Divide and Slide: Layer-Wise Refinement for Output Range Analysis of Deep Neural Networks

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, Qi Zhu

***Abstract*—In this paper, we present a layer-wise refinement method for neural network output range analysis. While approaches such as nonlinear programming (NLP) can directly model the high nonlinearity brought by neural networks in output range analysis, they are known to be difficult to solve in general. We propose to use a convex polygonal relaxation (over-approximation) of the activation functions to cope with the nonlinearity. This allows us to encode the relaxed problem into a mixed integer linear program (MILP), and control the tightness of the relaxation by adjusting the number of segments in the polygon. Starting with a segment number of 1 for each neuron, which coincides with a linear programming (LP) relaxation, our approach selects neurons layer by layer to iteratively refine this relaxation. To tackle the increase of the number of integer variables with tighter refinement, we bridge the propagation-based method and the programming-based method by dividing and sliding the layer-wise constraints. Specifically, given a sliding number s, for the neurons in layer l, we only encode the constraints of the layers between $l-s$ and l. We show that our overall framework is sound and provides a valid over-approximation. Experiments on deep neural networks demonstrate significant improvement on output range analysis precision using our approach compared to the state-of-the-art.**

***Index Terms*—Neural networks, output range analysis, refinement, linear programming, mixed integer linear programming.**

I. Introduction

Neural networks have shown promising applications in a variety of domains, including safety-critical systems such as self-driving cars and medical devices. However, to ensure system safety and security, more formal analysis of neural networks is needed before they can be widely applied in practice. As observed in [1], some of the key correctness problems of neural networks, such as adversarial robustness [2], [3], [4] and reachability analysis of neural-network controlled systems [5], [6], [7], can be converted to the analysis of their output range. Thus, addressing the output range analysis problem is vital to provide guarantees for the safety and security of neural networks.

Informally, output range analysis solves the following problem: given a neural network f and the input range $\mathcal{X}$, compute the output range of $f(\mathcal{X})$. Since a neural network is highly nonlinear due to the large number of parameters and nonlinear activation functions, it is generally difficult to compute the exact range. In most cases, we use an overapproximation $\overline{\mathcal{Y}}$ such that $f(\mathcal{X}) \subseteq \overline{\mathcal{Y}}$. Such overapproximation can provide an explicit bound for determining whether the neural network output falls into an unwanted region. Early work shows that basic interval bound propagation (IBP) can be used to tackle this problem, but often leads to an overly loose estimation due to the loss of dependencies across layers [8].

State-of-the-art methods for output range analysis mainly fall into two categories: symbolic interval propagation (SIP) [9], [10] and constraint programming (CP) [11], [12], where the overapproximation is computed in different manners. The main drawback with SIP is that it can hardly propagate the dependencies for nonlinear operations across layers, and the performance of these propagation-based methods declines with deeper networks. On the other hand, CP-based methods need to solve a large nonlinear programming problem encoding the entire network and suffer from the curse of dimension.

In this paper, we propose a layer-wise refinement method that bridges propagation-based methods with mixed-integer linear programming (MILP) by using sliding windows. Specifically, we first compute the interval relaxation for each operation with a propagation-based method as the initialization step. Based on the initial range, we use a linear programming (LP) relaxation approach to better approximate the variable range. Then, the relaxation can be further tightened by the MILP encoding. Our approach iteratively improves the approximation precision by increasing the number of integer variables. In addition, we refine the variable range such that fewer integer variables are needed to achieve a similar approximation precision. Furthermore, to alleviate the complexity of MILP brought from the large number of integer variables, we encode the constraints in a propagation manner, which divides and slides the neural network by layers and handles the encoding within each sliding windows. Intuitively, given a length of sliding window s, for the neuron in layer l, we only encode the constraints of the layers between $l-s$ and l. With these methods, we can effectively manage the size of MILP and handle deep networks.

In summary, our paper makes the following contributions:

- We propose an iterative framework for output range analysis of neural networks, using a combination of interval, LP, and MILP relaxation;
- We develop a convex polygonal relaxation method for nonlinear operations with the ability to tune the over-approximation tightness based on the number of integer

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.

C. Huang and Q. Zhu are with Northwestern University, Evanston, Illinois. (email: chao.huang@northwestern.edu, qzhu@northwestern.edu)

J. Fan and W. Li are with Boston University, Boston, Massachusetts. (email: jmfan@bu.edu, wenchao@bu.edu)

X. Chen is with University of Dayton, Dayton, Ohio. (email: xchen4@udayton.edu)

variables, and we prove soundness and convergence of the overapproximation;
- We leverage a sliding window method to encode partial constraints to further reduce MILP complexity, while maintaining the soundness of our approach;
- We show that on multiple benchmarks, our approach can provide significantly tighter output range overapproximation than the current state-of-the-art method [13]. Our approach also applies to neural networks with different architectures and activation functions.

The rest of the paper is structured as follows. Section II introduces the state-of-the-arts on the output range analysis via different techniques. Section III introduces the neural network model and formulates the output range analysis problem. Section IV and Section V present our approach, where Section IV shows the interval, LP, and MILP relaxation as well as the guarantees on soundness and convergence. Section V proposes the sliding-window based propagation method. Section VI presents the experimental results and Section VIII concludes the paper.

II. Related Work

A main idea on output range analysis of neural networks is interval bound propagation (IBP) [8], [14], [15]. It leverages the monotonicity of the operations. Thus the range of each layer, which is represented by an interval, can be easily propagated by interval arithmetic. Benefiting from that only simple algebraic operations are involved, IBP works efficiently and thus is also used in training procedure to evaluate the robustness [15]. However, due to the loss of layer dependencies, IBP can only provide loose estimation in most cases. Two types of methods are then proposed to conquer this problem: symbolic interval propagation (SIP) and constraint programming (CP).

Different from IBP, SIP methods denote the range of a neuron as a symbolic interval, where a symbol represents a variable in the previous layers [9], [10], [16]. For instance, ERAN uses symbolic zonotopes [10], while NNV adopts symbolic image-star representation [16], [17]. Such representation can keep the dependencies of previous layers and improve the estimation precision. However, when handling nonlinear operations, symbolic intervals have to be concretized to range intervals and lose the dependencies between dimensions. Even though the refinement procedure can improve the estimation accuracy [9], [18], state-of-the art methods can only be applied to ReLU activations.

CP methods encode the neural network as a constraint system and compute the output range with constraint programming techniques. The work in [11] and [19] extends the simplex algorithm to handle ReLU constraints with satisfiability modulo theories (SMT). MILP is also widely used to model ReLu networks equivalently and thus can obtain precise output range [20], [21], [22], [23]. For instance, the work in [12] and [23] presents an equivalent MILP transformation for ReLU activation functions and computes the exact output range. Besides the lack of support for general nonlinear activation functions, such encoding may also lead to large MILP formulations and suffer from low efficiency. A number of approaches have then been proposed to compute the over-approximation with various relaxation techniques. For instance, linear programming (LP) relaxation is used in [24] and solved with duality principle, e.g., via basic dual problem [25], Lagrangian relaxation [26], and Lagrangian decomposition [27]. In [28], [29], semi-definite programming (SDP) relaxation is used, although the proposed approaches cannot be easily applied to large networks due to the complexity of current SDP solvers. The work in [30] presents the interval neural network (INN), which is a simpler neural network with fewer neurons in each layer to abstract the original network. Then, the problem of overapproximating the output range of the original neural network can be reduced to solving a mixed integer programming problem on INN. Our work is similar to this work in spirit, where abstraction is used to reduce the problem complexity. However, our technique abstracts neural networks in a layer-wise manner and also provides a mechanism to refine the abstraction iteratively.

It is worthy noting that there are other approaches that try to leverage the Lipschitz continuity of neural networks [1], [5], [31], [32]. The work in [1] shows that a large number of neural networks are Lipschitz continuous and the Lipschitz constant can help in estimating the output range, which requires solving a global optimization problem. Based on the Lipschitz continuity, the work in [5] leverages Bernstein polynomials with a bounded interval to over-approximate a neural network, which is further improved by parallel computing in [31] and used for distilling a more verification-friendly controller [32]. However, these approaches rely on a large number of sampling for estimation and thus are time consuming.

III. Problem Formulation

Notation. Throughout the paper, we use $\mathbb{R}$ to denote the set of real numbers, and $\mathbb{R}^n$ to denote the n-dimensional Euclidean space. Intervals are represented by their endpoints. For instance, the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ is denoted by $[a, b]$. An array can be multidimensional. Given an n-dimensional (n-D) real-valued array $\vec{M}$, we use $\text{sum}(\vec{M})$ to denote the sum of all elements in $\vec{M}$, $\max(\vec{M})$ for the maximum element in $\vec{M}$, and $\vec{M}[i_1]\cdots[i_n]$ to denote the element in the position of the indices $i_1, \ldots, i_n$ in the n dimensions, respectively. We also represent a section of the elements in a dimension from the index i to j ($i \leq j$) with $i : j$. Given two n-D arrays $\vec{M}_1$ and $\vec{M}_2$ that are of the same size, we use $\vec{M}_1 \odot \vec{M}_2$ to denote their element-wise product. We denote $\vec{M}_1 \sim \vec{M}_2$ for $\sim \in \{<, >, \leq, \geq, =\}$ if $\vec{M}_1, \vec{M}_2$ are of the same size and $m_1 \sim m_2$ for every elements m_1, m_2 in the same position in $\vec{M}_1, \vec{M}_2$, respectively. We use $\cdot$ for scalar multiplication.

