Eager to Stop: Efficient Falsification of Deep Neural Networks

Guanqin ZHANG[1,2]

[1] University of New South Wales
[2] CSIRO, Data61
Guanqin.zhang@unsw.edu.au

Abstract. Deep Neural Networks (DNNs), extensively used in safety-critical domains, require methods to detect misbehavior and ensure provable specifications. DNN testing encounters limitations in time and coverage, affecting effectiveness. DNN verification divides into exact and approximated approaches. Due to scalability challenges, exact methods yield precise outcomes but are suitable for smaller networks. Approximated techniques using abstractions tend to be over-approximated for soundness. Over-approximated verifiers might produce more misleading counterexamples than actual violations, impacting the identification of flaws. This paper proposes a falsifier to efficiently identify counterexamples for DNN robustness by refuting specifications. The proposed approach gradient information to fast approach local optima against specifications, collecting relevant counterexamples effectively.

Keywords: Robustness · Verification · Falsification

1 Introduction

DNNs are widely implemented and impressively deployed in the safety-critical domains, such as autonomous vehicles [1], program analysis [18, 2], and airborne collision avoidance systems [9]. Although DNNs possess remarkable abilities, the growing apprehension surrounding their potentials, such as adversarial perturbations [7, 12] for misclassification and unforeseeable decisions. This motivates the understanding of the reliability and quality assurance of the underlying models.

There has been a notable upsurge in the exploration and development of analysis and verification techniques for neural network robustness. Existing approaches can be mainly categorized as testing and verification. Testing [14, 21, 7, 23] is usually providing counterexamples, such as adversaries, to reject the robustness of the DNNs. During the testing, the evaluation is based on specific test inputs and criteria, which may still lead to unforeseen issues. On the other hand, verification [8, 20, 5, 6, 16] can mathematically certify models with provably guaranteed robustness or supply a counterexample that violates the expected behaviors of models.

2 Problem Statement and Related Work

A network model $N : \mathbb{R}^n \to \mathbb{R}^m$ maps an n-dimensional input vector to an m-dimensional output vector. The model N takes an input $\boldsymbol{x}$ and outputs $N(\boldsymbol{x})$. The typical verification problem for the network model N with a specification $\phi : \mathbb{R}^{n+m} \to \{\mathbb{T}, \mathbb{F}\}$ is a decidable problem whether there existing $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$ holds $(N(\boldsymbol{x}) = \boldsymbol{y}) \wedge \phi(\boldsymbol{x}, \boldsymbol{y})$. ϕ represents the desired property that model behaviors followed human expectation. In this context, we use (l_p, ϵ)-adversary to denote a perturbed input region centered at $\boldsymbol{x}_0$ with ϵ radius, i.e., $\{\boldsymbol{x}' \mid \|\boldsymbol{x}' - \boldsymbol{x}_0\|_p \leq \epsilon\}$ measured by l_p norm. The satisfied results indicate the specification holds, whereas the unsatisfiability expresses the existence of a counterexample $\boldsymbol{x}$ within the (l_p, ϵ)-adversary, violates the specification.

Adversarial Examples Adversarial robustness responds to the ML models' reliability, which has recently attracted significant attention. Some adversaries generated by attack methods can indicate the existence of a violation of the specification. Szegedy *et al.* [19] first proposed the generation method for adversarial perturbations and leveraged a hybrid loss to approximate the solution of inner maximization. Furthermore, Goodfellow *et al.* [7] introduced an efficient FGSM method to generate the adversarial inputs for misleading the model behavior. However, the efficiency property comes from the linear loss function, which leads to the vulnerability to iteration attacks. In response to this problem, Madry *et al.* [12] pushed this method into the multi-iteration attack and released their gradient-based PGD method for inner maximization solving. Following that, a line of works emerged and boosted the development of adversarial attacks [13].

Neural Network Testing Neuron coverage [14] is utilized to count each neuron's activation status, and Ma *et al.* [11] extend with a set of test criteria for deep learning systems. Xie *et al.* [21] detect potential defects of neural networks by coverage-guided fuzzing framework with metamorphic testing. However, test-based approaches suffer from the limited number of crafted testing samples, such that they cannot enumerate all possible inputs to denote erroneous behavior of the network.

Neural Network Verification Formal verification techniques [10] can certify neural networks based on specifications to ensure proof or identify violations. One important aspect of verification is gauging the robustness of the neural network model against input adversarial perturbations. Verification approaches fall into two main categories: (1) *Exact* verification typically utilizes mixed integer linear programming (MILP) solver [20] or satisfiability modulo theories (SMT) solvers [8], which suffers low scalability due to solving such an 'NP-hard' problem. (2) *Approximated* verification often reduces the non-linear properties into linear inequalities, such as instantiated neural network properties into the abstract domain, such as polytope [17], zonotope [16].

