Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

DeepGlobal: A framework for global robustness verification of feedforward neural networks

Weidi Sun, Yuteng Lu, Xiyue Zhang, Meng Sun *

School of Mathematical Sciences, Peking University, Beijing, China

ARTICLE INFO

Keywords:
Feed forward neural networks
Robustness
Global verification

ABSTRACT

Feed forward neural networks (FNNs) have been deployed in a variety of domains, though achieving great success, also pose severe safety and reliability concerns. Existing adversarial attack generation and automatic verification techniques cannot formally verify a network globally, i.e., finding all adversarial dangerous regions (ADRs) of a network is out of their reach. To address this problem, we develop a global robustness verifiable FNN framework DeepGlobal with four components: 1) a rule-generator finding all potential boundaries of a network by logical reasoning; 2) a new network architecture *Sliding Door Network (SDN)* enabling rule generation in a feasible way; 3) a selector which selects real boundaries from the generated potential boundaries; 4) a filter finding ADRs with meaningful adversarial examples. The ADRs can be represented by the identified real boundaries. We demonstrate the effectiveness of our approach on both synthetic and real datasets.

1. Introduction

Feed forward neural networks (FNNs) have been applied to a variety of domains and achieved great success. Reliance on FNNs' decisions in safety-critical applications makes their behavior correctness of high importance. Recent researches have shown that the correctness of FNNs is threatened by their susceptibility to human-imperceptible adversarial perturbations [1–3].

To explore FNNs' robustness, existing research attempts mainly fall into three categories: *crafting adversarial examples*, *automatic verification*, and *prediction explanation*. Given an input sample, adversarial example generation techniques [4–8] cannot guarantee that no adversarial example exists around the given input, when they fail to generate adversarial examples. *Automatic verification* mainly focuses on the guarantee of local robustness [9–13], i.e., the robustness of a single input's neighborhood. Such verification approaches can provide a rigorous local robustness proof that adversarial examples do not exist in a local region. However, the local robustness only takes a small part of the input space into account, and thus cannot guarantee the robustness of the whole network for every possible input. Along with the *automatic verification* thread of local robustness, the technique developed in [14] goes a step further. It evaluates the local robustness of each sample in a test dataset and treats the expected value of evaluation results as the indicator of "global robustness". The technique in [14] can be considered as finding expected maximum safe radius over the test dataset. However, the selection of the test dataset would directly influence the estimation of global robustness. The efforts in *prediction explanation* related to our work focus on attributing predictions to input features. They are based on reachable set computation [15], back-propagating the prediction score [16], computing local linear approximations [17] or cooperative game theory [18]. The closest work to ours, focuses on inferring input–output properties of neural networks [19]. Nevertheless, these techniques cannot provide a comprehensive and formal analysis for the whole input space. We can easily identify two stumbling blocks on the path of FNNs' global verification: the *complex activation patterns*[1] and the *large input space*. It is computationally unacceptable to analyze all possible activation patterns or traverse input space to provide evidence for FNNs' dependable operation.

In this paper, we develop a framework for global robustness verification of FNN named DeepGlobal, which can perform in the role of FNNs and explicitly present the decision boundaries. With the help of these boundaries, we can warn the machine learning engineers of all DeepGlobal's adversarial dangerous regions (ADRs, i.e., the input regions consisting of inputs which are susceptible to small adversarial perturbations). DeepGlobal consists of four components:

- a rule-generator which is used for mapping classification rules from output to input to find potential boundaries;
- a new network *Sliding Door Network (SDN)* that enables feasible rule-generation;

* Corresponding author.
E-mail address: sunm@pku.edu.cn (M. Sun).

[1] Activation pattern is the state about which neurons are activated during the execution of a FNN.

https://doi.org/10.1016/j.sysarc.2022.102582
Received 29 January 2022; Received in revised form 30 March 2022; Accepted 19 May 2022
Available online 28 May 2022

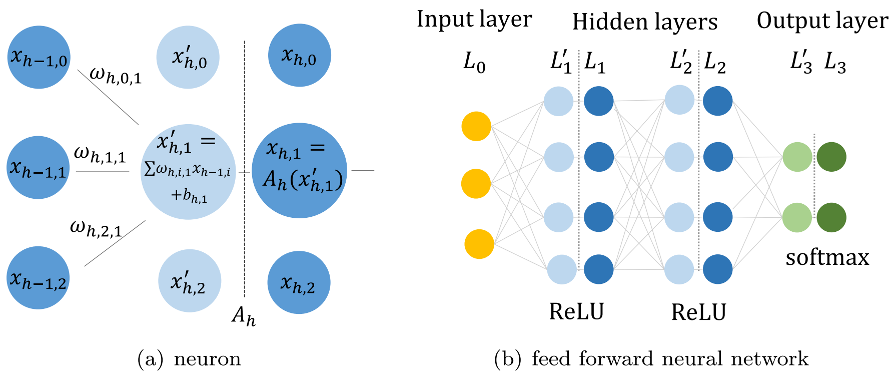


Fig. 1. Architectures of neuron and a FNN

- a selector which selects real boundaries from the generated potential boundaries;
- an ADR filter for finding the ADRs with meaningful adversarial examples.

Real boundaries are the decision boundaries of classification which divide the inputs into different classes. The ADRs can be represented as the neighborhood of real boundaries, since adversarial examples only appear around these boundaries. Then we use the ADR filter to select the ADRs with "meaningful" adversarial examples.

Particularly, we address the "two stumbling blocks" by means of the following methods, respectively. Firstly, we design a new activation function *Sliding Door Activation (SDA)*, with which the number of possible activation patterns is dramatically reduced to circumvent the complexity issue. Secondly, instead of treating inputs as the basis of robustness analysis like existing works, we cluster the input space into multiple regions to address the input space explosion challenge. To the best of our knowledge, this is the first work that can verify global robustness formally. We evaluate the effectiveness of our framework on the MNIST [20] and Fashion-MNIST [21] datasets. We also design a synthetic case study to show the feasibility of our global robustness verification framework.

The rest of this paper is structured as follows. We firstly provide the background of FNNs and ADRs in Section 2. Secondly, we introduce the naive rule-generation method in Section 3. The SDN network design and the corresponding rule-generator are elaborated in Section 4. Section 5 then presents the selection approach for SDN's rule-generation result and Section 6 shows the workflow of ADR filter. We demonstrate SDNs' effectiveness in reducing rule-generation cost and the feasibility of global verification in Section 7. Finally, we conclude and present the possible future work in Section 8.

2. Background

2.1. Feed forward neural networks

A FNN consists of an input layer, some hidden layers, and an output layer. Each layer is composed of neurons and these neurons connect the adjacent layers by weighted edges. In this paper, each neuron (layer) is treated as two virtual neurons (layers): pre-activation and activation neuron (layer), denoted by $x'_{h,i}$ (L'_h) and $x_{h,i}$ (L_h), respectively. An example is shown in Fig. 1. In Fig. 1(a), the activation neurons in $Layer_h$ are $x_{h,0}, x_{h,1}, x_{h,2}$ and the pre-activation neurons are $x'_{h,0}, x'_{h,1}, x'_{h,2}$. In Fig. 1(b), the activation layers are L_1, L_2, L_3 and the pre-activation ones are L'_1, L'_2, L'_3. The formal definition of FNN is as follows:

Definition 1 (*Feed Forward Neural Networks*). A FNN can be defined as a quaternion (L, L', W, A) where

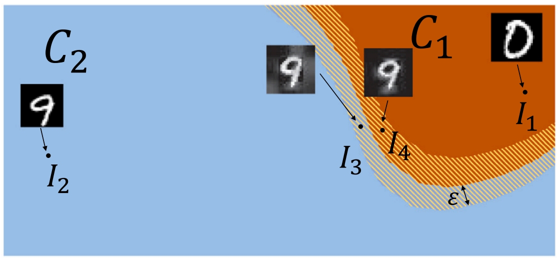


Fig. 2. A binary classification task where the inputs in orange regions C_1 are classified as "0" and the inputs in blue region C_2 are classified as "9". I_is (0<i<5) are the inputs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- $L = \{L_h \mid h \in \{0, \dots, N\}\}$ is the set of layers in which L_0 is the *input* layer, L_hs $(0 < h < N)$ are the activation hidden layers, and L_N is the activation *output* layer. The neurons in these layers are represented as $x_{h,i}$, which means the ith neuron in L_h.
- $L' = \{L'_h \mid h \in \{1, \dots, N-1\}\}$ is the set of pre-activation layers. The neurons in these layers are represented as $x'_{h,i}$, which means the ith neuron in L'_h.
- $W = \{(W_h, B_h) \mid h \in \{1, \dots, N\}\}$ include matrices of weights W_hs and bias arrays B_hs. Each W_h represents the matrix that consists of the weights $\omega_{h,i,j}$s connecting the neurons $x_{h-1,i}$ in L_{h-1} and $x'_{h,j}$ in L'_h. Each B_h represents the biases of L'_h, and the ith element of B_h is denoted by $b_{h,i}$.
- $A = \{A_h \mid h \in \{1, \dots, N\}\}$ is a set of activation functions $A_h : L'_h \to L_h$ such as ReLU or softmax.

The forward-propagation can be defined as a function whose input x is a vector:

$$F(x) = A_N(W_N A_{N-1}(\dots A_1(W_1 x + B_1)\dots) + B_N)$$

There is an ideal classification function $f : X \to C$ for every classification task, where X and C are used to represent the input set and class set, respectively. This function f maps every input to the correct class, and $F(x)$ is trained to approximate f.

