A Meta-MDP Approach to Exploration for Lifelong Reinforcement Learning

Francisco M. Garcia
University of Massachusetts
Amherst, Massachusetts, USA
fmgarcia@cs.umass.edu

Philip Thomas
University of Massachusetts
Amherst, Massachusetts, USA
pthomas@cs.umass.edu

ABSTRACT

In this paper we consider the problem of how a reinforcement learning agent that is tasked with solving a sequence of reinforcement learning problems (a sequence of Markov decision processes) can use knowledge acquired early in its lifetime to improve its ability to solve new problems. Specifically, we focus on the question of how the agent should *explore* when faced with a new environment. We show that the search for an optimal exploration strategy can be formulated as a reinforcement learning problem itself, albeit with a different timescale. We conclude with experiments that show the benefits of optimizing an exploration strategy using our proposed approach.

KEYWORDS

Reinforcement Learning; Hierarchical RL; Exploration

1 INTRODUCTION

One hallmark of human intelligence is our ability to leverage knowledge collected over our lifetimes when we face a new problem. When we first drive a new car, we do not re-learn from scratch how to drive a car. Instead, we leverage our experience driving to quickly adapt to the new car (its handling, control placement, etc.). Standard *reinforcement learning* (RL) methods lack this ability. When faced with a new problem—a new *Markov decision process* (MDP)—they typically start from scratch, initially making decisions randomly to *explore* and learn about the current problem they face.

The problem of creating agents that can leverage previous experiences to solve new problems is called *lifelong learning* or *continual learning*, and is related to the problem of *transfer learning*. One could also argue that there is a relation to the problem *curriculum learning*, where the agent learns to solve a few simple tasks to allow him to solve more complex ones. In this paper, however, we focus on one aspect of lifelong learning: when faced with a sequence of MDPs sampled from a distribution over MDPs, how can a reinforcement learning agent learn an optimal policy for exploration? Specifically, we do not consider the question of *when* an agent should explore or *how much* an agent should explore, which is a well studied area of reinforcement learning research, [2, 6, 13, 20, 22]. Instead, we study the question of, given that an agent is going to explore, which action should it take?

After formally defining the problem of searching for an *optimal exploration policy*, we show that this problem can itself be modeled as an MDP. This means that the task of finding an optimal exploration strategy for a learning agent can be solved by another reinforcement learning agent that is solving a new *meta-MDP*. This meta-MDP operates at a different timescale from the RL agent solving specific MDPs—one time step of the meta-MDP corresponds to an entire lifetime of the RL agent. This difference of timescales distinguishes our approach from previous meta-MDP methods for optimizing components of reinforcement learning algorithms, [4, 9, 10, 23, 24].

We contend that using random action selection during exploration (as is common when using Q-learning, [25], Sarsa, [21], and DQN, [15]) ignores useful information from the agent's experience with previous similar MDPs that could be leveraged to direct exploration. We separate the policies that define the agent's behavior into an exploration policy (which governs behavior when the agent is exploring) and an exploitation policy (which governs behavior when the agent is exploiting).

In this paper we make the following contributions: **1)** we formally define the problem of searching for an optimal exploration policy, **2)** we prove that this problem can be modeled as a new MDP, and describe one algorithm for solving this meta-MDP, and **3)** we present experimental results that show the benefits of our approach. Although the search for an optimal exploration policy is only one of the necessary components for lifelong learning (along with deciding *when* to explore, how to represent data, how to transfer models, etc.), it provides one key step towards agents that leverage prior knowledge to solve challenging problems.

2 RELATED WORK

There is a large body of work discussing the problem of *how* an agent should behave during exploration *when faced with a single MDP*. Simple strategies, such as ϵ-greedy with random action-selection, *Boltzmann action-selection* or *softmax action-selection*, make sense when an agent has no prior knowledge of the problem that it is facing. The performance of an agent exploring with random action-selection reduces drastically as the size of the state-space increases [26]. The performance of Boltzmann or softmax action-selection hinges on the accuracy of the action-value estimates. When these estimates are poor (e.g., early during the learning process), it can have a drastic negative effect on the overall performance of the agent. More sophisticated methods search for subgoal states to define temporally-extended actions, called *options*, that explore the state-space more efficiently, [7, 14], use state-visitation counts to encourage the agent to explore states that have

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 2019, Montreal, Canada

https://doi.org/doi

not been frequently visited, [13, 22], or use approximations of a state-transition graph to exploit structural patterns, [11, 12].

