Using Z3 for Formal Modeling and Verification of FNN Global Robustness

Yihao Zhang*, Zeming Wei*, Xiyue Zhang*†, Meng Sun*‡
*School of Mathematical Sciences, Peking University, Beijing, China
{zhangyihao, weizeming}@stu.pku.edu.cn, {zhangxiyue,sunm}@pku.edu.cn

***Abstract*—While Feedforward Neural Networks (FNNs) have achieved remarkable success in various tasks, they are vulnerable to adversarial examples. Several techniques have been developed to verify the adversarial robustness of FNNs, but most of them focus on robustness verification against the local perturbation neighborhood of a single data point. There is still a large research gap in global robustness analysis. The global-robustness verifiable framework DeepGlobal has been proposed to identify *all* possible Adversarial Dangerous Regions (ADRs) of FNNs, not limited to data samples in a test set. In this paper, we propose a complete specification and implementation of DeepGlobal utilizing the SMT solver Z3 for more explicit definition, and propose several improvements to DeepGlobal for more efficient verification. To evaluate the effectiveness of our implementation and improvements, we conduct extensive experiments on a set of benchmark datasets. Visualization of our experiment results shows the validity and effectiveness of the approach.**

***Index Terms*—Feedforward Neural Networks, Global Robustness Verification, Social Aspects of Artificial Intelligence**

I. Introduction

Feedforward Neural Networks (FNNs) have achieved remarkable success in various fields. Despite their success, the existence of adversarial examples [3] highlights the vulnerability of FNNs and raises concerns about their safety in critical domains. Adversaries can easily deceive FNNs by introducing small and imperceptible perturbations to natural inputs, resulting in erroneous predictions. Although adversarial training [5] is considered the most effective approach for training models that are resistant to adversarial attacks, it still has a serious weakness, i.e., the lack of formal guarantees of the robustness.

To solve this problem, an avenue of research involves formally modeling and verifying the robustness of given models [4]. These methods can provide provable verification of local robustness, which pertains to specific input samples. However, simply evaluating a model's local robustness against a test set cannot provide global robustness. To explore global robustness verification, [7] proposed to approximate the globally robust radius utilizing the Hamming distance. Despite this, the approach in [7] still depends on a test set, which is not entirely satisfactory for global robustness verification.

Another inherent challenge in neural network verification is the computational complexity. The number of activation patterns, that is the potential activation status of non-linear neurons, can be of an exponential order of magnitude. Therefore, it is not practical to cover all possible patterns as the model size increases rapidly nowadays. To address this issue, existing approaches used linear relaxation [10] and abstract interpretation [2] techniques for adversarial training and verification. However, these methods all focus on local robustness. To achieve global robustness analysis, DeepGlobal [8] was proposed to facilitate global robustness verification of FNNs. It introduces a novel neural network architecture, Sliding Door Network (SDN), where all adversarial regions can be more efficiently generated. As rigorous formalization is crucial for safety verification, further steps must be taken to formally prove the global robustness of the new neural network SDN.

In this paper, we build upon the DeepGlobal framework and use the SMT solver Z3 [6] to provide a complete specification and implementation of the framework. Specifically, we provide formal definitions of DeepGlobal and algorithms with several improvements for more efficient generation of adversarial dangerous regions. To demonstrate how the Z3 solver can be applied to verify the global robustness of FNNs, we conduct extensive experiments on the MNIST and Fashion-MNIST datasets. The code is available at https://github.com/weizeming/Z3_for_Verification_of_FNN_Global_Robustness.

The paper is organized as follows. In Section II, we provide preliminaries on Feedforward Neural Networks (FNNs), Adversarial Dangerous Regions (ADRs) and Sliding Door Activation (SDA). We introduce the Z3 specification for FNNs and SDNs in Section III. In Section IV, we further show the Z3 specifications of SDNs and ADRs, which provides an explicit definition of the DeepGlobal framework. Furthermore, we present algorithmic implementation details in Section V. Section!VI concludes the paper.

II. Preliminaries

We consider a K-classification neural network $F : \mathcal{X} \to \mathcal{Y}$, which maps an input space $\mathcal{X} \subset \mathbb{R}^d$ to an output space $\mathcal{Y} = \{1, 2, \cdots, K\}$. Let $\tilde{F}(x)$ denote the ground-truth classification result for $x \in \mathcal{X}$ as determined by a human expert.

