

# 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 accuracy. However, existing techniques suffer from the dependency on hyperparameters to select neuron groupings for reducing massive convex computations, which heavily limits their generality. Based on the observation that the abstract polytope of multi-neuron relaxation with a bigger volume may contain intersections of the polytopes from other multi-neuron relaxations.

This paper proposes a volume approximation-based approach to select neuron groupings automatically, for making multi-neuron convex relaxation methods more effective and efficient. We compute the volume approximation for each pre-grouping and then choose the neuron groupings with smaller volumes to calculate their convex hull. This approach reduces the convex hull computations yet maintains the verification precision.

We implement our approach as the neural network verification tool FAGMR and evaluate it based on benchmarks from previous works. The experimental results show that FAGMR can be more efficient than state-of-the-art multi-neuron convex relaxation tools. Specifically, on average, FAGMR spends 18.8% less verification time than existing works for each verification task while achieving better performing verification results.

**Keywords:** Neuron 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 of a neural network. For example, forged traffic signs can fool certain autonomous driving systems [?]. They look almost the same for 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 [?]). 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 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 [?, ?, ?, ?]. 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 groupings of neurons [?, ?, ?]. An essential problem of multi-neuron relaxation-based methods lies in convex hull computations. Typically, in the first step, these methods select groupings of neurons of size  $k$  ( $k \geq 2$ ) in the same activation layer. For each grouping, 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 groupings of neurons are considered and the more overlap between groupings is allowed, the more precise verification results can be achieved. However, the NP-hardness of the convex hull computation problem limits the number of groupings to be selected. For instance, adopting an exact convex hull computing algorithm, KPOLY [?] partitions the neurons of an activation layer into small sets of size  $n_s \leq 5$  and only selects groupings of  $k \leq 3$  neurons within each partition. PRIMA [?] proposes a polynomial-time method for approximating convex hulls, hence allowing to consider larger number of groupings in a reasonable time limit. Similarly, it partitions all neurons concerning  $n_s$  and selects a subset of all size- $k$  groupings 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 groupings 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 groupings, in the sense that the constraints of their convex hulls are implied by the constraints generated by other groupings. Convex hull computations of these redundant groupings 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 groupings. 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 groupings with small estimated volumes will be selected while groupings 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 (**F**ast **G**rouping for **M**ulti-neuron **R**elaxation). We compare FAGMR with state-of-the-art tools PRIMA [?],  $\alpha\beta$ CROWN [?] and ERAN [?] on neural networks trained by the widely-used datasets MNIST and CIFAR-10. The experimental results show that FAGMR runs 18.8%, 38.5%, and 23.1% faster on average than PRIMA,  $\alpha\beta$ CROWN, and ERAN respectively, with similar verifieds with PRIMA and ERAN, and 30.1% verifieds than  $\alpha\beta$ CROWN.

Our contributions are summarized as follows:

- We propose a volume approximation-based approach to selecting neuron groupings 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 ?? presents the overview of FAGMR. Section ?? gives the necessary background on neural network verification and the multi-neuron relaxation method from one of the state-of-the-art, kPOLY. Then draws for the objection of FAGMR, which is detailed in Section ?? and evaluated in Section ?. We give some information about related works and conclude the paper in Section ?? and Section ?.

## 2 Overview

Figure ?? presents the main idea behind FAGMR. To verify the property  $\varphi$  of a neural network  $\mathbf{h}$  precisely, FAGMR uses multi-neuron relaxations in each intermediate layer to obtain tighter constraints. In order to automatically reduce the massive convex hull computation involved in multi-neuron relaxations, FAGMR under and over-approximates the volume corresponding to each grouping with low-dimensional computation and picks out some groupings through these approximated volumes. Next, FAGMR inputs these selected groupings to the existing implementation for their corresponding multi-neuron relaxation by high-dimensional computation. The intersection of all the multi-neuron relaxations from selected groupings in one layer is the constraints for the following layers.

*Notation.* We reserve lowercase Latin and Greek letters  $a, b, x, \dots, \lambda, \dots$  for scalars, bold  $\mathbf{a}$  for vectors or coordinate points, capitalized bold  $\mathbf{A}$  for matrices, and calligraphic  $\mathcal{A}$  or blackboard bold  $\mathbb{A}$  for sets. Similarly, scalar functions are denoted as  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ . Given  $n$  elements, the number of  $k$ -combinations is denoted by  $\binom{n}{k}$ . We call a set of neurons a grouping. Mark  $\{0, 1, \dots, n-1\}$  as  $\llbracket n \rrbracket$ .

## 2.1 Neural Network Verifier

The property discussed in this paper is  $\varphi := \arg \max_j \mathbf{h}(\mathbf{x})_j = \arg \max_j \mathbf{h}(\mathbf{x}')_j$ , where  $\mathbf{x}' \in \mathbb{B}$  and  $\mathbf{x}$  is the input of neural network  $\mathbf{h}$ . Most of the state-of-the-art aim to verify the property  $\varphi$ . They compute the convex approximation of the output  $\mathbf{h}(\mathbf{x})$  in an input domain  $\mathbb{B}$ . If  $\varphi$  is satisfied with the network  $\mathbf{h}$  for arbitrary  $\mathbf{x}'$  in the input domain  $\mathbb{B}$ , it is said that the verifier can prove  $\varphi$  in this case. Otherwise, the verifier failed to verify  $\varphi$ . Various verifiers present different trade-offs between the time of approximation and precision, and most of them depend on the settings of hyperparameters.

