LEARNING REUSABLE OPTIONS FOR MULTI-TASK REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) has become an increasingly active area of research in recent years. Although there are many algorithms that allow an agent to solve tasks efficiently, they often ignore the possibility that prior experience related to the task at hand might be available. For many practical applications, it might be unfeasible for an agent to learn how to solve a task from scratch, given that it is generally a computationally expensive process; however, prior experience could be leveraged to make these problems tractable in practice. In this paper, we propose a framework for exploiting existing experience by learning reusable *options*. We show that after an agent learns policies for solving a small number of problems, we are able to use the trajectories generated from those policies to learn reusable options that allow an agent to quickly learn how to solve novel and related problems.

1 INTRODUCTION

Reinforcement learning (RL) techniques have experienced much of their success in simulated environments, such as video games (Mnih et al., 2015) or board games (Silver et al., 2016; Tesauro, 1995). One of the main reasons why RL has worked so well in these applications is that we are able simulate millions of interactions with the environment in a relatively short period of time. In many real world applications, however, where the agent interacts with the physical world, it might not be easy to generate such a large number of interactions. The time and cost associated with training such systems could render RL an unfeasible approach for training in large scale.

As a concrete example, consider training a large number of humanoid robots (agents) to move quickly, as in the Robocup competition (Farchy et al., 2013). Although the agents have similar dynamics, subtle variations mean that a single policy shared across all agents would not be an effective solution. Furthermore, learning a policy from scratch for each agent is too data-inefficient to be practical. As shown by Farchy et al. (2013), this type of problem can be addressed by leveraging the experience obtained from solving a related task (e.g., walking) to quickly learn a policy for each individual agent that is tailored to a new task (e.g., running). These situations also occurs in industry, such as robots tasked with sorting items in fulfillment centers. A simple approach, like using PD controllers, would fail to adapt to the forces generated from picking up objects with different weight distributions, causing the arm to drop the objects. RL is able to mitigate this problem by learning a policy for each arm that is able to make corrections quickly, which is tailored to the robot's dynamics. However, training a new policy for each agent would be far too costly to be a practical solution.

In these scenarios, it is possible to use a small number of policies learned a subset of the agents, and then leverage the experience obtained from learning those policies to allow the remaining agents to quickly learn their corresponding policies. This approach can turn problems that are prohibitively expensive to solve into relatively simple problems.

To make use of prior experience and improve learning on new related problems in RL, several lines of work, which are complementary to each other, have been proposed and are actively being studied. *Transfer learning* (Taylor & Stone, 2009) refers to the problem of adapting information acquired while solving one task to another. One might consider learning a mapping function that allows for a policy learned in *one* task to be used in a different task (Ammar et al., 2015) or simply learn a mapping of the value function learned in one task to another (Taylor et al., 2007). These techniques

can be quite effective, but are also limited in that they consider mapping information from *one* source task to *another* target task. Another approach to reusing prior knowledge is through *meta learning* or learning to learn (Schmidhuber, 1995; Schmidhuber et al., 1998). In the context of RL, the goal under this framework is usually for an agent to be exposed to a number of tasks where it can learn some general behavior that generalizes to new tasks. For example, Finn et al. (2017) showed that an agent who learns how to walk forward is able to find a general policy that can quickly be adapted to learn to walk backwards.

One last technique to leverage prior experience, and the one this paper focuses on, is through temporally extended actions or *temporal abstractions* (McGovern & Sutton, 1998; Sutton et al., 1999). While in the standard RL framework the agent has access to a set of primitive actions (i.e., actions that last for one time step), temporally extended actions allow an agent to execute actions that last for several time-steps. They introduce a bias in the behavior of the agent which, if appropriate for the problem at hand, results in dramatic improvements in how quickly the agent learns to solve a new task. A popular representation for temporally extended actions is the *options* framework (Sutton & Precup, 1998; Sutton et al., 1999) (formally introduced in the next section), which is the focus of this work. It has been shown that options learned in a specific task or set of tasks, can be reused to improve learning on new tasks (Machado et al., 2017; Bacon et al., 2017); however, this often requires knowledge from the user about which options or how many options are appropriate for the type of problems the agent will face.

In this paper, we propose learning reusable options for a set of related tasks with minimal information provided by the user. We consider the scenario where the agent must solve a large numbers of tasks and show that after learning a (near)-optimal policy for a small number of problems, we can learn an appropriate number of options that facilitates learning in a remaining set of tasks. To do so, we propose learning a set of options that minimize the expected number of decisions needed to represent trajectories generated from the (near)-optimal policies learned by the agent, while also maximizing the probability of generating those trajectories.

2 Background and Notation

A *Markov decision process* (MDP) is a tuple, $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma, d_0)$, where $\mathcal{S}$ is the set of possible states of the environment, $\mathcal{A}$ is the set of possible actions that the agent can take, $P(s, a, s')$ is the probability that the environment will transition to state $s' \in \mathcal{S}$ if the agent executes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, $R(s, a, s')$ is the expected reward received after taking action a in state s and transitioning to state s', d_0 is the initial state distribution, and $\gamma \in [0, 1]$ is a discount factor for rewards received in the future. We use t to index the time-step and write S_t, A_t, and R_t to denote the state, action, and reward at time t. A *policy*, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, provides a conditional distribution over actions given each possible state: $\pi(s, a) = \Pr(A_t = a | S_t = s)$. We denote a trajectory of length t as $h_t = (s_0, a_0, r_0, \ldots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$, that is, h_t is defined as a sequence of states, actions and rewards observed after following some policy for t time-steps. This work focuses on learning options that can be used for a set of related tasks. We consider the setting where an agent must solve a set of related tasks, where each task is an MDP, $M = (\mathcal{S}, \mathcal{A}, P_M, R_M, \gamma, d_0^M)$; that is, each task is an MDP with its own transition function, reward function and initial state distribution, with shared state and action sets.

An option, $o = (\mathcal{I}_o, \mu_o, \beta_o)$, is a tuple in which $\mathcal{I}_o \subseteq \mathcal{S}$ is the set of states in which option o can be executed (the *initiation set*), μ_o is a policy that governs the behavior of the agent while executing o, and $\beta_o : \mathcal{S} \to [0, 1]$ is a termination function that determines the probability that o terminates in a given state. We assume that $\mathcal{I}_o = \mathcal{S}$ for all options o; that is, the options are available at every state. The options framework does not dictate how an agent should choose between available options or how options should be discovered. A common approach to selecting between options is to a learn a *policy over options*, which is defined by the probability of choosing an option in a particular state. Two recent popular approaches to option discovery are eigenoptions (Machado et al., 2017) and the option-critic architecture (Bacon et al., 2017).

The *eigenoptions* (Machado et al., 2017) of an MDP are the optimal policies for a set of implicitly defined reward functions called *eigenpurposes*. Eigenpurposes are defined in terms of *proto-value functions* (Mahadevan, 2005), which are in turn derived from the eigenvectors of a modified adjacency matrix over states for the MDP. The intuition is that no matter the true reward function, the eigenoptions allow an agent to quickly traverse the transition graph, resulting in better exploration

of the state space and faster learning. However, there are two major downsides: 1) the adjacency matrix is often not known *a priori*, and may be difficult to construct for large MDPs, and 2) for each eigenpurpose, constructing the corresponding eigenoption requires solving a new MDP.

