This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081



# Output Range Analysis for Feed-forward Deep Neural Networks via Linear Programming

Zhiwu Xu, Yazheng Liu, Shengchao Qin, Zhong Ming

**Abstract**—The success of deep neural networks and their potential use in many safety-critical applications has motivated research on formal verification of deep neural networks. A fundamental primitive enabling the formal analysis of neural networks is the output range analysis. Existing approaches on output range analysis either focus on some simple activation functions like $relu$ or compute a relaxed result for some activation functions like $elu$. This paper proposes an approach to compute the output range for feed-forward deep neural networks via linear programming. The key idea is to encode the activation functions, such as $elu$ and $sigmoid$, as linear constraints in term of the line between the left and right end-points of the input range and the tangent lines on some special points in the input range. A strategy to partition the network to get a tighter range is presented. The experimental results show that our approach gets a tighter result than RobustVerifier on $elu$ networks and $sigmoid$ networks. Moreover, our approach performs better than (the linear encodings implemented in) Crown on $elu$ networks with $\alpha = 0.5, 1.0$ and $sigmoid$ networks, and better than CNN-Cert and DeepCert on $elu$ networks with $\alpha = 0.5$ or $1.0$. For $elu$ networks with $\alpha = 2.0$, our approach can achieve results that are closed to Crown, CNN-Cert and DeepCert. Finally, we also found that the network partition helps to achieve a tighter result as well as to improve the efficiency for $elu$ networks.

**Index Terms**—Output Range Analysis, Deep Neural Networks, Linear Programming, ELU, Sigmoid.

## 1 Introduction

Deep neural networks (DNNs) have emerged as a promising approach for creating scalable and robust systems. It can solve the classification problems that are difficult or even impossible to solve in a relatively short time, and has many applications in academic and commercial fields, such as computer vision [1], speech recognition [2], natural language processing [3], malware detection [4], and even in safety-critical fields, such as autonomous vehicles [5], network security [6], surveillance [7], drones and robotics [8].

Unfortunately, Szegedy et al. [9] discovered that the robustness of neural networks encounters a major challenge when adding imperceptible non-random perturbation to input. Such perturbed examples are called as *adversarial examples*. This phenomenon implies that deep neural networks are vulnerable to adversarial examples, which limits its applications and could cause huge economic losses and personal safety, especially in safety-critical areas. For example, in a security-critical system like ACAS Xu [10], an incorrectly handled corner case can easily be exploited by an attacker to cause significant damage costing thousands of lives.

Hence, there is an urgent need to formally verify properties of DNNs, which is an NP-complete problem [11], even for the simplest type feed-forward DNNs. Generally, a property can be formulated as a statement that if the input belongs to some set $\mathcal{I}$, then the output will belong to or be disjoint to some set $\mathcal{O}$ [12]. Many approaches [13], [14], [15], [16], [17] to approximately compute the output range (*i.e.*, the reachable set) of DNNs have been proposed. However, most of these approaches focus on DNNs with $relu$ activation function and may not suitable for the ones with non-$relu$ activation functions, such as $elu$ and $sigmoid$[1]. There are some approaches [18], [19], [20] taking some non-$relu$ activation functions into account, but the output ranges obtained by these approaches could be either too relaxed or difficult to get a sound implementation. For example, MaxSens [19] can be applied on any activation functions, but it performs as poorly as the simple approach via interval arithmetic (*i.e.*, the naive one taking respectively the lower and upper bounds for the negative and positive weights). Crown [21], a framework for general activation functions, computes the output range via the tangent lines. Although sound theoretically, the tangent lines are obtained by a binary search such that it may yield an unsound result in some cases.

In this paper, we propose a simple and sound output range analysis for feed-forward deep neural networks with non-$relu$ activation functions, in particular $elu$ and $sigmoid$ activation functions. In detail, we firstly encode the activation functions into linear constraints, based on the line between the left and right end-points of the input range and the tangent lines on some special points in the input range. All these lines are sound with respect to the output range and can be implemented easily. Then with these encodings, we compute an output range via linear programming. And we also present some strategies to get a tighter range. A novel one is to partition the network into several sub-networks so as to make a trade-off on dependences and

• *Zhiwu Xu, Yazheng Liu, Shengchao Qin and Zhong Ming are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China. Shengchao Qin and Zhong Ming are the corresponding authors. Email: xuzhiwu@szu.edu.cn, 719811493@qq.com, shengchao.qin@gmail.com, and mingz@szu.edu.cn.*

1. It may be easy to approximate some non-$relu$ activation functions using a combination of $relu$ units, which is either inefficient or relaxed.

accumulation errors between layers.

We conduct a series of experiments to evaluate our approach. First, we perform experiments on different networks with $elu$ or $sigmoid$ that are trained on MNIST dataset. The experimental results show that our approach computes a tighter output range on both $elu$ networks and $sigmoid$ networks than RobustVerifier [20] does. In detail, in our experiments on $elu$ networks, about $93.07\%$ of our results are included in RobustVerifier's, while only one time that RobustVerifier's result is a subset of ours. For experiments on $sigmoid$ networks, these two percentages are $71.09\%$ and $0.00\%$, respectively. Compared to Crown [21], our approach performs better on $elu$ networks with $\alpha = 0.5$ or $1.0$ and $sigmoid$ networks, but a little worse on $elu$ networks with $\alpha = 2.0$. Specifically, in our experiments on $elu$ networks with $\alpha = 0.5$ or $1.0$, all our results are included in Crown's, while the opposite percentage is only $0.38\%$. And for $sigmoid$ networks, $68.24\%$ of our results are included in Crown's, while $22.03\%$ of Crown's results are included in ours. But, on $elu$ networks with $\alpha = 2.0$, only $2.83\%$ of our results are included in Crown's but $81.46\%$ of Crown's results are subsets of ours. But our average radii of the output ranges are quite close to Crown's. Compared to CNN-Cert [22] and DeepCert [23], our approach performs better on $elu$ networks with $\alpha = 0.5$ or $1.0$, but a little worse on $elu$ networks with $\alpha = 2.0$ and $sigmoid$ networks. In detail, for $elu$ networks with $\alpha = 0.5$ or $1.0$, $95.38\%$ ($84.30\%$ resp.) of our results are included in CNN-Cert's (DeepCert's resp.), while $0.98\%$ of CNN-Cert's ($1.40\%$ of DeepCert's resp.) are included in ours. These two percentages are $1.21\%$ and $94.94\%$ ($0.77\%$ and $91.5\%$ for DeepCert resp.) for $elu$ networks with $\alpha = 2.0$, and are $20.37\%$ and $43.35\%$ for $sigmoid$ networks ($1.03\%$ and $29.60\%$ for DeepCert resp.). Nevertheless, our results are quite close to both *CNN-Cert* and *DeepCert*. For $elu$ networks with $\alpha = 2.0$, the averages of all the average radii for our approach, *CNN-Cert* and *DeepCert* are $38818.23$, $37912.91$ and $37929.39$, respectively.

Finally, we also perform experiments to evaluate the effectiveness of the network partition and found that the network partition helps to achieve a tighter result as well as to improve the efficiency for $elu$ networks. For example, the network partition on span 2 or 3 for $elu$ networks gets the minimum results, with fewer runtimes than those needed by the whole networks.

The remainder of this paper is organized as follows. Section 2 introduces some preliminaries. Section 3 describes our approach, followed by the experimental results in Section 4. Section 5 presents the related work, followed by some concluding remarks in Section 6.

## 2 Preliminaries

In this section, we briefly introduce some preliminaries.

### 2.1 Feed-forward deep neural networks

A Feed-Forward Deep Neural Network (FFDNN), also known as multilayer perceptron (MLP), is an artificial neural network in which the connections between nodes does not form a cycle. A feed-forward deep neural network is comprised of an input layer, an output layer, and multiple hidden layers in between. A layer is comprised of multiple neurons (so-called nodes), each of which is connected to neurons of the next layer. And there is no connection between neurons in the same layer. Furthermore, the output of each neuron in the hidden layer is assigned by a linear combination of the neuron outputs of the previous layer, and then applying a non-linear activation function. By assigning values to the input layer and then feeding them forward through the network, values for each layer can be computed from the values of the previous layer, finally resulting in values for the output layer. Formally, FFDNN is defined as follows:

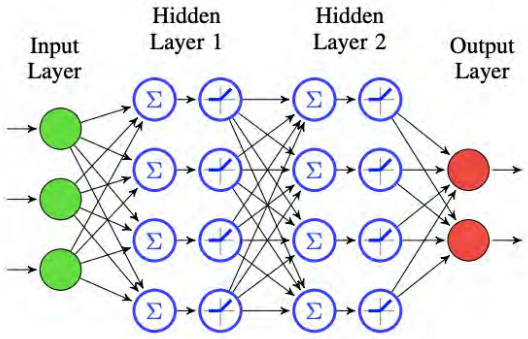



Fig. 1. A network with 2 hidden layers

***Definition 1.*** A feed-forward deep neural network $\mathcal{N}$ is a list of layers $[L^0, L^1, \ldots, L^n, L^{n+1}]$, where

- $L^0$ is the input layer, $L^{n+1}$ is the output layer, and $L^1, \ldots, L^n$ are the hidden layers.
- $L^0 = x^0$, where $x^0$ is the input vector.
- for each non-input layer $L^i$, $L^i = (W^i, b^i, \phi^i, x^i)$, where $W^i \in \mathcal{R}^{s_i \times s_{i-1}}$ is the weight matrix by which the neurons in $L^i$ are connected to neurons from the preceding layer $L^{i-1}$, $b^i$ is the bias vector that is used to assigned bias values to the neurons in $L^i$, $\phi^i$ is the activation function for all the neurons in $L^i$ , $x^i$ is the vector corresponding to the values of the neurons in the layer $L^i$, and $s_i$ is the number of neurons in $L^i$.
- for each non-input layer $L^i$, the value vector $x^i$ for the neurons in $L^i$ are determined by the value vector of $x^{i-1}$ with the weight matrix $W^i$, the bias vector $b^i$ and the activation function $\phi^i$, that is, $x^i = \phi^i(W^i x^{i-1} + b^i)$ with the activation function $\phi^i$ being applied element-wise.

Figure 1 shows a FFDNN with 2 hidden layers, wherein the numbers of neurons for each layer are respectively $3, 4, 4, 2$, the activation functions for the hidden layers are $relu$, and the functions for the output layer are identity.

