Experience-based Exploration for Reinforcement Learning

Abstract

In Reinforcement Learning, an agent may have to face several tasks pertaining to a specific class of problem, or domain. For example, when in a simple navigation problem, an agent may have to learn to navigate several different mazes. Even though all of these may be distinct, they obey the same rules of navigation set forth by the domain. Once an agent has learned how to solve a few of these problems, it is possible to leverage the acquired knowledge to facilitate solving novel problems from the same domain. This work is motivated by the observation that the optimal policies to several tasks taken from a single domain will tend to reveal an underlying pattern for solving novel problems from the same domain. In this paper, we develop a data-driven approach for extracting options and present an exploration strategy that takes advantage of those options. We show meaningful patterns being extracted for discrete and continuous action spaces, and present evidence demonstrating how our approach uses previous knowledge to improve performance.

1. Introduction

Reinforcement Learning (RL) is an active area of research in the machine learning community concerned with the problem of an agent learning from interaction with its environment. In this framework, an agent is at a state s_t at time-step t, takes an action a_t, receives a reward r_t and moves to a state s_{t+1}. A policy, π, determines the behavior of an agent with its environment. A sample from this behavior policy from some initial state s_0 to some terminal state s_k is referred to as a trajectory, and determines a sequence of decisions.

In the last few years, we have seen some impressive results of what this learning framework is capable of achieving (Mnih et al., 2013; Lillicrap et al., 2015). None of these approaches, however, are able to escape a crucial difficulty of RL: balancing the trade-off between exploration and exploitation. Exploration is concerned with taking actions that gather information about the underlying problem, and exploitation deals with making decision based on the gained knowledge. Exploration strategies can have a huge impact on how quickly a learning algorithm can find a solution; many commonly used strategies, such as ϵ-*greedy* or Boltzmann, can spend large amounts of time unnecessarily exploring due to a random action selection criterion, or due to inaccurate initial estimates of the value of a state.

In many instances, an agent is asked to learn how to behave in different tasks belonging to a same type of domain with the same set of rules. For example, in the typical toy problem of ”grid-world”, an agent may have to learn to navigate several different environments, all of which are separate problems taken from the grid-world domain, thus, involving a fixed set of rules. In these situations, the common underlying structure can be exploited to guide exploration by leveraging previous experience with the domain. To do so, we take advantage of temporal extended actions, also called ”options” or ”macros”, (Sutton et al., 1999; Precup, 2000; Barto & Mahadevan, 2003).

Whereas primitive actions last for one time-step, taking the agent from state s_t to s_{t+1}, temporal extended action last for $n > 1$ time-steps, taking the agent from s_t to s_{t+n}. By noticing that a sample from a policy is simply a sequence of actions, and a sequence of actions can be represented in a more compact manner through options, we seek to find options that produce compact policy representations. This has the effect of identifying options that occur frequently in policies of the given domain. To that end, we look for patterns in samples from optimal policies π^* to problems pertaining to a specific domain. This leads us to obtain open-loop options, sequences of actions where that do not rely on feedback from the environment, which bias learning efforts towards more promising action sequences, and allow the agent to quickly reach parts of the state space that would otherwise would not be reachable by random exploration. These options define sequences of actions that ”make sense” in the context of the given domain.

In this work, we propose a method for obtaining options from samples taken from policies for continuous and discrete action spaces. Given these options, we introduce a data-driven exploration strategy, agnostic to the learning al-

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

gorithm used, which outperforms traditional methods. The rest of this paper is organized as follows: section 2 introduces background information pertinent to the problem at hand, section 3 formally motivates our work by borrowing ideas from the compression literature, sections 4 and 5 develop our approach for the discrete and continuous action spaces, respectively. Finally 6 demonstrates empirically that this exploration strategy leads to faster learning over different domains.

2. Background

Reinforcement learning (RL)(Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998) addresses the problem of how an autonomous agent can learn to behave by interacting with its environment in a way such that it maximizes the received reward. A large portion of RL research focuses on the Markov Decision Process (MDP) framework, a discrete time stochastic or deterministic control process. In this work, we focus on the deterministic, finite horizon setting.