Recent research concerning exploration has also taken the approach of adding an exploration "bonus" to the reward function. VIME [8] takes a Bayesian approach by maintaining a model of the dynamics of the environment, obtaining a posterior of the model after taking an action, and using the KL divergence between these two models as a bonus. The intuition behind this approach is that encouraging actions that make large updates to the model allows the agent to better explore areas where the current model is inaccurate. Pathak et al. [16] define a bonus in the reward function by adding an intrinsic reward. They propose using a neural network to predict state transitions based on the action taken and provide an intrinsic reward proportional to the prediction error. The agent is therefore encouraged to make state transitions that are not modeled accurately. Another relevant work in exploration was presented by Fernandez and Veloso [4], where the authors propose building a library of policies from prior experience to explore the environment in new problems more efficiently. These techniques are useful when an agent is dealing with a single MDP or class of MDPs with the same state-transition graph, however they do not provide a means to guide an agent to explore intelligently when faced with a novel task with different dynamics.

The idea of meta-learning, or learning to learn, has also been a recent area of focus. Andrychowicz et al. [1] proposed learning an update rule for a class of optimization problems. Given an objective function f and parameters θ, the authors proposed learning a model, g_ϕ, such that the update to parameters θ_k, at iteration k are given according to $\theta_{k+1} = \theta_k + g_\phi(\nabla f(\theta_k))$. RL has also been used in meta-learning to learn efficient neural network architectures [18]. However, even though one can draw a connection to our work through meta-learning, these methods are not concerned with the problem of exploration.

In the context of RL, a similar idea can be applied by defining a meta-MDP, i.e., considering the agent as part of the environment in a larger MDP. In multi-agent systems, Liu et al. [10] considered other agents as part of the environment from the perspective of each individual agent. Thomas and Barto [23] proposed the conjugate MDP framework, in which agents solving meta-MDPs (called CoMDPs) can search for the state representation, action representation, or options that maximize the expected return when used by an RL agent solving a single MDP.

Despite existing meta-MDP approaches, to the best of our knowledge, ours is the first to use the meta-MDP approach to specifically optimize exploration for a set of related tasks.

3 BACKGROUND

A *Markov decision process* (MDP) is a tuple, $M = (\mathcal{S}, \mathcal{A}, P, R, d_0)$, 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 takes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, $R(s, a, s')$ is a function denoting the reward received after taking action a in state s and transitioning to state s', and d_0 is the initial state distribution. We use $t \in \{0, 1, 2, \ldots, T\}$ to index the time-step, and write S_t, A_t, and R_t to denote the state, action, and reward at time t. We also consider the *undiscounted* episodic setting, wherein rewards are not discounted based on the time at which they occur. We assume that T, the maximum time step, is finite, and thus we restrict our discussion to *episodic* MDPs; that is, after T time-steps the agent resets to some initial state. We use I to denote the total number of episodes the agent interacts with an environment. A *policy*, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, provides a conditional distribution over actions given each possible state: $\pi(s, a) = \Pr(A_t = a | S_t = s)$. Furthermore, we assume that for all policies, π, (and all tasks, $c \in C$, defined later) the expected returns are normalized to be in the interval $[0, 1]$.

One of the key challenges within RL, and the one this work focuses on, is related to the *exploration-exploitation dilemma*. To ensure that an agent is able to find a good policy, it needs to take actions with the sole purpose of gathering information about the environment (exploration). However, once enough information is gathered, it should behave according to what it believes to be the best policy (exploitation). In this work, we separate the behavior of an RL agent into two distinct policies: an *exploration* policy and an *exploitation* policy. We assume an ϵ-greedy exploration schedule, i.e., with probability ϵ_i the agent explores and with probability $1 - \epsilon_i$ the agent exploits, where $(\epsilon_i)_{i=1}^{I}$ is a sequence of exploration rates where $\epsilon_i \in [0, 1]$ and i refers to the episode number in the current task.

