Simplifying Neural Networks Using Formal Verification

Sumathi Gokulanathan[1], Alexander Feldsher[1], Adi Malca[1], Clark Barrett[2], and Guy Katz[1](✉)

[1] The Hebrew University of Jerusalem, Jerusalem, Israel
{sumathi.giokolanat,feld,adimalca,guykatz}@cs.huji.ac.il
[2] Stanford University, Stanford, USA
barrett@cs.stanford.edu

Abstract. Deep neural network (DNN) verification is an emerging field, with diverse verification engines quickly becoming available. Demonstrating the effectiveness of these engines on real-world DNNs is an important step towards their wider adoption. We present a tool that can leverage existing verification engines in performing a novel application: neural network simplification, through the reduction of the size of a DNN without harming its accuracy. We report on the work-flow of the simplification process, and demonstrate its potential significance and applicability on a family of real-world DNNs for aircraft collision avoidance, whose sizes we were able to reduce by as much as 10%.

Keywords: Deep neural networks · Simplification · Verification · Marabou

1 Introduction

Deep neural networks (*DNNs*) are revolutionizing the way complex software is produced, obtaining unprecedented results in domains such as image recognition [28], natural language processing [5], and game playing [27]. There is now even a trend of using DNNs as controllers in autonomous cars and unmanned aircraft [2,18]. With DNNs becoming prevalent, it is highly important to develop automatic techniques to assist in creating, maintaining and adjusting them.

As DNNs are used in tackling increasingly complex tasks, their sizes (i.e., number of neurons) are also increasing—to a point where modern DNNs can have millions of neurons [13]. DNN size is thus becoming a liability, as deploying larger networks takes up more space, increases energy consumption, and prolongs response times. Network size can even become a limiting factor in situations where system resources are scarce. For example, consider the ACAS Xu airborne collision avoidance system for unmanned aircraft, which is currently being developed by the Federal Aviation Administration [18]. This is a highly safety-critical system, for which a DNN-based implementation is being considered [18]. Because this system will be mounted on actual drones with limited

R. Lee et al. (Eds.): NFM 2020, LNCS 12229, pp. 85–93, 2020.
https://doi.org/10.1007/978-3-030-55754-6_5

memory, efforts are being made to reduce the sizes of the ACAS Xu DNNs as much as possible, without harming their accuracy [17,18].

Most work to date on DNN simplification uses various heuristics, and does not provide formal guarantees about the simplified network's resemblance to the original. A common approach is to start with a large network, and reduce its size by removing some of its components (i.e., neurons and edges) [12,15]. The parts to be removed from the network are determined heuristically, and network accuracy may be harmed, sometimes requiring additional training after the simplification process has been performed [12].

Here, we propose a novel simplification technique that harnesses recent advances in DNN verification (e.g., [9,19,32]). Using verification queries, we propose to identify components of the network that *never affect its output*. These components can be safely removed, creating a smaller network that is completely equivalent to the original. We empirically demonstrate that many such removable components exist in networks of interest.

We implement our technique in a proof-of-concept tool, called *NNSimplify*. The tool uses the following work-flow: (i) it performs lightweight simulations to identify parts of the DNN that are candidates for removal; (ii) it invokes an underlying verification engine to dispatch queries that determine which of those parts can indeed be removed without affecting the network's outputs; and (iii) it constructs the simplified network, which is equivalent to the original. A major benefit of the proposed verification-based simplification is that it does not require any retraining of the simplified network, which may be expensive.

Our implementation of NNSimplify (available online [10]) can use existing DNN verification tools as a backend. For the evaluation reported here, we used the recently published Marabou framework [21] as the underlying verification engine. We evaluated our approach on the ACAS Xu family of DNNs for airborne collision avoidance [18], and were able to reduce the sizes of these DNNs by up to 10%—a highly significant reduction for systems where resources are scarce.

The rest of the paper is organized as follows. In Sect. 2, we provide a brief background on DNNs and their verification and simplification. Next, we describe our verification-based approach to simplification in Sect. 3, followed by an evaluation in Sect. 4. We then conclude in Sect. 5.