This framework proceeds as follows: at time-step t, the agent is in state $s_t \in S$, takes action $a_t \in A$ and moves to state $s_{t+1} \in S$. Upon taking action a_t, the environment responds with a reward signal r_t. The goal of the agent is to find a policy $\pi^*(s_t)$ which maximizes the cumulative reward obtained throughout the process. The agent learns the value of a state $V(s)$ or state-action pair $Q(s,a)$ by trading off between exploration (taking actions for the sake of learning about s or (s,a)) and exploitation (following a policy based on the gained knowledge). The most common mechanisms for exploring are ϵ-*greedy*, in which an action is selected randomly with probability ϵ, or Boltzmann, in which an action is selected according to a probability distribution derived from the current estimate of the q-values.

For any policy π, the value $V^\pi(s)$ denotes the expected discounted return of following π given that the agent is at state s. This is defined as:

$$V^\pi(s) = E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = s, \pi\} \quad (1)$$

where $0 \leq \gamma \leq 1$ is a discount factor.

While the above equations looks at the value of a state, it is also possible to look at the value of a state-action pair $Q^\pi(s,a)$, defined as:

$$\begin{aligned} Q^\pi(s,a) &= E\{r_t + \gamma r_{t+1} + && (2) \\ &\quad \gamma^2 r_{t+2} + \dots | s_t = s, a_t = a, \pi\} && (3) \end{aligned}$$

The above framework, looks at primitive actions lasting a single time-step, but ignores the possibility of learning about actions that extend in time. A Semi-Markov Decision Process (SMDP) (Barto & Mahadevan, 2003) is another formulation that looks into temporally extended actions, *or options*, comprised of a sequence of primitive actions. An option can be thought of as a sub-policy μ within the policy π, where π selects between primitive actions $a \in A$ and options $o \in O$, and μ_o selects actions from A. Analogous to $Q^\pi(s,a)$, the state-option value for option o is defined as:

$$Q^\pi(s,o) = E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | I(o,s,t)\}$$

where, $I(o,s,t)$ denotes the event that option o is executed in state s at time t. The reward obtained by executing option o of duration k at state s is given by

$$r_s^o = E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^k r_{t+k} | I(o,s,t)\}$$

As pointed out by Sutton et al. (Sutton et al., 1999; Sutton & Precup, 1998), options can be used over MDPs to learn about primitive actions and other options, through *intra-option* methods. If option o at state s_t executes the sequences of actions $a_t, a_{t+1}, \dots, a_{t+k}$, then not only is the agent able to learn about $Q(s_t, o)$, but also about $Q(s_t,a_t), Q(s_{t+1},a_{t+1}), \dots, Q(s_{t+k},a_{t+k})$. In light of this, an agent equipped with options that guide exploration should be able to focus its efforts to learning $V(s)$ or $Q(s,a)$ for states and state-action pairs that are relevant to a given MDP.

3. Problem Formulation

We are interested in obtaining reusable options that would help an agent reach an optimal policy π^* in a given task τ sampled from a domain Γ. We refer to a particular instance of an MDP drawn from the class of problems present in a specific domain as a task. Let us define an action set $A = \{a_1, a_2, \dots, a_n\}$ containing all actions available in Γ, $P = \{p_1, p_2, \dots, p_n\}$ a set of probabilities where p_i corresponds to the probability of seeing a_i in samples taken from the optimal policies over all tasks under domain Γ. It is then possible to create a codebook $C(A,P)$ generating binary encodings c_i for $i = 1, \dots, n$ for action each a_i, that is, each a_i can be represented as a string of 0s and 1s. Drawing an analogy to the compression literature, by using Huffman coding (Huffman, 1952), a trajectory from a policy π can be expressed as $\omega = \{c^1, c^2, \dots, c^t\}$, where c^t is the codeword of the action performed at time-step t.

