Specification-Guided Safety Verification for Feedforward Neural Networks

Weiming Xiang [*], Hoang-Dung Tran [*], Taylor T. Johnson [*]

March 22, 2019

Abstract

This paper presents a specification-guided safety verification method for feedforward neural networks with general activation functions. As such feedforward networks are memoryless, they can be abstractly represented as mathematical functions, and the reachability analysis of the neural network amounts to interval analysis problems. In the framework of interval analysis, a computationally efficient formula which can quickly compute the output interval sets of a neural network is developed. Then, a specification-guided reachability algorithm is developed. Specifically, the bisection process in the verification algorithm is completely guided by a given safety specification. Due to the employment of the safety specification, unnecessary computations are avoided and thus computational cost can be reduced significantly. Experiments show that the proposed method enjoys much more efficiency in safety verification with significantly less computational cost.

1 Introduction

Artificial neural networks have been widely used in machine learning systems. Though neural networks have been showing effectiveness and powerful ability in resolving complex problems, they are confined to systems which comply only to the lowest safety integrity levels since, in most of time, a neural network is viewed as a *black box* without effective methods to assure safety specifications for its outputs. Neural networks are trained over a finite number of input and output data, and are expected to be able to generalize to produce desirable outputs for given inputs even including previously unseen inputs. However, in many practical applications, the number of inputs is essentially infinite, this means it is impossible to check all the possible inputs only by performing experiments and moreover, it has been observed that neural networks can react in unexpected and incorrect ways to even slight perturbations of their inputs [1], which could result in unsafe systems. Hence, methods that are able to provide formal guarantees are in a great demand for verifying specifications or properties of neural networks. Verifying neural networks is a hard problem, even simple properties about them have been proven NP-complete problems [2]. The difficulties mainly come from the presence of activation functions and the complex structures, making neural networks large-scale, nonlinear, non-convex and thus incomprehensible to humans.

The importance of methods of formal guarantees for neural networks has been well-recognized in literature. There exist a number of results for verification of feedforward neural networks, especially for Rectifier Linear Unit (ReLU) neural networks, and a few results are devoted to neural networks with broad classes of activation functions. Motivated to general class of neural networks such as those considered in [3], our key contribution in this paper is to develop a specification-guided method for safety verification of feedforward neural network. First, we formulate the safety verification problem in the framework of interval arithmetic, and provide a computationally efficient formula to compute output interval sets. The developed formula is able to calculate the output intervals in a fast manner. Then, analogous to other state-of-the-art verification methods, such as counterexample-guided abstraction refinement (CEGAR) [4] and property directed reachability (PDR) [5], and inspired by the Moore-Skelboe algorithm [6], a specification-guided algorithm is developed. Briefly speaking, the safety specification is utilized to examine the existence of intersections between output intervals and unsafe regions and then determine the bisection actions in the verification algorithm. By making use of the information of safety specification, the computation cost can be reduced significantly. We provide experimental evidences to show the advantages of specification-guided approach, which shows that our approach only needs about 3%–7% computational cost of the method proposed in [3] to solve the same safety verification problem.

2 Related Work

Many recent works are focusing on ReLU neural networks. In [2], an SMT solver named Reluplex is proposed for a special class of neural networks with ReLU activation functions. The Reluplex extends the well-known Simplex algorithm from linear functions to ReLU functions by making use of the piecewise linear feature of ReLU functions. In [7], A layer-by-layer approach is developed for the output reachable set com-

[*]Authors are with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, Tennessee 37212, USA. Email: xiangwming@gmail.com (Weiming Xiang); Hoang-Dung Tran (trhoangdung@gmail.com); taylor.johnson@gmail.com (Taylor T. Johnson).

putation of ReLU neural networks. The computation is formulated in the form of a set of manipulations for a union of polyhedra. A verification engine for ReLU neural networks called AI^2 was proposed in [8]. In their approach, the authors abstract perturbed inputs and safety specifications as zonotopes, and reason about their behavior using operations for zonotopes. An Linear Programming (LP)-based method is proposed [9], and in [10] authors encoded the constraints of ReLU functions as a Mixed-Integer Linear Programming (MILP). Combining output specifications that are expressed in terms of LP, the verification problem for output set eventually turns to a feasibility problem of MILP. In [11, 12], an MILP based verification engine called Sherlock that performs an output range analysis of ReLU feedforward neural networks is proposed, in which a combined local and global search is developed to more efficiently solve MILP.

Besides the results for ReLU neural networks, there are a few other results for neural networks with general activation functions. In [13, 14], a piecewise-linearization of the nonlinear activation functions is used to reason about their behaviors. In this framework, the authors replace the activation functions with piecewise constant approximations and use the bounded model checker hybrid satisfiability (HySAT) [15] to analyze various properties. In their papers, the authors highlight the difficulty of scaling this technique and, currently, are only able to tackle small networks with at most 20 hidden nodes. In [16], the authors proposed a framework for verifying the safety of network image classification decisions by searching for adversarial examples within a specified region. A adaptive nested optimization framework is proposed for reachability problem of neural networks in [17]. In [3], a simulation-based approach was developed, which used a finite number of simulations/computations to estimate the reachable set of multi-layer neural networks in a general form. Despite this success, the approach lacks the ability to resolve the reachable set computation problem for neural networks that are large-scale, non-convex, and nonlinear. Still, simulation-based approaches, like the one developed in [3], present a plausibly practical and efficient way of reasoning about neural network behaviors. The critical step in improving simulation-based approaches is bridging the gap between finitely many simulations and the essentially infinite number of inputs that exist in the continuity set. Sometimes, the simulation-based approach requires a large number of simulations to obtain a tight reachable set estimation, which is computationally costly in practice. In this paper, our aim is to reduce the computational cost by avoiding unnecessary computations with the aid of a specification-guided method.