2.2. Adversarial dangerous regions

FNNs have been deployed to a range of safety-critical applications which makes their robustness of high importance. A robust FNN F must satisfy the *smoothness* assumption [22], i.e., for any input x, and a small perturbation δ, $F(x + \delta) \approx F(x)$. This assumption is in line with the actual human visual capabilities. For humans, if A looks similar to B, A and B should belong to the same class.

However, FNNs are susceptible to adversarial perturbations, in other words, not robust. To introduce FNNs' susceptibility more intuitively,

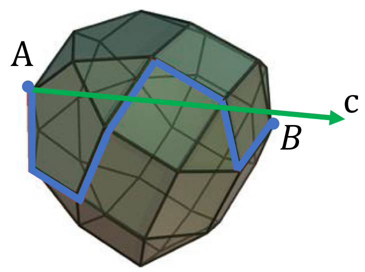


Fig. 3. Simplex algorithm starts from A and moves along the edges and finally arrives B which is the extreme point in direction c.

we use Fig. 2 to introduce a binary classification task where the inputs in orange region C_1 are classified as "0" and the inputs in blue region C_2 are classified as "9".

As shown in Fig. 2, I_1, I_2, I_3, I_4 are four inputs, and the black line between C_1 and C_2 is FNN's boundary. The input I_3 near the boundary is correctly classified as "9" by the FNN. We slightly perturb I_3 by $I_4 - I_3$. Though the result I_4 looks like I_3, I_4 is wrongly classified as "0". If we limit the size of perturbation δ to $\|\delta\|_2 \le \epsilon$, all inputs belonging to the shadow region in Fig. 2 are susceptible to adversarial perturbations. Since the shadow region is the boundary's neighborhood and the neighborhood radius is ϵ. ADRs can be defined as $\{x \mid \|x-y\|_k < \epsilon, y \in B\}$ where $k \ge 1$, ϵ is the given dangerous distance and B is the set of inputs in real boundaries. In this paper, we focus on finding all the boundaries, so as to find all ADRs.

2.3. Simplex algorithm

In mathematical optimization, simplex method [23] is a popular linear programming method. Simplex method operates on linear programs in the following canonical form

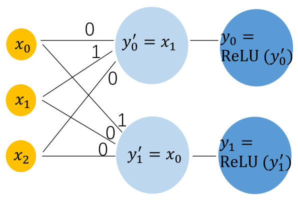
- maximize $\mathbf{c}^{\mathbf{T}}\mathbf{x}$
- subject to $A\mathbf{x} \le \mathbf{b}$ *and* $\mathbf{x} \ge 0$

where $\mathbf{c} = (\mathbf{c_1}, \dots, \mathbf{c_n})$ is the coefficients of objective function, $\mathbf{x} = (\mathbf{x_1}, \dots, \mathbf{x_n})$ is the solution, a matrix A and $\mathbf{b} = (\mathbf{b_1}, \dots, \mathbf{b_n})$ are the constraints of the solution. Any linear program can be converted into this standard form straightforwardly, so using this form results in no loss of generality.

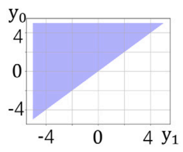
The workflow of simplex algorithm can be divided into two steps. In the first step, we find a feasible extreme point as the start point. If there is no feasible solution, the algorithm stops. In the second step, if the start point is the maximum point of the objective function, the algorithm stops and return the current point. Otherwise, there must be an edge away from the current point so that the objective function's value strictly increases. If the edge is finite, we can find another extreme point where the value of objective function is greater, or else, the objective function is unbounded and the problem has no solution. Simplex algorithm walks along the edges to find extreme points with greater and greater objective values, until it reaches the maximum value, or the unbounded edge is visited. Taking Fig. 3 as an example, $A\mathbf{x} \le \mathbf{b}$ *and* $\mathbf{x} \ge 0$ defines a polygon as a feasible region. The simplex algorithm finds a vertex A and moves along the edges of the polygon until it reaches the optimal vertex in the direction of the objective function. In this paper, we use simplex algorithm to judge whether two classification regions are connected and find the feasible input in the classification regions.

3. Naive rule-generation

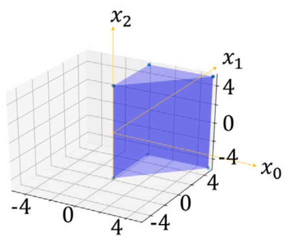
FNNs compare the output values to classify the inputs, e.g., if a output value y_k is bigger than other outputs $\bigwedge_{j \ne k} y_k > y_j$, the corresponding input belongs to class k. It is natural to use some inequations like $y_k > y_j$ to divide the input space into several regions, so as to address the input space explosion challenge. These inequations named classification rules in the input space could be achieved by the rule-generation method, as elaborated below. Before introducing the classification rules in detail, we first present a warm up example.



(a) One-layer network



(b) Region of $(y_0 > y_1)$



(c) Region of $(x_0 > 0 \wedge x_1 > 0 \wedge x_1 > x_0)$

Fig. 4. The one-layer network and regions of classification rules. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Example 1. Considering the one-layer network in Fig. 4(a), a classification rule in the output space is $(y_0 > y_1)$ which represents a blue region in output space shown as Fig. 4(b). The rule-generation method we proposed aims for mapping classification rules to the input space. For example, the activation pattern "all neurons are active" means that $(y_0 = y_0' \wedge y_1 = y_1' \wedge y_0' > 0 \wedge y_1' > 0)$. As $(y_0' = x_1 \wedge y_1' = x_0)$, $(y_0 > y_1)$ and $(y_0' > 0 \wedge y_1' > 0)$ are equivalent to $(x_1 > x_0)$ and $(x_1 > 0 \wedge x_0 > 0)$, respectively. Thus $(y_0 > y_1)$'s mapping result under this activation pattern is $(x_0 > 0 \wedge x_1 > 0 \wedge x_1 > x_0)$. If we change the activation pattern to "y_0 is active and y_1 is inactive", the equivalent condition of this activation pattern is $(y_0 = y_0' \wedge y_1 = 0 \wedge y_0' > 0 \wedge y_1' < 0)$, because ReLU assigns 0 to y_1. Thus $(y_0' > 0 \wedge y_1' < 0)$ is equivalent to $(x_1 > 0 \wedge x_0 < 0)$; $(y_0 > y_1)$ is equivalent to $(x_1 > 0)$; the mapping result is $(x_0 < 0 \wedge x_1 > 0)$. Obviously, the activation pattern determines the mapping result. The mapping result $(x_0 > 0 \wedge x_1 > 0 \wedge x_1 > x_0)$ represents a blue region in input space shown in Fig. 4(c).

With the intuition from the warm up example, we now elaborate the rule-generation for FNNs. The classification rules are some inequations recorded as $P_{h,\gamma,\eta}$ in $Layer_h$. These inequations make up the disjunctive normal form[2] $\bigvee_\gamma \bigwedge_\eta P_{h,\gamma,\eta}$ where γs are the indexes of conjunctions and ηs are the indexes of propositions, i.e., inequations in these conjunctions. We need to map the rules in output space like $\bigvee_{j \ne k} y_k > y_j$ to input space.

The mapping is divided into two parts: the output and hidden layer part. For the output layer, the comparison rules like $y_k > y_j$ can be directly mapped to the corresponding pre-activation layer based on the order-preserving activation function. Taking softmax as an example, $y_j = softmax(\sum_i \omega_{N,i,j} x_{N-1,i} + b_{N,j})$ and $y_k = softmax(\sum_i \omega_{N,i,k} x_{N-1,i} + b_{N,k})$ lead to a result:

$$y_k > y_j \Leftrightarrow (\sum_i \omega_{N,i,k} x_{N-1,i} + b_{N,k} > \sum_i \omega_{N,i,j} x_{N-1,i} + b_{N,j})$$

[2] In boolean logic, a disjunctive normal form (DNF) is a canonical normal form of a logical formula consisting of a disjunction of conjunctions; it can also be described as an OR of ANDs. For example, $(A \wedge B) \vee C$ (A, B, C are three propositions) is a DNF meaning (A and B) or C.

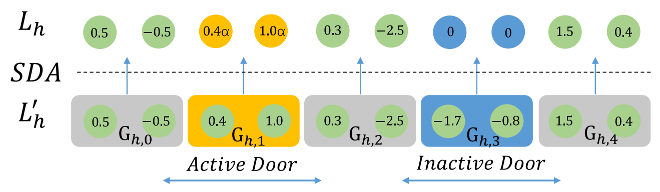


Fig. 5. Sliding Door Activation. $G_{h,1}$ and $G_{h,3}$ are active door and inactive door respectively and other activation results are the copy of other groups.