We define a FNN f as a tuple (m, N, W, A), where m is the number of layers in f, $N = (n_1, n_2, \cdots, n_m)$ is a vector specifying the number of neurons in each layer, $W = (w_1, b_1, \cdots, w_m, b_m)$ is a set of parameters for f, where $w_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and $b_i \in \mathbb{R}^{n_i}$, and $A = (a_1, a_2, \cdots, a_m)$ is a

† Current Address: Department of Computer Science, University of Oxford, Oxford, UK.
‡ Corresponding author.

DOI reference number: 10.18293/SEKE23-110

set of activation functions for each layer, where $a_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$. Thus we have $f(x) = a_m(w_m \cdots (w_1 \cdot x + b_1)) + b_m)$.

Note that the input dimension of the FNN f satisfying that $n_0 = d$, and the output dimension $n_m = K$. Given $f(x) = (f(x)_1, f(x)_2, \cdots, f(x)_K)$, the FNN returns its prediction $F(x) = \arg\max_i f(x)_i$.

Adversarial examples [9] are inputs with a small perturbation δ added to a benign sample x such that the model misclassifies the perturbed sample $F(x+\delta) \neq F(x)$. The perturbation δ is constrained by an l_p-norm ball, such as $|\delta|_p \leq \epsilon$. The concept of *Adversarial Dangerous Regions* (ADRs) is introduced to characterize global robustness. ADRs characterize the potential regions where the model's prediction is near the decision boundary and the samples in it has clear semantics. We can formally model these conditions as

$$\begin{aligned} Ad_F := \{x \mid \exists y, i \neq j : \|x-y\|_p \leq \epsilon, \\ F(y)_i = F(y)_j \geq F(y)_k (\forall k \neq i, j), \tilde{F}(y) = i\}. \end{aligned} \tag{1}$$

In the DeepGlobal framework, the SDA function is proposed to reduce the number of activation patterns in the SDNs. SDA divides the neurons in each layer $(\hat{h}_{i,1}, \hat{h}_{i,2}, \cdots, \hat{h}_{i,n_i})$ into groups of k neurons. Let the divided groups be denoted as $G_{h,0}, G_{h,1}, \cdots, G_{h,l}$, where $l = \frac{n_i}{k}$. SDA finds the first group $G_{h,Act}$, in which pre-activated neurons are all positive, from left to right. This group presents the property that each neuron within it is active and is preferred for activation. Therefore, SDA names this group the *Active Door* and multiplies it by a constant $\alpha > 1$ to stimulate the active neurons as activation. Additionally, SDA searches for an *Inactive Door* $G_{h,Ina}$ in which neurons are all negative and multiplies them by 0 to penalize the inactive neurons. The remaining $l-2$ doors are named *Trivial Doors*, which SDA neither activates nor deactivates but retains their values after activation. SDN leverages SDA to achieve comparable accuracy to general FNNs while significantly reducing the magnitude of activation patterns, making it an efficient candidate for verification.

III. Formalization of Sliding Door Networks

A. Formulation of FNNs

As the concept of SDNs is based on FNNs, we first demonstrate how to use Z3 [6] to formally model a given FNN. We assume the FNN configuration (e.g., input dimension d) has already been declared and represent each variable in the input, hidden, and output layers as a 'Real' object in Z3:

```
Input = [Real(f"x_{i}") for i in range(d)]
Hidden = [[Real(f"h_{i}_{j}") for j in range(N[i])] for i in
    range(m-1)]
Output = [Real(f"y_{i}") for i in range(K)]
```

In this way, the input and output variables are named 'x_i', 'y_i' respectively, where i indicates the i-th input (output) variable (counting from zero). For $0 \leq i \leq m-2$, the j-th hidden variable in the i-th layer is named 'h_i_j' (counting from zero), and note that the $m-1$-th layer is the output layer.

The constraints between input, hidden, and output layers highly depend on the activation patterns. Therefore, we can only model the constraints for each potential activation pattern respectively, which include four parts:

The constraints on input domain $\mathcal{X}$. Taking the MNIST dataset as example, since each pixel value is restricted to $[0, 1]$, let s be the initialized solver to be used later, we have

```
s = Solver()
s.add([Input[i] >= 0 for i in range(d)])
s.add([Input[i] <= 1 for i in range(d)])
```

We denote these constraints as C_{Input}.

