kProp: Multi-neuron Relaxation Method for Neural Network Robustness Verification

Xiaoyong Xue, Xiyue Zhang, and Meng Sun(✉)

School of Mathematical Sciences, Peking University, Beijing 100871, China
{xuexy,zhangxiyue,sunm}@pku.edu.cn

Abstract. With the increasing application of neural networks in safety-critical domains, their robustness becomes a crucial concern. In this paper, we present a multi-neuron relaxation-based verification framework *kProp* for ReLU neural networks with adversarial distortions in general norms. In contrast with existing verification methods tackling general distortion norms, the proposed multi-neuron relaxation method is able to capture the relations among a group of neurons, thus providing tighter convex relaxations and improving verification precision. In addition, existing methods based on linear relaxation may include infeasible inputs to the neural network for robustness verification, which further leads to verification precision loss. To address this problem, we propose a region clipping method to exclude infeasible inputs to further improve the verification precision. We implement our verification framework and evaluate its performance on open-source benchmarks. The experiments show that *kProp* can produce precise verification results where existing verification methods fail to produce conclusive results, and can be applied to neural networks with more than 4k neurons in general distortion norms.

Keywords: Robustness · Verification · Neural network

1 Introduction

Neural networks (NNs) have been increasingly used in a broad range of applications and made inspiring breakthroughs in many safety-critical domains, such as autonomous driving, drone control, and medical diagnosis [1,3,10]. Meanwhile, a lot of studies have highlighted the vulnerability of neural networks against adversarial attacks. Adversarial attacks can be performed by applying small imperceptible perturbations to alter the NN's prediction result on the original image [9]. In addition, more practical attacks can be achieved by adding real-world physical perturbations [5]. With the increasing deployment of neural networks into safety-critical tasks, rigorous verification of NN's robustness against adversarial perturbations has gained substantial momentum in recent years.

Verification methods for neural networks mainly fall into two categories – complete and incomplete. Complete verification methods based on satisfiability

Current Address: Department of Computer Science, University of Oxford, Oxford, UK.

Published by Springer Nature Switzerland AG 2023
H. Hojjat and E. Ábrahám (Eds.): FSEN 2023, LNCS 14155, pp. 142–156, 2023.
https://doi.org/10.1007/978-3-031-42441-0_11

modulo theories (SMT) [4,11] or mixed integer linear programming (MILP) [13] can provide an exact answer of whether a neural network is robust. However, robustness verification of neural networks even with the piece-wise linear function ReLU (Rectified Linear Unit) is an NP-hard problem [11]. The worst-case exponential complexity severely restricts the application of such complete verifiers. In contrast, incomplete methods leverage various approximation techniques to attain better scalability. Approximation techniques include abstract interpretation [8,14–16] which captures the propagation from inputs to outputs in symbolic shapes, and linear relaxation [18–20] which computes linear upper and lower bounds for non-linear activation functions.

Given a neural network, the (local) robustness verification problem is to ensure that the neural network has the same prediction (such as predicted labels) on the neighborhood of an arbitrary input. Generally, the neighborhood of an input is characterized by an ℓ_p ball for a given radius $\epsilon \in \mathbb{R}^+$ with the input as the center. The aforementioned verification methods based on abstract interpretation, e.g. kPoly [15] and PRIMA [14], only consider ℓ_∞ perturbation neighborhood. However, some real-life perturbations such as adding black and white stickers [5] cannot be characterized by this formalization. It is more appropriate to capture such distortion in the form of ℓ_1 or ℓ_2 ball. Verification methods based on linear relaxation are able to verify robustness for general ℓ_p norms. The commonly-used Δ-relaxation [4] in these methods offers the tightest possible relaxation for one single neuron. However, due to the ignorance of the constraints between multiple neurons, methods based on Δ-relaxation still suffer from precision loss. In addition, verification methods based on linear relaxation make use of *Hölder Inequality* to calculate the global bounds. In this computation process, infeasible input regions are considered to derive the global bounds, which leads to more approximation loss.

In this work, we propose a propagation algorithm based on multi-neuron relaxation method to produce tighter relaxations for ReLU neural networks in Sect. 3. The overall framework of this algorithm is to propagate the verification objective from the output layer to the input layer, which yields a linear over-approximation of the original neural network and thus is able to apply to general ℓ_p distortions. The key insight of this algorithm is multi-neuron relaxation, shown in Sect. 4, to capture the relations among a group of neurons (in the same layer), which naturally leads to tighter approximation and increased verification precision. Moreover, we propose a region clipping method for infeasible input removal in Sect. 5 to further improve the verification precision.

2 Preliminary

In this section, we provide the preliminaries about neural networks, the local robustness property, and two kinds of polyhedron representation.