Given a function $f : \mathcal{R} \to \mathcal{R}$ and an input range $[l, u]$ with $l \leq u$, a function $f'$ is said to be a *lower* (resp. *upper*) bound of $f$ with respect to $[l, u]$ if $f'(x) \leq f(x)$ (resp. $f'(x) \geq f(x)$) for each input $x \in [l, u]$. A lower (resp. *upper*) bound $f_1$ of $f$ is said to be *tighter* than another one $f_2$ if $f_2(x) \leq f_1(x) \leq f(x)$ (resp. $f_2(x) \geq f_1(x) \geq f(x)$) for each input $x \in [l, u]$. Moreover, a constant lower bound $L$ and a constant upper one $U$ of function $f$ form a *sound* output range $[L, U]$ of $f$, that is, $f(x) \in [L, U]$ for each input

$x \in [l, u]$. In terms of FFDNN, a set $\mathcal{O}$ is a *sound* output range (*i.e.*, reachable set) for a FFDNN $\mathcal{N}$ with respect to an input range $\mathcal{I}$ if $\{\mathcal{N}(in) \mid in \in \mathcal{I}\} \subseteq \mathcal{O}$, and a sound range $\mathcal{O}$ is *tighter* than another one $\mathcal{O}'$ if $\{\mathcal{N}(in) \mid in \in \mathcal{I}\} \subseteq \mathcal{O} \subseteq \mathcal{O}'$ [24], [25], [26].

Given a FFDNN $\mathcal{N}$ and an input range $\mathcal{I}$, the output range analysis is to compute an output range $\mathcal{O}$ that is sound and as tight as possible. A simple solution to this problem [14] is to compute $\mathcal{O}$ via the interval arithmetic, which is given as follows:

$$\begin{array}{ll} \hat{l}_j^i & = \min_{x^{i-1} \in [l^{i-1}, u^{i-1}]} W_j^i \times x^{i-1} + b_j^i \\ & = [W_j^i]^+ \times l^{i-1} + [W_j^i]^- \times u^{i-1} + b_j^i \\ \hat{u}_j^i & = \max_{x^{i-1} \in [l^{i-1}, u^{i-1}]} W_j^i \times x^{i-1} + b_j^i \\ & = [W_j^i]^+ \times u^{i-1} + [W_j^i]^- \times l^{i-1} + b_j^i \\ l_j^i & = \phi^i(\hat{l}_j^i) \\ u_j^i & = \phi^i(\hat{u}_j^i) \end{array} \tag{1}$$

where $l^i$ and $u^i$ are respectively the lower and upper bounds for the layer $L^i$ after activation (*i.e.*, $x^i$), $l_j^i$ and $u_j^i$ are respectively the $j$-th element of $l^i$ and $u^i$, that is, the lower and upper bounds on the value of the $j$-th neuron in the layer $L^i$ after activation, $\hat{l}_j^i$ and $\hat{u}_j^i$ respectively denote the before-activation bounds for the $j$-th neuron in the layer $L^i$, and $[a]^+$ and $[a]^-$ respectively denote the positive and negative parts (*i.e.*, $\max(a, 0)$ and $\min(a, 0)$) of a scalar variable $a$ and can be element-wisely extended to vectors or matrixes.

### 2.2 Linear programming

Linear programming (LP) is the mathematical theory and method for studying the extremum of linear objective function problem under the condition of linear constraints. The constraints may be equalities or inequalities. The fundamental form of linear programming is:

$$\begin{array}{l} \max\ c^T x \\ \text{s.t. } A \times x \leq b \end{array} \tag{2}$$

where $A \in \mathcal{R}^{m \times n}$, $b \in \mathcal{R}^m$, $c \in \mathcal{R}^n$ and $x = [x_1, \ldots, x_n] \in \mathcal{R}^n$. Any LP problem can be converted to an LP problem in the form of (2) having the same feasible region and optimal solution set. Solving the problem (2) is equivalent to finding among all the solutions to the constraints (*i.e.*, $A \times x \leq b$) the one that maximizes the objective (*i.e.*, $c^T x$).

## 3 Approach

Given a neural network $\mathcal{N}$ and an input range $\mathcal{I}$, the output range analysis aims to compute an output range $\mathcal{O}$ such that $\mathcal{O}$ includes the ground ranges $\{\mathcal{N}(in) \mid in \in \mathcal{I}\}$ and could be as tight as possible. The key difficulty lies in the non-linear activation functions. Due to that $relu$ and some of its variants, such as $leakyReLU$, are piecewise linear functions, most existing approaches focus on $relu$ and compute their possible output ranges via the corresponding linear segments and/or the straight line connecting the left and right end-points (*i.e.*, the lower and the upper bounds) of the function on the input range (denoted as the lower-upper line) [17], [27], [28]. But these approaches cannot handle the non-linear segments or other activation functions, such as the other variants of $relu$ like $elu$. RobustVerifier [20] encodes the activation function $sigmoid$ as the constraints in terms of the 1-order Taylor expansion (*i.e.*, the tangent line) on the midpoint and the possible ranges of the error functions. However, the output range computed by this encoding would be too relaxed, as moving the tangent line upward or downward could make it no longer a tangent line. Crown uses the tangent lines through the left or right end-points, but they are not easy to obtain in practice. Indeed, finding a tangent line of a function through a point is equivalent to solving a system of non-linear equations, depending on the function. For a system involving the activation functions like $sigmoid$ or $elu$, it may not be solved exactly nor have general solutions, which makes the problem difficult. The algorithms solving them usually are iterative and approximated, such as Crown and CNN-Cert, yielding an unsound or relaxed result. Motivated by the above approaches, our idea is to make full use of the lower-upper line and the tangent lines on some special points so as to get a tighter output range, especially for the non-linear activation functions $elu$ and $sigmoid$.

In the following, we first give the linear encodings for the activation functions $elu$ and $sigmoid$, respectively. Then we present our algorithm for output range analysis, and finally discuss some strategies to reduce the errors.

### 3.1 Linear encoding for $elu$

Exponential Linear Unit ($elu$) [29] is a function that tends to converge cost to zero faster and produce more accurate results. Different to other activation functions, $elu$ has an extra constant $\alpha$ which should be a positive number. The function $elu$ is defined as

$$elu(x) = \begin{cases} x & x > 0 \\ \alpha * (e^x - 1) & x \leq 0 \end{cases} \tag{3}$$

Figure 2a shows different $elu$ functions with different $\alpha$s.

Consider an $elu$ function $y = elu(x)$ with the input range $x \in [l, u]$, where $l < u$. To get a tight output range, we perform a case analysis on the shape of the function with respect to the input range: $(i)$ $0 \leq l < u$, $(ii)$ $l < u \leq 0$; $(iii)$ $l < 0 < u$.

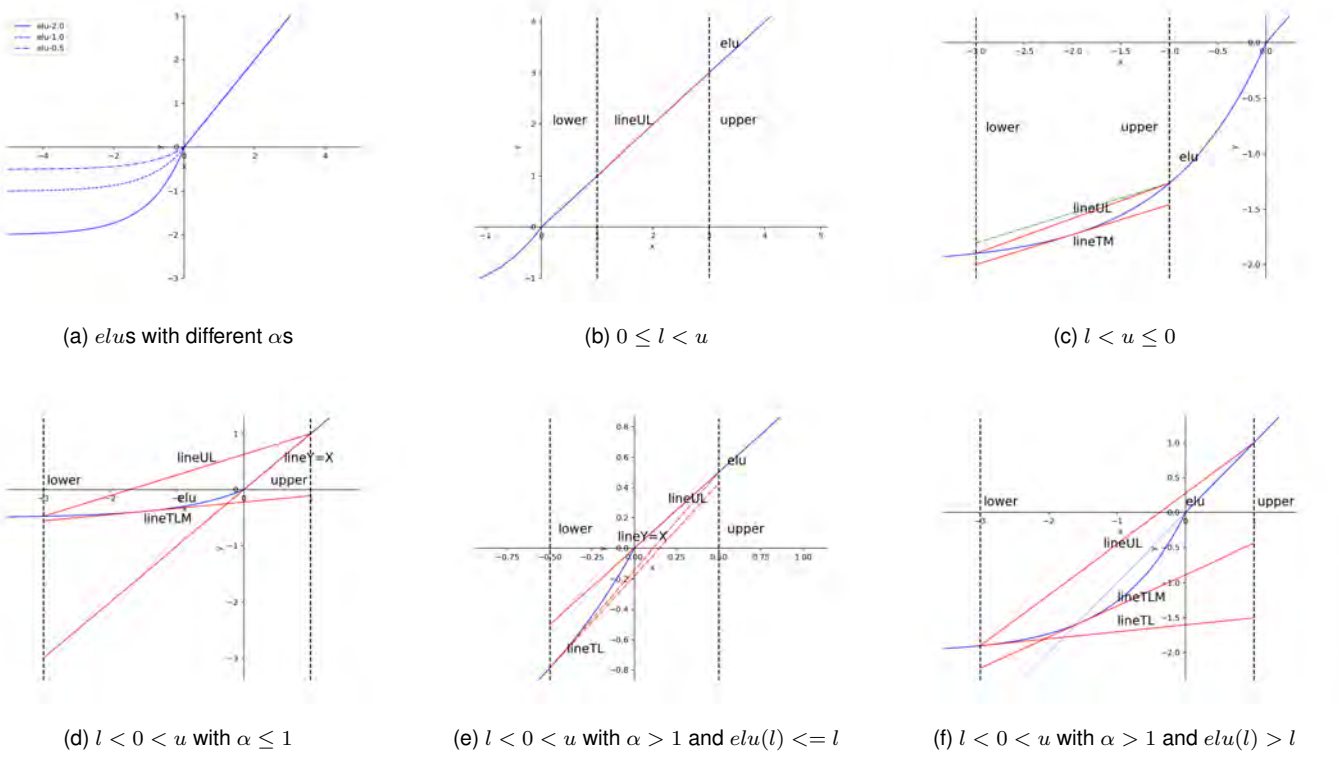
**Case** $0 \leq l < u$. From the definition, when all the input $x$ in the range $[l, u]$ is positive, $elu$ acts the same as the $relu$. Therefore, similar to most existing approaches on $relu$, we can use the linear segment of $elu$ directly, which is shown as the red line segment in Figure 2b. That is to say, the output range in this case is encoded as the following constraints

$$\begin{array}{l} y = x, \\ l \leq x \leq u. \end{array} \tag{4}$$

**Case** $l < u \leq 0$. When all the input $x$ in the range $[l, u]$ is negative, different from $relu$, $elu$ is no longer linear, not the exponential operation $e^x$. Nevertheless, like existing approaches on $relu$, we can still use the lower-upper line (*i.e.*, the red line lineUL in Figure 2c) as the upper bound. But we still need to find a linear encoding for the lower bound. For that, following RobustVerifier [20], we adopt the tangent line on the midpoint of the range $[l, u]$ (*i.e.*, the red

This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081






(a) $elu$s with different $\alpha$s (b) $0 \leq l < u$ (c) $l < u \leq 0$

(d) $l < 0 < u$ with $\alpha \leq 1$ (e) $l < 0 < u$ with $\alpha > 1$ and $elu(l) <= l$ (f) $l < 0 < u$ with $\alpha > 1$ and $elu(l) > l$

Fig. 2. Linear encodings for $elu$

line lineTM in Figure 2c). Formally, the possible range of $y$ is encoded as the following linear constraints:

$$\begin{aligned} & y \leq \frac{elu(u) - elu(l)}{u - l} * x + \frac{u * elu(l) - l * elu(u)}{u - l}, \\ & y \geq elu'(\frac{l+u}{2}) * (x - \frac{l+u}{2}) + elu(\frac{l+u}{2}) \\ & l \leq x \leq u. \end{aligned} \tag{5}$$

where $elu'$ is the derivative of $elu$. Let us apply RobustVerifier's idea on this case. Then it would move the tangent line on the midpoint upward to find an upper bound, which depends on the range of the error function of Taylor expansion. As shown in Figure 2c, the tightest encoding for the upper bound is the green line (passing through the right end-point), which is still larger than ours.

