Neural Network Verification with DSE

Benedikt Böing[1], Falk Howar[2,3], Jelle Hüntelmann[1], Emmanuel Müller[1] and Richard Stewing[2]

[1]*Research Center Trustworthy Data Science and Security, TU Dortmund*
[2]*Aqua-Group, LS 14, TU Dortmund University, Otto-Hahn-Straße 14, 44227 Dortmund, Germany*
[3]*Fraunhofer ISST, Emil-Figge-Str. 91, 44227 Dortmund, Germany*

Abstract

Neural network with Linear and ReLU nodes can be represented as sequential linear programs that are simple in structure but have many program paths: different combinations of ReLU activations correspond to paths in the corresponding program. Naive applications of conventional program analysis techniques for proving properties of such networks are hampered by the expontential number of activation patterns (i.e., program paths). In this paper, we explore a technique for scaling verification by decomposing the verification task into first finding feasible paths and then proving properties for individual paths, resulting in multiple small verification tasks (compared to monolithic analysis of the network). Moreover, this enables horizontal scaling, i.e., parallel execution, further decreasing analysis time. Finally, the proposed decomposition allows us to reuse a once computed set of feasible paths for the verfication of multiple properties, compounding performance gains when checking multiple properties on the same network.

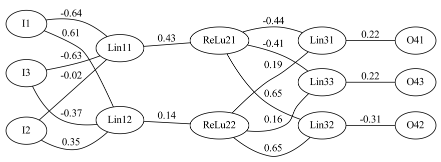
Keywords

Neural Network, Neural Network Verification, Dynamic Symbolic Execution, Formal Anaylsis

1. Introduction

The complexity required for neural networks to generalize knowledge brings classical program verification techniques to their limits [1]. For ReLU neural networks - nets consisting of Linear and ReLU nodes - methods for verification are established: they translate the problem into a Mixed Integer Linear Program (MILP) or to an instance for an SMT solver. Since it is unknown which ReLU configurations are feasible, these approaches implicitly have to iterate over all of them, hampering scalability. For a network with n ReLU nodes, 2^n activation patterns have to be explored. In our tests, we found that only a small portion (1%) of all 2^n configurations is feasible (i.e., can be triggered by inputs). In such cases, solvers waste a lot of effort on checking infeasible ReLU combinations — especially when multiple properties are analyzed for a single network as in VNN competition on the ACAS-dataset [2]. We mitigate this problem by decomposing the verification effort into two steps: we first enumerate the feasible ReLU combinations in a pre-processing step and then verify properties only on feasible combinations.

OVERLAY 2022: 4th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 28, 2022, Udine, Italy
benedikt.boeing@cs.tu-dortmund.de (B. Böing); falk.howar@tu-dortmund.de (F. Howar); jelle.huentelmann@cs.tu-dortmund.de (J. Hüntelmann); emmanuel.mueller@cs.tu-dortmund.de (E. Müller); richard.stewing@tu-dortmund.de (R. Stewing)



(a) Minimal Example Network

```
fun id(x : Double) : Double = x
fun relu(x : Double) : Double =
    if(x > 0) x else 0.0
val l11=id(i1*-0.64+i1*-0.02+i3*-0.63)
val l12=id(i1*0.61+i1*0.35+i3*-0.37)
val r21=relu(l11 * 0.43)
val r22=relu(l12 * 0.14)
```

(b) Implementation of the first and second layer in network of Figure 1a.

Related Work. Proving properties about neural networks is called neural network verification [3, 4, 5, 2]. One popular type of property is robustness against adversarial attacks as introduced by Szehedy et al. [6]. Some verifiers trade precision for efficiency by over-approximating the behavior of a system. Approximating parts of the network, e.g. its output [7, 8, 9], makes the verification problem easier and faster to solve. On the other hand, these approaches may reject networks that factually satisfy a given property. Other works attempt to develop precise techniques [10, 4, 11], often at the cost of limited scalability: these methods, implicitly or explicitly, enumerate all ReLU configurations corresponding to paths in the linear program of neural network. The exponential number of these configuration made them an attractive target for reducing the underlying complexity. Different approaches include adding constraints to particular ReLUs [11] or considering dependencies between different nodes [12].