The option-critic architecture (Bacon et al., 2017) is a more direct approach to learn options and a policy over options simultaneously using policy gradient methods. One issue that often arises within this framework is that the termination functions of the learned options tend to collapse to "always terminate". In a later publication, the authors built on this work to consider the case where there is a cost associated with switching options (Harb et al., 2018). This method resulted in the agent learning to use a single option while it was appropriate and terminate when an option switch was needed, allowing it to discover improved policies for a particular task. The authors argue that minimizing the use of the policy over options may be desirable, as the cost of choosing an option may be greater than the cost of choosing a primitive action when using an option. Recent work by Harutyunyan et al. (2019) approaches the aforementioned termination problem by explicitly optimizing the termination function of options to focus on small regions of the state space. However, in contrast to the work presented in these paper, these methods do not explicitly take into consideration that the agent might face many related tasks in the future.

We build on the idea that minimizing the number of decisions made by an agent leads to the discovery of general reusable options, and propose an offline method where they are learned by solving a small number of tasks. The options are then leveraged to quickly solve new problems the agent will face in the future. We use the trajectories generated while learning (near)-optimal policies, and learn an appropriate set of options by directly minimizing the expected number of decisions the agent makes while simultaneously maximizing the probability of generating the observed trajectories.

3 Learning Reusable Options from Experience

In this section, we introduce the objective for learning a set of reusable options for a set of related tasks. Our algorithm introduces one option at a time until introducing a new option does not improve the objective further. This procedure results in a natural way of learning an adequate number of options without having to predefine it; a new option is included if it is able to improve the probability of generating optimal behavior while minimizing the number of decisions made by the agent.

3.1 Problem Formulation

In the options framework, at each time-step, t, the agent chooses an action, A_t, based on the current option, O_t. Let T_t be a Bernoulli random variable, where $T_t = 1$ if the previous option, O_{t-1}, terminated at time t, and $T_t = 0$ otherwise. If $T_t = 1$, O_t is chosen using the policy over options, π. If $T_t = 0$, then the previous option continues, that is, $O_t = O_{t-1}$. To ensure we can represent any trajectory, we consider primitive actions to be options which always select one specific action and then terminate; that is, for an option, o, corresponding to a primitive, a, for all $s \in \mathcal{S}$, the termination function would be given by $\beta_o(s) = 1$, and the policy by $\mu(s, a') = 1$ if $a' = a$ and 0 otherwise.

Let $\mathcal{O} = \mathcal{O}_{\mathcal{A}} \cup \mathcal{O}_{\mathcal{O}}$ denote a set of options, $\{o_1, \ldots, o_n\}$, where $\mathcal{O}_{\mathcal{A}}$ refers to the set of options corresponding to primitive actions and $\mathcal{O}_{\mathcal{O}}$ to the set corresponding to temporal abstractions. Furthermore, let H be a random variable denoting a trajectory of length $|H|$ generated by a near-optimal policy, and let H_t be a random variable denoting the sub-trajectory of H up to the state encountered at time-step t. We seek to find a set, $\mathcal{O}^* = \{o_1^*, \ldots, o_n^*\}$, that maximizes the following objective:

$$J(\pi, \mathcal{O}) = \mathbf{E}\big[\sum_{t=1}^{|H|} \Pr(T_t = 0, H_t | \pi, \mathcal{O}) + \lambda_1 g(H, \mathcal{O}_{\mathcal{O}})\big], \tag{1}$$

where $g(h, \mathcal{O}_{\mathcal{O}})$ is a regularizer that encourages a diverse set of options, and λ_1 is a scalar hyperparameter. If we are also free to learn the parameters of π, then $\mathcal{O}^* \in \arg\max_{\mathcal{O}} \max_{\pi} J(\pi, \mathcal{O})$. One choice for g is the average KL divergence on a given trajectory over the set of m options being learned: $g(h, \mathcal{O}_{\mathcal{O}}) = \frac{2}{m(m-1)} \sum_{o,o' \in \mathcal{O}_{\mathcal{O}}} \sum_{t=0}^{|h|-1} D_{\mathrm{KL}}\left(\mu_o(s_t) || \mu_{o'}(s_t)\right)$. [1]

Intuitively, we seek to find options that are capable of generating near-optimal trajectories with a small number of terminations. Notice that minimizing the number of terminations is the same as

[1] This term is only defined when we consider more than one option. Otherwise, we set this term to 0.

minimizing the number of decisions made by the policy over options, as each termination requires the policy to choose a new option. Given a set of options, a policy over options, and a near-optimal sample trajectory, we can calculate the joint probability for a trajectory *exactly*. Therefore, we can obtain an accurate estimate for equation 1 by averaging over a *set* of near-optimal trajectories.

3.2 Optimization Objective for Learning Options

Given that the agent must learn the corresponding policy for a set of tasks, we can use the experienced gathered from solving a subset of tasks to obtain trajectories demonstrating the optimal behavior learned for these problems. Given a set, $\mathcal{H}$, of trajectories generated from an initial subset of tasks, we can now estimate the expectation in equation 1 to learn options that can be leveraged in the remaining problems. Because the probability of generating any trajectory approaches 0 as the length of the trajectory increases, we make a slight modification to the original objective that leads to better numerical stability, and arrive to the objective $\hat{J}$ that we optimize in practice.

$$\hat{J}(\pi, \mathcal{O}, \mathcal{H}) = \frac{1}{\mathcal{H}} \sum_{h \in \mathcal{H}} \Big(\underbrace{\lambda_2 \Pr(H = h | \pi, \mathcal{O})}_{\text{probability of generating } h} - \underbrace{\frac{\sum_{t=1}^{|h|} \mathbf{E}\left[T_t = 1 | H_t = h_t, \pi, \mathcal{O}\right]}{|h|}}_{\text{expected number of terminations}} + \underbrace{\lambda_1 g(h, \mathcal{O}_{\mathcal{O}})}_{\text{encourage diverse options}} \Big) . \tag{2}$$

A more detailed discussion on how we arrived to this objective from J is provided in Appendix A

We can express equation 2 entirely in terms of the policy over options π, options $\mathcal{O} = \{o_1, \ldots, o_n\}$ and the transition function, P. [2] The following theorems show how to calculate the first two terms in equation 2 from known quantities, allowing us to efficiently maximize the proposed objective.

Theorem 1. *Given a set of options, $\mathcal{O}$, and a policy, π, over options, the expected number of terminations for a trajectory h is given by:*

$$\sum_{t=1}^{|h|} \mathbf{E}\Big[T_t = 1 \Big| H_t = h_t, \pi, \mathcal{O}\Big] = \sum_{t=1}^{|h|} \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\sum_{o' \in \mathcal{O}} \mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}, \tag{3}$$

$$\Pr(O_t = o | H_t = h_t, \pi, \mathcal{O}) = \Big[\Big(\pi(s_t, o) \beta_o(s_t) \Big) + \Big(P(s_{t-1}, a_{t-1}, s_t) \alpha_{t-1}(o)(1 - \beta_o(s_{t-1})) \Big) \Big],$$

and $\Pr(O_0 = o | H_0 = h_0, \pi, \mathcal{O}) = \pi(s_0, o)$.