2 Background: DNNs, Verification and Simplification

DNNs are comprised of an input layer, an output layer, and multiple hidden layers in between. A layer is comprised of multiple nodes (neurons), each connected to nodes from the preceding layer using a predetermined set of weights (see Fig. 1). By assigning values to inputs and then feeding them forward through the network, values for each layer can be computed from the values of the previous layer, finally resulting in values for the outputs.

As DNNs are increasingly used in safety-critical applications (e.g., [2,18]), there is a surge of interest in verification methods that can provide formal guarantees about DNN behavior. A DNN verification query consists of a neural network and a property to be checked; and it results in either a formal guarantee

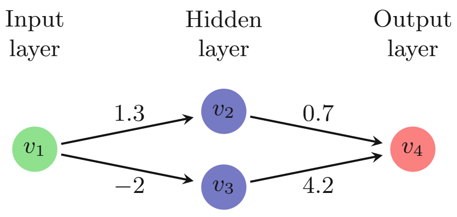

Fig. 1. A small neural network with 2 hidden nodes in one hidden layer. Weights are denoted over the edges. Hidden node values are typically determined by computing a weighted sum according to the weights, and then applying a non-linear activation function to the result.

that the network satisfies the property, or a concrete input for which the property is violated (a counter-example). Verification queries can encode various properties about DNNs; e.g., that slight perturbations to a network's inputs do not affect its output, and that it is thus robust to *adversarial perturbations* [1,4,30].

Recently, there has been significant progress on DNN verification tools that can dispatch such queries (see a recent survey [24]). Some of the proposed approaches for DNN verification include the use of specialized SMT solvers [14,19,21], the use of LP and MILP solvers [7,31], symbolic interval propagation [32], abstract interpretation [9], and many others (e.g., [3,6,8,25,26]). This new technology has been applied in a variety of contexts, such as collision avoidance [19], adversarial robustness [11,14,20], hybrid systems [29], and computer networks [22]. Although DNN verification technology is improving rapidly, scalability remains a major limitation of existing approaches. It has been shown that a common variant of the DNN verification problem is NP-complete, and becomes exponentially harder as the network size increases [19,23].

In recent years, enormous DNNs have been appearing in order to tackle increasingly complex tasks—to a point where DNN size is becoming a liability, because large networks take longer to train and even to evaluate when deployed. Techniques for neural network minimization and simplification have thus started to emerge: typically, these take an initial, large network, and reduce its size by removing some of its components [12]. The pruning phase involves the removal of edges from the network. The selection of which edges to remove is done heuristically, often by selecting edges that have very small weights, because these edges are less likely to significantly affect the network's outputs. If all edges connecting a node to the preceding layer or to the succeeding layer are removed, then the node itself can be removed. After the pruning phase, the reduced network is retrained [12,15].

3 Simplification Using Verification

Despite the demonstrated usefulness of pruning-based DNN simplification [12, 15], heuristic-based approaches might miss removable edges, if these edges do not have particularly small weights. However, such edges can be identified using verification. For example, consider the network shown in Fig. 2. As all edge weights have identical magnitudes, none of them would be pruned by a heuristic-based approach. However, using a verification engine, it is possible to check the property: “does there exist an input for which v_4 takes a non-zero value?”. If the verification tool answers “no”, as is the case for the network in Fig. 2 (because $v_4 = v_2 - v_3$ and $v_2 = v_3$), then we are guaranteed that v_4 is always assigned 0, regardless of the input. In turn, this means that v_4 can never affect nodes in subsequent layers. In this case, v_4 and all its edges can be safely removed from the network (rendering the network’s output constant). Due to the soundness of the verification process, we are guaranteed that the simplified DNN is completely equivalent to the original DNN, and thus no retraining is required.