The relation between adjacent layers under given activation patterns. The forward-pass from h_{i-1} (the $i-1$-th layer) to h_i (the i-th layer) can be formulated as $h_i = a_i(w_i \cdot h_{i-1} + b_i)$. For the sake of simplicity, we introduce variables '$\_h\_i\_j$' for the pre-activate neurons $\hat{h}_i = w_i \cdot h_{i-1} + b_i$:

```
_Hidden = [[Real(f"_h_{i}_{j}") for j in range(N[i])]
           for i in range(m-1)]
```

In this way, we can simplify the constraint from layer h_{i-1} to h_i with the aid of $\hat{h}_i$:

```
s.add(_Hidden[i][j] == Sum([W[i][j][k] * Hidden[i-1][k] for k
    in range(n_{i-1})]) + B[i][j])
s.add(Hidden[i][j] == a[i](_Hidden[i][j])) // pseudo-code
```

The activation condition of the given activation patterns. We defer this part in Section IV after introducing the SDA functions.

The objective property. For example, if we want to identify samples from class i, which are also near the decision boundary with class j, the constraints should be formulated as $f(x)_i = f(x)_j \bigwedge_{k \neq i,j} f(x)_i \geq f(x)_k$. This can be expressed with Z3 constraints as:

```
s.add(Output[i] == Output[j])
for k in range(K):
    if k == i or k == j:
        continue
    s.add(Output[i] >= Output[k])
```

The specification details of adversarial dangerous regions (ADRs) is presented in Section IV.

B. Formulation of SDNs

A SDN is a feedforward neural network f with the tuple (m, N, W, A, k), where m, N, and W are inherited from the definition of FNN, and $A = (a_1, \cdots, a_m)$ is the SDA function. The parameter k represents the number of neurons in each group. Let h_i denote the hidden variables in the i-th layer, with $h_0 = x$ being the input and $\hat{h}_{m+1} = f(x)$ being the output. We can recursively define the mapping f as follows:

$$\begin{aligned} &\hat{h}_i = w_i \cdot h_{i-1} + b_i, \\ &\begin{cases} Act_i = \arg\min_g \ \forall (g-1)\cdot k < j \leq g \cdot k, \quad \hat{h}_{i,j} > 0, \\ Ina_i = \arg\min_g \ \forall (g-1)\cdot k < j \leq g \cdot k, \quad \hat{h}_{i,j} < 0. \end{cases} \\ &h_{i,j} = \begin{cases} \alpha \cdot \hat{h}_{i,j}, & (Act_i - 1)\cdot k < j \leq Act_i \cdot k \\ 0, & (Ina_i - 1)\cdot k < j \leq Ina_i \cdot k \\ \hat{h}_{i,j}, & else, \end{cases} \end{aligned} \tag{2}$$

Note that the Act_i or Ina_i in (2) may not exist in some layers. In this case, SDN simply abandons the active or inactive door when mapping through these layers.

IV. Complete Modeling

A. Modeling the activation conditions

As discussed in Section III-A, the specification of FNNs depends on the Activation Patterns (AP), i.e., the different configurations of active and inactive neurons in the network. For a SDN with m layers, we define an activation pattern $\mathcal{A} = (Act_1, Ina_1, \cdots, Act_m, Ina_m)$, where Act_i and Ina_i correspond to the indices of the active and inactive doors in layer i, respectively (counting from 0 to be consistent with the code). If the active or inactive door does not exist, we fill Act_i or Ina_i with $\frac{n_i}{k}$ (the number of groups in this layer).

Therefore, given an activation pattern $\mathcal{A}$, we give the specification of activation conditions as:

```
for i in range(m):
    if Act[i] != n[i]// k:
        s.add([_Hidden[i][j] > 0 for j in range(Act[i] * k, (
            Act[i]+1) * k)])
    if Inc[i] != n[i]// k:
        s.add([_Hidden[i][j] < 0 for j in range(Ina[i] * k, (
            Ina[i]+1) * k)])
```

The constraint is denoted as $C_{AP}(\mathcal{A})$ and skipped when $Act[i]$ or $Ina[i]$ is equal to $\frac{n_i}{k}$. Note that we do not explicitly model the minimality of $Act[i]$ or $Ina[i]$, which may result in covered and common boundaries of activation regions.

The above issues are addressed by successively eliminating already-covered or common boundaries in [8]. For instance, to remove covered or common boundaries with a previous region $\bigwedge P_j$, they conjunct each $\neg P_j$ with $\bigwedge_i R_i$ to create a new region. Using this approach, we only need to consider $\neg C_{AP}(\mathcal{A}') \wedge C_{AP}(\mathcal{A})$ to remove covered and common boundaries with $\mathcal{A}'$ for $\mathcal{A}$.

