Formal Specification for Deep Neural Networks

Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont,
Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte,
and Xiangyu Yue

University of California, Berkeley

Abstract. The increasing use of deep neural networks in a variety of applications, including some safety-critical ones, has brought renewed interest in the topic of verification of neural networks. However, verification is most meaningful when performed with high-quality formal specifications. In this paper, we survey the landscape of formal specification for deep neural networks, and discuss the opportunities and challenges for formal methods for this domain.

1 Introduction

Deep neural networks (DNNs) are increasingly being deployed in domains where trustworthiness is a major concern, including automotive systems [41], health care [3], computer vision [35], and cyber security [13,53]. This increasing use of DNNs has brought with it a renewed interest in the topic of verification of neural networks, and more generally, in the topics of *verified artificial intelligence (AI)* and *AI safety* [47,52,4].

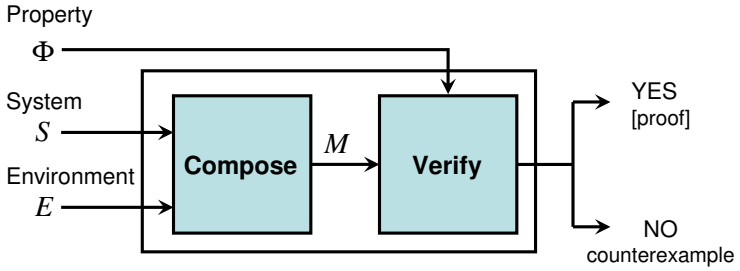

Fig. 1. Typical formal verification procedure: S is the system under verification, E is a model (or specification) of its environment, and Φ is the specification that system S must satisfy when composed with E.

Verification is most meaningful when performed with high-quality formal specifications, i.e., with a high-quality, mathematically rigorous specification of desired behavior that lends itself to algorithmic checking. As shown in Fig. 1, a typical formal verification procedure takes in not only a representation of the system under verification, but also the specification to be verified as well as a model (or specification) of the environment. Even as there is growing interest in the verification of DNNs (e.g., [32,20]), there

is suprisingly little that has been written about formal specification for deep neural networks, in particular about properties that are particularly relevant for neural networks as opposed to other types of systems.

In this paper, we seek to address this gap by exploring the landscape of formal specification for deep neural networks (DNNs). We begin by exploring the use cases of neural networks in learning-based systems today, presenting a brief taxonomy of DNN-based systems under verification. We then consider the literature on the design, (adversarial) analysis, and verification of DNNs. These works have implicitly or explicitly specified a variety of properties. We present these properties, organizing them along two dimensions. First, we present a *semantic* classification of properties, based on their meaning and relevance for the verification of systems based on deep neural networks. Second, we present a *trace-theoretic* classification, where we take the standard view of properties defined using sets of traces, and discuss how the various properties fit into those categories.

Our overall goal is to lay an initial foundation for formalizing and reasoning about properties of DNNs, and for using these properties in a rigorous design and verification methodology. We conclude with a brief discussion of challenges and opportunities for applying formal methods to the design and analysis of DNNs.[1]

2 Deep Neural Networks: Background and Use Cases

We are assuming that the reader is familiar with the basics of deep neural networks (DNNs). For those not familiar with DNNs, we suggest one of the books on the topic (e.g., [25]). The goal of this section is to define basic notation and describe common patterns of DNN-based systems.

2.1 Notation

We will use fairly standard notation about machine learning in the supervised setting.

Consider a sample space Z of the form $X \times Y$, and an ordered training set $S = ((x_i, y_i))_{i=1}^m$, where $x_i \in X$ is the data and $y_i \in Y$ is the corresponding label. Let H be a hypothesis space (e.g., a particular neural network architecture parameterized by a weight vector w). If the network computes a function from X to Y, we will denote it by f_w; i.e., $f_w(x) = y$. There is a loss (or risk) function $\ell : H \times Z \mapsto \mathbb{R}$ so that given a hypothesis $w \in H$ and a sample $(x,y) \in Z$, we obtain a loss $\ell(w,(x,y))$. We consider the case where we want to minimize the average loss over the training set S,

$$L_S(w) = \frac{1}{m} \sum_{i=1}^{m} \ell(w,(x_i,y_i)) \; + \; \lambda \mathcal{R}(w).$$

In the equation given above, $\lambda > 0$ and the term $\mathcal{R}(w)$ is called the *regularizer*; the latter seeks to enforce a notion of "simplicity" in w. Since S is fixed, we sometimes denote

[1] An early version of this paper appeared in [51].