We seek to find a set of deterministic options O, such that if a new action set $A' = A \cup O$ were used, it would achieve a more compact representation of sample trajectories taken from π^* for tasks drawn from some domain Γ. More formally, let $B(\omega, \tau)$ denote the number of bits needed to represent trajectory ω for task τ, and let $E_{\tau \sim \Gamma}[B_A(\omega^{\pi^*}, \tau)]$ be the expected number of bits needed to represent a trajectory ω sampled from an optimal policy π^* given an action set A in domain Γ. We wish to find a new action set $A' = A \cup O$, such that:

$$E_{\tau\sim\Gamma}[B_{A'}(\omega^{\pi^*},\tau)] \ll E_{\tau\sim\Gamma}[B_A(\omega^{\pi^*},\tau)]$$

The reasoning behind this objective is that finding a set of options leading to compressed trajectory representations implies that these options are encountered frequently, given that the length of a codeword depends on its frequency. Therefore, it is likely that they capture underlying patterns in the optimal policy π^* where the trajectories were sampled from, providing the agent guidance for exploration while learning a new task. To do so, we can attempt to minimize the following objective function:

$$\begin{aligned} J(A') &= E_{\tau\sim\Gamma}[B_{A'}(\omega^{\pi^*},\tau)] + \lambda|A'| \\ &= \sum_{\tau=1}^{T} P(\tau|\Gamma)(B_{A'}(\omega^{\pi^*},\tau)) + \lambda|A'| \end{aligned} \tag{4}$$

The first term of the equation seeks to minimize the expected number of bits needed to encode trajectories sampled from π^*, while the second term can be thought of as a regularizer on the size of the action space, which includes both primitive actions and options. If A' were to become too large, it would do a disservice to the agent, since there would be too many actions and options to learn about.

In practice, it is infeasible to find A' that minimizes this expression, given that the agent can only sample tasks from Γ. However, we can approximate the solution to $J(A')$ from sample tasks by sampling their optimal policies. For a single sample from a task with deterministic π^*, minimizing $B_A(\omega^{\pi^*},\tau)$ corresponds to adding the sampled trajectory as an option to A. However, for several tasks, the best options will correspond to repeating sequences of actions that were frequently encountered. Assuming an agent equipped with a budget of N options of length l, we derive an exploration strategy for deterministic environments in the discrete and continuous actions spaces. Our experiments show that guiding exploration in this fashion can have a significant impact on the performance of an agent.

4. MDPs in Discrete Action Space

Let k be the number of time-steps needed to execute a trajectory ω^{π^*} from policy π^*, such that the agent is in state s_0 at $t=0$ and in terminal state s_k at $t=k$, and let A be the action set. We define

$$seq_{\omega^{\pi^*}} = \{a_0, a_1, \ldots, a_{k-1}, a_k\}$$

as the sequence of actions executed by ω^{π^*} from a specific initial state. A sub-sequence of actions $sub_{j,l}$ of length l starting at index $j < k-l$ is composed of $\{a_j, a_{j+1}, \ldots, a_{j+l}\}$. For the rest of this section we consider sub-sequences as open-loop deterministic options.

Equality between two sub-sequences of length l, sub_1 and sub_2, can be defined as $a^1_i = a^2_i$, for $i = k,\ldots,k+l$, where $sub_1 = \{a^1_k, a^1_{k+1}, \ldots, a^1_{k+l}\}$ and $sub_2 = \{a^2_k, a^2_{k+1}, \ldots, a^2_{k+l}\}$. That is, two sub-sequences are equivalent if the actions at step i are the same. Following the premise of creating compact policy representations, we consider a Huffman encoding over options. To do so, we need access to the true probability p_o of taking option o. Given that we can only sample from Γ, we cannot obtain p_o, but we can obtain an estimate by looking at its frequency count C_o over all sampled trajectories. Given a set $\Omega = \{\omega_1, \omega_2, \ldots, \omega_P\}$, where ω_i denotes the a trajectory sampled from an optimal policy to task $\tau \in \Gamma$, and let $sub^i_{j,l}$ denote the sub-sequence of length l starting at index j of ω_i. The frequency count C_o for a sub-sequence sub_o is defined by the following equation:

$$C_o = \sum_{i=1}^{P} \sum_{j=1}^{|seq_{\pi^*_i}|-l} I(sub^i_{j,l} = sub_o)$$

where I is the indicator function.