Thus, we replace every variable y_j in inequations with the corresponding polynomial $\sum_i \omega_{N,i,j} x_{N-1,i} + b_{N,j}$ to obtain the classification rules in $Layer_{N-1}$. Generally, the derived rules for the output layer can be formalized by the following function

$$MAP-OUT(y_k > y_j) = (\sum_i \omega_{N,i,k} x_{N-1,i} + b_{N,k} > \sum_i \omega_{N,i,j} x_{N-1,i} + b_{N,j})$$

and the mapping result of $\bigwedge_{j(j\neq k)} y_k > y_j$ is $\bigwedge_{j(j\neq k)} MAP-OUT(y_k > y_j)$, i.e., the classification rules in $Layer_{N-1}$. The mapping function for the hidden layer is formalized as *MAP-HIDDEN*. Hidden layers cannot be processed in the same way as output layer because the activation patterns influence the mapping result as shown in Example 1. We denote the set of active neurons' indexes by Θ and use Θ to represent the activation pattern. For simplicity, we record $P_{h,\gamma,\eta}$ as P_h and the mapping result of P_h under Θ_h as *MAP-FIX*(Θ_h, P_h) where Θ_h is the activation pattern of $Layer_h$. *MAP-FIX*(Θ_h, P_h) is the conjunction of some classification rules in $Layer_{h-1}$. *MAP-FIX* works like the rule-generation with fixed activation pattern in Example 1. The function *MAP-HIDDEN* is shown as follows where Δ_h denotes all activation patterns of $Layer_h$:

$$MAP-HIDDEN(\Delta_h, P_h) = \bigvee_{\Theta_h \in \Delta_h} MAP-FIX(\Theta_h, P_h)$$

The mapping result of $\bigvee_\gamma \bigwedge_\eta P_{h,\gamma,\eta}$ is $\bigvee_\gamma \bigwedge_\eta \bigvee_{\Theta_h \in \Delta_h}$*MAP-FIX*$(\Theta_h, P_{h,\gamma,\eta})$, i.e., all possible $Layer_{h-1}$'s classification rules which lead to the $\bigvee_\gamma \bigwedge_\eta P_{h,\gamma,\eta}$ in $Layer_h$. However, the immense time cost makes the rule-generation infeasible for FNNs. Taking a FNN with ReLU activation function as an example, each neuron has two activation states, there are 2^{m_h} activation patterns in Δ_h where m_h is the number of neurons in $Layer_h$. The time cost of whole rule-generation is $O(\Pi_h 2^{m_h}) = O(2^{\sum_h m_h})$. Thus, we present a network design which enables rule-generation in a feasible way.

4. Sliding door network for feasible rule-generation

To handle the complexity issue, we present a network design, SDN, and the corresponding rule-generation method M_{SDN}. SDN reduces the number of activation patterns by grouping the neurons in each layer to overcome the infeasibility problem in rule-generation for FNNs.

4.1. Sliding door network

Compared with FNNs, SDN has two different components: an activation function SDA and the loss function design for supporting SDA.

Sliding Door Activation. SDA takes a pre-activation layer into account and divides neurons into several groups evenly. The grouping bases on neurons' subscripts, e.g., if we divide the layer L'_h in Fig. 5 with 10 neurons into 5 groups, the adjacent neurons are in the same group and the grouping result is $G_{h,j} = \{x'_{h,i} | 2j \le i < 2j+2\}$ $(0 \le j < 5)$. Neurons in the same group behave the same way, i.e., they are in the same activation state, so as to relieve the explosion of activation pattern number, which is caused by the combination of neurons' activation states. These groups are classified by SDA into three categories: *active group* with all positive neurons (e.g., $G_{h,1}$ and $G_{h,4}$ in Fig. 5), *inactive group* in which all neurons are negative (e.g., $G_{h,3}$ in Fig. 5), and *trivial groups* with mixing of both positive and negative neurons (i.e., $G_{h,0}$ and $G_{h,2}$ in Fig. 5).

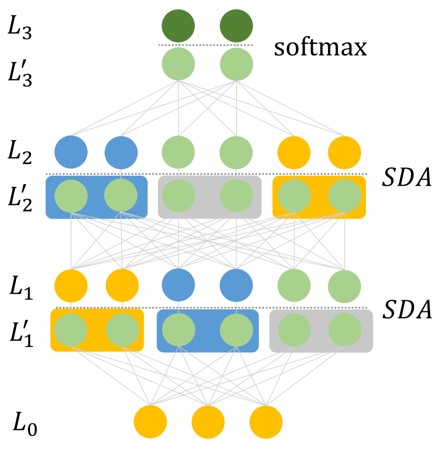


Fig. 6. Architecture of SDN.

In order to reduce the complexity, SDA selects the first active (inactive) group as *active* (*inactive*) *door* for each pre-activation layer. For example in Fig. 5, $G_{h,1}$ and $G_{h,3}$ are active door and inactive door respectively. Based on the assigned doors, we define SDA as:

$$x_{h,i} = SDA(x'_{h,i}) = \begin{cases} 0 & \text{if } x'_{h,i} \text{ belongs to inactive door;} \\ \alpha x'_{h,i} & \text{if } x'_{h,i} \text{ belongs to active door;} \\ x'_{h,i} & \text{otherwise.} \end{cases}$$

To increase the network expressiveness, SDA strengthens the active door by α and assigns 0 to inactive door's neurons. Other groups are sent to the corresponding pre-activation layer directly. During execution, for each pre-activation layer, the position of the two doors might change instantly up to the states of the groups, behaving like a sliding door, thus the name of our activation function. Fig. 6 shows the entire network architecture, replacing the ReLU in FNNs with the proposed SDA for each layer.

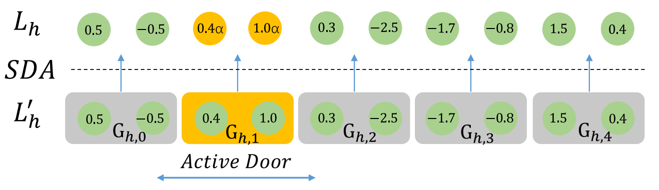
Loss function design. If a pre-activation layer cannot provide an *active* or *inactive* door, the expressiveness of SDN will be weakened. To avoid this issue, we design a regularization term to penalize the absence of either of the two doors. If the active (inactive) door does not appear in L'_h, we will find the group $G_{h,\alpha}$ $(G_{h,\beta})$ in L'_h with most active (inactive) neurons, and adjust the weights to make the negative (positive) neurons in $G_{h,\alpha}$ $(G_{h,\beta})$ tend to be positive (negative) so as to create active (inactive) groups. Thus, besides the typical data fitting loss, we add a regularization term to encourage the emergence of such groups, defined as:

$$Loss(W) = (\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_h (\sum_{x'_{h,i} \in G_{h,\alpha}, x'_{h,i} < 0} -x'_{h,i} + \sum_{x'_{h,i} \in G_{h,\beta}, x'_{h,i} > 0} x'_{h,i}))$$

where W denotes all the weights and biases to be trained, and λ is the user-given penalty parameter, $\sum_{x'_{hb_{i,h-1}i} \in G_{h,\alpha}, x'_{h,i} < 0} -x'_{h,i}$ forces $G_{h,\alpha}$ to become an active door and $\sum_{x'_{h,i} \in G_{h,\beta}, x'_{h,i} > 0} x'_{h,i}$ forces $G_{h,\beta}$ to become an inactive door.

4.2. Rule-generation for SDN

As there is no difference between FNNs' and SDNs' output layer activation functions, *MAP-OUT* can be reused for the rule-generation of SDN's output layer. Thus, we focus on mapping between hidden layers in this section. The construction process of *MAP-HIDDEN* for SDN is as follows.



L_h
0.5 −0.5 0.4α 1.0α 0.3 −2.5 −1.7 −0.8 1.5 0.4
SDA
L'_h
0.5 −0.5 $G_{h,0}$ 0.4 1.0 $G_{h,1}$ 0.3 −2.5 $G_{h,2}$ −1.7 −0.8 $G_{h,3}$ 1.5 0.4 $G_{h,4}$
Active Door

Fig. 7. Sliding Door Activation. $G_{h,1}$ is the active door and there is no inactive door.

We denote the set of neurons in the active door of layer L'_h as Θ^A_h, the set of neurons in the inactive door as Θ^I_h, and other neurons are in Θ^T_h. Considering the condition that a rule in L_h is $P_h = \sum_i c_i x_{h,i} + b > 0$, and the activation pattern Θ_h is fixed where $\Theta_h = \{\Theta^A_h, \Theta^I_h, \Theta^T_h\}$, we record the mapping result of P_h under Θ_h as MAP-FIX(Θ_h, P_h) with three components:

1. $SDNInherit$.

$$SDNInherit = (\sum_{i\in\Theta^A_h} \alpha c_i(\sum_t \omega_{h,t,i}x_{h-1,t} + b_{h,i}) + \sum_{i\in\Theta^T_h} c_i(\sum_t \omega_{h,t,i}x_{h-1,t} + b_{h,i}) + b > 0)$$

where we replace the neurons in P_h belonging to Θ^A_h with the corresponding polynomial, i.e., $\sum_t \omega_{h,t,i}x_{h-1,t} + b_{h,i}$ multiplied by α due to SDA activation, the neurons in Θ^T_h with the corresponding polynomial meanwhile remove the neurons in Θ^I_h to obtain $SDNInherit$. $SDNInherit$ is P_h's direct mapping result which does not contain the rules describing activation states.

2. $SDNActiveCon$.

$$SDNActiveCon = \bigwedge_{i\in\Theta^A_h} (\sum_t \omega_{h,t,i}x_{h-1,t} + b_{i,h-1} > 0)$$

It describes the activation state that "all the corresponding pre-activation neurons of Θ^A_h are greater than 0" and we replace these pre-activation neurons with corresponding polynomial.

3. $SDNInactiveCon$.

$$SDNInactiveCon = \bigwedge_{i\in\Theta^I_h} (\sum_t -\omega_{h,t,i}x_{h-1,t} - b_{h,i} > 0)$$

It describes the activation state that "all the corresponding pre-activation neurons in Θ^I_h are less than 0" and we replace these pre-activation neurons with corresponding polynomial.

The function *MAP-FIX* can be further represented as follows:

$$MAP-FIX(\Theta_h, P_h) = SDNInherit \wedge SDNActiveCon \wedge SDNInactiveCon$$

Taking all the activation patterns into account, we can obtain the function *MAP-HIDDEN*.

