Formal Specification for Deep Neural Networks

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

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-25
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-25.html

May 3, 2018

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
Xiangyu Yue
EECS Department, University of California, Berkeley

May 2, 2018

Abstract

The increasing use of deep neural networks (DNNs) 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 only meaningful when paired with high-quality formal specifications. In this note, we survey the landscape of formal specification for deep neural networks. Our 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.

1 Introduction

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. Verification is only meaningful when paired with high-quality formal specification. Even so, relatively little has been written about formal specification for deep neural networks.

In this paper, we survey the landscape of formal specification for deep neural networks (DNNs). The literature on the design, (adversarial) analysis, and verification of DNNs has 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 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 are assuming that the reader is a 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., [15]). In the following sections, 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 $S = ((x_i, y_i))_{i=1}^{m}$ ($x_i \in X$ 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* and enforces a notion of "simplicity" in w. Since S is fixed, we sometimes denote $\ell_i(w) = \ell(w,(x_i,y_i))$ 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 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 properties that fall into multiple categories.

2.1 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. [32, 8]), if the DNN is used for a perceptual task that mimicks 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 for engineered systems. Traditional specification formalisms, such as temporal logics, may be employed for the system-level specification. 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 [31].

2.2 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 (e.g., [17, 27, 26, 33, 35, 5, 24]). Techniques used to demonstrate a lack of robustness are often referred to as "adversarial analysis."

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., $\mathbb{R}^n$.

The decision version of this optimization problem states that, given a bound β and input x, we wish to find a perturbation δ such that the following formula is satisfied:

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

Thus, the robustness property (for a fixed β) is of the form:

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

An alternative, equivalent formulation involves reasoning a over pairs of inputs (x_1, x_2), as follows:

$$\forall x_1, x_2.\ \ \neg\varphi(x_1 - x_2)$$

Another formulation (e.g., [24]) 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 [24], 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.

2.3 Input-Output Relation

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 $Q(x, y)$.

Researchers have identified special cases of pre/post-condition pairs for deep neural networks. For example, Dutta et al. [9] 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. [10] give examples of a similar class of restricted pre/post-condition pairs.

Deep neural networks are being used for other kinds of functional computations, such as neural Turing machines [19] and other neural programming architectures [4]. For these programs and formalisms, traditional classes of functional program specifications will also apply.

2.4 Semantic Invariance

For some applications, the input space X can be partitioned into equivalence classes 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, [7, 23, 21, 11, 16]).

2.5 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 [36, 10].

2.6 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.

This is still an evolving area, and there are many different formulations of fairness; see, e.g., [20, 13, 3, 2, 1, 22, 14]. 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. [22]).

2.7 Input/Distributional Assumptions

Many theoretical guarantees about machine learning algorithms are predicted on the assumption that the learned model is tested only on input drawn from the same distribution that it was trained on. Such distributional assumptions therefore form an important class of specifications. A property language that captures such assumptions must inherently be probabilistic. Probabilistic programming languages (e.g., [18, 25]) and probabilistic logics are thus a natural fit for such specifications. A recent example of a probabilistic programming language for specifying scenarios that can be used to generate input data for neural networks is given in [12].

2.8 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. [28].

2.9 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. [29], while Taylor and Farrah [34] describe extracting rules from neural networks for verification, testing, and other purposes.

2.10 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. [30] on using interactive theorem proving to detect errors in systems that implement machine learning algorithms based on stochastic computation graphs.

3 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* [6]. 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.

3.1 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).

3.2 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.

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.

Acknowledgments

This work was supported in part by NSF grants 1545126 and 1646208, the DARPA BRASS and Assured Autonomy projects, and by the iCyPhy center.

References

[1] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya Nori. Fairness as a program property. 2016. arXiv:1610.06067.

[2] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V Nori. Fairsquare: probabilistic verification of program fairness. *Proceedings of the ACM on Programming Languages*, 2017.

[3] Reuben Binns. Fairness in machine learning: Lessons from political philosophy. 2017. arXiv:1712.03586.

[4] Jonathon Cai, Richard Shin, and Dawn Song. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611*, 2017.

[5] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 2017.

[6] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, September 2010.

[7] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *IEEE International Conference on Computer Vision*, 2017.

[8] Tommaso Dreossi, Somesh Jha, and Sanjit A. Seshia. Semantic adversarial deep learning. In *30th International Conference on Computer Aided Verification (CAV)*, 2018. To appear.

[9] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. 2017. arXiv:1709.09130.

[10] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. 2018. arXiv:1803.06567.

[11] Alhussein Fawzi and Pascal Frossard. Manitest: Are classifiers really invariant? 2017. arXiv:1507.06535.

[12] Daniel Fremont, Xiangyu Yue, Tommaso Dreossi, Shromona Ghosh, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: Language-based scene generation. Technical Report UCB/EECS-2018-8, EECS Department, University of California, Berkeley, Apr 2018.

[13] Sorelle A Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. On the (im) possibility of fairness. 2016. arXiv:1609.07236.

[14] Sorelle A Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P Hamilton, and Derek Roth. A comparative study of fairness-enhancing interventions in machine learning. 2018. arXiv:1802.04422.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[16] Ian Goodfellow, Honglak Lee, Quoc V Le, Andrew Saxe, and Andrew Y Ng. Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems*, 2009.

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2014. arXiv:1412.6572.

[18] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *FOSE 2014*, pages 167–181. ACM, 2014.

[19] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[20] Moritz Hardt, Eric Price, Nati Srebro, et al. Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems*, 2016.

[21] Can Kanbak, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Geometric robustness of deep networks: analysis and improvement. 2017. arXiv:1711.09115.

[22] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In *Advances in Neural Information Processing Systems*, 2017.

[23] David G Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, 1999.

[24] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. 2017. arXiv:1706.06083.

[25] Brian Milch, Bhaskara Marthi, and Stuart Russell. Blog: Relational modeling with unknown objects. In *ICML 2004 workshop on statistical relational learning and its connections to other fields*, pages 67–73, 2004.

[26] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 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.

[27] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1511.04508*, 2015.

[28] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[29] Pedro Rodrigues, J Félix Costa, and Hava T Siegelmann. Verifying properties of neural networks. In *International Work-Conference on Artificial Neural Networks*, pages 158–165. Springer, 2001.

[30] Daniel Selsam, Percy Liang, and David L Dill. Developing bug-free machine learning systems with formal mathematics. In *International Conference on Machine Learning*, pages 3047–3056, 2017.

[31] Sanjit A. Seshia. Compositional verification without compositional specification for learning-based systems. Technical Report UCB/EECS-2017-164, EECS Department, University of California, Berkeley, Nov 2017.

[32] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards Verified Artificial Intelligence. *ArXiv e-prints*, July 2016.

[33] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2013. arXiv:1312.6199.

[34] Brian J. Taylor and Marjorie A. Darrah. Rule extraction as a formal method for the verification and validation of neural networks. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, volume 5, pages 2915–2920. IEEE, 2005.

[35] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. 2018. arXiv:1801.10578.

[36] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. Deep lattice networks and partial monotonic functions. In *Advances in Neural Information Processing Systems*, 2017.