Proceedings of Machine Learning Research 176:319–324, 2022 NeurIPS 2021 Competition and Demonstration Track

Pylon: A PyTorch Framework for Learning with Constraints

Kareem Ahmed AHMEDK@CS.UCLA.EDU
University of California, Los Angeles

Tao Li TLI@CS.UTAH.EDU
University of Utah

Thy Ton THYNT@UCI.EDU
University of California, Irvine

Quan Guo GUOQUAN@SCU.EDU.CN
Sichuan University

Kai-Wei Chang KWCHANG@CS.UCLA.EDU
University of California, Los Angeles

Parisa Kordjamshidi KORDJAMS@MSU.EDU
Michigan State University

Vivek Srikumar SVIVEK@CS.UTAH.EDU
University of Utah

Guy Van den Broeck GUYVDB@CS.UCLA.EDU
University of California, Los Angeles

Sameer Singh SAMEER@UCI.EDU
University of California, Irvine

Editors: Douwe Kiela, Marco Ciccone, Barbara Caputo

Abstract

Deep learning excels at learning low-level task information from large amounts of data, but struggles with learning high-level domain knowledge, which can often be directly and succinctly expressed. In this work, we introduce PYLON, a neuro-symbolic training framework that builds on PyTorch to augment procedurally trained neural networks with declaratively specified knowledge. PYLON allows users to programmatically specify *constraints* as PyTorch functions, and compiles them into a differentiable loss, thus training predictive models that fit the data *whilst* satisfying the specified constraints. PYLON includes both exact as well as approximate compilers to efficiently compute the loss, employing fuzzy logic, sampling methods, and circuits, ensuring scalability even to complex models and constraints. A guiding principle in designing PYLON has been the ease with which any existing deep learning codebase can be extended to learn from constraints using only a few lines: a function expressing the constraint and a single line of code to compile it into a loss. We include case studies from natural language processing, computer vision, logical games, and knowledge graphs, that can be interactively trained, and highlights PYLON's usage.

Keywords: Neuro-symbolic Learning, Learning with Constraints, Deep Learning

Introduction

Deep learning models are able to learn even the most complex of tasks, provided enough data is available. However, they often struggle with learning high-level domain knowledge that can often be much more succinctly expressed declaratively, such as using programmatic constraints. Unfortunately, existing frameworks are not able to learn from such declarative knowledge, and instead attempt to learn it from available data, leading to overfitting to spurious patterns, learning functions that are unfaithful to rules of the underlying task.

Neuro-symbolic reasoning methods aim to straddle the line between deep learning and symbolic reasoning, combining high-level procedural knowledge with data, during learning. They aim to learn functions that fit the data while remaining faithful to the rules of the underlying domain, which empirically translates into performance improvements and more efficient learning. These systems are not without their challenges, however. Most frameworks make use of custom languages (Rajaby Faghihi et al.; Guo et al., 2020; Manhaeve et al., 2018; Stewart and Ermon, 2017) or logic (Bach et al., 2017; Diligenti et al., 2017; Fischer et al., 2019; Hu et al., 2016; Li and Srikumar, 2019; Nandwani et al., 2019; Rocktäschel et al., 2015; Xu et al., 2018; Zhang et al., 2016) to express such knowledge, making it unnatural, unwieldy, or even impossible to express many forms of knowledge. Furthermore, they often require porting to their own ecosystem, making them arduous to integrate with existing code. Finally, each such method presents with a unique set of trade-offs, and is effective on a limited set of domains and constraints, often unbeknownst to the user.

We introduce Pylon[1], a package built on top of PyTorch that offers practitioners the ability to seamlessly integrate declarative knowledge into deep learning models. The user expresses the knowledge as a Python function that defines the constraint in terms of PyTorch tensors. Picking an appropriate compiler, Pylon compiles the function into an efficient, differentiable loss that is compatible with PyTorch trainers, providing a unifying interface to existing neural-symbolic methods that integrate declarative knowledge into learning.

Pylon Overview

Example Consider the code snippet in Figure 1, where we consider the task of entity-relation extraction. That is, given a sentence `x`, the `model` on line 14 classifies each word into a corresponding entity (e.g. `person`, `organization`), and for every pair of entities whether they are related, and if so, the type of relation that holds between them (e.g. `works for`).

```
1  # Only a person can live in a location
2  def check_livesin_subj(entity, relation):
3    # If a word is subject of livesIn, it should be PER
4    return all(entity[relation==LIVESIN_SUBJ] == PER)
5
6  livesin_loss = constraint_loss(check_livesin_subj)
7
8  # There should be more non-people tokens than people
9  numppl_loss = constraint_loss(
10   lambda entity: sum(entity!=PER) > sum(entity==PER))
11
12 for i in range(train_iters):
13   ...
14   entity_logits = entity_model(x)
15   relation_logits = relation_model(x)
16   loss = livesin_loss(entity_logits, relation_logits)
17   loss += CE(relation_logits, relation_labels)
18   loss += numppl_loss(entity_logits)
```

Figure 1: Enforcing a constraint using Pylon

We wish to enforce two constraints that capture our domain knowledge on the learned `model`: 1) the subject of a `lives in` relation is always a `person`, and 2) that the majority of predicted entities are not `person`. The above constraints are

1. Pylon website is available at https://pylon-lib.github.io/

expressed as the PyTorch function `check_livesin_subj` defined in Figure 1 on lines 2-4 and the lambda function on line 10, respectively.

