Placeholder title: Formal methods in constraint-based learning

Jane Open Access
Dummy University Computing Laboratory, [optional: Address], Country
My second affiliation, Country

Abstract

Abstract.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding *Jane Open Access*: (Optional) author-specific funding acknowledgements

1 Introduction

Using Interactive Theorem Provers (ITPs) in combination with machine learning (ML) is a very new approach dating only a few years back CITE. The integration of formal methods with machine learning poses many challenges, stemming from the inherent differences between the two domains, from the black-box nature of many ML models, through scalability issues especially in the case of neural networks (NNs) to the lack of well-defined specifications required in formal methods.

However in the recent years there has been a growing interest in the intersection of these two fields. One of the first attempts to use Coq in conjunction with machine learning is MLCert [2], a system for mechanized proofs of the generalisation bounds of an ML model in Coq, providing bounds on expected error. MLCert is compatible with deep learning frameworks such as TensorFlow but the classifier does need to be encoded by hand. Coq can be easier used for reasoning about well defined smaller machine learning algorithms, for example for proving convergence of a one-layer perceptron in specific conditions [11].

The existing work is not limited to Coq. It has been demonstrated that a neural network can be encoded in a theorem prover, in this case Isabelle/HOL, in order to prove its safety [3]. Vehicle, a tool for enforcing specifications on neural networks, is also capable of exporting a verified specification to Agda to enable reasoning about the network when considering safety of a larger system[5].

The existing attempts at combining ITPs and machine learning have encountered some of the same problems that originate form the inherent nature of machine learning as a field and are challenging for ITPs. Among the most theoretical ones are the necessity to deal with real domain as well as vectors in order to reason about data as well as need for support for linear algebra and certain areas of calculus. This mathematical complexity is inherent especially to neural networks but is widely known to pose a challenge to formalisation tools. Lastly there is also an issue of scale when formalising large models.

There are many aspects of neural networks that can be formalised.

Natalia: Need to find a nice transition in this paragraph

In particular one we investigate in this work is differentiable logics.

$$
\begin{array}{ll}
\langle expr \rangle \ni e ::= x \mid f \mid r \in \mathbb{R} \mid i \in \mathbb{N} \mid b \in \mathbb{B} & \langle type \rangle \ni \tau ::= s \to \tau \mid \mathrm{s} \\
\quad \mid \; e\, e \mid \mathrm{lam}\ (x : \tau)\ .\ e \mid \mathrm{let}\ (x : \tau) = e\ \mathrm{in}\ e & \\
\quad \mid \; \wedge \mid \vee \mid \neg \mid \Rightarrow \mid + \mid \text{-} \mid \times & \langle simple\ type \rangle \ni s ::= \\
\quad \mid \; \neq \mid \leq \mid \geq \mid < \mid > \mid == & \quad \mid \; \mathrm{Bool} \\
\quad \mid \; [e, ..., e] \mid \, ! & \quad \mid \; \mathrm{Real} \\
 & \quad \mid \; \mathrm{Vec}\ n \mid \mathrm{Index}\ n \qquad \text{for } n \in \mathbb{N}
\end{array}
$$

Figure 1 Specification Language of LDL: expressions and types.

Differentiable logics belong to a method of property-based training of neural networks. The idea behind differentiable logic is to use a property that is desirable in a neural network and use it to generate a custom property based loss function. A standard *loss function* $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ given a pair $(\mathbf{x}, \mathbf{y})$ of input and desired output, a neural network f, calculates how much $f(\mathbf{x})$ differs from the desired output $\mathbf{y}$ [15].

A property based loss will instead aim to penalise the network for deviating from said property. Training a neural network with a combination of a property based loss function and a standard data-driven one will result in a network trained to fit both the data and the property.

A DL itself consists of syntax - the logical language used to express properties - and semantics - the "translation" from the syntax to a loss function. A general typed syntax and semantics for expressing different popular DLs that we use as a reference point is Logic of Differentiable Logics (LDL) CITE. The syntax of LDL is a typed first order logic language with addition of lambdas however in this paper we will consider its propositional fragment which can be seen in Figure 1. Importantly the type system includes the existance of vectors, vector indicies and real numbers.

The syntax of LDL is then interpreted into a loss function. The semantics are parametric based on the choice of interpretation for comparisons and logical connectives which differ between various differentiable logics.

> Natalia: Let me know if I should expand on LDL here or in the background section instead. Running example might be good to illustrate it since I think this explanation alone is likely confusing but I don't know what sort of running example would be compelling for ITP audience

The problem that we address is that DL formalisation papers are often unclear or wrong in claims about certain properties of DLs they claim CITE MYSELF, STL. The approach we use is formalisation of the the meta-language LDL in the Coq proof assistant. While as mentioned the mathematical context needed to reason about DLs is challenging for theorem provers, we believe that there exist current state of the art formalisation tools which are expressive enough and we use paper as demonstration that current state of the art already allows for application of formal approaches to well-defined problems in machine learning.

