Towards Combining Deep Learning, Verification, and Scenario-Based Programming

Guy Katz
guykatz@cs.huji.ac.il
The Hebrew University of Jerusalem
Jerusalem, Israel

Achiya Elyasaf
achiya@bgu.ac.il
Ben-Gurion University of the Negev
Beersheba, Israel

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; **Robotics**; **Embedded systems**; • **Theory of computation → Logic and verification**; **Verification by model checking**; • **Software and its engineering → Software verification and validation**.

Deep learning (DL) [4] is dramatically changing the world of software. The rapid improvement in deep neural network (DNN) technology now enables engineers to train models that achieve superhuman results, often surpassing algorithms that have been carefully hand-crafted by domain experts [19, 20]. There is even an intensifying trend of incorporating DNNs in safety-critical systems, e.g. as controllers for autonomous vehicles and drones [1, 12].

Although DNN-based systems demonstrate excellent performance, they are far from perfect. It has been observed, in multiple domains, that systems that rely on DNN components can err dramatically when encountering situations they had not encountered before [21]. These errors are highly troubling if DNNs are to be used in critical autonomous systems; and they are detrimental to the wide adoption of these systems and their acceptance by regulators and the public. Unfortunately, DNN opacity prevents us from applying industry best practices for quality assurance, which are designed for hand-crafted code. There is thus an acute need for techniques and approaches for improving the reliability and maintainability of DNN-based systems.

One promising approach for tackling this difficulty is through formal verification: the rigorous and automated examination of a DNN-based system, in order to prove that it satisfies a specification. The formal verification of DNNs is a fairly new topic, which has received significant attention in recent years (e.g., [3, 10, 15]). A major barrier to DNN verification is *scalability*: the underlying decision problem is NP-complete, making it difficult to verify large DNNs [15]. Consequently, great efforts are being put into devising scalable verification tools, by using optimized decision procedures, parallelization, abstraction-refinement techniques, and others. However, an equally significant problem, which has received only limited attention, is how to make verification technology useful to engineers: namely, how to effectively integrate DNN verification tools into the development cycle of DNN-based systems [8]. One challenge is how to come up with meaningful specifications to verify; and another is how to use a discovered bug (e.g., a counter-example returned by the verifier) in order to correct and improve the system. Addressing these challenges is vital for the adoption of DNN verification technology.

We argue here that a promising way of tackling these challenges lies in drawing upon the many decades of work in software engineering (SE). Specifically, we argue that exploiting synergies between DL and SE could provide the means for domain experts to use their knowledge to generate meaningful specifications for DNNs. Additionally, appropriate SE infrastructure could assist in generalizing discovered bugs into code components that could be reintegrated into the system, thus paving the way for using verification to increase overall system reliability.

There have been recent attempts to combine deep learning with software engineering techniques, which are in line with our view — although these have not focused on verification. A notable approach is to augment ML-based systems with hand-crafted code modules [17, 18]. So far, this approach has mostly been applied to augment a DL model with ad-hoc, external *override rules*, which guard it from performing clearly-catastrophic mistakes. These encouraging attempts highlight the potential of augmenting DL with more advanced SE techniques.

In order to bridge the existing gap, we propose a more intensive integration between DL and SE — one that will allow hand-crafted modules to both *guide* the DL model by partaking in its training, and also to *guard* its actions after deployment. We believe that such methods, when combined with evolving DNN verification technology, will constitute a key ingredient in making DL sufficiently safe for deployment in safety-critical settings. Below we briefly describe some of the research directions that we are pursuing, as well as initial results.

Scenario-Based Programming. We identify the *scenario-based programming* (SBP) [9] paradigm as being particularly suited for integration with DL. SBP is a software development approach, particularly aimed at designing systems in a translucent, incremental and intuitive way, which is aligned with how humans perceive system requirements [5, 6, 9]. In SBP, a user specifies scenarios that represent behaviors that the system should include or avoid. Each scenario is standalone and self-contained, and concerns itself with one aspect of the system, typically a single requirement. The scenarios are interwoven at runtime, in a way that produces cohesive system behavior. An illustrative example appears in Fig. 1.

We propose to apply the SBP principles in devising a novel methodology for creating software systems that leverage big data on one hand, and domain knowledge on the other. Specifically, we

VARS'21, May 18, 2021, Nashville, TN, USA

ACM ISBN 978-1-4503-8444-5/21/05.
https://doi.org/10.1145/3459086.3459631

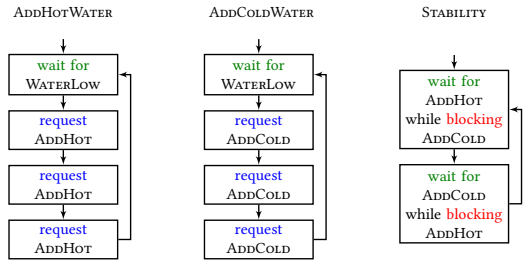

Figure 1: (From [7]) A scenario-based system that controls the water level in a tank. The AddHotWater scenario repeatedly waits for WaterLow events and requests three times the event AddHot, signifying the addition of hot water; and the AddColdWater scenario performs a symmetrical operation with cold water. To maintain the stability of the water temperature in the tank, the Stability scenario enforces the interleaving of AddHot and AddCold events by using event blocking. For additional details, see [7].

propose to (i) integrate domain-expert scenarios into the training process of DL-enabled systems, for *guiding* it; and (ii) augment the resulting DL model with additional scenarios that will *guard* it from making catastrophic errors. We argue that the special characteristics of SBP, and specifically the presence of intuitive and independent scenarios, make it highly suitable for this task.