From the N most frequent sub-sequences we can determine a probability distribution D_o where option o is sampled with probability $p_o = \frac{C_o}{\sum_{k=1}^{N} C_k}$. Exploring in this manner has the effect of biasing exploration towards options that occurred more frequently, i.e. options that lead to a more compact policy representation. When facing a new problem, the agent selects actions according to the following strategy:

$$\pi_{exp}(s) = \begin{cases} argmax_{u\in A'} Q(s,u) & \eta \geq \epsilon \\ \begin{cases} random(a) \in A & \delta \leq \frac{1}{N+1} \\ o \sim D_o & otherwise \end{cases} & otherwise \end{cases}$$

where η and δ are random numbers drawn uniformly between 0 and 1, A is the action space composed of only primitive actions and $A' = A \cup O$, where O is the set of N extracted options. The idea above defines an ϵ-*greedy* exploration strategy, where the agent chooses to explore with probability ϵ. If it does choose to explore, then with probability $\frac{1}{1+N}$ it will select a random primitive action, otherwise it will select an option according to the distribution D_o.

5. MDPs in Continuous Action Space

In this setting, we take the intuition built in the previous section and seek to find an analogous approach to obtaining options from sampled policies. However, continuous action spaces present some unique challenges. Unlike in discrete action spaces, there are now an infinite number of possible primitive actions, and it is unlikely that two sequences of actions will be identical. To deal with this situation, we could discretize this space, but that raises a new question: what is the right resolution to select? If the resolution is too coarse, the agent might not have important action at its disposal. On the other hand, if the resolution is too fine-grained, it becomes unlikely that two sequences will appear twice and the agent will have too many actions to learn.

In this situation, it helps to think of a trajectory ω_{π^*} as a signal

$$\omega_{\pi^*} = \{a_0, a_1, \ldots, a_{k-1}, a_k\}$$

where each $a \in A$ is a real number. As mentioned above, we cannot simply compare segments of a trajectory by looking for exact equality between trajectories. Hence, a different similarity measure needs to be established to obtain repeating patterns. To that end, we use Dynamic Time Warping (DTW) to obtain a similarity metric.

DTW (Ann & Keogh) is an algorithm developed for measuring similarity between two signals that may vary in time. It produces a mapping from one signal to the other, as well as a distance measure of such mapping. We define equality between two trajectory segments seg_a and seg_b as:

$$\begin{cases} seg_a = seg_b & dtw(seg_a, seg_b) < \alpha \\ seg_a \neq seg_b & otherwise \end{cases}$$

where α is a environment dependent similarity threshold defined by the user and $dtw(seg_a, seg_B)$ is the mapping distance between seg_a and seg_b given by DTW. To obtain a frequency count, we group segments based on similarity and compute the mean segment of each group as a possible open loop option. Let $a_{i,j}$ be the i^{th} action in segment j, then for a set M of segments of length l, the mean segment is defined as:

$$\bar{M} = \{\bar{a_0}, \bar{a_1}, \ldots, a_{(\bar{l-1})}, \bar{a_l}\}$$

where $\bar{a_i} = \frac{\sum_{j=1}^{|M|} a_i}{|M|}$.

Let k_p be the total number of segments obtained from the p sampled trajectories. Beginning with k_p clusters $S_i \in G$, where $S_i = \{seg_i\}$ for $i = 1, \ldots, k_p$, and G denotes the set containing all clusters, the segments are clustered as given in algorithm 1.

For every segment seg_i, the procedure finds the group S_{min} for which the dtw distance between its mean segment $\bar{S_{min}}$ and seg_i is minimum. If this value is smaller than α, seg_i is removed from its group, inserted into S_{min} and $\bar{S_{min}}$ is recomputed. At the end of the procedure, the frequency count C_i for mean segment $\bar{S}_i$ is given by the number of segments that compose that group: $C_i = |S_i|$. The mean

Algorithm 1 Repeating Sub-sequence Approximation

```
1:  Input: $G$ (the set of segment clusters)
2:  $k_p \leftarrow |G|$
3:  for i=1 to $k_p$ do
4:    $min \leftarrow \infty$
5:    for $S \in G$ do
6:      if $seg_i \in S$ then
7:        $T \leftarrow S$
8:      else if $dtw(seg_i, \bar{S}) < min$ then
9:        $min \leftarrow dtw(seg_i, \bar{S})$
10:       $S_{min} \leftarrow S$
11:
12:   if $min < \alpha$ then
13:     $T \leftarrow T \setminus \{seg_i\}$
14:     $S_{min} \leftarrow S_{min} \cup \{seg_i\}$
15:     compute mean of $S_{min}$
```

segments are then used as open loop options according to the exploration strategy described in the previous section.

