Hierarchical Imitation Learning via Variational Inference of Control Programs

Roy Fox[1], **Richard Shin**[1], **William Paul**[1], **Yitian Zou**[1],

Dawn Song[1], **Ken Goldberg**[1,2], **Pieter Abbeel**[1], **Ion Stoica**[1]

Abstract

Autonomous controllers can be trained by imitation learning from demonstrations of the intended control. Hierarchical imitation learning in the parametrized hierarchical procedures (PHP) framework can reduce the required number of demonstrations by allowing each procedure to specialize in specific behavior and abstract away from transient state features. We propose a variational inference method for discovering the latent hierarchical structure in observation–action traces of teacher demonstrations. We train an inference model to approximate the posterior distribution of the latent call-stack of hierarchical procedures, and sample from it to guide the training of the hierarchical controller. Our method requires 40 demonstrations, less than half as many as end-to-end RNN training, to achieve 88% success rate in training the BubbleSort algorithm.

1 Introduction

Autonomous controllers that operate in dynamical systems can be trained to perform specified tasks when provided with informative learning signals. In imitation learning (IL), a teacher provides demonstrations of successful performance of the task, which consist of traces of sensory observations and the intended control actions during execution of the teacher's control policy [1, 2, 3, 4] or of the learning controller [5, 6, 7]. The main benefit of IL is that the rich supervision signal often keeps sample efficiency high, requiring a manageable amount of interaction with the system and the teacher. By comparison, the reward function in reinforcement learning, which only scores system states and control actions, is a much weaker and sparser supervision signal that often entails very high sample complexity.

Control structure, and in particular hierarchical control, has the potential to further reduce the required number of demonstrations for learning successful controllers that generalize to states not seen during training. Modularity and hierarchy facilitate specialization and abstraction, so that each distinct component of the controller can focus on simpler behavior in a subset of system states, thus reducing the complexity of the relevant features and of the noise structure. We take a hierarchical imitation learning (HIL) approach, by modeling the controller as a set of parametrized hierarchical procedures (PHPs) [8], where each procedure can either invoke a sub-procedure, take a control action, or terminate and return to its caller. We train the control policy's parameters from a dataset of observation–action trajectories executed by a teacher controller.

The challenge in this setting is that the hierarchical structure is latent in the teacher's demonstrations and must be inferred. We propose to use amortized variational inference to train hierarchical controllers from demonstration traces, a method that showed success in unsupervised learning [9, 10,

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
[2]Department of Industrial Engineering and Operations Research, University of California, Berkeley

Infer to Control: NIPS 2018 Workshop on Probabilistic Reinforcement Learning and Structured Control.

11, 12]. We train an inference model to approximate the posterior distribution of the latent call-stack of hierarchical procedures given the observable demonstration, and use it to impute the sequence of sub-procedure calls and their arguments, and guide the training of the generator model, i.e. the procedures themselves. This parametric approach to inference allows us to extend prior work in inference of hierarchical controllers [13, 14, 15, 16, 8] to learn deeper hierarchies of hybrid structure: discrete procedure calls that accept continuous arguments.

We train a 4-level hierarchical controller to perform the BubbleSort algorithm from demonstrations of the algorithm's execution. Our method requires 40 demonstrations, less than half as many as end-to-end training of a 4-layer RNN, to completely match 88% of test traces. These preliminary results suggest that variational inference can successfully discover the latent structure from teacher demonstrations, and use it to efficiently train hierarchical controllers.

2 Related Work

Frameworks of hierarchical control often include explicit or implicit call-stacks. In the popular options framework [17], options can use "intra-option" control to call other options. Stack RNNs [13] can explicitly perform stack operations as part of their control. Neural Programmers–Interpreters [18] control program execution by maintaining a call-stack of procedures and their arguments. Neural Program Lattices [19] add the ability to train the same NPI model when the hierarchical structure is latent in some of the demonstrations, using lattices to group the exponentially many latent stack trajectories into a manageable set. Parametrized hierarchical procedures (PHPs) [8] maintain a similar call-stack, and train it with an exact inference method that is limited to shallow hierarchies without procedure arguments. In this work, we extend PHPs to train multi-level hierarchies of procedures that take arguments, via a variational inference method based on the principles of autoencoders [10, 11]. Variational inference was previously used to model time series [20, 21, 22]. Our method extends SRNNs [22] to model hierarchical control.