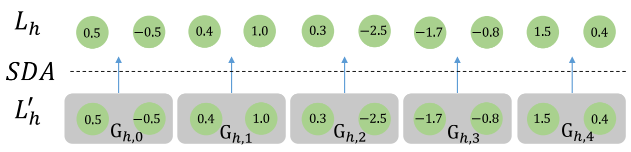
$$MAP-HIDDEN(\Delta_h, P_h) = \bigvee_{\Theta_h\in\Delta_h} MAP-FIX(\Theta_h, P_h)$$

The complete rule-generation function M_{SDN} is defined as the combination of *MAP-OUT* and *MAP-HIDDEN*:

$$M_{SDN}(\Delta_h, P_h) = \begin{cases} MAP-OUT(P_h) & h = N \\ MAP-HIDDEN(\Delta_h, P_h) & h < N \end{cases}$$

The mapping result of all the rules $\bigvee_\gamma \bigwedge_\eta P_{h,\gamma,\eta}$ in $Layer_h$ is $\bigvee_\gamma \bigwedge_\eta M_{SDN}(\Delta_h, P_{h,\gamma,\eta})$ which is the collection of rules for $Layer_{h-1}$.

For a SDN's hidden layer, the activation pattern can be divided into three categories. In the category first, the layer has two doors like Fig. 5. In this class, we select 2 doors from m_h groups where m_h is the number of groups in this layer. Thus this class has $m_h^2 - m_h$ activation patterns.



L_h
0.5 −0.5 0.4 1.0 0.3 −2.5 −1.7 −0.8 1.5 0.4
SDA
L'_h
0.5 −0.5 $G_{h,0}$ 0.4 1.0 $G_{h,1}$ 0.3 −2.5 $G_{h,2}$ −1.7 −0.8 $G_{h,3}$ 1.5 0.4 $G_{h,4}$

Fig. 8. Sliding Door Activation. $G_{h,1}$ is the active door and there is no inactive door.

In the second class, the layer has only one door like Fig. 7. This class of activation patterns select only one door from m_h groups. Thus there are $2m_h$ activation patterns in this class. The third class has no door like Fig. 8. There is only one activation pattern in this class. Thus, each $|\Delta_h|$ equals to $m_h^2 + m_h + 1$.

Compared with the naive rule-generation, the SDN has $O(\Pi_i m_i^2)$ activation patterns which is greatly less than the number of FNN's activation patterns $O(2^{\sum_h m_h})$, enabling the classification rules to be generated in a more feasible manner.

5. Selection for generated rules

This section is divided into two parts: (1) the pre-processing of generated rules; (2) the real boundaries selection approach.

5.1. Pre-processing of generated rules

The generated rules are in the form of $\bigvee_\gamma \bigwedge_\eta P_{0,\gamma,\eta}$ where each conjunction form $\bigwedge_\eta P_{0,\gamma,\eta}$ corresponds to an activation pattern of SDN. We define a strict total order $\prec$ for these conjunction forms, and assign serial numbers to the conjunction forms according to the order.

Definition 2 (*Strict Total Order $\prec$ of Conjunction Forms*)**.** Given two conjunction forms of rules R_1 and R_2 the corresponding activation patterns are Θ_1 and Θ_2. $Layer_h$ is the bottom layer where Θ_1 is different from Θ_2. $G_{h,i}$ and $G_{h,j}$ are the active doors of Θ_1 and Θ_2 in $Layer_h$ respectively. $G_{h,i'}$ and $G_{h,j'}$ are the inactive doors respectively. If an activation pattern does not have active door (inactive door) in $Layer_h$, the active door (inactive door) will be recorded as $G_{h,num}$ ($G_{h,num'}$) where *num* (*num'*) is the number of groups in $Layer_h$. $R_1 \prec R_2$ iff $i < j \vee (i = j \wedge i' < j')$.

As regions and activation patterns are in one-to-one correspondence to conjunction forms, we use the serial numbers to represent regions and activation patterns. The formal definition of activation patterns' (conjunction forms') serial number is shown as follow:

$$\Theta.num = \sum_{h=1}^{n-1}(\prod_{j=h+1}^{n-1} |\Delta.j|) * (m_h^2 + m_h - ((g_h * m_h + g'_h - 1) \textbf{ if } (g_h < g'_h) \textbf{ else } (g_h * m_h + g'_h)))$$

where n is the number of layers, h represents the index of $Layer_h$, g_h (g'_h) is the index of $Layer_h$'s active (inactive) door, and m_h is the number of groups in L'_h. Two activation patterns Θ_1 and Θ_2 satisfy $\Theta_1.num > \Theta_2.num$ iff

$$(\exists_k \forall_{h(h<k)} \Theta_1.g_h = \Theta_2.g_h \wedge \Theta_1.g'_h = \Theta_2.g'_h) \wedge (\Theta_1.g_k < \Theta_2.g_k \vee (\Theta_1.g_k = \Theta_2.g_k \wedge \Theta_1.g'_k < \Theta_2.g'_k))$$

where $\Theta_i.g_t$ ($\Theta_i.g'_t$) is the index of activation pattern Θ_i's active (inactive) door in $Layer_t$. For example, the serial number of the activation pattern Θ in Fig. 6 is 162 because the indexes of active (inactive) door in L_2 and L_1 are 2 (0) and 0 (1), respectively. If the inactive door in $L2$ moves to group 1 the serial number of new activation pattern Θ' is 161 which means $\Theta' \prec \Theta$.

In this way, these serial numbers can help us to store the rules in a B+ tree and retrieve the conjunction forms. Specially, the coverage relationship of regions is implied in serial numbers which serves as

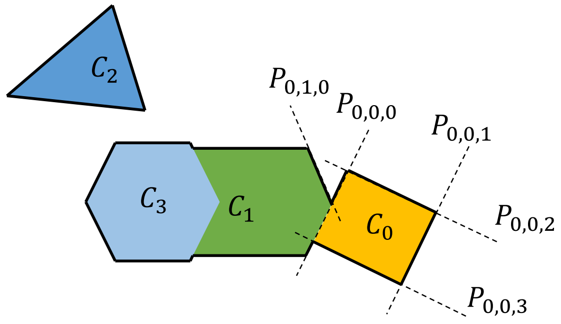


Fig. 9. The regions represented by $\vee_\gamma \wedge_\eta P_{0,\gamma,\eta} (0 \leqslant \gamma \leqslant 3)$.

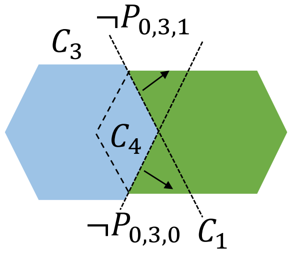


Fig. 10. C_3 is the region formed by conjunction form R_i and $C_4 \cup C_1$ is formed by R_j, the region C_4 satisfying R_j is covered by R_i's region.

Algorithm 1 Selection Approach

```
Require: an SDN $N$, classification rules $Cons$
Ensure: real boundaries set $T$
 1: $T \leftarrow Cons$
 2: $T \leftarrow$ PreProcess($T$)
 3: for $leaf$ in $T$ do
 4:     if ValidSimplex($leaf$) then
 5:         $leaf \leftarrow$ None
 6:     end if
 7: end for
 8: for $leaf$ in $T$ do
 9:     if $leaf$ != None then
10:         for potential connected $R$ in $T[:leaf.index]$ do
11:             $leaf \leftarrow$ RemoveCover($leaf$,$R$)
12:             $leaf.connect$.append($R$)
13:         end for
14:     end if
15: end for
16: for $leaf$ in $T$ do
17:     for $R$ in $leaf.connect$ do
18:         $leaf \leftarrow$ RemoveCommon($leaf$,$R$)
19:         $R \leftarrow$ RemoveCommon($R$,$leaf$)
20:     end for
21: end for
22: return $T$
```

the basis of the searching scope narrowing strategy in real boundaries selection.

5.2. Real boundaries selection approach

Before presenting the real boundaries selection approach, we need to answer two questions: (1) what is the relation between the rules and the potential boundaries of SDNs; (2) how can we select the real boundaries from the potential boundaries.

• ***Question 1: what is the relation between the rules and the potential boundaries of SDNs?***

The classification rules $\bigvee_\gamma \bigwedge_\eta P_{0,\gamma,\eta}$ represent some regions in input space as shown in Fig. 9. Each region corresponds to a conjunctive form in DNF, e.g., C_0 corresponds to $\bigwedge_{0\leqslant\eta\leqslant3} P_{0,0,\eta}$. The four inequations $P_{0,0,\eta}$ $(0 \leqslant \eta \leqslant 3)$ are boundaries of C_0. These boundaries (inequations) are the potential boundaries, namely that the rules are the potential boundaries. However, only some of potential boundaries are real boundaries indicated by solid lines in Fig. 9.

• ***Question 2: how can we select the real boundaries from the potential boundaries?***

To answer this question, we should figure out what kind of potential boundaries cannot be real. A potential boundary is not real if it is inside the regions. More specifically, given a region C and its potential boundary P_0, P_0 is not real in two cases:

1. P_0 is covered by C's connected region;
2. P_0 is the common boundary of C and C's connected region.

The first case is caused by the overlapping of regions. The regions represented by conjunction forms may cover each other as shown in Fig. 10 (Fig. 10 is a part of Fig. 9, i.e., $C_1 \cup C_3$ in Fig. 9). C_3 is the region formed by R_i and $C_4 \cup C_1$ is formed by R_j (R_i and R_j are conjunction forms like $\bigwedge_\eta P_{0,\gamma,\eta}$ in the generated rules $\bigvee_\gamma \bigwedge_\eta P_{0,\gamma,\eta}$). The dashed lines are potential boundaries in R_j, however, they are covered by C_3. That is why dashed lines are not real. The second case is shown in Fig. 9. The common boundary of C_0 and C_1 is not real, since it is inside the $C_0 \cup C_1$.

