Noname manuscript No.
(will be inserted by the editor)

Synthesis of ranking functions via DNN

Wang Tan [1,2] · Yi Li [2]

the date of receipt and acceptance should be inserted later

Abstract We propose a new approach to synthesis of non-polynomial ranking functions for loops via deep neural network(DNN). Firstly, we construct a ranking function template by DNN structure. And then the coefficients of the template can be learned by the train-set we construct to get a candidate ranking function. Finally, the candidate ranking function will be verified to show if it is a real ranking function. The experimental results show us that for some of loops from other work, we can find their ranking functions efficiently. Moreover, for some loops having multi-phase ranking functions obtained by existing methods, our method can directly detect their global ranking functions. Especially, our method can also detect the global ranking functions for some loops with transcendental terms.

Keywords Ranking function · DNN · Termination · Loop programs

1 Introduction

The analysis and verification of the termination of loop programs has become a hot topic in the field of computer programming research, and has attracted the attention of many scholars at home and abroad[12][17][28]. However, the termination verification of program itself is a very difficult problem, which is undecidable[7][27][26]. It is generally known that a standard method of proving loop termination is to find a ranking function. A function that can map a program state into an element of some well-founded ordered set, such that the value descends in the appropriate order whenever the loop completes an iteration, can be defined as a ranking function. Since descent in a well-founded set cannot be infinite, the existence of ranking functions implies the termination of loop programs.

Nowadays, there are some methods for synthesizing linear ranking functions and polynomial ranking functions. For loops with linear guards and linear updates, many methods[9][24][10][30][22] are established to generate their linear ranking functions. However, not all linear loops have a linear ranking function. Motivated by that, many researchers propose methods[3][2][6][4][5][18][20] to combine multiple linear ranking functions to capture more complex patterns. For loops with polynomial guards and polynomial updates, [8][11][25][29] develop some methods to generate polynomial ranking functions. Existing methods of synthesizing ranking functions usually predefine a linear or polynomial ranking functions template first. And then, different technologies such as linear programming(LP)[9][24][10][3], semidefinite programming (SDP)[11][25] or quantifier elimination (QE)[8] are used to get the values of the parameters in the template. For instance, in [8] the detection of polynomial ranking functions is reduced to the semi-algebraic systems solving. And in [29], the synthesis of polynomial ranking functions is reduced to the classification problem, where candidate ranking functions are obtained by Support-Vector Machines (SVM)[13][21] and then certified by *Z3*[23]. Nevertheless, for some loops, their ranking functions may be non-polynomial, such as transcendental ranking functions. Unfortunately, if such case occurs, *Z3* cannot verify if such a candidate ranking function containing transcendental terms is a real ranking function. Moreover, existing methods cannot deal with the loops containing transcendental terms.

In this paper, we propose a new approach to synthesize ranking functions via DNN, which extends the form of ranking functions from polynomial forms to non-polynomial forms. In other words, we obtain a candidate ranking function by DNN first, then verify if this function we gotten is a ranking function by a new method which

✉Yi Li
E-mail: liyi@cigit.ac.cn

1 University of Chinese Academy of Sciences, Beijing, 100049, China.
2 Chongqing Key Laboratory of Automated Reasoning and Cognition, Automated Reasoning and Cognition Center,Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, 400714, China.

is independent of Z3. It is easy to see that the ranking functions template defined by DNN is transcendental rather than polynomial. And the weights w_i and w_{ji} in such template can be regarded as the parameters. Thus, our goal is to obtain the values of the parameters w_i and w_{ji} by training.

The rest of this paper is structured as follows. Sect. 2 introduces some preliminaries on ranking functions and DNN. Sect. 3 introduces our approach to the synthesis of ranking functions by DNN. Sect. 4 gives a new method to verify whether or not a candidate ranking function is a ranking function. Sect. 5 demonstrates experimental results. Sect. 6 concludes.

2 Preliminaries

In this section we will introduce some basic notions on loop programs, ranking functions, and deep neural network.

2.1 Forms of loop programs

In this paper we mainly consider loops of the following forms:

$$while \bigwedge_{i=1}^{m} g_i(x) \leq 0 \ do \bigwedge_{i=1}^{n} x'_i = f_i(x) \tag{1}$$

where $x = (x_1, x_2, \cdots, x_n)^T \in R^n$ is an n-dimensional vector of variables, the conjunction of inequalities $\bigwedge_{i=1}^{m} g_i(x) \leq 0$ is guard condition over the variables, and $\bigwedge_{i=1}^{n} x'_i = f_i(x)$ is called the deterministic update, which means that for a given x satisfying the loop condition defined as above, there is only at most one x' satisfying the update constraint $\bigwedge_{i=1}^{n} x'_i = f_i(x)$, where x'_i denotes the new value of x_i after each loop iteration. Let $x' = ({x_1}', {x_2}', \cdots, {x_n}')^T$ and $f(x) = (f_1(x), f_2(x), \cdots, f_n(x))$. So, $x' = f(x)$. In this paper, we only focus on loops with deterministic updates defined as above.

Let

$$\Omega = \{x \in R^n : \bigwedge_{i=1}^{m} g_i(x) \leq 0 \ \}. \tag{2}$$

We take an example to illustrate the concepts mentioned above.

Example 1 *Consider the following loop:*

$$\begin{gathered} while \ \ {x_1}^2 + {x_2}^2 \leq 1 \quad do \\ {x_1}' = x_1 - {x_2}^2 + 1, {x_2}' = x_2 + {x_1}^2 - 1 \end{gathered}$$

In this loop, the guard condition is the inequality ${x_1}^2 + {x_2}^2 \leq 1$ and $\Omega = \{(x_1, x_2) \in R^2 : {x_1}^2 + {x_2}^2 \leq 1\}$. The update is $x'_1 = x_1 - {x_2}^2 + 1, x'_2 = x_2 + {x_1}^2 - 1$

2.2 Ranking functions

A loop program is terminating over the reals, if it terminates on all initial values of x over R^n . It is well known that the termination of loops is undecidable. The dominant method to proving termination is to synthesize ranking functions, since the existence of ranking functions implies termination.

Definition 1 *Given a loop program, let Ω be a set specified by its loop guards. A function $r(x)$: $R^n \to R$ is a ranking function for the loop, if the following two conditions are satisfied.*

Bounded. $\forall x \in \Omega \to r(x) \ \geq \ 0$

Decreasing. $\forall x \in \Omega \to r(x) - r(f(x)) \geq c^+$

Where $c^+ > 0$. The existence of a ranking function implies termination.

2.3 Deep Neural Network

Neural network work with functionalities similar to human brain. Deep-learning methods are representation-learning methods with multiple levels of representation. We can extract features from original data by simple but non-linear modules. As long as these modules are enough, even very complex models can be represented.[16]. And in this work, the deep neural network (DNN) we used can extract features from the train-set and detect complicated ranking functions.

The universal approximation theorem[14][15][19] states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R, under mild assumptions on the activation function. Therefore, we attempt to leverage the DNN techniques for training a ranking function for a loop program.

3 DNN-Based Synthesis of Ranking Functions

There are many methods to synthesize ranking functions, most of which focus on linear ranking functions and polynomial ranking functions. But there are few researches on non-polynomial ranking functions[21]. In this paper, we take a further step and propose a new method to find non-polynomial ranking functions by neural network.

In the section, we give a particular function $U(x)$ which maps Ω to $U(\Omega)$ firstly such that $U(\Omega)$ is bounded. Then, we build a neural network architecture and introduce a method to get sample points for constructing the train-set. we can leverage the train-set to detect a candidate ranking function by DNN finally.

The key step in our method is to get sample points. The existing sampling methods are random sampling, but when $U(\Omega)$ is not bounded, the sampling points may not be representative by random sampling. Therefore, we first introduce a function $U(x)$ to map Ω to the $U(\Omega)$, such that the image set $U(\Omega)$ is a bounded set. And then by Theorem 1, we reduce equivalently the synthesis problem of ranking functions over Ω to $U(\Omega)$. Since $U(\Omega)$ is bounded, we can construct the sample points set, such that the union of neighborhoods of all sample points covers $U(\Omega)$.

Let $U(x) = (m(x_1), m(x_2), \cdots m(x_n))$ be a mapping from Ω to $U(\Omega)$, where $m(x_i) = \frac{1}{1+e^{-x_i}}$ is a sigmoid function. Thus, we have the image set $U(\Omega)$ of Ω,

$$U(\Omega) = \{(u_1, u_2, \cdots, u_n) \in R^n : \bigwedge_{i=1}^{n} u_i = m(x_i), for\ all\ (x_1, x_2, \cdots, x_n) \in \Omega\}. \tag{3}$$

Since the sigmoid function $m(x_i)$ is bounded and continuous, $U(\Omega)$ is bounded. More importantly, since sigmoid function defined as $u_i = \frac{1}{1+e^{-x_i}}$ is invertible, which has inverse function $x_i = \ln(\frac{u_i}{1-u_i})$, $U(\Omega)$ can be rewritten as:

$$U(\Omega) = \{(u_1, u_2, \cdots, u_n) \in R^n : \bigwedge_{i=1}^{\mathrm{m}} g_i((\ln(\frac{u_1}{1-u_1}), \ln(\frac{u_2}{1-u_2}), \cdots, \ln(\frac{u_n}{1-u_n}))) \le 0\,\}\}. \tag{4}$$

It is easy to see that $U(\Omega) \subset (0,1)^n$, since the range of $u_i = \frac{1}{1+e^{-x_i}}$ is $(0,1)$.

Since $u_i = m(x_i)$, we have

$$u = (u_1, u_2, \cdots, u_n) = (m(x_1), m(x_2), \cdots, m(x_n)) = U(x) \tag{5}$$

$m(x_i)$ is a sigmoid function, which is inverse function.

Let $u_i{}' = m(x_i{}')$, so $x_i{}' = m^{-1}(u_i{}')$. Since $x_i = m^{-1}(u_i)$ and $x_i' = m^{-1}(u_i')$, we have

$$x = (x_1, x_2, \cdots, x_n) = (m^{-1}(u_1), m^{-1}(u_2), \cdots, m^{-1}(u_n)) = U^{-1}(u) \tag{6}$$

$$x' = (x_1', x_2', \cdots, x_n') = (m^{-1}(u_1'), m^{-1}(u_2'), \cdots, m^{-1}(u_n')) = U^{-1}(u'). \tag{7}$$

Since $x_i{}' = f_i(x)$ and $x' = f(x)$, we have

$$u' = (u_1{}', u_2{}', \cdots, u_n{}') = (m(x_1{}'), m(x_2{}'), \cdots, m(x_n{}')) = U(x') = U(f(x)) = U(f(U^{-1}(u))). \tag{8}$$

Example 2 *Consider further, the loop program defined by Example 1, we choose sigmoid function as component function of $U(x)$, i.e.,*

$$u_1 = m(x_1) = \frac{1}{1+e^{-x_1}}, u_2 = m(x_2) = \frac{1}{1+e^{-x_2}}$$
$$(u_1, u_2) = (m(x_1), m(x_2)) = (\frac{1}{1+e^{-x_1}}, \frac{1}{1+e^{-x_2}})$$

Theorem 1 *With the above notion, let $U(x)$ be defined as above. Then, there is a ranking function over Ω if and only if there is a ranking function over $U(\Omega)$.*

proof. Assume $R(x)$ is a ranking function over Ω. Then, $R(x)$ satisfies the following two conditions of ranking functions.

(a) $\forall x \in \Omega \Rightarrow R(x) \ \ge\ 0$.

(b) $\forall x \in \Omega \Rightarrow R(x) - R(f(x)) \ge c^+$, where c^+ is a positive number.