CNN Operations. In this work, we consider Convolutional Neural Networks (CNNs) [33], which have been widely used in applications such as image analysis and natural language processing. For instance, a CNN may take a three-dimensional image with width W, height H and depth D, and produce an output to classify the input. The high-level structure of such CNNs is shown in Fig. 1. The layers in this CNN can

be categorized to three groups: the input layer that takes an input image, the hidden layers that process the image, and the output layer that outputs the result. Within the hidden layers, a convolution layer performs two sequential operations. It first convolves the input, and then processes each element in the result with an activation function. The function is usually of ReLU type, but in this work we also consider sigmoid or hyperbolic tangent functions. The details of the two operations are discussed below.

Convolution operation: A convolution operation transforms an input image to a feature map according to a finite group of filters, biases and strides. To simplify the notations, we provide a formal description in below for the operation that has only a single filter, bias and stride number. The cases of multiple numbers are straightforward extensions.

A *filter* (or *kernel*) is defined by a 2-D real-valued array W_F of the size $I_F \times J_F$. It defines the processing window for the convolution operation. A bias $b_F \in \mathbb{R}$ is a constant value that is added onto the result of every step in convolution. A *stride* can be viewed as a stepsize measured by the number of elements and is used to determine how much distance the processing window should be moved in the width or height dimension after every step. In this paper, we allow to use independent strides in the dimensions of width and height, so that the composite stride is denoted by a pair of positive integers $S_W \times S_H$.

Given an input image $\vec{X} \in \mathbb{R}^{W\times H\times D}$ with the parameters W_F, b_F and $S_x \times S_y$, the convolution result is a 2-D array $\vec{Y} \in \mathbb{R}^{((W-I_F)/S_W+1)\times((H-J_F)/S_H+1)}$, which is obtained as

$$\begin{aligned}\vec{Y}[i][j] = \text{sum}(\vec{X}[(i-1)S_W \ : \ (i-1)S_H + I_F] \\ [(j-1)S_W \ : \ (j-1)S_H + J_F] \\ [1:D] \odot W_F) + b_F.\end{aligned}$$

We may also use constraints over the arrays of variables $\vec{x}, \vec{y}$ to describe the above relation: $\vec{y} = \text{Conv}(\vec{x})$. For any real-valued arrays $\vec{X}, \vec{Y}$, satisfying the constraints $\vec{Y} = \text{Conv}(\vec{X})$ implies that $\vec{Y}$ is the result of $\vec{X}$ under convolution.

When there are multiple filters, biases and strides, the output array for all filters-stride settings are stacked in the third dimension and $\vec{Y}$ becomes a 3-D array.

Activation operation: An activation operation produces an output array $\vec{Y}$ that is of the same size as the input array $\vec{X}$, such that every element in $\vec{Y}$ is the image of the corresponding element in $\vec{X}$ under the mapping of the activation function. We consider the following activation function types:

$$\begin{array}{lrcl}\text{ReLU:} & \sigma_{\text{ReLU}}(x) & = & \max(0,x), \\ \text{Sigmoid:} & \sigma_{\text{sigmoid}}(x) & = & \frac{1}{1+e^{-x}}, \\ \text{Hyperbolic Tangent:} & \sigma_{\tanh}(x) & = & \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}},\end{array}$$

where x is the input and has a real value.

Similar to the convolution operation, we may also use constraints over the array variables $\vec{x}, \vec{y}$ to describe the input-output relation: $\vec{y} = \sigma(\vec{x})$.

Pooling layer: A pooling layer is often used to extract the dominant features of the input image. It performs a *max pooling* or *average pooling* operation. Given that the input is represented by a 3-D real-valued array $\vec{X}$ and we use a processing window of the size $I_F \times J_F$ and a stride $S_W \times S_H$, the output image $\vec{Y}$ after *max pooling* is obtained by

$$\begin{aligned}\vec{Y}[i][j][k] = \max(\vec{X}[(i-1)S_W \ : \ (i-1)S_W + I_F] \\ [(j-1)S_H \ : \ (j-1)S_H + J_F][k])\end{aligned}$$

The operation of *average pooling* can be defined similarly:

$$\begin{aligned}\vec{Y}[i][j][k] = \frac{1}{I_F\, J_F}\text{sum}\{\vec{X}[(i-1)S_W \ : \ (i-1)S_W + I_F] \\ [(j-1)S_H \ : \ (j-1)S_H + J_F][k]\}\end{aligned}$$

Both pooling operations can also be represented by constraints: $\vec{y} = \text{MaxP}(\vec{x})$, $\vec{y} = \text{AvgP}(\vec{x})$.

Flatten layer: Given an input image represented by a 3-D array $\vec{X} \in \mathbb{R}^{W\times H\times D}$, a flatten layer performs the flattening operation to transform $\vec{X}$ to a 1-D array $\vec{Y}$ such that

$$\vec{Y}[(k-1)\cdot W\cdot H + (j-1)\cdot I + i] = \vec{X}[i][j][k]$$

for all $1 \le i \le W, 1 \le j \le H, 1 \le k \le D$. Again, we may denote its constraint description by $\vec{y} = \text{Flat}(\vec{x})$.

Fully-Connected (FC) layer: An FC layer first performs an affine mapping $\vec{Y} = W_A\vec{X} + b_A$ for every input 1-D array $\vec{X}$ to generate a 1-D output array $\vec{Y}$, where W_A is a constant matrix and b_A is a constant vector of the appropriate sizes, and then performs an activation operation on $\vec{Y}$. Similar to the other operations, we may use $\vec{y} = W_A\vec{x} + b_A$ to represent the affine mapping relation.

Network output: Given an input 3-D array $\vec{X}$ for a CNN, the network output 1-D array $\vec{Y}$ is the result of consecutively applying a series of above operations. For instance, if there are n ordered operations in the network: $\text{OP}_1, \ldots, \text{OP}_n$, then Y is obtained as

$$\vec{Y} = \text{OP}_n(\text{OP}_{n-1}(\cdots \text{OP}_1(\vec{X})\cdots)).$$

An example is shown in Figure 2, in which the output array $\vec{Y}$ for an input array X is evaluated by

$$\vec{Y} = \sigma(W_A\ \text{Flat}(\ \text{MaxP}(\ \sigma(\text{Conv}(\vec{X})))) + b_A)$$

where σ is a ReLu activation.

Output Range of a CNN. Given a CNN with n operations and an interval range $\mathcal{X}$ of the input $\vec{X}$ (i.e., every element of $\mathcal{X}$ is an interval), its output range is defined by the set $\mathcal{Y} = \{\text{OP}_n(\text{OP}_{n-1}(\cdots \text{OP}_1(\vec{X})\cdots)) \,|\, \forall \vec{X}.(\underline{\mathcal{X}} \le \vec{X} \le \overline{\mathcal{X}})\}$, where $\underline{\mathcal{X}}$ is the array that has the lower bounds for the corresponding elements in $\mathcal{X}$, $\overline{\mathcal{X}}$ is the array that has the upper bounds for the corresponding elements in $\mathcal{X}$, $\underline{\mathcal{X}} \le \vec{X} \le \overline{\mathcal{X}}$ means that $\vec{X}$ is contained in the interval array $\mathcal{X}$. Hence, the problem of *output range analysis* tries to compute $\mathcal{Y}$ for a given input range $\mathcal{X}$.

However, even finding the upper and lower bounds in each dimension of $\mathcal{Y}$ is very difficult since it requires to solve a complex nonlinear programming problem on the constraint representations for all of the involved operations. For the CNN in Figure 2, finding the upper bound of the first output

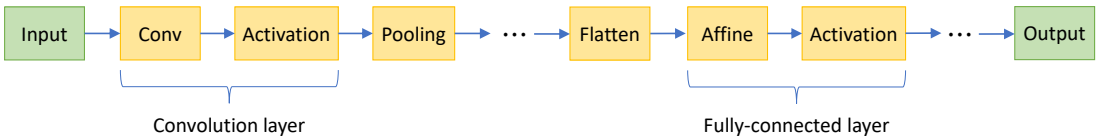


Fig. 1: CNN structure: A typical CNN example is shown. A three-dimensional input image is first processed by multiple convolution operations, each of which is followed by an activation function operation. Then, after flattened to a one-dimensional array, the intermediate result passes though multiple linear transformation layers and activation layers, and yields the output.