Let C be the set of all tasks, $c = (\mathcal{S}, \mathcal{A}, P_c, R_c, d_0^c)$. That is, all $c \in C$ are MDPs sharing the same state-set $\mathcal{S}$ and action-set $\mathcal{A}$, which may have different transition functions P_c, reward functions R_c, and initial state distributions d_0^c. An agent is required to solve a set of tasks or levels $c \in C$, where we refer to the set C as the *problem class*. For example, if C refers to learning to balance a pole, each task $c \in C$ could refer to balancing a pole with a given height and weight, determining different degree of difficulty. The agent has a task-specific policy, π, that is updated by the agent's own learning algorithm. This policy defines the agent's behavior during exploitation, and so we refer to it as the *exploitation policy*. The behavior of the agent during exploration is determined by an *advisor*, which maintains a policy tailored to the problem class (i.e., it is shared across all tasks in C). We refer to this policy as an *exploration policy*, $\mu : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$.

The agent will have $K = IT$ time-steps of interactions with each of the sampled tasks. Hereafter we use i to denote the index of the current episode on the current task, t to denote the time step within that episode, and k to denote the number of time steps that have passed on the current task, i.e., $k = iT + t$, and we refer to k as the *advisor time step*. At every time-step, k, the advisor suggests an action, U_k, to the agent, where U_k is sampled according to μ. If the agent decides to explore at this step, it takes action U_k, otherwise it takes action A_k sampled according to the agent's policy, π. We refer to an optimal policy for the agent solving a specific task, $c \in C$, as an *optimal exploitation policy*, π_c^*. More formally: $\pi_c^* \in \underset{\pi}{\operatorname{argmax}}\ \mathbf{E}[G|\pi, c]$, where $G = \sum_{t=0}^{T} R_t$ is referred to as the return. Thus, the agent solving a specific task is optimizing the standard expected return objective. From now on we refer to the agent solving a specific task as the *agent* (even though the advisor can also be viewed as an agent).

Intuitively, we consider a process that proceeds as follows. First, a task, $c \in C$ is sampled from some distribution, d_C, over C. Next, the agent uses some pre-specified reinforcement learning algorithm (e.g., Q-learning or Sarsa) to approximate an optimal policy on the sampled task, c. Whenever the agent decides to explore, it uses an action provided by the *advisor* according to its policy, μ. After the agent completes I episodes on the current task, the next task is sampled from C and the agent's policy is reset to an initial policy. Notice that the goals of the advisor and agent solving a specific task are different: the agent solving a specific task tries to optimize the expected return on the task at hand, while the advisor searches for an exploration policy that causes the agent to learn quickly across all tasks. As such, the advisor may learn to suggest bad actions if that is what the agent needs to see to learn quickly.

4 PROBLEM STATEMENT

We define the performance of the advisor's policy, μ, for a specific task $c \in \mathcal{C}$ to be $\rho(\mu, c) = \mathbf{E}\left[\sum_{i=0}^{I}\sum_{t=0}^{T} R_t^i \middle| \mu, c\right]$, where R_t^i is the reward at time step t during the i^{th} episode.

Let C be a random variable that denotes a task sampled from $d_{\mathcal{C}}$. The goal of the advisor is to find an *optimal exploration policy*, μ^*, which we define to be any policy that satisfies:

$$\mu^* \in \underset{\mu}{\operatorname{argmax}} \quad \mathbf{E}\left[\rho(\mu, C)\right]. \tag{1}$$

We cannot directly optimize this objective because we do not know the transition and reward functions of each MDP, and we can only sample tasks from $d_{\mathcal{C}}$. In the next section we show that the search for an exploration advisor policy can be formulated as an RL problem where the advisor is itself an RL agent solving an MDP whose environment contains both the current task, c, and the agent solving the current task.