Proof. See Appendix B. □

Theorem 2. *Given a set of options $\mathcal{O}$ and a policy π over options, the probability of generating a trajectory h of length $|h|$ is given by:*

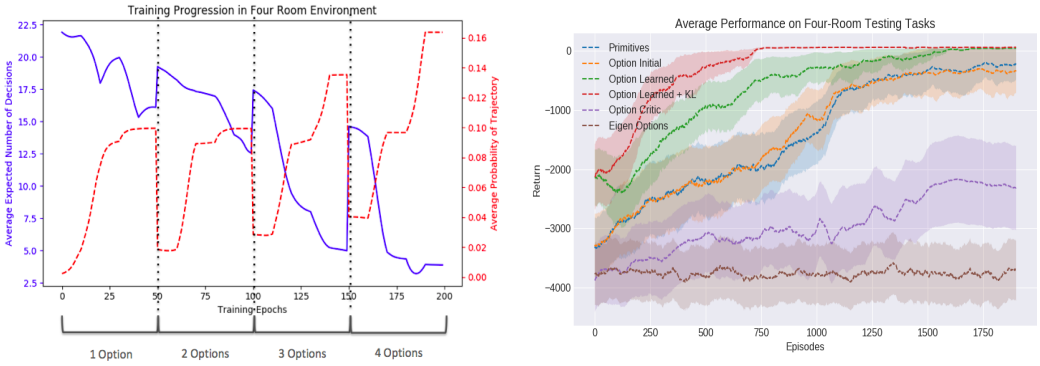
$$\Pr(H_{|h|} = h_{|h|} | \pi, \mathcal{O}) = d_0(s_0) \Big[\sum_{o \in \mathcal{O}} \pi(s_0, o) \mu_o(s_0, a_0) f(h_{|h|}, o, 1) \Big] \prod_{k=0}^{|h|-1} P(s_k, a_k, s_{k+1}),$$

where f is a recursive function defined as:

$$f(h_t, o, i) = \begin{cases} 1, & \textit{if } i = t \\ \Big[\Big(\beta_o(s_i) \sum_{o' \in \mathcal{O}} \pi(s_{i+1}, o') \mu_{o'}(s_{i+1}, a_{i+1}) f(h_t, o', i+1) \Big) & \\ + \Big((1 - \beta_o(s_i)) \mu_o(s_{i+1}, a_{i+1}) f(h_t, o, i+1) \Big) \Big] & \textit{otherwise} \end{cases}$$

Proof. See Appendix C. □

[2] When the transition function is not known, we can estimate it from data.



(a) Visualization of loss over 200 training epochs for the four rooms domain. The decreasing average number of decisions made by π is shown in blue and the increasing probability of generating the sampled trajectories is shown in red.

(b) Performance comparison on four rooms domain. Six tasks were used for training and 24 different for testing. The plot shows the average return (and standard error) on the y-axis as a function of the episode number on the test tasks.

Figure 1: Results on four-room domain. Six tasks were used for training and 24 for testing.

Given a parametric representation of the option policies and termination functions for each $o \in \mathcal{O}$ and for the policy π over options, we use Theorems 1 and 2 to differentiate the objective in equation 2 with respect to their parameters and optimize with any numerical optimization technique.

3.3 Learning Options Incrementally

One common issue in option discovery is identifying how many options are needed for a given problem. Oftentimes this number is predefined by the user based on intuition. In such a scenario, one could learn options by simply randomly initializing the parameters of a number of options and optimizing the proposed objective in equation 2. Instead, we propose not only learning options, but also the number of options needed, by the procedure shown in Algorithm 1. This algorithm introduces one option at a time and optimizes the objective $\hat{J}$ with respect to the policy over options π_θ, with parameters θ, and the newly introduced option, $o' = (\mu'_\phi, \beta'_\psi)$, with parameters ϕ and ψ, for N epochs. Optimizing both o' and π_θ allows us to estimate how much we can improve $\hat{J}$ given that we keep any previously introduced option fixed. After the new option is trained, we measure how much $\hat{J}$ has improved; if it fails to improve above some threshold, Δ, the procedure terminates. This results in a natural way of obtaining an appropriate number of options, as options stop being added once a new option no longer improves the ability to represent the demonstrated behavior.

Algorithm 1 Option Learning Framework - Pseudocode

Collect set of trajectories $\mathcal{H}$
Initialize option set $\mathcal{O}$ with primitive options
done = false
$\hat{J}_{prev} = -\infty$
while done == false **do**
    Initialize new option $o' = (\mu'_\phi, \beta'_\psi)$, initializing parameters for ϕ and ψ.
    $\mathcal{O}' = \mathcal{O} \cup o'$
    Initialize parameters θ of policy π_θ
    for k=1,...,N **do**
        $\hat{J}_k = \hat{J}(\pi_\theta, \mathcal{O}', \mathcal{H})$
        $\theta = \theta + \alpha \frac{\partial \hat{J}_k}{\partial \theta}$
        $\phi = \phi + \alpha \frac{\partial \hat{J}_k}{\partial \phi}$
        $\psi = \psi + \alpha \frac{\partial \hat{J}_k}{\partial \psi}$
    if $\hat{J}_N - \hat{J}_{prev} < \Delta$ **then**
        done = true
    else
        $\mathcal{O} = \mathcal{O}'$
        $\hat{J}_{prev} = \hat{J}_N$
Return new option set $\mathcal{O}$

4 Experimental Results

This section describes experiments used to evaluate the proposed technique approach. We show results in the "four rooms" domain to allow us to visualize and understand the options produced by our method, and to show empirically that these options produce a clear improvement in learning. We use this domain to show that options generated by our method are able to generalize to tasks

where the option-critic architecture (Bacon et al., 2017) and eigenoptions (Machado et al., 2017) would fail to do so. We then extend our experiments to evaluate our technique in a few selected problems from the Atari 2600 emulator provided by OpenAI Gym (Brockman et al., 2016). These experiments demonstrate that by using the trajectories obtained from solving a small subset of tasks, our approach is able to discover options that significantly improve the learning ability of the agent in the tasks it has yet to solve.

4.1 Experiments on Four Rooms Environment

We tested our approach in the four rooms domain: a gridworld of size 40×40, in which the agent is placed in a start state and needs to reach a goal state. At each time-step, the agent executes one of four available actions: moving left, right, up or down, and receives a reward of -1. Upon reaching the goal state, the agent receives a reward of $+10$. We generated 30 different task variations (by changing the goal, start locations, and transition functions) and collected six sample trajectories from optimal policies learned in six tasks. We evaluated our method on the remaining 24 tasks.

