Testing-based formal verification for ReLU-based neural networks

Haiyi Liu
Hiroshima University
d200101@hiroshima-u.ac.jp

Shaoying Liu
Hiroshima University
sliu@hiroshima-u.ac.jp

Ai Liu
Hiroshima University
liuai@hiroshima-u.ac.jp

Yujun Dai
Hiroshima University
d201609@hiroshima-u.ac.jp

Abstract

Since neural networks are widely used, how to ensure the reliability of neural networks has become a hot research topic. The first difficulty is that it is difficult to give the pre-condition and post-condition for neural networks, and the second difficulty also exists in traditional software, i.e., the problem of exploding execution paths. Fortunately, the output range of a neural network is easier to give and when an input is given, we can obtain the activation order of neurons in the neural network. Based on the above facts, we propose DeepTBFV, a method for pre-trained neural networks, which uses a testing-based formal verification algorithm to derive the pre-condition of the neural network on a specified path by post-condition. The purpose is to verify and explain the behavior of the neural network.

1. Introduction

Although neural network has been widely used in many fields such as NLP (natural language processing), image processing and even MMML（Multi-modal Machine Learning）, its poor reliability has also been criticized by users for a long time. Specifically, there are two aspects. One is that the neural network is difficult to be verified. The reason is that the architecture of the neural network is based on experience, and the parameter is constructed by back-propagation of the training data. It is difficult to give a formal specification like traditional software, and the verification of the neural network is an NP-hard problem[1], which makes it difficult to achieve complete verification of the large models. The second is that the neural network is difficult to be explained, that is, it is difficult for us to figure out what features the result of neural network reasoning is based on. For instance, although a neural network correctly recognizes the cat in the picture, we cannot determine whether the neural network correctly recognizes the cat through its features or the watermark in the picture.

Generally, the post-condition of a neural network is easy to give. If the pre-condition can be derived from the neural network by using the post-condition, then users can use the pre-condition to explain and verify the reliability of the neural network. However, since the verification of neural networks is an NP-hard problem, it is very difficult to derive the preconditions from the post-conditions simply by using the verification algorithm.

Existing methods mainly focus on how to accelerate the verification speed of neural network under the premise of given formal specification [2, 3, 4, 5]. These studies provide a basis for the verification of neural networks, but do not discuss how to give a pre-condition to explain neural networks. Test-based neural network interpretation algorithm can explain the behavior of the local neural network by testing and constructing the local interpretation model of the neural network [6, 7, 8]. But they cannot be used to formally verify the reliability of neural networks. In other words, it can not be guaranteed that there is no adversarial examples in the neural network within a certain range.

In this paper, We propose *DeepTBFV*, a method to give the pre-condition of neural network by backward derivation of the post-condition of neural network. *DeepTBFV* is inspired by Testing-based formal verification (TBFV) [9, 10, 11] in traditional software. First, we select a test case that can be correctly recognized by the neural network and execute it. During the execution of the test case, the activation of each neuron is recorded. Here, we compare the activation sequence of neurons to the execution path of the test case in traditional software. Therefore, recording the activation sequence of neurons is the execution path of the test case generated by the neural network. Then, according to the execution results of the neural network, the post-condition of the test case is given, which is used to derive the pre-condition.

Finally, we combine the post-condition and the execution path on the neural network to form a system of linear inequalities with multiple variables. The pre-condition of the neural network is obtained by solving the system of linear inequalities with multiple variables.

In brief, the main contributions of this paper are as fellows:

- To my best knowledge, we develop the first testing-based formal verification method for DNNs.
- We design a verification based neural network retraining algorithm.

2. Preliminary

2.1. Testing-based formal verification

The testing-based formal verification (TBFV) is proposed to ensure the correctness of all traversed program paths in traditional software. The first step of TBFV is to generate a test case T based on the test condition in the formal specification. The second step is to obtain a traversed program *path* by executing the test case T execution program P, where the *path* contains a series of conditions. The third step is to verify the reliability of the *path* under the formal specification by using symbolic execution or Hoare logic [12].