$\ell_i(w) = \ell(w,(x_i,y_i))$ as a function only of w. The training problem is to find a w that minimizes $L_s(w)$; i.e., we wish to solve the following optimization problem:

$$\min_{w \in H} L_S(w)$$

This optimization problem is also sometimes termed *empirical risk minimization*.

2.2 DNN-Based Systems

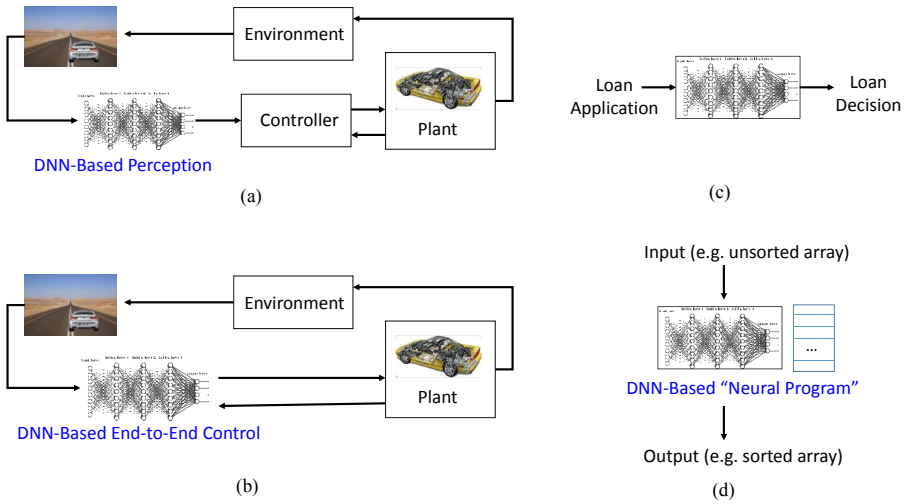

Fig. 2. Four use cases for DNNs in systems: (a) only for perception in a larger closed-loop system; (b) for end-to-end decision making, from perception to control, in a closed-loop system; (c) for open-loop decision making, and (d) for general-purpose programming.

DNNs have been used in a variety of systems. Figure 2 shows a selection of the types of DNN-based systems developed in research and development. Arguably their biggest impact to date has been in perceptual tasks, such as vision, natural language processing, speech recognition, etc. Thus, a major use case for DNNs is to perform perceptual tasks within the context of a larger closed-loop system, such as an autonomous vehicle, depicted in Fig. 2(a). An example of such a system is the automatic emergency braking system (AEBS) described by Dreossi et al. [15], where images taken by a camera mounted in front of an autonomous vehicle are fed to a DNN performing object detection and classification, whose output is sent to a controller that controls the steering angle and throttle of the autonomous vehicle. This vehicle then interacts with the rest of its environment (other vehicles, pedestrians, etc.) and the resulting interaction generates new sensor (image) data, closing the loop. In this case, the DNN is one component of a larger engineered system, which usually has its own specification that provides context for the design of the DNN.

The use of DNNs has also been demonstrated for so-called "end-to-end control", where neural networks go from sensor data to generating decisions and controlling actuation, as shown in Fig. 2(b). This example differs from Fig. 2(a) in that the DNN is used not just for perception, but also control. An example is an experimental self-driving system developed by a team at Nvidia [7].

Open-loop decision-making systems based on DNNs have also been proposed, such as a system that decides which loan applications to approve. This kind of system is depicted in Fig. 2(c). In this case, the DNN is the overall system under design and analysis.

Finally, the versatility of DNNs has also been demonstrated in general-purpose programming, such as learning programs for tasks such as sorting or string processing, shown in Fig 2(d). This use case for DNNs has specifications similar to those arising in traditional program verification problems.

There are other use cases for DNNs not shown in Fig. 2, such as the use of stateful neural networks (e.g., recurrent neural networks) or the use of DNNs for reinforcement learning, where the DNN is used for sequential prediction and decision-making tasks.