Consider (a), since $x = U^{-1}(u)$, we have

$$R(U^{-1}(u)) \ \ge\ 0 \Rightarrow R \circ U^{-1}(u) \ \ge\ 0,\ for\ all\ u \in U(\Omega). \tag{9}$$

Consider (b), since $x = U^{-1}(u)$, we have

$$R(U^{-1}(u)) - R(f(U^{-1}(u))) \ge c^+ \Rightarrow R \circ U^{-1}(u) - R \circ f \circ U^{-1}(u) \ge c^+ \Rightarrow R \circ U^{-1}(u) - R \circ U^{-1} \circ U \circ f \circ U^{-1}(u) \ge c^+. \tag{10}$$

Let $r = R \circ U^{-1}$, $\forall u \in U(\Omega)$, by (8), we know $u' = U \circ f \circ U^{-1}(u)$. So, we have

$$R \circ U^{-1}(u) - R \circ U^{-1} \circ U \circ f \circ U^{-1}(u) \ge c^+ \Rightarrow R \circ U^{-1}(u) - R \circ U^{-1}(u') \ge c^+. \tag{11}$$

Then, by (9) and (11), we get $r(u) \ge 0$ and $r(u) - r(u') \ge c^+$.

According to definition of ranking function, $r(u)$ is exactly a ranking function over $U(\Omega)$. Conversely, assume that $r(u)$ is a ranking function over $U(\Omega)$. Similar analysis can be applied to prove that there is a ranking function $R(x)$ over Ω, since $u = U(x)$ and u is invertible. We omit the details here. □

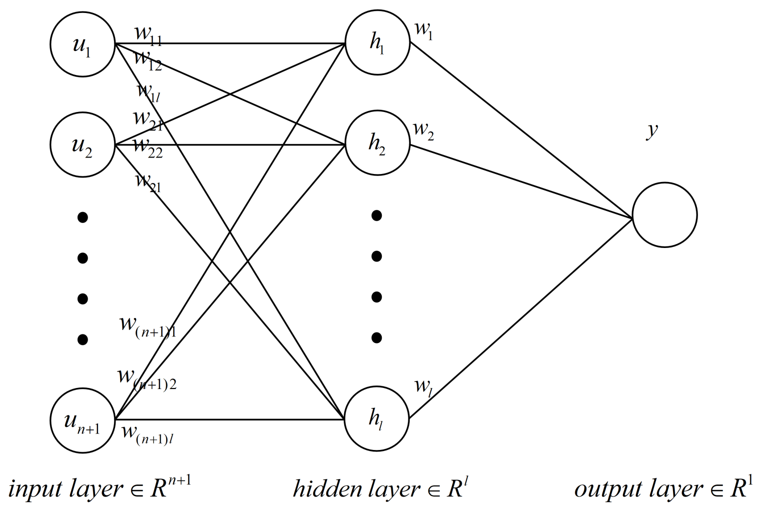

Fig. 1: Neural network structure

Remark 1 *If a given loop program has a ranking function $R(x)$ over Ω, then we can find the corresponding ranking function $r(u)$ over the image set $U(\Omega)$ of Ω, vice versa.*

Theorem 1 enables us to reduce equivalently the synthesis problem of ranking functions over Ω to $U(\Omega)$. In another words, we just need to leverage DNN to detect ranking functions over $U(\Omega)$. To do this, firstly, we give the neural network structure and then, we construct train-set over $U(\Omega)$. We get a candidate ranking function by training finally.

3.1 Model Building

The basic structure of neural network consists of input layer, hidden layer and output layer. It is noteworthy that different activation functions and different connection mode can be applied between adjacent network layers.

Since we need to construct ranking function template through neural network structure, in this paper, we choose full-connected.

The neural network model is related to the following terms:

Neurons. In this paper, the inputs of neural network consists of sample points and bias terms, so the number of neurons in the input layer is $n+1$, where n denotes the number of loop programs. Meanwhile, the number of the neuron in the output layer is one. Furthermore, the number of the neurons in the hidden layer can be adjusted in the experimental process. In our experiments, we usually set it to 3.

Activation function. As seen from the above, in order to make a ranking function template satisfy the ***bounded*** condition of ranking functions, the output layer should select a bounded function as its activation function. In our experiments, we set all activation functions in the neural network to be sigmoid functions.

Layers. The ranking function template depends on the number of layers of neural network. Therefore, we can adjust different layers to obtain different training effects. In this paper, we use three-layer network structure.

To sum up, one can construct a ranking function template $r(u)$ over $U(\Omega)$ by the DNN structure shown in Figure 1, which is as follows:

$$r(u) = \frac{1}{1+e^{-(\sum_{i=1}^{l} w_i \frac{1}{1+e^{-(\sum_{j=1}^{n+1} w_{ji}\cdot u_j)}})}}. \tag{12}$$

Note that $u = (u_1, u_2, \cdots, u_{n+1})$, where $u_{n+1} = 1$ is the bias term. In Figure 1, u_i is the input of neuron in the input layer and h_i is the output of neuron in the hidden layer, while y is the output of neuron in the output layer. w_i is the weight for linking hidden layer and output layer. And w_{ji} is the weight for linking hidden layer and input layer. Thus, finding a ranking function of loop programs is to derive the values of weights w_i and w_{ji} such that $r(u)$ satisfies the two conditions of ranking functions.

3.2 Sample points set

In this section, we will introduce how to construct sample points set. The following theorem tells us that $r(u)$ defined as in (12) is bounded.

Theorem 2 *With the above notion. Suppose the values of the parameters w_i and w_{ji} are given. Then, the function $r(u)$ must be bounded.*

Proof. Suppose the output layer structure is shown in Figure 2,

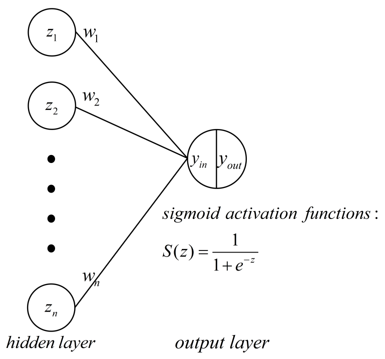

Fig. 2: Output layer structure

Note that $z_i, i \in (1, 2, \cdots, n)$ is the output of hidden layer, and $w_i, i \in (1, 2, \cdots, n)$ is the weight. y_{in} is the input of output layer, and y_{out} is the output of output layer. y_{out} can be calculated as follows:

$$\begin{aligned} y_{in} &= \sum_{i\in(1,n)} w_i z_i \\ y_{out} &= \frac{1}{1+e^{-(y_{in})}}. \end{aligned} \tag{13}$$

It is easy to see that y_{in} is the function of z_i, while y_{out} is the function of y_{in}. Owing to the range of the sigmoid function $S(z) = \frac{1}{1+e^{-z}}$ is $(0, 1)$ for all $z \in (-\infty, +\infty)$. T hus $y_{out} \in (0, 1)$, hence the function obtained by DNN must be bounded.□

Remark 2 *Theorem 2 tells us that the* **bounded** *condition can be satisfied naturally when the activation function of output layer is a sigmoid function.*

In what follows, Owing to the ***bounded*** condition is naturally satisfied by $r(u)$, we next just need to consider the ***decreasing*** condition. That is, we want to obtain the value w_i and w_{ji}, such that $\forall u \in U(\Omega) \Rightarrow r(u) - r(u') \geq c^+$ for a certain positive number c^+. Since $u' = U \circ f(U^{-1}(u))$, the above formula can also be written as:

$$\forall u \in U(\Omega) \Rightarrow r(u) - r(U \circ f(U^{-1}(u))) \geq c^+. \tag{14}$$

A natural technique is to use quantifier elimination (QE) to eliminate u from (14). However, since $r(u)$ is non-polynomial, QE-based technique cannot be applied to deal with (14). Thus, in order to get the value of w_i and w_{ji} to guarantee that $r(u) - r(U \circ f(U^{-1}(u))) \geq c^+$ holds on all u in $U(\Omega)$, we propose a method to construct a finite points set N_{sample} from $U(\Omega)$, such that N_{sample} has the following properties:

a. The union of neighborhoods of all points in N_{sample} can cover $U(\Omega)$.

b. For any a point $\alpha \in N_{sample}$, if $p(\alpha) \geq \varepsilon > 0$, then for $\forall \theta \in \mathrm{O}(\alpha, \delta)$, we have $p(\theta) \geq c^+$, where $\mathrm{O}(\alpha, \delta)$ is a neighborhood centered at α with radius δ.

In the following, we will show how to construct N_{sample}. Since $U(\Omega) \subseteq (0, 1)^n$, we divide $(0, 1)^n$ into a finite number of hypercubes with the same size d by gridding technology. Here, how to choose step d will be discussed in Theorem 3.

The following Theorem 3 will show that under a certain condition, for given a point α in N_{sample}, if it satisfies $p(\alpha) \geq \varepsilon$, then for any point θ in its neighborhood $\mathrm{O}(\alpha, \sqrt{n}d)$, we have $p(\theta) \geq \varepsilon - ndc$.

Let $\mathcal{H}$ be a convex set and $\Omega \subseteq \mathcal{H}$. Thus, clearly, the $\mathcal{U}(\mathcal{H})$ contains $U(\Omega)$, since $U(x)$ is invertible.

Theorem 3 *Let $p(u) = r(u) - r(U \circ f(U^{-1}(u)))$. Suppose that the upper bound in a hyperrectangle $\mathcal{U}(\mathcal{H})$ of $|\bigtriangledown p(u)|$ is c, for a given point α, if $p(\alpha) \geq \varepsilon > 0$, then for $\forall \theta \in \mathrm{O}(\alpha, \sqrt{n}d)$, we have $p(\theta) \geq \varepsilon - ndc$, where p is a n-variate function, and $\mathrm{O}(\alpha, \sqrt{n}d)$ is a neighborhood of radius $\sqrt{n}d$ with α as its center.*

Proof. Let $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ and $\theta = (\theta_1, \theta_2, \cdots, \theta_n)$.

Suppose there is a point $\theta \in \mathrm{O}(\alpha, \sqrt{n}d)$, such that $p(\theta) < \varepsilon - ndc$.

Since $p(u)$ is continuous over R^n, according to the Lagrange mean value theorem, we get

$$p(\alpha) - p(\theta) = \bigtriangledown p(\beta)(\alpha_1 - \theta_1, \alpha_2 - \theta_2, \cdots, \alpha_n - \theta_n)^T \tag{15}$$

where $\beta = (\theta + t \cdot (\alpha - \theta)) \in \mathcal{U}(\mathcal{H}), t \in (0, 1)$.

Thus, by (15), we have

$$|p(\alpha)-p(\theta)| = |\bigtriangledown p(\beta)(\alpha_1-\theta_1,\alpha_2-\theta_2,\cdots,\alpha_n-\theta_n)^T|. \tag{16}$$

Since $p(\theta)<\varepsilon-ndc$ and $p(\alpha)\geq\varepsilon$, it follows that

$$|\mathrm{p}(\alpha)-\mathrm{p}(\theta)| \ \geq p(\alpha)-p(\theta)\geq\varepsilon-p(\theta)>\varepsilon-(\varepsilon-ndc)>ndc. \tag{17}$$

In addition, since

$$|\bigtriangledown p(\beta)(\alpha_1-\theta_1,\alpha_2-\theta_2,\cdots,\alpha_n-\theta_n)^T| \ \leq\ |\bigtriangledown p(\beta)|\cdot|(\alpha_1-\theta_1,\alpha_2-\theta_2,\cdots,\alpha_n-\theta_n)^T| \tag{18}$$

and

$$\begin{aligned}|(\alpha_1-\theta_1,\alpha_2-\theta_2,\cdots,\alpha_n-\theta_n)^T| &= \sqrt{(\alpha_1-\theta_1)^2+(\alpha_2-\theta_2)^2+\cdots+(\alpha_n-\theta_n)^2}\\ &\leq\sqrt{(\sqrt{\mathrm{n}}\mathrm{d})^2+(\sqrt{\mathrm{n}}\mathrm{d})^2+\cdots+(\sqrt{\mathrm{n}}\mathrm{d})^2}=nd,\end{aligned} \tag{19}$$

we have

$$|\bigtriangledown p(\beta)(\alpha_1-\theta_1,\alpha_2-\theta_2,\cdots,\alpha_n-\theta_n)^T| \ \leq|\bigtriangledown \mathrm{p}(\beta)|\cdot nd. \tag{20}$$

