Reaching Out Towards Fully Verified Autonomous Systems

Sriram Sankaranarayanan[1](✉), Souradeep Dutta[1], and Sergio Mover[2]

[1] University of Colorado, Boulder, USA
{srirams,souradeep.dutta}@colorado.edu
[2] Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France
smover@lix.polytechnique.fr

Abstract. Autonomous systems such as "self-driving" vehicles and closed-loop medical devices increasingly rely on learning-enabled components such as neural networks to perform safety critical perception and control tasks. As a result, the problem of verifying that these systems operate correctly is of the utmost importance. We will briefly examine the role of neural networks in the design and implementation of autonomous systems, and how various verification approaches can contribute towards engineering verified autonomous systems. In doing so, we examine promising initial solutions that have been proposed over the past three years and the big challenges that remain to be tackled.

Keywords: Formal verification · Autonomous systems · Constraint solvers

1 Introduction

This paper presents a brief overview of recent progress towards the verification of autonomous systems. A system is defined as *autonomous* if it can operate in a reliable manner without requiring "frequent" human intervention. As such, the definition encompasses a wide variety of autonomous systems that are characterized by varying levels of human involvement, including teleoperated surgical robotic systems that *translate* the surgeon's actions from a remote terminal into precise movements of the surgical instruments placed inside the body of the patient [31]; closed loop medical devices such as pacemakers and artificial insulin delivery systems; autonomous "self-driving" cars, and unmanned aerial vehicles (UAVs). The examples mentioned above clearly demonstrate that autonomous systems are *safety critical*: even as we expect these systems to operate with limited human intervention, we also expect them to perform in a provably safe manner despite uncertainties about the environment and the numerous limitations on the system's ability to sense, compute and actuate.

Over the past decade, machine learning approaches have become default "go-to" approaches for building autonomous systems. These approaches use a variety of mathematical and computational models that are trained during design time using input-output examples in a supervised manner, or continuously learn and adapt from "past mistakes" using ideas such as reinforcement learning. In both cases, the use of *neural network* models has become quite popular due to the ability of neural networks to approximate complex nonlinear functions and the availability of powerful optimization tools that can infer these models from the given data. Neural network models have been widely applied in a variety of tasks. For instance, feedforward neural network models are widely used to build *perception stacks* for autonomous vehicles that can be used to process large amounts of sensor data from cameras, Lidars and other sensors to recognize other vehicles, pedestrians, road signs and traffic lights [17,22]. Current "end-to-end" driving pipelines seek to go from raw sensor data directly to steering and throttle commands that can help drive the vehicle, skipping the need for a human designed controller [5]. In applications such as robotic surgery, neural networks can be potentially applied to enable decision support by monitoring pre-operative, intra-operative and post-operative data to minimize the overall risk at each stage of patient care [19]. Neural networks have also been used to predict future blood glucose levels to help make real-time treatment decisions for patients with type-1 diabetes [14].

The key challenge in all of these applications lies in building systems with neural network components that are also guaranteed to satisfy key safety and liveness properties, even in the presence of significant uncertainties in the environment. This challenge is significant, since autonomous systems are often too large and complex to reason about manually. Furthermore, besides the system, the operating environment can also be complex and uncertain. Finally, it is challenging to arrive at well-defined specifications for such systems. For instance, it is highly challenging to specify a deep neural-network based object detection component for a self driving car. Such a specification must describe a stream of images from a road scene and the output of the object detector for these scenes: a task that has not proven easy, to date.

Despite these challenges, the broad area of verified autonomous systems has rapidly gained prominence in the formal verification community. We will briefly examine existing approaches, their advantages and drawbacks. Despite these promising steps, a lot more work needs to be carried out to move this area forward.

2 Preliminaries: Neural Networks

In this section, we will briefly explain background on feedforward neural networks, their role in learning-enabled autonomous systems. Our presentation will be brief and at a high level. We refer the reader to standard textbooks for further details [18].

