Synthesis of Differentiable Functional Programs for Lifelong Learning

Lazar Valkov [1] **Dipak Chaudhari** [2] **Akash Srivastava** [1] **Charles Sutton** [1] **Swarat Chaudhuri** [2]

1. Summary

Differentiable programming languages (Paszke et al., 2017; Bosnjak et al., 2017; Gaunt et al., 2016; Bunel et al., 2016) have recently emerged as a powerful approach to representing architectures with input-dependent structure, such as deep networks over trees (Socher et al., 2013; Allamanis et al., 2017) and graphs (Li et al., 2016; Kipf & Welling, 2017). Being able to induce differentiable programs purely from data can have tremendous value in many applications. Unfortunately, inferring the structure of differentiable programs is a fundamentally hard problem. While some recent approaches (Gaunt et al., 2017; 2016) partially solve this problem, a detailed hand-written template of the program is required for even the simplest tasks.

In this paper, we show that *functional programming abstractions* and *symbolic program synthesis* can be useful tools in overcoming this difficulty. The goal in program synthesis (Alur et al., 2013; Solar-Lezama, 2013; Feser et al., 2015) is to discover programs that accomplish a given task, out of all programs in a language. While combinatorially hard, the problem has seen substantial recent progress. Indeed, in a recent head-to-head comparison, symbolic program synthesis was reported to outperform gradient-based program induction (Gaunt et al., 2016).

Functional programming is especially appealing for synthesis of differentiable programs because functional combinators can compactly describe many common neural architectures (Olah, 2015). Symbolic synthesis is appealing for learning neural libraries because their strengths complement those of stochastic gradient descent (SGD): while SGD is remarkably effective for learning network parameters, each step of a symbolic search can explore large changes to the network structure.

Concretely, we present a *neurosymbolic* learning framework, called HOUDINI, that is the first symbolic program synthesis method for differentiable programs. In HOUDINI, a program synthesizer is used to search over networks described as compact, strongly typed functional programs, whose parameters are then tuned end-to-end using SGD. Programs in HOUDINI specify the architecture of the network, and can also facilitate learning transfer, by letting the synthesizer choose among previously trained modules.

We show that the HOUDINI approach is especially powerful in the setting of *lifelong learning* (Thrun & Mitchell, 1995). Specifically, we evaluate HOUDINI on a sequence of tasks that mix perception and procedural reasoning. Two challenges in lifelong learning are *catastrophic forgetting* and *negative transfer*. Our use of a neural library avoids both problems, as the library allows us to freeze and selectively re-use portions of networks that have been successful. Our results indicate that HOUDINI leads to more significant transfer than several natural baselines.

2. The HOUDINI Framework

HOUDINI has two components. The first is a typed functional language of differentiable programs. The second is a learning procedure split into a symbolic module and a neural module.

The language. Our language has three key features:

- The ubiquitous use of *function composition* to glue together different networks. Specifically, HOUDINI programs can compose two classes of functions: differentiable functions $\oplus_w$ from a library $\mathcal{L}$, that are parameterized by weights w and implemented by neural networks, and a set of symbolic combinators (described below).
- The heavy use of *symbolic, higher-order combinators* like $\mathbf{map}$, $\mathbf{fold}$, and $\mathbf{conv}$ (convolution). These allow compact expression of complex neural architectures: deep feedforward networks can be represented by $\oplus_1 \circ \ldots \oplus_k$, where each $\oplus_i$ is a neural function and $\circ$ denotes composition; recurrent nets can be expressed as $\mathbf{fold}\ \oplus$, where $\oplus$ is a neural function. The same combinators can also express patterns of recursion in procedural tasks.
- The use of a strong *type discipline* to distinguish between neural computations over different forms of data, and to avoid generating provably incorrect programs during symbolic exploration. Types in HOUDINI include base types such as reals and booleans, tensors over these base types, lists and graphs whose nodes are tensors, and first-

[1]University of Edinburgh [2]Rice University. Correspondence to: Swarat Chaudhuri <swarat@rice.edu>.

Published at the ICML workshop Neural Abstract Machines & Program Induction v2 (NAMPI) — Extended Abstract, Stockholm, Sweden, 2018.

	Task 1	Task 2
RNN	0.37	5.96
HOUDINI	0.38	1.53

(a) Low-level transfer (task sequence GS1).

	Task 1	Task 2	Task 3
RNN	1.21	5.33	6.16
HOUDINI	1.32	1.64	3.44

(b) High-level transfer (task sequence GS2).

Table 1. Lifelong learning on graphs. Column 1: RMSE on speed/distance from image. Column 2: RMSE on shortest path.

class functions.

Learning. Our learning algorithm consists of a symbolic program synthesis module and a gradient-based optimization module. The former module repeatedly generates parameterized programs and "proposes" them to the latter module, which uses SGD to find optimal parameters. HOUDINI currently uses two synthesis algorithms: *top-down iterative refinement*, as in the λ^2 synthesizer (Feser et al., 2015), and an *evolutionary algorithm* inspired by work on functional genetic programming (Briggs & O'neill, 2006). The former algorithm performed better in our experiments.

To enable transfer across a series of learning tasks, we add back to the library all neural functions whose parameters have been discovered during a round of learning. The parameter vectors of these functions are frozen and can no longer be updated by subsequent tasks. Thus, we prevent catastrophic forgetting by design. Importantly, it is always possible for the synthesizer to introduce "fresh networks" whose parameters have not been pretrained.

3. Results: Shortest path in a grid of images

Figure 1. A grid of 32x32x3 images from the GTSRB dataset (Stallkamp et al., 2012). The least-cost path from the top left to the bottom right node is marked.