By (16), (17) and (20), since

$$|\bigtriangledown \mathrm{p}(\beta)|\cdot nd\geq|\bigtriangledown p(\beta)(\alpha_1-\theta_1,\alpha_2-\theta_2,\cdots,\alpha_n-\theta_n)^T|=|\mathrm{p}(\alpha)-\mathrm{p}(\theta)|\ >ndc, \tag{21}$$

we get

$$|\bigtriangledown \mathrm{p}(\beta)|>c. \tag{22}$$

Clearly, (22) contradicts the assumption $|\bigtriangledown p(u)|\leq c$ for all $u\in\mathcal{U}(\mathcal{H})$. Therefore, $\forall\theta\in \mathrm{O}(\alpha,\sqrt{n}d)$, we have $p(\theta)\geq\varepsilon-ndc$. □

Remark 3 *Since $p=r(u)-r(U\circ f(U^{-1}(u)))$, we can see that if $|\bigtriangledown(r(u)-r(U\circ f(U^{-1}(u))))|$ have a upper bound c and a point α in N_{sample} satisfy $r(\alpha)-r(U\circ f(U^{-1}(\alpha)))\geq\varepsilon$, then for any point θ in the neighborhood $\mathrm{O}(\alpha,\sqrt{n}d)$, we have $r(\theta)-r(U\circ f(U^{-1}(\theta)))\geq\varepsilon-ndc$.*

According to the ***decreasing*** condition of ranking functions, which requires $p=r(u)-r(U\circ f(U^{-1}(u)))\geq c^+$. We need to ensure that the $\varepsilon-ndc$ is a positive number. Because $r(u)$ is a template with the parameters w_i and w_{ji}, c is a function of w_i and w_{ji}. At same times, since w_i and w_{ji} are unknowns, we need to assume a value c_{assume} of c in advance. With above assumption, we set

$$d=\frac{\varepsilon}{n\cdot c_{assume}} \tag{23}$$

while d defined by (23) and $c<c_{assume}$, the vlue of $\varepsilon-ndc$ must be a positive number.

After we get the value of step d, next we need to construct N_{sample}. By Theorem 1, we just need to find ranking functions over $U(\Omega)$ by DNN. Thus, we need to give a method to get the sample points N_{sample} over $U(\Omega)$ firstly. Most importantly, we need to make sure that the union of the neighborhoods of each sample point in sample points set can cover $U(\Omega)$. Next, we illustrate how to construct sample points set N_{sample} such that the union of neighborhoods of all points in N_{sample} can cover $U(\Omega)$

Intuitively, given step d by (23), the $(0,1)^n$ can be divided into a finite number of small hypercubes with the length d by gridding technology. Since $U(\Omega)\subseteq(0,1)^n$ is bounded, the $U(\Omega)$ is clearly covered by the collection H of some hypercubes. It is not difficult to see that in any hypercube Q, since the longest distance between any two points in Q is $\sqrt{n}d$, thus for any a point q in Q, the neighborhood $\mathrm{O}(q,\sqrt{n}d)$ of radius $\sqrt{n}d$ centered at q must cover the hypercube Q. So if we arbitrarily take a sample point from each hypercube in the collection H, so then union of the neighborhoods of these sample points can cover H. And Since H can cover $U(\Omega)$, thus the union of the neighborhoods of our sample points can cover $U(\Omega)$.

From the above, it is easy to see that the union of the neighborhoods of sample points indeed can cover $U(\Omega)$, if we can take at least one sample point from each hypercube. Next, we will describe a method to construct N_{sample}.

Let $N_{sample}=N_{sample\_inside}+N_{sample\_bound}$. Note that N_{sample} denotes the set of the sample points, while $N_{sample\_\mathrm{inside}}$ denotes the set of sample points in $U(\Omega)$ and N_{sample_bound} denotes the set of sample points on the boundary of $U(\Omega)$. That is N_{sample} can be divided into two parts. By gridding technology, we can get the collection H of hypercubes, which covers $U(\Omega)$. Suppose $U(\Omega)$ is shown in (1) of Figure 3.

The N_{sample_inside} is the set of the lattice points inside $U(\Omega)$, which is shown in (2) of Figure 3.

The N_{sample_bound} is the set of intersection points between the boundary of $U(\Omega)$ and the boundary of hypercubes, which is shown in (2) of Figure 3.

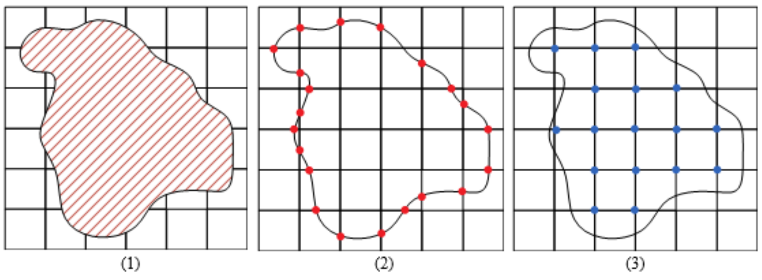

Fig. 3: Sampling process

The procedure to construct the sample points set N_{sample} is presented in Algorithm 1. Note that n is number of program variables.

Algorithm 1 Construction of the sample points set N_{sample}

Input: the c_{assume}, the ε and the number n.
Output: return sample points set N_{sample}.

```
1: Let $D = [\,]$ be an empty list;
2: Let $N_{sample_{inside}} = [\,]$ be an empty list;
3: Let $N_{sample_{bound}} = [\,]$ be an empty list;
4: $d = \frac{\varepsilon}{n \cdot c_{assume}}$;        //calculate the step $d$
5: function HYPERCUBE($n$,$d$)
6:     if n==1 then
7:         $i = d$;
8:         while $i \leq 1 - d$ do
9:             append $[i]$ to $D$;
10:            $i+ = d$;
11:        end while
12:    else $n > 1$
13:        for each $j$ in $hypercube(n-1, d)$ do
14:            $i = d$;
15:            while $i \leq 1 - d$ do
16:                append $[i]$ to $j$;
17:                append $[j]$ to $D$;
18:                $i+ = d$;
19:            end while
20:        end for
21:    end if
22:    return $D$;
23: end function
24: $hypercube(n)$;
25: for each $u$ in $D$ do        //get the $N_{sample\_inside}$
26:     if $u$ in $U(\Omega)$ then
27:         append $u$ to $N_{sample\_inside}$
28:     end if
29: end for
30: for each $u$ in $D$ do        //get the $N_{sample\_bound}$
31:     for $i := 1$ to $n$ do
32:         Delete $u_i$ in $u$;
33:         let $k = ln(\frac{u_i}{1-u_i})$ is the value to be solves;
34:         for $j := 1$ to $m$ do        // solve the intersections between the boundary of $U(\Omega)$ with hypercube
35:             $res = Solve(g_j((ln(\frac{u_1}{1-u_1})), (ln(\frac{u_2}{1-u_2})), \cdots, k, \cdots, (ln(\frac{u_n}{1-u_n}))) \leq 0)$
36:             for each $k$ in $res$ do        //tell the res whether is in $U(\Omega)$
37:                 if $\bigwedge_{k=1, k\neq i}^{m} g_k(g_j((ln(\frac{u_1}{1-u_1})), (ln(\frac{u_2}{1-u_2})), \cdots, k, \cdots, (ln(\frac{u_n}{1-u_n}))) \leq 0)$ then
38:                     append $(u_1, u_2, \cdots, k, \cdots, u_n)$ to $N_{sample\_bound}$;
39:                 end if
40:             end for
```

```
41:         end for
42:     end for
43: end for
44: append $N_{sample\_inside}$ $N_{sample\_bound}$ to $N_{sample}$
45: return $N_{sample}$
```

Remark 4 *Given the value of d, one can construct N_{sample} by Algorithm 1.*

The Theorem 4 next will show that the union of neighborhoods of each points in N_{sample} can cover $U(\Omega)$, if $U(\Omega)$ is a single connected region and N_{sample} obtained by Algorithm 1 is not empty.

Since the value of d depends on c_{assume} and c_{assume} is given in advance, so when $U(\Omega)$ or one of connected branches of $U(\Omega)$ is the set in Figure 4. Algorithm 1 may not find N_{sample} from the s.

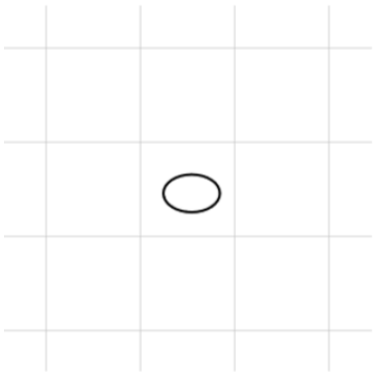

Fig. 4: The particular set s

When $U(\Omega)$ a simply connected region, and the sample point set N_{sample} is not an empty set, we exclude this special case. So the N_{sample} can cover the $U(\Omega)$.

Theorem 4 *Suppose $U(\Omega)$ is a simply connected region and $N_{sample} \neq \emptyset$. Let $U(\Omega)$ be the image set under $U(x)$ of Ω. Then, for any a point p in $U(\Omega)$, there exists q in N_{sample}, such that $|p-q| \leq \sqrt{n}d$.*

Proof. For convenience, we next just consider the case when $n = 2$, and the similar analysis is applicable for the general case. For $n = 2$, W.L.O.G, $U(\Omega)$ is an irregular region shown in (1) of Figure 5. The N_{sample} is shown in (2) of Figure 5, where the red dots denote the points in N_{sample_bound} and the blue dots denote the points in N_{sample_inside}. Then, from (5) in Figure 5, we can see that the $U(\Omega)$ can be covered by a finite number of small squares with the length d. Let H be the collection of finitely many squares. Clearly, we have $U(\Omega) \subseteq H$. The H can be divided into two parts: $H_{boundary}$ and H_{inside}, where, $H_{boundary}$ denotes the set of squares which intersect with the boundary of $U(\Omega)$, and H_{inside} denotes the set of squares which are completely contained in $U(\Omega)$. $H_{boundary}$ and H_{inside} are shown in (3) and (4) of Figure 5, respectively. And each square at least has one sample point in N_{sample}, thus, the union of neighborhoods of all points in N_{sample} can cover these squares H. Since the H covers the $U(\Omega)$, the union of neighborhoods of all points in N_{sample} can cover $U(\Omega)$.□

Remark 5 *Theorem 4 proves that the union $\bigcup\limits_{q \in N_{sample}} \mathrm{O}(q, \sqrt{n}d)$ of neighborhoods of each point in N_{sample} can cover $U(\Omega)$, if $U(\Omega)$ is a single connected region.*

To sum up, by Theorem 4 and Algorithm 1, we can find the points set N_{sample} such that the union of the neighborhoods of each point in N_{sample} can cover $U(\Omega)$. In what follows, we propose a method to build the train-set of neural network according to the ***decreasing*** condition of ranking functions.

3.3 Train-set

To build the train-set, we need to build the inputs and outputs of neural network.