2.2. Floyd-Hoare Logic

Floyd Hoare logic, also known as Hoare logic, represents predicate logic and a set of axioms in the form of Hoare triples, and then defines the semantics of the programming language. The specific form of Hoare triples is as follows:

$$\{pre\}\ c\ \{post\}$$

where c is a specific program code, *pre* indicates the preconditions in the program, which describes the program state before executing c, and *post* indicates the post condition in the program, which describes the programs the program state after executing c. Such a Hoare logic triple indicates that if an input of the program c meets its *pre*, the output of c should meet *post* after executing program c. Otherwise, program c must have errors.

In order to formalize the program, Hoare logic defines inference rules for each of grammar clauses. The detailed rules are as follows.

$$Assign\ \ \frac{}{\{Q(E/x)\}\ x := E\ \{Q\}}$$

The rules is to perform the assignment statement $x := E$ on all x in the precondition $\{Q(E/x)\}$, that is, to replace all x in the precondition with E to obtain the post condition $\{Q\}$. Similarly, the following two rules can exist.

Preconditions reinforcement rule:

$$Pre\ \ \frac{P \Rightarrow P' \quad \{P'\}c\{Q\}}{\{P\}c\{Q\}}$$

Post conditional weakening rule:

$$Post\ \ \frac{\{P\}c\{Q'\} \quad Q' \Rightarrow Q}{\{P\}c\{Q\}}$$

By formalizing the program with these rules, we can get a series of constraints of the program, and then use SMT solver. Solve the corresponding rules.

2.3. Verification of neural networks

Deep neural network is a mapping in high-dimensional space, which can be formally expressed as $f : \mathbb{R}^n \to \mathbb{R}^m$.

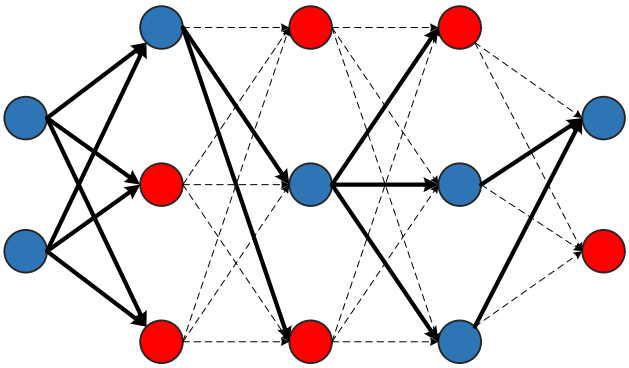

Figure 1. Execution path generation

if there is a set of constraints ϕ which is the precondition of f on $\mathbb{R}^n$, and existence a set of constraints φ which is the post condition of f on $\mathbb{R}^m$. Then, the problem of neural network verification is transformed into proving that $\forall x \in \mathbb{R}^n : \phi(x) \rightarrow \varphi(f(x))$ is satisfied or not.

3. Methodology

In this section, we propose an algorithm to derive the pre-condition of a pre-trained neural network from its post-condition, named DeepTBFV. The pre-condition can be used to verify and explain the behavior of the neural network and assist developers to increase the reliability of the neural network. First, we forward propagate a test case on a neural network, and generate the execution path of the test case. Second, we propose a technique to encode the execution path as a multivariate linear inequality system, and since the activation state of neurons is fixed in the execution path, the linear constraints generated by the execution path do not need to introduce the relaxation variables. Finally, we use the classification results obtained from this test case to define the post-condition of the test case and derive the pre-condition in reverse. The process of defining the post-condition is based on an assumption that will be described in the third summary.

3.1. Testing-based neural network execution path generation

The execution of a test case by the neural network means that the test case is used as the input for forward propagation. In the process of forward propagation, the state of each neuron is fixed. That is, we consider a neuron n_i in a neural network NN to be activated for the test case if its output is greater than a threshold value. Conversely, if n_i is less than the threshold, then the neuron n_i is not activated. We record the activation state of each neuron when the NN forward propagate for the test case.

Formally, we define all neurons in NN as the set $neurons$. Then, the neurons activated in NN can be defined as the set $ActiveN = \{n \mid n \in neurons \wedge Out(n) > \theta\}$, where θ is the threshold in neurons and $out(n)$ record as the output value of neuron. Similarly, the inactive neurons in NN can be defined as the set $InactiveN = \{n \mid n \in neurons \wedge Out(n) \leq \theta\}$. as shown in figure 1, in the forward propagation of neural network, we use blue for the neurons activated in the test case and red for the inactive neurons. Then, the process of assigning activation states to the neuron in $neurons$ through the test cases is called execution path generation.