Each of these use cases throws up different requirements. We will discuss the corresponding kinds of formal specifications in the following section.

3 Semantic Classification

We classify properties of deep neural networks based on the type of semantic behavior they capture. Each semantic category appears in a separate sub-section below; however, we note that these are not strict partitions, and there are some properties that fall into multiple categories.

3.1 System-Level Specification

Several systems use DNNs as one component in a larger system targeting a particular application. For example, consider the use of a DNN for object detection in an autonomous vehicle. In such settings, the end goal can typically be captured naturally in terms of a *system-level* specification —- a property over the entire system that addresses the target application. As argued in recent papers (e.g. [52,18]), if the DNN is used for a perceptual task that mimics human perception, then it is very hard, if not impossible, to write a formal specification for that task. The overall system's specification, in contrast, can be described precisely, at least for engineered systems. Traditional specification formalisms, such as temporal logics, may be employed for the system-level specification.

An example of this approach is to specify the behavior of the automatic emergency braking system whose closed-loop diagram is shown in Fig. 2(a). The function of this system is to automatically actuate the brakes on the vehicle when it detects an environment object (obstacle) to be close. The objective is to maintain, at all times, a minimum safe distance between the autonomous vehicle (AV) and environment objects while the AV is in motion. We can write this specification in a standard specification language

such as signal temporal logic (STL), as follows:

$$\mathbf{G}\ [\text{AV\_moving} \Rightarrow \text{dist}(\mathbf{x}_{AV}, \mathbf{x}_{env}) > \Delta]$$

However, to scale to large systems, compositional (modular) reasoning is necessary. This poses a challenge to perform compositional verification in the absence of traditional, assume-guarantee style compositional specifications [50]. In prior work [15,16,50], we have shown how to derive constraints on the input space of the DNN from a system-level specification. However, these constraints are a guidance on where to search for counterexamples rather than a specification for the DNN itself.

3.2 Input-Output Robustness

In recent years, a significant amount of work has addressed the robustness (or lack thereof) of neural networks to so-called "adversarial perturbations" of their inputs (for example, [27,43,42,54,58,5,9,38]). Techniques used to demonstrate a lack of robustness are often referred to as "adversarial analysis."

Optimization Formulation of Local Robustness: A common approach to adversarial analysis involves solving an optimization problem of the following form, given a fixed input x:

$$\begin{aligned} & \min_{\delta} \mu(\delta) \\ \text{s.t.} \quad & \delta \in \Delta \\ & f_w(x+\delta) \in T(x) \end{aligned} \tag{1}$$

Here μ is a cost function defined on the perturbations, typically a distance metric based on a norm ($L_1, L_2, or L_\infty$), Δ is a constrained domain set for δ, the constraint $f_w(x+\delta) \in T(x)$ ensures that the output of the NN to the perturbed input lies in the adversary's *target output set* $T(x)$ (which can be a function of x, e.g., $Y \setminus \{y\}$ where y is the correct label). Typically Δ is set to be the same as the domain of x, e.g., $\mathbb{R}^n$. This property is referred to as "local" robustness since it concerns robustness around a given input x.

For a recent survey (from a formal methods perspective) of techniques for analyzing robustness, see [18].

Decision Formulation of Local Robustness: The decision version of this optimization problem states that, given a bound β and input x, the adversarial analysis problem is to find a perturbation δ such that the following formula is satisfied:

$$\varphi(\delta) \doteq \mu(\delta) < \beta \wedge \delta \in \Delta \wedge f_w(x+\delta) \in T(x)$$

In other words, the robustness property is the negation of the above formula, $\neg\varphi(\delta)$:

$$[\mu(\delta) < \beta \wedge \delta \in \Delta] \Rightarrow [f_w(x+\delta) \notin T(x)]$$

Global Robustness: One can generalize the previous notion of robustness by universally quantifying over all inputs x, to get the following formula, for a fixed β:

$$\forall x.\ \forall \delta.\ \neg\varphi(\delta)$$

This is referred to as "global" robustness as we are not limited to analyzing robustness around a fixed point.