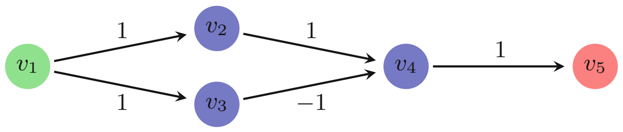

Fig. 2. Using verification, we can discover that node v_4 can safely be removed from the network.

Using verification to identify nodes that are always assigned 0 for every possible input, and can thus be removed, is the core of our technique. However, because verification is costly, posing this query for every node of the DNN might take a long time. To mitigate this difficulty, we propose the following work-flow:

1. Use lightweight simulations to identify nodes that are candidates for removal. Initially, all hidden nodes are such candidates. We then evaluate the network for random input values, and remove from the list of candidates any hidden node that is assigned a non-zero value for some input. With each simulation, the number of candidates for removal decreases.
2. For each remaining candidate node v, we create a separate verification query stating that $v \neq 0$, and use the underlying verification engine to dispatch it. If we get an `UNSAT` answer, we mark node v for removal. The candidates are explored in a layer-by-layer order, which allows us to only examine a part of the DNN for every query. For example, when addressing a candidate in layer #2, we do not encode layers #3 and on as part of our verification query, as a node’s assignment can only be affected by nodes in preceding layers. Because verifying smaller networks is generally easier, this layer-by-layer approach accelerates the process as a whole. In addition, this process naturally lends itself to parallelization, by running each verification query on a separate machine.
3. Finally, we construct the simplified network, in which the nodes marked for removal and all their incoming and outgoing edges are deleted. We can also remove any nodes that subsequently become irrelevant due to the removal of all of their incoming or outgoing edges (e.g., for the DNN in Fig. 2, after removing v_4 we can also remove v_2 and v_3, as neither has any remaining outgoing edges).

We note that our technique can be extended to simplify DNNs in additional ways, by using different verification queries. For example, it can identify separate nodes that are always assigned identical, non-zero values (duplicates) and unify them, thus reducing the overall number of nodes. It can also identify and remove nodes that can be expressed as linear combinations of other nodes.

4 Evaluation

Our proof-of-concept implementation of the approach, called NNSimplify, is comprised of three Python modules, one for performing each of the aforementioned steps. The tool is general, in two ways: (1) it can be applied to simplify any DNN, regardless of its application domain; and (2) it can use any DNN verification engine as a backend, benefiting from any future improvement in verification technology. For our experiments we used the Marabou [21] verification engine. In practice, it is required that the DNN in question be supported by the backend verification engine—for example, some engines may not support certain network topologies. Additionally, the DNN needs to be provided in a format supported by NNSimplify; currently, the tool supports the NNet format [16], and we plan to extend it to additional formats. The tool, additional documentation, and all the benchmarks reported in this section are available online [10].

We evaluated NNSimplify on the ACAS Xu family of DNNs for airborne collision avoidance [18]. This set contains 45 DNNs, each with 5 input neurons, 5 output neurons, and 300 hidden neurons spread across 6 hidden layers. The ACAS Xu networks are fully connected, and use the ReLU activation function in each of their hidden nodes—and are thus supported by Marabou.

For each of the 45 ACAS Xu DNNs, we ran the first Python module of NNSimplify (random simulations), resulting in a list of candidate nodes for removal. For each DNN we performed 20000 simulations, and this narrowed down the list of nodes that are candidates for removal to about 7% of all hidden nodes (see Fig. 3). The simulations were performed on points sampled uniformly at random, although other distributions could of course be used.

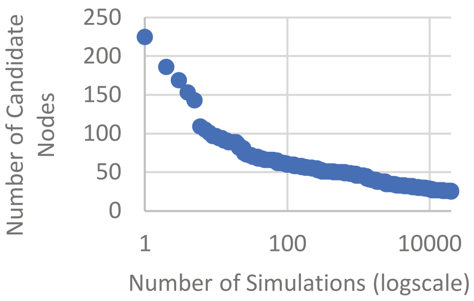

Fig. 3. Using simulation to identify nodes that are candidates for removal, on one ACAS Xu network.