3.2. Formal modeling of neural network pathes

Since the state of activation of each neuron is fixed, we can convert the state of each neuron into a multiple linear inequality. Let $Neuron_j^i = \{x_1^{i-1}, x_2^{i-1}, \cdots x_n^{i-1}\}$ is the set of inputs of the j-th neuron in the i-th layer of the neural network NN. If the activation function of Neuron is ReLU and is activated by the test case, the activation state of the neuron can be modeled as follows,

$$If : Neuron_j^i \in ActiveN$$

$$Then : Out(Neuron_j^i) = w_1^{i-1} \cdot x_1^{i-1} \cdots + w_n^{i-1} \cdot x_n^{i-1} > 0$$

Similarly, if the neuron is not activated by the test case, then the activation state of the neuron can be modeled as follows,

$$If : Neuron_j^i \in InactiveN$$

$Then : Out(Neuron_j^i) = w_1^{i-1} \cdot x_1^{i-1} \cdots + w_n^{i-1} \cdot x_n^{i-1} \leq 0$

According to the above rules, we can model the process of forward propagation of test cases in the neural network into a system of multivariate linear inequalities. Algorithm 1 specifically describes how to model the execution path of nn.

Algorithm 1: Formal modeling of NN pathes

Input: $Neurons$; $ActiveN$; $InactiveN$
// List of linear constraints corresponding to neural network
Output: $Constraint$

```
Constraint = list()
for neuron in Neurons do
    if neuron in ActiveN then
        Constraint.append(Out(neuron)>0)
         else
            Constraint.append(Out(neuron) ≤ 0)
        end
    end
end
return Constraint
```

3.3. Deriving Preconditions from Postconditions

When the state of each neuron in the neural network is fixed, we only need to carry out the Floyd-Hoare Logic assignment statement for the constraint of each neuron. This means that the constraints of each neuron are replaced by variables through the assignment statement. The constraints of each neuron we can get only include the input of the neural network.

Specifically, let $Inp = \{x_1, x_2, \cdots x_n\}, Inp \in R^n$ as the input of the NN and $Out = \{y_1, y_2, \cdots y_m\}, out \in R^m$ as the output of the NN. Moreover, We record the replacement expression of the j-th neuron in the i-th layer as f_j^i. Then the constraint corresponding to neurons can be denoted as

$$f_j^i(x_1, x_2, \cdots x_n) \leq 0$$

or

$$f_j^i(x_1, x_2, \cdots x_n) > 0$$

The constraints of the output layer of the neural network are different from those of the middle layer. It should conform to the user-defined post-condition. Here, the constraints of neurons in the output layer can be denoted as,

$$\alpha_j \leq y_j(x_1, x_2, \cdots x_n) \leq \beta_j$$

Where $y_j(x_1, x_2, \cdots x_n)$ is a multivariate linear polynomial.j represents the j-th output of the output layer. The interval $O_j = [\alpha_j, \beta_j]$ means the post-condition of y_j.

In addition, we also propose a method to construct a post-condition for the path generated by test cases. If the test case activates a neuron in the output layer, it means that the output value of the activated neuron is greater than that of other inactive neurons. Therefore, we can formally record this statement as $y_j^a > y_i^{in}$, where $i \in \{1, 2 \cdots m\} \backslash \{j\}$, a represents activated neuron, in represents inactive neurons. The activated neurons are marked as j. Since the output value of a neuron has an upper and lower limit, such as $INT8$ quantization of a neural network, the output values of neurons are quantized to the $[-128, 127]$. Therefore, we can also give the upper and lower limits of the output value of neurons in the output layer. Let the upper and lower bounds of the output layer neurons be $[l, u]$, where the l denote upper bound, and u denote lower bound. Then,

$$\forall y_i \in Out, \exists\ \alpha_i \leq y_i \leq \beta_i$$

where $i \in \{1, 2, \cdots m\}$, and m is the number of output layer.

