Logic of Differentiable Logics: Towards a Uniform Semantics of DL

Natalia Ślusarz[1], Ekaterina Komendantskaya[1], Matthew L. Daggitt[1], Robert Stewart[1], and Kathrin Stark[1]

Heriot-Watt University, Edinburgh, United Kingdom
nds1@hw.ac.uk

1 Introduction

Logics for reasoning with uncertainty and probability have long been known and used in programming: e.g. fuzzy logic [20], probabilistic logic [11] and variants of thereof in logic programming domain [3, 9, 10, 12]. Recently, rising awareness of the problems related to *machine learning verification* opened a novel application area for those ideas. *Differentiable Logics* (DLs) is a family of methods that applies key insights from fuzzy logic and probability theory to enhance this domain with property-driven learning [5].

As a motivating example, consider the problem of verification of neural networks. A neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ parametrised by a set of weights $\mathbf{w}$. A training dataset $\mathcal{X}$ is a set of pairs $(\mathbf{x}, \mathbf{y})$ consisting of an input $\mathbf{x} \in \mathbb{R}^n$ and the desired output $\mathbf{y} \in \mathbb{R}^m$. It is assumed that the outputs $\mathbf{y}$ are generated from $\mathbf{x}$ by some function $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}^m$ and that $\mathbf{x}$ is drawn from some probability distribution over $\mathbb{R}^n$. The goal of training is to use the dataset $\mathcal{X}$ to find weights $\mathbf{w}$ such that f approximates $\mathcal{H}$ well over input regions with high probability density. The standard approach is to use a *loss function* $\mathcal{L}$, that given a pair $(\mathbf{x}, \mathbf{y})$ calculates how much $f(\mathbf{x})$ differs from the desired output $\mathbf{y}$. Gradients of $\mathcal{L}$ with respect to the network's weights can then be used to update the weights during training.

However in addition to the dataset $\mathcal{X}$, in certain problem domains we may have additional information about $\mathcal{H}$ in the form of a mathematical property ϕ that we know $\mathcal{H}$ must satisfy. A common example of such a property is that $\mathcal{H}$ should be *robust*, i.e. small perturbations to the input only result in small perturbations to the output. For example, in image classification tasks changing a single pixel should not significantly alter what the image is classified as [2, 14].

Definition 1.1 (ϵ-δ-robustness)**.** Given constants $\epsilon, \delta \in \mathbb{R}$, a function f is *ϵ-δ-robust* around a point $\hat{\mathbf{x}} \in \mathbb{R}^n$ if:

$$\forall \mathbf{x} \in \mathbb{R}^n : ||\mathbf{x} - \hat{\mathbf{x}}|| \leq \epsilon \implies ||f(\mathbf{x}) - f(\hat{\mathbf{x}})|| \leq \delta \tag{1}$$

The problem of verifying the robustness of neural networks has received significant attention from the verification community [13, 19], and it is known to be difficult both theoretically [6] and in practice [1]. However, even leaving aside the challenges of undecidability of non-linear real arithmetic [7], and scalability [19], the biggest obstacle to verification is that the majority of neural networks do not succeed in learning ϕ from the training dataset $\mathcal{X}$ [5, 18].

The concept of a *differentiable logic* (DL) was introduced to address this challenge by verification-motivated training. This idea is sometimes referred to as continuous verification [8, 2], referring to the loop between training and verification. The key idea in differentiable logic is to use ϕ to generate an alternative *logical loss function* $\mathcal{L}_\phi$, that calculates a penalty depending on how much f deviates from satisfying ϕ. When combined with the standard data-driven loss function $\mathcal{L}$, the network is trained to both fit the data and satisfy ϕ. A DL therefore has two components: a suitable language for expressing the properties and an interpretation function that can translate expressions in the language into a suitable loss function.

Although the idea sounds simple, developing good principles of DL design has been surprisingly challenging. The machine-learning community has proposed several DLs such as DL2 for supervised learning [4], and a signal temporal logic based-DL for reinforcement learning (STL) [17]. However, both approaches had shortcomings from the perspective of formal logic: the former fell short of introducing quantifiers as part of the language, and the latter only covered propositional fragment.

Solutions were offered from the perspective of formal logic. In [15, 16] it was shown that one can use various propositional fuzzy logics to create loss functions. However, these fuzzy DLs did not stretch to cover the features of the DLs that came from machine learning community, which need not just quantification over infinite domains, but also a formalism to express properties concerning vectors and functions over vectors.