2.1 Neural Networks

Neural networks belong to a class of *connectionist* models that are loosely inspired by the way neurons are connected to each other in human and ani-

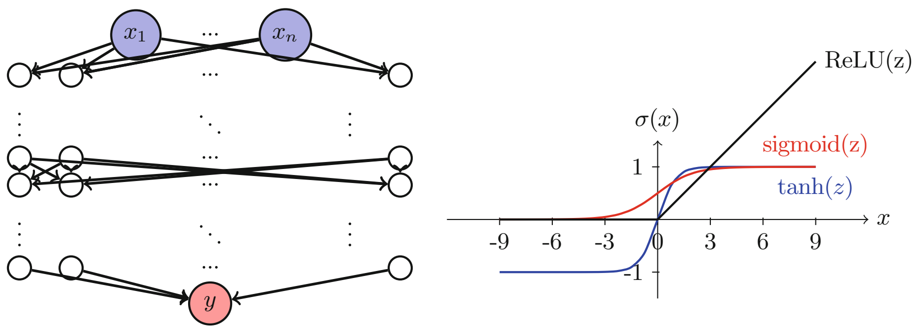

Fig. 1. (Left) A schematic diagram of a feedforward neural network with n inputs $x_1, \ldots, x_n$ and a single output y. Intermediate nodes are shown as unfilled circles. (Right) Commonly used activation functions.

mal brains. There are many types of neural networks, some including units that can store information. We classify neural networks broadly into two types: (a) *feedforward* networks: that do not have internal states; and (b) *recurrent* networks: that include units that can store information internally to the network. The difference between feedforward and recurrent networks is (roughly speaking) analogous to that between combinatorial boolean circuits and sequential circuits in digital logic. Most of our discussions in this paper will be centered around feedforward neural networks.

A feedforward network can be seen as a directed acyclic graph that represents the output as a function of the input. The nodes of this graph can be input nodes, output nodes or intermediate nodes. Each edge of the network is a directed edge from some node i to another node j with an associated real-valued weight $w_{i,j}$. The inputs to the network are fed to the input nodes, which do not have incoming edges. Likewise, the outputs are available at output nodes, which do not have any outgoing edges. Figure 1 shows a schematic diagram of a feedforward neural network.

Each intermediate node j of a feedforward network is associated with an activation function σ_j computed as follows:

1. Let $(i_1, j), \ldots, (i_k, j)$ be the incoming edges at node j, with associated weights $w_1, \ldots, w_k$ respectively.
2. Let $y_1, \ldots, y_k$ be the values computed at nodes $i_1, \ldots, i_k$, respectively.
3. The output at node j is given by $\sigma_j(\sum_{i=1}^{k} w_i y_i + b_j)$, wherein b_j is a constant called the *bias* at node j.

The activation functions associated with nodes are typically nonlinear functions. Popularly used functions are depicted in Fig. 1.

1. **ReLU:** The ReLU unit is defined by the activation function $\sigma(z)$: $\max(z, 0)$.
2. **Sigmoid:** The sigmoid unit is defined by the activation function $\sigma(z)$: $\frac{1}{1+e^{-z}}$.
3. **Tanh:** The activation function for this unit is $\sigma(z)$: $\tanh(z)$.

Note also that besides intermediate nodes with such activation functions, neural networks (especially networks used in image classification) employ specialized nodes such as *max-pooling* and *softmax* nodes that are not discussed here. They are explained in detail elsewhere [18]. A neural network computes a function of its inputs as follows: (a) the value of the input nodes are set according to the inputs to the network; (b) each intermediate node is enabled as soon as values are available at the target nodes for its incoming edges; and (c) once enabled, a node computes its output by applying its activation function. The computation terminates as soon as all output nodes are evaluated. Note that since the network is a DAG, a topological ordering of the nodes can be used to identify an evaluation order of the nodes in the network.