4. Case Study

In this section, we use a three-layer neural network to show how DeepTBFV works and to demonstrate its effectiveness. Consider a ReLU-based neural network G. The structure and weights of this network are shown in Fig 2. Without loss of generality, we can assume that $bias = 0$ for each neuron in G. We denote the

neurons of the input layer, the neurons of the hidden layer and the neurons of the output layer as $\{x_1, x_2\}$, $\{h_1, h_2, h_3\}$, $\{y_1, y_2\}$ respectively.

To execute DeepTBFV for G, all the steps are given in sequence as follows:

Step 1: Select the test case of interest as input to the neural network, suppose we use $x_1 = 1$, $x_2 = 2$ as the test case.

Step 2: Record the activation of each neuron in the hidden layer of the neural network,

$$h_1 = ReLU(-x_1 + 2x_2 = 3) \in ActiveN$$

$$h_2 = ReLU(2x_1 - x_2 = 0) \in InactiveN$$

$$h_3 = ReLU(x_1 + x_2 = 3) \in ActiveN$$

Step 3: Generate neural network middle layer constraints,

$$h_1 = -x_1 + 2x_2 > 0$$

$$h_2 = 2x_1 - x_2 \leq 0$$

$$h_3 = x_1 + x_2 > 0$$

Step 4: Generate output layer constraints, here we assume that the output layer has constraints,

$$5 < y_1 = h_1 - h_2 + h_3 < 7$$

$$3 < y_1 = h_1 + 2h_2 + 0.5h_3 < 5$$

Step 5: Through Floyd-Hoare Logic, the system of inequality equations for each neuron in the input layer is derived backwards,

$$2x_1 - x_2 \leq 0$$

$$-x_1 + 2x_2 > 0$$

$$-x_1 - x_2 < 0$$

$$3 < -0.5x_1 + 2.5x_2 < 5$$

$$5 < 3x_2 < 7$$

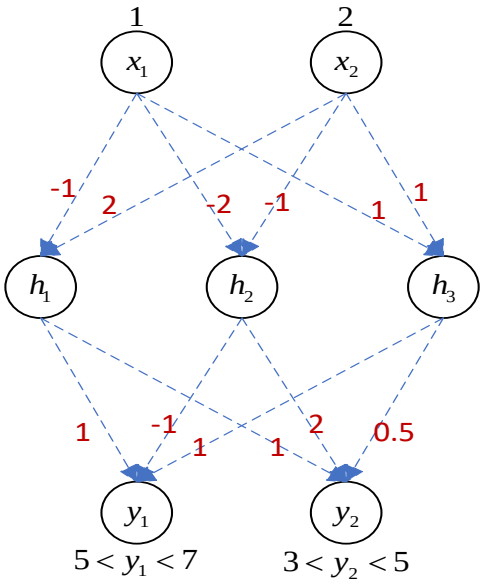

Figure 2. Example showing How to execute DeepTBFV

5. Related Work

In recent years, with the extensive application of deep learning, its reliability and interpretability have been increasingly concerned by the security community. Heuristic algorithms provide a fast and efficient way to interpret neural networks.

The heuristic algorithm provides a fast and effective way to explain the neural network. LIME[13] uses a testing-based method combine regression to explain the local characteristics of neural networks. Similar algorithms are anchors, etc[14, 15, 16]. Although heuristic algorithms can provide quick explanations, the resulting explanations are not guaranteed by formal verification.

Meanwhile, rigorous formal verification provides another perspective to ensure reliability and provide explanations. The work of reluplex et al.[17, 18, 19] performs formal verification of neural networks based on specification. It can guarantee that the verified neural network is safe and secure under formal specification.

However, reluplex[**?**] also proved that the verification problem of neural networks is NP hard. And the interpretation of verification-based neural networks is likewise based on verification solvers. Therefore, Trustable XAI et al.[20, 21] faces the same problem of difficulty in interpreting large neural networks.

6. Conclusion

We propose DeepTBFV, a test-based approach to generate execution paths for a test case in a neural network and derive the pre-condition(linear constraints) of the input to the neural network backwards by means of a custom post-condition. To my best knowledge, we are the first to apply the idea of TBFV to the field of verification and interpretation of neural networks.