The first problem is thus: *There does not exist a DL that formally covers a sufficient fragment of first-order logic to express key properties in machine learning verification.*

The second problem has to do with formalisation of different DLs. *In many of the existing DL approaches syntax, semantics and pragmatics are not well-separated, which inhibits their formal analysis.* We have already given an example of DL2 treating quantifiers empirically outside of the language. But the problem runs deeper. To illustrate, let us take a fragment of syntax on which all DLs are supposed to agree. It will give us a toy propositional language

$$a := p \mid a \wedge a$$

They assume that each propositional variable p is interpreted in a domain $\mathcal{D} \subseteq \mathbb{R}$. The domains vary vastly across the DLs (from fuzzy set $[0,1]$ to (∞, ∞) in STL) and the choice of a domain can have important ramifications for both the syntax and the semantics. For example DL2's domain $[0, \infty]$ severely restricts the translation of negation compared to other DLs. In STL [17] the authors redefine the syntax for conjunction itself thus obtaining a different language

$$a := p | \bigwedge_M (a_1, \ldots, a_M)$$

where $\bigwedge_M$ denotes a (non-associative!) conjunction for M elements.

As consequence of the above two problems, the third problem is *lack of unified, general syntax and semantics able to express multiple different DLs and modular on the choice of DL, that would make it possible to choose one best suited for concrete task or design new DLs in an easy way.*

In this paper, we propose a solution to all of these problems. The solution comes in a form of a meta-DL, which we call *a logic of differentiable logics (LDL).* On the syntactic side, it is a typed first-order language with negation and universal and existential quantification that can express properties of functions and vectors.

On the semantic side, interpretation is defined to be parametric on the choice of the interpretation domain $\mathcal{D}$ or a particular choice of logical connectives. This parametric nature of interpretation simplifies both the theoretical study and implementations that compare different DLs. Moreover, the language has an implicit formal treatment of neural networks via a special kind of context – a solution that we found necessary in achieving a sufficient level of generality in the semantics. For the first time the semantics formally introduces the notion of a probability distribution that corresponds to the data from which the data is assumed to be drawn. We demonstrate the power of this approach by using LDL to prove soundness of various DLs and systematically compare their geometric properties in Section.

References

[1] S. Bak, C. Liu, and T. Johnson. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results, 2021. Technical Report. http://arxiv.org/abs/2109.00498.

[2] Marco Casadio, Ekaterina Komendantskaya, Matthew L. Daggitt, Wen Kokke, Guy Katz, Guy Amir, and Idan Refaeli. Neural network robustness as a verification property: A principled case study. In *Computer Aided Verification (CAV 2022)*, Lecture Notes in Computer Science. Springer, 2022.

[3] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100:5–47, 2015.

[4] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1931–1941. PMLR, 09–15 Jun 2019.

[5] Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 5478–5485. ijcai.org, 2022.

[6] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *International Conference on Computer Aided Verification*, 2017.

[7] Wen Kokke, Ekaterina Komendantskaya, Daniel Kienitz, Bob Atkey, and David Aspinall. Neural networks, secure by construction: An exploration of refinement types. In *APLAS*, 2020.

[8] Ekaterina Komendantskaya, Wen Kokke, and Daniel Kienitz. Continuous verification of machine learning: a declarative programming approach. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming*, pages 1:1–1:3. ACM, 2020.

[9] Thomas Lukasiewicz. Probabilistic logic programming. In *European Conference on Artificial Intelligence*, pages 388–392, 1998.

[10] Raymond Ng and Venkatramanan Siva Subrahmanian. Probabilistic logic programming. *Information and computation*, 101(2):150–201, 1992.

[11] Nils J Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986.

[12] Fabrizio Riguzzi. *Foundations of probabilistic logic programming: Languages, semantics, inference and learning.* CRC Press, 2022.

[13] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3:41:1–41:30, 2019.

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[15] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy implications. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 893–903, 2020.

[16] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022. ISSN 0004-3702.

[17] Peter Varnai and Dimos V. Dimarogonas. On robustness metrics for learning stl tasks. In *2020 American Control Conference (ACC)*, pages 5394–5399, 2020. doi: 10.23919/ACC45564.2020.9147692.

[18] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Advances in neural information processing systems*, 31, 2018.

[19] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems*, pages 29909–29921, 2021.

[20] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.