An alternative formulation of global robustness involves specifying that the DNN outputs a similar answer on all pairs of inputs (x_1, x_2) that are "close", as follows:

$$\forall x_1, x_2.\ [\mu(x_1 - x_2) < \beta \wedge (x_1 - x_2) \in \Delta] \Rightarrow [f_w(x_1) \approx f_w(x_2)]$$

where "$\approx$" is a suitably-defined notion of similarity between outputs of the DNN.

Loss-based Robustness: Another formulation (e.g., [38]) involves finding a δ that maximizes the loss:

$$\mathbb{E}_{(xy)\sim D}[\max_{\delta \in \Delta} \ell(w, (x+\delta, y))] \tag{2}$$

where D is the distribution of the input space. In [38], the authors use the L_∞ norm to describe Δ as a bounded neighborhood around x.

This is a probabilistic formulation that involves knowledge of the distribution. In the absence of such knowledge, one may consider the worst case over the (x, y) space.

Additional Robustness Properties: Other authors have proposed alternative definitions of robustness in the literature. For instance, Bastani et al. [5] define notions of *adversarial frequency* (how often the DNN fails to be locally robust) and *adversarial severity* (the average robustness value exceeding a given threshold, averaged over inputs x chosen from some given input distribution). Cheng et al. [11] provide a definition of *maximum resilience* that is a global notion of robustness applying to multi-classification DNNs.

While these notions of robustness have been useful in demonstrating the limitations of DNNs for classification and other prediction tasks, as has been recently argued [18], they are not enough by themselves. We need to tie them to the overall application semantics. We discuss this point further in Sec. 3.11.

3.3 Input-Output Relations

Feedforward neural networks are programs that compute functions of their input. For such programs, one can write formal specifications in the standard manner: assuming a pre-condition on the inputs, $P(x)$, guarantee a post-condition $Q(x, y)$, i.e., $\forall x, y.\ P(x) \Rightarrow Q(x, y)$, where x and y are the inputs and outputs of the DNN respectively.

Researchers have identified special cases of pre/post-condition pairs for deep neural networks. For example, Dutta et al. [19] analyze properties of the form $P(x) \implies Q(y)$ where P and Q are restricted to certain kinds of geometric regions. Similarly, Dvijotham et al. [20] give examples of a similar class of restricted pre/post-condition pairs. These are typically partial specifications of sequential program correctness.

Deep neural networks are being used for other kinds of functional computations, such as neural Turing machines [30] and other neural programming architectures [8]. This case is depicted in Fig. 2(d). For these programs and formalisms, traditional classes of functional program specifications, those that provide complete specifications of program behavior, will also apply.

3.4 Semantic Invariance

For some applications, the input space X can be partitioned into equivalence classes $X_1, X_2, X_3, \ldots$, such that for each equivalence class $X_i \subseteq X$, and pair of inputs $x_{i1}, x_{i2} \in X_i$, we require that $f_w(x_{i1}) = f_w(x_{i2})$.

For instance, consider a DNN that must detect whether or not there is a car in an image. One may want to specify that the binary output of the network $(\texttt{car}, \neg\texttt{car})$ be invariant to translation or scaling of objects in the image. Examples of such properties are typically domain-specific. We refer to such properties as *semantic invariance*, an example of which is *geometric invariance* (see, for example, [14,37,34,21,26]).

3.5 Monotonicity

In certain applications, the input space X admits a natural partial order $\preceq$, and one expects the output of the classifier to be monotonic with respect to this ordering. A common example is a DNN used for approving loan applications: if Applicant A's income is strictly greater than Applicant B's, all else being equal, then one might expect that A's application would be granted if B's was.

One can formalize this property as follows:

$$\forall x_1, x_2 \in X . x_1 \preceq_X x_2 \implies f_w(x_1) \preceq_Y f_w(x_2)$$

where $\preceq_X$ indicates a preference order on X while $\preceq_Y$ denotes such an ordering on the output space Y.

For examples of papers discussing monotonicity properties, see [59,20].

3.6 Fairness

Over the last decade, there is a growing literature on the need to ensure that machine learning (ML) systems produce outputs that are "fair" in some way. The notion of fairness typically has to do with certain attributes of the input vector x being sensitive, and that the decisions should not be influenced (perhaps in a statistical way) by those sensitive attributes. For DNNs, fairness is typically discussed in the context of decision-making systems similar to the one shown in Fig. 2(c).