Pseudocode for the full implementation of the algorithm is given in 2. The procedure takes as input a list of sampled trajectories Ω, the number of options N of length l to extract and the number of episodes to train e and a list O_{sort} with all extracted segments sorted by frequency. Lines 1-6, creates a temporary list of all sequences and their frequency count. Lines 8-11 create the set of options O containing the N most frequent options. The rest of the procedure boils down to intra-option learning, (Sutton & Precup, 1998), taking actions according to the exploration policy π_{exp}. No specific update rule for q-values is given, since this method is agnostic to the specific learning algorithm used.

6. Experiments

We carried out experiments in two different domain for the discrete and continuous action space settings. In the discrete case, we tested the standard benchmarks of *grid-world* whereas in the continuous case we used *mountain car*, (Moore, 1990). Sample optimal policies were collected by using *Q-Learning*, (Watkins & Dayan, 1992), and *Deterministic Policy Gradient* (DPG), (Silver et al., 2014), in different tasks drawn from each domain. Our method was then tested on problems that were not part of the dataset against an agent with no options and an agent with options drawn at random.

6.1. Grid-world Domain

For this set of experiments, we trained the agent to find optimal policies on 20 different mazes. The agent receives a reward of -1 at each time-step and a reward of +10 once it reaches a pre-defined goal. The agent has at his disposal

Algorithm 2 Data-driven Exploration

```
1:  Input: $\Omega, N, l, e, O_{sort}$
2:  $O \leftarrow \emptyset$
3:  for $i = 1$ to $N$ do
4:      $o \leftarrow top(O_{sort})$
5:      $O \leftarrow \cup\{o\}$
6:      $O_{sort} \leftarrow O_{sort} \setminus \{o\}$

8:  for $i = 1$ to $e$ do
9:      $s \leftarrow s_0$
10:     $done \leftarrow false$
11:     for $t = 1$ to $T$ do
12:         $o \leftarrow \pi_{exp}(s)$
13:         if $o$ is primitive then
14:             $s', r \leftarrow$ take action $o$
15:             if $s'$ is terminal then
16:                 $done \leftarrow true$
17:         else
18:             $r \leftarrow 0, s_{temp} \leftarrow s$
19:             for $a \in o$ do
20:                 $s', r' \leftarrow$ take action $a$
21:                 update $Q(s_{temp}, a)$ with reward $r'$
22:                 if $s'$ is terminal then
23:                     $done \leftarrow true$
24:                     break
25:                 $r \leftarrow r + r'$
26:                 $s_{temp} \leftarrow s'$
27:         update $Q(s, o)$ with reward $r$
28:         $s \leftarrow s'$
29:         if $done = true$ then
30:             break
```

four different actions: moving right (R), down (D), left (L) and up (U). All mazes were taken from a set of challenging benchmarks, (Sturtevant, 2012), based on environments from the video game Baldur's Gate. Figure 1 shows a performance comparison for 4 different test environments that were not part of the training set over 5000 episodes. In green we show the performance of an agent with no options, in red an agent equipped with random options and in blue an agent with extracted options. In figures 1a, 1b the agent is equipped with options of length 4, while in 1c, 1d the options have length 8.

While the performance in 1a,1c is only marginally better, our approach clearly outperforms the other two in 1b, 1d. This illustrates that the options extracted are guaranteed to improve performance on every task, but in expectation over tasks sampled from the domain on which the agent has trained. Table 1 shows the 4 most frequent sequences of actions extracted for options of length 4 and length 8.