**Case** $l < 0 < u$. This case is a little more complicated than the above two cases, as the input range contains both positive and negative values, that is, the output range involves the linear segment and the non-linear segment. To make this kind of situation easier, we further split it into three independent subcases: (i) $\alpha \leq 1$, (ii) $\alpha > 1$ and $elu(l) <= l$, and $\alpha > 1$ and $elu(l) > l$.

**Subcase** $\alpha \leq 1$. In this subcase, all the negative output is above (the negative extension of) the linear segment of $elu$ (*i.e.*, the tangent line $y = x$ on the right end-point in Figure 2d), as all the derivatives on the negative inputs are not larger than $\alpha$. Similar to existing approaches on ReLU, we would like to adopt the "triangle relaxation"[2], except that we need to find a linear encoding for the lower bound. As shown in Figure 2d, we can take the tangent line on $l/2$ (*i.e.*, the midpoint of the negative inputs). So, the linear constraints for this subcase are

$$\begin{aligned} & y \leq \frac{elu(u) - elu(l)}{u - l} * x + \frac{u * elu(l) - l * elu(u)}{u - l}, \\ & y \geq x \\ & y \geq elu'(\frac{l}{2}) * (x - \frac{l}{2}) + elu(\frac{l}{2}) \\ & l \leq x \leq u. \end{aligned} \tag{6}$$

Figure 3 shows the linear encodings for Case $l < 0 < u$ with $\alpha \leq 1$ that may obtained by different tools. Crown [21] takes the tangent line at a point in the positive (negative resp.) set that passes the left (right resp.) end-point as the upper (lower resp.) bounds. Specifically, Crown performs a limited-iteration binary search to look for a point in the positive (negative resp.) set such that the slope of the line between the point and the left (right resp.) end-point is as close as possible to the one of the tangent line of the point. The ideal lower and upper bounds in this case is the line $y = x$ and the lower-upper line (the red ones in Figure 3), respectively. However, both bounds are always not the tangent lines on the correspond points, (even with respect to a tolerated error 0.01), so a limited-iteration binary search may yield an unsound result (see the green points in Figure 3). To avoid unsound results as possible, Crown moves the result line up (down resp.) with respect to the tolerated error. This may still be unsound or yield a relaxed result (see the green dashed lines in Figure 3), especially for activation functions with unbounded output results. Indeed, for this case, the required tangent lines do not exist. CNN-Cert [22] uses the same idea as Crown but employs a different implementation. CNN-Cert returns the upper

2. Indeed, it is a quadrilateral.

(lower resp.) point of the search range in the last iteration for the upper (lower resp.) bound (see the blue points in Figure 3). Moreover, CNN-Cert compares the slopes of the result line and the tangent line on the result point, and takes the smaller one to ensure soundness (see the blue dotted lines in Figure 3). Note that the upper bound in this case is exactly the lower-upper line. DeepCert [23] improves CNN-Cert by moving the lower-upper line down (up resp.) if all the outputs are below (above resp.) it, which can be done by a similar binary search used in CNN-Cert (see the purple dot-and-dash lines in Figure 3). As the best point for this case is always smaller than 0, the point returned by DeepCert would be smaller than the one obtained by CNN-Cert. Although none of the lower bounds is tighter than any others, the area formed by our bounds is minimum, which probably leads to a tight result.

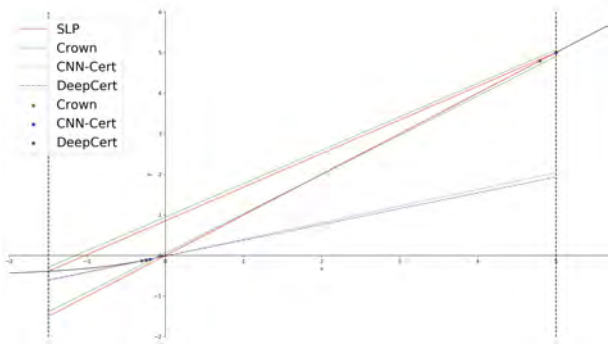



Fig. 3. Constraints for $elu$ in $l < 0 < u$ with $\alpha \leq 1$ by different tools.

**Subcase** $elu(l) <= l$. When $\alpha > 1$, not all the negative outputs are above (the negative extension of) the linear segment of $elu$. However, if $elu(l) <= l$, as shown in Figure 2e, then all the negative outputs are below the linear segment instead, which could be taken as an upper bound. The remain is to find a linear lower bound. A possible tight solution is the tangent line on a negative input $x_0$ such that it passes through the right end-point. But it requires to solve the equation $elu'(x_0) * (u - x_0) + elu(x_0) = elu(u)$ to find the input $x_0$, which involves the exponential operation and is not easy to get the exact solutions. For simplicity and soundness, we take either the upper-lower line or the tangent line on the left end-point (the dashdot lines in Figure 2e), depending on their slopes: if the slope of the upper-lower line is larger than the one of the left end-point tangent line, then the upper-lower line is not a lower bound, due to that there are some outputs that are below it. In short, the linear constraints for this subcase are

$$\begin{array}{l} y \leq x \\ \text{if } elu'(l) < \dfrac{elu(u) - elu(l)}{u - l},\ y \geq elu'(l) * (x - l) + elu(l) \\ \text{else } y \geq \dfrac{elu(u) - elu(l)}{u - l} * x + \dfrac{u * elu(l) - l * elu(u)}{u - l} \\ l \leq x \leq u. \end{array} \tag{7}$$

**Subcase** $elu(l) > l$. Different from the two above subcases, some negative outputs are above (the negative extension of) the linear segment of $elu$ as well as some are not. So the linear segment is no longer suitable for encoding. But from Figure 2f, we found the upper-lower line is clearly an upper bound. Moreover, we take as the lower bounds both the tangent line on $l/2$ (*i.e.*, the midpoint of the negative inputs)[3] and the tangent line on the left end-point, which are shown in Figure 2f. Based on the above discussion, we have

$$\begin{array}{l} y \leq \dfrac{elu(u) - elu(l)}{u - l} * x + \dfrac{u * elu(l) - l * elu(u)}{u - l}, \\ y \geq elu'(l) * (x - l) + elu(l) \\ y \geq elu'(\dfrac{l}{2}) * (x - \dfrac{l}{2}) + elu(\dfrac{l}{2}) \\ l \leq x \leq u. \end{array} \tag{8}$$

From the above discussion, we have the linear encoding for $elu$ is sound:

***Lemma 1.*** $\forall x \in [l, u]$, we have $elu(x)$ satisfies the linear constraints.

*Proof:* It can be proved by cases: in each case, for each lower (resp. upper) bound $f$, we can prove $elu(x) - f(x) \geq 0$ (resp. $f(x) - elu(x) \geq 0$) for each $x \in [l, u]$. □

### 3.2 Linear encoding of some other activation functions

**Sigmoid**. The activation function $sigmoid$ takes a real value as input and outputs another value between 0 and 1. It is defined as

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

Although it gives rise to a problem of "vanishing gradients", $sigmoid$ is easy to work with and has all the nice properties of activation functions: it's non-linear, continuously differentiable, monotonic, and has a fixed output range. Likewise, according to the shape of $sigmoid$, the encoding of $sigmoid$ is divided into three independent cases: $(i)$ $0 \leq l < u$, $(ii)$ $l < u \leq 0$; $(iii)$ $l < 0 < u$.

**Case** $0 \leq l < u$. The shape of $sigmoid$ in this input range is a continuous incremental curve. Similar to the second case of $elu$, we conservatively represent the output range as the area between the upper-lower line and the tangent line on the midpoint.

**Case** $l < u \leq 0$. As $sigmoid$ is centrosymmetric, the second case could be encoded in the similar way as the first case.

**Case** $l < 0 < u$. Thanks to the centrosymmetry of $sigmoid$ again, the encoding of the upper bounds is quite similar to the one of the lower bounds. So we only present the encoding of the lower bounds for these cases. When the slope of the upper-lower line is not larger than the one of the tangent line on the left end-point, then all the outputs are above the upper-lower line, and thus we can use the upper-lower line as a lower bound. Otherwise, we take both the tangent lines on the left end-point and on the symmetry point of the right end-point (*i.e.*, $-u$) as the lower bounds.

**Generalization**. Our encoding can be easily extended to $elu$-like and $sigmoid$-like activation functions, such as $selu$ and $tanh$. It can also be generalized to any activation function $\phi$ that are "pieces" of linear and logistic functions (or convex or concave functions): the cases of the input range contained in only one segment can be encoded as the lower-upper line and the tangent line on the midpoint, while for the other cases the input range crossing over more than one segment would be dependent on the segment functions and require a careful exploration on the lower-upper line and the tangent lines on some special points.

3. It is possible that the right end-point is below this tangent line, and then we should move it downward appropriately.

Note that, similar to [15], [17], we can use mixed integer linear constraints to simplify these cases involving two or more segments, but at the cost of constraint solving time. This is left as a future work.

### 3.3 Output range algorithm

Our algorithm *OutputRange* for output range analysis is given in Algorithm 1, which takes a neural network $\mathcal{N}$ and an input range $\mathcal{I}$ as input and returns an output range $\mathcal{O}$. *OutputRange* consists of two main steps: ($i$) linear encoding for network (lines $1-10$) and ($ii$) the constraint solving (lines $11-12$). To start with for linear encoding, *OutputRange* first initializes the constraint set $C_{\mathcal{N}}$ as an empty set (line 1) and sets the current output range $O_{cur}$ as the input range $\mathcal{I}$ (line 2). Then it proceeds the following operations (lines $3-10$) for the network layer by layer. For each layer, *OutputRange* saves the output range of the last layer in $O_{pre}$ (line 4) and resets the current output range $O_{cur}$ as empty (line 5). And then for each node in the current layer, *OutputRange* computes the relaxed output range before the activation function via the simple algorithm[4] given in Section 2 (line 7), which is the basis for constraint generation. Depending on the output range, *OutputRange* encodes the output range of the current node as the constraint set $C$ (line 8), according to the equations presented in Section 3.1. After that, the constraint set $C$ is added into the set $C_{\mathcal{N}}$ (line 9), and the relaxed output range after the activation function is computed (line 10) as well. While for the second step, according to the constraint set $C_{\mathcal{N}}$, *OutputRange* tries to find respectively the minimum and the maximum as the lower bound and the upper bound for each node of the output layer (line 11), which are kept record in $\mathcal{O}$. Finally, *OutputRange* returns $\mathcal{O}$.