3.3.1 The inputs of neural work

According to the ***decreasing*** condition $r(u) - r(u') \geq c^+$ of ranking function, thus we need not only the value of $r(u)$, but also the value of $r(u')$, such that $r(u) - r(u') \geq c^+$. At the above steps, we have already got the

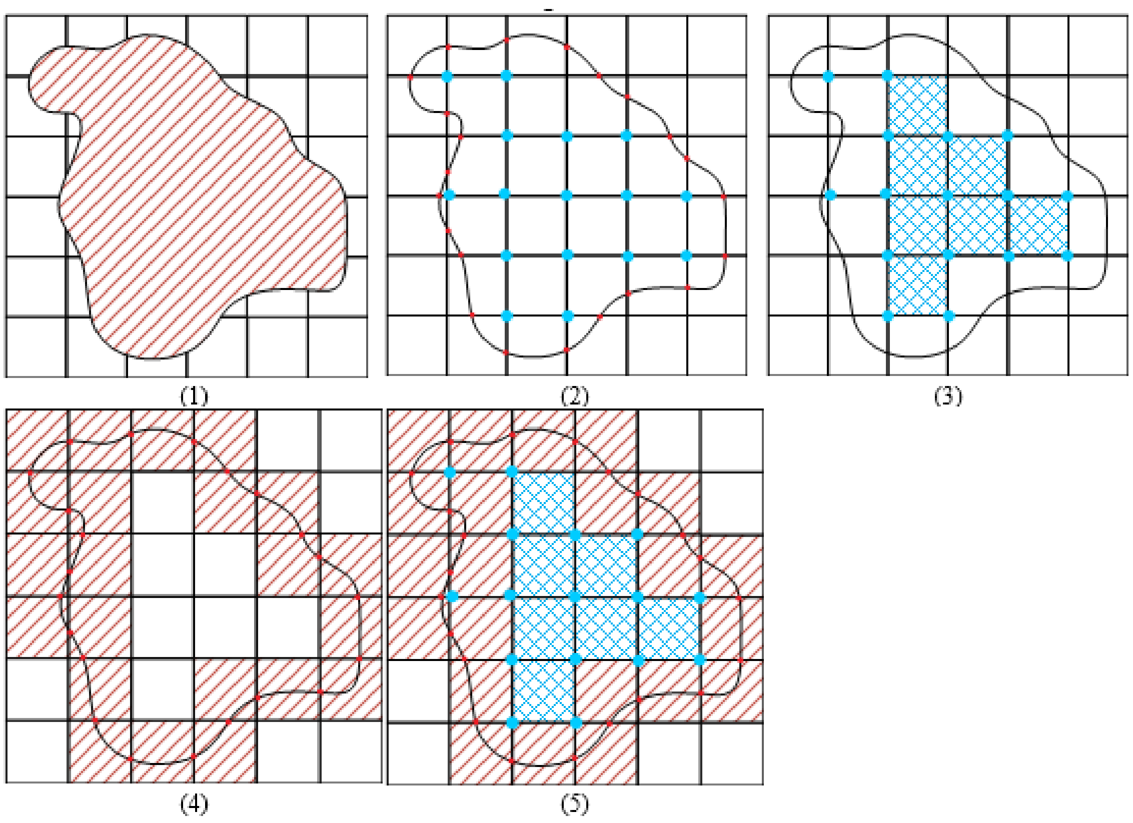

Fig. 5: Coverage processes

sample points set N_{sample} for u by Algorithm 1. Thus, since u is taken from N_{sample} and $u' = U \circ f(U^{-1}(u))$, we can get another sample points set N'_{sample} about u'.

Let

$$\overline{N}_{sample} = N_{sample} \cup N'_{sample} \tag{24}$$

is the sample points set for DNN. Note that $N_{sample} = (u, 1)$ and $N'_{sample} = (u', 1)$, where 1 is the bias term.

3.3.2 The outputs of neural network

At the above steps, we get the $\overline{N}_{sample}$, which is the inputs of neural network. Next we need to determine an outputs for each sample point in $\overline{N}_{sample}$ according to the ***decreasing*** condition.

According to the ***decreasing*** condition of ranking functions, we construct train-set $D_{train} = \{(u, 1, r(u))\} \cup \{(u', 1, r(u'))\}$, such that $r(u) - r(u') > 0$. For convenience, label output λ for u and another output λ' for u'. To guarantee that $r(u) - r(u') > 0$, we set $\lambda \in (n_1, 1)$ and $\lambda' = \lambda - l$, where $l \in (n_2, n_1)$ and $1 \geq n_1 > n_2 \geq 0$. Here n_1, n_2 are experimental tunable parameters.

To summarize, the train-set D_{train} consists of inputs and outputs, as follows:

Table 1: train-set D_{train}

inputs		outputs
$(u, 1)$	by DNN	λ
$(u', 1)$		λ'

Note that 1 is the bias term and the datas $(u, 1, \lambda)$, $(u', 1, \lambda')$ satisfy: $r(u) - r(u') = \lambda - \lambda' > 0$.

The procedure to construct train-set $D_{train} = \{(z, 1, r(z)), z \in \overline{N}_{sample}\}$ is embedded in Algorithm 2. By this means, a candidate ranking function can be trained by DNN.

algorithm 2 Construction of a train-set D_{train}

```
Input: the sample points set $N_{sample}$, the guard condition,the update function $x_i' = f_i(x)$, the positive number $n_1$ and $n_2$.
Output: return $D_{train}$
 1: $D_{train} = [\,]$;
 2: for each $u$ in $N_{sample}$ do
 3:     append 1 to u; // add the bias term
 4:     $\lambda = random(n_1, 1)$; // label the output $\lambda$ for $u$
 5:     append $[\lambda]$ to $u$;
 6:     append $u$ to $D_{train}$;
 7:     $u_i' = \frac{1}{1+e^{f_i(ln(\frac{u_1}{1-u_1}),ln(\frac{u_2}{1-u_2}),\cdots,ln(\frac{u_n}{1-u_n}))}}$; // calculate $u'$
 8:     append 1 to $u'$;
 9:     $l = random(n_2, n_1)$;
10:     $\lambda' = \lambda - l$; //label output $\lambda'$ for $u'$
11:     append $[\lambda']$ to $u'$;
12:     append $u'$ to $D_{train}$;
13: end for
14: return $D_{train}$
```

Note that $n_1 \in [0, 1]$ and $n_2 \in [0, n_1]$. In most of our experiments, we take $n_1 = 0.5$ and $n_2 = 0$. However, in some cases, we may adjust the n_1 and n_2 to achieve the best training effect.

Example 3 *Let Ω be a set defined in Example 1 and let $U(x) = (m(x_1), m(x_2))$ be a mapping defined in Example 2. That is,*

$$
\begin{aligned}
&u_1 = \tfrac{1}{1+e^{-x_1}}, u_2 = \tfrac{1}{1+e^{-x_2}}\\
&u_1{}' = \tfrac{1}{1+e^{-x_1{}'}} = \tfrac{1}{1+e^{-(x_1-x_2{}^2+1)}}, u_2{}' = \tfrac{1}{1+e^{-x_2{}'}} = \tfrac{1}{1+e^{-(x_2+x_1{}^2-1)}}
\end{aligned}
$$

So we have the image set $U(\Omega)$ under $U(x)$ of Ω:

$$U(\Omega) = \{(u_1, u_2) \in R^2 : (\ln(\frac{u_1}{1-u_1}))^2 + (\ln(\frac{u_2}{1-u_2}))^2 \le 1\}$$

Since this loop only has two variables, the $n = 2$. And we set $c_{assume} = 2.5$ and $\lambda = 0.01$.
Then, the $d = \frac{\lambda}{n \cdot c_{assume}} = 0.002$.
Besides, we set $n_1 = 0.5$ and $n_2 = 0.2$. The train-set D_{train} is as follows:

Table 2: the output of D_{train}

	0.27333333	0.27333333	0.28	$\cdots$	0.73	0.73	0.73
$(u_1, u_2, 1)$	0.44777785	0.55222215	0.418576	$\cdots$	0.52	0.522	0.524
	1	1	1	$\cdots$	1	1	1
	0.49456765	0.49456765	0.48688954	$\cdots$	0.87955424	0.87941145	0.87925482
$(u_1', u_2', 1)$	0.43693574	0.54133016	0.39254807	$\cdots$	0.51732226	0.5193231	0.52132403
	1	1	1	$\cdots$	1	1	1
λ	0.9359949	0.54966248	0.9203678	$\cdots$	0.7432145	0.91482292	0.86196221
λ'	0.61006867	0.21434009	0.49003298	$\cdots$	0.35222001	0.54831943	0.43670077

3.4 test-set

From Theorem 4, we know that the N_{sample} obtained by Algorithm 1 can cover $U(\Omega)$, when $U(\Omega)$ is a simply connected region. Thus, in order to verify if each point $(u, u') \in N_{sample} \times N'_{sample}$ satisfies $r(u) - r(u') \ge c^+$, then the test-set is $N_{sample} \times N'_{sample}$.

In the experiments, we take the points (u, u') from $N_{sample} \times N'_{sample}$ one by one and verify them if $r(u) - r(u') \ge \varepsilon > 0$. If $r(u) - r(u') \ge \varepsilon$ holds for all points in $N_{sample} \times N'_{sample}$, then all points in N_{sample} satisfy the decreasing condition. We will output the weights to get a candidate ranking function. Here, the value of ε should be given in advance, which can be adjusted. And the smaller ε is, the greater the probability of all points satisfy $r(u) - r(u') \ge \varepsilon$.

Algorithm 3 describes the process of using neural network to detect the candidate ranking function $r(u)$.

algorithm 3 Synthesis of candidate ranking functions

```
Input: a train-set $D$,the network structure, the sigmoid activation function.
Output: return the weights $w_i$ and $w_{ji}$.
1: Choose the tunable parameters $\varepsilon$ and $c_{assume}$.
2: Construct $N_{sample}$ by Algorithm 1;
3: Construct $D_{train}$ by Algorithm 2;
4: Training by DNN through the train-set $D_{train}$;
5: if the accuracy of test set == 1 then
6:     return the weights $w_i$ and $w_{ji}$;
7: else
8:     adjust the tunable parameters and optimization algorithm;
9: end if
```

4 Exact Verification

By Algorithm 3, we can get the value of w_i and w_{ji}, which reduce a candidate ranking function $r(u)$ over $U(\Omega)$. In this section, we will check if the $r(u)$ is a ranking function. To verify $r(u)$, we propose a new method instead of symbolic computation like $Z3$ and $SMT-solver$. At first, in Section 4.1, we put forward a proposition, which states that while $c < c_{assume}$ and all the points in N_{sample} obtained by Algorithm 1 satisfy $p(u) = r(u) - r(U \circ f \circ U^{-1}(u)) \geq \varepsilon > 0$, the function $r(u)$ satisfies the ***decreasing*** condition of ranking functions over $U(\Omega)$. And in Section 4.2, we will propose a method to calculate the upper bound c of $| \bigtriangledown (r(u) - r(u')|$ in Proposition 1.

4.1 Verification Process

The verification method includes two steps. First, we need to prove the union of all neighborhood with radius $\sqrt{n}d$ of each point in N_{sample} can cover the $U(\Omega)$, which has been implemented by Theorem 4 and its Remark. Then, we need to prove the points in the neighborhood of each point in N_{sample} satisfy the ***decreasing*** condition of ranking functions, i.e., for any point α in N_{sample}, $r(u)-r(U\circ f\circ U^{-1}(u)) \geq c^+$ holds for any $u \in \mathrm{O}(\alpha, \sqrt{n}d)$, this guarantees that $r(u) - r(U \circ f \circ U^{-1}(u)) \geq c^+$ holds for any u in $U(\Omega)$, since the union of neighborhoods of all points in N_{sample} can cover $U(\Omega)$.

By Theorem 3, we know that if $| \bigtriangledown (r(u) - r(U \circ f(U^{-1}(u))))|$ have a upper bound c and α is a point α in N_{sample} satisfies $r(\alpha) - r(U \circ f(U^{-1}(\alpha))) \geq \varepsilon$, then any point θ in the neighborhood $\mathrm{O}(\alpha, \sqrt{n}d)$ satisfies $r(\theta) - r(U \circ f(U^{-1}(\theta))) \geq \varepsilon - ndc$. Next, according to the ***decreasing*** condition, we need to give a condition to ensure that $\varepsilon - ndc$ is a positive number.

The Proposition 1 will show that when step $d = \frac{\varepsilon}{n \cdot c_{assume}}$ and each point in N_{sample} satisfies the ***decreasing*** condition, then each point in $U(\Omega)$ will satisfy the ***decreasing*** condition too, if $c < c_{assume}$.

Proposition 1 *If $c < c_{assume}$ and each point α in N_{sample} satisfy the* **decreasing** *condition $r(\alpha) - r(U \circ f(U^{-1}(\alpha))) \geq \varepsilon > 0$, then $\forall u \in U(\Omega)$, we have $r(u) - r(U \circ f(U^{-1}(u))) \geq \varepsilon - ndc > 0$.*

Proof.

By Theorem 3, we have already proved that if each point α in N_{sample} satisfies $r(\alpha) - r(U \circ f(U^{-1}(\alpha))) \geq \varepsilon > 0$, then $\forall \theta \in \mathrm{O}(\alpha, \sqrt{n}d)$, we have $r(\theta) - r(U \circ f(U^{-1}(\theta))) \geq \varepsilon - ndc$.