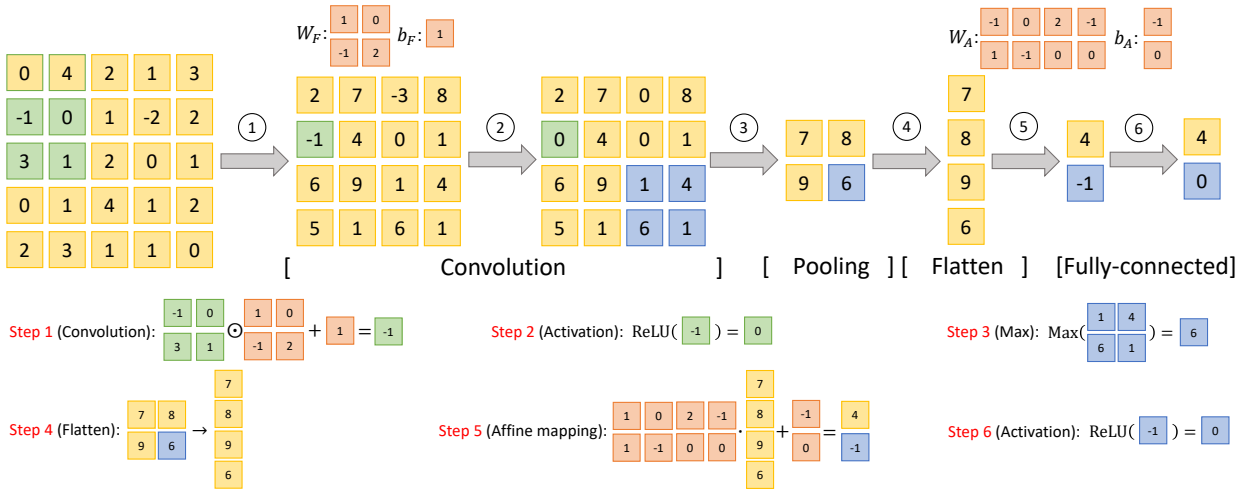


Fig. 2: A CNN example: This figure shows a concrete example of CNNs in Fig. 1. This CNN consists of four layers: a convolution layer, a pooling layer, a flatten layer and a fully-connected layer. The convolution layer contains a convolution operation and a ReLU activation, while the fully-connected layer contains an affine mapping and a ReLu activation. Steps 1-6 illustrate how these operations works on the example.

dimension over the input set $\mathcal{X}$ requires to solve the following optimization problem:

$$\max(\vec{y}[1]) \text{ s.t.}$$
$$\vec{y} = \sigma(W_A\vec{x}_4 + b_A) \ \wedge \ \vec{x}_4 = \text{Flat}(\vec{x}_3) \ \wedge \ \vec{x}_3 = \text{MaxP}(\vec{x}_2)$$
$$\wedge \ \vec{x}_2 = \sigma(\vec{x}_1) \ \wedge \ \vec{x}_1 = \text{Conv}(\vec{x}_0) \ \wedge \ \vec{x}_0 \in \mathcal{X}$$

where $\vec{y}$ has 2×1 variables, $\vec{x}_0$ has $5 \times 5 \times 1$ variables, $\vec{x}_1, \vec{x}_2$ both have $4 \times 4 \times 1$ variables, $\vec{x}_3$ has 2×2 variables, and $\vec{x}_4$ has 4×1 variables. Note that although the number of the variables is linear with respect to the input size and the number of operations in a CNN, the input image size or the CNN size is often very large in practice. Also, the activation function and the max pooling operation further make the problem nonlinear and intractable.

In this work, we seek to efficiently compute an upper or lower bound for each output dimension of a CNN, by solving an MILP relaxation of the original nonlinear problem.

IV. Relaxation and Layer-Wise Refinement

In this section, we introduce our layer-wise refinement approach for computing the output range of a CNN. Our main idea is as follows. Instead of solving the nonlinear problem

$$\max(\vec{y}[1]) \text{ s.t. } \vec{y} = \vec{x}_n \ \wedge \ \gamma_n(\vec{x}_{n-1}, \vec{x}_n) \ \wedge \\ \cdots \ \wedge \ \gamma_1(\vec{x}_0, \vec{x}_1) \ \wedge \ \vec{x}_0 \in \mathcal{X} \tag{1}$$

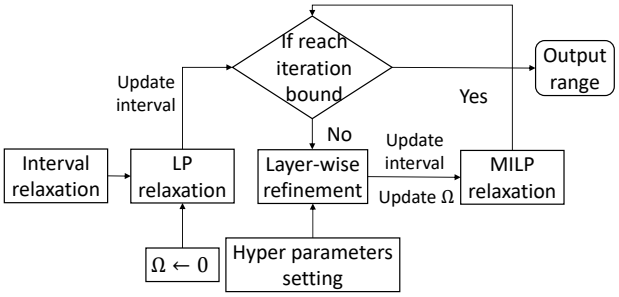


Fig. 3: Workflow of our approach.

for obtaining the upper bound of the first output dimension, we seek to solve a relaxation of it, that is

$$\max(\vec{y}[1]) \text{ s.t. } \vec{y} = \vec{x}_n \ \wedge \ \hat{\gamma}_n(\vec{x}_{n-1}, \vec{x}_n) \ \wedge \\ \cdots \ \wedge \ \hat{\gamma}_1(\vec{x}_0, \vec{x}_1) \ \wedge \ \vec{x}_0 \in \mathcal{X}, \tag{2}$$

such that γ_i is the constraint representation for the i-th operator in the CNN, and $\hat{\gamma}_i$ is a relaxation of it. Intuitively, solving the problem (2) gives us a larger value than problem (1) on the same CNN and the input set $\mathcal{X}$. The upper bounds for the other dimensions and the lower bounds are handled similarly.

The relaxed expressions $\hat{\gamma}_1, \ldots, \hat{\gamma}_n$ are updated iteratively in our framework, as shown in Figure 3, to repeatedly refine the obtained output range overapproximation.

Initially, all $\hat{\gamma}_1, \ldots, \hat{\gamma}_n$ are just interval relaxations. For all $1 \leq i \leq n$, $\hat{\gamma}_i(\vec{x}_{i-1}, \vec{x}_i)$ is represented as $\vec{x}_i \in B_i \wedge \vec{x}_{i-1} \in$

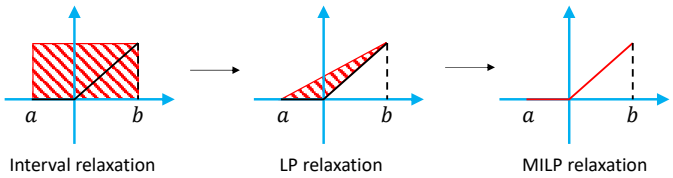


Fig. 4: Different relaxations for the input-output relation for the ReLu function. In interval relaxation, ReLU is simply relaxed as a rectangle and all the relation between x and y are dropped. In LP relaxation, a much tighter triangle is used for approximation and relatively main the input-output relation. MILP relaxation performs the equivalent transformation and thus the tightest among all the three relaxations.

B_{i-1}, where B_i and B_{i-1} are two intervals that contain the ranges of $\vec{x}_i$ and $\vec{x}_{i-1}$, respectively. If the output range estimation is too conservative, we will refine the relations to be linear. That is, $\hat{\gamma}_i(\vec{x}_{i-1}, \vec{x}_i)$ is represented as $\vec{x}_i \leq A\,\vec{x}_{i-1} + b$, which is a set of linear constraints and referred to as *linear programming (LP) relaxation*. As $\gamma_i(\vec{x}_{i-1}, \vec{x}_i) \Rightarrow \vec{x}_i \leq A\,\vec{x}_{i-1} + b$, the overapproximation property holds. If the obtained output range is still too conservative, we may further refine the constraints by introducing new integer variables to more accurately describe the operation relations, which is referred to as *mixed integer linear programming (MILP) relaxation*. That is, given a binary array $\vec{\Omega}_i$ that shares the same dimension with $\vec{y}_i$ and represents the number of integer variables for an element-wise operation, $\hat{\gamma}_i(\vec{x}_{i-1}, \vec{x}_i)$ is represented as $\vec{x}_i \leq A\,\vec{x}_{i-1} + C\,\vec{z}_{i-1} + b$, where $\vec{z}_i$ is the binary array and its dimension is determined by Ω_i. And it satisfies $\gamma_i(\vec{x}_{i-1}, \vec{x}_i) \Rightarrow \vec{x}_i \leq A\,\vec{x}_{i-1} + C\,\vec{z}_{i-1} + b$. We repeat this process until we obtain a reasonably accurate output estimation.

An example of the interval relaxation, LP relaxation and MILP relaxation for the ReLU relation $y = \max(0, x)$ is shown in Figure 4. And we find that the tightness is improved from interval relaxation to MILP relaxation.

A. Interval Relaxation

The initial interval relaxation can be obtained using the technique of interval-bound propagation (IBP) [15]. That is, starting from the interval range $\mathcal{X}$ for $\vec{x}_0$, we evaluate an interval range for $\vec{x}_1$ based on the operation mapping $\vec{x}_0$ to $\vec{x}_1$ using optimization techniques. We repeat this for all the operations. The procedure is shown in Algorithm 1. The obtained ranges build an valid interval relaxation of the original optimization problem.

Proposition 1. *Let Θ_i be the exact range of $\vec{x}_i$. The interval range I in the i-th iteration is at least an overapproximation of the exact range of $\vec{x}_i$: $\Theta_i \subseteq I$.*

It is worthy noting that a good initial solution will aid the convergence of iterative methods. Existing constraint programming (CP) based approaches do not scale well to large networks. For instance, for a ReLU FC with 10000 neurons, an MILP-encoding would contain 20000 integer variables and is thus beyond the capability of current solvers. In addition,

Algorithm 1: Construction of interval relaxation

Data: Relations for the operations $\gamma_1, \ldots, \gamma_n$, $\mathcal{X}$
Result: Interval relaxation I

$I \leftarrow (\underline{\mathcal{X}} \leq \vec{x}_0 \leq \overline{\mathcal{X}})$;
$B \leftarrow \mathcal{X}$;
for $i \leftarrow 1$ *to* n **do**
  Compute an interval range I_i for $\vec{x}_i$ based on $\vec{x}_i = \gamma_i(\vec{x}_{i-1})$ and $\vec{x}_{i-1} \in B$:
  $\underline{I}_i = \min_{\vec{x}_{i-1} \in B, \vec{x}_i = \gamma_i(\vec{x}_{i-1})} \vec{x}_i$,
  $\overline{I}_i = \max_{\vec{x}_{i-1} \in B, \vec{x}_i = \gamma_i(\vec{x}_{i-1})} \vec{x}_i$;
  $I \leftarrow I \wedge (\underline{I}_i \leq \vec{x}_i \leq \overline{I}_i)$;
  $B \leftarrow I_i$;
end
return I;

to obtain the ranges of the neurons in the intermediate layers, CP based approaches need to solve a constrained optimization problem for each neuron, which further exacerbates the computational cost. On the other hand, the symbolic interval propagation (SIP) based approaches in general can more efficiently compute the ranges for every neurons in the neural network with a single forward propagation. Among all the SIP based approaches, ERAN [10] has good performance and efficiency and has been used quite extensively [34], [35], [36]. Thus, in addition to using IBP, we also use ERAN for initialization. Specifically, we use the intersection of the ranges computed by IBP and ERAN for each neuron as the initial solution. Since both IBP and ERAN compute the relaxation of the exact range $\vec{x}$, their intersection is also a valid relaxation.