**Algorithm 1:** OutputRange

**Input:** network $\mathcal{N}$, input range $\mathcal{I}$
**Output:** output range $\mathcal{O}$

1 $C_{\mathcal{N}} \leftarrow \emptyset$
2 $O_{cur} \leftarrow \mathcal{I}$
3 **for** $layer\ in\ \mathcal{N}$ **do**
4  $O_{pre} \leftarrow O_{cur}$
5  $O_{cur} \leftarrow \{\}$
6  **for** $node\ in\ layer$ **do**
7  $[l, u] \leftarrow OR_{simple}(node, O_{pre}, \mathsf{act} = false)$
8  $C \leftarrow linearEncode(node, [l, u])$
9  $C_{\mathcal{N}} \leftarrow C_{\mathcal{N}} \cup C$
10  $O_{cur}[node] \leftarrow node.act([l, u])$
11 **for** $node\ in\ last(\mathcal{N})$ **do**
12  $\mathcal{O}[node] \leftarrow [min(node, C_{\mathcal{N}}), max(node, C_{\mathcal{N}})]$
13 **return** $\mathcal{O}$

### 3.4 Network partition strategy

Compared to the ground truth, the output range computed by our algorithm *OutputRange* is still an over-approximation. A naive strategy to reduce the output range is to divide the input range into several ones and then compute their output ranges separately, which has been employed by several existing approaches, such as MaxSens [19] and ReluVal [16], and is applicable here as well. Here we present another partition strategy.

Besides the partition on the input, we can also consider the partition on the network itself to reduce the output range. In detail, according to a span $k$, we split the network into several small ones of size (at most) $k$, that is, the subnetwork consisting of the 1st layer to the $k$-th layer, the subnetwork consisting of the $k+1$-th layer to the $2k$-th layer, and so on. Then we compute the output ranges of the subnetworks in turn, wherein the output range of the previous subnetwork is the input of the next subnetwork. This network partition seems a generalization of "layer-to-layer-refine" in [30], [31]. The algorithm is given in Algorithm 2, where $\mathcal{N}[m, n]$ represents a subnetwork consisting of the $m$-th layer to the $n$-th one in $\mathcal{N}$ and $m \leq n$.

**Algorithm 2:** OutputRangeByK

**Input:** network $\mathcal{N}$, input range $\mathcal{I}$, span $k(> 0)$
**Output:** output range $\mathcal{O}$

1 $n \leftarrow length(\mathcal{N})$
2 $m \leftarrow \lfloor n/k \rfloor$
3 $\mathcal{O} \leftarrow \mathcal{I}$
4 **for** $i = 1 \ldots m$ **do**
5  $\mathcal{O} \leftarrow OutputRange(\mathcal{N}[k * i - k + 1, k * i], \mathcal{O})$
6 **if** $k * m < n$ **then**
7  $\mathcal{O} \leftarrow OutputRange(\mathcal{N}[k * m + 1, n], \mathcal{O})$
8 **return** $\mathcal{O}$

We can find that the simple approach in Section 2 is the one that uses the partition with span 1. Clearly, the simple approach does not consider any dependences between layers, so that it is too relaxed. Conversely, *OutputRange* consider all the dependences between layers. However, it considers the network as a whole, and thus suffers from the accumulation errors (i.e., the relaxed encodings) of all the (previous) layers. Intuitively, the more dependences the algorithm considers, the smaller the output range would be; while the larger span the algorithm takes, the larger the accumulation errors would be. Therefore, one need to make a trade-off on them.

## 4 Experiments

We implemented our approach in Julia, wherein JuMP [32] is used as the modeling language for linear programming and GLPK[5] is employed as the constraint solver. With this implementation, we have conducted a series of experiments to evaluate our approach. All the experiments are run on a machine with 12 Intel Xeon Silver 4214 Processor@2.2GHz, 64GB RAM, running Ubuntu 18.04 operating system.

### 4.1 Dataset and Tools

In the evaluation, we use image classification networks trained for classifying 10 classes of hand-written images

4. One can compute a tighter result according to the collected constraints so far, at the cost of more time.

5. https://github.com/jump-dev/GLPK.jl.

of digits $0-9$ from the database MNIST [33]. The MNIST database is a large collection of hand-written images designed for training various image processing systems in which each image is of size $28 \times 28$ and in black and white. It has $60,000$ training input images and $10000$ testing input images, which belong to one of $10$ labels, namely, from the digit $0$ to the digit $9$. All the image classification networks are built from the Julia Machine Learning Library Flux[6].

We compare our approach, denoted as *SLP*, with RobustVerifier [20], Crown [21], CNN-Cert [22] and DeepCert [23]. In particular, we focus on the linear encodings used by various tools, without any optimization strategies. RobustVerifier [20] encodes the activation functions as the constraints in terms of the 1-order Taylor expansion (*i.e.* , the tangent line) on the midpoint and the possible ranges of the error function. We have implemented it in Julia as well, wherein two versions for computing the range of the error function $r(x)$ at the domain $[l, u]$ are provided. The first version (called the relaxed version, denoted as *RV-R*) computes the range via the interval arithmetic naturally, that is, $[\phi(a) - p(b), \phi(b) - p(a)]$, where $\phi$ is the activation function and $p(x)$ is the tangent line on the midpoint. While the second version (called the tight version, denoted as *RV-T*) computes the lower and the upper bounds according to the shapes of $\phi$ and $p(x)$, that is, from these special points that could make the difference $\phi(x) - p(x)$ to get the maximum/minimum value: the left end-point, the right end-point, the midpoint and the symmetry point of the midpoint (if in the domain). Crown [21] also bounds the activation functions depending on the three cases presented in Section 3 and takes the tangent line at a point in the positive (negative resp.) set that passes the left (right resp.) end-point as the upper (lower resp.) bounds for Case ($iii$). We reimplement the linear encodings for $sigmoid$ and $elu$ used in Crown following its implementation [7] (denoted as *Crown-10*), wherein a binary search, with a tolerated error $0.01$ and the maximum $10$ iterations, is performed to find the linear bound (*i.e.*, tangent lines). However, $10$ iterations may be not enough to get an acceptable result, for example, when the input range is too large or the tangent lines do not exist, especially for the activation function $elu$. So we also implement a variety of Crown (denoted as *Crown-1w*), where the maximum number of iterations is set as $10000$. CNN-Cert [22] and DeepCert [23] use the similar idea with Crown but empoly a conservative and sound binary search. We also reimplement their linear encodings following their implementations [8] [9]. The tools used in experiments are listed in Table 1.

TABLE 1
Tools used in Experiments.

| Tool | Description |
|---|---|
| SLP | Our approach |
| RV-R | Relaxed version of RobustVerifier |
| RV-T | Tight version of RobustVerifier |
| Crown-10 | Crown's Binary search with maximum 10 iterations |
| Crown-1w | Crown's Binary search with maximum 10000 iterations |
| Simple | The simple approach presented in Section 2 |
| CNN-Cert | CNN-Cert's linear encodings |
| DeepCert | DeepCert's linear encodings |

### 4.2 Performance on MNIST

In the section, we evaluate the performance of our approach via varied perturbations and network structures.

**Experiments on** $elu$. The network are trained on the MNIST dataset, so all the input layers for the networks consist of $784$ (*i.e.*, $28 \times 28$) neurons and all the output layers have $10$ neurons. While the number $n$ of hidden layers varies from $3$ to $10$ (also called as network scale here), and the number $s$ of neurons for each hidden layer is set depending on the number $n$: for a network, the numbers of neurons for all the hidden layers are the same and set as $10 * n$.

The activation functions of all the non-output layers are set to $elu$, whose extra constant $\alpha$ is selected from the candidate set $\{0.5, 1.0, 2.0\}$, and the activation functions of the output layer are the identity one. All the networks are trained on the training set via $10$ epochs with a testing accuracy larger than $95\%$.

To form an input range $\mathcal{I}$ for the input layer, we perform a small perturbation on a set of target pixels for each image in MNIST. In detail, similar to RobustVerifier [20], we take the block of size $5 \times 5$ on the top left corner of the image as the target pixels, and consider the disturbance radius $\epsilon$ of the perturbation from the candidate set $\{0.01, 0.1, 0.15, 0.2, 0.25, 0.3\}$. For example, assuming $\epsilon = 0.1$, perturbations varying within the range $[-0.1, 0.1]$ are imposed on the top left corner of the original image to form an input range. Besides, the result pixel values should be clipped with respect to the true values.

To quantitatively validate the output range $\mathcal{O}$, we use the average radius[10] $\overline{r}$ of the output range, that is, $\overline{r} = \sum_{r_i \in \mathcal{O}} r_i / |\mathcal{O}|$, where $r_i$ is the radius of the $i$-th dimension and $|\mathcal{O}|$ is the dimension of $\mathcal{O}$.

We performed our approach on the networks with the first $100$ images from the testing set. The experimental results are given in Figure 4, where the horizontal axis denotes the disturbance radius $\epsilon$, and the vertical axis denotes the average of the average radii $\overline{r}$ of these testing images. Due to similarities, a few results (*e.g.*, $n = 5, 8$) are not given here.

From Figure 4, we can see that the average radius of the output range obtained by our approach is quite small, compared to the simple approach. For example, when the network scale $n$ is smaller than $5$, the average of $\overline{r}$ is smaller than $50$ (see Figures 4a-4b, 4g-4h, 4m-4n). As the network scale $n$ increases, $\overline{r}$ increases fast. The reason may be that more errors could be accumulated rapidly for the larger networks. Moreover, the larger disturbance radius $\epsilon$ could yield the larger $\overline{r}$, due to the monotonicity of $elu$.