Detailed results on HOUDINI are available in an online technical report. In this abstract, we only give experimental evidence on a single benchmark example. This example generalizes a navigation task previously proposed by Gaunt et al. (2017).

Suppose we are given a grid of images (e.g., Figure 1), whose elements represent speed limits and are connected horizontally and vertically, but not diagonally. Passing through each node induces a penalty, which depends on the node's speed limit, with lower speed limit having a higher penalty. At a high level, our goal is to predict the minimum cost $d(u)$ incurred while traveling from a fixed starting point $init$ (the top left element) to each of the remaining nodes u.

To evaluate transfer in this setting, we instantiate this task in two settings: one (shortest_path_street) in which the image grid consists of speed limit signs, and another (shortest_path_mnist) in which the image grid consists of MNIST digits. We also consider two elementary tasks: returning the speed value from an image of speed limit sign (regress_speed), and returning the value from a digit image from MNIST dataset (regress_mnist). Now we design the following task sequences, with the expectation that learning of earlier tasks will benefit learning of later tasks.

- **GS1:** Learning of complex algorithms.
  Task 1: regress_speed; *Task 2*: shortest_path_street
- **GS2:** High-level transfer of complex algorithms.
  Task 1: regress_mnist; *Task 2*: shortest_path_mnist; *Task 3:* shortest_path_street

Of these, the former sequence involves low-level transfer, in which earlier tasks are perceptual tasks like recognizing digits, while later tasks introduce higher-level algorithmic problems. The latter sequence involves higher-level transfer, in which earlier tasks introduce a high-level concept, later tasks require a learner to re-use this concept on different perceptual inputs. We compare HOUDINI against a monolithic network (specifically, an RNN).

Table 1 compares the Root Mean Square Error between the actual and predicted shortest path distance in the two task sequences. We see that the programs learned by HOUDINI on the later tasks in the sequences have significantly less error than the RNN. For the shortest_path_street task in GS1, HOUDINI learns a graph-convolution-based program "$\mathbf{repeat}(9, \mathbf{conv\_g}(\mathrm{nn\_gs1\_2})) \circ \mathbf{map\_g}(\mathrm{lib.nn\_gs1\_1})$", where $\mathrm{nn\_gs1\_2}$ is a learned function that is broadly similar to the "relaxation" operator in classic shortest-path algorithms, and $\mathrm{lib.nn\_gs1\_1}$ is the neural module learned for the earlier task regress_speed. This algorithm can be seen as an approximation of the dynamic-programming-based Bellman-Ford shortest path algorithm.

For the shortest_path_street task in the graph sequence GS2, HOUDINI learns a program "$\mathbf{repeat}(9, \mathbf{conv\_g}(\mathrm{lib.nn\_gs2\_2})) \circ \mathbf{map\_g}(\mathrm{nn\_gs2\_5})$", where $\mathrm{nn\_gs2\_5}$ is the newly learned regression function for the street signs and $\mathrm{lib.nn\_gss\_2}$ is the relaxation function already learned from the earlier task shortest_path_mnist. Thus, a graph program with the relaxation function learned on MNIST can be applied to a graph of street signs, suggesting that a domain-general operation is being learned.

References

Allamanis, Miltiadis, Chanthirasegaran, Pankajan, Kohli, Pushmeet, and Sutton, Charles. Learning continuous semantic representations of symbolic expressions. In *International Conference on Machine Learning (ICML)*, 2017. URL http://arxiv.org/abs/1611.01423.

Alur, Rajeev, Bodík, Rastislav, Juniwal, Garvit, Martin, Milo M. K., Raghothaman, Mukund, Seshia, Sanjit A., Singh, Rishabh, Solar-Lezama, Armando, Torlak, Emina, and Udupa, Abhishek. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design*, FMCAD, pp. 1–17, 2013.

Bosnjak, Matko, Rocktäschel, Tim, Naradowsky, Jason, and Riedel, Sebastian. Programming with a differentiable forth interpreter. In *International Conference on Machine Learning, ICML*, pp. 547–556, 2017.

Briggs, Forrest and O'neill, Melissa. Functional genetic programming with combinators. In *Proceedings of the Third Asian-Pacific workshop on Genetic Programming, ASPGP*, pp. 110–127, 2006.

Bunel, Rudy R., Desmaison, Alban, Mudigonda, Pawan Kumar, Kohli, Pushmeet, and Torr, Philip H. S. Adaptive neural compilation. In *Advances in Neural Information Processing Systems 29*, pp. 1444–1452, 2016.

Feser, John K., Chaudhuri, Swarat, and Dillig, Isil. Synthesizing data structure transformations from input-output examples. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 229–239, 2015.

Gaunt, Alexander L., Brockschmidt, Marc, Singh, Rishabh, Kushman, Nate, Kohli, Pushmeet, Taylor, Jonathan, and Tarlow, Daniel. Terpret: A probabilistic programming language for program induction. *CoRR*, abs/1608.04428, 2016.

Gaunt, Alexander L., Brockschmidt, Marc, Kushman, Nate, and Tarlow, Daniel. Differentiable programs with neural libraries. In *International Conference on Machine Learning (ICML)*, pp. 1213–1222, 2017.

Kipf, Thomas N. and Welling, Max. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, and Zemel, Richard. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Olah, Christopher. Neural networks, types, and functional programming, 2015. http://colah.github.io/posts/2015-09-NN-Types-FP/.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. 2017.

Socher, Richard, Perelygin, Alex, Wu, Jean Y, Chuang, Jason, Manning, Christopher D, Ng, Andrew Y, and Potts, Christopher. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

Solar-Lezama, Armando. Program sketching. *STTT*, 15 (5-6):475–495, 2013.

Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0), 2012.

Thrun, Sebastian and Mitchell, Tom M. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1-2): 25–46, 1995.