7. Future Work

In this paper, we focus on how to use the principle of TBFV to generate the pre-condition of neural network. However, it did not analyze in detail how to generate test cases and how to use the generated constraints to verify or explain the behavior of neural networks. In future work, we will focus on the following two parts:

- How to choose the appropriate test case so that the constraints generated by DeepTBFV for that test case are more locally interpretable.
- How to analyze the constraints generated by DeepTBFV so that the constraint results can help people intuitively understand the neural network.

8. The Acknowledges

This authors acknowledge funding from the JST SPRING, Grant Number JPMJSP2132.

References

[1] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J. ‘Reluplex: An efficient smt solver for verifying deep neural networks’. In: Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30. (Springer, 2017. pp. 97–117

[2] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S. ‘Formal security analysis of neural networks using symbolic intervals’. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). (, 2018. pp. 1599–1614

[3] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R. ‘Efficient verification of relu-based neural networks via dependency analysis’. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34. (, 2020. pp. 3291–3299

[4] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., et al. ‘The marabou framework for verification and analysis of deep neural networks’. In: Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31. (Springer, 2019. pp. 443–452

[5] Shriver, D., Elbaum, S., Dwyer, M.B. ‘Dnnv: A framework for deep neural network verification’. In: Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33. (Springer, 2021. pp. 137–150

[6] Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R. ‘Deepconcolic: Testing and debugging deep neural networks’. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). (IEEE, 2019. pp. 111–114

[7] Pei, K., Cao, Y., Yang, J., Jana, S. ‘Deepxplore: Automated whitebox testing of deep learning systems’. In: proceedings of the 26th Symposium on Operating Systems Principles. (, 2017. pp. 1–18

[8] Goodfellow, I.J., Shlens, J., Szegedy, C.: ‘Explaining and harnessing adversarial examples’, *stat*, 2015, **1050**, pp. 20

[9] Liu, S. ‘Testing-based formal verification for theorems and its application in software specification verification’. In: Tests and Proofs: 10th International Conference, TAP 2016, Held as Part of STAF 2016, Vienna, Austria, July 5-7, 2016, Proceedings 10. (Springer, 2016. pp. 112–129

[10] Wang, R., Liu, S., Sato, Y.: ‘Sit-se: a specification-based incremental testing method with symbolic execution’, *IEEE Transactions on Reliability*, 2021, **70**, (3), pp. 1053–1070

[11] Liu, A., Liu, S.: ‘Enhancing the capability of testing-based formal verification by handling operations in software packages’, *IEEE Transactions on Software Engineering*, 2022, **49**, (1), pp. 304–324

[12] Pratt, V.R. ‘Semantical considerations on floyd-hoare logic’. In: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976). (IEEE, 1976. pp. 109–121

[13] Ribeiro, M.T., Singh, S., Guestrin, C. ‘” why should i trust you?” explaining the predictions of any classifier’. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. (, 2016. pp. 1135–1144

[14] Ribeiro, M.T., Singh, S., Guestrin, C. ‘Anchors: High-precision model-agnostic explanations’. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32. (, 2018.

[15] Lundberg, S.M., Lee, S.I.: ‘A unified approach to interpreting model predictions’, *Advances in neural information processing systems*, 2017, **30**

[16] Smilkov, D., Thorat, N., Kim, B., Viégas, F., Wattenberg, M.: ‘Smoothgrad: removing noise by adding noise’, *arXiv preprint arXiv:170603825*, 2017,

[17] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: ‘Efficient formal safety analysis of neural networks’, *Advances in neural information processing systems*, 2018, **31**

[18] Gehr, T., Mirman, M., Drachsler.Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M. ‘Ai2: Safety and robustness certification of neural networks with abstract interpretation’. In: 2018 IEEE symposium on security and privacy (SP). (IEEE, 2018. pp. 3–18

[19] Ehlers, R. ‘Formal verification of piece-wise linear feed-forward neural networks’. In: Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15. (Springer, 2017. pp. 269–286

[20] Ignatiev, A., Narodytska, N., Marques.Silva, J. ‘Abduction-based explanations for machine learning models’. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33. (, 2019. pp. 1511–1519

[21] Calegari, R., Ciatto, G., Omicini, A.: ‘On the integration of symbolic and sub-symbolic techniques for xai: A survey’, *Intelligenza Artificiale*, 2020, **14**, (1), pp. 7–32