Next, for each candidate for removal we ran the second Python module, which takes as input a DNN and a node v that is a candidate for removal. This module constructs a temporary, smaller DNN, where the candidate node v is the only output node (subsequent layers are omitted). These temporary DNNs were then passed to the underlying verification engine, with the query $v \neq 0$. Here, we encountered the following issue: the Marabou framework, like many linear-programming based tools, does not provide a way to directly specify that $v \neq 0$, but rather only to state that $v \geq \varepsilon$ for some $\varepsilon > 0$ (we assume all hidden nodes are, by definition, never negative, which is the case for the ACAS Xu DNNs). We experimented with various values of ε (see Fig. 4), and concluded that the choice of ε has very little effect on the outcome of the experiment—i.e., nodes tend to either be obsolete, or take on large values. The set of removed nodes was almost identical in all experiments, with minor differences due to different queries timing out for different values of ε.

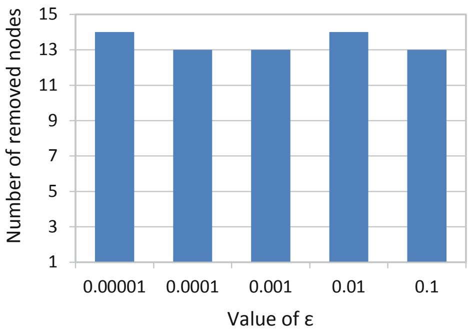

Fig. 4. Number of removed nodes as a function of the value of ε, on one of the ACAS Xu networks.

Finally, we ran the third Python module that uses the results of the previous steps to construct the simplified network.

We performed this process for each of the 45 DNNs. We ran the experiments on machines with Intel Xeon E5-2670 CPUs (2.60GHz) and 8GB of memory, and used $\varepsilon = 0.01$. Each verification query was given a 4-h timeout. Out of 1069 verification queries (1 per candidate node), 535 were `UNSAT` (node marked for removal), 15 were `SAT`, and 519 timed out (node not marked for removal). Thus, on average, 4% of the nodes were marked for removal (535 nodes out of 13500). Figure 5 depicts their distribution across the 45 DNNs. In most networks, between 11 and 15 nodes (out of 300) could be removed; but for a few networks, this number was higher. For one of the networks we discovered 29 neurons that could be removed—approximately 10% of that network's total number of neurons.

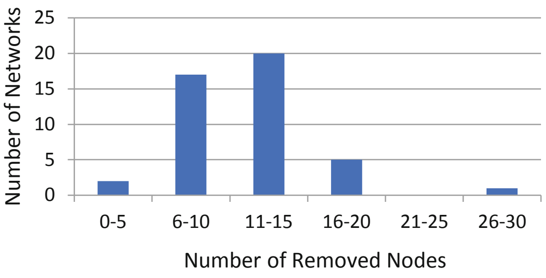

Fig. 5. Total number of removed nodes in the ACAS Xu networks.

5 Conclusion

DNN verification is an emerging field, and we are just now beginning to tap its potential in assisting engineers in DNN development. We presented here the NNSimplify tool, which uses black-box verification engines to simplify neural networks. We demonstrated that this approach can lead to a substantial reduction in DNN size. Although our experiments show that the tool is already applicable to real-world DNNs, its scalability is limited by the scalability of its underlying verification engine; but as the scalability of verification technology improves, that limitation will diminish. In the future, we plan to extend this work along several axes. First, we intend to explore additional verification queries, which would allow to simplify DNNs in more sophisticated ways—for example by revealing that some neurons can be expressed as linear combinations of other neurons, or that some neurons are always assigned identical values and can be merged. In addition, we plan to investigate more aggressive simplification steps, which may change the DNN's output, while using verification to ensure that these changes remain within acceptable bounds. Finally, we intend to apply the technique to additional real-world DNNs and case studies.

Acknowledgements. This project was partially supported by grants from the Binational Science Foundation (2017662), the Israel Science Foundation (683/18), and the National Science Foundation (1814369).