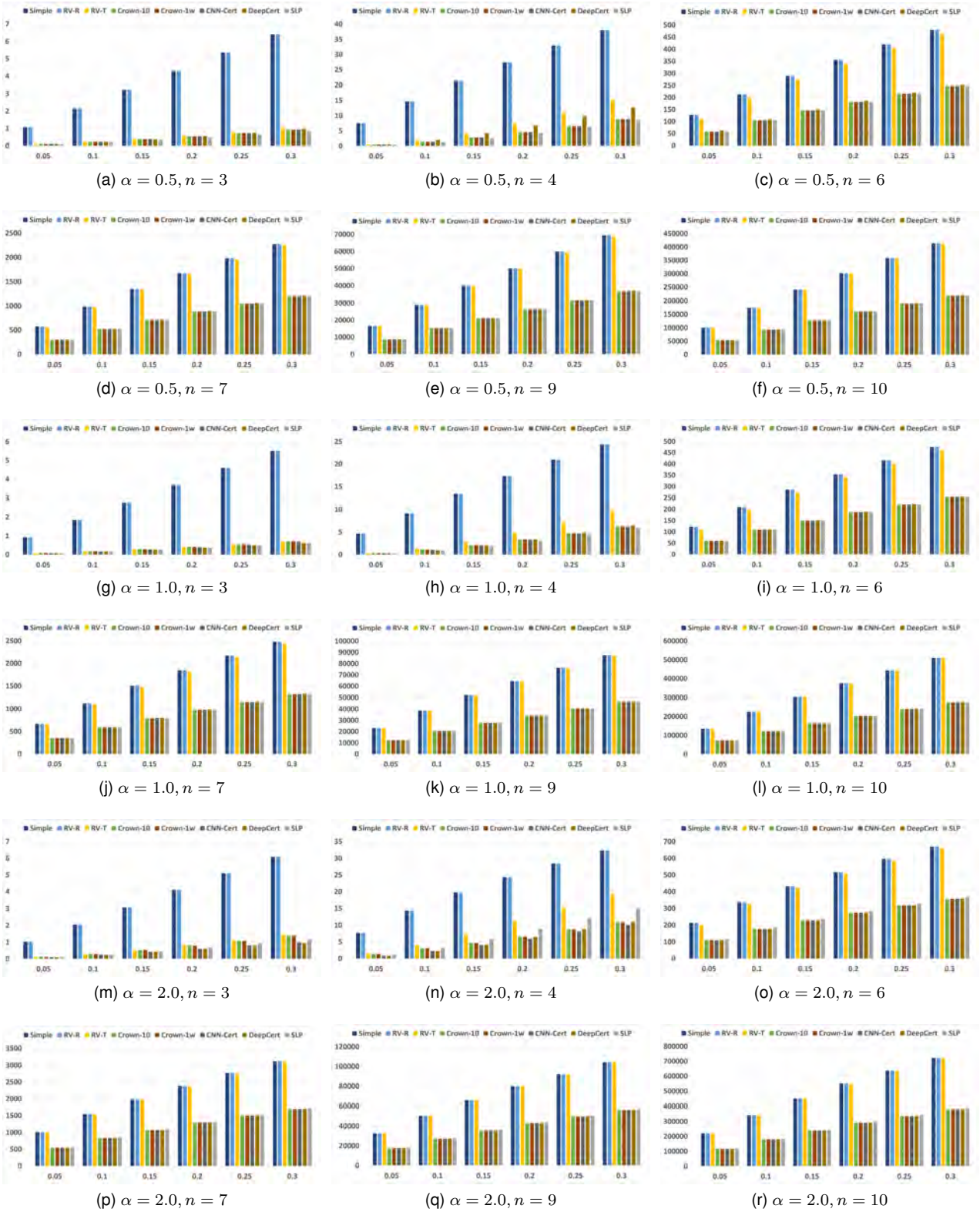
Compared to the tight version *RV-T* of RobustVerifier, our approach *SLP* can obtain a smaller average radius $\overline{r}$ for all cases. As the disturbance radius $\epsilon$ or the network

6. https://fluxml.ai/Flux.jl/stable/.
7. https://github.com/CROWN-Robustness/Crown.
8. https://github.com/AkhilanB/CNN-Cert.
9. https://github.com/VivianWu0512/DeepCert.

10. In our implementation, we also use the integral of the output range $\mathbf{I}(= \prod_{r_i \in \mathcal{O}} r_i)$ to quantify the results, which is quite similar to the average radius but with a larger scale, and thus is omitted.

This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081






(a) $\alpha = 0.5, n = 3$ (b) $\alpha = 0.5, n = 4$ (c) $\alpha = 0.5, n = 6$

(d) $\alpha = 0.5, n = 7$ (e) $\alpha = 0.5, n = 9$ (f) $\alpha = 0.5, n = 10$

(g) $\alpha = 1.0, n = 3$ (h) $\alpha = 1.0, n = 4$ (i) $\alpha = 1.0, n = 6$

(j) $\alpha = 1.0, n = 7$ (k) $\alpha = 1.0, n = 9$ (l) $\alpha = 1.0, n = 10$

(m) $\alpha = 2.0, n = 3$ (n) $\alpha = 2.0, n = 4$ (o) $\alpha = 2.0, n = 6$

(p) $\alpha = 2.0, n = 7$ (q) $\alpha = 2.0, n = 9$ (r) $\alpha = 2.0, n = 10$

Fig. 4. Averages of $\overline{r}$ for $elu$ networks with different parameters $\alpha$s, sizes $n$s and disturbance radii $\epsilon$s

scale $n$ increases, the differences between *SLP* and *RV-T* increase as well. Moreover, the results of *RV-T* get close to the ones of *Simple* for large networks or large perturbations. For example, when the network scale $n$ is larger than 6, the results are very close to the ones of *Simple* for all considered disturbance radii (see Figures 4c-4f, 4i-4l, 4o-4r). Even worse, the relaxed version *RV-R* of RobustVerifier performs as poor as the simple approach presented in Section 2. It is easy to show that the error bounds computed by the relaxed RobustVerifier are the same as the bounds computed by

the simple approach, since they are all computed by the simple interval arithmetic. The above results indicate that our approach performs better than RobustVerifier.

Compare our approach to *Crown*. For that, let us first consider the $elu$ networks with $\alpha = 0.5$ or $1.0$. Compared to *Crown-10*, our approach *SLP* gets a smaller average radius $\overline{r}$ for small networks or small perturbations, but obtains a larger $\overline{r}$ for large networks or large perturbations. There are two reasons for this. First, *Crown-10* tries to look for a line connecting the left (right resp.) end-point and a point in the positive (negative resp.) set in $10$ iterations such that it is close to the "tangent" line within a tolerated error. But this may be unsound. To avoid unsound results as possible, *Crown-10* would move the result line up (down resp.) with respect to the tolerated error, which could yield a relaxed result, especially for small network or small perturbations. Second, when the input range is too large or the tangent line does not exist, $10$ iterations may not be enough to get an acceptable result for large networks or large perturbations, even with the fine movement. For example, when $n >= 8$, most of the average radii are larger than $1024$ (*i.e.*, $2^{10}$) and the binary search would stop at the 10th iteration with an unsound bound that still have a gap larger than the tolerated error (*i.e.*, $1024/2^{10} > 0.01$). We do believe the final result of *Crown-10* for a whole network is sound in sense that it contains the ground truth due to the relaxed encodings and the fine movement, but the soundness of the binary search is hard to guaranteed. While compared to *Crown-1w*, our approach *SLP* performs as good as or better than *Crown-1w* for all the cases. This is because more iterations could make *Crown-1w* to take a line that is more close to a specialized line connecting the left (right resp.) end-point and either end-point of the positive (negative resp.) set, which are used in our approach. Moreover, our approach uses more constraints for some cases such as Case ($iii$).

When $\alpha = 2.0$, our approach *SLP* gets a better result than both *Crown-10* and *Crown-1w* for $n = 3$, but obtains a worse result for the other cases. A reason for this is that, as discussed in Section 3, the linear encoding used in our approach is more relaxed than the one in Crown for Subcase $elu(l) \leq l$, wherein the tangent line for the lower bound exists. However, similar to the discussion above, we are not sure the better results of Crown are due to the tighter bounds or the unsound bounds obtained by the binary search. Nevertheless, our results (with an average $38818.23$ of all the average radii) are quite close to both *Crown-10* (with an average $37912.91$) and *Crown-1w* (with an average $37929.39$). There are two possible reasons. First, one more bound is used in our approach for Subcase $elu(l) > l$ to get a tighter result, as shown in Figure 2f. Second, a better bound is used for Subcase $elu(l) \leq l$: as shown in Figure 2e, we use the line connecting the right end-point and the origin as the upper bound, which could be better than Crown's encoding (*i.e.*, the line connecting the left end-point and the origin), as $l$ is quite small and $u$ may be large.

We also compare our approach to *CNN-Cert* and *DeepCert*, whose results are also given in Figure 4. Similarly to *Crown*, *SLP* gets a better result when $\alpha = 0.5$ or $1.0$, but a worse result when $\alpha = 2.0$. The reasons may be that: (i) the target tangent lines do not exist when $\alpha = 0.5$ or $1.0$, and as discussed in Section 3, the conservative search always yields a "suboptimal" bound; (ii) when $\alpha = 2.0$, the target tangent line exists in most cases, the conservative search enables *CNN-Cert* and *DeepCert* to get a bound closed to the tangent line. Although worse, our results (with an average $38818.23$ of all the average radii) are quite close to both *CNN-Cert* (with an average $37937.22$) and *DeepCert* (with an average $38114.04$).

To analyze the results further, we consider the inclusions (*i.e.*, tighter relations in Section 2) of the output ranges obtained by different approaches. The results are given in Figure 5, where the horizontal axis denotes the network scale $n$, *A-in-B* denotes the percentage of the output ranges obtained by *A* are included in those by *B*, and *S*, *Rt*, *C10*, *C1w*, *CC* and *DC* are short for *SLP*, *RV-t*, *Crown-10*, *Crown-1w*, *CNN-Cert* and *DeepCert*, respectively. Due to similarities, a few results (*e.g.*, $\epsilon = 0.15, 0.25$) are not given here. As shown in Figure 5, most (about $93.07\%$ on average) of our results are included in RobustVerifier's, while few (only one time in total in all our experiments) of RobustVerifier's results are subsets of ours. Moreover, the larger perturbation, the larger percentage, and all our results are included in RobustVerifier's for large networks. This demonstrates that our encoding is tighter than RobustVerifier's on $elu$.

Considering the $elu$ networks with $\alpha = 0.5$ or $1.0$, all our results are included in *Crown-1w*'s, but only $0.38\%$ of *Crown-1w*'s results are included in ours (see *S-in-C1w* in Figures 5a-5h). It is a bit strange for *Crown-10*: from Figures 5a-5h, we also found that all our results are included in *Crown-10*'s for small networks (*e.g.*, $n = 3, 4, 5$) but, on the contrary, all *Crown-10*'s results are included in ours for large networks (*e.g.*, $n = 8, 9, 10$). As explained above, the relaxed results of small networks is mainly due to the tolerated errors, while the "tight" results of large networks is mainly due to the insufficient iterations (*i.e.*, unsound bounds). This can be further demonstrated by the inclusions between *Crown-10* and *Crown-1w*: all *Crown-10*'s results are included in *Crown-1w*'s, while few of *Crown-1w*'s results are in *Crown-10*'s. In addition, it is not easy to guarantee that $10000$ iterations are sufficient to find a sound bounds for $elu$, especially for the case wherein there are no tangent lines. In fact, in our experiments, we also found that there are still many times that the binary search stops at the $10000$th iteration.