Figure 1a shows the change in the average expected number of terminations and average probability of generating the observed trajectories while learning options, as new options are introduced and adapted to the sampled trajectories. Options were learned over the six sampled optimal trajectories and every 50 epochs a new option was introduced. For every new option, the change in probability of generating the observed trajectories as well as the change in expected number of decisions reaches a plateau after 30 or 40 training epochs. When a new option is introduced, there is a large jump in the loss because a new policy, π, is initialized arbitrarily to account for the new option set being evaluated. However, after training the new candidate option, the overall loss improves beyond what it was possible before introducing the new option.

In Figure 1b, we compare the performance of Q-learning on 24 novel test tasks using options discovered by our method (with and without regularization using KL divergence), eigenoptions, and option critic. We allowed each competing method to learn options from the same six training tasks and, to ensure a fair comparison, we used the original code provided by the authors. As baselines, we also compare against primitive actions and randomly initialized options. It might seem surprising that both eigenoptions and the option-critic failed to reach an optimal policy when they were shown to work well in this type of problem; for that we offer the following explanation. Our implementation of four rooms is defined in a much larger state space than the ones where these methods were originally tested, making each individual room much larger. Since the options identified by these methods tend to lead the agent from room to room, it is possible that, once in the correct room, the agent executes an option leading to a different room before it had the opportunity to find the goal. When testing our approach in the smaller version of the four room problem, we found no clear difference in performance of the competing methods. In this experiment, we set the threshold Δ for introducing a new option to 10% of $\hat{J}$ at the previous iteration and the hyperparameter $\lambda_2 = 100.0$. When adding KL regularization, we set $\lambda_1 = 0.001$.

Figure 2 shows a visualization of the policy learned by the agent for a specific task. The policy leads the agent to navigate from a specific location in the bottom-left room to a location in the top-right room in a small "four-room" domain of size 10×15. [3] The new task to solve is shown in the top-left figure, while the solution found is shown in the top-right figure. Each of the remaining rows of images shows how each option was learned and used in the new task. The first row shows how options are initialized before training, the middle shows the learned options after training, and the bottom row shows a heat-map depicting where each option is likely to be called. The figure shows that while the options are defined over the entire state space, they are only useful in specific regions—that is, they are specialized. These options, when used in combination in specific regions, allow the agent to learn how to solve new problems more efficiently.

4.2 Experiments using Atari 2600 Games

We evaluated the quality of the options learned by our framework in two different Atari 2600 games: Breakout and Amidar. We trained the policy over options using A3C (Mnih et al., 2016) with grayscale pixel input. Options were represented by a two layer convolutional neural network, and were given the previous two frames as input. The full implementation details for these experiments are given in Appendix E. Figures 3a and 3b show the performance of the agent as a function of

[3] We show a smaller domain than used in the experiments for ease of visualization

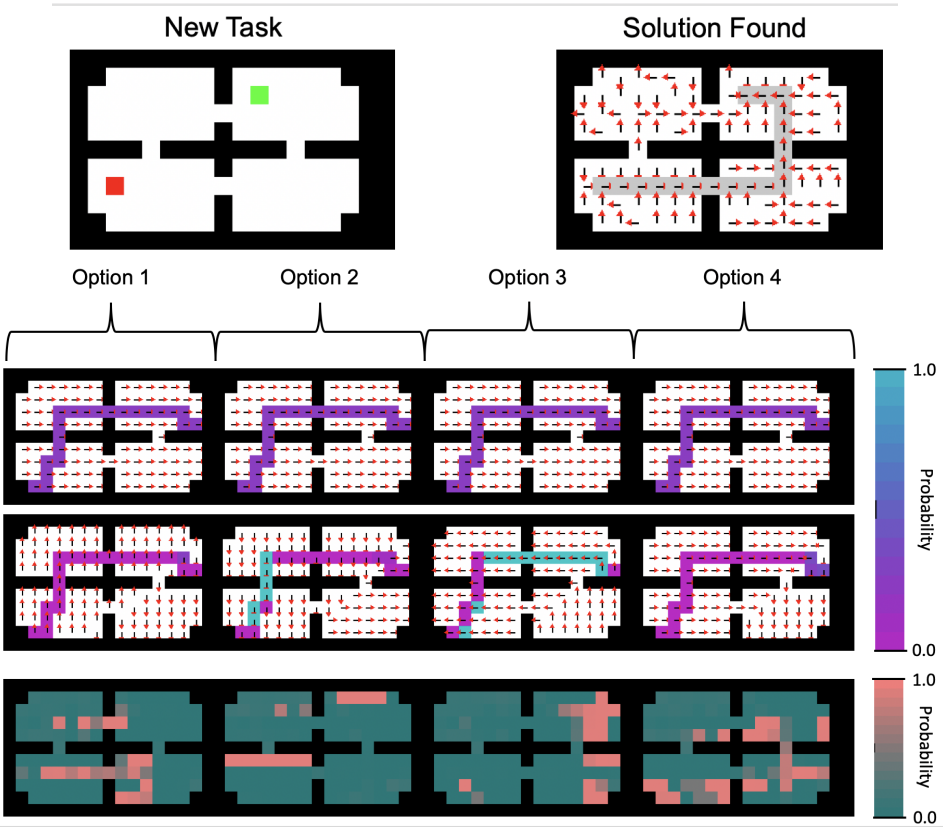
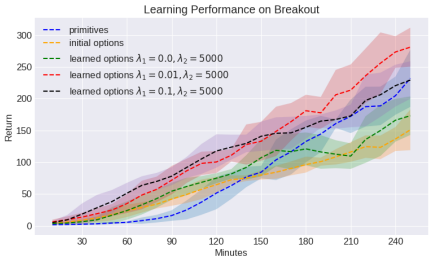


Figure 2: Visualization of our framework in four rooms domain. A novel task is seen in the top left, where the agent (red) has to navigate to a goal (green). On the top right, we show the solution found by the agent. The three rows below show how the options were learned and exploited in the new task. The highlighted area in the top two rows show a sample trajectory and the color corresponds to the probability that the option would take the demonstrated action. Notice that this trajectory was obtained on to a previous tasks, so it does not correspond to the new task on top. The arrows show the action that is most likely at each state. Before training (first row), each option is randomly initialized, but after training (second row) each option specializes in a specific skill (a navigation pattern). In this case, the demonstrated trajectory can be generated by using option 3 and 2. The last row shows a heat-map indicating where each options is likely to be used by the policy learned in the new task. The agent learns to use each option in very specific situations; for example, option 1 is likely to be called to make the agent move up, if it is located in one of the bottom rooms.

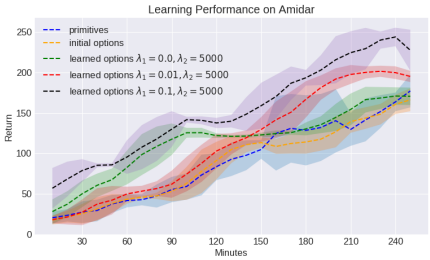
training time in Breakout and Amidar, respectively. [4] The plots show that given good choices of hyperparameters, the learned options led to a clear improvement in performance during training. For both domains, we found that $\lambda_2 = 5,000$ led to a reasonable trade-off between the first two term in $\hat{J}$, and report results with three different regularization values: $\lambda_1 = 0.0$,, $\lambda_1 = 0.01$ and $\lambda_1 = 0.1$.