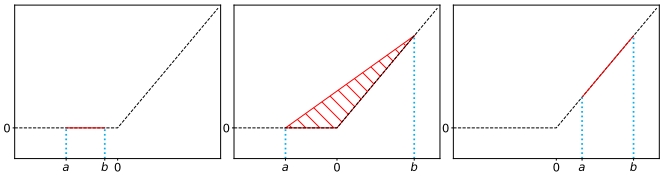
B. LP Relaxation

If the interval relaxation could not give us a reasonably tight overapproximation for the output range, we may tighten the constraints to linear forms, such that the optimization problem for finding an upper or lower bound in an output dimension becomes an LP problem. The LP relaxation will carry the dependencies from the previous layers and refine the output range from the interval relaxation. The refined range after each operation can be updated and generates tighter range for the output.

From the operation relations shown in Section III, only the max pooling and the activation functions are nonlinear. We show their linear relaxations below.

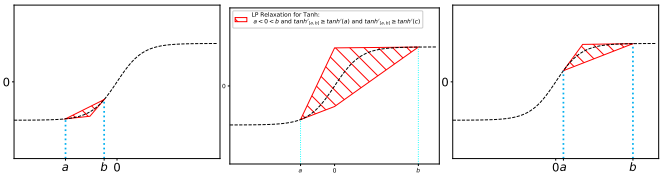
Max Pooling. Assume that the interval relaxation for a max pooling operation is given by $\underline{B}_x \leq \vec{x} \leq \overline{B}_x \wedge \underline{B}_y \leq \vec{y} \leq \overline{B}_y$, where $\vec{x}$ represents the input and y represents the output, the linear relaxation of the max pooling relation can be obtained as $\Phi^{\text{LP}}_{\text{MaxP}}(\vec{x}, \vec{y}) = \bigwedge_{i,j,k}(h(\vec{x}, \vec{y})[i][j][k] \leq 0)$, where

$$h(\vec{x}, \vec{y})[i][j][k] \leq 0 : (\underline{B}_x \leq \vec{x} \leq \overline{B}_x) \wedge (\underline{B}_y \leq \vec{y} \leq \overline{B}_y) \wedge \bigwedge_{\substack{1 \leq i' \leq I_F \\ 1 \leq j' \leq J_F}} (\vec{y}[i][j][k] \geq \vec{x}[(i-1)S_W + i', (j-1)S_H + j', k]). \tag{3}$$



(a) (b) (c)

Fig. 5: LP relaxation for ReLU: Figure 5a shows the LP relaxation (red area, as well as Figure 5b and Figure 5c) when $b \leq 0$. Figure 5b shows the LP relaxation when $a<0<b \wedge \sigma'_{[a,b]}>\sigma'_+(a) \wedge \sigma'_{[a,b]}\leq\sigma'_-(b)$. Figure 5c shows the relaxation when $a \geq 0$.



(a) (b) (c)

Fig. 6: LP relaxation for tanh: Figure 6a shows the LP relaxation (red area, as well as Figure 6b and Figure 6c) when $b \leq 0$. Figure 6b shows the LP relaxation when $a<0<b \wedge \sigma'_{[a,b]} > \sigma'_+(a) \wedge \sigma'_{[a,b]} > \sigma'_-(b)$. Figure 6c shows the relaxation when $a \geq 0$.

Lemma 1 (Validity of LP Relaxation for MaxP)**.** *Assume that the interval relaxation for a max pooling operation is given by $\underline{B}_x \leq \vec{x} \leq \overline{B}_x \wedge \underline{B}_y \leq \vec{y} \leq \overline{B}_y$, where $\vec{x}$ represents the input and y represents the output. Let Θ be the exact range of $\vec{x}$ and $h(\vec{x},\vec{y})[i][j][k]$ be defined as Equation* (3)*. We have:*

$$\{(\vec{x},\vec{y}) \,|\, \vec{y} = \mathit{MaxP}(\vec{x}) \wedge \vec{x} \in \Theta\} \;\subseteq\; \{(\vec{x},\vec{y}) \,|\, \Phi^{LP}_{\mathit{MaxP}}(\vec{x},\vec{y})\}.$$

Activation Function. We consider three types of activation functions: ReLu, sigmoid, and tanh, which are all nonlinear.

First, given an interval $[a,b]$, the LP relaxation of the activation function differs with respect to the value of a and b. Let $\sigma'_{[a,b]} = (\sigma(b)-\sigma(a))/(b-a)$ be the slope between the points $(a,\sigma(a))$ and $(b,\sigma(b))$.

If $b \leq 0$, notice that ReLU, sigmoid and tanh are all convex over the interval $[a,b]$. Thus, we can leverage the left/right derivative to derive the LP relaxation:

$$\begin{aligned} g_{[a,b]}(x,y) \leq 0 : \quad & (-y + \sigma'_+(a)(x-a) + \sigma(a) \leq 0) \wedge \\ & (-y + \sigma'_-(b)(x-b) + \sigma(b) \leq 0) \wedge \\ & (y - \sigma'_{[a,b]}(x-a) - \sigma(a) \leq 0). \end{aligned} \tag{4}$$

If $a \geq 0$, notice that ReLU, sigmoid and tanh are all concave over the interval $[a,b]$. Similar to the case $b \leq 0$, we have:

$$\begin{aligned} g_{[a,b]}(x,y) \leq 0 : \quad & (y - \sigma'_+(a)(x-a) - \sigma(a) \leq 0) \wedge \\ & (y - \sigma'_-(b)(x-b) - \sigma(b) \leq 0) \wedge \\ & (-y + \sigma'_{[a,b]}(x-a) + \sigma(a) \leq 0). \end{aligned} \tag{5}$$

If $a < 0 < b$, let $C_a = (0, y_a)$ be the intersection between the y-axis and the tangent line of σ at a, where $y_a = \sigma(a) - a\sigma'_+(a)$. Let $C_b = (0, y_b)$ be the intersection between the y-axis and the tangent line of σ at b, where $y_b = \sigma(b) - b\sigma'_-(b)$. To make sure the relaxation is convex, our LP relaxation differs with respect to the value of $\sigma'_{[a,b]}$ as follows.

$$g_{[a,b]}(x,y) \leq 0 :$$
$$\begin{cases} \begin{cases} -y + \sigma'_{[a,b]}(x-a) + \sigma(a) \leq 0 & \sigma'_{[a,b]} \leq \sigma'_+(a) \\ \begin{cases} -y + \sigma'_+(a)(x-a) + \sigma(a) \leq 0, \\ -y + \frac{\sigma(b)-y_a}{a}x + \sigma(a) + y_a \leq 0, \end{cases} & \sigma'_{[a,b]} > \sigma'_+(a) \end{cases} \\ \begin{cases} -y + \sigma'_{[a,b]}(x-a) + \sigma(a) \geq 0 & \sigma'_{[a,b]} \leq \sigma'_-(b) \\ \begin{cases} -y + \sigma'_-(b)(x-b) + \sigma(b) \geq 0, \\ -y + \frac{\sigma(a)-y_b}{b}x + \sigma(a) + y_b \geq 0, \end{cases} & \sigma'_{[a,b]} > \sigma'_-(b) \end{cases} \end{cases} \tag{6}$$

Lemma 2 (Validity of LP Relaxation for Activation Function)**.** *Assume that the interval relaxation for an activation function σ is given by $a \leq x \leq b \wedge \sigma(a) \leq \sigma(b) \leq u_y$, where x represents the input, y represents the output, and σ can be ReLu, sigmoid or tanh. Let $g_{[a,b]}(x,y) \leq 0$ be the LP relaxation and Θ be the exact range of x. We have:*

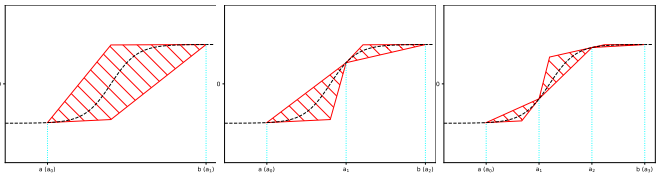
$$\{(x,y) \,|\, y = \sigma(x) \wedge x \in \Theta\} \;\subseteq\; \{(x,y) \,|\, g_{[a,b]}(x,y) \leq 0\}.$$

C. MILP Relaxation

Due to the high nonlinearity of the deep neural networks, only using LP relaxation may be hard to capture the true mapping and provide tight enough output range for safety verification. Thus, based on the construction of the LP relaxation, we further tighten the relaxation by introducing integer variables to capture the nonlinear mappings in deep neural networks like activation and pooling. We form this relaxation as an MILP problem. With more integer variables added, the relaxation approaches the exact mapping gradually. We utilize this property to refine the range after every operation by adding integer variables and update the range to tighten the relaxation. We show the convergence result at the end of this section. Following are the detailed MILP encoding methods for nonlinear operations in deep neural networks.