2.1 Neural Network

Neural networks are sequential programs that consist of an input layer, several hidden layers, and an output layer. The adjacent layers are connected

with weighted edges. A neural network N with n-dimensional input and m-dimensional output can be regarded as a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. For every neuron in hidden layers, we split it into the pre-activation neuron and the post-activation neuron. The neural network $y = f(x)$ can be formulated as follows:

$$z_{0,i} = x_i \qquad \forall i = 1 \dots n \tag{1}$$

$$\hat{z}_{l,i} = \sum_{j=1}^{n_{l-1}} w_{i,j}^l z_{l-1,j} + b_i^l \qquad \forall l = 1 \dots L, i = 1 \dots n_l \tag{2}$$

$$z_{l,i} = \sigma(\hat{z}_{l,i}) \qquad \forall l = 1 \dots L, i = 1 \dots n_l \tag{3}$$

$$y_i = z_{L,i} \qquad \forall i = 1 \dots m \tag{4}$$

The input layer is represented in Eq. (1), where each neuron takes one-dimensional value of the input data. This network has $L - 1$ hidden layers and n_l neurons for layer l. Equations (2) and (3) describe the behavior of affine transformations and non-linear transformations in terms of activation functions. Here $z_{l,i}$ is the output of the i-th neuron in layer l and $\hat{z}_{l,i}$ is the corresponding pre-activation output value. $w_{i,j}^l$ and b_i^l denote the connection weights and biases between neurons of adjacent layers. The activation function that we consider in this paper is ReLU, that is $\sigma(x) = max\{0, x\}$. Equation (4) represents the output layer where the i-th dimension of the output is y_i, also denoted as $f_i(x)$. In classification tasks, for a given input x, the neural network determines that x belongs to class t if $f_t(x) > f_k(x), \forall k \neq t, 1 \leq k \leq m$.

2.2 Robustness Property

In real-world deployment, neural networks are expected to stay stable when small perturbations occur to the input data. This safety property is referred to as local robustness [20], which states that all data that is close to the original input x_0 has the same prediction label as x_0.

Specifically, local robustness can be formalized as follows. Given a neural network f, its input domain D_f, an input data x_0 with ground-truth label l, and the distortion radius ϵ, we say the neural network satisfies local robustness in the neighborhood $\mathbb{B}_p(x_0, \epsilon)$ if

$$\forall x' \in D_f, ||x' - x_0||_p \leq \epsilon, \forall j \neq l : f_l(x') > f_j(x'). \tag{5}$$

The local robustness is represented by the conjunction of a set of inequalities, which can be verified by checking the satisfiability of each constraint.

2.3 Polyhedron Representation

The abstract domain of polyhedron is generally used in abstract interpretation for neural network verification. A bounded polyhedron can be represented as the intersection of a set of half-spaces, or the convex hull of a set of points. The former representation is called the H-representation, and the latter one is called the V-representation. Here are the formal definitions of these two representations.

Definition 1 (***H*-representation).** *A is an $m \times n$-matrix, and b is a column vector in $\mathbb{R}^m$. A polyhedron in H-representation is a region $P \subseteq \mathbb{R}^n$ that satisfies a set of linear constraints.*

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

Definition 2 (***V*-representation).** *Let $R = \{r_1, r_2, \ldots, r_m\}$ be a set of points in $\mathbb{R}^n$. A bounded polyhedron in V-representation is the convex hull of R.*

$$P = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^{m} \lambda_i r_i,\ \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \geq 0, i = 1 \ldots m\}$$

Both representations can describe a polyhedron, but each has its advantages and disadvantages. Computing intersection of polyhedra is simpler in H-representation. And the V-representation makes it easier to compute the convex hull. We can use the Double Description Method [6] to transform one to another.

3 Propagation Framework

In this section, we present the general propagation framework with multi-neuron relaxation and region clipping to compute tighter convex relaxation of neural networks and more precise verification results against adversarial distortions in general ℓ_p norms, which is shown in Fig. 1.

The idea of layer-by-layer propagation from output to the input has been widely used in many neural network verification methods [16,17,19,20]. As shown in Fig. 1, the propagation procedure begins from the output layer. Specifically, the verification objective $t = y_o - y_l$ is characterized by the linear inequalities in Eq. (5), where l is the prediction label of the given input and $o \neq l$. If $t < 0$, then the local robustness of the neural network on the given input data within the adversarial distortions is guaranteed.

The verification objective t is then propagated from the output to the input layer by layer. However, the non-convexity of activation functions is the obstacle in the backward propagation of the linear objective. Therefore, in this process,

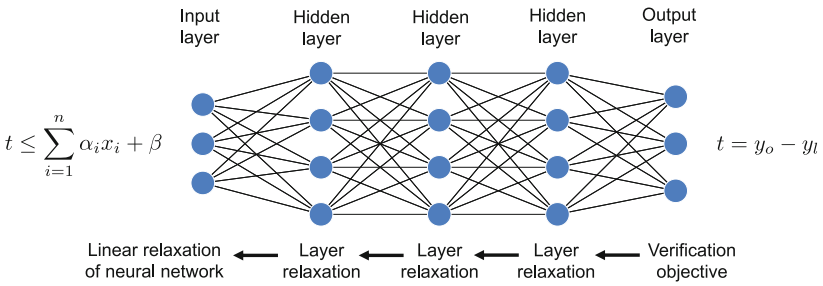


Fig. 1. Backward propagation framework

we maintain an over-approximation of t by performing linear relaxations for the activation functions in each hidden layer. In this way, we can transform the linear inequalities to the input layer and obtain a linear relaxation of the neural network. When the verification objective is propagated to the input layer, the over-approximation of t is a linear combination of input variables. We then calculate the upper bound of t restricted by the input constraints.

