Optimal Hierarchical Policy Extraction From Noisy Imperfect Demonstrations

Karan Goel
Machine Learning Department
Carnegie Mellon University
kgoel93@gmail.com

Tong Mu
Electrical Engineering Department
Stanford University
tongm@stanford.edu

Emma Brunskill
Computer Science Department
Stanford University
ebrun@cs.stanford.edu

Abstract

In many real-world scenarios, we want to learn the optimal policy for a task with access to only (possibly imperfect) demonstrations for the task. In this work, we propose a gradient based approach to extract a set of option policies in the function approximation setting, directly from off-policy demonstration data. Our method builds on recent work in option extraction in the imitation learning setting [4], extending it to the full reinforcement learning setting, with our extracted policies directly maximizing the expected reward on the task. A preliminary set of experiments on a small task demonstrate that our method is much more robust to noisy demonstrations than prior work.

1 Introduction and Related Work

In recent years, there has been rapid progress in hierarchical reinforcement learning, making it possible to learn hierarchical policies end-to-end [2, 5, 4, 14], leveraging hierarchical policies for multi-task learning [1, 10], using them to speed up on-policy learning [4], etc.

Demonstrations can be a readily available, rich source of information for learning a task. However, demonstrations are not always perfect, and it is useful if we can leverage demonstrations which exhibit varied returns on the task, to learn the optimal policy. Several recent papers have demonstrated promising results for imitation learning [8, 7, 3, 6] but there is limited work on learning hierarchical policies from imperfect demonstrations.

At the same time there is a long history of work in off-policy policy learning [11, 9] from batch data. The main challenge in learning from off-policy data is that the behavior policy used to generate the dataset may be quite sub-optimal.

In this paper, we describe a method for learning option policies from off-policy data, where the objective is to find a high performing policy on the task. We assume that the options being executed at each step are latent variables and only the number of options is known in advance. Our work is closely related to recent work in extracting options for imitation learning from demonstrations [4], and extends it to the general reinforcement learning setup by optimizing directly for the extracted policy's performance.

Hierarchical Reinforcement Learning Workshop at the 31st Conference on Neural Information Processing Systems (HRL@NIPS 2017), Long Beach, CA, USA.

The eventual goal of this work is to be able to use the extracted options for multi-task learning, demonstrate speedup for tasks where some demonstration data could be used to bootstrap on-policy learning, and test whether forcing the learned policy to be hierarchical can improve the ability of off-policy learning to find a better policy than flat off-policy learning.

Section 3 describes our approach in detail, while Section 4 presents a preliminary results on a small domain.

2 Preliminaries

Markov Decision Process (MDP). An MDP is a 6 tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, p_0 \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions (for ease of exposition, these are assumed to be discrete sets), $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1)$ is a discount factor used to weigh the relative importance of future rewards and $p_0 : \mathcal{S} \to [0, 1]$ is an initial state distribution. A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines a conditional distribution over the actions given a state. Let $\tau = (s_0, a_0, r_0, \ldots, s_H)$ denote a trajectory sequence of states, primitive actions and rewards of length H.

The value of a policy π is defined as the expected reward that we can accumulate in H steps on following π: $V_\pi = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{H-1} \gamma^t r_t \right]$ where $p(\tau) = p_0(s_0) \prod_{t=0}^{H-1} \pi(a_t|s_t) \mathcal{P}(s_{t+1}|s_t, a_t)$.

Options Framework. An option is a temporally extended action defined by a closed-loop policy [12]. An option $o \in \Omega$ is a 3 tuple $\langle \mathcal{I}_o, \pi_o, \psi_o \rangle$ where $\mathcal{I}_o \subseteq \mathcal{S}$ is set of states in which o can be initiated, $\pi_o : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is an intra-option policy and $\psi_o : \mathcal{S} \to [0, 1]$ is a termination function. As in Fox et al. [4], we assume that options are available everywhere, *i.e.* $\mathcal{I}_o = \mathcal{S}, \ \forall o \in \Omega$. A meta-policy $\eta : \mathcal{S} \times \Omega \to [0, 1]$ decides how options should be picked.