The three steps of real boundary selection are as follows:

1. ***Find the connected regions for every region.*** Both cases for a region's unreal boundaries are caused by its connected regions. Thus, we firstly use simplex method [23] to find the connected regions for every region. To judge whether the regions of R_i and R_j are connected, we apply simplex method to $R_i \wedge R_j$.[3] If the result is nonempty, R_i and R_j are connected. In addition, we narrow the searching scope for this step by Theorem 1. For every R_j, only R_is satisfying the necessary condition in Theorem 1 are possible to be connected to R_j. Thus, we do not need to search all conjunction forms for finding R_j's connected regions which greatly reduces the time cost.
2. ***Remove the covered parts of regions.*** If the covered parts of regions are removed, unreal boundaries in the first case are removed as well. For each R_i, we "flip" the potential boundaries of its connected regions and conjunct these flipped boundaries with R_i to cut the covered parts out of R_i's region. Taking Fig. 10 as an example, we flip the boundaries P_hs of C_3 to get $\neg P_h$s and conjunct each $\neg P_h$ with R_j to form new regions $\neg P_h \wedge R_j$s. Then we abandon the empty new regions and merge the others. Cut by $\neg P_{0,3,0}$ and $\neg P_{0,3,1}$, the merged new region is C_1 which does not include the covered part.
3. ***Remove the common boundaries.*** We realize this step in a similar way like step 2. For example, in Fig. 9, $P_{0,0,0}$ is the common potential boundary of C_0, C_1. If we want to remove the common part to get C_0's real boundary, we flip the boundaries of C_1 and use them to cut the dashed common part out of $P_{0,0,0}$. Then we merge the nonempty segment to get the real boundary. Cut by $P_{0,1,0}$, the solid part of $P_{0,0,0}$ is the selection result and the dashed common part is removed.

Theorem 1. *Given regions C_i and C_j $(i > j)$ with activation patterns Θ_i and Θ_j respectively, $Layer_h$ is the bottom layer where Θ_i is different from Θ_j. $G_{h,k}$ and $G_{h,t}$ are the active doors of Θ_i and Θ_j in $Layer_h$ respectively*

[3] We replace $>$ in $R_i \wedge R_j$ with $\geqslant$ to make simplex method feasible.

and $G_{h,k'}$ and $G_{h,t'}$ are the inactive doors.

C_i *and* C_j *are connected* $\Rightarrow (k \neq t \wedge k' = t') \vee (k = t \wedge k' \neq t')$

To prove Theorem 1, we need the following two definitions of the connecting relation and the common boundary, and Lemma 1.

Definition 3 (*Connecting Relation*)**.** Two regions C_i and C_j are connected in high-dimensional space iff

$$\exists_x \exists_{\epsilon>0}\{(\forall_{x' \in I}(x' \in C_i \vee x' \in C_j)) \wedge (I \cap C_i \neq \emptyset) \wedge (I \cap C_j \neq \emptyset)\}$$

where $I = \{x' | \; \|x' - x\|_\infty < \epsilon\}$.

Two regions have a common boundary means that they stick together on the boundary which is formally defined as follows.

Definition 4 (*Common Boundary*)**.** Given two rules R_i and R_j, the corresponding classification regions are C_i and C_j. A boundary B ($\sum c_i x_i + b = 0$) which belongs to C_i and C_j is defined as a common boundary iff

$$\exists_{x_0} \exists_{\epsilon_0>0}(\forall_{\mathbb{H}' \neq \mathbb{H}} \; I \cap \mathbb{H}' = \emptyset \wedge (\forall_{x \in I}(x \in C_i \vee x \in C_j)) \wedge (I \cap C_i \neq \emptyset) \wedge (I \cap C_j \neq \emptyset))$$

where $\mathbb{H} = \{x \mid \sum c_i x_i + b = 0\}$ is the set of inputs on B, $\mathbb{H}'$ is the set of points on other boundaries B' of c_i and c_j, and $I = \{x \mid \|x - x_0\|_\infty < \epsilon_0\}$.

Lemma 1. *Given two regions C_i and C_j, they are connected iff they share a common boundary.*

Proof. Given two connected regions C_i and C_j. According to the definition of connection in Definition 3, they satisfy

$$\exists_{\epsilon>0}\{(\forall_{x' \in I}(x' \in C_i \vee x' \in C_j)) \wedge (I \cap C_i \neq \emptyset) \wedge (I \cap C_j \neq \emptyset)\}$$

where $I = \{x' | \; \|x' - x_0\|_\infty < \epsilon\}$.

There must be a x in I on a boundary B which satisfies $\exists_\epsilon \forall_{\mathbb{H}_i \neq \mathbb{H}} C \cap \mathbb{H}_i = \emptyset$ where $C = \{x' | \; \|x' - x\|_\infty < \epsilon\}$, $\mathbb{H}$ and $\mathbb{H}_i$ are the point set of boundary B and B_is. Otherwise we can find a series of points $u_i (i \geqslant 0)$ and corresponding balls $I_i = \{x | \; \|x - u_i\|_\infty < \sigma_i\}$ satisfying:

- u_i is on the boundary B_i and $\forall_{\sigma_i} \mathbb{H}_{i+1} \cap I_i \neq \emptyset$
- $u_{i+1} \in I_i$, u_{i+1} on B_{i+1}, $I_{i+1} \subsetneq I_i$ and $I_{i+1} \cap \mathbb{H}_i = \emptyset$

Since we have only finite number of boundaries, here comes the contradiction. Hence we can find a u_k in I on B which is a boundary of both C_i and C_j and a corresponding I_k satisfies the above condition. As u_k is on the boundary, it satisfies $\forall_{x \in I_k}(x \in C_i \vee x \in C_j) \wedge (I_k \cap C_i \neq \emptyset) \wedge (I_k \cap C_j \neq \emptyset)$. Thus u_k satisfies

$$\exists_{\epsilon_k>0}(\forall_{\mathbb{H}' \neq \mathbb{H}_k}(I_k \cap \mathbb{H}' = \emptyset) \wedge \forall_{x \in I_k}(x \in C_i \vee x \in C_j) \wedge (I_k \cap C_i \neq \emptyset) \wedge (I_k \cap C_j \neq \emptyset))$$

and B is a common boundary shared by C_i and C_j.

The sufficient condition is obvious. □

Now we can prove Theorem 1.

Proof.

The activation patterns Θ_i and Θ_j of C_i and C_j satisfy $\Theta_i.num = i > \Theta_j.num = j$ thus

$$\forall_{t(t<k)} \Theta_i.g_t = \Theta_j.g_t \wedge \Theta_i.g'_t = \Theta_j.g'_t \wedge$$
$$\{\Theta_i.g_k < \Theta_j.g_k \vee (\Theta_i.g_k = \Theta_j.g_k \wedge \Theta_i.g'_k < \Theta_j.g'_k)\}$$

This indicates that the first change of door happens on L'_k. The change of activation pattern leads to the "mutation" of neurons in $L'_t (t > k)$. However the change of neurons in $L'_t (t \leq k)$ is continuous. According to the proof of Lemma 1, we can find a path which crosses and only crosses B. Thus, when the point on this path approaches the boundary, there is at most one neuron corresponding to B approaches 0 in $L'_t (t \leq k)$. B may come from three conditions:

- B comes from $Layer_t (t > k)$. The change of sign of ${x'}_{t,l}$ will not influence the activation pattern in L'_k which contradicts to "the first change of door happens on L'_k".
- B comes from $Layer_t$ $(t < k)$. If it changes the activation pattern of $Layer_t$, it contradicts to "the first change of door happens on L'_k". Otherwise, there must be another neuron in L'_k changes the sign at the same time which contradicts to "there is at most one neuron corresponding to B approaches 0".
- B comes from door in $Layer_k$. If the activation pattern does not change, we have the contradiction to "the first change of door happens on L'_k". If the activation pattern changes, either the index of *active door* g_k or *inactive door* g'_k becomes greater, because "there is at most one neuron corresponding to B approaches 0" meaning that the path can only breaks one door.

We eventually find out that B comes from rules of C_i by a process of elimination. In addition, the shortest path can only change one door in $Layer_k$, that is why Θ_i and Θ_j must have one same door in $Layer_k$. □

Algorithm 1 is the real boundaries selection approach. We explain Algorithm 1 to show how the selection approach works. The first part (Line 1 to 2) pre-processes the classification rules. With the help of the order in Definition 2, we assign serial numbers to conjunction forms and store them in a B+ tree T. The second part (Line 3 to 7) eliminates the conjunction forms without feasible region to reduce the time cost. We apply simplex algorithm to R_is. If the result is **None**, the corresponding region of R_i is empty, i.e., R_i is invalid. Then we assign **None** to invalid R_is to eliminate them. The third part (Line 8 to 15) corresponds to step 1 and 2. It finds the connected regions according to step 1 and uses step 2 to remove the covered parts for each valid R_i. The connected regions for each R_i are recorded in this part as well. The fourth part (Line 16 to 21) is step 3 which removes the common boundaries. If R_i shares a common boundary with its connected region, we remove the common part for both of them. Line 22 finally returns the real boundaries.

6. Filter for adversarial dangerous region

Adversarial examples only appear around real boundaries. However, not all real boundaries are surrounded by "meaningful" inputs. Some inputs like I and I' in Fig. 11 are "rubbish examples". Such "rubbish examples" cannot be recognized by human. In this section, we introduce the final component of our framework: the ADR filter (ADRF) which can judge whether there are "meaningful" adversarial examples in each ADR.

ADRF aims to find the "meaningful" sample in the ADR like I_α in Fig. 11, if I_α is "meaningful" enough, there must be another sample I_β in the other side of the real classification boundary which is "meaningful" enough as well. Because the small difference between them does not influence their visual effect, but it makes the classification result of model different. Thus, if we can find a pair of meaningful samples in the ADR and they belong to different side of the real classification boundary, we can say that the ADR has meaningful adversarial examples. Without the loss of generality, we can only find one meaningful sample on the real boundary to represent the two meaningful adversarial examples. For example in Fig. 11, I_γ can be seen as the projection of I_α and I_β on the real classification boundary. These three samples look similar, because the width (i.e., the given dangerous distance) of ADR is small. Thus, if I_γ is meaningful, the samples in ADR projecting into I_γ are also meaningful.