3 Background

3.1 Feedforward Neural Networks

Generally speaking, a neural network consists of a number of interconnected neurons and each neuron is a simple processing element that responds to the weighted inputs it received from other neurons. In this paper, we consider feed-forward neural networks, which generally consist of one input layer, multiple hidden layers and one output layer. The action of a neuron depends on its activation function, which is in the form of

$$y_i = \phi\left(\sum\nolimits_{j=1}^{n} \omega_{ij} x_j + \theta_i\right) \tag{1}$$

where x_j is the jth input of the ith neuron, ω_{ij} is the weight from the jth input to the ith neuron, θ_i is called the bias of the ith neuron, y_i is the output of the ith neuron, $\phi(\cdot)$ is the activation function. The activation function is generally a nonlinear continuous function describing the reaction of ith neuron with inputs x_j, $j = 1, \cdots, n$. Typical activation functions include ReLU, logistic, tanh, exponential linear unit, linear functions, for instance. In this work, our approach aims at being capable of dealing with activation functions regardless of their specific forms.

A feedforward neural network has multiple layers, and each layer ℓ, $1 \leq \ell \leq L$, has $n^{\{\ell\}}$ neurons. In particular, layer $\ell = 0$ is used to denote the input layer and $n^{\{0\}}$ stands for the number of inputs in the rest of this paper. For the layer ℓ, the corresponding input vector is denoted by $\mathbf{x}^{\{\ell\}}$ and the weight matrix is

$$\mathbf{W}^{\{\ell\}} = \left[\omega_1^{\{\ell\}}, \ldots, \omega_{n^{\{\ell\}}}^{\{\ell\}}\right]^\top \tag{2}$$

where $\omega_i^{\{\ell\}}$ is the weight vector. The bias vector for layer ℓ is

$$\theta^{\{\ell\}} = \left[\theta_1^{\{\ell\}}, \ldots, \theta_{n^{\{\ell\}}}^{\{\ell\}}\right]^\top. \tag{3}$$

The output vector of layer ℓ can be expressed as

$$\mathbf{y}^{\{\ell\}} = \phi_\ell(\mathbf{W}^{\{\ell\}}\mathbf{x}^{\{\ell\}} + \theta^{\{\ell\}}) \tag{4}$$

where $\phi_\ell(\cdot)$ is the activation function of layer ℓ.

The output of $\ell - 1$ layer is the input of ℓ layer, and the mapping from the input of input layer, that is $\mathbf{x}^{[0]}$, to the output of output layer, namely $\mathbf{y}^{[L]}$, stands for the input-output relation of the neural network, denoted by

$$\mathbf{y}^{\{L\}} = \Phi(\mathbf{x}^{\{0\}}) \tag{5}$$

where $\Phi(\cdot) \triangleq \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_1(\cdot)$.

3.2 Problem Formulation

We start by defining the neural network output set that will become of interest all through the rest of this paper.

Definition 3.1 *Given a feedforward neural network in the form of (5) and an input set $\mathcal{X} \subseteq \mathbb{R}^{n^{\{0\}}}$, the following set*

$$\mathcal{Y} = \left\{\mathbf{y}^{\{L\}} \in \mathbb{R}^{n^{\{L\}}} \mid \mathbf{y}^{\{L\}} = \Phi(\mathbf{x}^{\{0\}}),\ \mathbf{x}^{\{0\}} \in \mathcal{X}\right\} \tag{6}$$

is called the output set of neural network (5).

The safety specification of a neural network is expressed by a set defined in the output space, describing the safety requirement.

Definition 3.2 *Safety specification $\mathcal{S}$ formalizes the safety requirements for output $\mathbf{y}^{[L]}$ of neural network (5), and is a predicate over output $\mathbf{y}^{[L]}$ of neural network (5). The neural network (5) is safe if and only if the following condition is satisfied:*

$$\mathcal{Y} \cap \neg\mathcal{S} = \emptyset \tag{7}$$

where $\mathcal{Y}$ is the output set defined by (6), and $\neg$ is the symbol for logical negation.

The safety verification problem for the neural network (5) is stated as follows.

Problem 3.1 *How does one verify the safety requirement described by (7), given a neural network (5) with a compact input set $\mathcal{X}$ and a safety specification $\mathcal{S}$?*

The key for solving the safety verification Problem 3.1 is computing output set $\mathcal{Y}$. However, since neural networks are often nonlinear and non-convex, it is extremely difficult to compute the exact output set $\mathcal{Y}$. Rather than directly computing the exact output set for a neural network, a more practical and feasible way for safety verification is to derive an over-approximation of $\mathcal{Y}$.

Definition 3.3 *A set $\mathcal{Y}_o$ is an over-approximation of $\mathcal{Y}$ if $\mathcal{Y} \subseteq \mathcal{Y}_o$ holds.*

The following lemma implies that it is sufficient to use the over-approximated output set for the safety verification of a neural network.

Lemma 3.1 *Consider a neural network in the form of (5) and a safety specification $\mathcal{S}$, the neural network is safe if the following condition is satisfied*

$$\mathcal{Y}_o \cap \neg\mathcal{S} = \emptyset \tag{8}$$

where $\mathcal{Y} \subseteq \mathcal{Y}_o$.

Proof. Due to $\mathcal{Y} \subseteq \mathcal{Y}_o$, (8) implies $\mathcal{Y} \cap \neg\mathcal{S} = \emptyset$. □

From Lemma 3.1, the problem turns to how to construct an appropriate over-approximation $\mathcal{Y}_o$. One natural way, as the method developed in [3], is to find a set $\mathcal{Y}_o$ as small as possible to tightly over-approximate output set $\mathcal{Y}$ and further perform safety verification. However, this idea sometimes could be computationally expensive, and actually most of computations are unnecessary for safety verification. In the following, a specification-guided approach will be developed, and the over-approximation of output set is computed in an adaptive way with respect to a given safety specification.