5 A GENERAL SOLUTION FRAMEWORK

Our framework can be viewed as a meta-MDP—an MDP within an MDP. From the point of view of the agent, the environment is the current task, c (an MDP). However, from the point of view of the advisor, the environment contains both the task, c, and the agent. At every time-step, the advisor selects an action U and the agent an action A. The selected actions go through a selection mechanism which executes action A with probability $1 - \epsilon_i$ and action U with probability ϵ_i at episode i. Figure 1 depicts the proposed framework with action A (exploitation) being selected. Even though one time step for the agent corresponds to one time step for the advisor, one episode for the advisor constitutes a lifetime of the agent (all of its interactions with a sampled task). From this perspective, wherein the advisor is merely another reinforcement learning algorithm, we can take advantage of the existing body of work in RL to optimize the exploration policy, μ.

In this work, we experimented training the advisor policy using two different RL algorithms: REINFORCE, [27], and Proximal Policy Optimization (PPO), [19]. Pseudocode for an implementation of our framework using REINFORCE, where the meta-MDP is trained for I_{meta} episodes, is described in Algorithm 1.

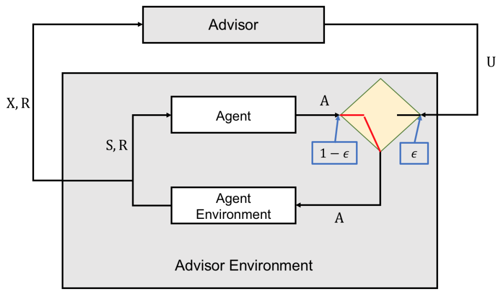


Figure 1: MDP view of interaction between the advisor and agent. At each time-step, the advisor selects an action U and the agent an action A. With probability ϵ the agent executes action U and with probability $1 - \epsilon$ it executes action A. After each action the agent and advisor receive a reward R, the agent and advisor environment transitions to states S and X, respectively.

Algorithm 1 Agent + Advisor - REINFORCE

```
Initialize advisor policy $\mu$ randomly
for $i_{meta} = 0, 1, \ldots, I_{meta}$ do
    Sample task $c$ from $d_c$
    for $i = 0, 1, \ldots, I$ do
        Initialize $\pi$ to $\pi_0$
        $s_t \sim d_0^c$
        for $t = 0, 1, \ldots, T$ do
            $a_t \sim \begin{cases} \mu \text{ with probability } \epsilon_i \\ \pi \text{ with probability } (1 - \epsilon_i) \end{cases}$
            take action $a_t$, observe $s_t, r_t$
        for $t = 0, 1, \ldots, T$ do
            update policy $\pi$ using REINFORCE with $s_t, a_t, r_t$
    for $k = 0, 1, \ldots, IT$ do
        update policy $\mu$ using REINFORCE with $s_k, a_k, r_k$
```

5.1 Theoretical Results

Below, we formally define the meta-MDP faced by the advisor and show that an optimal policy for the meta-MDP optimizes the objective in (1). Recall that R_c, P_c, and d_0^c denote the reward function, transition function, and initial state distribution of the MDP $c \in \mathcal{C}$.

To formally describe the meta-MDP, we must capture the property that the agent can implement an arbitrary RL algorithm. To do so, we assume the agent maintains some memory, M_k, that is updated by some learning rule l (an RL algorithm) at each time step, and write π_{M_k} to denote the agent's policy given that its memory is M_k. In other words, M_k provides all the information needed to determine π_{M_k} and its update is of the form $M_{k+1} = l(M_k, S_k, A_k, R_k, S_{k+1})$ (this update rule can represent popular RL algorithms like Q-Learning and actor-critics). We make no assumptions about which learning algorithm the agent uses (e.g., it can use Sarsa, Q-learning, REINFORCE, and even batch methods like Fitted Q-Iteration), and consider the learning rule to be unknown and a source of uncertainty.

Proposition 1. Consider an advisor policy, μ, and episodic tasks $c \in C$ belonging to a problem class C. The problem of learning μ can be formulated as an MDP, $M_{\text{meta}} = (\mathcal{X}, \mathcal{U}, T, Y, d_0')$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ the action space, T the transition function, Y the reward function, and d_0' the initial state distribution.

