Imitation Learning of Hierarchical Programs via Variational Inference

Roy Fox [*1] Richard Shin [*1] Pieter Abbeel [1] Ken Goldberg [1 2] Dawn Song [1] Ion Stoica [1]

The design of controllers that operate in dynamical systems to perform specified tasks has traditionally been manual. Machine learning algorithms enable data-driven generation of controllers, also called policies or programs, and differ in how a user may convey what task the controller should perform. In Imitation Learning (IL), the user demonstrates a *supervisor* control signal in a set of execution traces, and the objective is to train from this data a controller that performs the computation correctly on unseen inputs.

This paper takes a hierarchical imitation learning approach to program synthesis. We model the controller as a set of Parametrized Hierarchical Procedures (PHPs) (Fox et al., 2018), each of which can invoke a sub-procedure, take a control action, or terminate and return to its caller. The PHP model maintains a similar call-stack to that of Neural Programmers–Interpreters (NPI) (Reed & De Freitas, 2015; Li et al., 2016), but makes discrete and interpretable procedure calls, rather than smoothing over continuous values.

We consider a dataset of *weakly supervised* demonstrations, in which the user provides observation–action trajectories executed by an expert controller. Inferring hierarchical structure from weak supervision in which the structure is never observed is challenging, particularly with deep multi-level hierarchies. To facilitate discovery of the structure we augment the dataset with *strongly supervised* demonstrations of not only the control actions to take, but also the internal structure of the program flow that led to these actions. Training from a mixture of strongly and weakly supervised trajectories can discover highly informative structures from few strongly supervised trajectories, and leverage these structures in learning models with good generalization from a larger amount of weakly supervised trajectories.

We propose to use autoencoders to train hierarchical procedures from weakly supervised data, a method that showed success in unsupervised learning (Baldi, 2012; Kingma & Welling, 2013; Rezende et al., 2014; Zhang et al., 2017). We present a novel method of stochastic variational inference (SVI), where a hierarchical inference model is trained to approximate the posterior distribution of the latent hierarchical structure given the observable traces, and used to impute the call-stack of hierarchical procedures and guide the training of the generative model, i.e. the procedures themselves.

The contributions of this paper are: (1) extending the PHP model with procedures that take arguments; (2) a hierarchical variational inference method for training PHPs from weak supervision. Our method generalizes Stochastic Recurrent Neural Networks (SRNNs) (Fraccaro et al., 2016) to hierarchical controllers. Compared to level-wise hierarchical training via the Expectation–Gradient (EG) method (Fox et al., 2018), our SVI approach applies to deeper hierarchies and to procedures that take arguments.

Hierarchical Variational Inference. A Partially Observable Markov Decision Process (POMDP) with states $s_t \in \mathcal{S}$, observations $o_t \in \mathcal{O}$, and actions $a_t \in \mathcal{A}$ has dynamics $p(s_{t+1}, o_{t+1} | s_t, a_t)$. A controller with memory state $m_t \in \mathcal{M}$ has policy $p_\theta(m_t, a_t | m_{t-1}, o_t)$.

We consider weakly supervised demonstrations, which are observation–action traces $\xi = (o_0, a_0, \ldots, o_{T-1}, a_{T-1})$, and strong supervision further augmented by the trajectory of internal agent states $\zeta = (m_0, \ldots, m_{T-1})$. In our extension of the Parametrized Hierarchical Procedures (PHP) model (Fox et al., 2018), the memory state is a call-stack $m = [(h_0, u_0, \tau_0), \ldots, (h_d, u_d, \tau_d)]$, each frame in the stack consisting of the identifier $h \in \mathcal{H}$ of a PHP, its argument $u \in \mathcal{U}$, and its program counter $\tau \in \mathbb{Z}^+$. A PHP is a function $\pi^h : (u, \tau, o) \mapsto \texttt{operation}$, represented by a neural network. We repeatedly take a step of the top PHP h_d with inputs (u_d, τ_d, o_t), deciding to either (1) terminate and be popped from the stack; (2) call a sub-procedure h_{d+1} with argument u_{d+1}, pushing $(h_{d+1}, u_{d+1}, 0)$ onto the stack; or (3) perform an action a_t, setting m_t to the state of the call-stack at that point. The counter τ_d advances for each non-terminating PHP step. PHPs generalize nested *options* (Sutton et al., 1999) by allowing their operation to depend on u and τ.