We employ the following option execution model: the agent first picks an option o according to the meta-policy η, then follows π_o until a termination signal b is sampled from ψ_o. The process then repeats. We will concern ourselves with the function-approximation setting, letting θ parameterize all our policies and termination functions – $\eta_\theta(o_t|s_t)$ is the meta-policy over options, $\pi_\theta(a_t|s_t, o_t)$ are intra-option policies and $\psi_\theta(b_{t+1}|o_t, s_{t+1})$ are the option-termination functions. The number of options K is fixed and assumed to be known a-priori.

Importance Sampling. Importance sampling is a statistical technique that can be used to evaluate the expectation of a random variable $E_{X \sim p(X)}[f(X)]$ when it is difficult to sample directly from its distribution $p(X)$ [11]. We can instead sample from a different distribution $q(X)$ and use $E_{X \sim q(X)} \left[\frac{p(X)}{q(X)} f(X) \right]$ to compute the expectation. Importance sampling is typically used in RL for off-policy learning [13].

3 Off-Policy Option Extraction

We assume a setting where we are given a dataset $\mathcal{D} = (\tau_1, \ldots, \tau_n)$ consisting of trajectories with $\tau_1, \ldots, \tau_n \sim \tilde{\pi}$ where $\tilde{\pi}$ is a behavior policy. We denote by $\zeta = (o_0, b_1, o_1, \ldots, o_H)$ the latent option labeling associated with any trajectory $\tau = (s_0, a_0, r_0, \ldots, s_H)$. Note that ζ is unknown and must be estimated in conjunction with the parameters θ.

Our objective is to find a θ such that $V_\theta = \mathbf{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{H-1} \gamma^t r_t \right] = \mathbf{E}_{\tau \sim p_\theta(\tau)} \left[R_\tau \right]$ is maximized *i.e.* finding the policy with highest value. This is the off-policy policy learning problem. Under the generative model used by the agent to act, we can write out $p_\theta(\tau, \zeta)$ as,

$$p_\theta(\tau, \zeta) = \left(p_0(s_0) \eta_\theta(o_0|s_0) \prod_{t=0}^{H-1} \pi_\theta(a_t|s_t, o_t) \mathbb{P}_\theta(o_{t+1}, b_{t+1}|s_{t+1}, o_t) \mathcal{P}(s_{t+1}|s_t, a_t) \right)$$

$$\text{where } \mathbb{P}_\theta(o_{t+1}, b_{t+1} = 1|s_{t+1}, o_t) = \psi_\theta(b_{t+1}|o_t, s_{t+1}) \eta_\theta(o_{t+1}|s_{t+1})$$

$$\text{and } \mathbb{P}_\theta(o_{t+1}, b_{t+1} = 0|s_{t+1}, o_t) = (1 - \psi_\theta(b_{t+1}|o_t, s_{t+1})) \mathbb{I}[o_t = o_{t+1}]$$

and thus $p_\theta(\tau) = \sum_\zeta p_\theta(\tau, \zeta)$. Note that $\sum_\zeta p_\theta(\tau, \zeta)$ involves summing over all possible configurations (an exponential number) of the latent option labeling ζ. However, it is possible to do this efficiently using a forward-backward dynamic programming algorithm. The details of this algorithm remain identical to that laid out in Fox et al. [4] (see Appendix A of [4]) and can be computed without any knowledge of the environment dynamics. We omit discussion of this due to lack of space.

Our setup is similar to Fox et al. [4] with some crucial differences. Fox et al. [4] operate in a setting where reward information is absent in the demonstrations, and the demonstrations are assumed to be of expert or near-expert quality. Their objective is imitation learning – they seek parameters θ that maximize the likelihood of the dataset *i.e.* such that $\mathcal{L}(\mathcal{D}; \theta)$ is maximized.

In our setting, by contrast, we seek the optimal policy without making any assumptions on the quality of the behavior policy used to generate the dataset. In fact, our results will apply in cases where the dataset may come from multiple behavior policies, although we omit this discussion due to lack of space.