3 Hierarchical Imitation Learning

3.1 Imitation Learning as Stochastic Inference

We model an agent's interaction with its environment as a Partially Observable Markov Decision Process (POMDP). At time t, the environment is in state $s_t \in \mathcal{S}$, and emits an observation $o_t \in \mathcal{O}$. Upon seeing the observation, the agent makes a stochastic choice of an update for its internal memory state to $m_t \in \mathcal{M}$, and of an action $a_t \in \mathcal{A}$, according to a policy $p_\theta(m_t, a_t | m_{t-1}, o_t)$. The environment then makes a stochastic transition to the next state and observation $p(s_{t+1}, o_{t+1} | s_t, a_t)$.

We consider the imitation learning setting in which a teacher provides a set of demonstrations $\mathcal{D} = \{x_j\}$, where each demonstration is an observable trace $x = o_0, a_0, o_1, a_1, \ldots, o_T, a_T$ induced by the execution of a teacher policy in the environment. Here a_T is the first occurrence in the sequence of a *termination* action $\varnothing$. The learning problem is to find the parameter θ of a policy p_θ that gives high likelihood to the observed data, i.e. to maximize $\log p_\theta(\mathcal{D}) = \sum_j \log p_\theta(x_j)$. Denoting the latent sequence of memory states by $z = m_0, \ldots, m_T$, we have

$$\log p_\theta(x) = \log \sum_z p_\theta(z, x) = \log \sum_z \prod_{t=0}^{T} p_\theta(m_t, a_t | m_{t-1}, o_t) + \text{const}, \tag{1}$$

with the constant incorporating the log-probability of the environment steps $p(s_{t+1}, o_{t+1} | s_t, a_t)$.

When the memory update is non-deterministic, the support of z has size exponential in the horizon T, which prevents direct optimization of (1). Existing approaches are either to have no memory state, allowing only reactive policies $p_\theta(a_t | o_t)$, or more generally to have deterministic memory updates $m_t = f_\theta(m_{t-1}, o_t)$, and by extension $z = f_\theta(x)$, e.g. using recurrent neural networks (RNNs).

In this work, we propose to instead use variational inference (VI), i.e. to train an *inference model* $q_\phi(z|x)$ to guide the training of the *generator model* $p_\theta(z, x)$. While this can be done under any structural constraint on these models, we consider controllers that have the hierarchical structure detailed below.

3.2 Parametrized Hierarchical Procedures

Structure in the policy p_θ is an inductive bias that can facilitate sample-efficient generalization from the demonstrated behavior to execution on unseen states. In this work, we build on the hierarchical control framework of parametrized hierarchical procedures (PHPs) [8], in which a hierarchical controller is modelled as a set of procedures $\mathcal{H}$. Each procedure is tasked with performing a specific behavior, which it does by breaking the task down into a sequence of simpler sub-tasks, and calling a sequence of sub-procedures and elementary actions to perform these sub-tasks. Inspired by the analogy to procedural programming, we extend PHPs to have procedures pass real-vector arguments to sub-procedures and actions, and return real-vector values to their callers upon termination.

Similarly to computer programs, the hierarchical controller's execution is managed by a call-stack, onto which sub-procedures are pushed when called, and from which they are popped when they terminate. Each *frame* in the stack has the format (h, u, τ, h^-, u^-), consisting of: (1) the procedure identifier $h \in \mathcal{H}$; (2) its argument $u \in \mathbb{R}^d$; (3) the counter $\tau \in \mathbb{N}$ of steps that the procedure executed so far; (4) the procedure's most recent sub-procedure or action, $h^- \in \mathcal{H} \cup \mathcal{A}$; and (5) $u^- \in \mathbb{R}^d$, the return value of h^-. All identifiers are one-hot encoded. For simplicity, vectors have a fixed length d.