Hence we have chosen to use Mathematical Components (MathComp) [13] in Coq CITE which stands as a testament to the expressive power of formal proof systems in addressing advanced mathematical concepts. At its core, MathComp leverages the Calculus of Inductive Constructions (CIC), the underlying type theory of Coq, to encode mathematical abstractions in a way that allows bridging the gap between mathematics and the formalization of machine learning concepts.

Our first contribution is the formalisation of several best known DLs using Coq and MathComp. Our formalisation of the LDL framework is designed to be generic and high-level so that it allows for easy extension to other DLs. EXPAND

Secondly, our implementation of the syntax is non-trivial. It utilises dependent types

interestingly. We have discovered some unintended behaviours in Coq implementation of dependent types which caused standard tactics to fail in proofs.

> Natalia: Should I explain the problem with R as context vs variable here, or in some of the later chapters?

Another of our contributions is demonstrating the capabilities of MathComp, including latest additions to the library CITE, when applied to machine learning. This is the first successful attempt at formalisation of the LDL calculus which includes dealing with concepts known to be problematic in formalisation and yet prevalent in machine learning such as working in the real domain and dealing with vectors.

Lastly we provide formal proofs of theorems that were not rigorously proven in ML literature CITE MYSELF ALSO STL. This refers to both theorems that were stated incorrectly, not listing all of the necessary assumptions as well as incorrect claims about certain DLs holding specific desirable mathematical properties. We would like this work to inform design choices in DLs and help fix preexisting issues. We believe that using formal methods approaches can be beneficial not only in verification, but also in helping to deepen the understanding of machine learning as well as enforcing a more formal approach to more mathematical areas of machine learning.

> Natalia: Should I go into detail here what the actual issue was or should that be left for later? Currently plan to discuss the STL paper shadow-lifting/assoc/idempotence example in later chapter as per plan

2 Related work

To the best of our knowledge there is very little work that is closely related to our approach and involves applying formal methods and ITPs to the training approaches itself instead of an already trained machine learning model or its properties. Furthermore majority of approaches involving formal methods and machine learning apply to already trained machine learning models, and not approaches such as constraint-based training itself and thus are not very closely related to this work. This area of research is much less explored as of yet, primarily due the difficulty of expressing the mathematical theory underlying neural networks.

Relevant work dealing with already trained models does involve encoding trained networks in ITPs, such as Isabelle/HOL [3]. This Isabelle prototype supports importing feedforward neural networks from TensorFlow, eliminating the need to embed each network by hand and it is the first formalisation of neural networks in an interactive theorem prover. Similiarly another recent work to set up other interactive theorem provers for verification of neural network by formalising piecewise affine activation functions in Coq [1].

Other works focus on different properties important to the machine learning community such as generalisation, providing formal guarantees of the degree to which the train network will generalise to new data [2]. Since scale and lack of precise mathemtaical definitions are some of the main issues for theorem provers, reasoning about properties of certain smaller but well defined machine learning algorithms lend themselves better to formalisation. We can take as an example proving convergance of a as single-layer perceptron [11].

While they do not directly formalise networks, there has also been some research related

to algorithms, instead of trained models such as formalisation of an embedding of linear temporal logic and an associated evaluation function in Isabelle to prove its soundness - and furthermore are able to produce OCaml versions of the aforementioned functions that can be integrated with Python and PyTorch eliminating many risks associated with ad-hoc implementations[4]. Another related example are formalisation of other types of algorithms [6] or formalisation of neural networks using Haskell [16]. They, while not directly related to the subject matter, utilise similar approaches and also have to contend with similar issues related to mathematical complexity.

Lastly, a larger but less directly related area area where more broadly understood formal methods and machine learning combine is verification. This field has been rapidly gaining traction in the last few years and there are many different approaches present in neural network verification[14, 10, 7] including abstract interpretation, SMTs (Satisfiability Modulo Theory) [9] solvers or MILP (Mixed Integer Linear Programming). There has also been similar work done for other machine learning approaches than neural networks such as for support vector machines (SVMs) [12] or Decision Tree Ensembles [8] though we do not focus on such models in our work.

3 Background - notes

Notes on what probably needs to make its way into background (non-exhaustive)

- A good explanation of DL and similar methods in machine learning. Including syntax, semantics in tables. Also explanation of loss functions and their role in training.
- Possibly properties here, not later.

4 Temporary Name: Formalisation - notes

General notes to self on formalisation.

Different functions, lemmas, and things that perhaps stand out.

4.1 Syntax and semantics

Implementation of the syntax: *expr* with it's custom dependant type *simple_type*. Since the type system prevents non-termination of the lambda expressions they, together with let statements, have been omitted in the Coq formalisation (argumentation why it's okay goes here).

Implementation of the semantics:

- *type_translation* semantics of types
- a word on translation of indices and vectors using tuples
- mention Real library from MathComp to handle reals
- *translation* - a semantics for all fuzzy DLs. Here there needs to be explanation maybe of how DL2 and STL are separate from the fuzzy logics as they are different enough to warrant a separate definition and lemmas, not enough overlap translation itself is dependently typed, handles 4 different fuzzy logics through use of global variable (for now it's global variable, up for discussion)

No quantifiers in the semantics. That does not impact the formalisation of most properties aside from soundness as the other properties do not concern quantifiers.