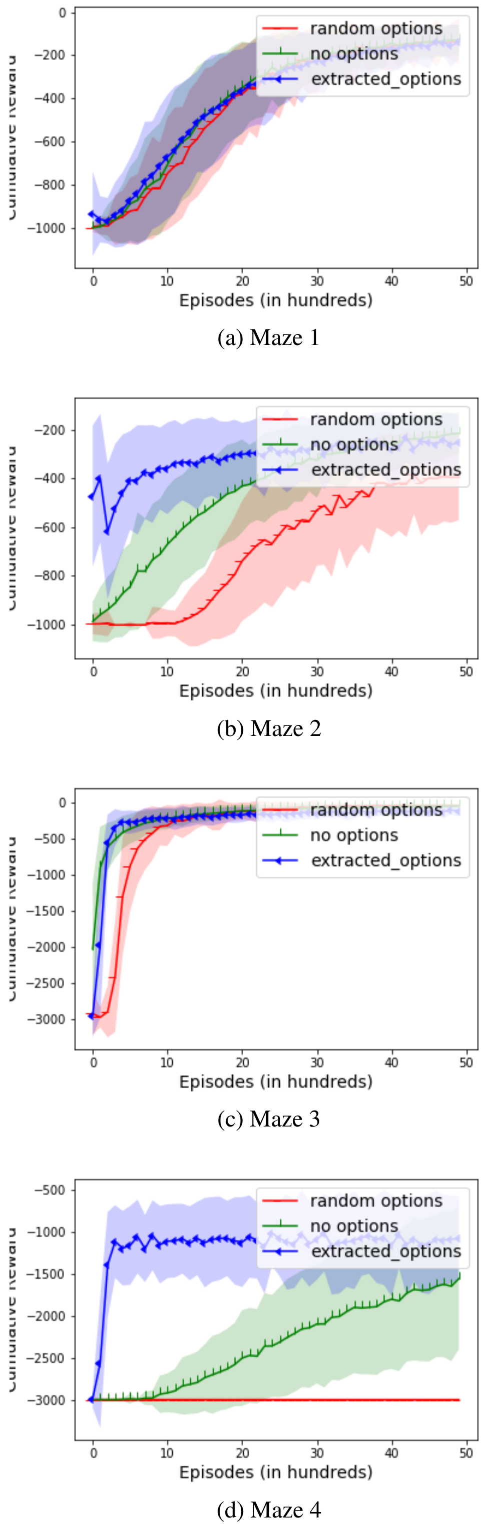


Figure 1. Experiments in grid-world domain comparing performance between agents with no options (green), random options (red), extracted options (blue).

Table 1. Options extracted from training maze environments for option lengths 4 and 8.

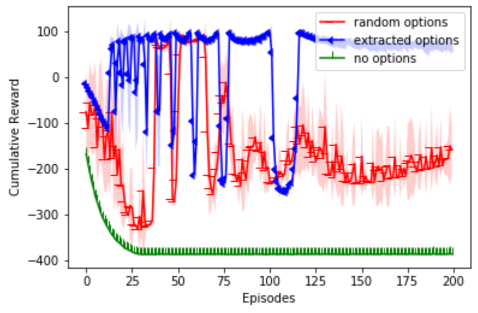
OPTIONS EXTRACTED
$o_1 = \{D, D, D, D\}$ $o_2 = \{R, R, R, R\}$ $o_3 = \{U, U, U, U\}$ $o_4 = \{L, L, L, L\}$
$o_1 = \{D, D, D, D, D, D, D, D\}$ $o_2 = \{L, L, L, L, L, L, L, L\}$ $o_3 = \{R, R, R, R, R, R, R, R\}$ $o_4 = \{R, D, D, D, D, D, D, D\}$

6.2. Mountain Car Domain

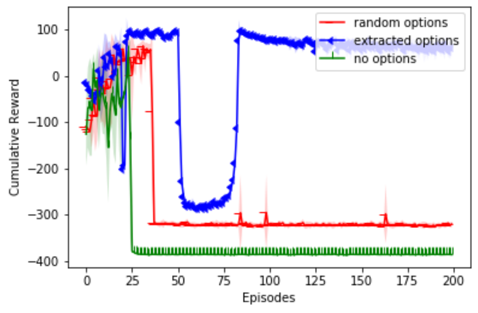
In this set of experiments, we used mountain car as the domain for testing learned options in continuous action spaces. The state variables are given in terms of a 4 dimensional vector representing the current position and velocity in the x and y axis. The action space is limited to one dimensional real values between $[a_{min}, a_{max}]$ representing a range of accelerations the agent is able to produce, where $a_{min} < 0$ and $a_{max} > 0$. The problem is considered solved when the agent is able to reach a reward of 90. To create training policies, we defined several variations of the problem withing the mountain car domain. These variations consisted in changing goal position, maximum velocity in the positive and negative x-axis, and maximum acceleration the agent is able to produce. Figure 2 shows two test problems, different from the ones the agent trained on. It compares an agent with extracted options (blue), an agent with no options (green) and an agent with random option (red). The results demonstrate that the agent is able to exploit the patterns that it has learned, and that simply creating temporally extended actions with no guidance can actually be detrimental to learning a new task. As in the case with grid-world, figure 3 demonstrate that we are also able to obtain meaningful sequences of actions as open loop options in continuous action spaces. The figure shows the 3 most frequent segments of length 6 and 8 extracted from the training policies.