4 Safety Verification

4.1 Preliminaries and Notation

Let $[x] = [\underline{x}, \overline{x}]$, $[y] = [\underline{y}, \overline{y}]$ be real compact intervals and $\circ$ be one of the basic operations addition, subtraction, multiplication and division, respectively, for real numbers, that is $\circ \in \{+, -, \cdot, /\}$, where it is assumed that $0 \notin [b]$ in case of division. We define these operations for intervals $[x]$ and $[y]$ by $[x] \circ [y] = \{x \circ y \mid x \in [y], x \in [y]\}$. The width of an interval $[x]$ is defined and denoted by $w([x]) = \overline{x} - \underline{x}$. The set of compact intervals in $\mathbb{R}$ is denoted by $\mathbb{IR}$. We say $[\phi] : \mathbb{IR} \to \mathbb{IR}$ is an interval extension of function $\phi : \mathbb{R} \to \mathbb{R}$, if for any degenerate interval arguments, $[\phi]$ agrees with ϕ such that $[\phi]([x, x]) = \phi(x)$. In order to consider multidimensional problems where $\mathbf{x} \in \mathbb{R}^n$ is taken into account, we denote $[\mathbf{x}] = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n] \in \mathbb{IR}^n$, where $\mathbb{IR}^n$ denotes the set of compact interval in $\mathbb{R}^n$. The width of an interval vector $\mathbf{x}$ is the largest of the widths of any of its component intervals $w([\mathbf{x}]) = \max_{i=1,\ldots,n}(\overline{x}_i - \underline{x}_i)$. A mapping $[\Phi] : \mathbb{IR}^n \to \mathbb{IR}^m$ denotes the interval extension of a function $\Phi : \mathbb{R}^n \to \mathbb{R}^m$. An interval extension is inclusion monotonic if, for any $[\mathbf{x}_1], [\mathbf{x}_2] \in \mathbb{IR}^n$, $[\mathbf{x}_1] \subseteq [\mathbf{x}_2]$ implies $[\Phi]([\mathbf{x}_1]) \subseteq [\Phi]([\mathbf{x}_2])$. A fundamental property of inclusion monotonic interval extensions is that $\mathbf{x} \in [\mathbf{x}] \Rightarrow \Phi(\mathbf{x}) \in [\Phi]([\mathbf{x}])$, which means the value of Φ is contained in the interval $[\Phi]([\mathbf{x}])$ for every $\mathbf{x}$ in $[\mathbf{x}]$.

Several useful definitions and lemmas are presented.

Definition 4.1 *[18] Piece-wise monotone functions, including exponential, logarithm, rational power, absolute value, and trigonometric functions, constitute the set of standard functions.*

Lemma 4.1 *[18] A function Φ which is composed by finitely many elementary operations $\{+, -, \cdot, /\}$ and standard functions is inclusion monotone.*

Definition 4.2 *[18] An interval extension $[\Phi]([\mathbf{x}])$ is said to be Lipschitz in $[\mathbf{x}_0]$ if there is a constant ξ such that $w([\Phi]([\mathbf{x}])) \leq \xi w([\mathbf{x}])$ for every $[\mathbf{x}] \subseteq [\mathbf{x}_0]$.*

Lemma 4.2 *[18] If a function $\Phi(\mathbf{x})$ satisfies an ordinary Lipschitz condition in $[\mathbf{x_0}]$,*

$$\|\Phi(\mathbf{x}_2) - \Phi(\mathbf{x}_1)\| \leq \xi \|\mathbf{x}_2 - \mathbf{x}_1\|, \ \mathbf{x}_1, \mathbf{x}_2 \in [\mathbf{x}_0] \tag{9}$$

then the interval extension $[\Phi]([\mathbf{x}])$ is a Lipschitz interval extension in $[\mathbf{x}_0]$,

$$w([\Phi]([\mathbf{x}])) \leq \xi w([\mathbf{x}]), \ [\mathbf{x}] \subseteq [\mathbf{x_0}]. \tag{10}$$

The following trivial assumption is given for activation functions.

Assumption 4.1 *The activation function ϕ considered in this paper is composed by finitely many elementary operations and standard functions.*

Based on Assumption 4.1, the following result can be obtained for a feedforward neural network.

Theorem 4.1 *The interval extension $[\Phi]$ of neural network Φ composed by activation functions satisfying Assumption 4.1 is inclusion monotonic and Lipschitz such that*

$$w([\Phi]([\mathbf{x}])) \leq \xi^L \prod_{\ell=1}^{L} \left\| \mathbf{W}^{\{\ell\}} \right\| w([\mathbf{x}]), \ [\mathbf{x}] \subseteq \mathbb{IR}^{n^{\{0\}}} \tag{11}$$

where ξ is a Lipschitz constant for all activation functions in Φ.

Proof. Under Assumption 4.1, the inclusion monotonicity can be obtained directly based on Lemma 4.1. Then, for the layer ℓ, we denote $\hat{\phi}_\ell(\mathbf{x}^{\{\ell\}}) = \phi_\ell(\mathbf{W}^{\{\ell\}}\mathbf{x}^{\{\ell\}} + \boldsymbol{\theta}^{\{\ell\}})$. For any $\mathbf{x}_1, \mathbf{x}_2$, it has

$$\begin{aligned}\left\|\hat{\phi}_\ell(\mathbf{x}_2^{\{\ell\}}) - \hat{\phi}_\ell(\mathbf{x}_1^{\{\ell\}})\right\| &\le \xi \left\|\mathbf{W}^{\{\ell\}}\mathbf{x}_2^{\{\ell\}} - \mathbf{W}\mathbf{x}_1^{\{\ell\}}\right\| \\ &\le \xi \left\|\mathbf{W}^{\{\ell\}}\right\| \left\|\mathbf{x}_2^{\{\ell\}} - \mathbf{x}_1^{\{\ell\}}\right\|.\end{aligned}$$