Neural networks have many desirable properties as universal function approximators: they can uniformly approximate any given continuous function f over a compact domain C to any desired accuracy [11]. Neural networks are used primarily for two important tasks: (a) *classify* an input into one of many discrete categories: for instance, categorize an image of a road sign as being a stop sign, a speed limit sign or a pedestrian crossing sign; and (b) *represent* a function from inputs to outputs learned from data through regression. Neural networks are applied in other ways besides just classification. For instance, networks can be used to identify a bounding box around objects of interest in a given image. Since the networks are too complex to design by hand, they are constructed by machine learning techniques that learn the weights and biases of the network given the *topology* of the network that includes the nodes, edges, the activation functions at each node; the input/output data in terms of training examples and a loss function that penalizes discrepancies between the output predicted by the neural network and the actual output in the training data.

There are many algorithms for “learning” the network weights and biases from given training data [18]. The most popular algorithms use variants of a strategy called *stochastic gradient descent* that updates the weights by calculating the gradient over a randomly chosen batch from the training data in order to achieve a local minimum for the loss function. Often, activation functions such as the ReLU function discussed above are *smoothed* in order to make it differentiable. There are many popular tools that automate the training process, notably TensorFlow and PyTorch [1,25]. These tools allow the user to create a neural network topology with unknown weights and biases, specify a loss function and perform the stochastic gradient descent. The networks are then evaluated on a “held-back” test data set that is not part of the data over which it was trained to evaluate its ability to generalize. The recent advent of GPUs that can perform rapid vector and matrix calculations along with the availability of large amounts of data has led to *deep neural networks* with hundreds of thousands of nodes.

3 Verification of Neural Networks

Even though deep neural networks, are essentially acyclic computation graphs formed by composing simple activation functions, the overall behavior of the network can be exceedingly complex and highly non-linear. In this section, we present a brief overview of the existing verification tools and techniques for neural networks and systems that incorporate neural networks in them.

In general, neural networks are used as components inside a closed loop autonomous system. As a result, verification problems have involved component-wise specification involving just the neural network or an end-to-end approach that studies the network in composition with other parts of the system. We distinguish different but closely related verification problems over neural networks: (a) BNNs have been shown to be quite effective for regression and classification tasks. The unit weights also yield computational savings and are amenable to implementation as digital circuits. One of the first attempts at verifying BNN's was proposed by Narodytska et al. [23]. Another recent approach proposed by Shih et al. [3] learns an Ordered Binary Decision Diagram (OBDD) locally to abstract parts of the neural networks. Cheng et al. [9] reduce the problem of BNN verification to hardware verification problem, and have reported speed ups in performance.

3.1 Abstract Interpretation for Neural Networks

Abstract interpretation originally formalized by Cousot and Cousot was developed to systematically propagate sets of reachable states of a program through individual program statements in order to establish properties of a program as a whole [10]. Such techniques rely on abstract domains to represent the reachable set of states [24]. This idea can be applied to neural networks which represent loop free computations involving the application of nonlinear activation functions.

Vechev et al. use zonotopes as an abstract domain to perform image computation across a neural network [16]. In particular, zonotopes are used to over-approximation the non-convex set of possible outputs for each layer of the network. This allows for a layer-by-layer analysis to compute sound over-approximations for the output of the neural network.

Xiang et al. that computes the output ranges as a union of convex polytopes [36]. This approach does not use SMT or MILP solvers unlike other approaches and thus can lead to highly accurate estimates of the output range. However, judging from preliminary evaluation reported, the cost of manipulating polyhedra is quite expensive, and thus, the approach is currently restricted to smaller networks when compared to SMT/MILP-based approaches.

Range computations using symbolic intervals were attempted in Reluval [33], which essentially relied on affine arithmetic techniques to reduce the over-approximation errors, and handle the case splitting imposed by ReLU units. Likewise, Cheng et al. [8] propose a heuristic approach to compute tight ranges for individual neurons.

3.2 Training with Robustness

Verification approaches have been incorporated to improve the process of learning networks from data [32,34]. For instance Jana et al. use the output set estimates computed by verification tools in order to incorporate robustness in the training phase wherein the network is rendered somewhat immune to small perturbations of the input. This has been proposed as a means to defend against any adversarial perturbations of the input. However, the computational cost can be orders magnitude more expensive than standard approaches to adversarially robust training that do not involve expensive verification tools in the loop.