Given an ADR A_i and the corresponding real boundary B_i, the brief workflow of ADRF can be divided into three steps:

- Firstly, ADRF find one feasible solution x^0 in B_i;
- Secondly, ADRF optimizes x^0 to make it as "meaningful" as possible, and ensures that the result x^t is in B_i;

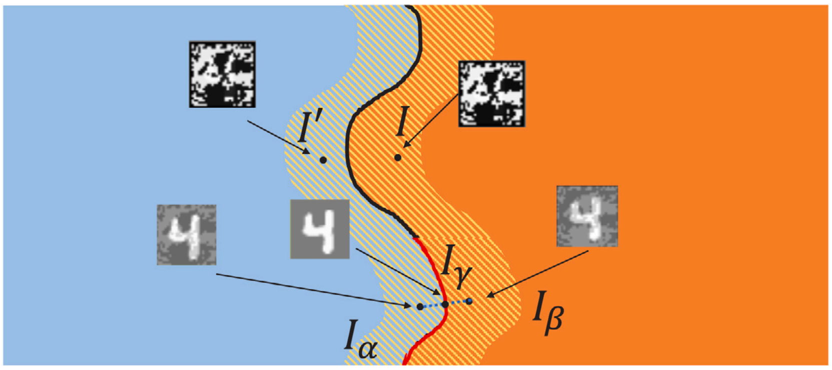


Fig. 11. In the input space of handwritten digit recognition, the "meaningful" samples can be recognized by human, the "rubbish" samples cannot.

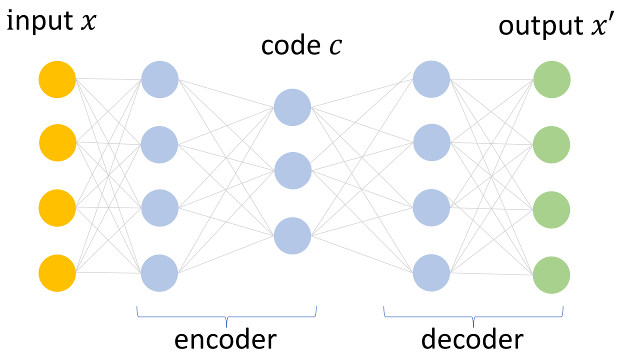


Fig. 12. The autoencoder take the input x and output the x' and c is the low-dimensional representation of x.

- Finally, ADRF can judge whether x^t is "meaningful". if x^t is meaningful we say that A_i is meaningful, otherwise, we abandon A_i.

6.1. Finding the feasible solution

In the first step of ADRF, we hope to find one feasible solution in B_i. This B_i is constrained by the conjunction form of some inequations and one equation: $B_i = \{x \mid x\ satisfies\ ((\bigwedge_\eta P_\eta) \wedge P')\}$ where P_η are inequations $\sum_i c_i x_i + b \geqslant 0$ and P' is an equation $\sum_i c_i x_i + b = 0$. Thus, the problem can be treated as a linear programming problem and we can combine the preprocess and simplex method to find B_i's feasible solution. The algorithm is shown in Algorithm 2.

To simplify the finding of feasible solution, we firstly reduce the dimension of the constraints. As there is an equation P' in the constraints, we can represent x_n by $-\frac{1}{c_n}(\sum_{i(i\neq n)} c_i x_i + b)$. The representation is in line 1 of Algorithm 2. Then from line 2 to 4, we replace the x_n in $(\bigwedge_\eta P_\eta)$ with $-\frac{1}{c_n}(\sum_{i(i\neq n)} c_i x_i + b)$. After the replacement, ADRF adds the max and min constraints for x_is from line 5 to line 11. Finally, ADRF converts the new constraints A into standard form in line 12 and uses simplex algorithm in 13 to find the feasible solution. If there is no feasible solution, the algorithm returns **None** and the corresponding rules A.

6.2. Search for meaningful samples

In the second step, we use an autoencoder [24] to guide the optimization of feasible solution. The autoencoder is a type of DNN used to learn efficient codings of data. It extracts the low dimensional feature for a set of data and ignore the unimportant noise. The framework of autoencoder is shown in Fig. 12.

Algorithm 2 Find Feasible Solution

```
Require: the conjunction form $(\wedge_\eta P_\eta) \wedge P'$
Ensure: a feasible solution $x$
 1: $x' \leftarrow -\frac{1}{c_n}(\sum_{i(i\neq n)} c_i x_i + b)$
 2: for $P_\eta$ in $(\bigwedge_\eta P_\eta)$ do
 3:     replace the $x_n$ in $P_\eta$ with $x'$ to get $P'_\eta$
 4: end for
 5: $P_\alpha \leftarrow -\frac{1}{c_n}(\sum_{i(i\neq n)} c_i x_i + b) \leqslant Max(x_n)$
 6: $P_\beta \leftarrow -\frac{1}{c_n}(\sum_{i(i\neq n)} c_i x_i + b) \geqslant Min(x_n)$
 7: $A \leftarrow (\bigwedge_\eta P'_\eta) \wedge P_\alpha \wedge P_\beta$
 8: for $x_i$ in $i \neq n$ do
 9:     $A \leftarrow A \wedge x_i \leqslant Max(x_i)$
10:     $A \leftarrow A \wedge x_i \geqslant Min(x_i)$
11: end for
12: convert $A$ into standard form $A'$
13: $x \leftarrow$ Simplex$(A')$
14: return $x$, A
```

To extract the features of each class, we use the training dataset of SDN to train an autoencoder. For each input x in the dataset, we abandon its last dimension x_n to get a new input and use these new inputs to train the autoencoder. Inspired by the domain generalization technique [25] and the regularization method [26], the loss function of autoencoder contains three parts:

- $L_{task} = \|X' - X\|_1$ is the main part of the loss function where X is a batch of input and X' is the corresponding batch of output. It aims to make the outputs equal to the inputs and deploy self-supervised training.
- $L_{reg} = \|Encoder(X)\|_1 + \alpha|T - Var(Encoder(X))|^2$ has two parts. The first part is $\|Encoder(X)\|_1$ which regularizes the code to guarantee that the extracted features are sparse. $|T - Var(Encoder(X))|^2$ is the second part which makes the latent code component's variance greater than a given threshold T over a set of sparse representations, so as to prevent the collapse in the codes.
- $L_{meta} = L_{global} + L_{local}$ has two parts L_{global} and L_{local} as well. $L_{global} = \sum_{j,k}(D_K L(s_j\|s_k) + D_K L(s_k\|s_j))$ separates the average of different classes' codes. The s_j and s_k in L_{global} is defined as

$$s_j = Softmax(\frac{1}{N_j}\sum_{X_i\ (Y_i=j)} Encoder(X_i)),$$

where N_j is the number of inputs belonging to class j in the batch, X_i and Y_i are the ith input and label in batch, respectively. L_{local} aims to make samples from the same class closer than from different classes. It is defined as

$$L_{local} = \frac{1}{N}\sum_i d_{loc}(X_i, X_{i+1}),$$

where N is the batch size and d_{loc} is defined as

$$d_{loc}(X_i, X_j) = \begin{cases} \|Encoder(X_i) - Encoder(X_j)\|_1 & Y_i = Y_j \\ max\{0, \epsilon - \|Encoder(X_i) - Encoder(X_j)\|_1\} & Y_i \neq Y_j, \end{cases}$$

The loss function can be defined as

$$Loss = L_{task} + \lambda(L_{reg} + L_{meta}).$$

With the help of autoencoder, we can calculate the average code of each class j (i.e., the class specified by rules) that

$$AC_j = \frac{1}{N_j}\sum_{i\ (Y_i=j)} Encoder(X_i).$$

The optimization of x^0 aims to make the corresponding codes close to AC_j, in other words, to make x^0 more meaningful. The optimization method is shown in Algorithm 3.

Algorithm 3 Find Meaningful Solution

```
Require: the encoder $Encoder$, the input $x^0$, the average code $AC_j$, the
    step threshold $maxstep$, the new rules $A$, the coefficient $\lambda$ and $\mu$
Ensure: a solution $x^t$
1: $t \leftarrow 0$
2: while t<$maxstep$ do
3:     $AC^t \leftarrow Encoder(x^t)$
4:     $loss \leftarrow ||AC^t - AC_j||_1$
5:     $d \leftarrow -\frac{\partial}{\partial x^t} loss$
6:     $x^{t'} \leftarrow Intersection(x^t, d, A)$
7:     $x^{t+1} \leftarrow x^t + \lambda(x^{t'} - x^t)$
8: end while
9: return $x^t$
```

In this algorithm, we optimize the input until it reaches the $maxstep$. Line 2 judges whether the optimization reaches the $maxstep$ and whether x^t goes beyond the bounds of A. In line 3, we encode x^t by the autoencoder to get the corresponding AC^t. The $\|AC^t - AC_j\|_1$ in line 4 makes AC^t and AC_j closer. Line 5 calculates the optimization direction d and Line 6 finds the intersection of the boundary and the half-line on d from x^t. Line 7 updates x^t where $\lambda < 1$ is a given learning rate and Line 9 returns the optimization result.

6.3. Judge whether the solution is meaningful

There are two ways for judging whether the input is meaningful. One is getting human involved. We can provide the candidate solutions in ADRs and let the users judge whether these inputs are meaningful, so as to select the meaningful ADRs. The other way is automatic recognition. Different tasks needs different recognition approach. In this paper, we take the computer vision tasks as an example. We utilize the perceptual similarity measurement to judge whether the feasible solution "looks like" a certain input in the training set. If the answer is yes, we decide that the solution and the corresponding ADR are meaningful.