Due to $\mathbf{x}^{\{\ell\}} = \hat{\phi}_{\ell-1}(\mathbf{x}^{\{\ell-1\}})$, $\ell = 1, \ldots, L$, we have $\xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\|$ the Lipschitz constant for Φ, and (11) can be established by Lemma 4.2. □

4.2 Interval Analysis

First, we consider a single layer $\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \boldsymbol{\theta})$. Given an interval input $[\mathbf{x}]$, the interval extension is $[\phi](\mathbf{W}[\mathbf{x}] + \boldsymbol{\theta}) = [\underline{y}_1, \overline{y}_1] \times \cdots \times [\underline{y}_n, \overline{y}_n] = [\mathbf{y}]$, where

$$\underline{y}_i = \min_{\mathbf{x}\in[\mathbf{x}]} \phi\left(\sum\nolimits_{j=1}^{n} \omega_{ij}x_j + \theta_i\right) \tag{12}$$

$$\overline{y}_i = \max_{\mathbf{x}\in[\mathbf{x}]} \phi\left(\sum\nolimits_{j=1}^{n} \omega_{ij}x_j + \theta_i\right). \tag{13}$$

To compute the interval extension $[\phi]$, we need to compute the minimum and maximum values of the output of nonlinear function ϕ. For general nonlinear functions, the optimization problems are still challenging. Typical activation functions include ReLU, logistic, tanh, exponential linear unit, linear functions, for instance, satisfy the following monotonic assumption.

Assumption 4.2 *For any two scalars $z_1 \le z_2$, the activation function satisfies $\phi(z_1) \le \phi(z_2)$.*

Assumption 4.2 is a common property that can be satisfied by a variety of activation functions. For example, it is easy to verify that the most commonly used such as logistic, tanh, ReLU, all satisfy Assumption 4.2. Taking advantage of the monotonic property of ϕ, the interval extension $[\phi]([z]) = [\phi(\underline{z}), \phi(\overline{z})]$. Therefore, $\underline{y}_i$ and $\overline{y}_i$ in (12) and (13) can be explicitly written out as

$$\underline{y}_i = \sum\nolimits_{j=1}^{n} \underline{p}_{ij} + \theta_i \tag{14}$$

$$\overline{y}_i = \sum\nolimits_{j=1}^{n} \overline{p}_{ij} + \theta_i \tag{15}$$

with $\underline{p}_{ij}$ and $\overline{p}_{ij}$ defined by

$$\underline{p}_{ij} = \begin{cases} \omega_{ij}\underline{x}_j, & \omega_{ij} \ge 0 \\ \omega_{ij}\overline{x}_j, & \omega_{ij} < 0 \end{cases} \tag{16}$$

$$\overline{p}_{ij} = \begin{cases} \omega_{ij}\overline{x}_j, & \omega_{ij} \ge 0 \\ \omega_{ij}\underline{x}_j, & \omega_{ij} < 0 \end{cases}. \tag{17}$$

From (14)–(17), the output interval of a single layer can be efficiently computed with these explicit expressions. Then, we consider the feedforward neural network $\mathbf{y}^{\{L\}} = \Phi(\mathbf{x}^{\{0\}})$ with multiple layers, the interval extension $[\Phi]([\mathbf{x}^{\{0\}}])$ can be computed by the following layer-by-layer computation.

Theorem 4.2 *Consider feedforward neural network (5) with activation function satisfying Assumption 4.2 and an interval input $[\mathbf{x}^{\{0\}}]$, an interval extension can be determined by*

$$[\Phi]([\mathbf{x}^{\{0\}}]) = [\hat{\phi}_L] \circ \cdots \circ [\hat{\phi}_1] \circ [\hat{\phi}_0]([\mathbf{x}^{\{0\}}]) \tag{18}$$

where $[\hat{\phi}_\ell]([\mathbf{x}^{\{\ell\}}]) = [\phi_\ell](\mathbf{W}^{\{\ell\}}[\mathbf{x}^{\{\ell\}}] + \boldsymbol{\theta}^{\{\ell\}}) = [\mathbf{y}^{\{\ell\}}]$ in which

$$\underline{y}_i^{\{\ell\}} = \sum\nolimits_{j=1}^{n^{\{\ell\}}} \underline{p}_{ij}^{\{\ell\}} + \theta_i^{\{\ell\}} \tag{19}$$

$$\overline{y}_i^{\{\ell\}} = \sum\nolimits_{j=1}^{n^{\{\ell\}}} \overline{p}_{ij}^{\{\ell\}} + \theta_i^{\{\ell\}} \tag{20}$$

with $\underline{p}_{ij}^{\{\ell\}}$ and $\overline{p}_{ij}^{\{\ell\}}$ defined by

$$\underline{p}_{ij}^{\{\ell\}} = \begin{cases} \omega_{ij}^{\{\ell\}}\underline{x}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} \ge 0 \\ \omega_{ij}^{\{\ell\}}\overline{x}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} < 0 \end{cases} \tag{21}$$

$$\overline{p}_{ij}^{\{\ell\}} = \begin{cases} \omega_{ij}^{\{\ell\}}\overline{x}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} \ge 0 \\ \omega_{ij}^{\{\ell\}}\underline{x}_j^{\{\ell\}}, & \omega_{ij}^{\{\ell\}} < 0 \end{cases}. \tag{22}$$

Proof. We denote $\hat{\phi}_\ell(\mathbf{x}^{\{\ell\}}) = \phi_\ell(\mathbf{W}^{\{\ell\}}\mathbf{x}^{\{\ell\}} + \boldsymbol{\theta}^{\{\ell\}})$. For a feedforward neural network, it essentially has $\mathbf{x}^{\{\ell\}} = \hat{\phi}_{\ell-1}(\mathbf{x}^{\{\ell-1\}})$, $\ell = 1, \ldots, L$ which leads to (18). Then, for each layer, the interval extension $[\mathbf{y}^{\{\ell\}}]$ computed by (19)–(22) can be obtained directly from (14)–(17). □

