Reproducibility Experiments on the Verification Results of Neural Networks

Dennis Gross[1]

Radboud University, Toernooiveld 200-222, 6525 EC Nijmegen, The Netherlands
dgross@science.ru.nl
https://sws.cs.ru.nl/HomePage

Abstract. Artificial neural networks have been widely adopted for various applications, such as machine vision, autonomous driving, and smart factories. However, reproducibility, a critical factor in many neural network studies, is frequently overlooked. In this study, we focus on the reproducibility of formal verification studies related to neural networks. Specifically, we investigate the reproducibility of three studies in this field by analyzing the reproducibility of their experimental results.

Keywords: Formal Verification · Neural Networks · Reproducibility.

1 Introduction

Neural networks (NNs) have delivered impressive results across various applications [9,1,17]. A NN is modeled after the structure and function of the human brain. It takes input data and passes it through multiple layers of interconnected neurons to output a value. The network's ability to learn from data comes from adjusting the strength of connections between neurons, known as weights, based on a measure of the network's performance on a training data set [2]. The output value of the neural network can represent various things depending on the application, such as a classification label for an input [12].

However, their widespread use has also raised concerns about their safety [6]. A small perturbation in the input can cause a NN to produce different outputs, leading to unintended consequences like the misclassification of an image [4,13]. Several researchers have proposed formal verification methods to address the previously mentioned issues to ensure that a NN satisfies a given specification, such as robustness under all possible perturbations in a particular input range [20,14,5,10].

Reproducing the results of papers that propose these formal verification methods for neural network classifiers is important for validating their results, comparing approaches, and identifying their limitations [3].

This report examines the reproducibility of various formal verification papers in the field of neural network verification and highlights the challenges that can arise during the reproduction process. First, we provide an introduction to the formal verification of neural network classifiers. Next, we present our analysis results and then discuss the implications of our findings.

2 Formal Verification of Neural Network Classifiers

In this section, we briefly introduce formal verification for neural network classifiers and discuss how these techniques can be integrated into the training pipeline of neural networks to improve their robustness. If the reader is not familiar with neural network classifications, we refer here to multiple surveys [18,8].

Definition 1 (Neural Network). *A neural network* $f\colon X \to Y$ *maps an input* $x \in X$ *to an output* $y \in Y$*, where* $X, Y \subseteq \mathbb{R}$*.*

A neural network verification problem can be cast into the following decision problem: *Given a neural network* $f(x)$*, an input domain* $C \subseteq \mathbb{R}$*, and a property* $P \subseteq \mathbb{R}$*. For all* $x \in C$*, does* $f(x)$ *satisfy* P*?* The property P is a set of desirable outputs of the NN, which are dependent on the inputs C.

Example 1. Let $f(x)$ be a binary neural network classifier and x_0 be a positive example such that $f(x_0) \geq 0$. In this scenario, we can define the set P as the set of non-negative real numbers ($\mathbb{R}^+$) and restrict the input x to a bounded l_∞-norm ball $C = \{x \mid |x - x_0|_\infty \leq \epsilon\}$. The success ($\forall x \in C f(x) \in P$) of the verification process guarantees that the label of x_0 cannot be flipped for any perturbed inputs within C [20]. We refer to such a success as the neural network $f(x)$ being *robust*.

The *complete verification setting* requires the verifier to provide a definite "robust/not-robust" answer for a property under verification. Early complete verifiers relied on techniques such as satisfiability modulo theory (SMT) and mixed integer linear programming (MILP). In the case of MILP, the verifier is required to solve the optimization problem specified in Equation (1) while also meeting additional constraints in order to determine the global minimum. Unfortunately, these methods often lack scalability due to too many variables needing to be checked in the MILP and SMT encoding of state-of-the-art neural networks [11].

$$min_{x \in C} f(x) \tag{1}$$

On the other hand, *incomplete solvers* are more scalable, but they can only provide a sound analysis. This means that they can only approximate the lower bound of $\min_{x \in C} f(x)$ as $\underline{f}$ and can verify the property when $\underline{f} \geq 0$. However, no conclusion can be drawn when $\underline{f} < 0$ [20].

Branch-and-bound (BaB) frameworks have been adopted for efficient verification. BaB attempts to solve the optimization problem $\min_{x \in C} f(x)$ by recursively splitting the input domain into multiple subdomains and using an incomplete verifier as a bounding procedure to provide relatively tight bounds for each subdomain. The verification process using BaB is sound as long as the bounding method used for each subdomain is sound. However, a sound bounding method is not always complete [20]. It additionally requires feasibility checking in the bounding method. For example, a bounding method may lose the feasibility information encoded by the sub-domain constraints, ultimately resulting in an

incomplete verification. An incomplete bounding method that supports feasibility checking is called *Linear Programming (LP)*. LP finds relatively tight bounds for each sub-domain, but it has relatively expensive solving costs. Our first two studies focus on accelerating the bounding of BaB.