3.1 Policy Evaluation

We start by rewriting our objective using importance sampling, so that it depends on the distribution over trajectories induced by the behavior policy $\tilde{\pi}$,

$$V_\theta = \mathbf{E}_{\tau \sim p_\theta(\tau)} \left[R_\tau \right] = \mathbf{E}_{\tau \sim p_{\tilde{\pi}}(\tau)} \left[\frac{p_\theta(\tau)}{p_{\tilde{\pi}}(\tau)} R_\tau \right] \geq \mathbf{E}_{\tau \sim p_{\tilde{\pi}}(\tau)} \left[p_\theta(\tau) R_\tau \right] \tag{1}$$

The last inequality assumes a discrete action space (although the state space can be continuous since the dynamics in the numerator/denominator cancel out), so that $p_{\pi_b}(\tau) \leq 1$. In situations where we may not have access to the behavior policy π_b, we can instead optimize this lower bound on our objective (this may be a loose lower bound if π_b is very stochastic which is more likely in a partially observable MDP setting). Note that $p_\theta(\tau)$ does not contain the dynamics of the system, which disappear in the importance sampling ratio – we will abuse notation and continue to use $p_\theta(\tau)$, without the terms corresponding to the system dynamics.

Lastly, to evaluate the quality of our current θ, we can estimate $\hat{V}_\theta = \frac{1}{n} \sum_{i=1}^n p_\theta(\tau_i) R_{\tau_i}$.

3.2 Policy Improvement

We now compute the gradient of our objective with respect to the parameters θ,

$$\begin{aligned}
\nabla_\theta \mathbf{E}_{\tilde{\pi}} \left[\frac{p_\theta(\tau)}{p_{\tilde{\pi}}(\tau)} R_\tau \right] &= \mathbf{E}_{\tilde{\pi}} \left[\frac{R_\tau}{p_{\tilde{\pi}}(\tau)} \nabla_\theta p_\theta(\tau) \right] = \mathbf{E}_{\tilde{\pi}} \left[\frac{R_\tau}{p_{\tilde{\pi}}(\tau)} \nabla_\theta \sum_\zeta p_\theta(\tau, \zeta) \right] \\
&= \mathbf{E}_{\tilde{\pi}} \left[\frac{R_\tau}{p_{\tilde{\pi}}(\tau)} \sum_\zeta \nabla_\theta p_\theta(\tau, \zeta) \right] = \mathbf{E}_{\tilde{\pi}} \left[\frac{R_\tau}{p_{\tilde{\pi}}(\tau)} \sum_\zeta p_\theta(\tau, \zeta) \nabla_\theta \log p_\theta(\tau, \zeta) \right] \\
&= \mathbf{E}_{\tilde{\pi}} \left[\frac{R_\tau}{p_{\tilde{\pi}}(\tau)} \sum_\zeta p_\theta(\zeta|\tau) p_\theta(\tau) \nabla_\theta \log p_\theta(\tau, \zeta) \right] \\
&= \mathbf{E}_{\tilde{\pi}} \left[\frac{R_\tau}{p_{\tilde{\pi}}(\tau)} p_\theta(\tau) \sum_\zeta p_\theta(\zeta|\tau) \nabla_\theta \log p_\theta(\tau, \zeta) \right] \\
&= \mathbf{E}_{\tilde{\pi}} \left[\frac{p_\theta(\tau)}{p_{\tilde{\pi}}(\tau)} R_\tau \mathbf{E}_\zeta \left[\nabla_\theta \log p_\theta(\tau, \zeta) | \tau \right] \right] \\
&\geq \mathbf{E}_{\pi_b} \left[p_\theta(\tau) R_\tau \mathbf{E}_\zeta \left[\nabla_\theta \log p_\theta(\tau, \zeta) | \tau \right] \right]
\end{aligned}$$

where the last inequality holds for the lower bound objective (applicable for discrete action spaces) described in Section 3.1.

To take a stochastic gradient step on the parameters, we sample a trajectory $\tau_i \sim \mathcal{D}$ from the dataset and then update $\theta \leftarrow \theta + \alpha \cdot \frac{p_\theta(\tau_i)}{p_{\tilde{\pi}}(\tau_i)} R_{\tau_i} \mathbf{E}_\zeta \left[\nabla_\theta \log p_\theta(\tau_i, \zeta) | \tau_i \right]$ where α is the learning rate.