3.3 Closed Loop Verification

Until this point we have been interested in verifying properties of a single neural network *in isolation*. However, as mentioned previously, autonomous systems employ neural networks as components in a closed loop that controls a physical process. Such physical processes can often be described by ordinary differential equations (ODEs). The simplest such situation involves a neural network that applies a feedback control to a physical process modeled as an ODE. This setup has been studied recently in order to perform reachability analysis of the resulting closed loop behaviors [13,21,30,35].

Dutta et al. [13], propose a technique to compute Taylor models (polynomial + error) as approximations of the behavior of the neural network in a compact domain. This was then used in conjunctions with standard reachability tools like Flow* [6] to compute reachable set of states of the closed loop involving an ODE and a neural network. A followup approach [20] approximates the neural network controller with Bernstein polynomials to deal with activation functions that are more general than ReLU. Ivanov et al. [21] propose a technique whereby activation functions such as sigmoid and tanh are modeled using differential equations evolving over time to encode a network as an ODE itself. This allows the transformation of a single layer of the neural network into a hybrid system. Which could then be used in standard reachability analysis tools for such systems. Another recent work by Xiang et al. considers the combination of neural networks in feedback with piecewise linear dynamical systems [37] using the techniques presented in [36].

Barrier certificates serve as an important approach to establish safety properties of dynamical systems [26]. Tuncali et al. [30] present an approach to synthesize barrier certificates using an SMT solver to prove properties of ODEs with neural networks as feedback.

However, neural networks are also employed in autonomous systems to classify a large volume of sensor data from cameras and LIDAR sensors. It is an enormous challenge to specify the behavior of these sensors with respect to changes in the environment and the vehicle. Shoukry et al. present a recent step towards verifying robotic systems that employ LIDAR sensors by means of simplifying the LIDAR system to consider a finite set of angles along with the system finds ranges [28]. The approach also “hard codes” a fixed environment with obstacles having fixed positions and geometries. The authors use a SMT based approach to construct a finite state abstraction of the closed loop system using fixed set of predicates to partition the state-space. This abstraction is then used to check reachability properties.

3.4 Falsification and Testing

We have focused our attention entirely on the use of formal verification approaches to prove properties of autonomous systems with neural network components. The problems of "best-effort" falsification to find counterexamples and that of systematic testing have also received a lot of attention. We mention a few representative approaches that relate closely to the verification approaches mentioned above without claiming to be a comprehensive survey on falsification/testing approaches for autonomous systems. An important line of work (e.g., [12,29,38]) focuses on the falsification problem for systems containing neural network components, as autonomous vehicles. The falsification problem consist of finding an execution of the system that violates a requirement, and the falsification algorithms for cyber-physical systems (e.g., S-TaLiRo [4]) implement efficient heuristics to search for a system's input that can falsify a requirement.

One challenge addressed in [38] is to find adversarial examples, a perturbation of the input that falsifies a temporal logic formula, for a closed loop control system formed by a neural network controller and a dynamical system. The proposed solution tries to find an adversarial example minimizing the robustness function of the Signal Temporal Logic (STL) formula via gradient descent.

Recent approaches also address the falsification of autonomous vehicles where neural networks are used in the perception stack. Dreossi et al. [12] propose an approach that falsifies STL formulas compositionally, first falsifying an abstraction of the neural network component and the cyber-physical system, and then confirming the counterexample in the neural network component. An alternative approach proposed by Fainekos et al. [29] focuses on perturbing driving scenarios for autonomous vehicles that can result in reaching undesired state (e.g., a crash). The scenarios are expressed in STL, and the approach generates input test cases from different combinations of discrete parameters of the system.

4 Challenges

We conclude our discussion by briefly mentioning some of the important challenges that remain to be tackled in this rapidly emerging area.