In this work, we apply precise software verification techniques to neural networks. Concrretely, we use dynamic symbolic execution [13] to enumerate a network's paths. The same technique is used to analyze the taint flow of programs [14, 15] or to enhance static analyses [16, 17] and has been shown to scale well to complex programs [18]. Schlüter et al. used symbolic execution to generate TADS to check network equivalence and explainability [19]. A unique benefit of this approach is that we produce an intermediate result that can be re-used and is much less complex than the original network (from the perspective of a verifier).

2. From Networks to Programs

The size and concurrent activation pattern of neural networks can make them hard to understand. This computation model is not well suited for classical program analysis. We show this by example and transform a small neural network into a program's linear representation. Figure 1a shows the ReLU network. It consists of four layers (including the output layer): three linear layers and a single ReLU layer. We assume, but without loss of generality, that ReLU neurons only have a single input.

We transform ReLU networks into a program that uses only multiplication, addition and function calls. For the above network's layers one and two and their activation functions result in the code from Figure 1b: We multiply the vector of weights with the input and pass the result to the activation function. The name of every neuron consists of its activation function, its layer, and its position in this layer.

We can now apply dynamic symbolic execution (DSE) to the network. Symbolic execution

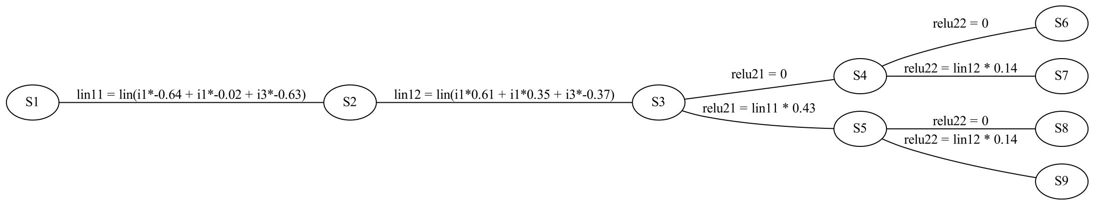


Figure 2: Partial program of the minimal example network. We annotate every state transition with the assignment changing the state.

replaces concrete values with symbolic values. The branching operators (`if`, `while`, etc.) decide which paths to follow by checking satisfiability of path constraints collected during execution. This may result in multiple branches being feasable and hence being followed. DSE consequently enumerates all feasible (i.e., satisfiable) paths through a program, filtering paths as soon as they become unsatisfiable (each unsatisfiable path prefix leads to the root of an unsatisfiable sub-tree of paths), and resulting in few additional queries to a constraint solver for many unsatisfiable paths. Figure 2 shows all paths through the code in Figure 1b. Notice that all combinations of assignments for the two ReLUs are possible[1] and on different paths. A path becomes unsatisfiable if no input can fulfill the path constraints. Take for example a network with a single input i and two ReLU nodes with weights w and $-w$. Only one of those nodes can be positive at a time and hence, the path where both activate positively is unsatisfiable. Whenever we talk about the paths of a neural network, we refer to the paths of the program into which we translated a neural network.