The ultimate goal of robustness verification is constructing a training method to lower certified error in the test data [16]. Several existing approaches integrate formal verification results directly into the training process of neural networks [19,7,15]. However, training neural networks with verifiable robustness guarantees is challenging. Therefore, our third study focuses on improving the training method.

3 Reproduction Results

In this report, we focus on formal verification methods to verify neural network classifiers and a formal verification method that improves the guaranteed robustness of neural network classifiers.

Paper selection clarification. We would like to clarify that our selection of the following studies should not be interpreted as accusing or blaming anyone. We recognize the significance of these influential works in our field and have chosen them based on their relevance to our research. It is important to note that we hold the results of these studies in high regard and deeply respect their contributions to the advancement of knowledge in our area of study. Another noteworthy point is that these studies are several years old. Given the rapid developments in fields such as machine learning and formal verification, it is intriguing to examine their reproducibility over time.

Our procedure. It is important to clarify the way how we reproduced the experiments. For the environment setup, we worked on each maximum of one hour. If the working time succeeded that threshold, we concluded that reproducing the experiments was impossible. For running the environments, we set a threshold of 12 hours. If the run time succeeded that threshold, we concluded that we could not verify the experiment results.

3.1 Fast and Complete: Enabling Complete Neural Network This study presents a complete verification procedure. Verification with Rapid and Massively Parallel Incomplete Verifiers

A key factor for completeness involves feasibility checking in the bounding method. Some incomplete verifiers, such as LiRPA, can not tell the infeasibility of subdomains. LP, on the other hand, always detects infeasibility in the bounding method. However, LP is not the fastest method. This study combines an optimized LiRPA method with LP in a BaB framework to create a faster formal verification method for neural network classifiers called *Fast and Complete (FAC)* [20].

Experiments. In their experiments, the authors compared their FAC verification method against several state-of-the-art verifiers, including BABSR, MIPPlanet, GNN, GNN-ONLINE, and BDD+BABSR. They evaluated the performance of all of the previously mentioned methods on various types of neural networks, including easy, medium, hard, wide, and deep architectures [20], using different verification properties and the CIFAR-10 dataset.

The authors demonstrate that, in most cases, their method FAC not only outperforms the other verification methods but is also more scalable.

Reproducibilty

General. The already trained models and the datasets are provided. This saves training time.

Setup The setup is done by installing an Anaconda environment. A reference to the benchmark methods is provided. The reproduction of these benchmark methods needs to be done separately.

Executing During executing the Python script, an error message occurred and mentioned that our local machine GPU was not supported by the current PyTorch version. After fixing the requirements, we could resolve this issue but then run into another GPU related error.

3.2 Scaling the convex barrier with active sets

This study presents an accelerated branch method. The *Active Set solver (ASS)* proceeds by repeatedly solving modified instances of the initial optimization problem (see Equation (1)), where the set of constraints in the initial problem is replaced by a smaller set of constraints. The solution of the modified instance is a lower bound and more iterations lead to tighter bounds. This results in a solver that is both efficient in terms of memory usage and capable of producing tight bounds for the initial optimization problem, even when dealing with many constraints. The computed bounds can be utilized for incomplete verification or as the bounding component of BaB to achieve complete verification [14].

Experiments. The researchers demonstrated that ASS yields significant formal verification speed-ups. Their results show that scalable tightness is critical to the efficiency of neural network verification. They compare their approach in an incomplete and complete verification setting.

Reproducibility

General. The already trained models and the datasets are provided. In this study, we have access to both the implementation and the other methods used for comparison, unlike in many situations where the other methods may not be available.

Setup. It has been observed that the Git command contained an erroneous github-repository name, which necessitated correction from *plnn-bab* to *scaling-the-convex-barrier*.

Executing. We were unable to execute the experiment scripts on our machine in their default configuration. To run the experiments, we had to make code alterations, which involved removing the *.cuda*() function call. This resulted in a slower runtime and we had to stop the experiments after 12 hours. Providing runtime details for each script would be helpful.

Result visualization. A visualization script for all their plots is provided. To generate these plots, it was necessary to manually install a Python package (Seaborn).

Take-Aways. It is advisable to perform a thorough review to ensure the accuracy of modifications made to the GitHub repository. Furthermore, it should be noted that formal verification can be time-consuming. In this regard, providing approximate details on the runtime can serve as a useful aid. Supporting a script that also executes the other methods to which the developed method is compared is essential. Using virtual environments or containers makes it easy to reproduce the experiments. A link that refers to their documentation is helpful. However, potential conflicts with different third-party libraries and hardware components (for example. GPUs) should be considered.