Note that our results do not necessarily show that the options result in a better final policy, but they improve exploration early in training and enable the agent to learn more effectively. Figure 4a depicts the behavior for one of the learned options on Breakout. The option efficiently catches the ball after it bounces off the left wall, and then terminates with high probability before the ball has to be caught again. Bear in mind that the option remains active for many time-steps, significantly reducing the number of decisions made by the policy over options. However, it does not maintain control for so long that the agent is unable to respond to changing circumstances. Note that the option is only useful in specific case; for example, it was not helpful in returning a ball bounced off the right wall. That is to say, the option specialized in a specific sub-task within the larger problem: a highly desirable property for generally useful options.

[4] We chose training time as the independent variable to consider as it can be more easily related to costs in training (e.g., cost of computation time) than other standard metrics like the number of episodes an agent trains.



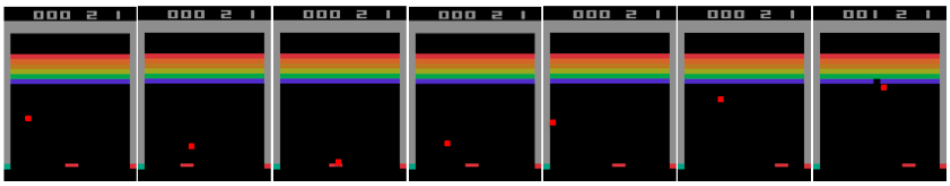
(a) Average returns on novel tasks for Breakout.



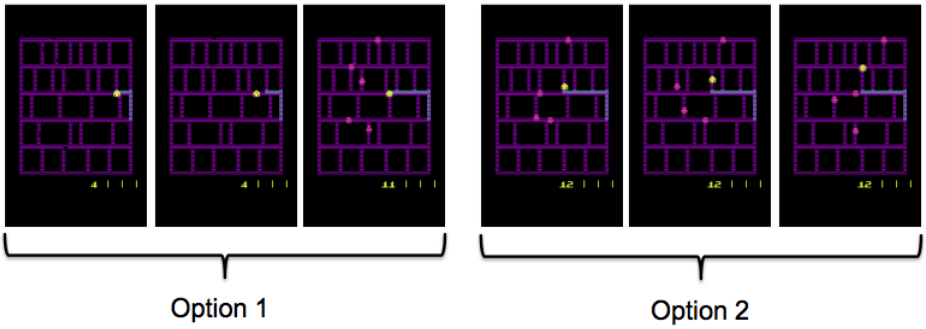
(b) Average returns on novel tasks for Amidar.

Figure 3: Comparison on Atari domains for primitives (blue), options before training (orange) and learned options for different values of λ_1 and λ_2. Shaded regions indicate standard error.

Figure 4b shows the selection of two of the options learned for Amidar when starting a new game. At the beginning of the game, option 1 is selected, which takes the agent to a specific intersection before terminating. The agent then selects option 2, which chooses a direction at the intersection, follows the resulting path, and terminates at the next intersection. Note that the agent does not need to repeatedly select primitive actions in order to simply follow a previously chosen path. Having access to these types of options enables an agent to easily replicate known good behaviors, allowing for faster and more meaningful exploration of the state space.



(a) Visualization of a learned option executed until termination on Breakout. The option learned to catch the ball bouncing off the left wall and terminates with high probability before the ball bounces a wall again (ball size increased for visualization).



(b) Visualization of two learned options on Amidar. The agent is shown in yellow and enemies in pink. Option 1 learned to move up, at the beginning of the game, and turn left until reaching an intersection. Option 2 learned to turn in that intersection and move up until reaching the next one.

5 Conclusion and Future Work

In this work we presented an optimization objective for learning options offline from demonstrations of near-optimal behavior on a set of tasks. Optimizing the objective results in a set of options that allows an agent to reproduce the behavior while minimizing the number of decisions made by the policy over options, which are able to improve the learning ability of the agent on new tasks. We provided results showing how options adapt to the trajectories provided and showed, through several experiments, that the identified options are capable of significantly improving the learning ability of an agent. The resulting options encode meaningful abstractions that help the agent interact with and learn from its environment more efficiently.

REFERENCES

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E. Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 2504–2510. AAAI Press, 2015. ISBN 0-262-51129-0. URL http://dl.acm.org/citation.cfm?id=2886521.2886669.

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, 2016.

Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proc. of 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/finn17a.html.

Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. In *AAAI*, 2018.

Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic. In *AISTAT*, 2019.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian Framework for Option Discovery in Reinforcement Learning. *CoRR*, 2017.

Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*, pp. 553–560. ACM, 2005.

A. McGovern and R. Sutton. Macro actions in reinforcement learning: An empirical analysis. Technical report, University of Massachusetts - Amherst, Massachusetts, USA, 1998.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/mniha16.html.

Jürgen Schmidhuber, Jieyu Zhao, and Nicol N. Schraudolph. Learning to learn. chapter Reinforcement Learning with Self-modifying Policies, pp. 293–309. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0-7923-8047-9. URL http://dl.acm.org/citation.cfm?id=296635.296658.

Jürgen Schmidhuber. On learning how to learn learning strategies. Technical report, 1995.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Richard S. Sutton and Doina Precup. Intra-option learning about temporally abstract actions. In *In Proceedings of the 15th International Conference on Machine Learning (ICML-1998)*, 1998.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1577069.1755839.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *J. Mach. Learn. Res.*, 8:2125–2167, December 2007. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1314498.1314569.

Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995. ISSN 0001-0782. doi: 10.1145/203330.203343. URL http://doi.acm.org/10.1145/203330.203343.

A APPENDIX

The following list defines the notation used in all derivations:

1. A_t: random variable denoting action taken at step t.
2. S_t: random variable denoting state at step t.
3. H_t: random variable denoting history up to step t. $H_t = (S_0, A_0, S_1, A_1, \ldots, S_t)$.
4. T_t: random variable denoting the event that the option used at step $t-1$ terminates at state S_t.
5. π: policy over options.
6. P: transition function. $P(s, a, s')$ denotes the probability of transitioning to state s' by taking action a in state s
7. O_t: random variable denoting the option selected for execution at state S_t.
8. o: option defined as $o = (\mu_o, \beta_o)$, where μ_o is the option policy for option and β_o is the termination function.
9. Assume primitives are options that perform only 1 action and last for 1 time-step.
10. $\mathcal{O}$: set of available options.

We can compute the probability of an option terminating at state s_t and generating a trajectory h_t as:

$$\Pr(T_t = 1, H_t = h_t|\pi, \mathcal{O}) = \Pr(T_t = 1|H_t = h_t, \pi, \mathcal{O}) \Pr(H_t = h_t|\pi, \mathcal{O}) \tag{4}$$

To compute the proposed objective J we need to find an expression for $\Pr(T_t = 1|H_t = h_t, \pi, \mathcal{O})$ and $\Pr(H_t = h_t|\pi, \mathcal{O})$ in terms of known quantities.