Algorithm 1 Enumerating all satisfiable paths and checking satisfiability after a layer is added.

```
$P \leftarrow$ the paths including the first layer $l_1$.
**for** every $l \in l_2 \cdots l_m$ **do**
    $Q \leftarrow \emptyset$
    **for** every **satisfiable** $t \in P \times l$ **do**
        Add $t$ to $Q$
    **end for**
    $P \leftarrow Q$
**end for**
**return** $P$
```

Algorithm 2 Verification of φ over all satisfiable paths P - Safety

```
$B \leftarrow \top$
**for** Every path $p \in P$ **do**
    $B \leftarrow B \wedge p \wedge \neg\varphi$ is unsatisfiable
**end for**
**return** $B$
```

3. Decomposing Verification

A neural network N corresponds to a set of program paths P, divided into satisfiable (S) and unsatisfiable (U) paths: $S \cup U = P$ with $S \cap U = \emptyset$. We decompose the verification of some property φ on N, i.e., checking if $N \models \varphi$, into checking φ on all paths, i.e., $p \models \varphi$ for $\forall p \in S$

[1]When we say “possible”, we are not saying they are all executable at runtime. Some paths may have an unsatisfiable path constraint. “Possible” here refers to the syntactic category. When we try to restrict this set to executable paths, we refer to them as satifiable.

Table 1
Results of Preliminary Experiments.

Task	WCT
Naive Paths Enumeration (N1)	DNF (>5d)
Layer-wise Enumeration (N1, Algorithm 1 without parallel execution of loops)	5h
Parallel Layer-wise Enumeration (N1, Algorithm 1 with parallel execution of loops)	50min
Parallel Layer-wise Enumeration (N2, Algorithm 1 with parallel execution of loops)	15h
Parallel Verification of Network Equivalence (N2)	5m25s
Monolithic verification of Network Equivalence (N2, MathSAT)	43m
Monolithic verification of Network Equivalence (N2, Z3)	5.9d

(Algorithm 2). To this end, we have to first compute S. Here, we optimize the runtime by varying when we check for satisfiability of path prefixes in a breath-first exploration of all paths. Assume paths p, p_1, p_2 with path constraints c, c_1, c_2 respectively, such that $p = p_1p_2$, and $c = c_1 \wedge c_2$. If c_1 is unsatisfiable, c is also unsatisfiable and need not be explored. Thus, checking path constraints sooner, e.g. after p_1 instead of after p, reduces the number of path constraints to check for satisfiability. Checking path constraints after each layer (Algorithm 1) resulted in the best trade-off between runtime and potential for parallelization in our tests.

4. Preliminary Results and Conclusion

We demonstrate our approach by analyzing properties of two networks: we use neural network N1 (19 ReLUs) to analyze performance of mutiple path enumeration strategies and network N2 (38 ReLUs) to compare verification against two monolithic approaches.[2]

Table 1 summarizes all results: Filtering unsatisfiable paths makes the approach tractable. Enumerating all paths and checking their path constraints afterward, was infeasible for $524\,288 = 2^{19}$ paths. Parallel layer-wise path enumeration was the optimal strategy. For the verification of a simple assertion on outputs φ, we use monolithic verification with MathSAT and Z3 as a baseline comparisons. MathSAT, a solver optimized for linear arithmetics, can solve the verification task in 43 minutes. The popular $Z3$ SMT solver needs more than five days. Our approach needs (in a very unoptimized implementation) 15 hours for enumerating all satisfiable paths and 5 minutes and 25 seconds for checking the property on these paths. This makes us confident, that with further optimization of the implementation, the approach will pay off when multiple properties have to be checked.

As next steps, we plan to thoroughly evaluate and compare scalability to different sizes of networks and types of properties. We also plan to explore extending the technique to back-feeding neural networks and more activation functions. Ultimately, we are interested in definitions of properties that are of interest for neural networks.

[2]The hardware was a research server: Common KVM processor with 72 processors and 135 GB memory running Linux `5.4.0-125-generic`. The SMT-Solver was `z3` (https://github.com/Z3Prover/z3) in Version `4.11.0`.

Acknowledgments

This research is partially funded by the German Federal Ministry of Education and Research (BMBF) within the Data Trust Models Funding Action (16DTM106A). The authors are responsible for the content of this publication. This work was also supported by the Research Center Trustworthy Data Science and Security, an institution of the University Alliance Ruhr.