We denote the set image for neural network Φ as follows

$$\Phi([\mathbf{x}^{\{0\}}]) = \{\Phi(\mathbf{x}^{\{0\}}) : \mathbf{x}^{\{0\}} \in [\mathbf{x}^{\{0\}}]\}. \tag{23}$$

Since $[\Phi]$ is inclusion monotonic according to Theorem 4.1, one has $\Phi([\mathbf{x}^{\{0\}}]) \subseteq [\Phi]([\mathbf{x}^{\{0\}}])$. Thus, it is sufficient to claim the neural network is safe if $[\Phi]([\mathbf{x}^{\{0\}}]) \cap \neg\mathcal{S} = \emptyset$ holds by Lemma 3.1.

According to the explicit expressions (18)–(22), the computation on interval extension $[\Phi]$ is fast. In the next step, we should discuss the conservativeness for the computation outcome of (18). We have $[\Phi]([\mathbf{x}^{\{0\}}]) = \Phi([\mathbf{x}^{\{0\}}]) + E([\mathbf{x}^{\{0\}}])$ for some interval-valued function $E([\mathbf{x}^{\{0\}}])$ with $w([\Phi]([\mathbf{x}^{\{0\}}])) = w(\Phi([\mathbf{x}^{\{0\}}])) + w(E([\mathbf{x}^{\{0\}}]))$.

Definition 4.3 *We call $w(E([\mathbf{x}^{\{0\}}])) = w([\Phi]([\mathbf{x}^{\{0\}}])) - w(\Phi([\mathbf{x}^{\{0\}}]))$ the excess width of interval extension of neural network $\Phi([\mathbf{x}^{\{0\}}])$.*

Explicitly, the excess width measures the conservativeness of interval extension $[\Phi]$ regarding its corresponding function Φ. The following theorem gives the upper bound of the excess width $w(E([\mathbf{x}^{\{0\}}]))$.

Theorem 4.3 *Consider feedforward neural network (5) with an interval input* $[\mathbf{x}^{\{0\}}]$*, the excess width* $w(E([\mathbf{x}^{\{0\}}]))$ *satisfies*

$$w(E([\mathbf{x}^{\{0\}}])) \leq \gamma w([\mathbf{x}^{\{0\}}]) \tag{24}$$

where $\gamma = \xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\|$.

Proof. We have $[\Phi]([\mathbf{x}^{\{0\}}]) = \Phi([\mathbf{x}^{\{0\}}]) + E([\mathbf{x}^{\{0\}}])$ for some $E([\mathbf{x}^{\{0\}}])$ and

$$\begin{aligned} w(E([\mathbf{x}^{\{0\}}])) &= w([\Phi]([\mathbf{x}^{\{0\}}])) - w(\Phi([\mathbf{x}^{\{0\}}])) \\ &\leq w([\Phi]([\mathbf{x}^{\{0\}}])) \\ &\leq \xi^L \prod_{\ell=1}^{L} \left\|\mathbf{W}^{\{\ell\}}\right\| w([\mathbf{x}^{\{0\}}]) \end{aligned}$$

which means (24) holds. □

Given a neural network Φ which means $\mathbf{W}^{\{\ell\}}$ and ξ are fixed, Theorem 4.3 implies that a less conservative result can be only obtained by reducing the width of input interval $[\mathbf{x}^{\{0\}}]$. On the other hand, a smaller $w([\mathbf{x}^{\{0\}}])$ means more subdivisions of an input interval which will bring more computational cost. Therefore, how to generate appropriate subdivisions of an input interval is the key for safety verification of neural networks in the framework of interval analysis. In the next section, an efficient specification-guided method is proposed to address this problem.

4.3 Specification-Guided Safety Verification

Inspired by the Moore-Skelboe algorithm [6], we propose a specification-guided algorithm, which generates fine subdivisions particularly with respect to specification, and also avoid unnecessary subdivisions on the input interval for safety verification, see Algorithm 1.

The implementation of the specification-guided algorithm shown in Algorithm 1 checks that the intersection between output set and unsafe region is empty, within a pre-defined tolerance ε. This is accomplished by dividing and checking the initial input interval into increasingly smaller sub-intervals.

- **Initialization.** Set a tolerance $\varepsilon > 0$. Since our approach is based on interval analysis, convert input set $\mathcal{X}$ to an interval $[\mathbf{x}]$ such that $\mathcal{X} \subseteq [\mathbf{x}]$. Compute the initial output interval $[\mathbf{y}] = [\Phi]([\mathbf{x}])$. Initialize set $\mathcal{M} = \{([\mathbf{x}], [\mathbf{y}])\}$.

- **Specification-guided bisection.** This is the key in the algorithm. Select an element $([\mathbf{x}], [\mathbf{y}])$ for specification-guided bisection. If the output interval $[\mathbf{y}]$ of sub-interval $[\mathbf{x}]$ has no intersection with the unsafe region, we can discard this sub-interval for the subsequent dividing and checking since it has been proven safe. Otherwise, the bisection action will be activated to produce finer subdivisions to be added to $\mathcal{M}$ for subsequent checking. The bisection process is guided by the given safety specification, since the activations of bisection actions are totally determined by the non-emptiness of the intersection between output interval sets and the given unsafe region. This distinguishing feature leads to finer subdivisions when the output set is getting close to the unsafe region, and on the other hand coarse subdivisions are sufficient for safety verification when the output set is far wary from the unsafe area. Therefore, unnecessary computational cost can be avoided. In the experiments section, it will be clearly observed how the bisection actions are guided by safety specification in a numeral example.