The challenge then is, how to integrate these discrete, Boolean functions with differentiable learning. We will show how this can be achieved by interpreting the model outputs as inducing a distribution over the output space, and reducing our problem to one of probabilistic reasoning: we wish to find the set of parameters that maximize the probability of satisfying the user-defined constraints under the network's probability distribution.

A Probability Distribution over Structured Outputs Let θ be the parameters of the neural network model defined over a set of variables $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$, where each Y_i denotes a target. Let p be a vector of probabilities for the variables $\mathbf{Y}$, where p_i denotes the predicted probability of variable Y_i and corresponds to a single output of the network. The network's outputs *induce* a distribution $p_\theta(\cdot)$ over all possible instantiations $\mathbf{y}$ of $\mathbf{Y}$

$$p_\theta(\mathbf{y}|x) = \prod_{i:\mathbf{y}\models Y_i} \mathsf{p}_i \prod_{i:\mathbf{y}\models \neg Y_i} (1 - \mathsf{p}_i) \tag{1}$$

where $\mathbf{y} \models Y_i$ and $\mathbf{y} \models \neg Y_i$ denote that Y_i is true or false in the instantiation $\mathbf{y}$, respectively. Although we use boolean valued variables for notational simplicity, the ideas directly extend to other discrete valued variables as well.

Training objective Having defined a distribution over all possible outputs, we now consider the problem of learning with constraints through a probabilistic lens: the problem of integrating our declaratively-defined functions into the learning process reduces to optimizing for the set of network parameters such that the probability allocated by the network to satisfying the constraints is maximized. Formally, we would like to minimize the following:

$$\arg\min_\theta \mathcal{L}(\theta|\mathcal{C}, x) = \arg\min_\theta -\log \mathbb{E}_{\mathbf{y}\sim p_\theta(\cdot|x)} \big[\mathbf{1}\{\mathcal{C}(\mathbf{y})\}\big] \tag{2}$$

where, for a given constraint $\mathcal{C}$, we penalize the network with a loss that is proportional to the extent to which the network's beliefs violate the constraint, as measured by the probability mass allocated by the network to all instantiations violating the constraint $\mathcal{C}$.

Calculating the above expectation naively requires enumerating all instantiations $\mathbf{y}$ in a *brute force* manner, of which there are exponentially many, and is feasible only for the simplest of constraints. For example, for a model defined over the edges in a $n \times n$ grid (i.e. $2n^2 - 2n$ boolean variables), an instantiation is an assignment to each of the variables, of which there are 2^{2n^2-2n} many possibilities.

Constraint functions We encode the aforementioned declarative knowledge by means of *constraint functions*. A constraint function is a Python function that accepts any number of tensor arguments, each of shape (`batch_size`, ...) and returns a Boolean tensor of shape (`batch_size`,). Each argument corresponds to a (batched) *decoding* from a model. A decoding is an assignment to all variables of a model, each variable sampled with a probability corresponding to its likelihood under the model's posterior. For example, in our entity-relation extraction example, a decoding of `relation_logits` (or `entity_logits`) constitutes a relation (or entity, resp.) assigned to each word in the sentence.

A constraint function defines a *predicate* $\mathcal{C}$ on the decodings of any number of models, and returns whether or not the given decodings satisfy the constraint. For instance, lines

2-4 define a constraint function over the decodings of the entity and relations classifiers that captures that the subject of a `lives in` relation is always a `person`. Line 10 defines a lambda constraint function over the decoding of the entity classifier, and encodes that the majority of entities are not person. While the first constraint can be easily expressed in logic, the same does not hold true for the second constraint: we would need to conjoin all decodings satisfying the constraint, which would scale exponentially with the length of the sentence — unless we resort to introducing auxiliary variables. Using Python/PyTorch, we can capture the constraint succinctly.

Exploiting Structure of Constraint Definition Although the user can use all of PyTorch/Python syntax to write the constraint, we parse the constraint function to see if it expresses known structures, such as logic. When the constraints exhibit structural properties that allow us to reuse intermediate computations, we can sidestep the intractability of Eq (2) by compiling them into *logical circuits* (Xu et al., 2018). This does not, in general, escape the complexity of Eq (2) as the circuit grows exponentially in the constraint size. In these cases, we can utilize approximations based on fuzzy logic, computing differentiable probabilities of logical statements without grounding them, such as using product (Rocktäschel et al., 2015), or Łukasiewicz (Bach et al., 2017; Kimmig et al., 2012) *T-norms*.