Proof. To show that M_{meta} is a valid MDP we need to characterize the MDP's state set, X, action set, U, transition function, T, reward function, Y, and initial state distribution d_0'. We assume that when facing a new task, the agent memory, M, is initialized to some fixed memory M_0 (defining a default initial policy and/or value function). The following definitions capture the intuition provided previously:

- $\mathcal{X} = \mathcal{S} \times \mathcal{I} \times \mathcal{C} \times \mathcal{M}$. That is, the state set $\mathcal{X}$ is a set defined such that each state, $x = (s, i, c, M)$ contains the current task, c, the current state, s, in the current task, the current episode number, i, and the current memory, M, of the agent.
- $\mathcal{U} = \mathcal{A}$. That is, the action-set is the same as the action-set of the problem class, C.
- T is the transition function, and is defined such that $T(x, u, x')$ is the probability of transitioning from state $x \in \mathcal{X}$ to state $x' \in \mathcal{X}$ upon taking action $u \in \mathcal{U}$. Assuming the underlying RL agent decides to explore with probability ϵ_i and to exploit with probability $1 - \epsilon_i$ at episode i, then T is as follows. If s is terminal and $i \neq I - 1$, then $T(x, u, x') = d_0^c(s')\mathbf{1}_{c'=c,\, i'=i+1,\, M'=l(M,s,a,r,s')}$. If s is terminal and $i = I - 1$, then $T(x, u, x') = d_C(c')d_0^{c'}(s')\mathbf{1}_{i'=0,\, M'=M_0}$. Otherwise, $T(x, u, x') = \big(\epsilon_i P_c(s, u, s') + (1 - \epsilon_i)\sum_{a \in A_c} \pi_M(s, a)P_c(s, a, s')\big) \times \mathbf{1}_{c'=c,\, i'=i,\, M'=l(M,s,a,r,s')}$
- Y is the reward function, and defines the reward obtained after taking action $u \in \mathcal{U}$ in state $x \in \mathcal{X}$ and transitioning to state $x' \in \mathcal{X}$ $Y(x, u, x') = \frac{\epsilon_i P_c(s,u,s')R_c(s,u,s') + (1-\epsilon_i)\sum_{a \in \mathcal{A}} \pi_M(a,s)P_c(s,a,s')R_c(s,a,s')}{\epsilon_i P_c(s,u,s') + (1-\epsilon_i)\sum_{a \in \mathcal{A}} \pi_M(a,s)P_c(s,a,s')}$.
- d_0' is the initial state distribution and is defined by: $d_0'(x) = d_C(c)d_0^c(s)\mathbf{1}_{i=0}$.

□

To show that an optimal policy of M_{meta} is an optimal exploration policy, we will first establish the following lemma to help us in our derivation.

Lemma 1.

$$
\begin{aligned}
&\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} (\epsilon_i P(A_k = a | S_k = s, \mu) \\
&+ (1 - \epsilon_i)P(A_k = a | S_k = s, \pi))P_c(s, a, s')R_c(s, a, s') \\
&= \sum_{x \in \mathcal{X}} \sum_{u \in \mathcal{A}} \sum_{x' \in \mathcal{X}} P(U_k = u | X_k = x)T(x, u, x')Y(x, u, x')
\end{aligned}
$$

Proof.