FAGMR can efficiently compute the convex approximation through automatically balancing time and precision while maintaining tight approximation for  $\mathbf{h}(\mathbf{x})$ .

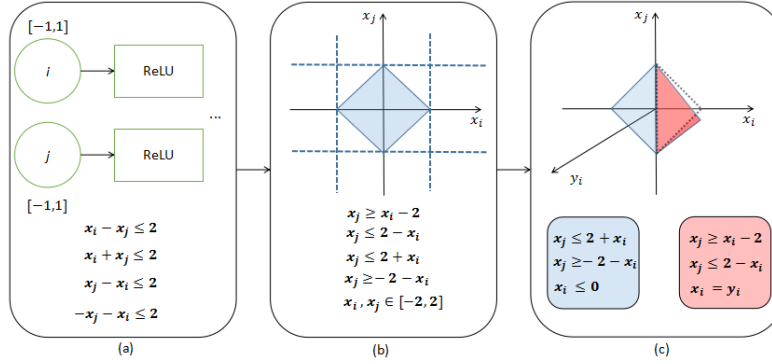
## 2.2 Illustrative Example

We consider the 2-neuron relaxations with several neurons in one fully connected layer, the neurons are  $x_i, x_j, \dots, x_k$ , as shown in Figure. ???. Let  $s$  denote a certain grouping, for example  $s = \{x_i, x_j\}$ . Each layer generates a pair of lower and upper bounds for each neuron, bounding the output of the next ReLU layer until completing the last layer, then the verifier checks whether  $\varphi$  is satisfied or not. Additionally, we use ReLU as the activation function in the following example, FAGMR can be used with other activation functions as well.

**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 bind the output 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 both of the neurons  $i$  and  $j$  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. ??(a). Then, add bounds of  $i$  and  $j$  to the constructed constraints, and the input  $k$ -dimensional polyhedron  $\mathcal{P}$  can be generated as the blue part shown in Figure. ??(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  $\{\mathbf{x} \in \mathbb{R}^2 \mid x_i \geq 0\}$  and  $\{\mathbf{x} \in \mathbb{R}^2 \mid x_i \leq 0\}$ . The resulting parts are  $\{\mathbf{x} \in \mathbb{R}^2 \mid \mathbf{x} \in \mathcal{P} \wedge x_i \geq 0\}$  and  $\{\mathbf{x} \in \mathbb{R}^2 \mid \mathbf{x} \in \mathcal{P} \wedge x_i \leq 0\}$ . Next, we extend these two parts to  $(k+1)$  dimension by adding a new dimension  $y_i$  to  $\mathbf{x}$ . 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  $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{x} \in \mathcal{P} \wedge x_i \geq 0 \wedge y_i \in \mathbb{R}\}$  and  $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{x} \in \mathcal{P} \wedge x_i \leq 0 \wedge y_i \in \mathbb{R}\}$ . Note that  $\mathcal{P}$  with two parameters is put into the 3-dimensional space. We then use



**Fig. 1.** The formation process for  $\mathcal{C}_s$  and  $\mathcal{P}_s$  in ReLU network.

halfspace depending on activation functions to bind 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 bounding.  $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{x} \in \mathcal{P} \wedge x_i \geq 0 \wedge y_i = x_i\}$  (the red part in Figure. ??(c)) and  $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{x} \in \mathcal{P} \wedge x_i \leq 0 \wedge y_i = 0\}$  (the blue one in Figure. ??(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 groupings in one layer are taken an intersection as the constraints for the following layers.

KPOLY and PRIMA directly get  $2k$ -dimensional 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 a proof is given in Theorem. ?? .

In PRIMA, a state-of-the-art depended on KPOLY selects groupings 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 groupings 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 groupings for computing the convex hulls, i.e., a multi-neuron relaxation from a grouping may contain the intersection of the relaxations from other two or more groupings.

Assume that there are three groupings:  $s_0 = \{x_i, x_j\}$ ,  $s_1 = \{x_i, x_k\}$  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 grouping. For example, The output constraints  $\mathcal{C}_0$ ,  $\mathcal{C}_1$ , and  $\mathcal{C}_2$  can be expressed as:

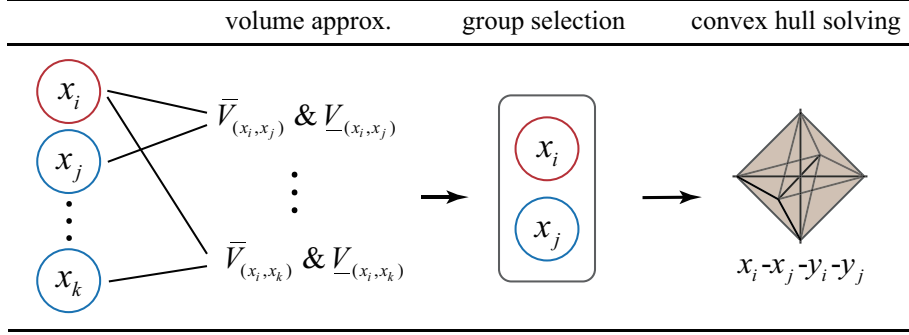
$$\begin{array}{lll}
 \mathcal{C}_0 = \{x \in \mathbb{R}^6 \mid & \mathcal{C}_1 = \{x \in \mathbb{R}^6 \mid & \mathcal{C}_2 = \{x \in \mathbb{R}^6 \mid \\
 -x_i + x_j - y_i - y_j \leq 0 & -x_k - y_k \leq -2 & x_j - x_k - y_j - y_k \leq 0 \\
 x_i - x_j + y_i - y_j \leq 2 & x_k + y_k \leq 2 & x_j - x_k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

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 groupings containing one neuron  $x_i$ , the redundancy may be 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 groupings, but this may overlook the information in the  $\mathcal{P}$  and  $\mathcal{C}$ , such as one neuron may influence the bounds of others.

However, it is unavoidable for high-dimensional calculation for picking out such redundancy precisely, which will spend a lot of time. Because such groupings always have a large value of volume, we turn to compute the approximation of volume through the lower dimension to discard redundant groupings as much as possible, the groupings discarded are also called worse groupings in the following.

FAGMR can efficiently identify the worse groupings before solving the convex hull problems. As a result, it produces neuron groupings that have fewer redundant groupings 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 groupings*. If the volume approximation for the grouping  $\{x_i, x_j\}$  is smaller than for any other groupings', such as  $\{x_i, x_k\}$ ,



**Fig. 2.** Illustration of FAGMR finding a partner neuron for  $x_i$  and send them to compute the four-dimensional convex hull

then we choose  $\{x_i, x_j\}$  as the selected grouping.

Figure. ?? 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 grouping can be time-consuming (i.e., the volume of the convex hull  $x_i-x_j-y_i-y_j$  in Figure. ??). Therefore, we compute the approximation method in Section ?? to under and over-approximate the volumes of convex hulls for each groupings, in order to select groupings which will be computed a  $(2k)$ -dimensional convex hull. The volume approximation method used in FAGMR is based on Betke et al. [?], 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 we have computed all under and over- approximate volume for the groupings, select the groupings for each neuron, like  $x_i$ , whose over-approximation volumes are smaller than most of the under-approximation volumes of other groupings. Our strategy prefers selecting the groupings which avoid the phenomenon in our motivation while maintaining groupings as many as possible to guarantee precision and saving time as possibly as possible. More details can be found in Section ??.

*Convex hull solving* Once the groupings of each neuron are determined, they are sent to derive their exact  $(2k)$ -dimensional convex hull, like  $\mathcal{G}_i = \{x_i, x_j\}$  in Figure. ?. We use the solver in PRIMA [?] 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 computed convex hull, the lower and upper 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 groupings for the subsequent layer.

In Section ??, we introduce the formation process for multi-neuron convex relaxation, which is used in Section ??, where we will 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  $\mathbf{h}(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}^{|\mathcal{Y}|}$  is a  $|\mathcal{Y}|$ -dimensional vector valued function from the input space  $\mathcal{X}$  to the output space  $\mathcal{Y}$ . Specifically,  $\mathbf{h}(\mathbf{x})$  is the interleaved composition of affine function layers  $\mathbf{g}_i(\mathbf{x}) = \mathbf{W}_i \mathbf{x} + \mathbf{b}_i$ , with non-linear activation layers  $\mathbf{f}_i(\mathbf{x})$ :

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

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

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

Formally, perturbed inputs are defined by a  $p$ -norm ball neighborhood of  $\mathbf{x}$ :

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

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

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

$$\arg \max_j \mathbf{h}(\mathbf{x})_j = \arg \max_j \mathbf{h}(\mathbf{x}')_j,$$

for each  $\mathbf{x}' \in \mathbb{B}_\varepsilon^p(\mathbf{x})$ .

If the above statement is true, then we say that the neural network  $\mathbf{h}$  is robust with respect to the input  $\mathbf{x}$  against the perturbation threshold  $\varepsilon$ .