Max Pooling. Assume that the interval relaxation for a max pooling operation is given by $\underline{B}_x \leq \vec{x} \leq \overline{B}_x \wedge \underline{B}_y \leq \vec{y} \leq \overline{B}_y$, where $\vec{x}$ represents the input and $\vec{y}$ represents the output. We can use the classical Big-M method [37] to construct the MILP transformation, which can be also found in [23]. Specifically, given the number of integer variables $\omega \geq I_F \cdot J_F$, the MILP relaxation of the max pooling relation can be obtained as $\Phi^{\text{MILP}}_{\text{MaxP}}(\vec{x},\vec{y},\omega) = \bigwedge_{i,j,k}(h^\omega(\vec{x},\vec{y})[i][j][k] \leq 0)$, where

$$h^\omega(\vec{x},\vec{y})[i][j][k] \leq 0 :$$
$$\begin{cases} \underline{B}_x \leq \vec{x} \leq \overline{B}_x, \quad \underline{B}_y \leq \vec{y} \leq \overline{B}_y, \\ \vec{y}[i][j][k] \geq \vec{x}[(i-1)S_W + i', (j-1)S_H + j', k], \\ \qquad 1 \leq i' \leq I_F, \quad 1 \leq j' \leq J_F, \\ \vec{y}[i][j][k] \leq M(1 - z[i'][j'][k]) + \\ \qquad\qquad \vec{x}[(i-1)S_W + i', (j-1)S_H + j', k], \\ \qquad 1 \leq i' \leq I_F, \quad 1 \leq j' \leq J_F, \\ \sum_{1 \leq i \leq I_F, 1 \leq j \leq J_F} z[i,j,r] = 1, \\ z \text{ is a binary matrix.} \end{cases} \tag{7}$$



(a) $\omega = 1$ (LP) (b) $\omega = 2$ (c) $\omega = 3$

Fig. 7: MILP relaxation for tanh over $[a, b]$ by Equation (8): Figure 7a shows the MILP relaxation (red area, as well as Figure 7b and Figure 7c) with one integer variable (i.e., LP relaxation). Figure 7b and Figure 7c show the cases for two and three integer variables, respectively.

Lemma 3 (Equivalence of MILP Relaxation for MaxP)**.** *Assume that the interval relaxation for a max pooling operation is given by* $\underline{B}_x \le \vec{x} \le \overline{B}_x \wedge \underline{B}_y \le y \le \overline{B}_y$*, where* $\vec{x}$ *represents the input and* $\vec{y}$ *represents the output. Let* Θ *be the exact range of* $\vec{x}$ *and* $h(\vec{x}, \vec{y})[i][j][k]$ *be defined as Equation* (3)*. We have:*

$$\{(\vec{x}, \vec{y}) \,|\, y = \mathit{MaxP}(\vec{x}) \wedge \vec{x} \in \Theta\} = \{(\vec{x}, \vec{y}) \,|\, \Phi^{\mathit{MILP}}_{\mathit{MaxP}}(\vec{x}, \vec{y}, \omega)\}.$$

Activation Function. Let the number of binary variables of the activation function be $\omega \ge 1$, we can use the following MILP relaxation.

$$g_{[a,b]}(x, y, \omega) \le 0 : \bigwedge_{1 \le i \le \omega} (g_{[a_i, a_{i+1}]}(x, y) \le M(1 - z[i])) \wedge (a_i = a + i\frac{b-a}{\omega}) \wedge \sum_{i=0}^{\omega} z[i] = 1. \tag{8}$$

In particular, for ReLU activation function, we can change the setting of a_i by letting a certain $a_i = 0$. Then, Equation (8) is a equivalent conversion. Note that similar MILP formulation for ReLU is also found in [12] and can be considered as a special case of our relaxation.

Lemma 4 (Validity of MILP Relaxation for Activation Function)**.** *Assume that the interval relaxation for an activation function* σ *is given by* $a \le x \le b \wedge \sigma(a) \le \sigma(b) \le u_y$*, where* x *represents the input,* y *represents the output, and* σ *can be ReLu, sigmoid or tanh. Let* Θ *be the exact range of* x*, and* $g_{[a,b]}(x, y, \omega) \le 0$ *be the MILP relaxation. We have:*

$$\{(x, y) \,|\, y = \delta(x) \wedge x \in \Theta\} \subseteq \{(x, y) \,|\, g_{[a,b]}(x, y, \omega) \le 0\}.$$

When the number of slack integer variables increases, finer polygons are used for approximation. As a result, the area of the polygon becomes smaller and in the limit will converge to the actual nonlinear activation function. An example of MILP relaxation for tanh over $[a, b]$ with different number of integer variables can be found in Figure 7. Intuitively, given any point $x_0 \in [a, b]$ and $\omega \ge 1$, we know that $x_0 \in [a_i, a_{i+1}]$, where $i = \lfloor \frac{(x_0 - a)\omega}{b-a} \rfloor$. Let $\mathcal{P}_i$ be the polygon determined by $g_{[a_i, a_{i+1}]}(x, y) \le 0$. Then we know the relaxation error $\epsilon = \max_{(x_0, y) \in \mathcal{P}_i} |y - \delta(x_0)|$ is smaller than $\delta(a_{i+1}) - \delta(a_i)$. Thus when $\omega \to \infty$, $\delta(a_{i+1}) - \delta(a_i) \to 0$, the relaxation error converges to zero.

Lemma 5 (Convergence of MILP Relaxation for Activation Function)**.** *Assume that the interval relaxation for an activation function* σ *is given by* $a \le x \le b \wedge \sigma(a) \le \sigma(b) \le u_y$*, where* x *represents the input,* y *represents the output, and* σ *can be ReLu, sigmoid or tanh. Let* Θ *be the exact range of* x*, and* $g_{[a,b]}(x, y, \omega) \le 0$ *be the MILP relaxation. We have:*

$$\{(x, y) \,|\, y = \delta(x) \wedge x \in \Theta\} \to \{(x, y) \,|\, g_{[a,b]}(x, y, \omega) \le 0\},$$

when $\omega \to \infty$.

Now given the number of slack binary variables for all the variables ω, we can define the corresponding MILP problem as follows.

$$\max(\vec{y}[1]) \text{ s.t. } \vec{y} = \vec{x}_n \ \wedge \ \hat{\gamma}_n(\vec{x}_{n-1}, \vec{x}_n) \ \wedge \ \cdots \ \wedge \ \hat{\gamma}_1(\vec{x}_0, \vec{x}_1) \ \wedge \ \vec{x}_0 \in \mathcal{X}, \tag{9}$$

where $\hat{\gamma}_i$ is defined as

$$\hat{\gamma}_i = \begin{cases} \Phi^{\text{MILP}}_{\text{MaxP}}(\vec{x}_{i-1}, \vec{x}_i, \Omega_i), & \text{Max pooling,} \\ g(\vec{x}_{i-1}, \vec{x}_i, \Omega_i) \le 0, & \text{Activation function,} \\ \gamma_i, & \text{Otherwise.} \end{cases}$$

Combined with Lemma 3 and Lemma 4, we can derive the relation between $y_{\text{MILP}}(\Omega)$ and y_{NLP}.

Theorem 1 (Soundness)**.** *Given a neural network with the input domain* $\mathcal{X}$*, let* y_{NLP} *and* $y_{MILP}(\Omega)$ *be the two optimal values for the optimization problems in* (1) *and* (9)*, respectively, we have:*

$$y_{\mathit{MILP}}(\Omega) \ge y_{\mathit{NLP}}. \tag{10}$$

Benefiting from Lemma 3 and Lemma 5, the convergence of the optimal value can be guaranteed.

Theorem 2 (Convergence)**.** *Given a neural network with the input domain* $\mathcal{X}$*, let* y_{NLP} *and* $y_{MILP}(\Omega)$ *be the two optimal values for the optimization problems in* (1) *and* (9)*, respectively, we have:*

$$\lim_{\Omega \to \infty} y_{\mathit{MILP}}(\Omega) = y_{\mathit{NLP}} \tag{11}$$

V. Propagation by Sliding Window

While the aforementioned MILP relaxation shows theoretical guarantees on convergence, solving the optimization problem in (9) could still be challenging due to two issues. The first is the number of integer variables. Note that the range computed based on IBP in the initialization phase can be rather loose and a large number of integer variables may be needed to obtain a good approximation for the MILP in (9), which greatly increases the complexity. The second issue is the overall scale of the programming problem. Even if we reduce Ω to 1, which means the MILP in (9) degenerates to a LP, the polynomial complexity of LP is still a challenging issue for large neural networks.

Thus, instead of the "global" coding scheme in (9), we adopt a "local" MILP formulation in this section to alleviate the computation burden brought by the two challenges above. To control the number of integer variables, we refine the variable range in the hidden layers before computing the range of the output layers. With a smaller range, fewer integer variable are

Algorithm 2: Propagation by sliding window

Data: Relations for the operations $\gamma_1, \ldots, \gamma_n$, input space $\mathcal{X}$, interval relaxation I, iterations J, refinement percentage p, sliding window s
Result: Upper bound of the output $y[1]$

```
// Initialize the number of integer variables
1  Ω ← 1
// Propagate intervals by sliding window
2  for j ← 1 to J do
3      for i ← 1 to n do
4          refineList ← Strategy(I, Ω_i, p);
5          for v ∈ refineList do
6              Ω_i[v] ← Ω_i[v] + 1;
7              I_i[v] ← UpdateRange(γ_i : γ_{i−s},
                 Ω_i : Ω_{i−s}, I, v);
8          end
9      end
10 end
11 return Ī_N[1];
```

needed to achieve the similar approximation precision for a variable. To control the scale of the programming problem for refining a variable, we only encode the constraints in the previous limited operations rather than all the previous operations. In addition, we only refine part of the variables in each operation to further reduce the number of the integer variables considered in each programming problem.