Specification: Despite initial approaches to verifying properties of neural networks in isolation, or as part of larger closed loops, the problem of formally specifying the behavior of these systems remains largely open for *perception* systems that classify sensor data including images and LIDAR data. The key challenge here lies in specifying what a valid image is in a logical formalism that is compatible with existing verification tools. This in turn requires a specification of the environment, and the imaging/sensing processes. To make matters more complicated, small changes to the orientation/pose of the vehicle can drastically alter the image generated. Current approaches sidestep functional specifications in favor of requiring the classifier to be "robust" to perturbations around some selected training examples. Alternatively, one may simplify the sensor's capabilities to make modeling easier. Another popular alternative uses generative models that specify inputs at a high level. Fremont et al. propose an approach that uses generative models for creating road scenes corresponding to simple programmatic specifications for the purposes of testing [15]. Extending such formalisms to verification problems remains an important challenge.

Scalability: Scalability of verification approaches remains yet another challenge. Simply put, the current state-of-the art networks are 100x or 1000x larger than the most efficient verification tools available. This gap needs to be considerably narrowed before verification approaches can be used on realistic systems. This challenge may requires to improve the existing verification techniques, for example improving the underlying constraint solvers by specializing them to handle neural networks. Alternative approaches such as using abstractions that are sufficient precise to show the correctness of the neural network can also be useful. The challenge lies in the definition of these abstractions and how they can be obtained for large networks without resorting to expensive verification tools in the first place.

Recurrent Networks: Another important challenge lies in tackling recurrent networks that involve units such as *long short term memory* (LSTM) with internal state. These networks are widely used in applications such as data-driven modeling and natural language processing. Verification of such networks is highly challenging for existing tools and techniques.

Runtime Verification: Runtime verification provides an important alternative to everything mentioned here that focuses on static/pre-deployment verification. The use of real-time monitors to predict and act against imminent property violations form the basis for runtime assurance using L1-Simplex architectures that switch between a lower performance but formally validated control when an impending failure is predicted [27]. However, the key issue lies in how impending failures are to be predicted. An alternative approach to verification to guarantee safety is shielding [2,40] that uses a supervisor (or so-called shield) to monitor the execution of the autonomous system and intervene to enforce temporal logic properties if a violation is imminent. Chen et al. present a different approach based on monitoring *viability* rather than safety in order to sidestep the need to reason about the controller [7]. Instead, their approach can perform lightweight reasoning just over the behavior of the plant model. A recent application of their approach involves monitoring geofences for unmanned aerial vehicle [39].

5 Conclusion

In conclusion, we have attempted to classify the rapidly emerging area of verifying autonomous systems involving neural networks. Our presentation has focused on some of the current successes and future challenges in this area.

Acknowledgments. This work was supported in part by the Air Force Research Laboratory (AFRL) and by the US NSF under Award # 1646556.

References

1. Abadi, M., Agarwal, A., Barham, P., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/
2. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding (2018). https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211
3. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by local automaton learning (2019). http://reasoning.cs.ucla.edu/fetch.php?id=193&type=pdf
4. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
5. Bojarski, M., et al.: End to end learning for self-driving cars. CoRR abs/1604.07316 (2016). http://arxiv.org/abs/1604.07316
6. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
7. Chen, X., Sankaranarayanan, S.: Model-predictive real-time monitoring of linear systems. In: IEEE Real-Time Systems Symposium (RTSS), pp. 297–306. IEEE Press (2017)
8. Cheng, C., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. CoRR abs/1705.01040 (2017). http://arxiv.org/abs/1705.01040
9. Cheng, C., Nührenberg, G., Ruess, H.: Verification of binarized neural networks. CoRR abs/1710.03107 (2017). http://arxiv.org/abs/1710.03107
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM Principles of Programming Languages, pp. 238–252 (1977)
11. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Sig. Syst. **2**, 303–314 (1989)
12. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 357–372. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_26
13. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Proceedings of the Hybrid Systems: Computation and Control (HSCC), HSCC 2019, pp. 157–168. ACM, New York (2019)