A.1 APPENDIX A - DERIVATION OF $\hat{J}$

Recall $J(\pi, \mathcal{O}, H) = \mathbf{E}\left[\sum_{t=1}^{|h|} \Pr(T_t = 0, H_t|\pi, \mathcal{O})\right]$, ignoring the regularization term. Assuming access to a set $\mathcal{H}$ of sample trajectories, we start by estimating J from sample averages and derive the objective $\hat{J}$ as follows:

$$\begin{aligned}
J(\pi, \mathcal{O}, \mathcal{H}) \approx & \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \sum_{t=1}^{|h|} \Pr(T_t = 0, H_t = h_t|\pi, \mathcal{O}) \\
= & \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \sum_{t=1}^{|h|} \big(1 - \Pr(T_t = 1|H_t = h_t, \pi, \mathcal{O})\big) \Pr(H_t = h_t|\pi, \mathcal{O}) \\
= & \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \sum_{t=1}^{|h|} \Big(1 - \mathbf{E}\big[T_t|H_t = h_t, \pi, \mathcal{O}\big]\Big) \Pr(H_t = h_t|\pi, \mathcal{O})
\end{aligned}$$

It can easily be seen that to maximize the above expression $\mathbf{E}\big[T_t|H_t = h_t, \pi, \mathcal{O}\big]$ should be minimized while $\Pr(H = h|\pi, \mathcal{O})$ should be maximized. Given that for long trajectories the expected number of terminations increases while the probability of generating the trajectories goes to 0, we normalize the number of terminations by the lenght of the trajectory, $|h|$, and adjust a hyperparameter, λ_2, to prevent one term from dominating the other during optimization. Based on this observation we propose optimizing the following objective:

$$\hat{J}(\pi, \mathcal{O}, \mathcal{H}) = \frac{1}{\mathcal{H}} \sum_{h_{|h|} \in \mathcal{H}} \lambda_2 \Pr(H = h|\pi, \mathcal{O}) - \frac{\sum_{t=1}^{|h|} \mathbf{E}\left[T_t|H_t = h_t, \pi, \mathcal{O}\right]}{|h|}.$$

This objective allow us to control a trade-off, through λ_2, of how much we care about the options reproducing the demonstrated trajectories vs. how much we want the agent to minimize the number of decisions.

A.2 APPENDIX B - PROOF OF THEOREM 1

Theorem 1 *Given a set of options $\mathcal{O}$ and a policy π over options, the expected number of terminations for a trajectory h of length $|h|$ is given by:*

$$\sum_{t=1}^{|h|} \mathbf{E}\left[T_t = 1 | H_t = h_t, \pi, \mathcal{O}\right] = \sum_{t=1}^{|h|} \left(\sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\sum_{o' \in \mathcal{O}} \mu_{o'}(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O})} \right),$$

where,

$$\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) = \Bigg[\Big(\pi(s_{t-1}, o) \beta_o(s_{t-1}) \Big) \Big(P(s_{t-2}, a_{t-2}, s_{t-1}) \mu_o(s_{t-2}, a_{t-2}) \times \Pr(O_{t-2} = o | H_{t-2} = h_{t-2}, \pi, \mathcal{O})(1 - \beta_o(s_{t-1})) \Big) \Bigg],$$

and $\Pr(O_0 = o | H_0 = h_0, \pi, \mathcal{O}) = \pi(s_0, o)$.

Proof. Notice that $\sum_{t=1}^{|h|} \mathbf{E}\left[T_t = 1 | H_t = h_t, \pi, \mathcal{O}\right] = \sum_{t=1}^{|h|} \Pr(T_t = 1 | H_t = h_t, \pi, \mathcal{O})\, 1$, so if we find an expression for $\Pr(T_t = 1 | H_t = h_t, \pi, \mathcal{O})$, we can calculate the expectation exactly. We define $\Pr(T_0 = 1 | H_1 = h_1, \pi, \mathcal{O}) = 1$ for ease of derivation even though there is no option to terminate at T_0.

$$\begin{aligned}
\Pr(T_t = 1 | H_t = h_t, \pi, \mathcal{O}) &= \sum_{o \in \mathcal{O}} \Pr(T_t = 1 | O_{t-1} = o, H_t = h_t, \pi, \mathcal{O}) \Pr(O_{t-1} = o | H_t = h_t, \pi, \mathcal{O}) \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \Pr(O_{t-1} = o | H_t = h_t, \pi, \mathcal{O}) \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, S_t = s_t, \pi, \mathcal{O}) \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\Pr(S_t = s_t | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, O_{t-1} = o, \pi, \mathcal{O})}{\Pr(S_t = s_t | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, \pi, \mathcal{O})} \\
&\quad \times \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, \pi, \mathcal{O}) \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\Pr(S_t = s_t | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, \pi, \mathcal{O})}{\Pr(S_t = s_t | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, \pi, \mathcal{O})} \\
&\quad \times \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, \pi, \mathcal{O}) \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, \pi, \mathcal{O}) \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\Pr(A_{t-1} = a_{t-1} | H_{t-1} = h_{t-1}, O_{t-1} = o, \pi, \mathcal{O}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\Pr(A_{t-1} = a_{t-1} | H_{t-1} = h_{t-1}, \pi, \mathcal{O})} \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\Pr(A_{t-1} = a_{t-1} | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\sum_{o' \in \mathcal{O}} \Pr(A_{t-1} = a_{t-1}, O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O})} \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
&\quad \times \big(\sum_{o' \in \mathcal{O}} \Pr(A_{t-1} = a_{t-1} | O_{t-1} = o', H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
&\quad \times \Pr(O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \big)^{-1} \\
&= \sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\sum_{o' \in \mathcal{O}} \mu_{o'}(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}
\end{aligned}
$$

We are left with finding an expression in terms of known probabilities for $\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})$.

$$
\begin{aligned}
\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) =& \big[\Pr(O_{t-1} = o, T_{t-1} = 1 | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
&+ \Pr(O_{t-1} = o, T_{t-1} = 0 | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \big] \\
=& \Big[\big(\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, T_{t-1} = 1, \pi, \mathcal{O}) \\
&\times \Pr(T_{t-1} = 1 | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \big) \\
&+ \big(\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, T_{t-1} = 0, \pi, \mathcal{O}) \\
&\times (1 - \Pr(T_{t-1} = 1 | H_{t-1} = h_{t-1}, \pi, \mathcal{O})) \big) \Big] \\
=& \Big[\big(\pi(s_{t-1}, o) \Pr(T_{t-1} = 1 | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \big) \\
&+ \big(\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, T_{t-1} = 0, \pi, \mathcal{O}) \\
&\times (1 - \Pr(T_{t-1} = 1 | H_{t-1} = h_{t-1}, \pi, \mathcal{O})) \big) \Big] \\
=& \Big[\big(\pi(s_{t-1}, o) \beta_o(s_{t-1}) \big) + \\
&\times \big(\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, T_{t-1} = 0, \pi, \mathcal{O})(1 - \beta_o(s_{t-1})) \big) \Big]
\end{aligned}
$$