References

1. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. In: Proceedings of 30th Conference on Neural Information Processing Systems (NIPS) (2016)
2. Bojarski, M., et al.: End to end learning for self-driving cars. Technical report (2016). http://arxiv.org/abs/1604.07316
3. Bunel, R., Turkaslan, I., Torr, P.H., Kohli, P., Kumar, M.P.: Piecewise linear neural network verification: a comparative study. Technical report (2017). https://arxiv.org/abs/1711.00455v1

4. Carlini, N., Katz, G., Barrett, C., Dill, D.: Provably minimally-distorted adversarial examples. Technical report (2017). https://arxiv.org/abs/1709.10207
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. (JMLR) **12**, 2493–2537 (2011)
6. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks. In: Proceedings of 10th NASA Formal Methods Symposium (NFM), pp. 121–138 (2018)
7. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Proceedings of 15th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 269–286 (2017)
8. Elboher, Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. Technical report (2019). http://arxiv.org/abs/1910.14574
9. 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: Proceedings of 39th IEEE Symposium on Security and Privacy (S&P) (2018)
10. Gokulanathan, S., Feldsher, A., Malca, A., Barrett, C., Katz, G.: The NNSimplify Code (2020). https://drive.google.com/open?id=19TbPS7P9fo-2tRXo8ENnggLY1LxxPCd1
11. Gopinath, D., Katz, G., Păsăreanu, C., Barrett, C.: DeepSafe: a data-driven approach for checking adversarial robustness in neural networks. In: Proceedings of 16th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 3–19 (2018)
12. Han, S., Mao, H., Dally, W.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. Technical report (2015). http://arxiv.org/abs/1510.00149
13. Howard, A., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications. Technical report (2017). http://arxiv.org/abs/1704.04861
14. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
15. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. Technical report (2016). http://arxiv.org/abs/1602.07360
16. Julian, K.: NNet Format (2018). https://github.com/sisl/NNet
17. Julian, K., Kochenderfer, M., Owen, M.: Deep neural network compression for aircraft collision avoidance systems. J. Guid. Control Dyn. **42**(3), 598–608 (2019)
18. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: Proceedings of 35th Digital Avionics Systems Conference (DASC), pp. 1–10 (2016)
19. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: an efficient SMT solver for verifying deep neural networks. In Proceedings of 29th International Conference on Computer Aided Verification (CAV), pp. 97–117 (2017)
20. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Towards proving the adversarial robustness of deep neural networks. In: Proceedings of 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV), pp. 19–26 (2017)
21. Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: Proceedings of 31st International Conference on Computer Aided Verification (CAV), pp. 443–452 (2019)

22. Kazak, Y., Barrett, C., Katz, G., Schapira, M.: Verifying deep-RL-driven systems. In: Proceedings of 1st ACM SIGCOMM Workshop on Network Meets AI & ML (NetAI), pp. 83–89 (2019)
23. Kuper, L., Katz, G., Gottschlich, J., Julian, K., Barrett, C., Kochenderfer, M.: Toward scalable verification for safety-critical deep networks. Technical report (2018). https://arxiv.org/abs/1801.05950
24. Liu, C., Arnon, T., Lazarus, C., Barrett, C., Kochenderfer, M.: Algorithms for verifying deep neural networks. Technical report (2019). http://arxiv.org/abs/1903.06758
25. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. Technical report (2017). http://arxiv.org/abs/1706.07351
26. Narodytska, N., Kasiviswanathan, S., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. Technical report (2017). http://arxiv.org/abs/1709.06662
27. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Technical report (2014). http://arxiv.org/abs/1409.1556
29. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC), pp. 147–156 (2019)
30. Szegedy, C., et al.: Intriguing properties of neural networks. Technical report (2013). http://arxiv.org/abs/1312.6199
31. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proceedings of 7th International Conference on Learning Representations (ICLR) (2019)
32. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. Technical report (2018). http://arxiv.org/abs/1804.10829