### 3.2 Formation for multi-neuron relaxation

This formation from KPOLY is used in both of PRIMA and FAGMR to generate output constraints  $\mathcal{C}_s$ . Formally, for a  $k$ -dimensional input polytope  $\mathcal{P}_s = \{\mathbf{x} \in \mathbb{R}^k \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$  generated by a set of  $k$  neurons  $s \subseteq \mathcal{S}$ ,  $\mathcal{S}$  is the set of whole neurons, the input polytope  $\mathcal{P}_s$  will generate a higher dimension 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 grouping  $s$  to guarantee  $\mathcal{P}_s$  is a convex polytope, with linear inequation 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  $t_i^+ = \{\mathbf{x} \in \mathbb{R}^k \mid x_i \geq c\}$  and  $t_i^- = \{\mathbf{x} \in \mathbb{R}^k \mid x_i \leq c\}$  (start from  $\forall x_i \in s$  is valid, and  $c$  is a constant [?]). 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  $\mathbf{x}$ , 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  $\mathcal{B}_i^{\{+, -\}} = \{\mathbf{x} \in \mathbb{R}^{k+1} \mid y_i = u_i^{\{+, -\}}(x_i) \cup y_i = l_i^{\{+, -\}}(x_i), \forall \mathbf{x} \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+, -\}})\}$  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. ??.

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

$$\begin{aligned} \mathcal{P}_s^{(i+1)} &= \text{conv}(e(\mathcal{P}_s^{(i)} \cap t_i^+) \cap \mathcal{B}_i^+ \\ &\quad \cup e(\mathcal{P}_s^{(i)} \cap t_i^-) \cap \mathcal{B}_i^-), \forall i \in \llbracket k \rrbracket \end{aligned} \quad (1)$$

where:

$$\begin{aligned} \mathcal{P}_s^{(0)} &= \{\mathbf{x} \in \mathbb{R}^k \mid \mathbf{x} \in \mathcal{P}_s\}; \mathcal{P}_s^k = \mathcal{C}_s \\ t_i^+ &= \{\mathbf{x} \in \mathbb{R}^{k+i} \mid x_i \geq c\}; t_i^- = \{\mathbf{x} \in \mathbb{R}^{k+i} \mid x_i \leq c\} \\ \mathcal{B}_i^{\{+, -\}} &= \{\mathbf{x} \in \mathbb{R}^{k+1} \mid y_i = u_i^{\{+, -\}}(x_i), y_i = l_i^{\{+, -\}}(x_i), \forall \mathbf{x} \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+, -\}})\} \\ e(\mathcal{D}) : \mathbb{R}^i &\leftarrow \mathbb{R}^i \times \mathbb{R}, \mathcal{D} = \{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^i\} \end{aligned}$$

$t_i^{\{+, -\}}$  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  $l_i^{\{+, -\}}(x_i) \leq f(x_i) \leq u_i^{\{+, -\}}(x_i), \forall x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^{\{+, -\}})$ .

## 4 Volume Approximation Based Grouping

In this section, we provide our improved neurons grouping strategy for  $k$ -neuron ( $k \in \mathbb{Z}^+, k \geq 2$ ) 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  $x_0, x_1, \dots, x_{n_p-1}$ , and a set  $\mathcal{S} = \{x_0, x_1, \dots, x_{n_p-1}\}$ . The formalization and a 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 convex hull computation, which is an NP-hard problem as is done in traditional methods. Rather than computing the convex hull for each  $k$ -neuron grouping as traditional methods, 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 volume approximation, denoted as  $\overline{V}(\mathcal{K})$  and  $\underline{V}(\mathcal{K})$  respectively, have the following relation [?]:

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

where exists a constant  $c$ . One such algorithms 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. ??.

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

$$\overline{V}_s = VoL(\mathcal{P}_s) \cdot \prod_{i \in \llbracket k \rrbracket} (\overline{f}_i - \underline{f}_i) \quad (2)$$

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

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:

$$\begin{aligned} \therefore \mathcal{B}_i^+ &\subseteq \left\{ \mathbf{x} \in \mathbb{R}^{k+i+1} \mid y_i \leq \bar{f}_i, x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^+) \right\}; \\ \mathcal{B}_i^- &\subseteq \left\{ \mathbf{x} \in \mathbb{R}^{k+i+1} \mid y_i \geq \underline{f}_i, x_i \in e(\mathcal{P}_s^{(i)} \cap t_i^-) \right\}; \\ \mathcal{P}_s^{(i+1)} &= \text{conv}(e(\mathcal{P}_s^{(i)} \cap t_i^+) \cap \mathcal{B}_i^+ \cup e(\mathcal{P}_s^{(i)} \cap t_i^-) \cap \mathcal{B}_i^-), \forall i \in \llbracket k \rrbracket \\ \therefore \mathcal{P}_s^{(i+1)} &\subseteq \text{conv}(e(\mathcal{P}_s^{(i)}) \cap \left\{ \mathbf{x} \in \mathbb{R}^{k+i+1} \mid y_i \leq \bar{f}_i \right\} \\ &\quad \cup e(\mathcal{P}_s^{(i)}) \cap \left\{ \mathbf{x} \in \mathbb{R}^{k+i+1} \mid y_i \geq \underline{f}_i \right\}, \forall i \in \llbracket k \rrbracket) \end{aligned}$$

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  $y_i = \bar{f}_i$  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 [?], 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  $y_i = \bar{f}_i$  and  $y_i = \underline{f}_i$ , corresponds the distance of parallel motion between planes  $y_i = \bar{f}_i$  and  $y_i = \underline{f}_i$ . Therefore, the over-approximation of  $\text{Vol}(\mathcal{P}_s^{(i+1)})$  is a volume of prism with a base content of  $\text{Vol}(\mathcal{P}_s^{(i)})$  and a height of  $\bar{f}_i - \underline{f}_i$ .