14. Dutta, S., Kushner, T., Sankaranarayanan, S.: Robust data-driven control of artificial pancreas systems using neural networks. In: Češka, M., Šafránek, D. (eds.) CMSB 2018. LNCS, vol. 11095, pp. 183–202. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99429-1_11
15. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: Proceedings of the ACM Programming Language Design and Implementation (PLDI), pp. 63–78 (2019)
16. 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: 2018 IEEE Symposium on Security and Privacy (SP), pp. 3–18, May 2018
17. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The Kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3354–3361, June 2012
18. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
19. Hashimoto, D.A., Rosman, G., Rus, D., Meireles, O.: Artificial intelligence in surgery: promises and perils. Ann. Surg. **268**, 70–76 (2018)
20. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: reachability analysis of neural-network controlled systems. CoRR abs/1906.10654 (2019). http://arxiv.org/abs/1906.10654
21. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Proceedings of the Hybrid Systems: Computation and Control (HSCC), HSCC 2019, pp. 169–178. ACM, New York (2019)
22. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 253–256, May 2010. https://doi.org/10.1109/ISCAS.2010.5537907
23. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. CoRR abs/1709.06662 (2017). http://arxiv.org/abs/1709.06662
24. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-662-03811-6
25. Paszke, A., et al.: Automatic differentiation in PyTorch. In: NIPS Workshop on Automatic Differentiation (2017). https://openreview.net/forum?id=BJJsrmfCZ
26. Prajna, S., Jadbabaie, A.: Safety verification using barrier certificates. In: Proceedings of the HSCC 2004, vol. 2993, pp. 477–492 (2004)
27. Sha, L.: Using simplicity to control complexity. IEEE Softw. **18**(4), 20–28 (2001)
28. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the Hybrid Systems: Computation and Control (HSCC), HSCC 2019, pp. 147–156. ACM, New York (2019)
29. Tuncali, C.E., Fainekos, G., Ito, H., Kapinski, J.: Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In: 2018 IEEE Intelligent Vehicles Symposium, pp. 1555–1562 (2018)
30. Tuncali, C.E., Kapinski, J., Ito, H., Deshmukh, J.V.: Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In: Proceedings of the Design Automation Conference, DAC 2018, pp. 30:1–30:6 (2018)
31. U.S Food and Drug Administration: Computer-assisted surgical systems (2019). https://www.fda.gov/medical-devices/surgery-devices/computer-assisted-surgical-systems. Accessed July 2019

32. Wang, S., Chen, Y., Abdou, A., Jana, S.: Mixtrain: scalable training of formally robust neural networks. CoRR abs/1811.02625 (2018). http://arxiv.org/abs/1811.02625
33. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. CoRR abs/1804.10829 (2018). http://arxiv.org/abs/1804.10829
34. Wong, E., Kolter, J.Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: Proceedings of the International Conference on Machine Learning, ICML, pp. 5283–5292 (2018). http://proceedings.mlr.press/v80/wong18a.html
35. Xiang, W., Tran, H., Johnson, T.T.: Reachable set computation and safety verification for neural networks with relu activations. CoRR abs/1712.08163 (2017). http://arxiv.org/abs/1712.08163
36. Xiang, W., Tran, H.D., Johnson, T.T.: Reachable set computation and safety verification for neural networks with relu activations (2107). https://arxiv.org/pdf/1712.08163.pdf. Posted on arxiv December 2017
37. Xiang, W., Tran, H.D., Rosenfeld, J.A., Johnson, T.T.: Reachable set estimation and verification for a class of piecewise linear systems with neural network controllers (2018). To Appear in the American Control Conference (ACC), invited session on Formal Methods in Controller Synthesis
38. Yaghoubi, S., Fainekos, G.: Gray-box adversarial testing for control systems with machine learning components. In: Proceedings of Hybrid Systems: Computation and Control, pp. 179–184 (2019)
39. Yoon, H., Chou, Y., Chen, X., Frew, E., Sankaranarayanan, S.: Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for UAVs (2019). In: Proceedings of the Runtime Verification 2019, October 2019 (to appear)
40. Zhu, H., Xiong, Z., Magill, S., Jagannathan, S.: An inductive synthesis framework for verifiable reinforcement learning. In: ACM Programming Language Design and Implementation (PLDI), pp. 686–701 (2019)