Our overall algorithm is shown in Algorithm 2. We first initialize the number of binary slack variables for all the variables (line 1). Then we iteratively refine selected variables operation-by-operation for J iterations (lines 2-10). Specifically, a heuristic approach is proposed to rank the importance of the variables in each operation, and select the top ranking variables with the number determined by the given percentage parameter p (line 4). For each selected variable that needs to be refined, we first increase the binary slack variables by 1 (line 6), and then use a sliding window based approach to only encode partial constraints of the network (line 7). Finally, we obtain the upper bound of $y[1]$ (line 16). The key issues are how to refine the variable range by sliding window and how to choose the most important variables to be refined.

We first show how to refine the variable range. Specifically, consider the variable $\vec{x}_i[v]$, following Equation 1, we can use the following NLP to compute its exact upper bound:

$$\max(\vec{x}_i[v]) \text{ s.t. } \gamma_i(\vec{x}_{i-1}, \vec{x}_i) \wedge \cdots \wedge \gamma_1(\vec{x}_0, \vec{x}_1) \wedge \vec{x}_0 \in \mathcal{X}. \tag{12}$$

Let $\hat{\gamma}_i$ be defined as

$$\hat{\gamma}_i = \begin{cases} \Phi^{\text{MILP}}_{\text{MaxP}}(\vec{x}_{i-1}, \vec{x}_i, \Omega_i), & \text{Max pooling,} \\ g(\vec{x}_{i-1}, \vec{x}_i, \Omega_i) \leq 0, & \text{Activation function,} \\ \gamma_i, & \text{Otherwise.} \end{cases}$$

Given the length of the sliding window of layers that consist of s operations, the number of slack binary variables Ω, and the interval relaxation I, we know that

$$(\hat{\gamma}_{i-s}(\vec{x}_{i-s-1}, \vec{x}_{i-s}) \wedge \cdots \wedge \ \hat{\gamma}_1(\vec{x}_0, \vec{x}_1) \wedge \ \vec{x}_0 \in \mathcal{X}) \subseteq I_{i-s}.$$

Following Equation (9), we can use the following MILP to compute an upper bound of $\vec{x}_i[v]$.

$$\max(\vec{x}_i[v]) \text{ s.t. } \hat{\gamma}_n(\vec{x}_{i-1}, \vec{x}_i) \wedge \cdots \wedge \hat{\gamma}_n(\vec{x}_{i-s+1}, \vec{x}_{i-s}) \wedge I_{i-s}. \tag{13}$$

Proposition 2. *Given a neural network with the input domain $\mathcal{X}$, let x_{NLP} and $x_{SMILP}(\Omega)$ be the two optimal values for the optimization problems in* (12) *and* (13)*, respectively, we have:*

$$x_{SMILP}(\Omega) \geq x_{NLP}. \tag{14}$$

Now we introduce how to choose variables by heuristics for refinement. In this paper we only perform refinement for activation functions, as no relaxation is needed for linear operations and max pooling (which can be equivalently transformed into linear constraints). Note that our heuristic approach to choose neurons to be refined for activation functions could be extended to max pooling operation, but requires a more sophisticated manner. We will leave it as future work.

Specifically, we define the "importance" of a variable $x_i[v]$ by the following heuristic ranking function:

$$\text{rank}(x_i[v]) = \begin{cases} (b_i[v] - a_i[v])/\Omega_i[v] & \text{Conv. layer} \\ (b_i[v] - a_i[v])/\Omega_i[v] \cdot W_{i+1}[v] & \text{FC layer,} \end{cases} \tag{15}$$

where $a_i[v]$ and $b_i[v]$ denote the two ends of the range, $\Omega_i[v]$ denotes the number of slack integer variables for the variable, $W_{i+1}[v]$ denotes the weight in the next linear transformation. Intuitively, $(b_i[v] - a_i[v])/\Omega_i[v]$ describes the granularity of the partition and $W_{i+1}[v]$ describes the impact of the variable on the following operations. Given a hyper-parameter p, we always pick the top $p\%$ variables in each operation with respect to the value of the ranking function to perform the refinement.

Remark 1. *Note that for each variable, the length of its range is finite and is not larger than the one given by the initial interval relaxation. Thus each variable has the chance to be refined for enough iterations. For instance, consider an activation function in a convolution layer with m variables and their ranges are defined as $[a_1, b_1], \cdots, [a_m, b_m]$, let $[a', b']$ be the smallest range. Then we can see that after $J \geq \sum_{i=1}^{m} \lceil (b_i - a_i)/(b' - a') \rceil$, every variable is refined for at least once.*

Remark 2. *If the length of the sliding window s is large enough, that is, MILP in* (13) *encodes all the constraints in the previous operations for each variable, convergence is still guaranteed since each variable will be refined when the iteration J is big enough (as explained in Remark 1). However, convergence is no longer maintained when $s < n$. It is due to the loss of dependencies of the operations that the constraints are not encoded. In the experiments, we can find such treatment will not greatly influence the output range precision in practice.*

VI. Experiments

We implement our approach in a tool called *LayR* (stands for Layerwise Refinement). We evaluate the output range improvement of *LayR* over the range provided in initial estimation

TABLE I: Neural network setting and experimental results: We do the experiments on data set: MNIST and CIFAR. For each neural network #, $Type$ denotes the neural network type, $Neurons$ denotes the total number of neurons of the deep neural networks, $Layers$ denotes the number of Fully-connected layers (FC) and convolutional layers (Conv), $Activation$ denotes the type of activation functions used in the network. We pick the first four test images in each dataset and for each image, we construct an input set as a box with the center on each image. The length of the edge is defined as $2\times$ perturbation: For MNIST, the perturbation is $\epsilon = 0.01$, and for CIFAR-10, the perturbation is $\epsilon = 0.001$. The initialization $Time$ shows the computation time for the picked initialization method. Under our approach, p is the percentage of the refined neurons out of total number of neurons in each layer. We also denote the number of refined neurons beneath the percentage number. s is the traceback layer number. J is the number of iterations chosen. We show the result as the range improvement after refinement after first iteration, it-1 Range Improvement, and the final refined range, Final Range Improvement. The runtime of the refinement process is shown under the columns of it-1 Time and Total iteration Time in seconds.

Data set	Neural network					Input set	Our approach							
	#	Type	Neurons	Layers	Activation		Initialization [13] Time (s)	p %	s	J	it-1 Improvement	it-1 Time (s)	Iteration Improvement	Iteration Time (s)
MNIST	I	FNN	500	FC: 5	sigmoid	MNIST-1	67	FC: 20 (200 neurons)	3	2	43.12%	380	54.53%	1001
						MNIST-2	16				37.91%	350	37.92%	909
						MNIST-3	43				17.37%	383	25.54%	933
						MNIST-4	23				33.92%	411	37.67%	1034
	II	FNN	700	FC: 7	sigmoid	MNIST-1	223	FC: 20 (280 neurons)			43.83%	452	57.27%	977
						MNIST-2	30				19.44%	577	22.07%	1626
						MNIST-3	135				5.60%	488	10.48%	1139
						MNIST-4	18				37.02%	585	37.02%	2645
	III	ConvSmall	14966	Conv: 2 FC: 3	ReLU + sigmoid	MNIST-1	62	Conv: 0.05 FC: 20 (264 neurons)		4	29.92%	684	29.92/%	3016
						MNIST-2	62				28.23%	699	28.23%	3062
						MNIST-3	64				31.39%	696	31.39%	3049
						MNIST-4	64				31.84%	693	31.84%	3049
	IV	ConvMed	20962	Conv: 4 FC: 3	ReLU + sigmoid	MNIST-1	2090	Conv: 0.05 FC: 20 (276 neurons)			36.58%	1835	60.17%	12459
						MNIST-2	1854				31.61%	2238	52.95%	10717
						MNIST-3	956				33.99%	1854	55.53%	12877
						MNIST-4	1021				37.21%	1861	59.55%	12913
	V	ConvBig	21262	Conv: 4 FC: 6	ReLU + sigmoid	MNIST-1	280	Conv: 0.05 FC: 20 (516 neurons			37.49%	1606	56.95%	15443
						MNIST-2	742				45.80%	1594	64.52%	15363
						MNIST-3	917				52.31%	1635	79.54%	14984
						MNIST-4	521				48.98%	1668	69.04%	18573
CIFAR	VI	ConvMed	32082	Conv: 4 FC: 3	ReLU + sigmoid	CIFAR-1	442	Conv: 0.05 FC: 20 (296 neurons)	3	4	20.85%	3145	29.87%	12864
						CIFAR-2	511				19.73%	4347	29.18%	15101
						CIFAR-3	510				19.73%	4322	29.18%	14793
						CIFAR-4	563				20.72%	4187	31.66%	14636
	VII	ConvSuper	87394	Conv: 4 FC: 6	ReLU + sigmoid	CIFAR-1	1647	Conv: 0.05 FC: 20 (644 neurons)	2	4	1.34%	2702	4.12%	9490
						CIFAR-2	1675				1.17%	2948	2.29%	10545
						CIFAR-3	1817				1.17%	3007	2.29%	9990
						CIFAR-4	5218				1.36%	2854	2.44%	6830

and compare the width of output range with NNV [16], [17]. Most of the output range analysis tools for neural network is compatible with our approach and can provide initial estimation for our approach. Here, we choose ERAN [13], [10], [18] as the initialization method due to its generality on supporting different activation functions and its efficiency.