The controller is executed by following Algorithm 1 in Appendix A. In step i of the execution, let the top stack frame be $(h_i, u_i, \tau_i, h_i^-, u_i^-)$. Then procedure h_i takes a stochastic step with distribution

$$p_\theta(h_i^+, u_i^+ | h_i, u_i, \tau_i, h_i^-, u_i^-, o_t), \tag{2}$$

where $h_i^+ \in \mathcal{H} \cup \mathcal{A} \cup \{\varnothing\}$ is the sub-procedure, action, or termination, and $u_i^+ \in \mathbb{R}^d$ is either the argument for that sub-procedure or action, or the return value for termination. For each h_i, the PHP $p_\theta(\cdot, \cdot | h_i, \ldots)$ is represented by a neural network, and jointly they all induce the stochastic process $z = \{z_i\}$, with $z_i = (h_i^+, u_i^+)$. Note that the observation o_t is not stored on the stack, and that the time t advances only in PHP steps i that take elementary actions $a_t = h_i^+ \in \mathcal{A}$.

4 Variational Inference of Control Programs

As discussed in Section 3.1, since we cannot directly optimize the objective (1), we will optimize a proxy. In amortized variational inference, this proxy is the evidence lower bound (ELBO): for any distribution $q_\phi(z|x)$, we have

$$\log p_\theta(x) = \mathbb{E}_{z|x\sim q_\phi}\left[\log \frac{p_\theta(z,x)}{q_\phi(z|x)}\right] + \mathbb{D}[q_\phi(z|x)\|p_\theta(z|x)] \geqslant \mathbb{E}_{z|x\sim q_\phi}\left[\log \frac{p_\theta(z,x)}{q_\phi(z|x)}\right]. \tag{3}$$

Maximizing the lower bound on the right-hand side of (3) over θ and ϕ is therefore trading off maximizing our log-likelihood objective on the left-hand side, with minimizing the Kullback–Leibler (KL) divergence of an inference model $q_\phi(z|x)$ from the true, computationally infeasible posterior of the generator model $p_\theta(z|x)$.

In the following, we rewrite the standard expression (3) as $\mathbb{D}[q_\phi(z, x|x)\|p_\theta(z, x)]$. This notation highlights a natural representation of q_ϕ: whereas $p_\theta(z, x)$ is the product of PHP step transitions (2), for $q_\phi(z, x|x)$ we can consider the same PHP step, with two differences. First, the distribution q_ϕ is conditioned on the entire demonstration x, both its past and future. Following stochastic recurrent neural networks (SRNNs) [22], we process the demonstration x using a bidirectional RNN $b_\phi(x)$, to obtain a time-dependent *posterior context* b_t, and by analogy to (2) compute

$$q_\phi(h_i^+, u_i^+ | h_i, u_i, \tau_i, h_i^-, u_i^-, b_t). \tag{4}$$

The second difference between *posterior PHP* steps of q_ϕ and the ordinary PHP steps of p_θ, is that we must ensure that $q_\phi(z, x_1|x_2) = 0$ for all $x_1 \neq x_2$. At time t, if we have prior knowledge that some sub-procedure cannot lead to choosing a_t, we mask in $\log q_\phi$ the logit of that sub-procedure to $-\infty$ before normalizing q_ϕ with softmax. In particular, we preclude any action selection other than a_t, including its argument.

4.1 Algorithmic Considerations

Analytic KL computation. Computing (3) "analytically", with an explicit sum over all values of z, often outperforms sampling $z|x \sim q_\phi$ [10]. This is attributed to the high variance of the gradient

of the log term in (3), particularly when $q_\phi(z|x)$ is a poor approximation of the posterior $p_\theta(z|x)$. In fact, the gradient of the log term with respect to ϕ has expectation 0, but its variance increases without bound as some values of $q_\phi(z|x)$ tend to 0. The latter is an important case in our setting, since we wish q to reveal the "correct" latent structure, and give negligible likelihood to other values of z. We also note that, in practice, the exploding variance manifests as bias (see Appendix B).