3.3 Towards Stable and Efficient Training of Verifiably Robust Neural Networks

Training neural networks with verifiable robustness guarantees is challenging, and various methods exist trying to tackle it. Each of these methods has its weaknesses and strengths. This study found out that a network trained using IBP can obtain good verified errors, but during training (especially in the beginning), it is unstable and hard to tune. The reason lies in the fact that IBP does not give tight bounds in the beginning of the training, which is essential to gradually learn to find a good set of weights. CROWN can give tighter lower bounds at the cost of high computational expenses, but it over-regularizes the network and forbids achieving good standard and verified accuracy. To address these issues, this study proposes a new approach called *CROWN-IBP* that combines the strengths of both methods. Specifically, CROWN-IBP initially uses CROWN to provide tighter bounds at the beginning of training. As the network learns, these bounds are gradually replaced by IBP bounds, leading to a model with learned tight IBP bounds in the end. This approach overcomes the weaknesses of IBP and CROWN and results in a more robust neural network [4].

Experiments. Their experiments show that CROWN-IBP consistently outperforms other baselines in standard and verified errors and achieves state-of-the-art verified test errors.

Reproducibility

Setup. We needed to manage to create our own virtual environment. No package dependency file was provided. This led to the result that executing the code was impossible. They set seeds that are necessary to gain the exact neural network training results.

Take-Aways. It is essential to support the repository with a list of all dependencies, so that after years it is still possible to reproduce the results.

4 Discussion

The major reproducibility problem in these studies is the dependency on third-party software and hardware components. Therefore, it is essential to provide artificats as independent of third-party tools as possible. To resolve conflicting third-party software dependencies, it is useful to isolate the developed tool in an isolated container like via Anaconda or Docker. Hardware component dependencies related to GPUs are more difficult to solve. If possible, we recommend creating additional small experiments on CPUs (with runtime information) to show that the method works and link them to the GPU experiments.

To also save time in verifying neural networks, try to support the repository with pretrained models that are used in the experiments.

Another experience is mostly that when having to compare to some other method, there is either no implementation available or the models used are not available (or both).

If a publicly available dataset is available, integrate the dataset downloading and preprocessing into the script that runs the experiments.

In the case of reproducing neural network training processes (and everywhere where randomness is involved), always set a seed to allow the reproduction of the same training results.

Minor mistakes, like changing the repository name but not changing the git-clone command in the README, can happen. GitHub bots and actions may help to track such mistakes.

References

1. Retraction note: An automatic tamil speech recognition system by using bidirectional recurrent neural network with self-organizing map. Neural Comput. Appl. **35**(4), 3575 (2023)
2. Bayram, B., Kulavuz, B., Ertugrul, B., Bayram, B., Bakirman, T., Çakar, T., Dogan, M.: Classification of skin lesion images with deep learning approaches. Balt. J. Mod. Comput. **10**(2) (2022)
3. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (VNN-COMP). CoRR **abs/2301.05815** (2023)

4. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: IEEE Symposium on Security and Privacy. pp. 39–57. IEEE Computer Society (2017)
5. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: ATVA. Lecture Notes in Computer Science, vol. 10482, pp. 269–286. Springer (2017)
6. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: ICLR (Poster) (2015)
7. Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T.A., Kohli, P.: On the effectiveness of interval bound propagation for training verifiably robust models. CoRR **abs/1810.12715** (2018)
8. Jena, B., Nayak, G.K., Saxena, S.: Convolutional neural network and its pretrained models for image classification and object detection: A survey. Concurr. Comput. Pract. Exp. **34**(6) (2022)
9. Karthik, K., Mahadevappa, M.: Convolution neural networks for optical coherence tomography (OCT) image classification. Biomed. Signal Process. Control. **79**(Part), 104176 (2023)
10. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV (1). Lecture Notes in Computer Science, vol. 10426, pp. 97–117. Springer (2017)
11. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In: CAV (1). Lecture Notes in Computer Science, vol. 11561, pp. 443–452. Springer (2019)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. pp. 1106–1114 (2012)
13. Moosavi-Dezfooli, S., Fawzi, A., Frossard, P.: Deepfool: A simple and accurate method to fool deep neural networks. In: CVPR. pp. 2574–2582. IEEE Computer Society (2016)
14. Palma, A.D., Behl, H.S., Bunel, R., Torr, P.H.S., Kumar, M.P.: Scaling the convex barrier with active sets. In: ICLR. OpenReview.net (2021)
15. Raghunathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: ICLR (Poster). OpenReview.net (2018)
16. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: NeurIPS. pp. 9832–9842 (2019)
17. Takayanagi, H., Enosawa, R., Furuya, S., Morishita, K., Saito, K.: Development of neural networks integrated circuit driving electrostatic motors for microrobot. Artif. Life Robotics **28**(1), 192–198 (2023)
18. Turay, T., Vladimirova, T.: Toward performing image classification and object detection with convolutional neural networks in autonomous driving systems: A survey. IEEE Access **10**, 14076–14119 (2022)
19. Wong, E., Kolter, J.Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: ICML. Proceedings of Machine Learning Research, vol. 80, pp. 5283–5292. PMLR (2018)
20. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: ICLR. OpenReview.net (2021)