This is still an evolving area, and there are many different formulations of fairness; see, e.g., [31,23,6,2,1,36,24]. One aspect shared by many is that they are probabilistic properties.

One class of fairness properties are *similarity-based fairness* properties, such as *individual fairness* (IF), which states that the neural network (ML model) maps similar inputs to similar outputs. This shares similarities with semantic invariance and robustness, except that the notion of similarity is different.

Another class of fairness properties are defined at the population level. An example is *demographic parity* which states that the probability of getting a particular output value is independent of the values of the sensitive attributes. In this respect, this property shares similarities with the notion of *non-interference* that has been researched in the formal methods and programming languages literature.

Yet another notion of fairness is counterfactual, relying on causal models (e.g. [36]). In this version of fairness, a decision output by a DNN is fair towards a particular input (individual) if that decision is the same in both the actual world and a counterfactual world where the input has a different value for one or more "protected" attributes (features).

3.7 Input/Distributional Assumptions

Many theoretical guarantees about machine learning algorithms are predicated on the assumption that the learned model is tested only on input drawn from the distribution it was trained on. Such distributional assumptions therefore form an important class of specifications. A specification language that captures such assumptions must inherently be probabilistic. We belive probabilistic programming languages (e.g., [29,39,10,45,28]), offer a natural and expressive way to specify distributions over the input space, and are thus a natural fit for such specifications. As an example, a recent probabilistic programming language for specifying input scenarios that can be used to generate input data for neural networks is described in [22].

3.8 Coverage Criteria

Formal specification can be useful even for testing or semi-formal verification of a system. This has been amply demonstrated in the design of digital circuits, where simulation-based verification of temporal logic assertions is standard. In this setting, formal specifications are often used to formalize *design coverage* objectives, e.g., to ensure that certain conditions are activated by a test suite.

We believe formal specifications could play a similar role for the analysis of DNNs. It is still unclear what sort of coverage properties are required. Some initial progress on coverage-driven testing of DNNs has been reported by Pei et al. [44].

3.9 Temporal Specifications

Stateful neural networks, such as recurrent neural networks (RNNs), essentially implement state machines. For such neural networks, the formalisms used to specify properties of state machines, and more broadly, of reactive systems, would apply. Temporal logics provide a suitable formalism to specify properties of such systems. An example of previous work in such a direction is that of Rodrigues et al. [46], while Taylor and Farrah [55] describe extracting rules from neural networks for verification, testing, and other purposes.

Another use of DNNs for stateful systems exhibiting temporally-varying behavior is in *deep reinforcement learning* (e.g., see [40]). In reinforcement learning (RL), an intelligent agent interacts with its environment through actions, observations and rewards [33]. Traditionally, specifications for RL have been given as quantitative objective (cost and reward) functions; however, there is also a large body of work on using temporal logics for specifying RL objectives (see, e.g., [48,57]),

3.10 Specifications on Learning Algorithms

Finally, one might want to specify properties on the learning algorithms themselves (and their implementations), rather than on specific learned models. Stochastic gradient descent (SGD) is a commonly used algorithm for training DNNs. As an example, we point out the recent work by Selsam et al. [49] on using interactive theorem proving to detect errors in systems that implement machine learning algorithms based on stochastic computation graphs.

3.11 Bridging System-Level Specifications with Component-Level Specifications

It has been recently observed [18] that although adversarial analysis of DNNs is useful, it is not sufficient. The relevance of adversarial attacks can be questioned when the impact on the overall system within which the DNN is used is unclear. Not all misclassifications are equally important. Thus, it is necessary to increase the use of application-level or system-level *semantics* in adversarial analysis and design of DNNs. There is a need to bridge system-level specifications with component-level specifications.

To this end, we believe it is important to devise a good notion of *semantic robustness* of DNNs to adversarial perturbations of the input. In order to do this, one needs to define the *semantic feature space* of the DNN — i.e., the feature space that captures application-level semantics and not just raw inputs (e.g., the pixel space for images). The raw input is obtained from the semantic feature vector through a process of "rendering", where we borrow the term from the rendering of images from high-level semantic configurations. As an example, consider the application of a DNN to perform object detection and classification in images captured for autonomous driving. In this case, the semantic feature space is one that captures high-level semantics of the scene around the vehicle – i.e., other agents that are present (cars, pedestrians, bicyclists, etc.) and their properties, parameters of the road and traffic scene, and other relevant characteristics. In this respect, this problem is similar to that of capturing input assumptions (see Sec. 3.7), although the emphasis here is more on the semantic features of the environment and less on the underlying distributions.

