PYLON: A PyTorch Framework for Learning with Constraints

Kareem Ahmed,[1] **Tao Li,**[2] **Thy Ton,**[3] **Quan Guo,**[4] **Kai-Wei Chang,**[1]
Parisa Kordjamshidi,[5] **Vivek Srikumar,**[2] **Guy Van den Broeck,**[1] **Sameer Singh**[3]

[1] Computer Science Department, University of California, Los Angeles
[2] School of Computing, College of Engineering, University of Utah
[3] Department of Computer Science, University of California, Irvine
[4] Department of Artificial Intelligence, Sichuan University
[5] Department of Computer Science and Engineering, Michigan State University
{ahmedk, kwchang, guyvdb}@cs.ucla.edu, {tli, svivek}@cs.utah.edu,
{thynt, sameer}@uci.edu, guoquan@scu.edu.cn, kordjams@msu.edu

Abstract

Deep learning excels at learning task information from large amounts of data, but struggles with learning from declarative high-level knowledge that can be more succinctly expressed *directly*. In this work, we introduce PYLON, a neuro-symbolic training framework that builds on PyTorch to augment procedurally trained models with declaratively specified knowledge. PYLON lets users programmatically specify *constraints* as Python 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. Crucially, a guiding principle in designing PYLON is the ease with which any existing deep learning codebase can be extended to learn from constraints in a few lines of code: a function that expresses the constraint, and a single line to compile it into a loss. Our demo comprises of models in NLP, computer vision, logical games, and knowledge graphs that can be interactively trained using constraints as supervision.

Listing 1: Enforcing a constraint using PYLON

```python
# Only a person can live in a location
def check_livesin_subj(entity, relation):
  # If a word is subject of livesIn, it should be PER
  return all(entity[relation==LIVESIN_SUBJ] == PER)

livesin_loss = constraint_loss(check_livesin_subj)

# There should be more non-people tokens than people
numppl_loss = constraint_loss(
  lambda entity: sum(entity!=PER) > sum(entity==PER))

for i in range(train_iters):
  ...
  entity_logits = entity_model(x)
  relation_logits = relation_model(x)
  loss = livesin_loss(entity_logits, relation_logits)
  loss += CE(relation_logits, relation_labels)
  loss += numppl_loss(entity_logits)
```

Introduction

Deep learning models, by virtue of being universal function approximators, are able to learn even the most complex of tasks with enough available data. However, some high-level domain knowledge can often be much more succinctly described *directly* in a declarative manner, such as programmatic constraints, which existing learning frameworks are not able to learn from. Instead, deep learning models attempt to extract the same knowledge from data available to them, leading to overfitting the spurious patterns, learning functions that are unfaithful to rules of the underlying domain.

Neuro-symbolic reasoning systems aim to straddle the line between deep learning and symbolic reasoning, combining high-level declarative knowledge with data during learning. They aim to learn functions that fit the data while remaining faithful to the rules of the underlying domain. Empirically, this translates into performance improvements using less labeled data. These systems are not without their challenges, however. Most such systems use custom syntax (Faghihi et al. 2021; Guo et al. 2020; Manhaeve et al. 2018) or logic (Bach et al. 2017; Diligenti, Roychowdhury, and Gori 2017; Fischer et al. 2019; Hu et al. 2016; Li and Srikumar 2019; Nandwani et al. 2019; Rocktäschel, Singh, and Riedel 2015; Xu et al. 2018; Zhang et al. 2016) to express the constraint, making it unnatural, unwieldy or even impossible to express many forms of knowledge. Other approaches (Ratner et al. 2017) exploit knowledge differently, using it as labeling functions, providing a form of weak supervision on the unlabeled data. Also related are probabilistic programming languages (Bingham et al. 2018) which incorporate *stochastic functions* into deterministic Pytorch training code to obtain trained models with uncertainty estimates. Such systems often require porting of existing code bases to fit within the design of the target framework, making them arduous to integrate into preexisting code. Lastly,

different approaches to integrating symbolic knowledge and neural models have their own specific strengths and weaknesses, and are thus effective on a limited set of domains and constraints, which is often not clear 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 directly as a Python predicate function that defines the constraint on tensor variables (such as model output). PYLON compiles this user-defined function to efficiently compute a differentiable loss compatible with PyTorch trainers, providing a common interface to existing neural-symbolic approaches that integrate declarative knowledge in the learning process. Thus, with a few lines of code (defining the constraint and adding the loss), the user is able to integrate declarative knowledge into their models, testing which of the existing approaches are most effective.

PYLON Overview

Example Consider the code snippet in listing 1. It describes how to enforce a user-defined constraint in an entity-relation extraction setting using PYLON. The `model` on line 14 accepts a sentence `x`, and for each word in the sentence, returns the likelihood of different relations, and entities, respectively. We wish to enforce two considerations on our learned representation: 1) that the subject of a `lives_in` relation should be a `person` entity, and 2) a statistical constraint that most entities in the sentence are not `person`.

Constraint functions We encode the aforementioned declarative knowledge (read: constraints) by means of *constraint functions*. A constraint function is a Python function that accepts any number of tensor arguments, each of shape $(\texttt{batch\_size}, ...)$ and returns a Boolean tensor of shape $(\texttt{batch\_size},)$. 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` (resp. `entity_logits`) constitutes a relation (resp. entity) assigned to each word in the sentence. On the other hand, for a classifier defined over $2n^2 - 2n$ Boolean variables – the edges in a $n \times n$ grid – and that predicts a path in the grid, a decoding constitutes an assignment to each of the $2n^2 - 2n$ variables, and there are 2^{2n^2-2n} such decodings.