The most crucial part in the propagation framework is the linear relaxation. For neuron $\hat{z}_{k,i}$, we compute its two scalar bounds $ub_{k,i}$ and $lb_{k,i}$ that satisfy $lb_{k,i} \leq \hat{z}_{k,i} \leq ub_{k,i}$ for any $x' \in \mathbb{B}_p(x, \epsilon)$. The neurons of hidden layers can be categorized into three types according to the scalar bounds.

- If the neuron is always activated ($lb_{k,i} \geq 0$), we have $z_{k,i} = \hat{z}_{k,i}$.
- If the neuron is always deactivated ($ub_{k,i} \leq 0$), we have $z_{k,i} = 0$.
- If the neuron is unstable ($lb_{k,i} < 0 < ub_{k,i}$), we perform linear relaxation.

Existing verification methods use an upper bounding function $\mathcal{U}_{k,i}$ and a lower bounding function $\mathcal{L}_{k,i}$ for each unstable neuron $z_{k,i}$. These two functions are subject to the following inequality: $\mathcal{L}_{k,i}(\hat{z}_{k,i}) \leq \sigma(\hat{z}_{k,i}) \leq \mathcal{U}_{k,i}(\hat{z}_{k,i})$. The most frequently used bounding functions are linear functions, which can be calculated based on the scalar bounds of the pre-activation neurons as follows:

$$\mathcal{U}_{k,i}(\hat{z}_{k,i}) = \frac{ub_{k,i}}{ub_{k,i} - lb_{k,i}}(\hat{z}_{k,i} - lb_{k,i}) \qquad \mathcal{L}^1_{k,i}(\hat{z}_{k,i}) = \hat{z}_{k,i} \qquad \mathcal{L}^2_{k,i}(\hat{z}_{k,i}) = 0$$

The above bounding functions are used in many verification methods [16,20]. Generally, the upper bounding function has only one candidate. But the lower bounding functions are adaptively selected from $\mathcal{L}^1_{k,i}$ and $\mathcal{L}^2_{k,i}$. The bounding function that minimizes the area between the activation function and lower bound is chosen, which means $\mathcal{L}_{k,i}(\hat{z}_{k,i}) = \hat{z}_{k,i}$ is selected if $ub_{k,i} + lb_{k,i} \geq 0$, and $\mathcal{L}_{k,i}(\hat{z}_{k,i}) = 0$ is selected if $ub_{k,i} + lb_{k,i} < 0$.

Existing works mentioned above only consider the relaxation on one single neuron, losing sight of relations among neurons in the same hidden layer. We aim to capture the relations between multiple neurons and obtain tighter convex relaxation by calculating the joint bounding function for a group of neurons.

We propose a multi-neuron relaxation based verification framework as shown in Algorithm 1. To calculate joint bounding functions for a group of neurons, the first step is to compute bounding functions for each single neuron through a fast linear relaxation method (line 1). In the backward propagation process, neurons that are always activated or deactivated can be directly propagated to the pre-activation layer (line 5 - 8). For the remaining unstable neurons, we gather them together (line 9) and perform multi-neuron relaxation (line 10 - 15). Computing joint bounding functions for all unstable neurons is practically infeasible for large-scale neuron networks. To achieve better scalability, we divide the unstable neurons into several non-overlapping groups and calculate bounding functions for each neuron group. Each group is formed by randomly selecting k unstable neurons.

Based on the multi-neuron relaxation method, we can propagate the verification objective to the preceding layer according to Eq. (2) (line 17). Through

Algorithm 1: Propagation Framework

Input: Verification objective $t = \sum_{i=1}^{n_{L-1}} c_{L-1,i} z_{L-1,i} + \beta_{L-1}$,
the given input x_0, radius ϵ, norm p, weights $w_{i,j}^l$, biases b_i^l,
number of neurons in a group k

Output: An upper bound of t

ub, lb, $\mathcal{U}$, $\mathcal{L} \leftarrow$ InitalBounding(x_0, ϵ, p)
for $l \leftarrow$ *L-1, . . . , 1* **do**
  `unstable_neurons` $\leftarrow \{\}$
  for $i \leftarrow 1$ **to** n_l **do**
    if $lb_{l,i} \geq 0$ **then**
      $t \leftarrow t - c_{l,i} z_{l,i} + c_{l,i} \hat{z}_{l,i}$
    else if $ub_{l,i} \leq 0$ **then**
      $t \leftarrow t - c_{l,i} z_{l,i} + 0 \cdot \hat{z}_{l,i}$
    else add $z_{l,i}$ to `unstable_neurons`
  while `unstable_neurons` *is not empty* **do**
    Pop k neurons $z_{l,u_1}, z_{l,u_2}, \ldots, z_{l,u_k}$ from `unstable_neurons`
    `U_group` $\leftarrow [\, \mathcal{U}_{l,u_1}, \mathcal{U}_{l,u_2}, \ldots, \mathcal{U}_{l,u_k} \,]$
    `L_group` $\leftarrow [\, \mathcal{L}_{l,u_1}, \mathcal{L}_{l,u_2}, \ldots, \mathcal{L}_{l,u_k} \,]$
    `upper_bound` $\leftarrow$ JointBound($\sum_{i=1}^{k} c_{l,ui} z_{l,ui}$, `U_group`, `L_group`, x_0, ϵ, p)
    $t \leftarrow t - \sum_{i=1}^{k} c_{l,ui} z_{l,ui} +$ `upper_bound`
  for $i \leftarrow 1$ **to** n_l **do**
    $t \leftarrow t - \hat{c}_{l,i} \hat{z}_{l,i} + \sum_{j=1}^{n_{l-1}} \hat{c}_{l,i} w_{i,j}^l z_{l-1,j} + \hat{c}_{l,i} * b_i^l$
`res` $\leftarrow$ GlobalBound(t, x_0, ϵ, p)
return `res`

repeating the above procedure for every hidden layer in a backward manner, we can obtain a linear relaxation of the verification objective, which is in the form of $t \leq \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n + \beta$.