$$\begin{aligned} \therefore \text{Vol}(\mathcal{P}_s^{(i+1)}) &\leq \text{Vol}(\mathcal{P}_s^{(i)}) \cdot (\bar{f}_i - \underline{f}_i) \\ \therefore \text{Vol}(\mathcal{C}_s) &\leq \text{Vol}(\mathcal{P}_s) \cdot \prod_{i \in \llbracket k \rrbracket} (\bar{f}_i - \underline{f}_i) \end{aligned}$$

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

## 4.2 Detailed algorithm of FAGMR

We now present the detailed workflow of FAGMR in Algorithm ?? . It takes as input a set  $\mathcal{S}$  that need 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 groupings containing  $x_i$  (lines 2 to 4). Line 6 calculates the halfspace-representation of  $\mathcal{P}_s$  [?]. Once the volume approximations of all possible groupings for  $x_i$  are calculated(line 7), FAGMR chooses the groupings that have smaller over-approximations as  $\mathcal{G}_i$  by Algorithm ??(line 8), the  $\text{Vol}(\mathcal{P}_s)$  with low dimension can be easily computed by Qhull [?].

**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  $\bar{V}_s = \emptyset$  then
6:       Create region:  $\mathcal{P}_s \leftarrow \text{GET\_REGION}(s, \mathcal{L}, \mathcal{U})$ 
7:       Volume approxiamtion:  $\bar{V}_s \leftarrow \text{Equation ??}$ ,  $\underline{V}_s \leftarrow \text{Equation ??}$ 
8:       Updating  $\mathcal{G}_i$ :  $\mathcal{G}_i \leftarrow \text{UPDATE}(\mathcal{G}_i, \bar{V}_s, \underline{V}_s)$ 
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  $\mathcal{G}$ 

```

---

Specifically, we compute all the volume  $\bar{V}_s$  and  $\underline{V}_s$  for  $s$  containing  $x_i$  using equation ?? and ??. Traverse  $\bar{V}_s$  and  $\underline{V}_s$  for all  $s \subseteq \mathcal{S}$ , if there is a grouping  $s \notin \mathcal{G}_i$  that satisfies  $\bar{V}_s \leq \max_{j \in \mathcal{G}_i}(\underline{V}_j)$ , let  $\mathcal{G}_i = \{s\}$ . If  $\underline{V}_s \geq \min_{j \in \mathcal{G}_i}(\bar{V}_j)$ , ignore this  $s$ , otherwise, add  $s$  to set  $\mathcal{G}_i$ . After that, continue to traverse  $\bar{V}_s$  and  $\underline{V}_s$  to make the set  $\mathcal{G}_i$  as the set of groupings selected for neuron  $i$ . Finally, adds  $\mathcal{G}_i$  to  $\mathcal{G}$  or do nothing if no grouping is found(line 10 to 11).

**Algorithm 2** a strategy for selecting groupings UPDATE

---

```

1: Input: grouping set  $\mathcal{G}_i$  and the approximations  $\bar{V}_s$  and  $\underline{V}_s$ .
2: if  $\mathcal{G}_i = \emptyset$  then
3:    $\mathcal{G}_i = \{s\}$ 
4: else if  $\bar{V}_s \leq \max_{j \in \mathcal{G}_i}(\underline{V}_j)$  then
5:    $\mathcal{G}_i = \{s\}$ 
6: else if  $\underline{V}_s \geq \min_{j \in \mathcal{G}_i}(\bar{V}_j)$  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 [?] to obtain their  $(2k)$ -dimensional convex hulls. From the convex hull constraints,  $\mathcal{L}$  and  $\mathcal{U}$  for each neuron member's lower and upper bounds, respectively, after the activation function can be obtained using an LP solver. Then Algorithm ?? can be recursively applied to find groupings 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. ?? through convexity preserving:

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

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. ?? is equivalent to Equation. ??.*

*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:

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

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 ??, 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  $l_i^{\{+, -\}}(x_i)$  in one layer. The problem is turned to find 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 [?]:

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

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

and  $l \leq x \leq u$ ,  $f(\cdot)$  denotes the Sigmoid or Tanh function. For example, on Sigmoid function with  $c \geq 0$  and  $l \leq 0 \leq u$ , the interval of  $x$  is split 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  $\bar{f}$  and  $\underline{f}$  of  $x$ .

## 5 Experiments

In this section, we firstly evaluate the efficiency and effectiveness of FAGMR on ReLU-based networks on a range of challenging benchmarks, and show that it improves upon the state-of-the-art multi-neuron convex relaxation verifier with 24.7% less runtime on average and one more verified. In addition, we split the runtime into several parts and analyse out that FAGMR can reduce the time of the high-dimensional calculation. Next, we evaluate the robustness radius obtained by FAGMR to show that it can build a good balance between efficiency and effectiveness. Then, we use FAGMR to compare with more state-of-the-art verifiers to show that FAGMR is able to be a competitive verifier. Finally, we evaluate the efficiency and effectiveness of FAGMR on Sigmoid and Tanh-based reducing 8.4% runtime on average and maintaining the same verifieds compared to the state-of-the-art verifier, the results demonstrate that FAGMR can be a general method with good performance on efficiency and effectiveness.