3 Proposed Solution

Verification to Falsification. Verifying the robustness of a model is akin to proving a theorem, which poses a significant challenge in accurately assessing a neural network. Specifically, acquiring proof for the network's robustness can be difficult, leading to the possibility of overestimating its robustness. Ideally, a proven robust model is necessarily reasoning *all* possible perturbation properties on a given input and indicating *non-existence* of violation. This verification is impossible when the networks are becoming larger and deeper. Consequently, the majority of verification methods that have been published either choose a subset of properties to analyze or utilize approximations of the model. We propose a falsification scheme to eagerly find the counterexample within the (l_p, ϵ)-adversary for the network model. Formally, the falsification problem keeps searching for a falsifying $\boldsymbol{x}$ that violates the specification ϕ. Falsification shortcuts the verification processes and only supplies the rejected violations.

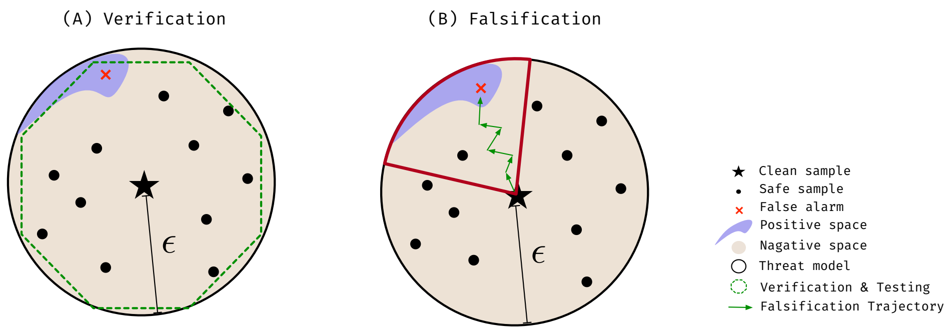

Fig. 1. Comparison between verification and falsification. We depict a threat model (threat model refers to a specification of the potential threats by some errors) suspected of containing positive feature space, including some false alarm samples. The false alarm samples can fool the model into making erroneous predictions.

Figure 1 demonstrates the comparison between verification and falsification of DNN. We construct a binary classification problem for a clean sample, which contains a false alarm sample within a feasible set of a threat model, as shown in (l_p, ϵ)-adversary. Verification can be described as a constraint of specifications in a green dashed-line octagon on the left-hand side. The octagon expresses specification constraints, which cover the ample feature space for the threat model. In practice, most of the existing exact verification methods borrowed SMT or MILP solvers scale poorly when the size of the network is growing larger. On the other hand, the approximated methods do not directly verify the property of the network but reduce it to a relaxation problem. So the designed/expected specifications do not precisely formalize the threat model behaviors. As shown in Figure 1 (A), the green octagon shape may overly cover the actual target specification from the model and cover most of the feature space.

Falsification attempts to reject the violations by eagerly falsifying the non-robust counterexamples beyond DNN models in a smaller search space. From this perspective, we here argue for falsifying the properties of the designed specification. The falsification aims to provide a convincing *optimal* point that helps counterfeit the verification specification in a smaller region contoured with the red line in Figure 1 (B). For a perturbation within (l_p, ϵ)-adversary to claim as a false alarm violated to a robust model specification, it is necessary to actually find a perturbed example $\boldsymbol{x}'$ that causes the model to make an error. This is similar to composing an adversary by using adversarial attack approaches. FGSM claims that linear behavior in high-dimensional spaces is sufficient to cause adversaries. However, adversarial attack approaches cannot assemble theoretic proof results to certify the model.

Falsification spreads across different domains [22], which refers to the concept that requirements (specifications) are falsified (not true). Guided by human-designed specifications, falsifiers can reach violations faster than verifiers when processing with a non-robust model. *Dohmatob* [4] finds robustness is impossible to achieve under some assumptions with the data. DNNF [15] reduces the neural network input and output property to an equivalent set of correctness problems.

Proposed Approach Our approach aims to search the counterexample and falsify the non-robust DNN model diligently. Based on the model, we process the differential activation function, as they are differentiable and continuous. Firstly, we define a specification to the network behavior:

$$\forall \boldsymbol{x} \in \{\boldsymbol{x}' \mid \|\boldsymbol{x}' - \boldsymbol{x}_0\|_p \leq \epsilon\}, N_{s_0}(\boldsymbol{x}) - N_{s_1}(\boldsymbol{x}) > 0, \tag{1}$$

where p is normalization, usually taken as ∞-norm, and s_0 is the original label for $\boldsymbol{x}_0$. Then, the model behaviour requires that for any of $\boldsymbol{x}$ within the (l_p, ϵ)-adversary should always be larger than the label value for s_1. We aim to determine whether the direction violates the specification. To decide the gradient, we use a vector Δ, that has the same dimension as the input domain, to identify the direction, namely, given an input $\boldsymbol{x}$, it holds the objective function $N_{s_0}(\boldsymbol{x} + \Delta) > N_{s_1}(\boldsymbol{x})$. The problem remains to decide Δ.