The last step of this algorithm is to find a global upper bound, the maximum value of t, with regard to the input perturbations. For any input x_0, we can use *Hölder Inequality* to find the solution in $\mathbb{B}_p(x_0, \epsilon)$ [20]. However, some regions of this ball are not included in the input domain of the neural network. To address this problem, we propose the region clipping method in Sect. 5.

4 Multi-neuron Relaxation

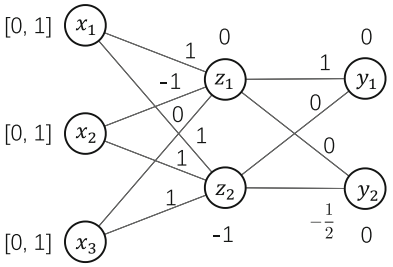
In this section, we introduce the insight of multi-neuron relaxation over single neuron relaxation and how to calculate the multi-neuron relaxation for a group of unstable neurons.

4.1 Motivation Example

We first show the superiority of multi-neuron relaxation with a simple example.

Example 1. Consider a neural network with one hidden layer. It has three neurons in the input layer, two neurons in the hidden layer, and two neurons in the output layer. The structure of this neural network is illustrated with the following equations and figure.

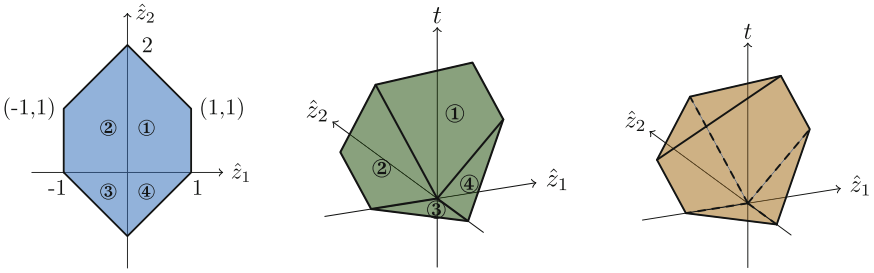
$$
\begin{aligned}
x_1, x_2, x_3 &\in [0,1] \\
\hat{z}_1 &= x_1 - x_2 \\
\hat{z}_2 &= x_1 + x_2 + x_3 - 1 \\
z_1 &= ReLU(\hat{z}_1) \\
z_2 &= ReLU(\hat{z}_2) \\
y_1 = z_1 \quad y_2 &= -\frac{1}{2} z_2
\end{aligned}
$$



The range of each input neuron is $[0,1]$*. In this example, we attempt to find the upper bound of* $t = y_1 - y_2 = z_1 + \frac{1}{2} z_2$*.*

We first perform single neuron relaxation using the bounding functions shown in Sect. 3. According to the value range of x_1, x_2, x_3, we can calculate that the lower scalar bound and upper scalar bound of $\hat{z}_1$ are $-1, 1$, and those of $\hat{z}_2$ are $-1, 2$. The single-neuron upper bounding function for z_1 and z_2 are $\mathcal{U}_1 = \frac{1}{2}\hat{z}_1 + \frac{1}{2}$ and $\mathcal{U}_2 = \frac{2}{3}\hat{z}_2 + \frac{2}{3}$.

With the above bounding function we have $t \leq \frac{1}{2}\hat{z}_1 + \frac{1}{3}\hat{z}_2 + \frac{5}{6}$. By substituting $\hat{z}_1, \hat{z}_2$ with x_1, x_2, x_3, we have $t \leq \frac{5}{6}x_1 - \frac{1}{6}x_2 + \frac{1}{3}x_3 + \frac{1}{2}$. Considering the input range, the upper bound of t with single neuron relaxation is $\frac{5}{3}$.



(a) Value range of $\hat{z}_1, \hat{z}_2$ (b) Formulated region of (a) (c) Convex hull of (b)

Fig. 2. Relations of hidden neurons in Example 1

In single neuron relaxation, we only make use of the scalar upper and lower bounds. The pre-activation neurons $\hat{z}_1$ and $\hat{z}_2$ are treated to be independent of each other. However, they are not independent. In this example, the values