For the $elu$ networks with $\alpha = 2.0$, most of *Crown*'s results, including *Crown-10* ($81.52\%$) and *Crown-1w* ($81.46\%$), are included in ours, while few of our results are included in *Crown*'s ($2.81\%$ for *Crown-10* and $2.83\%$ for *Crown-1w*). However, as shown in Figures 5i-5l, all *Crown-10*'s results are included in *Crown-1w*'s, while few of *Crown-1w*'s results are in *Crown-10*'s. Similar to the discussion above, we are not sure the better results of Crown are due to the tighter bounds or the unsound bounds.

Compared to *CNN-Cert* and *DeepCert*, on the $elu$ networks with $\alpha = 0.5$ or $1.0$, most of our results are included in *CNN-Cert*'s ($95.38\%$) and *DeepCert*'s ($84.30\%$), while few of their results are included in ours ($0.98\%$ for *CNN-Cert* and $1.40\%$ for *DeepCert*). This further demonstrates that our encodings can get a tighter results than *CNN-Cert* ($95.38\%$) and *DeepCert* the $elu$ networks with $\alpha = 0.5$ or $1.0$. However, when $\alpha = 2.0$, our encodings perform worse than *CNN-Cert* and *DeepCert*: $94.94\%$ of *CNN-Cert*'s results are included in ours, while $1.21\%$ of ours are included in *CNN-Cert*'s; and

This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081




(a) $\alpha = 0.5, \epsilon = 0.05$ (b) $\alpha = 0.5, \epsilon = 0.10$

(c) $\alpha = 0.5, \epsilon = 0.20$ (d) $\alpha = 0.5, \epsilon = 0.30$

(e) $\alpha = 1.0, \epsilon = 0.05$ (f) $\alpha = 1.0, \epsilon = 0.10$

(g) $\alpha = 1.0, \epsilon = 0.20$ (h) $\alpha = 1.0, \epsilon = 0.30$

(i) $\alpha = 2.0, \epsilon = 0.05$ (j) $\alpha = 2.0, \epsilon = 0.10$

(k) $\alpha = 2.0, \epsilon = 0.20$ (l) $\alpha = 2.0, \epsilon = 0.30$

Fig. 5. The inclusions between different approaches for $elu$ networks with different parameters $\alpha$s, sizes $n$s and disturbance radii $\epsilon$s

for *DeepCert* these two percentages are $91.5\%$ and $0.77\%$. Similar to the discussion above, *CNN-Cert* and *DeepCert* can find a bound that is closed to the tangent line if it exists; otherwise, it always returns a conservative result.

Finally, Figure 6 shows the total runtimes (in seconds) for $elu$ networks with $\alpha = 1$[11] under different sizes $n$s and different disturbance radii $\epsilon$s. Overall, our approach costs the most runtime with an average $6.02s$, since more constraints are used to get a tighter bound. While the simple approach costs the fewest runtime, since it does not involve constraint solving. Moreover, *RobustVerifier* (with an average runtime $2.32$) is more efficient than *Crown* (with an average runtime $4.87$), mainly due to the binary search used in *Crown* and

11. The runtimes for $\alpha = 0.5$ or $2.0$ are quite similar.

the relaxed constraints used in *RobustVerifier*. In particular, *Crown-10* performs a little better than *Crown-1w*, and *RV-R* performs much better than *RV-T* and is quite close to the simple approach. *DeepCert* (with an average runtime 3.26) is more efficient than *CNN-Cert* (with an average runtime 5.23), which is because that *DeepCert* improves *CNN-Cert* by using the lower-upper line if possible and thus avoids the binary search. As a notice, in fact, *Crown*[12] computes the output range via the closed-from bounds [21] and can be further optimized without constraint solving [25], thus would be more efficient than our implementation. However, the results would be similar to our implementation via linear programming [21].

**Experiments on** $sigmoid$. Experiments on networks with the activation function $sigmoid$ are performed to evaluate our approach. In these experiments, all the activation functions are set to $sigmoid$, except the ones of the output layers. Moreover, due to the "vanishing gradient" problem, we only consider the number $n$ of hidden layers from 3 to 8. The average of the average radius $\overline{r}$ and the inclusions between different approaches are shown in Figure 7, where the notations are the same to the ones in Figures 4 and 5.

Figures 7a-7f show that our approach performs better than RobustVerifier on $sigmoid$ networks in terms of the average radius. Similar to $elu$, for small networks, the results of our approach and *RV-T* are quite close. But as the size of networks increases, the difference between them would become larger. *RV-R* still performs as poor as the simple approach. Nevertheless, due to the bounded output of $sigmoid$ and the accumulation error, as the network scale or the disturbance radius increases, both the results obtained by our approach and *RV-T* get close to the one by the simple approach (*e.g.*, Figure 7f). Let us consider the inclusion. Although the results are close, most (above $90\%$) of our results are included in *RV-T*'s for the small networks (*e.g.*, Figure 7a). While for the large networks, this inclusion radio decreases instead, although the *RV-T*'s average radius is larger than ours (*e.g.*, Figure 7f). On average, in our experiments, about 71.09% of our results are included in *RV-T*'s, and none of *RV-T*'s results are included in ours. This demonstrates our encoding is tighter than RobustVerifier's on $sigmoid$.

Similar to $elu$, our approach gets a smaller average radius than *Crown* (including *Crown-10* and *Crown-1w*) for small networks or small perturbations, but obtains a larger average radius for large networks or large perturbations. And the differences would become larger as the network scale or the disturbance radius increases. The averages of all the average radii for *SLP* and *Crown* are $0.95$ and $0.76$, respectively. However, there are more cases that our results are included in *Crown*'s (including *Crown-10* and *Crown-1w*): on average, in our experiments, about $68.24\%$ of our results are included in *Crown*'s, while about $22.03\%$ of *Crown*'s results are included in ours. So, in terms of inclusion, our approach performs better than *Crown* in our experiments.

In addition, although *Crown-10* and *Crown-1w* perform the "same" on $sigmoid$ from the view of our approach, Figures 7g-7j show that all *Crown-10*'s results are included in *Crown-10*'s, but not all *Crown-1w*'s results are included in *Crown-10*'s. In other words, 10-iteration may not be sufficient for $sigmoid$ on some situation, as increasing the iteration could make the output range larger. For example, 10-iteration may not make the slope difference between the found line and the tangent line smaller than the tolerated error $0.01$, especially for large input ranges.

Compared to *CNN-Cert* and *DeepCert*, our approach gets a larger average radius, especially for larger networks or large perturbations. The averages of all the average radii for *SLP*, *CNN-Cert* and *DeepCert*, are $0.95$, $0.75$ and $0.73$, respectively. Concerning inclusion, about $20.37\%$ of our results are included in *CNN-Cert*'s, while about $43.35\%$ of *CNN-Cert*'s results are included in ours. For *DeepCert*, these two percentages become $1.03\%$ and $29.60\%$, respectively.

Finally, Figure 8 shows the total runtimes (in seconds) for $sigmoid$ networks under different sizes $n$s and different disturbance radii $\epsilon$s. Similar to the results on $elu$ networks, our approach costs the most runtime on average, followed by *Crown*, *CNN-Cert*, *DeepCert*, *RobustVerifier* and the simple solution, with the average runtimes $0.78$, $0.73$, $0.72$, $0.72$, $0.39$, and $0.0015$, respectively.

### 4.3 Network partition

We also perform experiments to evaluate the network partition strategy. The networks in these experiments are trained on the MNIST dataset as well, whose structure settings are as follows: the number $n$ of hidden layers ranges from $10$ to $50$, the numbers of neurons for each layers are randomly generated from $30$ to $100$, the disturbance radius $\epsilon$ is set to $0.1$, and the activation function is $elu$. All the testing accuracies of these networks are above $92\%$.

For a network with $n$ hidden layers, we take the partition span $k$ ranging from $1$ to $n$, respectively. And for each span $k$, we perform the output analysis on the first $5$ images from the testing set. The ratios of the average radius of the output ranges obtained by different spans to the one by the simple approach and the average runtimes are given in Figure 9, where the horizontal axis denotes the partition span $k$, the left and the right vertical axises denote the average radius ratios to the simple approach and the runtimes in seconds, respectively.

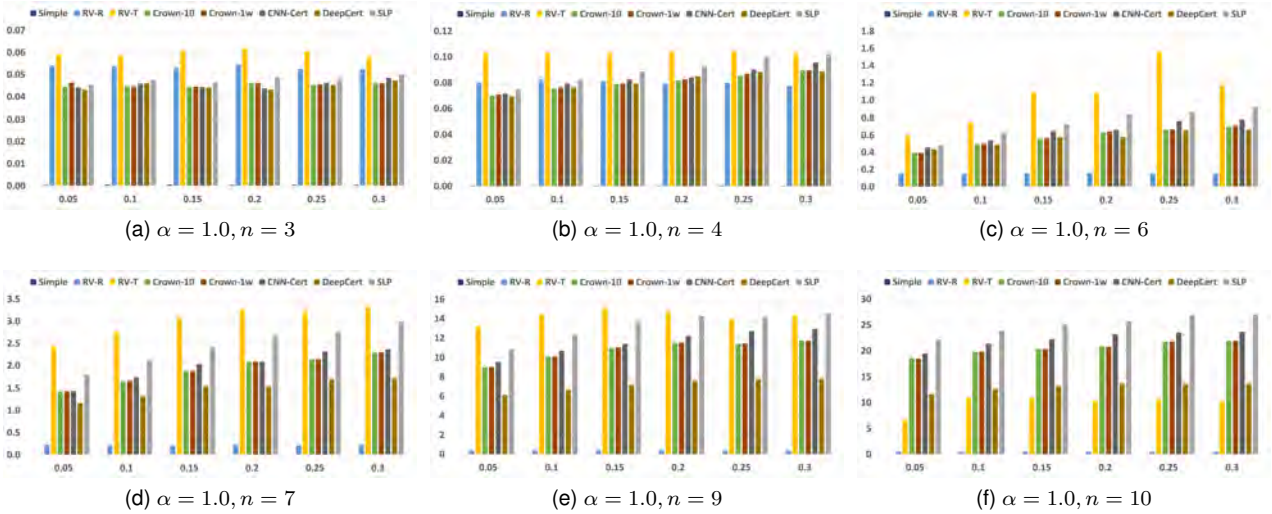
Firstly, the average radius decreases first and then increases. The main reason is that the gain of the dependences between layers would decrease gradually, while the effect of the accumulation errors would increase gradually. As shown in Figure 9, the minimum values of the average radius are obtained at span 2 or 3 for all the networks, which indicates that 2-layer or 3-layer dependences seems enough for the activation function $elu$. This indicates the network partition helps to achieve a tight result. Moreover, the maximum values are reached at span $1$ for all the networks, because, as discussed in Section 3.4, no dependences between layers are considered for span 1. Interestingly, the results obtained by span $n-1$ is always worse than the ones by span $n$, which is also due to that no dependences are considered for the last layer when taking span $n-1$. This demonstrates the usefulness of dependences.