By (23), since

$$d = \frac{\varepsilon}{n \cdot c_{assume}}$$

substitute d into $p(\theta) \geq \varepsilon - ndc$, we have $\forall \theta \in \mathrm{O}(\alpha, \sqrt{n}d)$

$$r(\theta) - r(U \circ f(U^{-1}(\theta))) \geq \varepsilon - ndc = \varepsilon - n\frac{\varepsilon}{n \cdot c_{assume}}c = \varepsilon - \varepsilon\frac{c}{c_{assume}} = \varepsilon(\frac{c_{assume} - c}{c_{assume}}). \tag{25}$$

Owing to $c < c_{assume}$, we have $r(\theta) - r(U \circ f(U^{-1}(\theta))) \geq \varepsilon(\frac{c_{assume}-c}{c_{assume}}) > 0$.

by above, we know that for any point α in N_{sample}, $r(u) - r(U \circ f \circ U^{-1}(u)) \geq \varepsilon - ndc > 0$ holds for any $u \in \mathrm{O}(\alpha, \sqrt{n}d)$. By Theorem 4 and its Remark, we know that the union of neighborhoods of all points in N_{sample} can cover $U(\Omega)$. Thus, $r(u) - r(U \circ f \circ U^{-1}(u)) \geq \varepsilon - ndc > 0$ holds for any u in $U(\Omega)$. □

Remark 6 *By Theorem 3 and Proposition 1, we can prove that when $d = \frac{\varepsilon}{n \cdot c_{assume}}$, if $c < c_{assume}$, the function $r(u)$ satisfies the* **decreasing** *condition of ranking functions, i.e., $\forall u \in U(\Omega)$, we have $r(u)-r(U\circ f(U^{-1}(u))) \geq \varepsilon - ndc > 0$. Owing to $r(u)$ satisfies the bounded function naturally by Theorem 2, thus $r(u)$ is exactly a ranking function over $U(\Omega)$. By Theorem 1, since $r = R \circ U^{-1}$ and $u = U(x)$, $r(u) = R \circ U^{-1}U(x) = R(x)$, which is a ranking function over Ω. By our verification method, different from existing methods, the verification of candidate ranking function is no longer limited by the verification tools such as $Z3$.*

To check if the candidate ranking function $r(u)$ is a real ranking function, by Proposition 1 and its Remark, we need to compute the value of c and check if $c < c_{assume}$. In the following, we will illustrate how to compute the value of c.

4.2 Computation of c

In this section, we will propose a method to get the upper bound c of $|\bigtriangledown(r(u)-r(u'))|$, where $r(u)$ is a candidate ranking function of following form.

$$r(u) = \frac{1}{1+e^{-(\sum_{i=1}^{l} w_i \frac{1}{1+e^{-(\sum_{j=1}^{n+1} w_{ji}\cdot u_j)}})}}$$

where the value of w_i and w_{ji} have been obtained by Algorithm 3. Since $u' = U \circ f(U^{-1}(u))$,

$$|\bigtriangledown(r(u)-r(u'))| = |\bigtriangledown r(u) - \bigtriangledown r\circ U\circ f\circ U^{-1}(u)| = \left|\begin{pmatrix} \frac{\partial r(u)}{\partial u_1} \\ \frac{\partial r(u)}{\partial u_2} \\ \vdots \\ \frac{\partial r(u)}{\partial u_n} \end{pmatrix} - \begin{pmatrix} \frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_1} \\ \frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_2} \\ \vdots \\ \frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_n} \end{pmatrix}\right|. \tag{26}$$

Owing to $[sigmoid(z)]' = sigmoid(z)\cdot(1-sigmoid(z))$ and $sigmoid(z) \in (0,1)$, thus, $[sigmoid(z)]' \in (0, \frac{1}{4})$. Thus, we have

$$[sigmoid(z)]' = \frac{e^{-z}}{(1+e^{-z})^2} \in (0, \frac{1}{4}). \tag{27}$$

By (27), we have

$$\begin{aligned} \left|\frac{\partial r(u)}{\partial u_k}\right| &= \left|\left(\frac{1}{1+e^{-(\sum_{i=1}^{l} w_i \frac{1}{1+e^{-(\sum_{j=1}^{n+1} w_{ji}\cdot u_j)}})}}\right)' \cdot \frac{\partial \sum_{i=1}^{l} w_i \frac{1}{1+e^{-(\sum_{j=1}^{n+1} w_{ji}\cdot u_j)}}}{\partial u_k}\right| \\ &= \left|\left(\frac{1}{1+e^{-(\sum_{i=1}^{l} w_i \frac{1}{1+e^{-(\sum_{j=1}^{n+1} w_{ji}\cdot u_j)}})}}\right)'\right| \cdot \left|\sum_{i=1}^{l}[w_i(\frac{1}{1+e^{-(\sum_{j=1}^{n+1} w_{ji}\cdot u_j)}})' \cdot w_{ki}]\right| \\ &\le \tfrac{1}{16}\sum_{i=1}^{l}|w_i\cdot w_{ki}|, k \in (1,2,\cdots,n) \end{aligned} \tag{28}$$

$$\begin{aligned} \left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right| &= \left|\left(\frac{1}{1+e^{-(\sum_{i=1}^{l} w_i \frac{1}{1+e^{-(w_{(n+1)i}+\sum_{j=1}^{n} w_{ji}\cdot(m\circ f_j\circ U^{-1}(u)))}})}}\right)' \cdot \frac{\partial \sum_{i=1}^{l} w_i \frac{1}{1+e^{-(w_{(n+1)i}+\sum_{j=1}^{n} w_{ji}\cdot(m\circ f_j\circ U^{-1}(u)))}}}{\partial u_k}\right| \\ &= \left|\left(\frac{1}{1+e^{-(\sum_{i=1}^{l} w_i \frac{1}{1+e^{-(w_{(n+1)i}+\sum_{j=1}^{n} w_{ji}\cdot(m\circ f_j\circ U^{-1}(u)))}})}}\right)'\right| \\ &\cdot \left|\sum_{i=1}^{l} w_i[(\frac{1}{1+e^{-(w_{(n+1)i}+\sum_{j=1}^{n} w_{ji}\cdot(m\circ f_j\circ U^{-1}(u)))}})' \cdot \frac{\partial \sum_{j=1}^{n} w_{ji}\cdot(m\circ f_j\circ U^{-1}(u))}{\partial u_k}]\right| \\ &\le \tfrac{1}{16}\sum_{i=1}^{l}\left|w_i\cdot\sum_{j=1}^{n} w_{ji}\cdot\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|\right|, k\in(1,2,\cdots,n). \end{aligned} \tag{29}$$

For $\left|\frac{\partial r(u)}{\partial u_k}\right|$ in (28), since w_i and w_{ji} are known, the upper bound of $\left|\frac{\partial r(u)}{\partial u_k}\right|$ is known. For $\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right|$ in (29), if the upper bound of $\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|$ can be obtained, then the upper bound of $\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right|$ is derived.

Next, based on the the upper bound of $\frac{\partial r(u)}{\partial u_i}$ and $\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_i}$, we propose a method to calculate the upper bound c of $|\bigtriangledown(r(u)-r(u'))|$.

Method.

$$\begin{aligned}\left|\bigtriangledown(r(u)-r\circ U\circ f\circ U^{-1}(u))\right| &= \left|\begin{pmatrix}\frac{\partial r(u)}{\partial u_1}\\ \frac{\partial r(u)}{\partial u_2}\\ \vdots\\ \frac{\partial r(u)}{\partial u_n}\end{pmatrix}-\begin{pmatrix}\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_1}\\ \frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_2}\\ \vdots\\ \frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_n}\end{pmatrix}\right| \\ &\le \sum_{\mathrm{i}\in(1,2,\cdots,\mathrm{n})}\left|\frac{\partial r(u)}{\partial u_i}\right|+\sum_{\mathrm{i}\in(1,2,\cdots,\mathrm{n})}\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_i}\right|.\end{aligned}\tag{30}$$

By (28) and (29), we can get the upper bound of $\left|\frac{\partial r(u)}{\partial u_k}\right|$ and $\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right|$. Then by (30), we get the upper bound c of $|\bigtriangledown(r(u)-r(u'))|$.

It can be seen from (30) that the upper bound c of $\left|\bigtriangledown(r(u)-r\circ U\circ f\circ U^{-1}(u))\right|$ depends on the upper bound of $\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right|$. By (29), we know that the upper bound of $\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right|$ depends on the upper bound of $\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|$. Thus, the upper bound c depends on the upper bound of $\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|$. Next, we will discuss when the upper bound of $\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|$ is computable.

Since $x=U^{-1}(u)$, $x_j'=f_j(x)$ and $x_i=m^{-1}(x_i)=\ln(\frac{u_i}{1-u_i})$, we have

$$\begin{aligned}\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right| &= \left|(m\circ f_j\circ U^{-1}(u))'\cdot\frac{\partial(f_j\circ U^{-1}(u))}{\partial u_k}\right| \\ &= \left|(m\circ f_j(x))'\cdot\frac{\partial(f_j(x))}{\partial x_k}\cdot\frac{\partial x_k}{\partial u_k}\right| \\ &= \left|(m\circ f_j(x))'\cdot\frac{\partial(f_j(x))}{\partial x_k}\cdot\frac{1}{u_k(1-u_k)}\right| \\ &= \left|\frac{m(x_j')(1-m(x_j'))}{m(x_k)(1-m(x_k))}\cdot\frac{\partial(f_j(x))}{\partial x_k}\right| \\ &= \left|\frac{e^{x_k}+e^{-x_k}+2}{e^{x_j'}+e^{-x_j'}+2}\cdot\frac{\partial(f_j(x))}{\partial x_k}\right|.\end{aligned}\tag{31}$$

Next, we will illustrate how to compute the upper bound of $\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|$. There are two cases to consider:

A. The $\mathcal{H}$ is bounded and $\frac{\partial(f_j(x))}{\partial x_k}$ is continuous. Since $\frac{e^{x_k}+e^{-x_k}+2}{e^{x_j'}+e^{-x_j'}+2}$ and $\frac{\partial(f_j(x))}{\partial x_k}$ is continuous and $\mathcal{H}$ is bounded, then $\left|\frac{e^{x_k}+e^{-x_k}+2}{e^{x_j'}+e^{-x_j'}+2}\right|$ and $\left|\frac{\partial(f_j(x))}{\partial x_k}\right|$ have upper bounds over $\mathcal{H}$. Thus, we can compute the upper bound of $\left|\frac{\partial(m\circ f_j\circ U^{-1}(u))}{\partial u_k}\right|$ over $\mathcal{H}$.

Example 4 *Consider the loop in Example 1:*

$$\begin{aligned}&\textit{while}\ \ {x_1}^2+{x_2}^2\le 1\quad \textit{do}\\ &{x_1}'=x_1-{x_2}^2+1, {x_2}'=x_2+{x_1}^2-1\end{aligned}$$

Let $\mathcal{H}=\{(x_1,x_2):1\ge x_1\ge -1,\ge x_2\ge -1\}$. *And we can get* $\frac{\partial {x_1}'}{\partial x_1}=1$, $\frac{\partial {x_1}'}{\partial x_2}=-2x_2$, $\frac{\partial {x_2}'}{\partial x_1}=2x_1$ *and* $\frac{\partial {x_2}'}{\partial x_2}=1$.