Black-box Optimization Alternatively, we can also approximate the loss in Eq (2) by *sampling* decodings from the model posterior. We can use the REINFORCE gradient estimator (Glynn, 1990; Williams, 1992) to rewrite the gradient of the expectation in Eq (2) as the expectation of the gradient, which can be estimated using Monte Carlo sampling

$$\nabla_\theta \mathbb{E}_{\mathbf{y}\sim p_\theta(\cdot|x)}\big[\mathbf{1}\{\mathcal{C}(\mathbf{y})\}\big] = \mathbb{E}_{\mathbf{y}\sim p_\theta(\cdot|x)}\big[\nabla_\theta \mathbf{1}\{\mathcal{C}(\mathbf{y})\}\log p_\theta(\mathbf{y}|x)\big] \tag{3}$$

This not only enables us to estimate the probability of otherwise-intractable constraints but also enables greater flexibility in defining our constraint functions: we can issue calls to non-differentiable resources and continue to yield a *differentiable* loss, hence *black-box*.

PYLON uses implementations of these approaches that are directly compatible with PyTorch, as seen in lines 16 and 17, including ones that utilize the structure in the user-defined code for efficiency (*T-norm* and *circuit*-based losses) and ones that work for any implementation (*brute-force* and *sampling*), and is easily extensible to other techniques.

Case Studies

We include four case studies that vary in their domain, and exhibit the versatility of PYLON, spanning vision, NLP and logical games (e.g. Li et al., 2019; Punyakanok et al., 2008):

- **MNIST Addition**: Presented with two MNIST digits, we require the model's predictions add to their summation. Model learns to predict single digits at test time.
- **NLI Transitivity**: The model is presented with sentence triples and predicts how each pair is related. The model's predictions are constrained to satisfy transitivity.
- **SRL Unique Role**: For each predicate in the semantic role labeling task, the model's predictions are constrained such that each core argument span appears at most once.
- **Sudoku**: Given a Sudoku, the model is trained to predict the missing entries in the puzzle such that the elements in each individual row, column and square are *unique*.

Acknowledgements

We would like to thank Sebastian Riedel and Sanjay Subramanian for important suggestions and feedback for the project. This work is supported in part by NSF grants #IIS-2046873, #CCF-1837129, #IIS-1956441, #IIS-1943641, #CNS-1801446, #IIS-1822877, NSF Career award #2028626, Office of Naval Research grant #N00014-20-1-2005, a Sloan Fellowship, and a gift from Allen Institute for AI.

References

Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18(109): 1–67, 2017. URL http://jmlr.org/papers/v18/15-631.html.

Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 920–923, 2017. doi: 10.1109/ICMLA.2017.00-37.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1931–1941. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/fischer19a.html.

Peter W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Commun. ACM*, 33(10):75–84, October 1990. ISSN 0001-0782. doi: 10.1145/84537.84552.

Quan Guo, Hossein Rajaby Faghihi, Yue Zhang, Andrzej Uszok, and Parisa Kordjamshidi. Inference-masked loss for deep structured output learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2754–2761, 7 2020. doi: 10.24963/ijcai.2020/382. URL https://doi.org/10.24963/ijcai.2020/382.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2410–2420, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1228. URL https://aclanthology.org/P16-1228.

Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012.

Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1028. URL https://aclanthology.org/P19-1028.

Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. A logic-driven framework for consistency of neural models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. *A Primal-Dual Formulation for Deep Learning with Constraints*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287, 2008. doi: 10.1162/coli.2008.34.2.257. URL https://aclanthology.org/J08-2005.

Hossein Rajaby Faghihi, Quan Guo, Andrzej Uszok, Aliakbar Nafar, and Parisa Kordjamshidi. DomiKnowS: A library for integration of symbolic domain knowledge in deep learning. In *Empirical Methods in Natural Language Processing: System Demonstrations*, pages 231–241, Online and Punta Cana, Dominican Republic, November . Association for Computational Linguistics. URL https://aclanthology.org/2021.emnlp-demo.27.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1118. URL https://aclanthology.org/N15-1118.

Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. *AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. URL https://ojs.aaai.org/index.php/AAAI/article/view/10934.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/xu18h.html.

Xiao Zhang, Maria Leonor Pacheco, Chang Li, and Dan Goldwasser. Introducing DRAIL – a step towards declarative deep relational learning. In *Workshop on Structured Prediction for NLP*, pages 54–62, Austin, TX, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-5906. URL https://aclanthology.org/W16-5906.