Secondly, the runtime increases gradually as the span increases. Generally, as the span increases, although the solving iteration decreases, the number of the generated

12. Similarly to *CNN-Cert* and *DeepCert*.

This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081

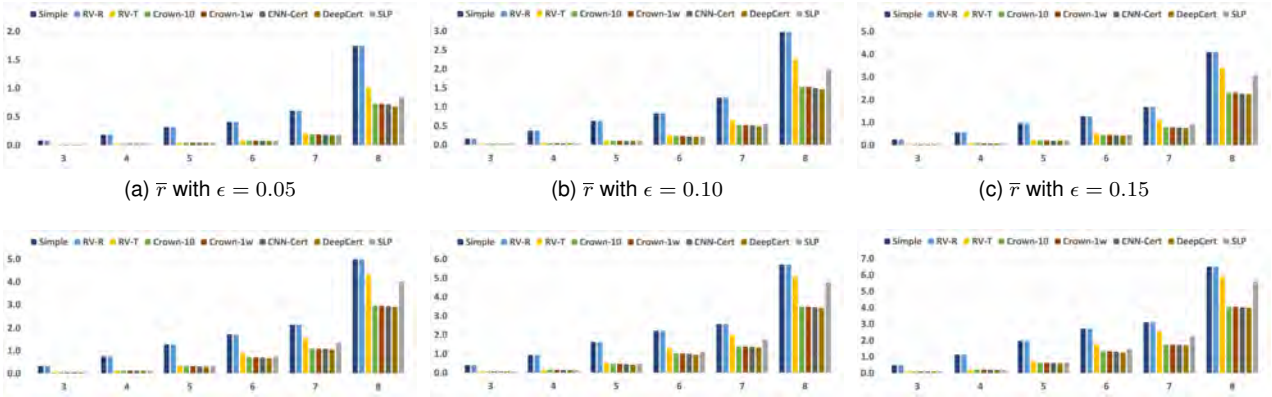





(a) $\alpha = 1.0, n = 3$ (b) $\alpha = 1.0, n = 4$ (c) $\alpha = 1.0, n = 6$

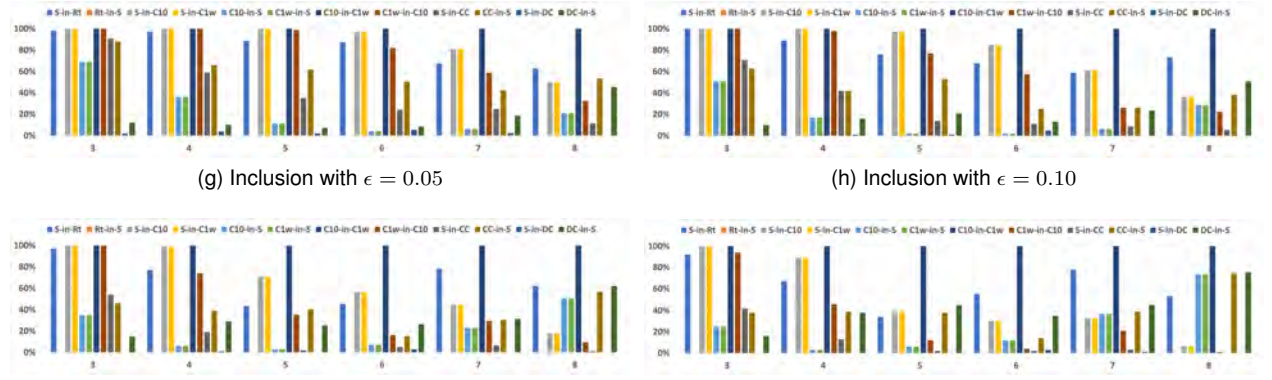
(d) $\alpha = 1.0, n = 7$ (e) $\alpha = 1.0, n = 9$ (f) $\alpha = 1.0, n = 10$

Fig. 6. Runtimes (in seconds) for $elu$ networks with $\alpha = 1$ under different sizes $n$s and different disturbance radii $\epsilon$s




(a) $\overline{r}$ with $\epsilon = 0.05$ (b) $\overline{r}$ with $\epsilon = 0.10$ (c) $\overline{r}$ with $\epsilon = 0.15$

(d) $\overline{r}$ with $\epsilon = 0.20$ (e) $\overline{r}$ with $\epsilon = 0.25$ (f) $\overline{r}$ with $\epsilon = 0.30$



(g) Inclusion with $\epsilon = 0.05$ (h) Inclusion with $\epsilon = 0.10$

(i) Inclusion with $\epsilon = 0.20$ (j) Inclusion with $\epsilon = 0.30$

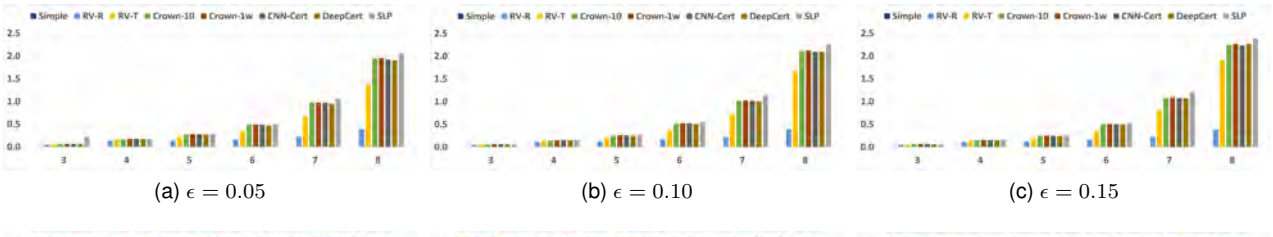
Fig. 7. The average radius $\overline{r}$ and the inclusions between different approaches for $sigmoid$ networks with different $n$s and $s$s

constraints for each subnetwork increases and thus the runtime for each subnetwork increases. Note that $k = n$ means the network partition is not used. This indicates the network partition helps to improve the efficiency. The maximum times among all the network partitions for networks with $n = 10, 20, 30, 40, 50$ are $0.267$s, $0.875$s, $0.851$s, $1.622$s, $4.719$s, respectively. Clearly, the larger the network, the more the runtime. Nevertheless, our approach is still efficient for large networks, as we can see that the solving time for $n = 50$ is smaller than $5$s and it costs less than $1$s for each $10$ layers on average.

Based on the above results, we found that a large span suffers from more accumulation errors and costs more runtime. Therefore, to make a tradeoff, one can perform the partition spans ranging from 2 to $\sqrt{n}$ parallelized and select the best result. Moreover, one can consider to split the

This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081

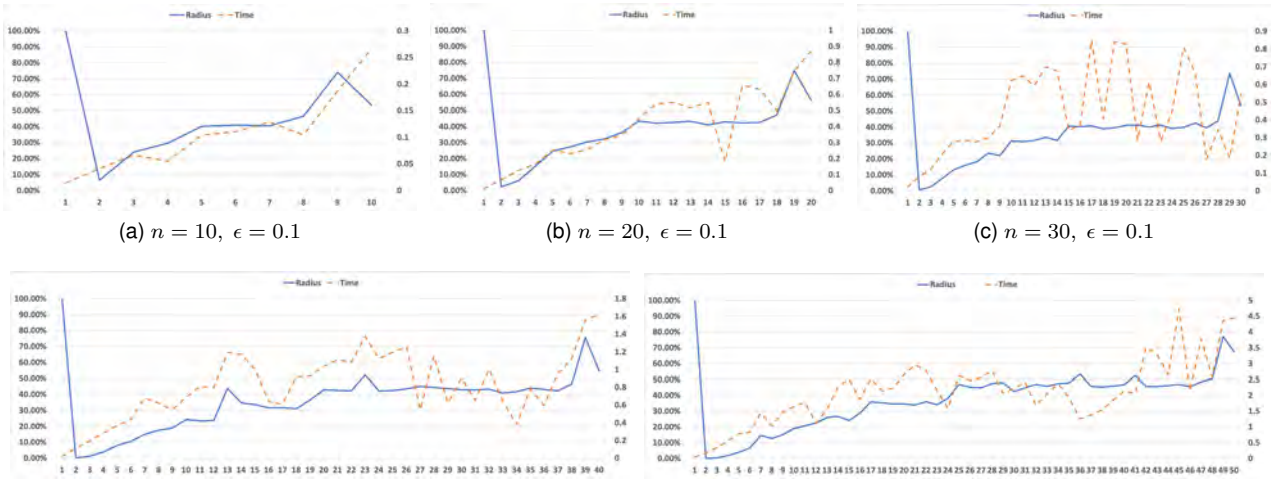





(a) $\epsilon = 0.05$ (b) $\epsilon = 0.10$ (c) $\epsilon = 0.15$

(d) $\epsilon = 0.20$ (e) $\epsilon = 0.25$ (f) $\epsilon = 0.30$

Fig. 8. Runtimes (in seconds) for $sigmoid$ networks with different $n$s and $s$s




(a) $n = 10,\ \epsilon = 0.1$ (b) $n = 20,\ \epsilon = 0.1$ (c) $n = 30,\ \epsilon = 0.1$

(d) $n = 40,\ \epsilon = 0.1$ (e) $n = 50,\ \epsilon = 0.1$

Fig. 9. The average radius ratios and runtimes (in seconds) for $elu$ networks with different scales and spans

network with different spans, for example, a network with 9 hidden layers can be partitioned into three subnetworks with 2, 3, and 4 hidden layers, respectively. This is left as a future work.

## 5 Related Work

Most approaches for output range analysis or robustness verification focus on the $relu$ or piecewise linear activation functions, as they are easy to encode as linear constraints (or linear in some senses). Katz [13] proposed a novel, scalable, and efficient technique for verifying properties (expressed as bound constraints) of feed-forward $relu$ neural networks, based on the simplex method, a standard algorithm for solving LP instances. Lomuscio and Maganti [28] proposed a reachability analysis for feed-forward $relu$ neural networks via linear programming. Ehlers [14] presented an approach to find an activation pattern of $relu$ networks that maps an input in $X$ to an output not in $Y$, where a SAT solver is integrated with for tree search in the function space. Xiang et al [31] proposed a layer-by-layer approach for the output reachable set computation of $relu$ neural networks, which is formulated in the form of a set of manipulations for a union of polytopes. Dutta et al [15], [34] presented an efficient range estimation algorithm that iterates between an expensive global combinatorial search, and a relatively inexpensive local optimization that repeatedly seeks a local optimum of the function represented by the Networks. Wang et al [16], [35] leveraged symbolic interval analysis along with several other optimizations to minimize overestimations of output bounds for $relu$-based DNNs. Bunel et al [36] used branch and bound to compute the output bounds of a network. Their approach has a modularized design that can serve as a unified framework that can support other methods such as Katz's approach [13] and Ehlers's approach [14]. Weng et al [11] provided for $relu$ networks two computationally efficient algorithms (Fast-Lin, Fast-Lip) that are able to certify non-trivial lower bounds of minimum adversarial distortions. Wong and Kolter [37] proposed an efficient optimization approach to estimate the bounds on the output for $relu$-based networks based on dual optimization. Gehr et al [38] introduced abstract transformers that capture the behavior (*i.e.* output ranges) of fully connected and convolutional neural network layers with $relu$ as well as max pooling layers. Tjeng et al [17]

also encoded the network as a set of mixed integer linear constraints, wherein both interval arithmetic and linear programming are used to tighten the bounds. Prabhakar and Afzal [39] presented a novel abstraction technique that constructs a simpler neural network with fewer neurons, albeit with interval weights called interval neural network (INN), which over-approximates the output range of the given neural network. However, these above approaches cannot handle the other activation functions, such as $sigmoid$ and $tanh$.