Let S represent the semantic feature space. Given $s \in S$, we obtain an input $x \in X$ by a process we will call *rendering* or *concretization*. (X is sometimes referred to as the "concrete feature space" to distinguish it from S.) Let R denote the rendering procedure; i.e., $R(s) = x$. Then, we introduce a notion of (global) *semantic robustness* as follows:

$$\forall s, s', x, x'.\ [s \approx_S s' \wedge R(s) = x \wedge R(s') = x'] \Rightarrow [f_w(x) \approx f_w(x')]$$

Similar to the notion of robustness described in Sec. 3.2, the above definition is based on a notion of similarity in the semantic feature space ($\approx_S$) and one on the output space of the DNN ($\approx$). Such a notion of similarity may well be based on a suitably-chosen norm and bound such as β or δ used in Sec. 3.2. However, we prefer the more abstract version given above given that much more work remains to be done in characterizing semantic feature spaces, the rendering process, and the relation of S and R to the operation of the DNN. Further, the rendering procedure R may take in additional parameters (similar to those of the DNN w), which we hide here for simplicity.

Initial work on defining semantic feature spaces and bridging system-level specifications with component-level ones is just emerging. To our knowledge, the first work in this direction was [15,16], which uses a simple "modification" space to represent semantic transformations to images. Fremont et al. [22] present a more expressive language to capture semantic properties of a scene. However, these are very preliminary results, and much more remains to be done, as described in [18].

4 Trace-Theoretic Classification

We conclude with a brief categorization of the above types of properties with respect to their trace-theoretic nature.

Most properties in the formal methods literature tend to be *trace properties*; i.e., the property is equivalent to specifying a set of correct or desired behaviors of the system. For such properties, one can examine a single trace (input-output behavior) of the system and determine whether or not it violates the property.

However, certain properties are not trace properties, but are instead characterized as sets of trace sets (or a set of correct systems) — these are called *hyperproperties* [12]. Notable examples of such properties include determinism and security properties such as confidentiality and integrity. For such properties, one must examine an ensemble of two or more traces in order to determine whether the property has been violated. Hyperproperties cover all non-trace properties.

4.1 Trace Properties

Several system-level properties, such as those specified in linear temporal logic or metric temporal logics, are trace properties. Similarly input-output relations, temporal specifications for stateful NNs, and specifications on machine learning algorithms tend to be trace properties. Input-output robustness for a fixed input is a trace property. Certain coverage properties can be evaluated over single traces (e.g., whether specific neurons were activated on an input).

4.2 Hyperproperties

Some system-level properties, such as those specifying security policies, can be hyperproperties. Input-output robustness in the general case (for all inputs) is a hyperproperty; one must examine all pairs of inputs to determine if the system is robust. Similarly, semantic invariance and monotonicity involve reasoning over pairs of (related) traces. We note that all of the hyperproperties in this context are so-called two-safety properties, and so in theory are not much harder to verify or test than ordinary safety properties [56].

Fairness and average-case robustness are also hyperproperties, but of a probabilistic nature. Distributional assumptions on the input space are also properties of an ensemble of traces. Finally, some coverage properties are aggregate measures over sets of traces and thus are naturally hyperproperties.

5 Conclusion

In order to understand the design and verification problem for deep neural networks, it is essential to have a good understanding of the landscape of formal specification for DNNs. In this paper, we have presented a classification of the kinds of specifications that have been found useful for reasoning about neural networks and the systems that employ them. This serves as a starting point for creating a more systematic design methodology for DNNs.

Formal specifications can be used not only for verification and testing, but also for retraining, e.g., using counterexamples [17], or by using specification-guided cost functions or features (say by augmenting the regularizer $\mathcal{R}(w)$) in the training process. Specifications are also crucial to capture, in a rigorous manner, the assumptions made during the design process of DNNs, so that these can be taken into account during the design and operation of the overall system containing the DNN. We believe the field of formal methods for the design and analysis of deep neural networks, and of machine learning systems in general, will be a rich domain for research for the foreseeable future, and that formal specification will play a foundational role in this research.