7. Conclusion

In this work, we present an approach for learning compact representation of policies. The motivation for this is that from compact representations, patterns will emerge from the sequences of actions in the solution policies that capture an underlying structure of the domain at hand. This work develops a data-driven option extraction method, and an exploration strategy that exploits these options to solve novel tasks. We formulate the problem as an optimization

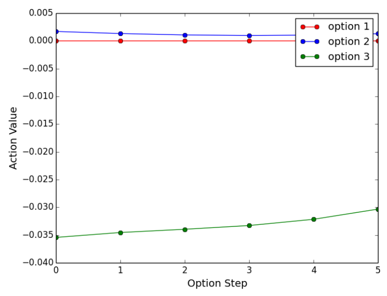


(a) Mountain Car Setting 1

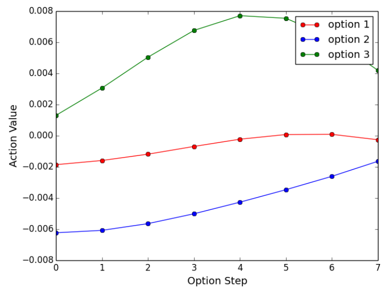


(b) Mountain Car Setting 2

Figure 2. Experiments on mountain car domain comparing performance between agents with no options (green), random options (red), extracted options (blue).



(a) Mountain Car Options - Length 6



(b) Mountain Car Options - Length 8

Figure 3. Options extracted from mountain car domain for options of length 6 (left) and 8 (right).

task, and develop an approximation to the solution based on sample policies. We perform a series of experiments that demonstrate the patterns that are being discovered from sample solutions, and provide evidence showing the benefits obtained by leveraging previous experience.

There are still several questions left unanswered. How do different option lengths affect performance? If the options are too short, the patterns that arise will be too general. On the other hand, if they are too long, they would be biased to the specific tasks the agent trained on. The number of options an agent has at its disposal presents a related problem, where too many options could make learning more difficult for the agent (the budget of options would be too general to be useful), while too few options would be beneficial in a very limited number of tasks (the budget of options would be too specific to help in many tasks). In summary, leveraging past experience brings great benefits when dealing with new problems from a domain which an agent has already encountered, and it is a promising direction for future research.

References

Ann, Chotirat and Keogh, Ratanamahatana Eamonn. Everything you know about dynamic time warping is wrong.

Barto, Andrew G. and Mahadevan, Sridhar. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003. ISSN 1573-7594. doi: 10.1023/A:1025696116075. URL http://dx.doi.org/10.1023/A:1025696116075.

Bertsekas, Dimitri P. and Tsitsiklis, John N. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108.

Huffman, David A. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.

Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1509.html#LillicrapHPHETS15.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning, 2013. URL http://arxiv.org/abs/1312.5602. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

Moore, Andrew William. Efficient memory-based learning for robot control. Technical Report UCAM-CL-TR-209, University of Cambridge, Computer Laboratory, November 1990. URL http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.pdf.

Precup, Doina. Temporal abstraction in reinforcement learning, 2000.

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In Jebara, Tony and Xing, Eric P. (eds.), *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395. JMLR Workshop and Conference Proceedings, 2014. URL http://jmlr.org/proceedings/papers/v32/silver14.pdf.

Sturtevant, N. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144 – 148, 2012. URL http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.

Sutton, Richard S. and Barto, Andrew G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Sutton, Richard S. and Precup, Doina. Intra-option learning about temporally abstract actions. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 556–564. Morgan Kaufman, 1998.

Sutton, Richard S., Precup, Doina, and Singh, Satinder P. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL http://dx.doi.org/10.1016/S0004-3702(99)00052-1.

Watkins, Christopher J. C. H. and Dayan, Peter. Q-learning. In *Machine Learning*, pp. 279–292, 1992.