Since the support of z has size exponential in the horizon T, we break down the objective into individual PHP steps, and compute each one analytically with

$$\mathbb{D}[q_\phi(z, x|x) \| p_\theta(z, x)] = \sum_i \mathbb{E}_{z_{<i}|x\sim q_\phi}[\mathbb{D}[q_\phi(z_i, x_i|z_{<i}, x) \| p_\theta(z_i, x_i|z_{<i}, x_{<i})]]. \tag{5}$$

For every PHP step i, we sample $z_{<i} = z_0, \ldots, z_{i-1}$ by extending $z_{<i-1}$ from the previous step, and compute the one-step KL analytically. Here $x_i = (a_t, s_{t+1}, o_{t+1})$ for every PHP step i in which an action a_t is taken, and note again that the environment dynamics factors out as a constant.

Sequential Score-Function Trick. The score-function trick is a method for taking the gradient of an expectation over a parametric distribution [23]. Usefully, it can also be decomposed over PHP steps similarly to (5), to compute

$$\nabla\, \mathbb{D}[q_\phi(z, x|x) \| p_\theta(z, x)] = \sum_i (\nabla D_i + D_i \nabla \log q_\phi(\hat{z}_{<i}|x)), \tag{6}$$

where $D_i = \mathbb{D}[q_\phi(\hat{z}_i, x_i|\hat{z}_{<i}, x) \| p_\theta(\hat{z}_i, x_i|\hat{z}_{<i}, x_{<i})]$ and $\hat{z}|x \sim q_\phi$.

5 Results

Figure 1 summarizes the preliminary results of training a 4-level hierarchical controller to perform the BubbleSort algorithm using our variational inference (VI) method, compared to end-to-end training of a 4-layer RNN (the architectures and hyperparameters are detailed in Appendix C). The error rate is the fraction of test traces in which the trained policy takes the correct action in every step. These results suggest that VI is able to extract more information than end-to-end RNN training from the same amount of data. With 40 demonstrations, VI already generalizes to exactly match 88% of test traces, while the RNN requires more than 90 demonstrations to achieve that performance.

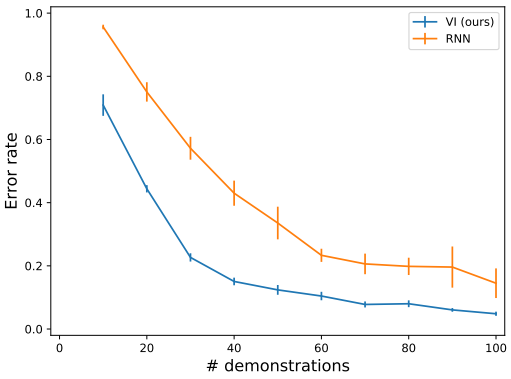

Figure 1: Fraction of completely correct test traces, averaged over 6 trials, after 50K steps of learning 4-level controllers with our VI method and end-to-end RNN training. VI is able to extract more information from the same data by discovering a latent hierarchical structure that generalizes better.

6 Conclusion

In this paper we proposed a variational inference method for training hierarchical controllers from demonstrations in which the structure is latent. In training the BubbleSort algorithm, discovery of the latent structure improved learning and reduced by more than half the number of demonstrations needed to perfectly generalize to 88% of test cases.

Our method can benefit from introducing control variates, such as RELAX [24], to reduce the variance of the gradient estimators. Tighter bounds on the log-likelihood objective, such as IWAE [25], can also help in cases where q_ϕ is insufficiently expressive to match the posterior, however an open question is how to combine such bounds with the considerations of Section 4.1.

References

[1] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[3] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[4] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

[5] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[6] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[7] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017.

[8] Roy Fox, Richard Shin, Sanjay Krishnan, Ken Goldberg, Dawn Song, and Ion Stoica. Parametrized hierarchical procedures for neural programming. In *International Conference on Learning Representations*, 2018.

[9] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49, 2012.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[12] Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *CVPR*, volume 1, page 6, 2017.