4.2 Soundness (and associated lemmas)

- *expr_ind′* custom induction principle was needed due to use of sequences. Explain sequences in MathComp (hidden tuples)? Explain that this is needed due to possibility of non-associative binary operations in the language.
- [*nameofDL*]_*translate*_*Bool*_*T*_01 - lemmas that prove that the fuzzy DL semantics fall within the range [0,1]

Inversion lemmas also mentioned here possibly.

4.3 Logical properties

On associativity and commutativity for the DLs that have them (and that associativity is much more tricky then commutativity to prove). Not entirely sure if antyhing more interesting is here specifically so likely not its own subsection.

4.4 Shadow-lifting

Using limit definition of partial differentiation.

It does work out of the box using MathComp)in that limits exist but does require more lemmas as it is a new library. Just a lot more detail on this proof when finished.

5 Other notes

Complexity increase depending on DL is. While "complexity" is not easy to measure for a DL on paper it is obvious in its impact in formalisation. Associativity of conjunction for Łukawsiewicz, Godel and product all have around 10 lines (get exact after cleanup) - but Yager takes 38 (again, correct after cleanup). If I prove STL mention that (if case split in analogous ways to other proofs we get 38 cases)

A lot of difficulty is added due to using nary operations for the sake of generality (bigops, etc.). Will need to have good reasoning for that (primarily htat it's following the very general idea behind LDL).

6 Placeholder title: stl example

References

1. Andrei Aleksandrov and Kim Völlinger. Formalizing piecewise affine activation functions of neural networks in coq. In *NASA Formal Methods Symposium*, pages 62–78. Springer, 2023.
2. Alexander Bagnall and Gordon Stewart. Certifying the true error: Machine learning in coq with verified generalization guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2662–2669, 2019.
3. Achim D Brucker and Amy Stell. Verifying feedforward neural networks for classification in isabelle/hol. In *International Symposium on Formal Methods*, pages 427–444. Springer, 2023.
4. Mark Chevallier, Matthew Whyte, and Jacques D Fleuriot. Constrained training of neural networks via theorem proving. *arXiv preprint arXiv:2207.03880*, 2022.
5. Matthew L Daggitt, Robert Atkey, Wen Kokke, Ekaterina Komendantskaya, and Luca Arnaboldi. Compiling higher-order specifications to smt solvers: How to deal with rejection constructively. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 102–120, 2023.

196 **6** Ieva Daukantas, Alessandro Bruni, and Carsten Schürmann. Trimming data sets: a verified
197 algorithm for robust mean estimation. In *23rd International Symposium on Principles and*
198 *Practice of Declarative Programming*, pages 1–9, 2021.
199 **7** Steve Easterbrook and John Callahan. Formal methods for verification and validation of
200 partial specifications: A case study. *Journal of Systems and Software*, 40(3):199–210, 1998.
201 **8** Gil Einziger, Maayan Goldstein, Yaniv Sa'ar, and Itai Segall. Verifying robustness of gradient
202 boosted models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33,
203 pages 2446–2453, 2019.
204 **9** Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim,
205 Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework
206 for verification and analysis of deep neural networks. In *Computer Aided Verification: 31st*
207 *International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings,*
208 *Part I 31*, pages 443–452. Springer, 2019.
209 **10** Moez Krichen, Alaeddine Mihoub, Mohammed Y. Alzahrani, Wilfried Yves Hamilton Adoni,
210 and Tarik Nahhal. Are formal methods applicable to machine learning and artificial intelli-
211 gence? In *2022 2nd International Conference of Smart Systems and Emerging Technologies*
212 *(SMARTTECH)*, pages 48–53, 2022. doi:10.1109/SMARTTECH54121.2022.00025.
213 **11** Charlie Murphy, Patrick Gray, and Gordon Stewart. Verified perceptron convergence theorem.
214 In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and*
215 *Programming Languages*, pages 43–50, 2017.
216 **12** Francesco Ranzato and Marco Zanella. Robustness verification of support vector machines.
217 In *Static Analysis: 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11,*
218 *2019, Proceedings 26*, pages 271–295. Springer, 2019.
219 **13** Mathematical Components Team. Mathematical components library, 2007. Last stable version:
220 2.1 (2023). URL: https://github.com/math-comp/math-comp.
221 **14** Caterina Urban and Antoine Miné. A review of formal methods applied to machine learning.
222 *arXiv preprint arXiv:2104.02466*, 2021.
223 **15** Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in
224 machine learning. *Annals of Data Science*, pages 1–26, 2020.
225 **16** Ningning Xie. Haskell for choice-based learning (keynote). In *Proceedings of the 16th ACM*
226 *SIGPLAN International Haskell Symposium*, pages 1–1, 2023.