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 which encodes our first constraint whereas line 10 defines a lambda constraint function over the decoding of the entity classifier, and encodes our second constraint. Note that 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 manage to capture the constraint succinctly in a single line of code.

Training objective Having defined our constraint function $\mathcal{C}$, our aim is to *compile* it into a differentiable loss function pushing the model towards satisfying the constraints by minimizing the probability that the constraint is violated.

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

Calculating the above naively requires enumerating all decodings $\mathbf{y}$ in a *brute force* manner, of which there are exponentially many, and is feasible only for simple constraints.

Exploiting Structure of Constraint Definition Even though the user is free to use all of PyTorch/Python to write the constraint, we parse the constraint code to see if it is expressing known structures, for example, first-order logic. When the constraints do exhibit structural properties that allow us to reuse intermediate computations, we can sidestep the intractability of eqn. 1 by compiling them into *logical circuits* (Xu et al. 2018). This does not, in general, escape the complexity of eqn. 1 as the compiled circuit can worst-case grow exponentially in the size of the constraint. In such a case, we can utilize approximations based on fuzzy logic, computing differentiable probabilities of logical statements without grounding them, such as using product *T-norm* (Rocktäschel, Singh, and Riedel 2015), or Łukasiewicz *T-norm* (Bach et al. 2017; Kimmig et al. 2012).

Black-box Optimization Alternatively, we can also approximate the loss in eqn. 1 by *sampling* decodings from the network's posterior. More precisely, we can use the REINFORCE gradient estimator (Glynn 1990; Williams 1992) to rewrite the gradient of the expectation in eqn 1 as the expectation of the gradient, which can be readily estimated using Monte Carlo sampling. 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 (e.g. external APIs, database queries, etc.) and continue to yield a *differentiable* loss function, hence the moniker *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

Our case studies span computer vision, NLP and logical games (e.g. Li et al. 2019; Punyakanok, Roth, and Yih 2008)

- **MNIST Addition**: Presented with two MNIST digits, we require the model's predictions add to their summation.
- **NLI Transitivity**: The model's predictions of connected triples of sentences are constrained to satisfy transitivity.
- **SRL Unique Role**: For each predicate in the SRL task, we require a core argument span to appear at most once.
- **Sudoku**: Given a Sudoku, we require that in each individual row, column and square, the elements are *unique*.

[1] PYLON website is available at https://pylon-lib.github.io/

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

Bach, S. H.; Broecheler, M.; Huang, B.; and Getoor, L. 2017. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *Journal of Machine Learning Research*, 18(109): 1–67.

Bingham, E.; Chen, J. P.; Jankowiak, M.; Obermeyer, F.; Pradhan, N.; Karaletsos, T.; Singh, R.; Szerlip, P.; Horsfall, P.; and Goodman, N. D. 2018. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*.

Diligenti, M.; Roychowdhury, S.; and Gori, M. 2017. Integrating Prior Knowledge into Deep Learning. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 920–923.

Faghihi, H. R.; Guo, Q.; Uszok, A.; Nafar, A.; Raisi, E.; and Kordjamshidi, P. 2021. DomiKnowS: A Library for Integration of Symbolic Domain Knowledge in Deep Learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 231–241. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Fischer, M.; Balunovic, M.; Drachsler-Cohen, D.; Gehr, T.; Zhang, C.; and Vechev, M. 2019. DL2: Training and Querying Neural Networks with Logic. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 1931–1941. PMLR.

Glynn, P. W. 1990. Likelihood Ratio Gradient Estimation for Stochastic Systems. *Commun. ACM*, 33(10): 75–84.

Guo, Q.; Rajaby Faghihi, H.; Zhang, Y.; Uszok, A.; and Kordjamshidi, P. 2020. Inference-Masked Loss for Deep Structured Output Learning. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 2754–2761. International Joint Conferences on Artificial Intelligence Organization. Main track.

Hu, Z.; Ma, X.; Liu, Z.; Hovy, E.; and Xing, E. 2016. Harnessing Deep Neural Networks with Logic Rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2410–2420. Berlin, Germany: Association for Computational Linguistics.

Kimmig, A.; Bach, S.; Broecheler, M.; Huang, B.; and Getoor, L. 2012. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*.

Li, T.; Gupta, V.; Mehta, M.; and Srikumar, V. 2019. A Logic-Driven Framework for Consistency of Neural Models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

Li, T.; and Srikumar, V. 2019. Augmenting Neural Networks with First-order Logic. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Nandwani, Y.; Pathak, A.; Mausam; and Singla, P. 2019. A Primal Dual Formulation For Deep Learning With Constraints. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Punyakanok, V.; Roth, D.; and Yih, W.-t. 2008. The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. *Computational Linguistics*, 34(2): 257–287.

Ratner, A.; Bach, S. H.; Ehrenberg, H.; Fries, J.; Wu, S.; and Ré, C. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proc. VLDB Endow.*, 11(3): 269–282.

Rocktäschel, T.; Singh, S.; and Riedel, S. 2015. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1119–1129. Denver, Colorado: Association for Computational Linguistics.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8(3–4): 229–256.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 5502–5511. PMLR.

Zhang, X.; Pacheco, M. L.; Li, C.; and Goldwasser, D. 2016. Introducing DRAIL – a Step Towards Declarative Deep Relational Learning. In *Proceedings of the Workshop on Structured Prediction for NLP*, 54–62. Austin, TX: Association for Computational Linguistics.