Evaluation datasets. We use the popular image datasets MNIST and CIFAR-10 in our experiment. MNIST contains gray-scale images of size 28×28 pixels, whereas CIFAR-10 contains RGB images of 32×32 with 3 channels.

Neural networks. TABLE I shows 7 different MNIST and CIFAR-10 feedforward networks (FNNs) and convolutional networks (CNNs) with heterogeneous activations in our experiment. We train all networks with cross-entropy loss, which is often used in classification tasks [2]. The largest networks in our experiments contains $> 87K$ neurons whereas the deepest network contains 10 layers.

Machine configuration. All experiments were ran on a 3.6Hz 12 core Intel(R) Core(TM) i7-6850K CPU with 128 GB of main memory. MILP problems are solved using Gurobi [38].

A. Effectiveness of Layer-wise Refinement

We focus on the range of the dimension of ground-truth logit (dimension corresponding to the true class) since it contains the most important information in classification tasks. Experiment results are shown in Table I. After the first iteration, *LayR* can already achieve a significant improvement over the initial estimation, and in most cases, such improvement of range volume continuously grows after four iterations. We show that bridging the propagation-based methods with programming-based method in a dividing and sliding manner is efficient and can bring tighter range than the pure propagation-based method. An interesting phenomenon is that the first iteration provides more significant improvement than the following iterations. One reason is that the most important neurons are selected and refined in the first iteration. It is also worthy not-

TABLE II: Comparison with NNV [16]. We compare NNV and our method by showing the width of the estimated range on the ground-truth logit obtained from both methods. The computation time is shown in seconds for both methods.

#[1]	Input set	NNV		*LayR*	
		Range	Time (s)	Range	Time (s)
I	MNIST-1	12.44	6	2.85	1068
	MNIST-2	12.78	7	1.52	925
	MNIST-3	30.36	7	22.10	976
	MNIST-4	12.64	7	2.41	1057
II	MNIST-1	10.50	11	2.24	1200
	MNIST-2	12.43	11	4.96	1656
	MNIST-3	28.44	12	25.44	1274
	MNIST-4	14.13	11	0.75	2663
III	MNIST-1	7.74	3456	2.29	3078
	MNIST-2	6.72	1782	3.07	3124
	MNIST-3	6.26	5954	2.05	3113
	MNIST-4	4.61	1404	2.38	3113

[1] On the remaining settings including MNIST IV-V and CIFAR VI-VII, NNV exceeded a timeout limit of 24 hours while the longest running time of our tool among these benchmarks was around 5 hours on the same machine. Thus we do not have the range comparison for those cases here.

ing that for the largest CNN considered here, i.e., ConvSuper for CIFAR data set, the improvement is relatively minor. The reason is that in order to control scale of the programming for the large network, we only select two traceback layers to refine (fewer than the other networks). The impact of the hyper parameters, including the iterations and refinement percentage, as well as the length of sliding window will be elaborated later.

We can also observe that *LayR* costs more time than the symbolic-propagation based approach ERAN (used as initialization) in some cases. The large number of neurons that are refined in each iteration brings this computation overhead. Especially when processing the final iteration, refining a neuron may need to solve an MILP with hundreds of integer variables. Even though the number of integer variables is much smaller than directly encoding the whole neural network as Equation (9), it still needs some time for computation with the current optimization techniques.

B. Comparison with NNV

We also compare *LayR* with the neural network verification tool NNV [16] on the same set of benchmarks. Note that most of other tools are limited to neural networks with a specific type of activation functions (e.g., [23] can only handle ReLU networks) and/or do not provide capability for dealing with convolutional layers. The comparison results with NNV are shown in Table II. We only include the results for MNIST I, II and III, since NNV did not terminate for the other networks (MNIST IV-V and CIFAR VI-VII) with a timeout of 24 hours, while the longest running time of our method is around 5 hours on the same machine.

Experimental results across the benchmarks show 10.55% (Network II, NNIST-3) to 94.69% (Network II, NNIST-4) improvement on output range estimation by our approach against NNV. In [16], NNV considers a special kind of pixel brightening attacks, where the brightness of a small number of pixels can be independently varied. In our experiments, we consider L_∞-norm-bounded perturbations, which is a prevalent model in adversarial attack literature [15], [10]. We refer to the variables that are used to express the input interval range $\mathcal{X}$ under this perturbation model as perturbation variables. Under L_∞-norm-bounded perturbations, we need the same number of perturbation variables as the number of image pixels to describe $\mathcal{X}$, which is significantly more than the number of perturbation variables considered in the brightening attack in [16]. We speculate that the image-star representation used in [16] may result in a more conservative estimate when the number of such variables is large, which explains the observed difference in output range estimation as shown in Table II.

In addition, when the neural networks become larger, the efficiency of NNV degrades quickly – NNV finished network # I and # II in seconds, # III in around an hour, but did not terminate for any of the other larger networks within 24 hours. In our case, the proposed "divide and slide" mechanism allows us to limit the size of each optimization problem and effectively cope with larger networks.

VII. Discussion

A. Effectiveness of LP Refinement

As aforementioned, the MILP-based layer-wise refinement (we use MILP refinement for short in this section) solves the optimization with a large number of integer variables when refining a neuron range, which is the efficiency bottleneck for our approach. It thus motivates the question of how effective an LP refinement would be, i.e., applying Algorithm 2 without line 6. Below, we compare LP refinement with MILP refinement while keeping the rest of our algorithm intact.

The experimental results on the MNIST dataset are shown in Figure 8. Without introducing integer variable, the complexity of LP refinement is significantly lower than MILP refinement. On the other hand, while LP refinement works well on shallow networks (ConvSmall), its performance degrades (compared to MILP refinement) when the neural networks become deeper (ConvMed and ConvBig). This is because LP relaxation captures less inter-layer dependencies than the MILP relaxation.

B. Impact of Hyper-parameters

In *LayR*, there are three hyper-parameters that need to be determined in advance: the percentage of neurons to be refined in each layer p, the length of the sliding window / traceback number s, and the iteration number J. In theory, each parameter would improve the performance when increasing. However, increasing the value of these parameters also makes our approach more time consuming. In this section, we demonstrate the trade-off between the precision and computation time empirically. In each set of experiments, we tune one parameter and fix the other two. The results under different settings of hyper-parameters on output range analysis of CNNs trained on MNIST dataset can be found in Figure 9.

Figure 9a, Figure 9b, and Figure 9c show the trade-off with respect to the iteration number, the length of sliding window, and neuron selection percentage of each layer, respectively. In terms of the output range precision, the output range is

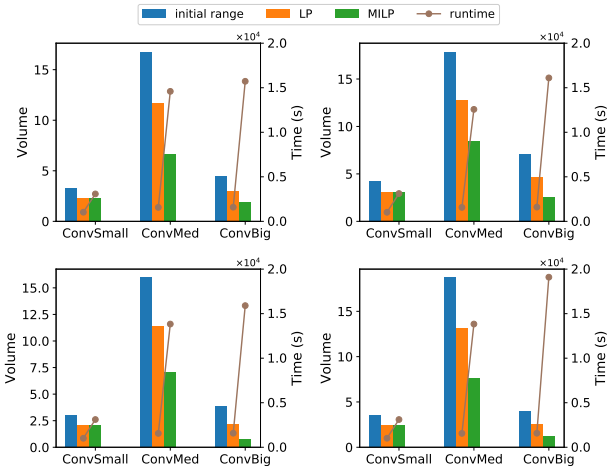


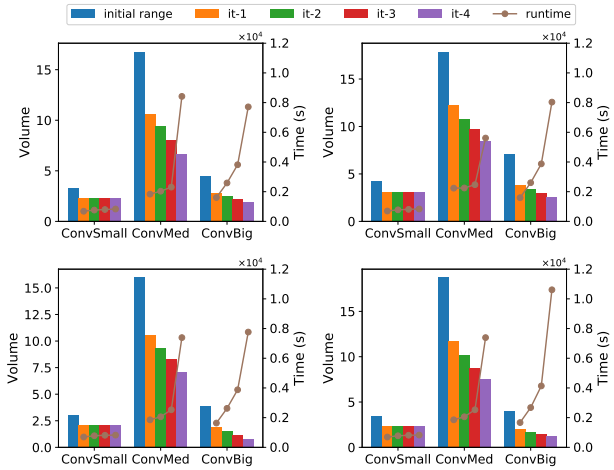
Fig. 8: Output volume and runtime comparison between LP refinement and MILP refinement on MNIST.

getting tighter with a larger hyper-parameter in most cases. We can also observe that the computation time grows largely with larger values of each hyper-parameter. Such phenomenon conforms to our expectations, since the number of integer variables grows linearly with each hyper-parameter increasing. It is worthy noting that for ConvSmall on MNSIT, each iteration costs similar time (Figure 9a). We speculate that this is due to that most integer variables are removed by the pre-solve mechanism of the MILP solver in Gurobi, which implies that our heuristic neuron selection algorithm does not pick the most important neurons.