Similarity measurements are designed to evaluate the "perceptual distance" of structured inputs, e.g., they measure how similar are pictures in a way coinciding with human judgment. There are many perceptually motivated distance metrics, such as HDR-VDP [27], SSIM [28], FSIM [29], and LPIPS [30]. In this paper, we use the Learned Perceptual Image Patch Similarity (LPIPS) to measure the similarity. The main workflow of LPIPS is shown as in Fig. 13:

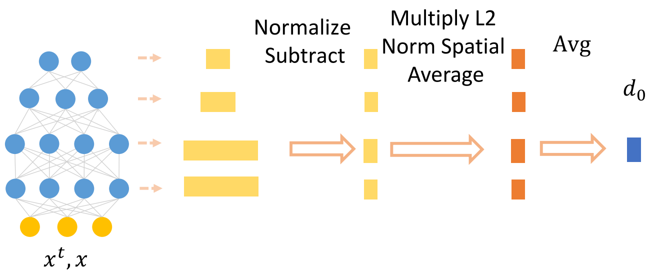


Fig. 13. The workflow of LPIPS.

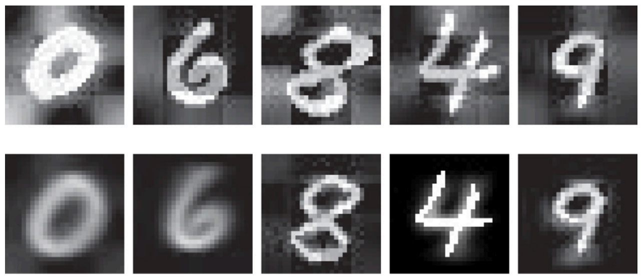


Fig. 14. Adversarial examples are in the first line and the samples in second line are classified correctly.

Given a network, LPIPS computes the deep embeddings (i.e., the values of neurons) firstly. Secondly, it normalizes the embedding in the channel dimension and scale each channel. Then it computes the L_2 distance between the results of two inputs. Finally, it averages the results across spatial dimension and across all layers. The distance can be defined as:

$$d(x^t, x) = \sum_l \sum_c \frac{w_{l,c}}{H_{l,c}W_{l,c}} \|\hat{y}^t_{l,c} - \hat{y}_{l,c}\|_2^2,$$

where $H_{l,c}$, $W_{l,c}$ and $w_{l,c}$ are the height, width and a given weight of channel c in $Layer_l$, respectively. $\hat{y}^t_{l,c}$ and $\hat{y}_{l,c}$ are x^t's and x's normalized deep embedding results in channel c of $Layer_l$. If there exists a training data x satisfying that $d(x^t, x)$ is smaller than a given bound, we say that both x^t and the corresponding ADR are meaningful.

7. Experiments

The evaluation of our work concentrates on two aspects: (1) the effectiveness of reducing rule-generation cost based on SDNs, (2) the feasibility of global verification. In the first part, we compare our method with the FNNs on *MNIST* and *Fashion-MNIST*. In the second part, we show the effectiveness of global verification.

7.1. Effectiveness of reducing rule-generation cost

We compare (1) the rule-generation cost of SDNs and FNNs with same capability (evaluated by accuracy on train set); (2) the capability of SDNs and FNNs with same rule-generation cost (evaluated by number of conjunction forms in generated rules). Table 1 and Table 2 are the comparison results on *MNIST* dataset and *Fashion-MNIST* dataset respectively. The details of SDNs are shown as follows:

- Each SDN has two hidden layers $Layer_1$ and $Layer_2$;
- These SDNs have 15, 21, 21, 31 groups in $Layer_1$, respectively and 11, 11, 16, 22 groups in $Layer_2$, respectively. We name these SDNs as (15,11), (21,11), (21,16), and (31,22) based on their architecture features;
- Each group in $Layer_1$ and $Layer_2$ has two neurons.
- The α in these SDNs are 2.

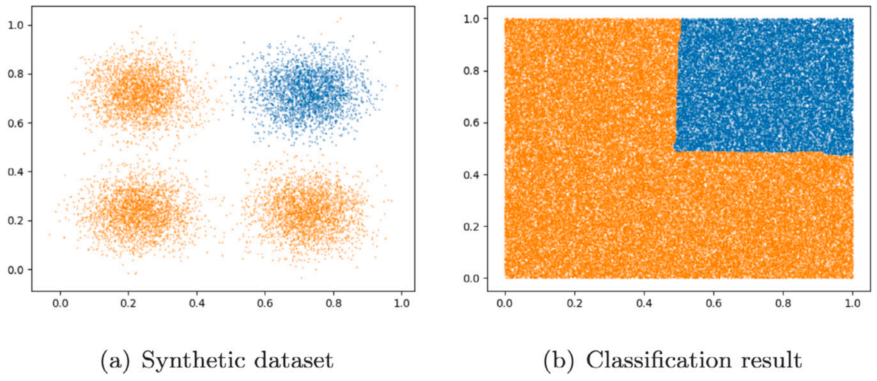


Fig. 15. Synthetic dataset and classification result.

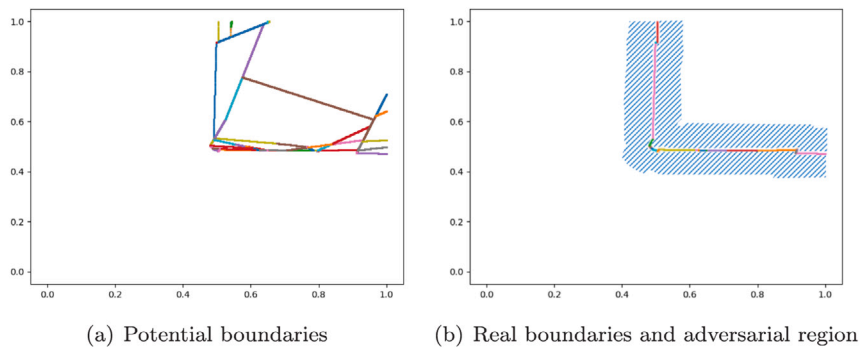


Fig. 16. Potential boundaries, real boundaries, and adversarial region.

Table 1
Evaluation on MNIST where sat-rate is the frequency of layers which can provide both active door and inactive door in evaluation.

SDN	Structure	(15,11)	(21,11)	(21,16)	(31,22)
	Accuracy(%)	94.92	95.02	95.28	95.42
	Sat-rate(%)	87.22	90.05	98.32	99.79
	Cost	32053	61579	126399	503451
FNNs (with same cost)	Structure	(8,7)	(10,6)	(10,7)	(10,9)
	Accuracy(%)	92.74	93.03	93.26	93.87
	Cost	32768	65536	131072	524288
FNNs (with same capability)	Structure	(14,14)	(15,15)	(16,16)	(17,17)
	Accuracy(%)	94.69	94.97	95.28	95.42
	Cost	$2.7 * 10^8$	$1.1 * 10^9$	$4.3 * 10^9$	$1.7 * 10^{10}$

Table 2
Evaluation on Fashion-MNIST.

SDN	Structure	(15,11)	(21,11)	(21,16)	(31,22)
	Accuracy(%)	86.04	86.35	86.42	86.51
	Sat-rate(%)	90.48	92.36	95.13	99.81
	Cost	32053	61579	126399	503451
FNNs (with same cost)	Structure	(8,7)	(10,6)	(10,7)	(10,9)
	Accuracy(%)	74.96	74.99	75.12	84.89
	Cost	32768	65536	131072	524288
FNNs (with same capability)	Structure	(15,15)	(18,18)	(19,19)	(20,20)
	Accuracy(%)	85.97	86.31	86.43	86.51
	Cost	$1.1 * 10^9$	$6.9 * 10^{10}$	$2.7 * 10^{11}$	$1.1 * 10^{12}$

Each FNN has two hidden layers. The structure is represented in a tuple, e.g., (20,20) represents a FNN with 20 neurons in two layers respectively. Cross entropy loss and Adam [31] are used to train all the networks.

The evaluation results in Table 1 show that compared with the FNNs which have roughly the same rule-generation cost, SDNs have greater capability. The accuracy of SDNs on train set are 2.18, 1.99, 2.01, 1.55 percent higher than FNNs respectively. Besides, the accuracies of SDNs and the *sat-rate* increase as the numbers of groups in each layer increase.

If the capability of SDNs and FNNs are roughly the same, the rule-generation costs of FNNs are 8374, 17436, 33979, 34124 times as SDNs'. The performance of SDNs is even better on Fashion-MNIST which is more complicated than MNIST. Compared with the FNNs with roughly the same rule-generation cost, accuracies of SDNs are 11.08, 11.36, 11.30, 1.62 percent higher respectively. Compared with the FNNs with roughly the same capability, the rule-generation costs of FNNs are 33498, 1115956, 2174684, 2183949 times as SDNs'.

Tables 1 and 2 show that SDNs greatly reduce the cost of rule-generation which makes the global robustness verification more feasible.

7.2. Feasibility of global verification

We verify the SDN (21,11) and show some adversarial examples selected from its meaningful ADRs in Fig. 14. To draw an visualized conclusion about whether our method correctly finds all the ADRs in input space, we visually demonstrate the feasibility of our approach on a two-dimensional synthetic dataset (as conclusion on high-dimensional space is hard to be visualized) shown in Fig. 15(a) where the top-right inputs belongs to the first class and others belong to the second class.

