22,30,19,29,33], see a survey in [16]). FAGMR belongs to this category. It applies convex relaxation to over-approximate non-linear activation functions, hence incomplete but are much faster and more scalable than complete methods.

Single-neuron Relaxation. Single-neuron convex relaxation-based methods consider each neuron separately and over-approximate its activation function. They are efficient enough [21,22,30,19,29,33,32,12], but their verification precision is proved to be limited by a convex relaxation barrier [19].

Multi-neuron Relaxation. Multi-neuron convex relaxation methods [20,25,18] suggest over-approximating jointly multiple neurons for higher precision, hence breaking down the barrier faced by single-neuron relaxation. Our work follows this promising direction and aims to improve the efficiency of existing techniques.

7 Conclusion

In this paper, we propose an automatic, fast, and effective neuron grouping strategy for multi-neuron convex relaxation methods. Our key idea is computing the volume approximations of all possible k-neuron groups and choose better ones for each neuron according to the computed volume. The experiments show our strategy costs much less time and has a precision advantage over the state-of-the-art tools.

References

1. Gurobi optimizer reference manual (2008), http://www.gurobi.com
2. 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
3. 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
4. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: Qhull: Quickhull algorithm for computing the convex hull. Astrophysics Source Code Library pp. ascl–1304 (2013)
5. Betke, U., Henk, M.: Approximating the volume of convex bodies. Discrete & Computational Geometry **10**, 15–21 (1993)
6. 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
7. 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
8. 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
9. Elekes, G.: A geometric inequality and the complexity of computing volume. Discrete & Computational Geometry **1**, 289–292 (1986)
10. Gaubert, S., Katz, R.D.: Minimal half-spaces and external representation of tropical polyhedra. Journal of Algebraic Combinatorics **33**(3), 325–348 (2011)
11. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016), http://www.deeplearningbook.org/
12. Goubault, E., Palumby, S., Putot, S., Rustenholz, L., Sankaranarayanan, S.: Static analysis of relu neural networks with tropical polyhedra. In: Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings 28. pp. 166–190. Springer (2021)
13. 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
14. 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

15. Li, A.: Convexity preserving interpolation. Computer Aided Geometric Design **16**(2), 127–147 (1999)
16. 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
17. 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
18. 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
19. 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)
20. 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
21. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf
22. 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
23. Sommerville, D.M.: Introduction to the Geometry of N Dimensions. Courier Dover Publications (2020)
24. SRI Lab: ETH robustness analyzer for neural networks (ERAN) (2022), https://github.com/eth-sri/eran
25. 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
26. 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

27. 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
28. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., 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., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 29909–29921. Curran Associates, Inc. (2021), https://proceedings.neurips.cc/paper/2021/file/fac7fead96dafceaf80c1daffeae82a4-Paper.pdf
29. 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
30. 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)
31. 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)
32. Zhao, Z., Zhang, Y., Chen, G., Song, F., Chen, T., Liu, J.: Cleverest: Accelerating cegar-based neural network verification via adversarial attacks. In: Static Analysis: 29th International Symposium, SAS 2022, Auckland, New Zealand, December 5–7, 2022, Proceedings. pp. 449–473. Springer (2022)
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