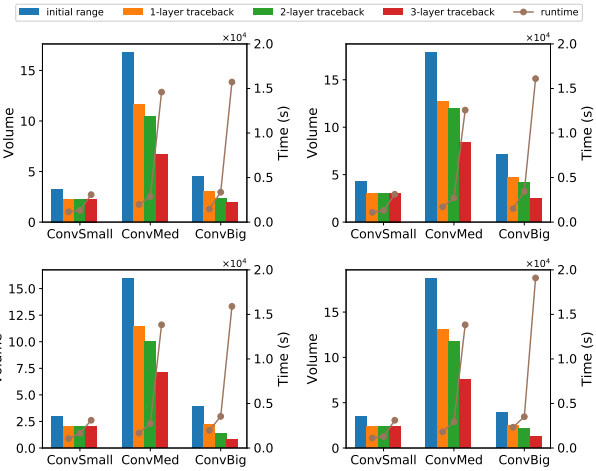
Although the large values of these hyper-parameters lead to tighter relaxation and better refinement, we can also observe that if we only use the programming-based method, the computation overhead will be extremely large and hard to solve. However, accompanying the propagation-based idea using the sliding window and iteratively solving the problem significantly reduce the size of the programming. The results also show that our approach generate much tighter range than the pure propagation-based approach. Even when the convergence guarantee is missing here, empirically the propagation by sliding window and iterative refinement gain great benefits in terms of computation and precision.

C. Limitations

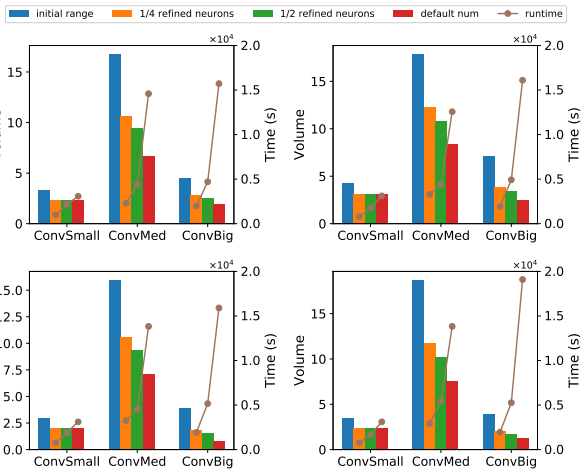
Observing the experimental results shown in the above sections, we can see that the main weakness of our approach is the efficiency. Though more iterations, larger refinement percentage and length of sliding window can help improve the performance, the aforementioned results show that we can hardly enlarge those hyper-parameters too much. The fundamental problem behind is that it would introduce too many integer variables, which is an important factor on the complexity of MILP. A potential improvement is to come up with finer variable selection strategy. One direction is to group and characterize the variables based on their impact on the output. Better the variables' range can be refined, fewer



(a) Output volume comparison for different iterations: The blue bar denotes the volume of the initial range. The orange/green/red/purple bar denotes the refined range after the first/second/third/fourth iteration, respectively.



(b) Output range volume comparison with different numbers of the length of the sliding window or traceback layers: The blue bar denotes the volume of the initial range. The orange/green/red bar denotes the refined range with the traceback layers equaling the one/two/three, respectively.



(c) Output range volume comparison with different numbers of refined neurons: The blue bar denotes the volume of the initial range. The orange/green/red bar denotes the refined range while refining $\frac{1}{4}/\frac{1}{2}/1$ of the neurons comparing to the original setting in Table I.

Fig. 9: Impact of each hyper-parameter on the precision and computation time for CNNs trained on MNIST dataset.

integer variables are needed. We will explore this direction in our future work.

VIII. Conclusion

In this paper, we propose an iterative method for the output range analysis of deep neural networks. The approach is based on the proposed convex polygonal relaxation for nonlinearity in networks, which enables MILP with the capability to tune the tightness of the relaxation by introducing more integer variables. In the initialization phase, we compute the primary range by IBP. In each iteration, our approach iteratively identifies neurons to refine the relaxation. To better manage the growth of the number of integer variables as the refinement progresses, when refining a variable, we encode only partial constraints by tracebacking a few previous layers, rather than all the layers. We show the overall framework is sound and provides a valid over-approximation. Our future work includes exploring other tools for initialization and better heuristics to identify the important neurons for refinement.

Acknowledgment

We would like to thank Hoang-Dung Tran and Taylor T. Johnson (Vanderbilt University, Tennessee) for their help with running NNV on our benchmarks for comparison. We would also like to thank Gagandeep Singh (ETH Zurich, Switzerland) for sharing and explaining their code in ERAN, which helps us integrate ERAN in our tool for initialization.

We gratefully acknowledge the support from NSF grants 1834701, 1834324, 1839511, 1724341, ONR grant N00014-19-1-2496, and the US Air Force Research Laboratory (AFRL) under contract number FA8650-16-C-2642. This work is also funded in part by the DARPA BRASS program under agreement number FA8750-16-C-0043 and NSF grant 1646497.

References

[1] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees." International Joint Conferences on Artificial Intelligence, 2018.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[4] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in neural information processing systems*, 2016, Conference Proceedings, pp. 2613–2621.

[5] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, "Reachnn: Reachability analysis of neural-network controlled systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–22, 2019.

[6] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Hybrid Systems: Computation and Control (HSCC)*. ACM Press, 2019, pp. 157–168.

[7] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 169–178.

[8] W. Xiang and T. T. Johnson, "Reachability analysis and safety verification for neural network control systems," *arXiv preprint arXiv:1805.09944*, 2018.

[9] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *USENIX Security Symposium*, 2018, pp. 1599–1614.

[10] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 802–10 813.

[11] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[12] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[13] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.

[14] W. Xiang, H.-D. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with relu activations," *arXiv preprint arXiv:1712.08163*, 2017.

[15] H. Zhang, P. Zhang, and C.-J. Hsieh, "Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications," in *AAAI Conference on Artificial Intelligence (AAAI)*, dec 2019.

[16] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," *International conference on Computer-Aided Verification*, 2020.

[17] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[18] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[19] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[20] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, Conference Proceedings, pp. 251–268.

[21] M. Fischetti and J. Jo, "Deep neural networks as 0-1 mixed integer linear programs: A feasibility study," *arXiv preprint arXiv:1712.06174*, 2017.

[22] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *arXiv preprint arXiv:1706.07351*, 2017.

[23] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *International Conference on Learning Representations*, 2019.

[24] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," ser. Automated Technology for Verification and Analysis. Springer International Publishing, 2017, Conference Proceedings, pp. 269–286.

[25] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*, 2018, pp. 5286–5295.

[26] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks." in *UAI*, vol. 1, 2018, p. 2.

[27] R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. H. Torr, and M. P. Kumar, "Lagrangian decomposition for neural network verification," *arXiv preprint arXiv:2002.10410*, 2020.

[28] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 877–10 887.

[29] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *arXiv preprint arXiv:1903.01287*, 2019.

[30] P. Prabhakar and Z. R. Afzal, "Abstraction based output range analysis for neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 762–15 772.

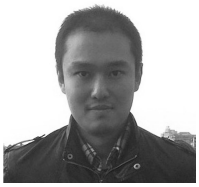
[31] J. Fan, C. Huang, W. Li, X. Chen, and Q. Zhu, "Reachnn*: A tool for reachability analysis ofneural-network controlled systems," in *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2020.

[32] ——, "Towards verification-aware knowledge distillation for neural-network controlled systems," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[33] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.
[34] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev, "Certifying geometric robustness of neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 287–15 297.
[35] X. Huang and L. Zhang, "Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification," in *Static Analysis: 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings*, vol. 11822. Springer, 2019, p. 296.
[36] M. Balunovic and M. Vechev, "Adversarial training and provable defenses: Bridging the gap," in *International Conference on Learning Representations*, 2020.
[37] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[38] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

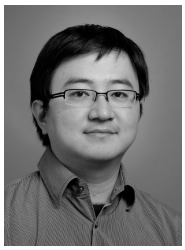
Chao Huang received the Ph.D. degree in computer science from Nanjing University, Nanjing, China, in 2018. He is currently a postdoc fellow with the ECE department, Northwestern University, Evanston, USA. His current research interests include verification and design towards safety and security for cyber physical systems, including but not limit to learning-enabled systems.

Jiameng Fan is a Ph.D. candidate in electrical engineering with Department of Electrical and Computer Engineering at Boston University, Boston, MA, USA. His research interests lie in the intersection of Machine Learning (ML), Formal Methods (Verification and Synthesis) and Robotics.



Xin Chen received his Doctor rerum naturalium (Doctor of natural sciences) from RWTH Aachen University, Germany in 2015. He is currently an assistant professor of Computer Science at the University of Dayton, Dayton, OH, USA. His research interests mainly focus on solving the safety and security problems for the dynamical systems equipped with AI controllers using numerical and formal methods.

Wenchao Li received his Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 2013. He is currently an Assistant Professor of Electrical and Computer Engineering at Boston University, Boston, MA, USA. His research interests lie broadly in the area of dependable computing, with a recent focus at the intersection of formal methods and machine learning, and with applications to cyber-physical systems, design automation, and A.I. safety.



Qi Zhu received the Ph.D. degree in electrical engineering and computer science from the University of California at Berkeley, Berkeley, CA, USA in 2008. He is currently an Associate Professor of Electrical and Computer Engineering at Northwestern University, Evanston, IL, USA. His research interests include design automation for cyber-physical systems (CPS) and Internet of Things, cyber-physical security, safe and secure machine learning for CPS, and system-on-chip design, with applications in domains such as automotive electronic systems, connected vehicles, and energy-efficient buildings.