- **Termination.** The specification-guided bisection procedure continues until $\mathcal{M} = \emptyset$ which means all sub-intervals have been proven safe, or the width of subdivisions becomes less than the pre-defined tolerance ε which leads to an uncertain conclusion for the safety. Finally, when Algorithm 1 outputs an uncertain verification result, we can select a smaller tolerance ε to perform the safety verification.

Algorithm 1 Specification-Guided Safety Verification

Input: A feedforward neural network $\Phi : \mathbb{R}^{n^{\{0\}}} \to \mathbb{R}^{n^{\{L\}}}$, an input set $\mathcal{X} \subseteq \mathbb{R}^{n^{\{0\}}}$, a safety specification $\mathcal{S} \subseteq \mathbb{R}^{n^{\{L\}}}$, a tolerance $\varepsilon > 0$
Output: Safe or Uncertain

$\underline{x}_i \leftarrow \min_{\mathbf{x}\in\mathcal{X}}(x_i), \overline{x}_i \leftarrow \max_{\mathbf{x}\in\mathcal{X}}(x_i)$
$[\mathbf{x}] \leftarrow [\underline{x}_1, \overline{x}_1] \times \ldots, \times [\underline{x}_{n^{\{0\}}}, \overline{x}_{n^{\{0\}}}]$
$[\mathbf{y}] \leftarrow [\Phi]([\mathbf{x}])$
$\mathcal{M} \leftarrow \{([\mathbf{x}], [\mathbf{y}])\}$
while $\mathcal{M} \neq \emptyset$ **do**
  Select and remove an element $([\mathbf{x}], [\mathbf{y}])$ from $\mathcal{M}$
  if $[\mathbf{y}] \cap \neg\mathcal{S} = \emptyset$ **then**
    Continue
  else
    if $w([\mathbf{x}]) > \varepsilon$ **then**
      Bisect $[\mathbf{x}]$ to obtain $[\mathbf{x}_1]$ and $[\mathbf{x}_2]$
      for $i = 1 : 1 : 2$ **do**
        if $[\mathbf{x}_i] \cap \mathcal{X} \neq \emptyset$ **then**
          $[\mathbf{y}_i] \leftarrow [\Phi]([\mathbf{x}_i])$
          $\mathcal{M} \leftarrow \mathcal{M} \cup \{([\mathbf{x}_i], [\mathbf{y}_i])\}$
        end if
      end for
    else
      return Uncertain
    end if
  end if
end while
return Safe

5 Experiments

5.1 Random Neural Network

To demonstrate how the specification-guided idea works in safety verification, a neural network with two inputs and two outputs is proposed. The neural network has 5 hidden layers, and each layer contains 10 neurons. The weight matrices and

Table 1: Comparison on number of intervals and computational time to existing approach

	Intervals	Computational Time
Algorithm 1	4095	21.45 s
Xiang et al. 2018	111556	294.37 s

bias vectors are randomly generated. The input set is assumed to be $[\mathbf{x}^{\{0\}}] = [-5, 5] \times [-5, 5]$ and the unsafe region is $\neg\mathcal{S} = [1, \infty) \times [1, \infty)$.

We execute Algorithm 1 with termination parameter $\varepsilon = 0.01$, the safety can be guaranteed by partitioning $[\mathbf{x}^{\{0\}}]$ into 4095 interval sets. The specification-guided partition of the input space is shown in Figure 1. A non-uniform input space partition is generated based on the specification-guided scheme. An obvious specification-guided effect can be observed in Figure 1. The specification-guided method requires much less computational complexity compared to the approach in [3] which utilizes a uniform partition of input space, and a comparison is listed in Table 1. The computation is carried out using Matlab 2017 on a personal computer with Windows 7, Intel Core i5-4200U, 1.6GHz, 4 GB RAM. It can be seen that the number of interval sets and computational time have been significantly reduced to 3.67% and 7.28%, respectively, compared to those needed in [3]. Figure 2 illustrates the union of 4095 output interval sets, which has no intersection with the unsafe region, illustrating the safety specification is verified. Figure 2 shows that the output interval estimation is guided to be tight when it comes close to unsafe region, and when it is far way from the unsafe area, a coarse estimation is sufficient to verify safety.

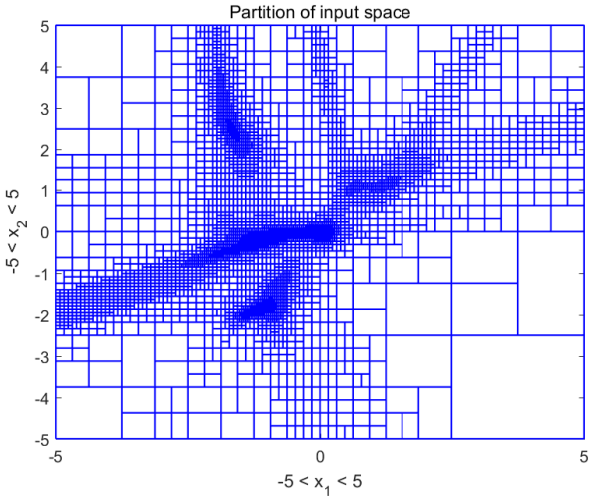

Figure 1: Specification-guided bisections of input interval by Algorithm 1. Guided by safety specification, finer partitions are generated when the output intervals are close to the unsafe region, and coarse partitions are generated when the output intervals are far wary.