[13] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198, 2015.

[14] Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 2016.

[15] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

[16] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

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

[18] Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[19] Chengtao Li, Daniel Tarlow, Alexander L Gaunt, Marc Brockschmidt, and Nate Kushman. Neural program lattices. 2016.

[20] Tejas D Kulkarni, Ardavan Saeedi, and Samuel Gershman. Variational particle approximations. *stat*, 1050:1, 2014.

[21] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

[22] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207, 2016.

[23] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[24] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017.

[25] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Appendix

A Hierarchical Control Structure

Algorithm 1 Execute hierarchical controller with root procedure h_0 and task specification u_0

Initialize stack $\leftarrow [(h_0, u_0, 0, \varnothing, \varnothing)]$
Initialize $i \leftarrow 0$
repeat
  Let $(h_i, u_i, \tau_i, h_i^-, u_i^-)$ be the top stack frame
  Draw (h_i^+, u_i^+) with distribution $p_\theta(h_i^+, u_i^+ | h_i, u_i, \tau_i, h_i^-, u_i^-)$
  if $h_i^+ = \varnothing$ **then** // h_i terminates
    Pop the top stack frame
    If stack is not empty, set the top return value to u_i^+
  Otherwise
    Increment the top step counter
    if $h_i^+ \in \mathcal{A}$ **then** // h_i takes elementary action h_i^+
      Set the top sub-procedure to h_i^+
      Take action h_i^+ with argument u_i^+ in the environment
      Set the top return value to the action's return value
    Otherwise // h_i calls sub-procedure h_i^+
      Set the top sub-procedure to h_i^+
      Push $(h_i^+, u_i^+, 0, \varnothing, \varnothing)$ onto the stack
  $i \leftarrow i + 1$
until stack is empty

B Analytic KL Avoids Biased Gradient Estimation

We note that in practice, the exploding variance of the log term in (3) also manifests as bias. For the purpose of this analysis, let $q(z|x)$ be ϵ for $z = 0$, and $1 - \epsilon$ for $z = 1$. If we use the score-function trick to estimate

$$\partial_\epsilon \, \mathbb{E}_{z|x\sim q}\left[\log \frac{p(z,x)}{q(z|x)}\right] \approx \partial_\epsilon \log \frac{p(\hat{z},x)}{q(\hat{z}|x)} + \log \frac{p(\hat{z},x)}{q(\hat{z}|x)} \, \partial_\epsilon \log q(\hat{z}|x) \qquad \hat{z}|x \sim q,$$

we find as ϵ tends to 0, and as the rare event $z = 0$ disappears from samples of $q(z|x)$, that the estimate is

$$-\partial_\epsilon \log(1-\epsilon) + (\log p(1,x) - \log(1-\epsilon)) \, \partial_\epsilon \log(1-\epsilon) \approx 1 - \frac{\log p(1,x)}{1-\epsilon} > 0.$$

Intriguingly, even as q approaches the correct distribution and samples correct values of z, the gradient indicates, incorrectly, that the objective would be improved by increasing ϵ. This hinders the convergence of q to a correct deterministic distribution.

C Experiment Details

The architecture of the hierarchical controller consists of 6 PHPs in a partial binary tree. Each PHP is represented by two multi-layer perceptrons (MLPs), to each of which softmax is applied, to select a sub-procedure and to generate its one-hot argument. The sub-procedure MLP has a hidden layer of size 16, and the argument MLP has no hidden layers. The posterior RNNs in q_ϕ have the same architecture, applied to the output of a 32-node bidirectional LSTM instead of the observation.

The baseline RNN is a 4-layer, 64-node LSTM, with each of the 4 layers feeding into the next. The input observation is preprocessed by a 64-node MLP, and the last layer's output is postprocessed by a 64-node MLP to which softmax is applied to select the action.

The optimization algorithm is Adam with weight decay 10^{-3} and PyTorch default hyperparameters. Each batch consists of 4 traces. The model is trained from demonstration datasets of varying sizes for 50K steps, where a step consists of one training trace sampled from the dataset.