Thus, by (31),

$$\begin{aligned}\left|\frac{\partial(m\circ f_1\circ U^{-1}(u))}{\partial u_1}\right| &= \left|\frac{e^{x_1}+e^{-x_1}+2}{e^{{x_1}'}+e^{-{x_1}'}+2}\cdot 1\right|\le\left|\frac{e^1+e^{-1}+2}{e^0+e^0+2}\right|\le 1.28,\\ \left|\frac{\partial(m\circ f_1\circ U^{-1}(u))}{\partial u_2}\right| &= \left|\frac{e^{x_2}+e^{-x_2}+2}{e^{{x_1}'}+e^{-{x_1}'}+2}\cdot(-2x_2)\right|\le\left|\frac{e^1+e^{-1}+2}{e^0+e^0+2}\cdot 2\right|\le 2.55,\\ \left|\frac{\partial(m\circ f_2\circ U^{-1}(u))}{\partial u_1}\right| &= \left|\frac{e^{x_2}+e^{-x_2}+2}{e^{{x_2}'}+e^{-{x_2}'}+2}\cdot 2x_1\right|\le\left|\frac{e^1+e^{-1}+2}{e^0+e^0+2}\cdot 2\right|\le 2.55,\\ \left|\frac{\partial(m\circ f_2\circ U^{-1}(u))}{\partial u_2}\right| &= \left|\frac{e^{x_1}+e^{-x_1}+2}{e^{{x_2}'}+e^{-{x_2}'}+2}\cdot 1\right|\le\left|\frac{e^1+e^{-1}+2}{e^0+e^0+2}\right|\le 1.28.\end{aligned}$$

Thus, by (29), we can get the upper bound of $\left|\frac{\partial r\circ U\circ f\circ U^{-1}(u)}{\partial u_k}\right|$. *Then, by (30), we can get the upper bound of* $|\bigtriangledown(r(u)-r(u'))|$.

B. The $\mathcal{H}$ is not bounded and $f_j(x)$ is linear, by (31), we just need to discuss whether the $\frac{e^{x_k}+e^{-x_k}+2}{e^{x_j'}+e^{-x_j'}+2}$ has a upper bound over $\mathcal{H}$.

It is easy to see that the function $y=e^x+e^{-x}+2$ monotonically decreases when $x\le 0$ and monotonically increases when $x>0$. Thus, when $x_j'>x_k>0$ or $x_j'<x_k<0$, $e^{x_j'}+e^{-x_j'}+2>e^{x_k}+e^{-x_k}+2$. The upper bound of $\left|\frac{e^{x_k}+e^{-x_k}+2}{e^{x_j'}+e^{-x_j'}+2}\right|$ is 1.

Next, we take an example to illustrate this, as follows:

Example 5 *Consider the following loop from [4]:*

$$\begin{array}{c} while\, x_1 > 0, x_1 \le x_2 \;\; do \\ {x_1}' = 2x_1, {x_2}' = x_2 + 1 \end{array}$$

Let $\mathcal{H} = \{(x_1, x_2) : x_1 > 0, x_2 \ge 0\}$. *And we can get* $\frac{\partial {x_1}'}{\partial x_1} = 2$, $\frac{\partial {x_1}'}{\partial x_2} = 0$, $\frac{\partial {x_2}'}{\partial x_1} = 0$ *and* $\frac{\partial {x_2}'}{\partial x_2} = 1$.

Thus, by (31), $\left|\frac{\partial(m \circ f_1 \circ U^{-1}(u))}{\partial u_1}\right| = \left|\frac{e^{x_1}+e^{-x_1}+2}{e^{{x_1}'}+e^{-{x_1}'}+2} \cdot 1\right|$.

Since $x_1' = 2x_1 > x_1$ *when* $x_1 > 0$, $e^{x_1'} + e^{-x_1'} + 2 > e^{x_1} + e^{-x_1} + 2$. *The upper bound of* $\left|\frac{e^{x_1}+e^{-x_1}+2}{e^{x_1'}+e^{-x_1'}+2}\right|$ *is 1. Thus, the upper bound of* $\left|\frac{\partial(m \circ f_1 \circ U^{-1}(u))}{\partial u_1}\right|$ *is 2.*

$\left|\frac{\partial(m \circ f_1 \circ U^{-1}(u))}{\partial u_2}\right| = \left|\frac{e^{x_2}+e^{-x_2}+2}{e^{{x_1}'}+e^{-{x_1}'}+2} \cdot 0 = 0\right|$ *and* $\left|\frac{\partial(m \circ f_2 \circ U^{-1}(u))}{\partial u_1}\right| = \left|\frac{e^{x_1}+e^{-x_1}+2}{e^{{x_2}'}+e^{-{x_2}'}+2} \cdot 0 = 0\right|$.

$\left|\frac{\partial(m \circ f_2 \circ U^{-1}(u))}{\partial u_2}\right| = \left|\frac{e^{x_1}+e^{-x_1}+2}{e^{{x_2}'}+e^{-{x_2}'}+2} \cdot 1\right|$.

Since $x_2' = x_2 + 1 > x_2$, $e^{x_2'} + e^{-x_2'} + 2 > e^{x_2} + e^{-x_2} + 2$. *The upper bound of* $\left|\frac{e^{x_2}+e^{-x_2}+2}{e^{x_2'}+e^{-x_2'}+2}\right|$ *is 1. Thus, the upper bound of* $\left|\frac{\partial(m \circ f_2 \circ U^{-1}(u))}{\partial u_2}\right|$ *is 1.*

By using the method, we can get an upper bound c of $|\bigtriangledown(r(u) - r(u'))|$ in some case. Then we compare c with c_{assume} to determine whether $r(u)$ is the ranking function over $U(\Omega)$.

To sum up, we can elicit the process of the algorithm, which embedded in Algorithm 4.

algorithm 4 Synthesis of ranking functions

```
Input: a train-set $D_{train}$,the network structure.
Output: return the ranking function $R(x)$.
 1: Choose the tunable parameters $\varepsilon$ and $c_{assume}$;
 2: Calculate the step size $d$: $d = \frac{\varepsilon}{n \cdot c_{assume}}$;
 3: Construct $N_{sample}$ by Algorithm 1;
 4: Construct a train-set $D_{train}$ by Algorithm 2;
 5: Detect the candidate ranking function $r(u)$ by Algorithm 3;
 6: Compute the upper bound $c$;
 7: if $c < c_{assume}$ then
 8:     return $R(x)$;
 9: else
10:     adjust $c_{assume}$ and the tunable parameters to train again;
11: end if
```

Remark 7 *In our experiments, in general, if c obtained by the method states in Section 4.2 still does not satisfy* $c < c_{assume}$, *then we increase the value of* c_{assume}. *The process may not be terminating. Thus, in our experiment, to guarantee the process is terminating, we limit the number of sample points does not exceed* 10^7. *At the same time, when* c_{assume} *becomes larger, the step d becomes smaller. This leads to the more sample points and the longer sampling time we need.*

Example 6 *Following the example[1-4],by Algorithm 3, we output the weight result, as follows:*

Table 3: Output of weights

the weights w_{ji}:	-2.0015616416931152	4.5431742668151855	-0.9806060791015625
	3.2906837463378906	-1.5724971294403076	-1.1101324558258057
	2.1770262718200684	-2.0126399993896484	0.1282458603382111
the weights w_i:	2.812188148498535	-0.9688100218772888	-0.7075088024139404

Note that

$$\begin{array}{l} w_{ji} : [[w_{11}, w_{21}, w_{31}][w_{12}, w_{22}, w_{32}][w_{13}, w_{23}, w_{33}]] \\ w_i : [w_1, w_2, w_3]. \end{array} \tag{32}$$

Then, we calculate the c according to the above method states in Section 4.2. We get $c = 7.88 < 12.5 = c_{assume}$.

By Proposition 1 and its Remark, the candidate ranking function $r(u)$ *is exact a ranking function for this loop.*

Owing to $r = R \circ U^{-1}$ *and* $u = U(x)$, *we have* $R(x) = r(u)$, *as follows:*

$$R := \frac{1}{1+e^{-\frac{50161}{17837\left(1+e^{\frac{47423}{23693\left(1+e^{-x1}\right)}-\frac{51983}{11442\left(1+e^{-x2}\right)}+\frac{64265}{65536}}\right)}+\frac{33795}{34883\left(1+e^{-\frac{63191}{19203\left(1+e^{-x1}\right)}+\frac{141064}{89707\left(1+e^{-x2}\right)}+\frac{16511}{14873}}\right)}+\frac{23311}{32948\left(1+e^{-\frac{38947}{17890\left(1+e^{-x1}\right)}+\frac{12579}{6250\left(1+e^{-x2}\right)}-\frac{33342}{259985}}\right)}}}$$

Fig. 6: Ranking function $R(x)$

5 Experiments

A prototype of our approach is implemented in Algorithm 4 and we use python to obtain the candidate ranking function $r(u)$ by the package tensorflow[1], which used in building neural network. All the computations were performed on a PC equipped with a 2.9 GHz Intel Core i5-9400f CPU and 8 GB 2666 MHz DDR4 RAM. The procedure to synthesize a candidate ranking function for a given loop is embedded in Algorithm 3. In order to facilitate the calculation of c, in our experiment, we choose three-layer network structure. The input layer contains $n+1$ neurons when n variables in the loop structure. The hidden layer contains three neurons three neurons and the output layer contains one neuron. And we set all activation functions as sigmoid function. Moreover, the selection of optimization algorithm, the ε and c_{assume} are set as the tunable parameter in the training.

5.1 Experimental results

Table 4 shows the loops involved in the experiment. All loops are generated based on two ways: one is from other work, the loops(1,13-14,17) are from [29], while loop(18) are from [4], and the others like the loops(2-12,15-16,19-25) are gotten by simply make some modifications on the loops which used in the work of [29] and [4].

And we choose different examples for one-dimension(i:1-6), two-dimensions(ii:7-20) and three-dimensions(iii:21-25). In addition, the loops with transcendental terms are marked •. And the loops without transcendental terms are marked ○.

Table 4: Loops for experiments.(○: loops without transcendental terms; •: loops with transcendental terms.)

#	dimension	loop	type
1	i	$while\, x \geq 1, x \leq 3 \quad do\, x' = 5x - x^2$	○
2	i	$while\, x > 4 \quad do\, x' = -2x + 4$	○
3	i	$while\, x > 1 \quad do\, x' = -x$	○
4	i	$while\, x^2 - 3x + 2 \leq 0 \quad do\, x' = 5x - x^2$	○
5	i	$while\; \sin(x) \geq 0, x \geq 1, x \leq 6 \quad do\, x' = -x$	•
6	i	$while\; \sin(x) \geq 0, x \geq 1, x \leq 6 \quad do\, x' = x + 1 + \cos(x)^2$	•
7	ii	$while\; x_1 - x_2 \geq 1, x_1 + x_2 \geq 1, x_1 \leq 2 \quad do\, x_1' = x_1 - 1, x_2' = x_2 - 1$	○
8	ii	$while\, x_1 \geq 0, x_2 - 2x_1 \geq 1, x_2 \leq 3 \quad do\, x_1' = -{x_1}^2 - 4{x_2}^2 + 1, x_2' = -x_1x_2 - 1$	○
9	ii	$while\, {x_1}^2 + {x_2}^2 \leq 1, \cos(x_1) \geq 0 \quad do\, x_1' = {x_1}^2 + 1, x_2' = {x_2}^2 - 1$	•
10	ii	$while\, {x_1}^2 + {x_2}^2 \leq 1 \quad do\, x_1' = x_1 - 1 - \cos(x_1), x_2' = x_2 - 1 - \cos(x_2)$	•
11	ii	$while\, {x_1}^2 + {x_2}^2 \leq 1, \cos(x_1) \geq 0 \quad do\, x_1' = x_1 - 1 - \sin(x_1)^2, x_2' = x_2 - 1 - \cos(x_2)^3$	•
12	ii	$while\, x_1 \geq 1, x_1 \leq x_2 \quad do\, x_1' = \frac{1}{16}x_1 + 2 + e^{-x_1}, x_2' = \frac{1}{16}x_2 + 1$	•
13	ii	$while\, {x_1}^2 + {x_2}^2 \leq 1 \quad do\, x_1' = x_1 - {x_2}^2 + 1, x_2' = x_2 + {x_1}^2 - 1$	○
14	ii	$while\, x_1 + x_2 \geq 1, x_1 \leq 3, {x_2}^2 \leq 1 \quad do\, x_1' = 5x_1 - x_1^2, x_2' = {x_2}^2 + x_2$	○
15	ii	$while\, {x_1}^2 \leq 1, {x_2}^2 \leq 1 \quad do\, x_1' = x_1 + x_2, x_2' = x_2 - 1$	○
16	ii	$while\, {x_2}^2 - x_2 + 1 \leq x_1 \quad do\, x_1' = -x_1, x_2' = -x_2$	○
17	ii	$while\, x_1 \geq 1, {x_2}^2 + 2x_1 \leq 3x_2 \quad do\, x_1' = 1 + \frac{1}{{x_1}^2}, x_2' = -x_1x_2 - 3x_2 + {x_2}^2 + 1$	○
18	ii	$while\, x_1 > 0, x_1 \leq x_2 \quad do\, x_1' = 2x_1, x_2' = x_2 + 1$	○
19	ii	$while\, x_1 \geq 1, x_1 \leq x_2 \quad do\, x_1' = x_1 + x_2, x_2' = -x_2$	○
20	ii	$while\, x_1 \geq 0, x_2 \geq 0 \quad do\, x_1' = x_1 + x_2, x_2' = x_2 - 1$	○
21	iii	$while\, x_2 \geq 0, x_2 \leq x_1, x_1 \leq 1 \quad do\, x_1' = x_1 + 1, x_2' = \frac{1}{4}\tan(x_2) + 1, x_3' = -x_3$	•
22	iii	$while\, {x_1}^2 + {x_2}^2 + {x_3}^2 \leq 1 \quad do\, x_1' = x_1 - 1 - \cos(x_3), x_2' = x_2 - 1 - \cos(x_1), x_3' = x_3 - 1 - \sin(x_3)$	•
23	iii	$while\, {x_1}^2 + {x_2}^2 + {x_3}^2 \leq 1, \sin(x_2) \geq 0 \quad do\, x_1' = x_1 - 1, x_2' = x_2, x_3' = \sin(x_3)^3$	•
24	iii	$while\, {x_1}^2 + {x_2}^2 \leq 1, x_3 \leq 0 \quad do\, x_1' = \frac{1}{4}x_1 + \frac{1}{8}x_2, x_2' = \frac{1}{4}{x_1}^2, x_3' = \frac{1}{6}x_3 - 1$	○