There are some approaches that take $sigmoid$ and $tanh$ functions into account. Pulina and Tacchella [40] proposed an abstraction-refinement approach to over-approximate the output of networks with logistic functions and then verify them. Dutta et al [15] also presented the encodings for more general activation functions including $sigmoid$ and $tanh$ functions as three or more linear "pieces", but the encoding have to include an error estimate that bounds away the differences between the original function and its piecewise approximation. Lin et al [20] proposed an approach for robustness verification of classification deep neural networks with $sigmoid$ activation function via linear programming, wherein the linear encodings could be extended to the other activation functions. Compared with our approach, these above approaches could be too relaxed. Ivanov et al [41] transformed the neural network into an equivalent hybrid system and computed the reachable space via existing reachability tools such as dReach and Flow$\star$. And their approach is further used in [42]. ERAN [43], [44] is a robustness analyzer for neural networks based on abstract interpretation. But besides $relu$, ERAN supports $sigmoid$ and $tanh$ activation functions as well. But these approaches may be too "abstract" such that they are too heavy to use or the bounds are too relaxed.

Moreover, some approaches are suitable for more non-$relu$ activation functions. Huang et al [30] developed an approach to construct a reachable set for feed-forward multilayer neural networks based on Satisfiability Modulo Theory (SMT). Ruan et al [18] proposed a reachability analysis tool for feed-forward deep neural networks with nonlinear activation functions, based on a Lipschitz constant of the network. Xiang et al [19] formulated the output reachable set estimation problem for neural networks into a chain of optimization problems. Dvijotham et al [45] proposed a verification framework that can apply to a general class of activation functions, wherein they solved a Lagrangian relaxation of the optimization problem to obtain bounds on the output. Raghunathan et al [46] adopted a semidefinite relaxation to compute over approximated certificates (i.e., bounds) for any activation function that is differentiable almost everywhere, but this approach only works for neural networks with only one hidden layer. Zhang et al [21] proposed a general framework Crown to certify robustness of neural networks with general activation functions. Crown bounds the activation functions depending on the three cases presented in Section 3 and takes the tangent line at a point in the positive (negative resp.) set that passes the left (right resp.) end-point as the upper (lower resp.) bounds for Case ($iii$). Although it is sound theoretically, Crown employs a binary search to find the tangent lines, which is prone to yield an unsound result. CNN-Cert [22] uses the same idea with Crown but employs a conservative and sound binary search, and DeepCert [23] improves CNN-Cert by using the lower-upper line down if possible. But they may get a relaxed result if the tangent lines do not exist. To get a tighter result, FROWN [25] performs a gradient-based search for optimal upper and lower bounds, at the cost of more runtimes.

## 6 Conclusion

In this paper, we have proposed an approach to compute the output range for neural networks via linear programming. The key idea is to encode the activation functions as linear constraints in term of the line between the left and right end-points of the input range and the tangent lines on some special points in the input range. We also have carried out some interesting experiments to evaluate our approach, which show that our approach can get a tight output range, and performs better than some existing approaches RobustVerifier, Crown, CNN-Cert and DeepCert on $elu$ networks with $\alpha = 0.5$ or 1.

As for future work, we will consider whether the bounds with smaller area lead to a tighter result or an output range with smaller area theoretically. We will also consider the verification problem based on the linear programming as well as the strategies to improve the performance of verification. More experiments on some other dataset are also considered.

## Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Nos. 61836005, 61972260, 61772347), the Guangdong Basic and Applied Basic Research Foundation (No. 2019A1515011577) and the Stable Support Programs of Shenzhen City (No. 20200810150421002).

## References

[1] A. C. Serban, E. Poll, and J. Visser, "Adversarial examples-a complete characterisation of the phenomenon," *arXiv preprint arXiv:1810.01185*, 2018.

[2] Z. Zhang, J. Geiger, J. Pohjalainen, A. E.-D. Mousa, W. Jin, and B. Schuller, "Deep learning for environmentally robust speech recognition: An overview of recent developments," *ACM Transactions on Intelligent Systems and Technology*, vol. 9, no. 5, pp. 1–28, 2018.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *27th Conference on Neural Information Processing Systems*, 2014, pp. 3104–3112.

[4] T. S. John and T. Thomas, "Adversarial attacks and defenses in malware detection classifiers," in *Handbook of Research on Cloud Computing and Big Data Applications in IoT*. 2019, pp. 127–150.

[5] K. Ren, Q. Wang, C. Wang, Z. Qin, and X. Lin, "The security of autonomous driving: Threats, defenses, and future directions," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 357–372, 2019.

[6] O. Ibitoye, R. Abou-Khamis, A. Matrawy, and M. O. Shafiq, "The threat of adversarial attacks on machine learning in network security–a survey," *arXiv preprint arXiv:1911.02621*, 2019.

[7] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *J. Big Data*, vol. 2, p. 1, 2015.

[8] A. Giusti, J. Guzzi, D. C. Ciresan, F. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics Autom. Lett.*, vol. 1, no. 2, pp. 661–667, 2016.

[9] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations*, 2014.

[10] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 42, 2019.

[11] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon, "Towards fast computation of certified robustness for relu networks," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 5273–5282.

[12] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for Verifying Deep Neural Networks," *arXiv preprint arXiv:1903.06758*, 2019.

[13] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proceedings of 29th International Conference on Computer Aided Verification*, 2017, pp. 97–117.

[14] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Proceedings of 15th International Symposium on Automated Technology for Verification and Analysis*, 2017, pp. 269–286.

[15] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *Proceedings of 10th International Symposium on NASA Formal Methods*, 2018, pp. 121–138.

[16] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *Proceedings of 27th USENIX Security Symposium*, 2018, pp. 1599–1614.

[17] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proceeding of 7th International Conference on Learning Representations*, 2019.

[18] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *Proceedings of 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 2651–2659.

[19] W. Xiang, H. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, 2018.

[20] W. Lin, Z. Yang, X. Chen, Q. Zhao, X. Li, Z. Liu, and J. He, "Robustness verification of classification deep neural networks via linear programming," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 418–11 427.

[21] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Proceedings of 32nd International Conference on Neural Information Processing Systems*, 2018, p. 4944–4953.

[22] A. Boopathy, T. Weng, P. Chen, S. Liu, and L. Daniel, "Cnn-Cert: An efficient framework for certifying robustness of convolutional neural networks," in *Proceedings of 33rd AAAI Conference on Artificial Intelligence*, 2019, pp. 3240–3247.

[23] Y. Wu and M. Zhang, "Tightening robustness verification of convolutional neural networks with fine-grained linear approximation," in *Proceedings of 35th AAAI Conference on Artificial Intelligence*, 2021, pp. 11 674–11 681.

[24] P. Henriksen and A. R. Lomuscio, "Efficient neural network verification via adaptive refinement and adversarial search," in *Proceedings of 24th European Conference on Artificial Intelligence*, 2020, pp. 2513–2520.

[25] Z. Lyu, C. Y. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel, "Fastened CROWN: Tightened Neural Network Robustness Certificates," in *Proceedings of 34th AAAI Conference on Artificial Intelligence*, 2020, pp. 5037–5044.

[26] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," *32nd Conference on Neural Information Processing Systems*, 2019, pp. 9832–9842.

[27] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *29th Conference on Neural Information Processing Systems*, 2016, pp. 2613–2621.

[28] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," *CoRR*, vol. abs/1706.07351, 2017.

[29] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *arXiv preprint arXiv:1511.07289*, 2016.

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

[31] W. Xiang, H. D. Tran, J. A. Rosenfeld, and T. T. Johnson, "Reachable set estimation and safety verification for piecewise linear systems with neural network controllers," in *Proceedings of American Control Conference*, 2018, pp. 1574–1579.

[32] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[33] Y. LeCun, C. Cortes, and C. J. Burges, *The MNIST database of handwritten digits*, 1998, http://yann.lecun.com/exdb/mnist/.

[34] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Learning and verification of feedback control systems using feedforward neural networks," in *Proceeding of 6th IFAC Conference on Analysis and Design of Hybrid Systems*, 2018, pp. 151–156.

[35] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *31st Conference on Neural Information Processing Systems*, 2018, pp. 6369–6379.

[36] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," in *31st Conference on Neural Information Processing Systems*, 2018, pp. 4795–4804.

[37] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proceedings of 35th International Conference on Machine Learning*, 2018, pp. 5283–5292.

[38] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *Proceedings of 2018 IEEE Symposium on Security and Privacy*, 2018, pp. 3–18.

[39] P. Prabhakar and Z. R. Afzal, "Abstraction based output range analysis for neural networks," in *32nd Conference on Neural Information Processing Systems*, 2019, pp. 15 762–15 772.

[40] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Proceedings of 22nd International Conference on Computer Aided Verification*, 2010, pp. 243–257.

[41] 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.

[42] K. D. Julian and M. J. Kochenderfer, "A reachability method for verifying dynamical systems with deep neural network controllers," *CoRR*, vol. abs/1903.00520, 2019.

[43] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," in *31st Conference on Neural Information Processing Systems*, 2018, pp. 10 825–10 836.

[44] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 41:1–41:30, 2019.

[45] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," in *Proceedings of 34th Conference on Uncertainty in Artificial Intelligence*, 2018, pp. 550–559.

[46] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *Proceedings of 6th International Conference on Learning Representations*, 2018.


**Zhiwu Xu** received the Ph.D. degree in Computer Science from University Paris Diderot - Paris 7 and University of Chinese Academy of Sciences under the joint cultivation in 2013. Zhiwu Xu is currently an associate professor with College of Computer Science and Software Engineering, Shenzhen University. His research interests include program analysis and verification, type systems, software security, automata theory and logic, and machine learning.

This article has been accepted for publication in IEEE Transactions on Reliability. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TR.2022.3209081



**Yazheng Liu** received the M.S. degree in Software Engineering from Shenzhen University in 2021. His research interests include program verification and machine learning.

**Shengchao Qin** received the Ph.D. degree in Applied Mathematics from Peking University in 2002. His research interests lie mainly in formal methods, software engineering and programming languages, in particular, formal specification and modelling, program analysis and verification, theories of programming, program logic such as separation logic.

**Zhong Ming** received the Ph.D. degree in Computer Science and Technology from Sun Yat-sen University in 2003. He is currently a professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include software engineering and web intelligence.