Algorithm 2: Computing Joint Bound

Input: Function $t = \sum_{i=1}^{k} c_i z_i$, upper bounding functions `U_group`, lower bounding functions `L_group`, the given input x_0, radius ϵ, norm p
Output: An upper bounding function of t

```
1  octahedron ← OctaheralAbstraction(U_group, L_group, x_0, ϵ, p)
2  segments ← Split(octahedron, k)
3  generators ← {}
4  for each segment in segments do
5      vertices ← GetVertices(segment)
6      for each (v_1, ..., v_k) in vertices do
7          lifted_v ← (v_1, ..., v_k, Σ_{i=1}^{k} ReLU(v_i))
8          add lifted_v to generators
9  bounds ← GetFacets(generators)
10 upper_bound ← BoundSelection(bounds, generators)
11 return upper_bound
```

of $\hat{z}_1$ and $\hat{z}_2$ are taken from blue region as shown in Fig. 2(a). Then the image of function $t = ReLU(\hat{z}_1) + \frac{1}{2}ReLU(\hat{z}_2)$ over the blue region is illustrated in Fig. 2(b). This image is composed by four planes as the function is linear in each quadrant. As this function is piece-wise linear, the formulated region in $(\hat{z}_1, \hat{z}_2, t)$-space is non-convex. To calculate the joint bounding functions, we calculate the convex hull of this region as shown in Fig. 2(c) where each facet of the convex hull can be transformed into a bounding function.

Considering the orientations of facets in Fig. 2(c), we can obtain two upper bounding functions. Using the bound selection algorithm (Sect. 4.3), $t \leq \frac{1}{2}\hat{z}_1 + \hat{z}_2 + \frac{1}{2}$ is selected to be the upper bounding function. After propagating this joint bound to the input layer, we have $t \leq x_1 + \frac{1}{2}x_3$. The upper bound of t is then calculated to be $\frac{3}{2}$ with multi-neuron relaxation, which is tighter than the obtained upper bound with single neuron relaxation.

4.2 Joint Bounding Function

In this subsection we present how to compute the joint bounding functions for multi-neuron relaxation.

The multi-neuron relaxation algorithm is shown in Algorithm 2. It can be divided into three steps: determining the value range of the pre-activation neurons, constructing the corresponding region in the $(\hat{\mathbf{z}}, t)$-space where $\hat{\mathbf{z}} = (\hat{z}_1, \ldots, \hat{z}_k)$, and computing the convex hull of this region.

We use octahedral abstraction [14,15] to determine the value range of the pre-activation neurons (line 1). Specifically, the octahedral abstraction is represented by a series of linear inequalities that over-approximate the values of a group of neurons, i.e., $\{\sum_{i=1}^{k} d_i \hat{z}_i \leq e_i \mid d_i \in \{-1, 0, 1\}, d_i \text{ are not all zero.}\}$. The constant term e_i is generated by computing the upper bound of $\sum_{i=1}^{k} d_i \hat{z}_i$, which utilizes the single neuron bounding functions.

Next we construct the region in the $(\hat{\mathbf{z}}, t)$-space, which corresponds to the value range of pre-activation neurons (line 2 - 7). As ReLU is linear in each orthant, the input domain is split into a list of subregions by adding constraints $\hat{z}_i \geq 0$ or $\hat{z}_i \leq 0$ (line 2). If there are k neurons in a group, the number of produced subregions is at most 2^k.

Example 2. The value range of $\hat{z}_1, \hat{z}_2$ in Example 1 can be described with the following linear inequalities.

$$\hat{z}_1 \leq 1, -\hat{z}_1 \leq 1, \hat{z}_1 \leq 2, -\hat{z}_1 \leq 1, \hat{z}_1 + \hat{z}_2 \leq 2, -\hat{z}_1 + \hat{z}_2 \leq 2, \hat{z}_1 - \hat{z}_2 \leq 1, -\hat{z}_1 - \hat{z}_2 \leq 1$$

This can be split into 4 subregions by adding inequalities like $\hat{z}_1 \sim 0, \hat{z}_2 \sim 0$ ($\sim$ is $\leq$ or $\geq$). For example, by adding $\hat{z}_1 \geq 0, \hat{z}_2 \geq 0$ the upper right quadrant (after simplification) is $\hat{z}_1 \leq 1,\ -\hat{z}_1 \leq 0,\ -\hat{z}_2 \leq 0,\ \hat{z}_1 + \hat{z}_2 \leq 2$.

In each subregion, as the activation function is linear, the constituted region in $(\hat{\mathbf{z}}, t)$-space is a plane, which can be represented with linear inequalities, i.e., H-representation. For the convenience of computing the convex hull of the constituted region in $(\hat{\mathbf{z}}, t)$-space, we transform this into the V-representation. We firstly use the Double Description Method to transform the subregion into the V-presentation and get its vertices (line 5). Then for each vertex v, we compute its corresponding t value and concatenate it with v (line 7). In this way, we lift this vertex to the $(\hat{\mathbf{z}}, t)$-space, and we get the V-representation of the formulated region in the $(\hat{\mathbf{z}}, t)$-space.

Example 3. The vertices of the quadrant in Example 2 are $(0, 2)$, $(0, 0)$, $(1, 0)$, $(1, 1)$. The output function is $t = ReLU(\hat{z}_1) + \frac{1}{2}ReLU(\hat{z}_2)$. For vertex $(0, 2)$, we have $t = ReLU(0) + \frac{1}{2}ReLU(2) = 1$. Concatenate this with $(0, 2)$, and we can get the lifted vertex $(0, 2, 1)$. In the same way, the other three lifted vertices are $(0, 0, 0)$, $(1, 0, 1)$, $(1, 1, \frac{3}{2})$. The corresponding region of the upper right quadrant in $(\hat{z}_1, \hat{z}_2, t)$-space is represented as the convex hull of these 4 lifted points, i.e., V-representation.