continued on next page

continued from previous page			
#	dimension	loop	type
25	iii	$while\, x_2 \geq 1, x_2 \leq x_1, x_3 \geq 0 \quad do\, x_1' = -x_1, x_2' = x_2, x_3' = x_3 + 1$	○

Table 5 depicted the tunable parameters in the experiment and the weights. The first column in the table denotes the loops from Table 4. The next four columns denote all experimental tunable parameters mentioned before. The fifth column denotes the weights obtained by Algorithm 3. The last column denotes the upper bound c.

As can be seen from the experimental results in Table 5, by our verification method states in Section 4.1, the verification of candidate ranking function is no longer limited by the verification tools such as *Z3*. Especially for the loop contains transcendental terms, such as one-dimensional (5-6), two-dimensional (9-12) or three-dimensional (21-23), one may detect their ranking functions by our DNN-based method. In particular, For loop (18), existing methods can only find their multi-phase linear ranking functions. However, we can get a candidate ranking function.

Table 5: The tunable parameters and experimental results

loop	c_{assume}	ε	d	optimization algorithm	*weight*	c
1	50	0.05	0.001	ADAM	[[-5.855921745300293 -2.330543041229248] [-20.364992141723633 18.527238845825195] [-6.136241912841797 -1.1225796937942505]] [[1.4383845329284668 1.3277370929718018 0.7966102957725525]]	10.09
2	50	0.05	0.001	ADAM	[[1.0260323286056519 2.6372127532958984] [2.592477560043335 -0.6153714060783386] [0.594828188419342 -2.5261006355285645]] [[-0.5611265301704407 2.0623490810394287 -1.0028291940689087]]	1.23
3	10	0.01	0.001	ADAM	[[-2.3436334133148193 -0.016625851392746] [-0.056898158043623 -1.8876415491104126] [4.553391933441162 -1.6753664016723633]] [[1.128777265548706 2.5279500484466553 -0.9741539359092712]]	0.98
4	10	0.01	0.001	ADAM	[[-2.920186996459961 1.7360831499099731] [-5.163623809814453 4.203181743621826] [1.6039228439331055 -0.1343504935503006]] [[1.128777265548706 2.5279500484466553 -0.9741539359092712]]	4.47
5	10	0.01	0.001	ADADELTA	[[1.8124676942825317 1.1652917861938477] [-2.872796058654785 0.2472831457853317] [-0.2946441173553467 -1.7745099067687988]] [[1.497663974761963 -2.4328577518463135 -0.796199381351471]]	1.25
6	10	0.01	0.001	ADAM	[[-2.45465350151062 -3.5235118865966797] [-3.573195219039917 -1.7213060855865479] [-9.615031242370605 6.443149089813232]] [[0.2788458168506622 0.4986115992069244 4.53493070602417]]	8.64
7	2.5	0.01	0.002	ADAM	[[-0.9724711775779724 -1.790913701057434 0.1311937719583511] [-0.4343473315238953 3.5478222370147705 -0.9729806184768677] [-2.210174798965454 -0.0758214443922043 -2.24971866607666]] [[-1.6925342082977295 1.333398461341858 -0.1803609728813171]]	1.51
8	12.5	0.05	0.002	ADAM	[[1.3154786825180054 0.4052276909351349 0.1627338230609894] [-0.0130754858255386 0.5321783423423767 -0.3574274182319641] [1.377364993095398 0.1813266575336456 1.0136044025421143]] [[2.0278115272521973 -0.3657838702201843 0.3367987871170044]]	5.56
9	12.5	0.05	0.002	ADAM	[[2.7846662998199463 -2.4136955738067627 -1.3070461750030518] [-4.715053081512451 2.150216579437256 2.7339258193969727] [-2.258000135421753 1.4488911628723145 0.3058335483074188]] [[-2.8017733097076416 2.469788074493408 -0.5276959538459778]]	6.28
10	12.5	0.05	0.002	ADAM	[[3.03637433052063 4.8203911781311035 -2.6361677646636963] [-2.137758255004883 0.5550006031990051 -0.438726007938385] [-1.7149194478988647 -1.5624099969863892 -1.8290647268295288]] [[1.4087376594543457 -0.779438316822052 -1.5182971954345703]]	3.67
11	12.5	0.05	0.002	ADAM	[[2.6632120609283447 2.6698906421661377 -1.0566554069519043] [-1.171133279800415 -1.5838327407836914 -1.721509337425232] [-1.7054489850997925 -2.4077670574188232 0.358491837978363]] [[1.700181484222412 -0.7515915632247925 -3.1739611625671387]]	5.68
12	2.5	0.01	0.002	ADAM	[[0.3941701352596283 -2.3517730236053467 0.2161412388086319] [0.8995246887207031 -2.7943689823150635 0.6710333824157715] [-2.3223273754119873 1.5255376100540161 1.8248270750045776]] [[-0.5101912617683411 -2.947547674179077 1.513954520225525]]	1.41
13	12.5	0.05	0.002	ADAM	[[-2.0015616416931152 4.5431742668151855 -0.9806060791015625] [3.2906837463378906 -1.5724971294403076 -1.1101324558258057] [2.1770262718200684 -2.0126399993896484 0.1282458603382111]] [[2.812188148498535 -0.9688100218772888 -0.7075088024139404]]	7.88
14	12.5	0.05	0.002	ADAM	[[3.6623892784118652 -1.410606861114502 -0.6618162989616394] [-3.011399745941162 0.7222720384597778 0.8571652770042419] [12.188863754272461 -3.7000672817230225 -3.3904097080230713]] [[0.3377119898796082 -1.1197441816329956 1.1122510433197021]]	3.90
continued on next page						

continued from previous page						
loop	c_{assume}	ε	d	optimization algorithm	*weight*	c
15	12.5	0.05	0.002	ADAM	[[-1.8072351217269897 5.725420951843262 -1.3815851211547852] [0.3504117131233215 -3.3405511379241943 -0.0264695025980473] [-2.0926411151885986 -2.5694515705108643 -3.046679973602295]] [[1.9843451976776123 -2.579214572906494 -0.310302734375]]	5.07
16	12.5	0.05	0.002	ADAM	[[2.5586698055267334 0.4331809282302856 -1.5553065538406372] [-0.8438130021095276 -0.5035918354988098 -1.6053627729415894] [-0.5477704405784607 -0.9439965486526489 -0.1063966006040573]] [[0.8558939695358276 -0.3794965744018555 -0.3961026072502136]]	0.46
17	12.5	0.05	0.002	ADAM	[[-0.3921792209148407 3.1874005794525146 -0.865526020526886] [2.1137895584106445 -1.799368977546692 -0.703197181224823] [-0.5424444079399109 4.590944290161133 -1.416419506072998]] [[0.6584115028381348 -1.321570634841919 1.2126175165176392]]	6.66
18	0.5	0.001	0.001	ADADELTA	[[1.048069953918457 2.0997307300567627 -2.3266866207122803] [3.4703633785247803 -0.5784722566604614 -2.340416431427002] [-1.4391871690750122 -1.5317275524139404 2.1755189895629883]] [[-0.760312020778656 -1.1201003789901733 1.9844242334365845]]	0.48
19	12.5	0.05	0.002	ADAM	[[-2.507380723953247 -0.1853876709938049 -2.0108020305633545] [-0.083501785993576 -2.4635047912597656 0.7140545845031738] [-0.3050325810909271 3.7262420654296875 -1.5262598991394043]] [[-0.5840973258018494 -1.3084328174591064 1.5039777755737305]]	1.93
20	2.5	0.01	0.002	ADAM	[[2.3963534832000732 3.0890965461730957 -0.8048329949378967] [-2.9532835483551025 4.063777923583984 -2.323068141937256] [-2.499047040939331 3.341022491455078 2.2167253494262695]] [0.4436832368373871 0.8593196868896484 0.7724987268447876]	0.88
21	20/3	0.1	0.005	ADAM	[[-2.174838066101074 -0.6407662034034729 -1.170314908027649 1.511257290840149] [-0.3066220283508301 2.0138161182403564 0.490244060754776 1.3864461183547974] [-2.763887882232666 -3.4592125415802 0.1967869102954865 3.6580541133880615]] [[0.8110175132751465 -1.3365472555160522 3.5431087017059326]]	3.59
22	20/3	0.1	0.005	ADAM	[[2.1336159706115723 1.1494638919830322 1.5633387565612793 -0.6107999682426453] [-1.323683500289917 -1.2304725646972656 -1.2066571712493896 -0.3168062269687653] [-1.8960381746292114 -3.18878436088562 -0.0216311700642109 0.9264867901802063]] [[1.537946343421936 -1.1959309577941895 -1.6288409233093262]]	5.93
23	20/3	0.1	0.005	ADADELTA	[[9.287436485290527 -0.319311797618866 -1.9479351043701172 -2.3329615592956543] [-2.340475082397461 -2.9602091312408447 -3.5425498485565186 -2.3635144233703613] [-3.0034594535827637 0.5185515880584717 0.9227628111839294 -0.2369051575660706]] [[1.9518927335739136 0.056944951415062 -2.5973238945007324]]	5.01
24	20/3	0.1	0.005	ADAM	[[2.5731589794158936 -1.297593116760254 -3.3742153644561768 -0.3204583525657654] [2.260584592819214 -3.5005502700805664 0.7450693845748901 -0.8528845906257629] [-2.1284077167510986 4.673758029937744 0.0090981414541602 -0.0352618470788002]] [[-0.6772419214248657 -2.3815860748291016 2.3263206481933594]]	4.89
25	20/3	0.1	0.005	ADAM	[[3.2802696228027344 0.0690874233841896 -0.435204029083252 -2.0471887588500977] [-2.0195391178131104 -0.7831394672393799 1.593719482421875 -0.6381518244743347] [1.090225338935852 -1.6407567262649536 -0.3074676394462585 -1.6080644130706787]] [[2.244703769683838 -1.851402759552002 -0.3808052837848663]]	2.21

Note that the n_1 and n_2 in Algorithm 2 also is the tunable parameter in the experiment, but in most of the next experiments, let $n_1 = 0.5$ $n_2 = 0.2$. However, for loop (18), we will adjust the $n_1 = 0.5$ and $n_2 = 0.3$ to achieve the best training effect.