Given that by convention, $\Pr(T_0 = 1 | H_0 = h_0, \pi, \mathcal{O}) = 1.0$, we are now left with figuring out how to calculate $\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, T_{t-1} = 0, \pi, \mathcal{O})$

$$
\begin{aligned}
\Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, T_{t-1} = 0, \pi, \mathcal{O}) =& \Pr(O_{t-2} = o, A_{t-2} = a_{t-2}, S_{t-1} = s_{t-1} | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
=& \Pr(A_{t-2} = a_{t-2}, S_{t-1} = s_{t-1} | O_{t-2} = o, H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
&\times \Pr(O_{t-2} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
=& \Pr(S_{t-1} = s_{t-1} | A_{t-2} = a_{t-2}, O_{t-2} = o, H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
&\times \Pr(A_{t-2} = a_{t-2} | O_{t-2} = o, H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
&\times \Pr(O_{t-2} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
=& P(s_{t-2}, a_{t-2}, s_{t-1}) \mu_o(s_{t-2}, a_{t-2}) \Pr(O_{t-2} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O}) \\
=& P(s_{t-2}, a_{t-2}, s_{t-1}) \mu_o(s_{t-2}, a_{t-2}) \Pr(O_{t-2} = o | H_{t-2} = h_{t-2}, \pi, \mathcal{O})
\end{aligned}
$$

where $\Pr(O_0 = o | H_0 = h_0, \pi, \mathcal{O}) = \pi(s_0, o)$

Using the recursive function $\Pr(O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O})$, the expected number of terminations for a given trajectory is given by:

$$\sum_{t=1}^{|h|} \mathbf{E}\left[T_t = 1 | H_t = h_t, \pi, \mathcal{O}\right] = \sum_{t=1}^{|h|} \left(\sum_{o \in \mathcal{O}} \beta_o(s_t) \frac{\mu_o(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o | H_{t-1} = h_{t-1}, \pi, \mathcal{O})}{\sum_{o' \in \mathcal{O}} \mu_{o'}(s_{t-1}, a_{t-1}) \Pr(O_{t-1} = o' | H_{t-1} = h_{t-1}, \pi, \mathcal{O})} \right),$$

□

A.3 Appendix C - Proof of Theorem 2

Theorem 2

Given a set of options $\mathcal{O}$ and a policy π over options, the probability of generating a trajectory h of length $|h|$ is given by:

$$\Pr(H_{|h|} = h_{|h|} | \pi, \mathcal{O}) = d_0(s_0) \left[\sum_{o \in \mathcal{O}} \pi(s_0, o) \mu_o(s_0, a_0) f(h_{|h|}, o, 1) \right] \prod_{k=0}^{|h|-1} P(s_k, a_k, s_{k+1}),$$

where f is a recursive function defined as:

$$f(h_t, o, i) = \begin{cases} 1, & \text{if } i = t \\ \Big[\beta_o(s_i) \sum_{o' \in \mathcal{O}} \pi(s_{i+1}, o') \mu_{o'}(s_{i+1}, a_{i+1}) f(h_t, o', i+1) & \\ + (1 - \beta_o(s_i)) \mu_o(s_{i+1}, a_{i+1}) f(h_t, o, i+1) \Big], & \text{otherwise} \end{cases}$$

Proof. We define $H_{i,t}$ to be the history from time i to time t, that is, $H_{i,t} = (S_i, A_i, S_{i+1}, A_{i+1}, \ldots, S_t)$, where $i < t$. If $i = t$, the history would contain a single state.

$$\begin{aligned}
\Pr(H_t = h_t | \pi, \mathcal{O}) =& \Pr(S_0 = s_0 | \pi, \mathcal{O}) \Pr(H_{1,t} = h_{1,t}, A_0 = a_0 | S_0 = s_0, \pi, \mathcal{O}) \\
=& d_0(s_0) \Pr(H_{1,t} = h_{1,t}, A_0 = a_0 | S_0 = s_0, \pi, \mathcal{O}) \\
=& d_0(s_0) \sum_{o \in \mathcal{O}} \Pr(H_{1,t} = h_{1,t}, A_0 = a_0, O_o = o | S_0 = s_0, \pi, \mathcal{O}) \\
=& d_0(s_0) \sum_{o \in \mathcal{O}} \Pr(O_0 = o | S_0 = s_0, \pi, \mathcal{O}) \Pr(H_{1,t} = h_{1,t}, A_0 = a_0 | S_0 = s_0, O_0 = o, \pi, \mathcal{O}) \\
=& d_0(s_0) \sum_{o \in \mathcal{O}} \pi_(s_0, o) \Pr(H_{1,t} = h_{1,t}, A_0 = a_0 | S_0 = s_0, O_0 = o, \pi, \mathcal{O}) \\
=& d_0(s_0) \sum_{o \in \mathcal{O}} \pi_(s_0, o) \Pr(A_0 = a_o | S_0 = s_0, O_0 = o, \pi, \mathcal{O}) \\
& \times \Pr(H_{1,t} = h_{1,t} | S_0 = s_0, O_0 = o, A_0 = a_0, \pi, \mathcal{O}) \\
=& d_0(s_0) \sum_{o \in \mathcal{O}} \pi_(s_0, o) \mu_o(s_0, a_o) \Pr(H_{1,t} = h_{1,t} | S_0 = s_0, O_0 = o, A_0 = a_0, \pi, \mathcal{O}).
\end{aligned}$$

We now need to find an expression to calculate $\Pr(H_{1,t} = h_{1,t} | S_0 = s_0, O_0 = o, A_0 = a_0, \pi, \mathcal{O})$. Consider the probability of seeing history $h_{i,t}$ given the previous state, s, the previous option, o, and the previous action, a:

$$
\begin{aligned}
&\Pr(H_{i,t} = h_{i,t} | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a) \\
=&\Pr(S_i = s_i | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a) \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a, S_i = s_i) \\
=&P(s, a, s_i) \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a, S_i = s_i) \\
=&P(s, a, s_i) \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i) \\
=&P(s, a, s_i) \big[\Pr(T_i = 1 | O_{i-1} = o, A_{i-1} = a, S_i = s_i) \\
&\times \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 1) \\
&+ \Pr(T_i = 0 | O_{i-1} = o, A_{i-1} = a, S_i = s_i) \\
&\times \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 0) \big] \\
=&P(s, a, s_i) \big[\beta_o(s_i) \\
&\times \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 1) \\
&+ (1 - \beta_o(s_i)) \\
&\times \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 0) \big].
\end{aligned}
$$

Even though the equation above might seem complicated, there are only two cases we need to consider: either the current option terminates and a new one must be selected (the first term), or the current option does not terminate (the second term). Let's consider each of them separately.

Case 1 - option terminates: If we terminate, we sum over new options:

$$
\begin{aligned}
&\Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 1) \\
=& \sum_{o' \in \mathcal{O}} \Pr(O_i = o' | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 1) \\
&\times \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 1, O_i = o') \\
=& \sum_{o' \in \mathcal{O}} \pi(s_i, o') \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 1, O_i = o') \\
=& \sum_{o' \in \mathcal{O}} \pi(s_i, o') \Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | S_i = s_i, O_i = o') \\
=& \sum_{o' \in \mathcal{O}} \pi(s_i, o') \Pr(A_i = a_i | S_i = s_i, O_i = o') \Pr(H_{i+1,t} = h_{i+1,t} | S_i = s_i, O_i = o', A_i = a_i) \\
=& \sum_{o' \in \mathcal{O}} \pi(s_i, o') \mu_{o'}(s_i, a_i) \Pr(H_{i+1,t} = h_{i+1,t} | S_i = s_i, O_i = o', A_i = a_i).
\end{aligned}
$$

Note that the expanded probability has the same form as $\Pr(H_{i,t} = h_{i,t} | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a)$.