### 5.1 Experiments Configuration

*Neural networks.* We conduct experiments on two commonly used datasets: MNIST and CIFAR-10 with 1000 images respectively. Especially, for a large network, like RestNet2b, uses 200 images. The following Table.?? shows the details about all of ReLU networks used in our experiments. These networks can be downloaded at ERAN homepage [?] or benchmarks of VNN-COMP 2021 [?], and all of the convolutional networks are trained by DiffAI [?].

*Robustness property.* We consider the  $l_\infty$ -norm perturbation on a correct classified input from the test set of the datasets above. In this setting, the perturbed inputs region is  $\mathbb{B}_\varepsilon^\infty$ . Given a perturbation threshold  $\varepsilon$ , we would like to verify that the neural network does not misclassify any perturbed input in this region. In

**Table 1.** MNIST and CIFAR-10 Neural Networks used in experiments.

DATASET	MODEL	TYPE	UNITS	LAYERS
MNIST	$5 \times 100$	FC	510	5
	$6 \times 200$	FC	1010	6
	CONVBIG	CONV	48064	6
CIFAR-10	CONVSMALL	CONV	4852	3
	RESNET2B	RESIDUAL	11364	13

principle, the perturbations  $\varepsilon$  are selected as the same as ones in experiments of PRIMA, except all of the verifiers failed to verify the benchmarks due to memory constraints, although the perturbation thresholds  $\varepsilon$  were the same as those used in the experiments of PRIMA.

*Parameters of FAGMR.* We choose  $k = 3$ , which is claimed as the optimal option in PRIMA, in the following experiments. In each activation layer, we find all the 3-neuron groupings formed by FAGMR, then these groupings are sent to the PDDM implementation [?] for tighter relations.

*Machine and software.* All experiments are done on a 12-core 2.20GHz Intel Xeon Sliver 4212 CPU platform with 64 GB of main memory. We implement FAGMR in Python 3.7 and use Gurobi 9.5.1 [?] for LP problem solving.

## 5.2 Comparison of Precision and Runtime

Since FAGMR is modified from PRIMA, we first compare the precision and runtime of FAGMR with PRIMA, focusing on the

grouping strategies for comparison which were discussed in Muller et al. [?].

**Comparison on MNIST and CIFAR-10** Table.?? shows the results of our comparison. PRIMA-para is the strategy in PRIMA’s implementation. It defines a parameter  $n_s$  to divide the neurons of a layer into sets of size  $n_s$  and then, for each set, chooses a subset of all  $C_{n_s}^k$  groupings. Additionally, there is another parameter  $s$  defined as the limited overlapping between arbitrary two groups, which means the maximum counts of identical neurons in two groupings is  $s$  ( $s = 1$  is claimed as a default option in PRIMA). PRIMA-all is the strategy considering all  $k$ -neuron groupings without parameters  $n_s$  and  $s$ , it is mainly used to show how PRIMA-para and FAGMR contribute to the precision and runtime by comparing with the most precise but time-cost strategy. In our experiments, we use the default value  $n_s = 70$  in PRIMA. For each method, given a network and a perturbation threshold  $\varepsilon$ , we list the average number of groupings (groups), average time cost (time(s)), and verified number out of inputs (veri. ).

**Table 2.** Average groupings number, average runtime, and verified number of FAGMR vs. PRIMA-all and PRIMA-para under perturbation threshold  $\varepsilon$  on ReLU networks. Acc. implies the number of original images recognized by the network. Symbol - means this strategy is out of memory under the corresponding network and perturbation threshold.

Network	Acc.	$\varepsilon$	PRIMA-all			PRIMA-para			FAGMR		
			groups	time(s)	veri.	groups	time(s)	veri.	groups	time(s)	veri.
MNIST	979	0.015	11.0	43.9	974	10.4	38.8	974	<b>8.7</b>	<b>33.5</b>	<b>974</b>
5×100		0.026	45.6	52.8	970	37.4	47.0	970	<b>25.5</b>	<b>40.3</b>	<b>970</b>
MNIST	972	0.015	1.6K	106.7	959	1.5K	68.6	958	<b>877.0</b>	<b>57.6</b>	<b>959</b>
6×200											
MNIST	929	0.03	-	-	-	-	-	-	<b>125.5</b>	<b>96.9</b>	<b>926</b>
ConvBig											
CIFAR-10	471	4/255	3.0K	49.0	452	2.3K	34.6	452	<b>1.4K</b>	<b>24.2</b>	<b>452</b>
ConvSmall											
CIFAR-10	161	1/255	-	-	-	7.0K	407.5	158	<b>1.1K</b>	<b>199.4</b>	<b>158</b>
ResNet2b											

According to the Table.??, we notice that FAGMR performs better than PRIMA-para and PRIMA-all in terms of runtime, precision, or both. Specifically, on MNIST 6×200 with  $\varepsilon = 0.015$ , FAGMR is 16.0% faster than PRIMA-para, with more verifieds, and FAGMR is 46.0% faster than PRIMA-all while having equivalent verification results. Besides, it is worth noting that in more complex networks, such as ConvBig(MNIST) with  $\varepsilon = 4/255$ , both PRIMA-all and PRIMA-para is out of memory, whereas FAGMR returns results using fewer memory resources. This means that FAGMR can be applied generally with limited memory.