5.2 Method Comparsion

Most of existing methods focus on the synthesis of polynomial ranking functions, but not all terminating loops have polynomial ranking functions. According to the universal approximation theorem, our method can detect more expressive ranking functions by DNN, which is non-polynomial. Moreover, compared with SVM-basedmethod, we propose a new method to check if a candidate ranking function is exact a ranking function. Different from existing method depends on *Z3*, our method can verify if a candidate ranking function with transcendental terms is exact a ranking function.

We next compared our DNN-based method with QE-based method[8], SVM-based method [29] and method for multi-phase linear ranking functions in [4].

A. For QE-based method[8], QE-based method can only deal with the polynomial loops and synthesis polynomial ranking functions. But our method can not only deal with the polynomial loops, but also the loops with transcendental terms. Moreover, the ranking functions obtained by our method are non-polynomial.

B. For SVM-based method[29], it is main idea is: Firstly, for a given loop, the synthesis of polynomial ranking functions of this loop is reduced to the classfication problem; Secondly, a candidate ranking function is obtained by SVM; Finally, the SMT tool *Z3* is used to check if the obtained candidate ranking function is exact a ranking function. Therefore, the SVM-based methods rely heavily on *Z3*. However, *Z3* cannot deal with the expressions with transcendental terms such as $\sin(\cdot)$ and $\cos(\cdot)$. Thus, the SVM-based method cannot deal with the loops (5-6,9-12,21-23) with transcendental terms. However, by Proposition 1 and its Remark, for the loops with transcendental terms, we propose a new approach to check if a candidate ranking function is exact a ranking function.

C. [4] presents an LP-based method to synthesize multi-phase linear ranking functions for single- path linear-constraints loops. For loop (18) having multi-phase ranking functions by [4], our method indeed can find its global ranking functions by DNN. One of the reasons is, we think, the neural network can approximate any function under given conditions according to the universal approximation theorem. Therefore, the multi-phase ranking functions of loop (18) may be approximated by the DNN. We illustrate this by following example.

Example 7 *Consider the following loop:*

$$\begin{array}{l} \textit{while}\, x_1 > 0, x_1 \le x_2 \quad \textit{do} \\ {x_1}' = 2x_1, {x_2}' = x_2 + 1 \end{array}$$

Firstly, we choose the neural network structure, as follows:

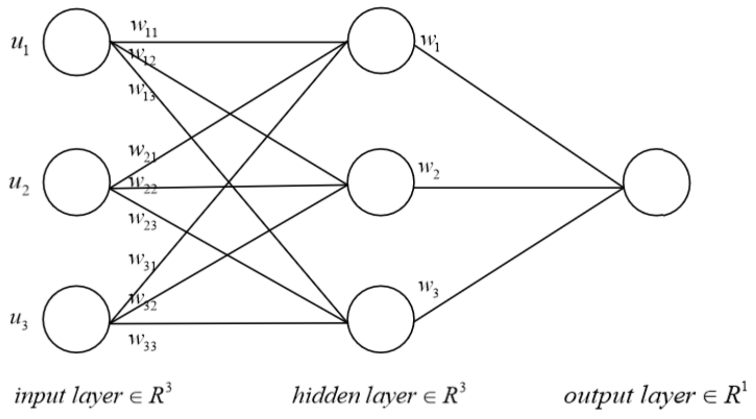

Fig. 7

Secondly, we assume that $c_{assume} = 0.50$ *and set* $\varepsilon = 0.001$. *Thus we can calculate the step* $d = \frac{\varepsilon}{n \cdot c_{assume}} = \frac{0.001}{a2 \cdot 0.50} = 0.001$ *and construct* N_{sample} *by Algorithm 1. Thirdly, we construct the train-set* D_{train} *by Algorithm 2. It should be noted that we set* n_1 *to 0.5 and set* n_2 *to 0.3 in Algorithm 2.*

We begin to adjust parameters and make a training until the accuracy of test-set is 1. We find that when we choose ADAM as optimization algorithm and set learning rate to 0.1, the accuracy of test-set reaches 1.

By Algorithm 3, we get

Table 6: Output weights

the weights w_{ji}:	1.048069953918457	2.0997307300567627	-2.3266866207122803
	3.4703633785247803	-0.5784722566604614	-2.340416431427002
	-1.4391871690750122	-1.5317275524139404	2.1755189895629883
the weights w_i:	-0.760312020778656	-1.1201003789901733	1.9844242334365845

By Table 6, Since w_{ji} *and* w_i *are known, we can get a candidate ranking function* $r(u)$. *By method for computing c states in Section 4.2, for the candidate ranking function* $r(u)$, *we get* $c = 0.48 < 0.50 = c_{assume}$.

Therefore, by Proposition 1 and its Remark, $r(u)$ *is the ranking function over* $U(\Omega)$. *By Theorem 1, we can get the desired ranking function* $R(x)$ *over* Ω *of loop (18), as follows:*

$$R := \frac{1}{1 + e^{\frac{97177}{127812\left(1+e^{-\frac{23133}{22072\left(1+e^{-x1}\right)} - \frac{62383}{29710\left(1+e^{-x2}\right)} + \frac{50524}{21715}}\right)} + \frac{31691}{28293\left(1+e^{-\frac{73537}{21190\left(1+e^{-x1}\right)} + \frac{11791}{20383\left(1+e^{-x2}\right)} + \frac{120384}{51437}}\right)} - \frac{33380}{16821\left(1+e^{\frac{14448}{10039\left(1+e^{-x1}\right)} + \frac{57233}{37365\left(1+e^{-x2}\right)} - \frac{31334}{14403}}\right)}}}$$

Fig. 8: Ranking function $R(x)$

6 Conclusion

In this paper, we propose a DNN-based method to synthesize non-polynomial ranking functions for loop programs. We leverage DNN to construct a candidate ranking function first, and then use the method states in Section 4.1, which states that if $c < c_{assume}$ holds, to verify if the candidate ranking function by DNN is exact a ranking function of the loop program. Compared with existing methods, our methods can deal with loops with transcendental terms and detect more expressive ranking functions. What's more, for some loops having multi-phase ranking functions, our method can also find their global ranking functions. However, our method still has its own limitations. For instance, when a candidate ranking function $r(u)$ is obtained by DNN, in order to verify check if $r(u)$ is a ranking function, we need to compare the upper bound c of $|\bigtriangledown(r(u)-r(u'))|$ with c_{assume}. In Section 4.2, by (28), (29) and (30), we know that the computation of the upper bound c depends on the upper bound of $\left|\frac{\partial(m \circ f_j \circ U^{-1}(u))}{\partial u_k}\right|$. But there is no general method to compute the upper bound of $\left|\frac{\partial(m \circ f_j \circ U^{-1}(u))}{\partial u_k}\right|$. If the upper bound of $\left|\frac{\partial(m \circ f_j \circ U^{-1}(u))}{\partial u_k}\right|$ cannot be obtained, our method fails. In addition, if $c < c_{assume}$ does not hold, we have to increase the value of c_{assume} to get a new $c^{(1)}_{assume}$. Then we invoke Algorithm 3 to get another candidate $r^{(1)}(u)$ and compute the value of $c^{(1)}$ to compare $c^{(1)}$ with $c^{(1)}_{assume}$. If $c^{(1)} < c^{(1)}_{assume}$ holds, then $r^{(1)}(u)$ is desire a ranking function over $U(\Omega)$. If not, the above process have to proceed.

Acknowledgements This research is partially supported by the National Natural Science Foundation of China NNSFC(61572024, 11771421,61103110),the Natural Science Foundation of Chongqing (cstc2019jcyj-msxmX0638) and Chinese Academy of Sciences "Light of West China" program.

Conflict of interest

The authors declare that no support, financial or otherwise, has been received from any organization that may have an interest in the submitted work and there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. URL http://tensorflow.org/, software available from tensorflow.org
2. Bagnara R, Mesnard F (2013) Eventual linear ranking functions. principles and practice of declarative programming pp 229–238
3. Ben-Amram AM, Genaim S (2014) Ranking functions for linear-constraint loops. J ACM 61(4), DOI 10.1145/2629488, URL https://doi.org/10.1145/2629488
4. Ben-Amram AM, Genaim S (2017) On multiphase-linear ranking functions. In: International Conference on Computer Aided Verification, Springer, pp 601–620
5. Bradley AR, Manna Z, Sipma HB (2005) Linear ranking with reachability. In: International Conference on Computer Aided Verification, Springer, pp 491–504
6. Bradley AR, Manna Z, Sipma HB (2005) The polyranking principle. In: International Colloquium on Automata, Languages, and Programming, Springer, pp 1349–1361
7. Braverman M (2006) Termination of integer linear programs. In: International Conference on Computer Aided Verification, Springer, pp 372–385
8. Chen Y, Xia B, Yang L, Zhan N, Zhou C (2007) Discovering non-linear ranking functions by solving semi-algebraic systems. In: International Colloquium on Theoretical Aspects of Computing, Springer, pp 34–49
9. Colón MA, Sipma HB (2002) Practical methods for proving program termination. In: International Conference on Computer Aided Verification, Springer, pp 442–454
10. Colóon MA, Sipma HB (2001) Synthesis of linear ranking functions. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, pp 67–81
11. Cousot P (2005) Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In: International Workshop on Verification, Model Checking, and Abstract Interpretation, Springer, pp 1–24
12. Cousot P, Cousot R (2012) An abstract interpretation framework for termination. ACM SIGPLAN Notices 47(1):245–258
13. Fan R, Chang K, Hsieh C, Wang X, Lin C (2008) Liblinear: A library for large linear classification. Journal of Machine Learning Research 9:1871–1874
14. Hornik K, Stinchcombe M, White H, et al. (1989) Multilayer feedforward networks are universal approximators. Neural networks 2(5):359–366

15. Hornik K, Stinchcombe MB, White H (1990) Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Networks 3(5):551–560
16. Lecun Y, Bengio Y, Hinton GE (2015) Deep learning. Nature 521(7553):436–444
17. Lee CS, Jones ND, Ben-Amram AM (2001) The size-change principle for program termination. SIGPLAN Not 36(3):81–92, DOI 10.1145/373243.360210, URL https://doi.org/10.1145/373243.360210
18. Leike J, Heizmann M (2014) Ranking templates for linear loops. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, pp 172–186
19. Leshno M, Lin V, Pinkus A, Schocken S (1993) Original contribution: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks 6(6):861–867
20. Li Y, Zhu G, Feng Y (2016) The l-depth eventual linear ranking functions for single-path linear constraint loops. In: 2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE), IEEE, pp 30–37
21. Li Y, Sun X, Li Y, Turrini A, Zhang L (2019) Synthesizing nested ranking functions for loop programs via svm. In: International Conference on Formal Engineering Methods, Springer, pp 438–454
22. Li Y, Wu W, Feng Y (2019) On ranking functions for single-path linear-constraint loops. International Journal on Software Tools for Technology Transfer pp 1–12
23. de Moura L, Bjørner N (2008) Z3: An efficient smt solver. In: Ramakrishnan CR, Rehof J (eds) Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 337–340
24. Podelski A, Rybalchenko A (2004) A complete method for the synthesis of linear ranking functions. In: Steffen B, Levi G (eds) Verification, Model Checking, and Abstract Interpretation, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 239–251
25. Shen L, Wu M, Yang Z, Zeng Z (2013) Generating exact nonlinear ranking functions by symbolic-numeric hybrid method. Journal of Systems Science & Complexity 26(2):291–301
26. Tiwari A (2004) Termination of linear programs. In: Alur R, Peled DA (eds) Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 70–82
27. Turing AM (1937) On computable numbers, with an application to the entscheidungsproblem. Proceedings of The London Mathematical Society 41(1):230–265
28. Urban C (2013) The abstract domain of segmented ranking functions. In: International Static Analysis Symposium, Springer, pp 43–62
29. Yuan Y, Li Y (2019) Ranking function detection via svm: A more general method. IEEE Access 7:9971–9979
30. Yuan Y, Li Y, Shi W (2019) Detecting multiphase linear ranking functions for single-path linear-constraint loops. International Journal on Software Tools for Technology Transfer pp 1–13