References

[1] K. Scheibler, F. Neubauer, A. Mahdi, M. Fränzle, T. Teige, T. Bienmüller, D. Fehrer, B. Becker, Accurate icp-based floating-point reasoning, in: 2016 Formal Methods in Computer-Aided Design (FMCAD), 2016, pp. 177–184. doi:10.1109/FMCAD.2016.7886677.

[2] A. Irfan, K. D. Julian, H. Wu, C. Barrett, M. J. Kochenderfer, B. Meng, J. Lopez, Towards verification of neural networks for small unmanned aircraft collision avoidance, in: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), 2020.

[3] K. Y. Xiao, V. Tjeng, N. M. M. Shafiullah, A. Madry, Training for faster adversarial robustness verification via inducing relu stability, in: 7th International Conference on Learning Representations, ICLR, 2019.

[4] R. Ehlers, Formal verification of piece-wise linear feed-forward neural networks, in: Automated Technology for Verification and Analysis - 15th International Symposium, ATVA, 2017.

[5] B. Böing, R. Roy, E. Müller, D. Neider, Quality guarantees for autoencoders via unsupervised adversarial attacks, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD, 2020.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: International Conference on Learning Representations, ICLR, 2014.

[7] G. Singh, T. Gehr, M. Mirman, M. Püschel, M. T. Vechev, Fast and effective robustness certification, in: Advances in Neural Information Processing Systems 31, NeurIPS, 2018.

[8] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, L. Daniel, Efficient neural network robustness certification with general activation functions, in: Advances in Neural Information Processing Systems, NeurIPS, 2018.

[9] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, I. S. Dhillon, Towards fast computation of certified robustness for relu networks, in: Proceedings of the 35th International Conference on Machine Learning, ICML, 2018.

[10] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An efficient SMT solver for verifying deep neural networks, in: Computer Aided Verification - 29th International Conference, CAV, 2017.

[11] V. Tjeng, K. Y. Xiao, R. Tedrake, Evaluating robustness of neural networks with mixed integer programming, in: International Conference on Learning Representations, ICLR, 2019.

[12] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, R. Misener, Efficient verification of relu-based neural networks via dependency analysis, in: The Thirty-Fourth AAAI Confer-

ence on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, 2020.

[13] J. C. King, Symbolic execution and program testing, Commun. ACM 19 (1976) 385–394. doi:10.1145/360248.360252.

[14] M. Mues, T. Schallau, F. Howar, Jaint: A framework for user-defined dynamic taint-analyses based on dynamic symbolic execution of java programs, in: B. Dongol, E. Troubitsyna (Eds.), Integrated Formal Methods, Springer International Publishing, Cham, 2020, pp. 123–140.

[15] E. J. Schwartz, T. Avgerinos, D. Brumley, All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask), in: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP '10, IEEE Computer Society, USA, 2010, p. 317–331. doi:10.1109/SP.2010.26.

[16] T. Avgerinos, A. Rebert, S. K. Cha, D. Brumley, Enhancing symbolic execution with veritesting, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, Association for Computing Machinery, New York, NY, USA, 2014, p. 1083–1094. doi:10.1145/2568225.2568293.

[17] Y. P. Khoo, B.-Y. E. Chang, J. S. Foster, Mixing type checking and symbolic execution, SIGPLAN Not. 45 (2010) 436–447. doi:10.1145/1809028.1806645.

[18] P. Godefroid, M. Y. Levin, D. A. Molnar, SAGE: whitebox fuzzing for security testing, Commun. ACM 55 (2012) 40–44. URL: https://doi.org/10.1145/2093548.2093564. doi:10.1145/2093548.2093564.

[19] M. Schlüter, G. Nolte, A. Murtovi, S. Bernhard, Towards rigorous understanding of neural networks via semantics-preserving transformations, International Journal on Software Tools for Technology Transfer (2022). To appear.