B. Modeling Sliding Door Activation

Recall from Section III-A that we have modeled the linear transformation from h_{i-1} to $\hat{h}_i$. Now, we provide the formal specification of the activation function $h_i = a_i(\hat{h}_{i-1})$, which is dependent on a given activation pattern $\mathcal{A}$.

```
for i in range(m):
    for j in range(n[i]// k):
        if Act[i] == j:  # Active Door
            s.add([Hidden[i][j+1] == alpha * _Hidden[i][j] for
                l in range(k)])
        elif Ina[i] == j:  # Inactive Door
            s.add([Hidden[i][j+1] == 0 for l in range(k)])
        else:  # Trivial Door
            s.add([Hidden[i][j+1] == _Hidden[i][j] for l in
                range(k)])
```

We denote this set of constraints (including the constraints on linear mappings) as $C_{Forward}(\mathcal{A})$.

C. Modeling the Adversarial Dangerous Regions

Recall our refined definition of ADRs in Section II, where we aim to find feasible y that satisfies the *boundary condition* (i.e., $\exists i \neq j$ such that $\forall k \neq i, j.\, F(y)_i = F(y)_j \geq F(y)_k$) and the *meaningful condition* (i.e., $\tilde{F}(y) = i$). In Section III-A, we present the Z3 specification for the boundary condition, which we denote as $C_{Boundary}(i, j)$. [8] attempt to find feasible and meaningful solutions in the ADRs instead of considering the meaningful condition. Specifically, a trained autoencoder [1] is used to optimize a feasible solution x^0 in a given ADR, while ensuring that it remains within the same ADR. However, this optimization-based method has some limitations. For instance, the meaningful solution may not always exist for all ADRs, which is a possible scenario when all samples in the region are deemed "rubbish". Additionally, optimizing the solution along certain directions within the region can be extremely time-consuming.

Therefore, we propose a new approach that allows for more straightforward identification of meaningful samples. Note that the *meaningful condition* is $\tilde{F}(y) = i$. While judging each sample by $\tilde{F}$ (*i.e.*, human-perception) is not practical, we can still use autoencoders as surrogate models.

For a given class $i \in \{1, 2, \cdots, K\}$, we hope to find a meaningful region by the surrogate model AE where $\tilde{F}(y) = i$. To achieve this, we train an autoencoder $E(\cdot)$ and leverage it to define the center of class i as $c_i = \frac{1}{|X_i|} \sum_{x \in X_i} E(x)$, where $E(\cdot)$ is the encoder function, and X_i represents the samples in the training set with class i, and define the *prototype* of class i as $P_i = D(c_i)$. The prototype model for class i is decoded from the average code of samples in that class, making it a standard representation of that class. Our assumption is that any meaningful sample y with $\tilde{F}(y) = i$ should not be significantly different from the prototype P_i. To ensure this, we restrict y to a *meaningful region* $|y - P_i|_p \leq r$, where r is a pre-specified radius. It's worth noting that the definition of the meaningful region is fundamentally different from that of adversarial examples (see Section II), where the perturbation δ is limited to a specific bound ϵ. Generally, r is much larger than ϵ, as all samples in this region are close to the prototype and potentially meaningful. The definition of adversarial examples is more restrictive than that of meaningful regions, as it only focuses on a small perturbed region.

Based on the above analysis, taking l_∞-norm as example, we specify the meaningful condition as follows:

```
for i in range(d):
    s.add([Input[i] - P[i] < r, P[i] - Input[i] < r])
```

We denote this set of constraints as $C_{Meaningful}(i)$. So far, we have completed all specifications for DeepGlobal in Z3. To find the feasible and meaningful sample y in the *target class* i, which is on the decision boundary of the *boundary class* j, with regard to activation pattern $\mathcal{A}$, one only need to solve the following constraints in Z3:

$$C_{Input} \wedge C_{AP}(\mathcal{A}) \wedge C_{Forward}(\mathcal{A}) \wedge C_{Boundary}(i, j) \wedge C_{Meaningful}(i).$$

V. Algorithmic Implementation Details

In this section, we demonstrate the implementation of using Z3 solver to specify the DeepGlobal framework and identify global adversarial regions.