Figure 1: Two-rooms environment. The goal state is marked in yellow while the agent's location is marked in green. The agent randomly spawns in a random location in the maze while the goal is fixed.

R_τ can be computed directly using the rewards in the demonstration trajectory. We have already described earlier how we can compute $p_\theta(\tau)$. $\mathbf{E}_\zeta\left[\nabla_\theta \log p_\theta(\tau,\zeta)|\tau\right]$ can also be estimated using the forward-backward procedure used for estimating $p_\theta(\tau)$. In fact, $\mathbf{E}_\zeta\left[\nabla_\theta \log p_\theta(\tau,\zeta)|\tau\right]$ is the gradient for the imitation objective optimized by Fox et al. [4]. We see that our off-policy learning objective additionally multiplies in two terms: (i) the trajectory's reward R_τ, which favors larger gradient steps for more rewarding trajectories; (ii) the importance sampling ratio $\frac{p_\theta(\tau)}{p_{\tilde{\pi}}(\tau)}$ which reweighs the trajectory.

4 Results

We present some preliminary experiments that compare variants of our method, Off-Policy Options Learning (OPOL) to Deep Discovery of Options [4] (DDO) on a small navigation task. In particular, we compare OPOL, DDO and the lower bound variant of Off-Policy Options Learning (OPOL-LB) which omits $p_{\tilde{\pi}}(\tau)$ in the gradient computation.

To have finer control over the experiments, we designed a small two-rooms environment, shown in Figure 1. For this environment, we defined a canonical 3-option hierarchical policy with which to generate the demonstrations. The first and second option in this policy are active in the two rooms while the third option is active in the corridor. Transitions in the environment are stochastic and can move the agent in a random direction 1/3rd of the time.

We tested several settings in which we either added noise to the meta policy (*i.e.* noise in the hierarchy) to the intra-option policy or both for the behavior policy. Noise in the meta-policy entails randomly picking an option with some probability, taking a random action and then terminating the option. This mimics a scenario where each option is an expert only where it is supposed to be active. Noise in the intra-option policy simply picks a random action with some probability. We generated 5000 demonstrations for our dataset using the behavior policy.

We define success as learning a policy that takes less than 20 steps to reach the goal on average. The performance of the policy must remain consistent over 5000 updates to be considered successful. We performed atmost 100,000 gradient updates (learning rate of 0.001 with Adam optimizer). The meta-policy, intra-option policies and termination functions were parameterized by a small neural network with 2 fully-connected layers of 8 units each, followed by separate output heads for each of the policies/termination functions. We set $K = 3$ for our experiments. Results of our experiments are summarized in Table 1.

We observe that both OPOL and OPOL-LB are able to learn a high performing policy even with very noisy demonstrations. Even with a large amount of noise in both the meta policy and the intra-option policies, our method is able to find a high performing policy successfully.

5 Conclusion

In this work, we introduced an off policy options learning framework to learn high performing policies directly from demonstration data. Preliminary experiments indicate the promise of our approach in dealing with noisy demonstrations. In the future, we hope to demonstrate our method on larger domains; applied to the multi-task setting, as well as to speed up learning using the extracted options in on-policy learning.

Meta Policy Noise	Intra-Option Policy Noise	OPOL	OPOL-LB Succeeded?	DDO
0%	0%	Yes	Yes	Yes
50%	0%	Yes	Yes	No
0%	25%	Yes	Yes	Yes
50%	25%	Yes	Yes	No
50%	50%	Yes	Yes	No

Table 1: Summary of results on the two-rooms domain.

References

[1] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[3] Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *CoRR*, abs/1703.07326, 2017.

[4] Roy Fox, Sanjay Krishnan, Ion Stoica, and Kenneth Y. Goldberg. Multi-level discovery of deep options. *CoRR*, abs/1703.08294, 2017.

[5] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option : Learning options with a deliberation cost. *CoRR*, abs/1709.04571, 2017.

[6] Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. *CoRR*, abs/1709.06683, 2017.

[7] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017.

[8] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

[9] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *NIPS*, 2016.

[10] Junhyuk Oh, Satinder P. Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017.

[11] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, 2000.

[12] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112:181–211, 1999.

[13] Philip S. Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*, 2016.

[14] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017.