We represent each PHP as a differentiable parametric model, outputting the log-probability of the PHP step $\log \pi^h(\texttt{operation} | u, \tau, o)$. The log-probability of each time step $\log p_\theta(m_t, a_t | m_{t-1}, o_t)$ breaks down into the sum

[*]Equal contribution [1]EECS, [2]IEOR, UC Berkeley. Correspondence to: Roy Fox <royf@berkeley.edu>.

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

of log-probabilities of the PHP steps (pops and pushes) that transition the call-stack from m_{t-1} to m_t. For strongly supervised demonstrations (ζ, ξ), we can thus use supervised learning to maximize the log-likelihood

$$\log p_\theta(\zeta, \xi) = \sum_{t=0}^{T-1} \log p_\theta(m_t, a_t | m_{t-1}, o_t) + \text{const.} \quad (1)$$

In weak supervision, where ζ is latent, we propose an amortized SVI method that replaces the log-likelihood $\log p_\theta(\xi)$ with its evidence lower bound (ELBO)

$$\mathbb{E}_{\zeta|\xi \sim q_\phi}\left[\log \frac{p_\theta(\zeta, \xi)}{q_\phi(\zeta|\xi)}\right], \quad (2)$$

where $q_\phi(\zeta|\xi)$ is an *inference network* that approximates the computationally infeasible posterior $p_\theta(\zeta|\xi)$ induced by the *generator network*, i.e. the actual PHPs.

We propose an architecture for the inference network q_ϕ that extends SRNNs (Fraccaro et al., 2016) to support our hierarchical structure. We start by concatenating each observation–action pair (o_t, a_t) in ξ, and feeding this sequence into a bidirectional RNN. The output b_t of the RNN at every time step is a *posterior context* — a sequence in which each element is a function of the entire trace ξ, allowing the decomposition $q_\phi(\zeta|\xi) = \prod_{t=0}^{T-1} q_\phi(m_t | m_{t-1}, b_t)$.

Using the posterior context, we proceed to define $q_\phi(m_t | m_{t-1}, b_t)$ similarly to $p_\theta(m_t, a_t | m_{t-1}, o_t)$, as a product of PHP steps, except that *posterior PHPs* $\pi^h_\phi(\texttt{operation} | u, \tau, b_t)$ are used instead of the usual PHPs $\pi^h_\theta(\texttt{operation} | u, \tau, o_t)$. Since each transition is conditioned on the true action taken in that time step, posterior PHPs have structural constraints on their allowed outputs, enforced via masking the output logits (before `log_softmax` normalization), namely: only ancestors of the true action in the call-graph can be called; the root PHP cannot terminate before the final time step; and all PHPs must terminate at the final time step.

SVI estimates the ELBO (2) by sampling ζ from the inference network q_ϕ and computing the log probability ratio of p_θ to q_ϕ. We minimize this loss by stochastic gradient descent on θ and ϕ. To allow both sampling and gradients of q_ϕ, we use the relaxed one-hot categorical distribution, i.e. apply `softmax` to the logits after adding independent Gumbel variables (Jang et al., 2016; Maddison et al., 2016). Sampling is facilitated by the straight-through approximation, i.e. using `argmax` for samples and `softmax` for log-probabilities and gradient backpropagation. Note that this prevents gradient backpropagation through PHP steps.