Case 2 - option does not terminate: This tells us that $O_i = o$, so we may drop the dependency on the $i - 1$ terms:

$$
\begin{aligned}
&\Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a, S_i = s_i, T_i = 0) \\
=&\Pr(H_{i+1,t} = h_{i+1,t}, A_i = a_i | S_i = s_i, O_i = o) \\
=&\Pr(A_i = a_i | S_i = s_i, O_i = 0) \Pr(H_{i+1,t} = h_{i+1,t} | S_i = s_i, O_i = o, A_i = a_i) \\
=&\mu_o(s_i, a_i) \Pr(H_{i+1,t} = h_{i+1,t} | S_i = s_i, O_i = o, A_i = a_i).
\end{aligned}
$$

Plugging these two cases back into our earlier equation yields:

$$
\begin{aligned}
&\Pr(H_{i,t} = h_{i,t} | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a) \\
=&P(s,a,s_i)\big[\beta_o(s_i) \sum_{o' \in \mathcal{O}} \pi(s_i, o') \mu_{o'}(s_i, a_i) \Pr(H_{i+1,t} = h_{i+1,t} | S_i = s_i, O_i = o', A_i = a_i) \\
&+ (1 - \beta_o(s_i)) \mu_o(s_i, a_i) \Pr(H_{i+1,t} = h_{i+1,t} | S_i = s_i, O_i = o, A_i = a_i)\big].
\end{aligned}
$$

Note that each term contains an expression of the same form, $\Pr(H_{i,t} = h_{i,t} | S_{i-1} = s, O_{i-1} = o, A_{i-1} = a)$. We can therefore compute the probability recursively. Our recursion will terminate when we consider $i = t$, as $H_{t,t}$ contains a single state, and we adopt the convention of its probability to be 1. Notice that for every recursive step, both inner terms will produce a $P(s, a, s_i)$ term. Consider the result when we factor every recursive $P(s, a, s_i)$ term to the front of the equation. We define the following recursive function:

$$
f(h_t, o, i) = \begin{cases} 1, & \text{if } i = t \\ \Big[\beta_o(s_i) \sum_{o' \in \mathcal{O}} \pi(s_{i+1}, o') \mu_{o'}(s_{i+1}, a_{i+1}) f(h_t, o', i+1) & \\ +(1 - \beta_o(s_i)) \mu_o(s_{i+1}, a_{i+1}) f(h_t, o, i+1)\Big], & \text{otherwise} \end{cases}
$$

.

Notice that this is the recursive probability described above, but with the $P(s, a, s')$ terms factored out. We now see that:

$$
\Pr(H_{i,t} = h_{i,t} | S_{i-1} = s_{i-1}, O_{i-1} = o, A_{i-1} = a_{i-1}) = f(h_t, o, i) \prod_{k=i-1}^{t-1} P(s_k, a_k, s_{k+1}).
$$

Plugging this all the back into our original equation for $\Pr(H_t = h_t | \pi, \mathcal{O})$ gives us the desired result:

$$
\Pr(H_{|h|} = h_{|h|} | \pi, \mathcal{O}) = d_0(s_0) \Big[\sum_{o \in \mathcal{O}} \pi(s_0, o) \mu_o(s_0, a_o) f(h_{|h|}, o, 1) \Big] \prod_{k=0}^{t-1} P(s_k, a_k, s_{k+1}).
$$

□

A.4 Appendix D - Empirical Validation of Derived Equations

To double check the derivation of the proposed objective and make sure the implementation was correct, we conducted a simple empirical test to compared the calculated expected number of decisions in a trajectory and the probability of generating each trajectory for a set of 10 trajectories on 10 MDPs. The MDPs are simple chains of 7 states with different transition functions. We randomly initialized four options and a policy over options, and estimated the probability of generating each trajectory and the expected number of terminations, for each sampled trajectory, by Montecarlo sampling $10,000$ trials. Table 1 presents results for the 10 trajectories verifying empirically that the equations were correctly derived and implemented. The table compares the empirical and true probability of generating a given trajectory, $\hat{\Pr}(H|\cdot)$ and $\Pr(H|\cdot)$, respectively, and the empirical and true sum of expected number of decisions an agent has to make to generate those trajectories, $\sum_{t=1}^{|H|} \hat{\mathbf{E}}\left[T_t|\cdot\right]$ and $\sum_{t=1}^{|H|} \mathbf{E}\left[T_t|\cdot\right]$, respectively.

H	$\hat{\Pr}(H\|\pi,\mathcal{O})$	$\Pr(H\|\pi,\mathcal{O})$	$\sum_{t=1}^{\|H\|}\hat{\mathbf{E}}\left[T_t\|H_t,\pi,\mathcal{O}\right]$	$\sum_{t=1}^{\|H\|}\mathbf{E}\left[T_t\|H_t,\pi,\mathcal{O}\right]$
h_1	0.0932	0.0957	3.060	3.178
h_2	0.0158	0.0173	4.139	4.154
h_3	0.2149	0.2122	1.965	2.178
h_4	0.0995	0.0957	2.979	3.178
h_5	0.0962	0.0957	3.024	3.178
h_6	0.1354	0.1384	2.9579	3.1596
h_7	0.00040	0.00038	9.750	8.794
h_8	0.1854	0.1881	2.820	3.072
h_9	0.0379	0.0368	4.2612	4.4790
h_{10}	0.1864	0.1881	2.8404	3.0723

Table 1: Validation of equations and implementation.

Note that the cases with largest discrepancy between the estimated and calculated number of terminations occur when the probability of generating a trajectory is low. This happens because, since the trajectory is unlikely to be generated, the Monte Carlo sampling is not able to produce enough samples of the trajectory.

A.5 Appendix E - Implementation Details for Atari Experiments

For these experiments we first learned a good performing policy with A3C for each game and sampled 12 trajectories for training. Each trajectory lasted until a life was lost, not for the entire duration of the episode. Each option was represented as a two-layer neural network, with 32 neurons in each layer, and two output layers: a softmax output layer over the four possible actions representing μ, and a separate sigmoid layer representing β. We implemented our objective using PyTorch which simplifies gradient calculations. The options were represented by a two-layer neural network, where the input was represented by gray scale images of the last two frames. We ran 32 training agents in parallel on CPUs, the learning rate was set to 0.0001 and the discount factor γ was set to 0.99.

Because the options can only learn from the states observed in the trajectories, it is possible that when using them, they will be executed in previously unseen states. When this happens, the termination function may decide to never terminate, as it has not seen that region of the state space before. To address this issue, we add a value of 0.05 to the predicted probability of termination per time-step that the option has been running since executed. Therefore, in our experiments an option cannot run for more than 20 time-steps in total.