The last step is to compute the convex hull of all subregions in $(\hat{z}_1, \ldots, \hat{z}, t)$-space. Since all formulated subregions are in V-representation, we just need to gather all the vertices (line 8) to get the V-representation of the convex hull. Then we can use Double Description Method again to transform the convex hull into H-representation (line 9). Each inequality of the H-representation is a facet of the convex hull, and thus a bounding function in multi-neuron relaxation. We can determine the orientation of a facet with the coefficient of the output variable.

Example 4. By gathering all the vertices, we can get the V-representation of the convex hull in Fig. 2(c). And with Double Description Method, we can get the facets of this convex hull.

$$-\hat{z}_1 + 2t \leq 2,\ -\hat{z}_1 - \hat{z}_2 + 2t \leq 1$$
$$-t \leq 0,\ \hat{z}_1 - t \leq 0,\ \hat{z}_2 - t \leq 0,\ 2\hat{z}_1 + \hat{z}_2 - 2t \leq 0$$

The coefficients of t in the first two inequalities are positive, thus they are the upper bounding functions.

4.3 Bounding Function Selection

We have introduced how to compute the bounding functions in the previous subsection. Note that the convex hull of the constructed region may have more than one facet, and thus the bounding functions may not be unique. For example, there are two upper bounding functions in Example 4. However, only one bounding function can be adopted in the propagation framework for computational efficiency. To address this problem, we propose an approach to select the bounding functions (line 10).

Similar to the adaptive selection in single neuron relaxation, we choose the bounding function that minimizes the difference between the bounding function and the original activation function. Specifically, we measure the difference between the bounding function and activation function on the region vertices. After splitting the value range of the pre-activation neurons, we can gather the vertices of all subregions together. We then calculate the sum of the differences between the bounding function and activation function on these vertices.

$$\text{difference} = \sum_{p \in \texttt{generators}} |bounding(p) - activation(p)|$$

The bounding function with the minimum difference is chosen to be the best and applied in the propagation framework.

Example 5. There are two upper bounding functions in Example 4: $f_1 = \frac{1}{2}\hat{z}_1 + \frac{1}{2}\hat{z}_2 + \frac{1}{2}$, *and* $f_2 = \frac{1}{2}\hat{z}_1 + 1$ *The vertices are* $(0, 2)$, $(1, 1)$, $(1, 0)$, $(0, 0)$, $(0, -1)$, $(-1, 0)$, $(-1, 1)$. *For bounding function* $f_1 = \frac{1}{2}\hat{z}_1 + \frac{1}{2}\hat{z}_2 + \frac{1}{2}$,

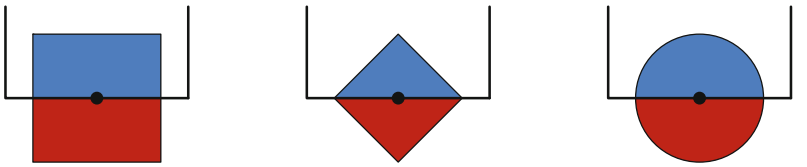
$$diff_1 = |\frac{3}{2} - 1| + |\frac{3}{2} - \frac{3}{2}| + |1 - 1| + |\frac{1}{2} - 0| + |0 - 0| + |0 - 0| + |\frac{1}{2} - \frac{1}{2}| = 1$$

For bounding function f_2, *we have* $diff_2 = 3$ *in the same way. The first bounding function is closer to the activation function than the second one. Therefore, we choose* f_1 *as the upper bounding function in Example 1.*

5 Region Clipping

As introduced in Sect. 3, simply using *Hölder Inequality* may lead to verification precision loss because the derived global upper bound of the neural network may take infeasible inputs into consideration. For example, as shown in Fig. 3, when the given data point (black dot) lies on the boundary of the input region, some portion of the ℓ_p ball (red part) is not included in the neural network's input domain.

For robustness verification, we only need to consider the intersection of the neural network's input domain and the distortion neighborhood, which is represented by the blue region in Fig. 3. Clipping out infeasible inputs to the neural network can assist in computing a tighter global upper bound, thus increasing the verification precision.



(a) l_∞ neighborhood (b) l_1 neighborhood (c) l_2 neighborhood

Fig. 3. The neighborhood of a borderline input.

Without loss of generality, we assume the value range of each input neuron is $[0, 1]$. Considering robustness verification with respect to $\mathbb{B}_p(x_0, \epsilon)$, computing the global bound of the neural network in the clipped input region can be formulated as the following constrained optimization problem

$$\gamma = \max_{x'} \sum_{i=1}^{n} \alpha_i x'_i + \beta \quad s.t.\ x'_i \in [0, 1],\ ||x' - x_0||_p \leq \epsilon$$

We can reformulate the above problem by setting $v_i = x'_i - x_{0,i}$.

$$\gamma = \max_{v} \sum_{i=1}^{n} \alpha_i v_i + \sum_{i=1}^{n} \alpha_i x_{0,i} + \beta \quad s.t. - x_{0,i} \leq v_i \leq 1 - x_{0,i},\ \sum_{i=1}^{n} |v_i|^p \leq \epsilon^p$$

The last two terms of γ are constants. So we just need to find the maximum value of the first term. Solving this optimization problem with respect to l_∞ neighborhood is trivial. We just need to clip the illegal value range of each input variable and the resulted feasible region is still a box. However, for the other cases, the feasible region is irregular. Next we propose the region clipping methods for ℓ_1 neighborhood and ℓ_p $(p \geq 2)$ neighborhood separately.