Experiment 1: MNIST Elevator. We evaluate our method on a new benchmark called *MNIST Elevator*, designed to test the ability of PHPs to learn to generate arguments for sub-procedures. The root procedure, `elevator`, receives as argument the target floor as an MNIST digit (LeCun, 1998). It should then pass the one-hot decoded digit to a `navigate` procedure, which decides based on the current floor, observed as another MNIST digit, whether to go `up` or `down`. The main challenge in this domain is to train convolutional neural networks — namely, the `navigate` procedure in strong supervision and both procedures in weak supervision — to classify MNIST digits from only the order relation between two digits.

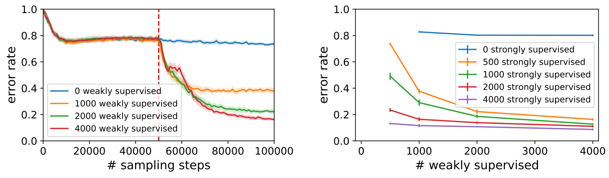

Figure 1. Results in the MNIST Elevator domain.

Table 1. Results for the Karel domain

	Program A				Program B				Program C			
# fully supervised	8	8	8	8	8	8	8	8	8	8	8	8
# total	8	16	32	64	8	16	32	64	8	16	32	64
Accuracy	93%	93%	**100%**	**100%**	81%	75%	**89%**	**89%**	88%	90%	91%	**94%**

We found that pre-training with only strong supervision is needed to prevent the weak supervision from obscuring the strong supervision signal. We therefore triggered a late onset of the weak supervision signal, adding it to the mixture after 50000 trajectory samples (Figure 1). This immediately showed a significant improvement in the error rate, i.e. the fraction of imperfectly reproduced test traces, matching that of a larger strongly supervised dataset.

Experiment 2: Karel. Karel is an educational programming language (Pattis, 1981; Devlin et al., 2017; Bunel et al., 2018), generating sequences of actions for a robot in a grid world. Each cell in the grid can contain either a wall, or between 0 and 10 *markers*. The robot can `move` (forward), `turnLeft`, `turnRight`, `pickMarker`, or `putMarker`. The observations consist of `leftIsClear`, `rightIsClear`, `frontIsClear`, and `markersPresent`.

Each Karel program has its own hierarchical structure. We provide strong supervision by automatically parsing a given Karel program to create one PHP for the top-level function, and one for each control-flow construct in the program.

Table 1 summarizes our results on three textbook Karel programs. Overall, we see a benefit from training on more weakly supervised traces. Notably, for programs A and B, using 8 strongly supervised demonstrations out of 64 total demonstrations achieved similar results to having all 64 demonstrations strongly supervised, showing effective learning from weakly supervised demonstrations.

Acknowledgements

This research is supported in part by DHS Award HSHQDC-16-3-00083, NSF CISE Expeditions Award CCF-1139158 Berkeley DeepDrive, NSF Grant No. TWC-1409915, DARPA Grant No. FA8750-17-2-0091, NSF NRI Award 1734633, and gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, GE, Google, Huawei, Intel, IBM, Microsoft, Scotiabank, Splunk, VMware, Siemens, Cisco, Autodesk, Toyota Research, Samsung, Knapp, and Loccioni Inc.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above organizations.

References

Baldi, P. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 37–49, 2012.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A.-r., and Kohli, P. Robustfill: Neural program learning under noisy i/o. *arXiv preprint arXiv:1703.07469*, 2017.

Fox, R., Shin, R., Krishnan, S., Goldberg, K., Song, D., and Stoica, I. Parametrized hierarchical procedures for neural programming. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJl63fZRb.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pp. 2199–2207, 2016.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Li, C., Tarlow, D., Gaunt, A. L., Brockschmidt, M., and Kushman, N. Neural program lattices. 2016.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Pattis, R. E. *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, Inc., 1981.

Reed, S. and De Freitas, N. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Zhang, R., Isola, P., and Efros, A. A. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *CVPR*, volume 1, pp. 6, 2017.