Algorithm 1: Find feasible and meaningful solutions

Input: SDN Network $f = (m, N, W, A, k)$; Target class i; Boundary class j

Output: Feasible and Meaningful solutions

1 $Initialize\ C_{Input},\ C_{Boundary}(i,j),\ C_{Meaningful}(i)$;
2 $C_{Checked} \leftarrow \emptyset$;
3 $Solutions \leftarrow \emptyset$;
4 **for** *All valid AP* $\mathcal{A}$ **do**
5 $s \leftarrow new\ Z3\ solver$;
6 $s.add([C_{Input},\ C_{Boundary}(i,j),\ C_{Meaningful}(i)])$;
7 $s.add(\ \neg C_{Checked}\ )$;
8 $s.add([C_{AP}(\mathcal{A}),\ C_{Forward}(\mathcal{A})]$;
9 **if** $s.solve() == sat$ **then**
10 $Solutions.Append(s.model())$;
11 **end**
12 $C_{Checked} \leftarrow C_{checked} \vee C_{AP}(\mathcal{A})$;
13 **end**
14 **return** $Solutions$;

To find samples y that belong to class i and are on the decision boundary of class j (*i.e.*, $F(y)_i = F(y)_j \geq F(y)_k (\forall k \neq i, j)$), we need to enumerate all target-boundary class pairs (i, j), which form a complete set of samples supporting the ADRs and are referred to as *boundary samples*.

Algorithm 1 presents a complete workflow for this implementation. Line 1 initializes the input, boundary, and meaningful constraints, which are shared for each valid activation pattern. In line 2, we use $C_{checked}$ to track the regions that have already been checked in previous activation patterns to avoid redundancy, as described in Section IV-A. The $Solutions$ list in line 3 stores the solved feasible and meaningful samples. In lines 4-8, we create a Z3 solver s for each valid activation pattern $\mathcal{A}$ and add the required constraints to it. If the constraints can be solved, we append the generated sample to $Solutions$ as shown in lines 9-11. A checked region is added to $C_{checked}$ in line 12 to avoid solving it again for other activation patterns. The algorithm returns all feasible and meaningful solutions for target class i and boundary class j.

We now discuss the details for implementing enumeration of activation patterns in line 4 of Algorithm 1. Recall that there are $\frac{n_i}{k}+1$ possible values for Act_i and Ina_i, respectively. The unique constraint on Act_i and Ina_i is $(Act_i \neq Ina_i) \vee Act_i = \frac{n_i}{k}$, since any group cannot be both active and inactive door, except one case that $Act_i = Ina_i = \frac{n_i}{k}$, *i.e.*, the groups are neither activated nor inactivated. We arrange all activation patterns in a tree structure. In this way, we can implement the enumeration of activation patterns by breadth-first searching and execute from the shallow layers to the deep layers.

The experiment includes two parts: the utilization of autoencoders and the generation of boundary and adversarial examples. Autoencoders are employed to generate prototypes for each dataset that represent meaningful samples with explicit semantics. The prototypes can be used for global verification, distinct from instance-wise local robustness. Boundary samples were produced for each class by identifying samples that are situated on the decision boundary between that class and the adjacent class. Adversarial examples were generated from both the exact and relaxed boundary regions. Starting from the boundary samples, perturbations were added to create adversarial examples. More details can be found in https://arxiv.org/abs/2304.10558.

VI. Conclusion

In this paper, we provide a complete and refined definition of SDNs and ADRs in the DeepGlobal framework. We then present a complete specification of the framework using the SMT solver Z3 and demonstrate its detailed algorithmic implementation. Additionally, we leverage prototypes crafted by autoencoder to improve the verification framework by searching for meaningful solutions. The experiments on two benchmark datasets show that increasing the activation coefficient α will lead to better model performance. Besides, the proposed specification support the generation of extensive boundary and adversarial samples, which can be used for identifying global ADRs of a given model. The selected customized tactics in Z3 further improve the effectiveness of our framework.

Acknowledgement

This research was sponsored by NSFC under Grant No. 62172019, and CCF-Huawei Populus Grove Fund.

References

[1] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[2] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018. doi:10.1109/SP.2018.00058.

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

[4] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[5] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[6] L. d. Moura and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[7] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance. International Joint Conferences on Artificial Intelligence Organization, 2019.

[8] W. Sun, Y. Lu, X. Zhang, and M. Sun. Deepglobal: A framework for global robustness verification of feedforward neural networks. *Journal of Systems Architecture*, 128:102582, 2022.

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

[10] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019.