For ℓ_1 neighborhood, we sort the perturbation variables in non-increasing order according to the absolute values of their coefficients. This is because variable with larger absolute value of coefficient has more influence on the optimization objective. Therefore, we maximize the perturbation variables one by one in this order until either reaching the boundary of the feasible region or exhausting the allowed distortions.

Region clipping for ℓ_p $(p \geq 2)$ neighborhood is presented in Algorithm 3. As with region clipping for ℓ_1 neighborhood, we only maximize the first term of γ. This optimization problem can be solved by Lagrange multiplier method. We can construct the Lagrangian function and obtain the Karush-Kuhn-Tucker (KKT) conditions. As the objective function is linear and the inequality constraints are continuously differentiable convex functions, the satisfaction of KKT conditions are sufficient and necessary conditions for the optimal solution. We can find the optimal solution along with the direction of the gradient. If the boundary of a linear constraint is encountered, we fix the corresponding distortion v_i and optimize the remaining variables. The solution found by Algorithm 3 satisfies the KKT conditions, thus is the optimal solution.

Algorithm 3: Region clipping for l_p neighborhood

Input: The objective function $\sum_{i=1}^{n} \alpha_i v_i$, the given data point x_0, neighborhood radius ϵ, norm p
Output: the maximum value of $\sum_{i=1}^{n} \alpha_i v_i$

$\mathtt{r}_i \leftarrow 0$ for all $i = 1, \ldots, n$
for $i \leftarrow 1 \ldots n$ **do**
  if $\alpha_{k_i} > 0$ **then**
    $\mathtt{r}_i \leftarrow 1 - x_{0,i}$
  else
    $\mathtt{r}_i \leftarrow -x_{0,i}$
$q \leftarrow \frac{1}{p-1}$
$\{\alpha_{k_1}, \ldots, \alpha_{k_n}\} \leftarrow$ sort $\{\alpha_1, \ldots, \alpha_n\}$ in non-decreasing order according to $\frac{\mathtt{r}_i}{\alpha_i^q}$
$\mathtt{remain}, \gamma \leftarrow \epsilon^p, 0$
for $i = 1$ **to** n **do**
  if $\frac{r_{k_i}^p}{\alpha_{k_i}^{pq}} \sum_{j=i}^{n} \alpha_{k_j}^{pq} \geq \mathtt{remain}$ **then**
    $\gamma \leftarrow \gamma + (\sum_{j=i}^{n} \alpha_{k_j}^{pq})^{\frac{1}{pq}} \cdot \mathtt{remain}^{\frac{1}{p}}$
    break
  else
    $\gamma \leftarrow \gamma + \alpha_{k_i} \cdot \mathtt{r}_{k_i}$
    $\mathtt{remain} \leftarrow \mathtt{remain} - r_{k_i}^p$
return γ

6 Experiments

We implement the propagation framework with multi-neuron relaxation and region clipping as *kProp*. To show the effectiveness of our algorithm, we compare *kProp* with two widely-used robustness verifiers, DeepPoly [16] and CROWN [20]. DeepPoly is an efficient verifier with high precision, but it can only be used for distortions of l_∞ norm. CROWN can verify the robustness of neural networks with regard to general l_p norms. But CROWN simply uses *Hölder Inequality* to calculate the global bound of the final optimization problem. Both of them adopt the single neuron relaxation.

Neural Networks and Datasets. The neural networks used in our experiments are well-trained models from the publicly available ERAN dataset [7]. We conduct experiments on both feed-forward neural networks (FNNs) and convolutional neural networks (CNNs) trained on MNIST [12] and CIFAR-10 [2] datasets. The feed-forward neural network with a hidden layers and b neurons per hidden layer is denoted as $a \times b$. The convolutional neural network denoted as Conv has two convolutional layers and two fully-connected layers. For each neural network, we use the first 1000 images from the corresponding test data as the test images and filter out the misclassified images.

Problem Settings. The properties considered in the experiments are local robustness with respect to distortions in ℓ_∞, ℓ_1 and ℓ_2 norms. The radius of ℓ_p ball ϵ is set to different values in different settings to avoid meaningless results. Experiments conducted on neural networks with respect to l_∞ norm are set to smaller radius than those with respect to l_1 norm. The detailed radius settings ϵ are shown in Table 1. For all neural networks and norms, we use $k = 3$ in $kProp$ to balance precision and time cost. This is shown in Table 2.

Table 1. Number of verified local robustness properties.