It is possible to find a better balance between precision and time cost by changing the hyperparameters in PRIMA-para. However, the optimal hyperparameters depend on the size of a layer in different networks, which would sacrifice the expandability. FAGMR avoids the issues with hyperparameters to save time but sacrifices a little bit of precision compared to PRIMA-all, and also performs well in some cases. By employing our grouping strategy, FAGMR can reduce the verification time by an average of 24.7% with minimal precision loss compared to PRIMA-para. This experiment confirms the effectiveness and efficiency of the grouping strategy.

**More Details about runtime** One of the reasons why FAGMR runs faster than both PRIMA-all and PRIMA-para is that it forms fewer groupings, which causes the LP solver to deal with lesser constraints. The second reason is that we abandon groupings through an approximation strategy based on volume approximation before sending it to PDDM for computing high-dimensional convex hulls. Furthermore, we will discuss these reasons by splitting the sum of runtime into three parts, the first part is the time of generating groupings that will be sent to PDDM ( $T_1$ ), the second is the time to gain constraints by PDDM ( $T_2$ ), and the third is the time cost by LP solver ( $T_3$ ). From Table.??, we observe that the sum of  $T_1$  and  $T_2$  for FAGMR is less than that of PRIMA-all or



PRIMA-para in all cases. For instance, in MNIST  $5 \times 100$  when  $\varepsilon = 0.015$ ,  $T_1 + T_2$  of FAGMR is 70% and 53.4% respectively faster than the one of PRIMA-all and PRIMA-para. The results imply that FAGMR reduces some of the redundant groupings whose corresponding constraints are 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 grouping 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 in inducing  $T_2$  and  $T_3$ , it increases  $T_1$  for the demand for selecting groupings. 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.

From both the experiments and complexity analysis, we find that abandoning groupings through an approximation strategy has the potential to be faster than the strategy using parameters. In the following, we will discuss the sacrificed precision as the trade-off.

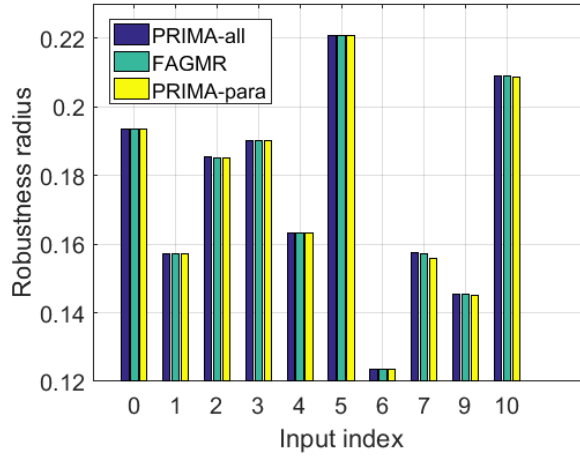
**Table 3.** The detailed runtime of FAGMR vs. PRIMA-all and PRIMA-para under perturbation threshold  $\varepsilon$ .  $T_1$  is the time of generating groupings,  $T_2$  is the time cost by PDDM, and  $T_3$  is the time of LP solving.

Network	$\varepsilon$	PRIMA-all			PRIMA-para			FAGMR		
		$T_1$ (s)	$T_2$ (s)	$T_3$ (s)	$T_1$ (s)	$T_2$ (s)	$T_3$ (s)	$T_1$ (s)	$T_2$ (s)	$T_3$ (s)
MNIST	0.015	0.4	7.7K	34.8K	0.4	5.0K	32.6K	<b>132</b>	<b>2.2K</b>	<b>30.0K</b>
$5 \times 100$	0.026	0.4	9.0K	41.1K	0.4	6.4K	39.0K	<b>350</b>	<b>2.8K</b>	<b>35.8K</b>
CIFAR-10	2/255	1.4	3.0K	2.6K	1.4	3.0K	2.6K	<b>455</b>	<b>2.1K</b>	<b>2.3K</b>
ConvSmall	4/255	2.3	15.0K	6.8K	2.3	9.9K	5.1K	<b>2.7</b>	<b>4.5K</b>	<b>3.2K</b>

### 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  $5 \times 100$  network using PRIMA-all, PRIMA-para, and FAGMR. The method of calculating robustness radius is dichotomy [?] and the iteration is 15. Figure. ?? presents the results. The blue bar in Figure. ?? 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 and 10. Figure. ?? means discarding some groupings 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. We use their default configurations and perform