We train a SDN with 100 neurons[4] on the synthetic dataset and generate SDN's boundaries. The generation is conducted using a laptop with 1 Intel i7-9750H 2.60 GHz CPU and 1 NVIDIA Rtx 2080Max-Q GPU which takes 193.61 s. The results of the trained SDN are visually shown in Fig. 15(b) where the inputs in the top-right region

[4] Each hidden layer has ten doors and each door has five neurons. The α in this SDN is 2.

are classified as the first class and other inputs are classified as the second class.

The potential boundaries found by our method are in Fig. 16(a). These potential boundaries are the boundaries of regions. Some of potential boundaries are real and the others are inside the top right region. The real boundaries selected by our approach are in Fig. 16(b). As the synthetic dataset has no “meaning”, we do not need to select the meaningful ADRs, thus all ADRs are shown as the shadow region around real boundaries. The results show that the proposed DeepGlobal framework is effective to identify the real boundaries and the ADRs.

8. Conclusion

In this paper, we present a global robustness verifiable FNN framework DeepGlobal. To the best of our knowledge, this is the first work that provides a complete solution to achieving global robustness verification for neural networks. Based on the rule-generation, we analyze the relationship between activation patterns and classification rules, and design a new network SDN to generate rules in a feasible way. The proposed selection approach further reduces computational complexity and makes the global robustness verification more efficiently. The ADR filter can find meaningful ADRs and generate representative adversarial examples for these ADRs. Our evaluation results show that the SDN can greatly reduce the rule-generation cost. Moreover, the global verification can be achieved, which is unattainable by existing techniques. The proposed global verification framework and the SDN network design are particularly useful for safety-critical applications, especially for classification tasks that are eager for rigorous robustness.

To develop verifiable framework for large-scale networks, we plan to further reduce the verification cost in our future work. This reduction includes the pruning of unreachable activation patterns and redundant potential boundaries in rule generation, and the utilizing of regions’ connecting relation which can avoid repetitive operations in the selector and ADR filter. A more expressive network design is in our plan as well.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, Rob Fergus, Intriguing properties of neural networks, in: Proceedings of 2nd International Conference on Learning Representations, ICLR 2014, April 14-16, 2014, Banff, AB, Canada, International Conference on Learning Representations, 2014.

[2] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, Explaining and harnessing adversarial examples, in: Proceedings of 3rd International Conference on Learning Representations, ICLR 2015, May 7-9, 2015, San Diego, CA, USA, International Conference on Learning Representations, 2015.

[3] Battista Biggio, Giorgio Fumera, Fabio Roli, Security evaluation of pattern classifiers under attack, 2017, CoRR abs/1709.00609.

[4] Fnu Suya, Jianfeng Chi, David Evans, Yuan Tian, Hybrid batch attacks: Finding black-box adversarial examples with limited queries, 2019, CoRR abs/1908.07000.

[5] Nicholas Carlini, David A. Wagner, Towards evaluating the robustness of neural networks, in: Proceedings of 38th IEEE Symposium on Security and Privacy, SP 2017, May 22-26, 2017, San Jose, CA, USA, IEEE Computer Society, 2017, pp. 39–57.

[6] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami, The limitations of deep learning in adversarial settings, in: Proceedings of 1st European Symposium on Security and Privacy, EuroS&P 2016, March 21-24, 2016,SaarbrÜCken, Germany, IEEE, 2016, pp. 372–387.

[7] Matthew Wicker, Xiaowei Huang, Marta Kwiatkowska, Feature-guided black-box safety testing of deep neural networks, in: Proceedings of 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018, April 14-20, 2018, Thessaloniki, Greece, in: LNCS, vol.10805, Springer, 2018, pp. 408–426.

[8] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, Mykel J. Kochenderfer, Reluplex: An efficient SMT solver for verifying deep neural networks, in: Proceedings of 29th International Conference on Computer Aided Verification, CAV 2017, July 24-28, 2017, Heidelberg, Germany, in: LNCS, vol.10426, Springer, 2017, pp. 97–117.

[9] Matthew Mirman, Timon Gehr, Martin T. Vechev, Differentiable abstract interpretation for provably robust neural networks, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, July 10-15, 2018, StockholmsmäSsan, Stockholm,Sweden, International Machine Learning Society, 2018, pp. 3575–3583.

[10] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, Min Wu, Safety verification of deep neural networks, in: Proceedings of 29th International Conference on Computer Aided Verification, CAV 2017, July 24-28, 2017, Heidelberg, Germany, in: LNCS, vol.10426, Springer, 2017, pp. 3–29.

[11] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, A game-based approximate verification of deep neural networks with provable guarantees, Theoret. Comput. Sci. 807 (2020) 298–329.

[12] Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, Reachability analysis of deep neural networks with provable guarantees, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, International Joint Conferences on Artificial Intelligence, 2018, pp. 2651–2659.

[13] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, Martin T. Vechev, AI2: safety and robustness certification of neural networks with abstract interpretation, in: Proceedings of 39th IEEE Symposium on Security and Privacy, SP 2018, 21-23 May 2018, San Francisco, California, USA, IEEE Computer Society, 2018, pp. 3–18.

[14] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, Marta Kwiatkowska, Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance, in: Sarit Kraus (Ed.), Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019, August 10-16, 2019, Macao, China, International Joint Conferences on Artificial Intelligence, 2019, pp. 5944–5952.

[15] Weiming Xiang, Hoang-Dung Tran, Taylor T. Johnson, Reachable set computation and safety verification for neural networks with ReLU activations, 2017, CoRR abs/1712.08163.

[16] Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, Sven Dähne, Learning how to explain neural networks: PatternNet and PatternAttribution, in: Proceedings of 6th International Conference on Learning Representations, ICLR 2018, April 30 - May 3, 2018, Vancouver, BC, Canada, International Conference on Learning Representations, 2018.

[17] Marco Túlio Ribeiro, Sameer Singh, Carlos Guestrin, "Why should I trust you?": Explaining the predictions of any classifier, in: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining, KDD 2016, August 13-17, 2016, San Francisco, CA, USA, ACM, 2016, pp. 1135–1144.

[18] Scott M. Lundberg, Su-In Lee, A unified approach to interpreting model predictions, in: Proceedings of 31st Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, in: NeurIPS 2017, Information Processing Systems Foundation, 2017, pp. 4765–4774.

[19] Divya Gopinath, Hayes Converse, Corina S. Pasareanu, Ankur Taly, Property inference for deep neural networks, in: Proceedings of 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, November 11-15, 2019, San Diego, CA, USA, IEEE, 2019, pp. 797–809.

[20] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, The MNIST DATABASE of handwritten digits, 1998, http://yann.lecun.com/exdb/mnist/ (Accessed 4 January 2020).

[21] Zalando Research, Fashion MNIST: An MNIST-like dataset of 70,000 28x28 labeled fashion images, 2017, https://github.com/zalandoresearch/fashion-mnist.

[22] Ian J. Goodfellow, Yoshua Bengio, Aaron C. Courville, Deep Learning, MIT Press, 2016.

[23] George Bernard Dantzig, Linear Programming and Extensions, Princeton University Press, 1998.

[24] Diederik P. Kingma, Max Welling, An introduction to variational autoencoders, Found. Trends Mach. Learn. 12 (4) (2019) 307–392.

[25] Qi Dou, Daniel Coelho de Castro, Konstantinos Kamnitsas, Ben Glocker, Domain generalization via model-agnostic learning of semantic features, in: Proceedings of 33rd Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14, December, 2019, Vancouver, BC, Canada, Neural, Information Processing Systems Foundation, 2019, pp. 6447–6458.

[26] Katrina Evtimova, Yann LeCun, Sparse coding with multi-layer decoders using variance regularization, 2021, CoRR abs/2112.09214.
[27] Rafal Mantiuk, Kil Joong Kim, Allan G. Rempel, Wolfgang Heidrich, HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions, ACM Trans. Graph. 30 (4) (2011) 40.
[28] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, Eero P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612.
[29] Lin Zhang, Lei Zhang, Xuanqin Mou, David Zhang, FSIM: a feature similarity index for image quality assessment, IEEE Trans. Image Process. 20 (8) (2011) 2378–2386.
[30] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, Oliver Wang, The unreasonable effectiveness of deep features as a perceptual metric, in: Proceedings of 31st IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, June 18-22, 2018, Salt Lake City, UT, USA, IEEE Computer Society, 2018, pp. 586–595.
[31] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, in: Proceedings of 3rd International Conference on Learning Representations, ICLR 2015, May 7-9, 2015, San Diego, CA, USA, International Conference on Learning Representations, 2015.

Weidi Sun is currently a Ph.D. student of Applied Mathematics in Department of Information and Computational Sciences, School of Mathematical Sciences, Peking University, Beijing, China. He received the Bachelor of Science degree in Peking University in 2018. His research interest includes formal methods, deep learning safety, and the interpretability of deep learning systems.

Yuteng Lu received the B.Sc degree in applied mathematics from Peking University, Beijing, China, in 2017. He is currently working toward a Ph.D. degree in School of Mathematical Sciences, Peking University, Beijing, China. His research mainly lies in AI Safety, model checking and testing techniques.

Xiyue Zhang received the Bachelor's Degree in information and computing science from Peking University in 2017 and is currently pursuing the Ph.D. degree in applied mathematics. Her research interests include formal methods, software engineering, and deep learning.

Meng Sun received his bachelor and Ph.D. degrees in applied mathematics from Peking University, in 1999 and 2005, respectively. He spent one year as a postdoctoral researcher with National University of Singapore from 2005 to 2006. From 2006 to 2010, he worked as a scientific staff member at CWI, the Netherlands. He has been a faculty member of Peking University since 2010 and received full professorship in 2017. His research interests include software engineering, formal methods, coalgebra theory, cyber–physical systems and deep learning systems.