Dataset	Model	ℓ_∞ norm			ℓ_1 norm			ℓ_2 norm		
		ϵ	$kProp$	DeepPoly	ϵ	$kProp$	CROWN	ϵ	$kProp$	CROWN
MNIST	6×100	0.026	**174**	160	2.5	**350**	223	0.3	**546**	287
	9 × 100	0.026	**186**	182	2.5	**309**	219	0.3	**456**	272
	6 × 200	0.015	**303**	292	2	**303**	144	0.25	**342**	116
	9 × 200	0.015	**262**	259	2	**188**	132	0.25	**276**	112
	Conv	0.12	**158**	**158**	1.5	**766**	367	0.6	**486**	137
CIFAR-10	6 × 100	0.002	**55**	54	1	**112**	97	0.07	**106**	99
	9 × 200	0.002	**63**	**63**	1	**136**	112	0.07	**127**	123
	Conv	0.01	**274**	256	0.5	**303**	268	0.12	**307**	280

Experiment Results. Table 1 shows the number of verified local robustness properties for common distortions in terms of ℓ_∞, ℓ_1, and ℓ_2 norms based on different verification algorithms. In general, our method demonstrates better verification precision and outperforms DeepPoly and CROWN or achieves comparable performance for all verification problems.

For ℓ_1 and ℓ_2 norms, $kProp$ shows great superiority over CROWN with tighter convex relaxation through the multi-neuron relaxation method and tighter global bounds through region clipping. For MNIST dataset, the number of verified properties by $kProp$ is at least 40% more than those of CROWN. The precision gain is especially noticeable on convolution neural networks. $kProp$ successfully verifies 766 problems for ℓ_1 norm and 486 problems for ℓ_2 norm, whereas CROWN verifies 367 and 486 problems respectively. For the CIFAR-10 dataset, the improvements on convolutional neural networks are also more significant compared with FNNs. For ℓ_∞ norm, we only perform comparison experiments with DeepPoly as it demonstrates better verification performance than CROWN in this case. DeepPoly also performs region clipping for distortions of ℓ_∞ norm. Therefore, the results mainly demonstrate the effect of multi-neuron relaxation. We can see that $kProp$ is able to verify more robustness problems than DeepPoly which indicates that multi-neuron relaxation can provider tighter bounds than single neuron relaxation. The computational cost of $kProp$ is acceptable. For the most complicated verification task, robustness of CNN trained on CIFAR-10 with respect to ℓ_2 norm, the average runtime cost of each problem is less than 12 min. For verification tasks on FNNs with respect to ℓ_1 or ℓ_∞ norms, $kProp$ is able to finish in a few seconds.

The Choice of k. To explore the influence of parameter k in *kProp*, we perform experiments on the 6×100 FNN and MNIST dataset for different k. The number of verified properties and corresponding runtime are shown in Table 2. With a larger k, we can capture more complicated relations among neurons, and thus generate tighter bounding functions. However, this can take plenty of time. On the contrary, smaller k costs less time but provides looser bounding functions. To balance precision and efficiency, we chose $k = 3$ in the previous experiment.

Table 2. Number of verified properties of 6×100 FNN and runtime for $k = 2, 3, 4$.

Norm	ϵ	k=2		k=3		k=4	
		verified(#)	time(s)	verified(#)	time(s)	verified(#)	time(s)
ℓ_∞	0.026	166	1.99	174	4.61	174	187.71
ℓ_1	2.5	344	5.90	350	9.83	351	131.92
ℓ_2	0.3	546	35.33	550	56.87	550	180.06

7 Conclusion

We presented a multi-neuron based robustness verification framework *kProp* to verify the local robustness of neural networks for general ℓ_p norms. *kProp* is featured with constraint propagation, multi-neuron relaxation, and region clipping. The propagation framework enables *kProp* to verify robustness properties for general ℓ_p norms. The multi-neuron relaxation and region clipping together improve the verification precision. We implement our algorithm and evaluate it on a set of neural networks with different sizes, which demonstrates the effectiveness of our method. In the future, we would like to extend the application range of our method to more activation functions and network architectures.

Acknowledgements. This research was sponsored by the National Natural Science Foundation of China under Grant No. 62172019, 61772038, and CCF-Huawei Formal Verification Innovation Research Plan.

References

1. Amato, F., López, A., Peña-Méndez, E.M., Vaňhara, P., Hampl, A., Havel, J.: Artificial neural networks in medical diagnosis. J. Appl. Biomed. **11**(2), 47–58 (2013)
2. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE (2017)
3. Chen, Z., Huang, X.: End-to-end learning for lane keeping of self-driving cars. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 1856–1860. IEEE (2017)

4. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
5. Eykholt, K., et al.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1625–1634 (2018)
6. Fukuda, K., Prodon, A.: Double description method revisited. In: Deza, M., Euler, R., Manoussakis, I. (eds.) CCS 1995. LNCS, vol. 1120, pp. 91–111. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61576-8_77
7. Gagandeep, S., et al.: Eran verification dataset. https://github.com/eth-sri/eran
8. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE (2018)
9. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations (ICLR), Conference Track Proceedings (2015)
10. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1–10. IEEE (2016)
11. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
13. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks (2017). https://arxiv.org/abs/1706.07351
14. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: general and precise neural network certification via scalable convex hull approximations. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022)
15. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 32, pp. 15072–15083 (2019)
16. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019)
17. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 31, pp. 6369–6379 (2018)
18. Wang, S., et al.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 34, pp. 29909–29921 (2021)
19. Weng, L., et al.: Towards fast computation of certified robustness for ReLU networks. In: Proceedings of the 35th International Conference on Machine Learning (ICML), vol. 80, pp. 5276–5285. PMLR (2018)
20. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 31, pp. 4944–4953 (2018)