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

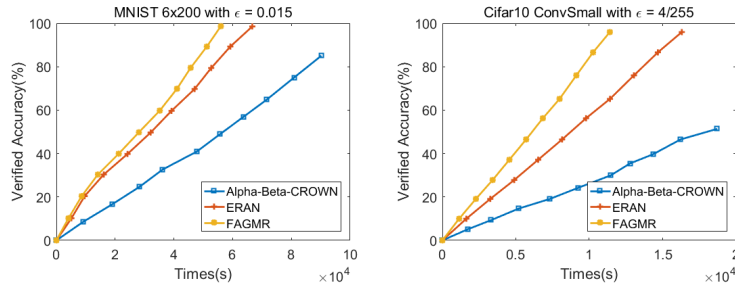
comparisons on the  $6 \times 200$  MNIST network and ConvSmall(CIFAR-10). The results show that  $\alpha\beta$ CROWN has lower precision and longer runtime than both ERAN and FAGMR. This can be attributed to  $\alpha\beta$ CROWN’s use of single-neuron relations that lead to lower precision and the fact 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.

#### 5.4 Comparison with the state-of-the-art

PRIMA has shown to perform well among many state-of-the-art verifiers, and FAGMR has been modified from PRIMA. However, we have conducted some comparisons with other state-of-the-art verifiers to demonstrate the advancement of FAGMR.

Figure ?? shows the comparison of runtime and precision of FAGMR with other state-of-the-art verifiers, including  $\alpha\beta$ CROWN [?] and ERAN, which are both outstanding verifiers in VNN-COMP 2021 [?].  $\alpha\beta$ CROWN is an efficient verifier that uses symbol propagation methods [?] represented as matrices, and its relaxation method is single-neuron based. ERAN is also an efficient verifier that relies on LP solvers, and its relaxation method is multi-neuron based. Notably, PRIMA is part of ERAN.



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

#### 5.5 Comparison with Tanh and Sigmoid activations

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  $6 \times 500$  (3000 neurons) and ConvMed (5704 neurons), both datasets can be obtained from the ERAN homepage. 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 ??, 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. Incomplete verification methods often relax non-linear activation neurons such as ReLU to speed up the verification and multi-neuron convex relaxation is one of the incomplete verification methods.

**Table 4.** Average groupings number, average runtime, and verified number of FAGMR vs. PRIMA-para under perturbation threshold  $\varepsilon$  on Sigmoid or Tanh networks

Network	Acc.	Activation	$\varepsilon$	PRIMA-para			FAGMR		
				groups	time(s)	veri.	groups	time(s)	veri.
MNIST	95	Sigmoid	0.012	6.2K	640.9	95	<b>3.8K</b>	<b>570.0</b>	<b>95</b>
6×500									
MNIST	99	Sigmoid	0.014	6.3K	255.6	99	<b>5.9K</b>	<b>252.5</b>	<b>99</b>
ConvMed									
MNIST	99	Tanh	0.005	-	-	-	<b>184.8</b>	<b>194.9</b>	<b>98</b>
6×500									
MNIST	98	Tanh	0.005	1.2K	220.4	98	<b>446.3</b>	<b>191.8</b>	<b>98</b>
ConvMed									

*Complete Verification.* Complete verification methods can describe the exact behavior of a neural network on an input region. They can be further classified into: (i) SAT/SMT based methods [?, ?, ?], which encodes the verification problem into an SAT/SMT query; (ii) Mixed-integer linear programming (MILP) based methods [?, ?, ?] with MILP; (iii) Branch-and-Bound based methods, which split non-linear activation functions into linear pieces [?, ?, ?, ?], or split the input region to be small enough so that the neural network behaves linearly on each input sub-region [?]. Complete methods are limited in scalability because of the computational complexity. Our the approach applies convex relaxation to over-approximate non-linear activation functions hence is incomplete but is much faster and more scalable.

*Incomplete Verification.* Most incomplete methods attempt to develop efficient and precise over-approximations for a given activation function, thus for each single neuron separately (e.g. [?, ?, ?, ?, ?, ?], see a survey in [?]).

*Single-neuron Relaxation.* Single-neuron convex relaxation-based methods consider each neuron separately and over-approximate its activation function, verifying rapidly and perform well on precision [?, ?, ?, ?, ?, ?]. The verification precision of such methods is proved to be limited by a convex relaxation barrier since they neglect the dependencies between neurons [?].

*Multi-neuron Relaxation.* Multi-neuron convex relaxation methods [?, ?, ?] 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 groupings and choose better ones for each neuron according to the computed volume. The experiments show

our strategy costs much less time and have more precision than the state-of-the-art tool.

## 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](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. 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](https://doi.org/10.1007/978-3-319-63387-9_5)
13. 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](https://doi.org/10.1007/978-3-030-25540-4_26)
14. 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/10.1561/24000000035>

15. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: International Conference on Machine Learning. pp. 3578–3586. PMLR (2018)
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.: 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>
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/2eed2bd94734e5dd392d8678bc64cdab-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.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>
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. 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>