Acknowledgments

The work of the authors on this paper was funded in part by the NSF VeHICaL project (#1545126), NSF projects #1646208 and #1739816, NSF Graduate Research Fellowships, DARPA under agreement number FA8750-16-C0043, the DARPA Assured Autonomy program, Berkeley Deep Drive, and by Toyota under the iCyPhy center. This paper was the outcome of discussions amongst the co-authors in early 2018. It has additionally benefited from conversations with Somesh Jha, Susmit Jha, Pushmeet Kohli, Aditya Nori, Jerry Zhu, and several participants in Dagstuhl Seminar 18121.

References

1. Albarghouthi, A., D'Antoni, L., Drews, S., Nori, A.: Fairness as a program property (2016), arXiv:1610.06067
2. Albarghouthi, A., D'Antoni, L., Drews, S., Nori, A.V.: Fairsquare: probabilistic verification of program fairness. Proceedings of the ACM on Programming Languages (2017)
3. Alipanahi, B., Delong, A., Weirauch, M.T., Frey, B.J.: Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. Nature biotechnology (2015)
4. Amodei, D., Olah, C., Steinhardt, J., Christiano, P.F., Schulman, J., Mané, D.: Concrete problems in AI safety. ArXiV e-prints abs/1606.06565 (2016)
5. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems (NIPS) 29. pp. 2613–2621 (2016)
6. Binns, R.: Fairness in machine learning: Lessons from political philosophy (2017), arXiv:1712.03586
7. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)

8. Cai, J., Shin, R., Song, D.: Making neural programming architectures generalize via recursion. arXiv preprint arXiv:1704.06611 (2017)
9. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: IEEE Symposium on Security and Privacy (SP) (2017)
10. Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., Riddell, A.: Stan: A probabilistic programming language. Journal of statistical software 76(1) (2017)
11. Cheng, C.H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 251–268. Springer (2017)
12. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security 18(6), 1157–1210 (Sep 2010)
13. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3422–3426. IEEE (2013)
14. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. IEEE International Conference on Computer Vision (2017)
15. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. In: NASA Formal Methods Symposium (2017)
16. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. CoRR abs/1703.00978 (2017), http://arxiv.org/abs/1703.00978
17. Dreossi, T., Ghosh, S., Yue, X., Keutzer, K., Sangiovanni-Vincentelli, A., Seshia, S.A.: Counterexample-guided data augmentation. In: 27th International Joint Conference on Artificial Intelligence (IJCAI) (2018)
18. Dreossi, T., Jha, S., Seshia, S.A.: Semantic adversarial deep learning. In: 30th International Conference on Computer Aided Verification (CAV) (2018)
19. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks (2017), arXiv:1709.09130
20. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., Kohli, P.: A dual approach to scalable verification of deep networks (2018), arXiv:1803.06567
21. Fawzi, A., Frossard, P.: Manitest: Are classifiers really invariant? (2017), arXiv:1507.06535
22. Fremont, D., Yue, X., Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: Language-based scene generation. Tech. Rep. UCB/EECS-2018-8, EECS Department, University of California, Berkeley (Apr 2018), http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-8.html
23. Friedler, S.A., Scheidegger, C., Venkatasubramanian, S.: On the (im) possibility of fairness (2016), arXiv:1609.07236
24. Friedler, S.A., Scheidegger, C., Venkatasubramanian, S., Choudhary, S., Hamilton, E.P., Roth, D.: A comparative study of fairness-enhancing interventions in machine learning (2018), arXiv:1802.04422
25. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://goodfeli.github.io/dlbook/
26. Goodfellow, I., Lee, H., Le, Q.V., Saxe, A., Ng, A.Y.: Measuring invariances in deep networks. In: Advances in Neural Information Processing Systems (2009)
27. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014), arXiv:1412.6572
28. Goodman, N.D., Mansinghka, V.K., Roy, D., Bonawitz, K., Tenenbaum, J.B.: Church: A language for generative models. In: Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence. pp. 220–229. UAI'08 (2008)

29. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: FOSE 2014. pp. 167–181. ACM (2014)
30. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv preprint arXiv:1410.5401 (2014)
31. Hardt, M., Price, E., Srebro, N., et al.: Equality of opportunity in supervised learning. In: Advances in Neural Information Processing Systems (2016)
32. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International Conference on Computer Aided Verification. pp. 3–29. Springer (2017)
33. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)
34. Kanbak, C., Moosavi-Dezfooli, S.M., Frossard, P.: Geometric robustness of deep networks: analysis and improvement (2017), arXiv:1711.09115
35. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
36. Kusner, M.J., Loftus, J., Russell, C., Silva, R.: Counterfactual fairness. In: Advances in Neural Information Processing Systems (2017)
37. Lowe, D.G.: Object recognition from local scale-invariant features. In: IEEE International Conference on Computer Vision (1999)
38. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks (2017), arXiv:1706.06083
39. Milch, B., Marthi, B., Russell, S.: Blog: Relational modeling with unknown objects. In: ICML 2004 workshop on statistical relational learning and its connections to other fields. pp. 67–73 (2004)
40. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529 (2015)
41. NVIDIA: Nvidia tegra drive px: Self-driving car computer (2015), http://www.nvidia.com/object/drive-px.html
42. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Proceedings of the 1st IEEE European Symposium on Security and Privacy. arXiv preprint arXiv:1511.07528 (2016)
43. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. arXiv preprint arXiv:1511.04508 (2015)
44. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 1–18. ACM (2017)
45. Pfeffer, A.: Figaro: An object-oriented probabilistic programming language. Tech. rep., Charles River Analytics (2009)
46. Rodrigues, P., Costa, J.F., Siegelmann, H.T.: Verifying properties of neural networks. In: International Work-Conference on Artificial Neural Networks. pp. 158–165. Springer (2001)
47. Russell, S., Dietterich, T., Horvitz, E., Selman, B., Rossi, F., Hassabis, D., Legg, S., Suleyman, M., George, D., Phoenix, S.: Letter to the editor: Research priorities for robust and beneficial artif icial intelligence: An open letter. AI Magazine 36(4) (2015)
48. Sadigh, D., Kim, E.S., Coogan, S., Sastry, S., Seshia, S.A.: A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In: Proceedings of the 53rd IEEE Conference on Decision and Control (CDC). pp. 1091–1096 (December 2014)
49. Selsam, D., Liang, P., Dill, D.L.: Developing bug-free machine learning systems with formal mathematics. In: International Conference on Machine Learning. pp. 3047–3056 (2017)

50. Seshia, S.A.: Compositional verification without compositional specification for learning-based systems. Tech. Rep. UCB/EECS-2017-164, EECS Department, University of California, Berkeley (Nov 2017), http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-164.html
51. Seshia, S.A., Desai, A., Dreossi, T., Fremont, D., Ghosh, S., Kim, E., Shivakumar, S., Vazquez-Chanlatte, M., Yue, X.: Formal specification for deep neural networks. Tech. Rep. UCB/EECS-2018-25, EECS Department, University of California, Berkeley (May 2018), http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-25.html
52. Seshia, S.A., Sadigh, D., Sastry, S.S.: Towards Verified Artificial Intelligence. ArXiv e-prints (July 2016)
53. Shin, E.C.R., Song, D., Moazzezi, R.: Recognizing functions in binaries with neural networks. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 611–626 (2015)
54. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013), arXiv:1312.6199
55. Taylor, B.J., Darrah, M.A.: Rule extraction as a formal method for the verification and validation of neural networks. In: IEEE International Joint Conference on Neural Networks (IJCNN). vol. 5, pp. 2915–2920. IEEE (2005)
56. Terauchi, T., Aiken, A.: Secure information flow as a safety problem. In: International Static Analysis Symposium. pp. 352–367. Springer (2005)
57. Wen, M., Ehlers, R., Topcu, U.: Correct-by-synthesis reinforcement learning with temporal logic constraints. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. pp. 4983–4990 (2015)
58. Weng, T.W., Zhang, H., Chen, P.Y., Yi, J., Su, D., Gao, Y., Hsieh, C.J., Daniel, L.: Evaluating the robustness of neural networks: An extreme value theory approach (2018), arXiv:1801.10578
59. You, S., Ding, D., Canini, K., Pfeifer, J., Gupta, M.: Deep lattice networks and partial monotonic functions. In: Advances in Neural Information Processing Systems (2017)