We report on some of our initial, favorable results towards this goal. In [2], we demonstrated how a scenario-based program could improve the performance of a deep learning controller for a virtual robot soccer player. There, we used SBP to model a naïve player, by creating scenarios that specified how the robot player should (i) turn; (ii) move towards the ball; and (iii) grab the ball. Each action was also parametrized by its speed and force. Next, we added additional scenarios, specifying the goals of the player. Finally, we integrated the internal states of these scenarios into the DL training procedure, in order to create a robot player that could successfully grab the ball. We observed that the resulting controller more successfully learned the correct amount of speed and force required for effectively grabbing the ball than a controller generated purely using DL [2].

In another recent project [13], we demonstrated how SBP can be used for guarding DNN-based systems. Specifically, we targeted state-of-the-art computer network systems, and demonstrated how manually-crafted scenarios could be used to correct various bugs discovered, e.g., through verification [16]. We focused on two kinds of properties: *safety properties*, which ensure that bad things do not happen; and *liveness properties*, which ensure that good things eventually happen. For example, we studied a DNN-based congestion controller [11]: a system used by Internet servers to select their outgoing bit rates, with the goal of utilizing (without exceeding) all available bandwidth. One concern with such a system is that it might be overly conservative, i.e. choose a low sending rate even though higher rates could be used. Thus, we augmented this system with SBP components that ensured that this did not happen, by identifying such a situation and forcing the system to try out a higher sending rate [13]. In another example, we studied a DNN-based resource manager [17]: a system that controls CPU and memory resources, and assigns them to incoming jobs in order to maximize throughout. DNN verification has revealed that this system suffers from certain security vulnerabilities [16], and that specific inputs could cause it to repeatedly assign resources to non-existing jobs, effectively starving other, legitimate jobs. In [13, 14] we showed how the system could be augmented with SBP components that generalized, and prevented, such undesirable behavior.

These initial examples demonstrate the feasibility and potential of integrating SBP into various aspects of a DNN-based system's life cycle. Other directions that we are pursuing include the automated and semi-automated generalization of a discovered counter-examples into scenarios, and also the use of scenarios as specification artifacts for the DNN, to be repeatedly re-verified as the DNN is updated and extended.

REFERENCES

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. 2016. End to End Learning for Self-Driving Cars. Technical Report. http://arxiv.org/abs/1604.07316.
[2] A. Elyasaf, A. Sadon, G. Weiss, and T. Yaacov. 2019. Using Behavioral Programming with Solver, Context, and Deep Reinforcement Learning for Playing a Simplified RoboCup-Type Game. In *Proc. 22nd Int. IEEE Conf on Model Driven Engineering Languages and Systems (MODELS)*. 243–251.
[3] T. Gehr, M. Mirman, D. Drachsler-Cohen, E. Tsankov, S. Chaudhuri, and M. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*. 3–18.
[4] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press.
[5] D. Harel, G. Katz, R. Marelly, and A. Marron. 2016. An Initial Wise Development Environment for Behavioral Models. In *Proc. 4th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*. 600–612.
[6] D. Harel, G. Katz, R. Marelly, and A. Marron. 2018. Wise Computing: Toward Endowing System Development with Proactive Wisdom. *IEEE Computer* 51(2) (2018), 14–26.
[7] D. Harel, G. Katz, A. Marron, and G. Weiss. 2012. Non-Intrusive Repair of Reactive Programs. In *Proc. 17th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS)*. 3–12.
[8] D. Harel, A. Marron, and J. Sifakis. 2019. Autonomics: In Search of a Foundation for Next Generation Autonomous Systems. Technical Report. https://arxiv.org/abs/1911.07133.
[9] D. Harel, A. Marron, and G. Weiss. 2012. Behavioral Programming. *Comm. of the ACM (CACM)* 55, 7 (2012), 90–100.
[10] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. 2017. Safety Verification of Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*. 3–29.
[11] N. Jay, N. Rotman, P. Brighten Godfrey, M. Schapira, and A. Tamar. 2018. Internet Congestion Control via Deep Reinforcement Learning. In *Proc. 32nd Conf. on Neural Information Processing Systems (NeurIPS)*.
[12] K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer. 2016. Policy Compression for Aircraft Collision Avoidance Systems. In *Proc. 35th Digital Avionics Systems Conf. (DASC)*. 1–10.
[13] G. Katz. 2020. Guarded Deep Learning using Scenario-Based Modeling. In *Proc. 8th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*. 126–136.
[14] G. Katz. 2021. Augmenting Deep Neural Networks with Scenario-Based Guard Rules. *Communications in Computer and Information Science (CCIS)* 1361 (2021), 147–172.
[15] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*. 97–117.
[16] Y. Kazak, C. Barrett, G. Katz, and M. Schapira. 2019. Verifying Deep-RL-Driven Systems. In *Proc. 1st ACM SIGCOMM Workshop on Network Meets AI & ML (NetAI)*. 83–89.
[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proc. 15th ACM Workshop on Hot Topics in Networks (HotNets)*. 50–56.
[18] S. Shalev-Shwartz, S. Shammah, and A. Shashua. 2017. On a Formal Model of Safe and Scalable Self-driving Cars. Technical Report. https://arxiv.org/abs/1708.06374.

[19] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. Dieleman. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (2016), 484–489.

[20] K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. Technical Report. http://arxiv.org/abs/1409.1556.

[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2013. Intriguing Properties of Neural Networks. Technical Report. http://arxiv.org/abs/1312.6199.