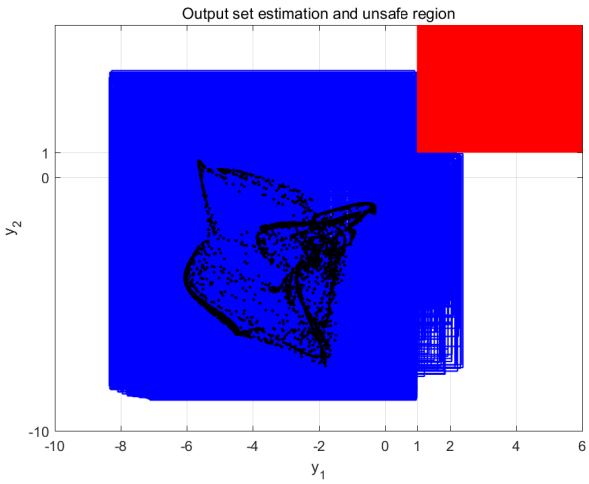

Figure 2: Output set estimation of neural networks. Blue boxes are output intervals, red area is unsafe region, black dots are 5000 random outputs.

5.2 Robotic Arm Model

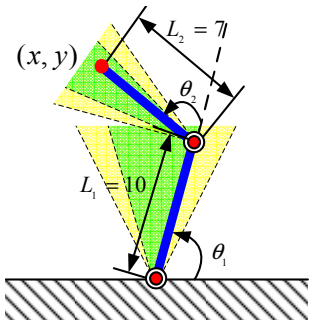

Figure 3: Robotic arm with two joints. The normal working zone of (θ_1, θ_2) is colored in green $\theta_1, \theta_2 \in [\frac{5\pi}{12}, \frac{7\pi}{12}]$. The buffering zone is in yellow $\theta_1, \theta_2 \in [\frac{\pi}{3}, \frac{5\pi}{12}] \cup [\frac{7\pi}{12}, \frac{2\pi}{3}]$. The forbidden zone is $\theta_1, \theta_2 \in [0, \frac{\pi}{3}] \cup [\frac{2\pi}{3}, 2\pi]$.

In [3], a *learning forward kinematics* of a robotic arm model with two joints is proposed, shown in Figure 3. The learning task is using a feedforward neural network to predict the position (x, y) of the end with knowing the joint angles (θ_1, θ_2). The input space $[0, 2\pi] \times [0, 2\pi]$ for (θ_1, θ_2) is classified into three zones for its operations: normal working zone $\theta_1, \theta_2 \in [\frac{5\pi}{12}, \frac{7\pi}{12}]$, buffering zone $\theta_1, \theta_2 \in [\frac{\pi}{3}, \frac{5\pi}{12}] \cup [\frac{7\pi}{12}, \frac{2\pi}{3}]$ and forbidden zone $\theta_1, \theta_2 \in [0, \frac{\pi}{3}] \cup [\frac{2\pi}{3}, 2\pi]$. The detailed formulation for this robotic arm model and neural network training can be found in [3].

The safety specification for the position (x, y) is $\mathcal{S} = \{(x, y) \mid -14 \leq x \leq 3 \text{ and } 1 \leq y \leq 17\}$. The input set of the robotic arm is the union of normal working and buffering zones, that is $(\theta_1, \theta_2) \in [\frac{\pi}{3}, \frac{2\pi}{3}] \times [\frac{\pi}{3}, \frac{2\pi}{3}]$. In the safety point of view, the neural network needs to be verified that all the outputs produced by the inputs in the normal working

zone and buffering zone will satisfy safety specification $\mathcal{S}$. In [3], a uniform partition for input space is used, and thus 729 intervals are produced to verify the safety property. Using our specification-guided approach, the safety can be guaranteed by partitioning the input space into only 15 intervals, see Figure 4 and Figure 5. Due to the small number of intervals involved in the verification process, the computational time is only 0.27 seconds for specification-guided approach.

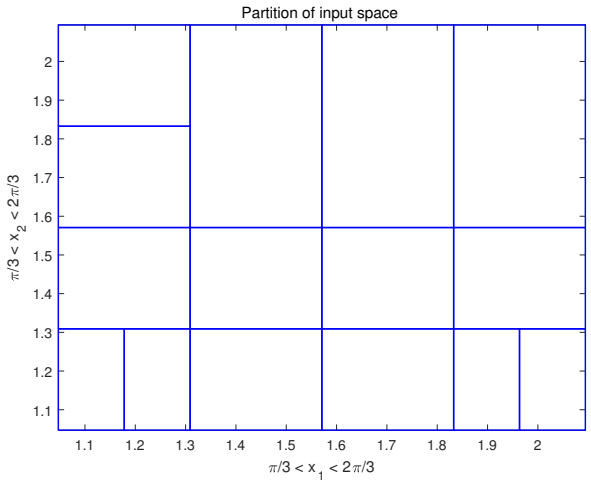

Figure 4: 15 sub-intervals for robotic arm safety verification.

5.3 Handwriting Image Recognition

In this handwriting image recognition task, we use 5000 training examples of handwritten digits which is a subset of the MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/), examples from the dataset are shown in Figure 6. Each training example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. We first train a neural network with 400 inputs, one hidden layer with 25 neurons and 10 output units corresponding to the 10 digits. The activation functions for both hidden and output layers are sigmoid functions. A trained neural network with about 97.5% accuracy is obtained.

Under adversarial perturbations, the neural network may produce a wrong prediction. For example in Figure 7(a) which is an image of digit 2, the label predicted by the neural network will turn to 1 as a 4×4 perturbation belonging to $[-0.5, 0.5]$ attacks the left-top corner of the image. With our developed verification method, we wish to prove that the neural network is robust to certain classes of perturbations, that is no perturbation belonging to those classes can alter the prediction of the neural network for a perturbed image. Since there exists one adversarial example for 4 perturbations at the left-top corner, it implies this image is not robust to this class of perturbation. We consider another class of perturbations, 3×3 perturbations at the left-top corner, see Figure 7(b). Using Algorithm 1, the neural network can be proved to be robust to all 3×3 perturbations located at at the left-top corner of the image, after 512 bisections.