We use the line search method to start from the given direction of Δx_k to move with a step length $t > 0$ to modulate how far along this direction we proceed. The direct aim would satisfy: $f(x_k + t\Delta x_k) < f(x_k)$. Armijo [3] step size constraint is a method used in optimization algorithms to determine the step size. We use the Armijo condition to search for a sufficient decrease to our objectives. The constraint states that if the step size is small enough, the value of the objective function will decrease in gradient descent or other optimization algorithms. The Armijo step size constraint restricts the step size at each iteration to ensure the algorithm's convergence. Specifically, the Armijo step size constraint requires that the step size at each iteration satisfies the following inequality:

$$f(x_k + t\Delta x_k) \leq f(x_k) + c_1 t \nabla f_k^T \Delta x_k \tag{2}$$

Here, x_k is the current value of the optimization variable, Δx_k is the search direction, t is the step size, $f(x)$ is the objective function, ∇f_k is the gradient of the objective function at the current point, and c_1 is a constant typically chosen between 0 and 1. If the step size t satisfies the above inequality, it is considered acceptable; otherwise, the step size needs to be reduced, and the search is restarted. When we repeat and collect enough points, we can falsify the model as it is not robust.

4 Conclusion and Future Work

We propose a new falsification approach to complement the existing neural network verification approaches in searching and identifying counterexamples to prove the existence of violations in a *non-robust* model. We propose utilizing the Armijo line search method to iteratively reach the counterexample. Armijo borrows the gradient information from the network model, of which the advantage is the falsification of the specification in the smaller search space.

In our upcoming research endeavors, we intend to incorporate and examine additional elements of the network model. In our current study, we exclusively focused on the differentiable activation function. An eminent obstacle lies in dealing with piece-wise activation functions like ReLU, wherein the output lacks continuity. Addressing this challenge can facilitate the analysis of a broader range of network models.

An additional aspect involves utilizing the falsification findings for the purpose of enhancing the network model's integrity. Falsification provides prompt identification of cases where violations occur during the initial stages. Armed with these instances of violation, we can effectively identify shortcomings, subsequently enhancing both the performance and robustness of the model.

References

1. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
2. Cheng, X., Zhang, G., Wang, H., Sui, Y.: Path-sensitive code embedding via contrastive learning for software vulnerability detection. In: ISSTA. pp. 519–531 (2022)
3. Dai, Y.H.: On the nonmonotone line search. Journal of Optimization Theory and Applications **112**, 315–330 (2002)
4. Dohmatob, E.: Generalized no free lunch theorem for adversarial robustness. In: International Conference on Machine Learning. pp. 1646–1654. PMLR (2019)
5. Fischer, M., Sprecher, C., Dimitrov, D.I., Singh, G., Vechev, M.: Shared certificates for neural network verification. In: CAV' 2022, Part I. pp. 127–148. Springer (2022)
6. 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: SP. pp. 3–18. IEEE (2018)
7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)

8. 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. pp. 97–117. Springer (2017)
9. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., 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. pp. 443–452. Springer (2019)
10. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for verifying deep neural networks. Foundations and Trends® in Optimization **4**(3-4), 244–404 (2021)
11. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., et al.: Deepgauge: Multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE international conference on automated software engineering. pp. 120–131 (2018)
12. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
13. Nicolae, M.I., Sinn, M., Tran, M.N., et al.: Adversarial robustness toolbox v1. 0.0. arXiv:1807.01069 (2018)
14. 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. pp. 1–18 (2017)
15. Shriver, D., Elbaum, S., Dwyer, M.B.: Reducing dnn properties to enable falsification with adversarial attacks. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). pp. 275–287. IEEE (2021)
16. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. Advances in neural information processing systems **31** (2018)
17. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)
18. Sui, Y., Cheng, X., Zhang, G., Wang, H.: Flow2vec: Value-flow-based precise code embedding. ACM **4**(OOPSLA), 1–27 (2020)
19. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
20. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. arXiv preprint arXiv:1711.07356 (2017)
21. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 146–157 (2019)
22. Zhang, Z., Arcaini, P., Hasuo, I.: Constraining counterexamples in hybrid system falsification: Penalty-based approaches. In: NASA Formal Methods Symposium. pp. 401–419. Springer (2020)
23. Zhao, Z., Chen, G., Wang, J., Yang, Y., Song, F., Sun, J.: Attack as defense: Characterizing adversarial examples using robustness. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 42–55 (2021)