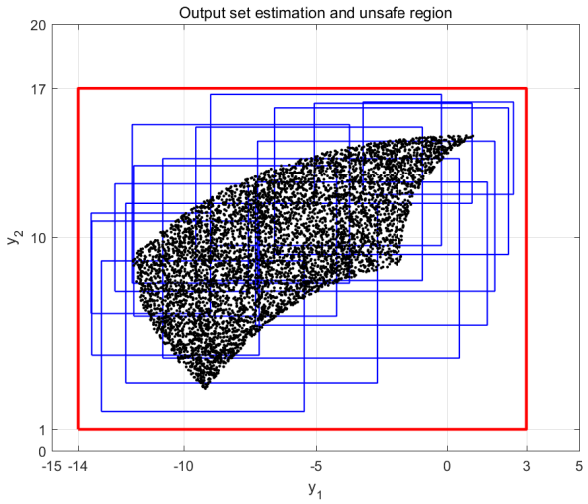

Figure 5: Safety verification for neural network of robotic arm. Blue boxes are output intervals, red box are boundary for unsafe region, black dots are 5000 random outputs. 15 output intervals are sufficient to prove the safety.

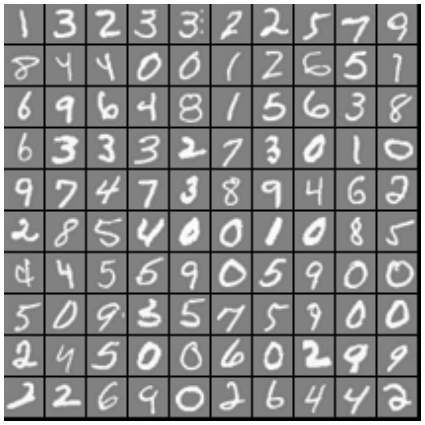

Figure 6: Examples from the MNIST handwritten digit dataset.

Moreover, applying Algorithm 1 to all 5000 images with 3×3 perturbations belonging to $[-0.5, 0.5]$ and located at the left-top corner, it can be verified that the neural network is robust to this class of perturbations for all images. This result means this class of perturbations will not affect the prediction accuracy of the neural network. The neural network is able to maintain its 97.5% accuracy even subject to any perturbations belonging to this class of 3×3 perturbations.

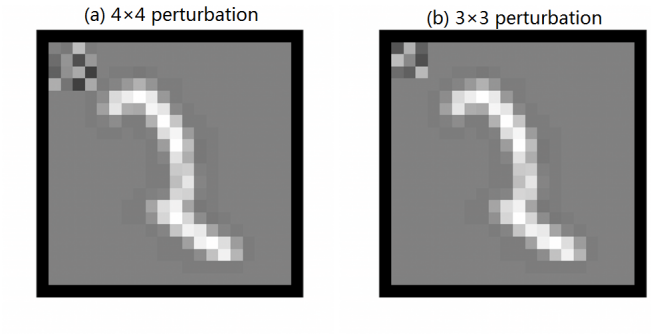

Figure 7: Perturbed image of digit 2 with perturbation in $[-0.5.0.5]$. (a) 4×4 perturbation at the left-top corner, the neural network will wrongly label it as digit 1. (b) 3×3 perturbation at the left-top corner, the neural network can be proved to be robust for this class of perturbations.

6 Conclusion and Future Work

In this paper, we introduce a specification-guided approach for safety verification of feedforward neural networks with general activation functions. By formulating the safety verification problem into the framework of interval analysis, a fast computation formula for calculating output intervals of feedforward neural networks is developed. Then, a safety verification algorithm which is called specification-guided is developed. The algorithm is specification-guided since the activation of bisection actions are totally determined by the existence of intersections between the computed output intervals and unsafe sets. This distinguishing feature makes the specification-guided approach be able to avoid unnecessary computations and significantly reduce the computational cost. Several experiments are proposed to show the advantages of our approach.

Though our approach is general in the sense that it is not tailored to specific activation functions, the specification-guided idea has potential to be further applied to other methods dealing with specific activation functions such as ReLU neural networks to enhance their scalability. Moreover, since our approach can compute the output intervals of a neural network, it can be incorporated with other reachable set estimation methods to compute the dynamical system models with neural network components inside such as extension of [3] to closed-loop systems [19] and neural network models of nonlinear dynamics [20].

References

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

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

[3] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multi-layer neural networks," *IEEE Transactions on Neural Network and Learning Systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[4] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *International Conference on Computer Aided Verification*, pp. 154–169, Springer, 2000.

[5] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pp. 125–134, FMCAD Inc, 2011.

[6] S. Skelboe, "Computation of rational interval functions," *BIT Numerical Mathematics*, vol. 14, no. 1, pp. 87–95, 1974.

[7] 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.

[8] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI^2: Safety and robustness certification of neural networks with abstract interpretation," in *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

[9] R. Ehlers, "Formal verification of piecewise linear feedforward neural networks," in *International Symposium on Automated Technology for Verification and Analysis*, pp. 269–286, Springer, 2017.

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

[11] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari, "Output range analysis for deep neural networks," *arXiv preprint arXiv:1709.09130*, 2017.

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

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

[14] L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Communications*, vol. 25, no. 2, pp. 117–135, 2012.

[15] M. Fränzle and C. Herde, "HySAT: An efficient proof engine for bounded model checking of hybrid systems," *Formal Methods in System Design*, vol. 30, no. 3, pp. 179–198, 2007.

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

[17] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," *arXiv preprint arXiv:1805.02242*, 2018.

[18] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*, vol. 110. Siam, 2009.

[19] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson, "Reachable set estimation and verification for neural network models of nonlinear dynamic systems," in *Safe, Autonomous and Intelligent Vehicles*, pp. 123–144, Springer, 2019.

[20] W. Xiang, H. Tran, J. A. Rosenfeld, and T. T. Johnson, "Reachable set estimation and safety verification for piecewise linear systems with neural network controllers," in *2018 Annual American Control Conference (ACC)*, pp. 1574–1579, June 2018.