$$
\begin{aligned}
&\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} (\epsilon_i P(A_k = a | S_k = s, \mu) \\
&+ (1 - \epsilon_i)P(A_k = a | S_k = s, \pi))P_c(s, a, s')R_c(s, a, s') \\
&= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} (\epsilon_i P(A_k = a | S_k = s, \mu) \\
&\times P_c(s, a, s') + (1 - \epsilon_i)P(A_k = a | S_k = s, \pi) \\
&\times P_c(s, a, s'))R_c(s, a, s') \\
&= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} (\epsilon_i P(A_k = a | S_k = s, \mu) \\
&\times P_c(s, a, s')R_c(s, a, s') \\
&+ (1 - \epsilon_i)P(A_k = a | S_k = s, \pi) \\
&\times P_c(s, a, s')R_c(s, a, s')) \\
&= \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} (\sum_{u \in \mathcal{A}} \epsilon_i P(A_k = u | S_k = s, \mu) \\
&\times P_c(s, u, s')R_c(s, u, s') \\
&+ (1 - \epsilon_i) \sum_{a \in \mathcal{A}} P(A_k = a | S_k = s, \pi) \\
&\times P_c(s, a, s')R_c(s, a, s')) \\
&= \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} (\sum_{u \in \mathcal{A}} \epsilon_i P(A_k = u | S_k = s, \mu) \\
&\times P_c(s, u, s')R_c(s, u, s') \\
&+ \sum_{u \in \mathcal{A}} P(A_k = u | S_k = s, \mu) \\
&\times (1 - \epsilon_i) \sum_{a \in \mathcal{A}} P(A_k = a | S_k = s, \pi) \\
&\times P_c(s, a, s')R_c(s, a, s')) \\
&= \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} \sum_{u \in \mathcal{A}} P(A_k = u | S_k = s, \mu) \\
&(\epsilon_i P_c(s, u, s')R_c(s, u, s') \\
&+ (1 - \epsilon_i) \sum_{a \in \mathcal{A}} P(A_k = a | S_k = s, \pi) \\
&\times P_c(s, a, s')R_c(s, a, s'))
\end{aligned}
$$

Recall that given task c, episode i, advisor time-step k, the agent state s_k corresponds to the state of the advisor $x_k = (s, i, c, M_k)$. Continuing with our derivation.

$$
\begin{aligned}
&\sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} \sum_{u \in \mathcal{A}} P(A_k = u | S_k = s, \mu) \\
&(\epsilon_i P_c(s, u, s')R_c(s, u, s') \\
&+ (1 - \epsilon_i) \sum_{a \in \mathcal{A}} P(A_k = a | S_k = s, \pi) \\
&\times P_c(s, a, s')R_c(s, a, s'))
\end{aligned}
$$

$$= \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{u\in\mathcal{A}} P(A_k = u|S_k = s, \mu)$$

$$(\epsilon_i P_c(s,u,s')$$

$$+ (1-\epsilon_i)\sum_{a} P(A_k = a|S_k = s, \pi)$$

$$\times P_c(s,a,s'))$$

$$\times Y(x = (s,i,c,M_k), u, x' = (s',i,c,M_{k+1})) \quad \text{(by definition of } Y\text{)}$$

$$= \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{u\in\mathcal{A}} P(U_k = u|X_k = (s,i,c,M_k), \mu)$$

$$\times T(x = (s,i,c,M_k), u, x' = (s',i,c,M_{k+1})$$

$$\times Y(x = (s,i,c,M_k), u, x' = (s',i,c,M_{k+1}))$$

$$= \sum_{x\in\mathcal{X}}\sum_{x'\in\mathcal{X}}\sum_{u\in\mathcal{A}} P(U_k = u|X_k = x, \mu)$$

$$\times T(x,u,x')Y(x,u,x')$$

□

Theorem 5.1. *An optimal policy for M_{meta} is an optimal exploration policy, μ^*, as defined in* (1).

Proof. To show that an optimal policy of M_{meta} is an optimal exploration policy as defined in this paper, it is sufficient to show that maximizing the return in the meta-MDP is equivalent to maximizing the expected performance. That is, $\mathbf{E}\left[\rho(\mu, C)\right] = \mathbf{E}\left[\sum_{k=0}^{K} Y_k \middle| \mu, \right]$.

$$\mathbf{E}\left[\rho(\mu,C)\right] = \sum_{c\in C}\Pr(C=c)\,\mathbf{E}\left[\sum_{i=0}^{I}\sum_{t=0}^{T} R_t^i \middle| \mu, C=c\right]$$

$$= \sum_{c\in C}\Pr(C=c)\sum_{i=0}^{I}\sum_{t=0}^{T}\ \mathbf{E}\left[R_t^i \middle| \mu, C=c\right]$$

$$= \sum_{c\in C}\Pr(C=c)\sum_{i=0}^{I}\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\Pr(S_{iT+t}=s|C=c,\mu)$$

$$\times\ \mathbf{E}\left[R_t^i \middle| \mu, C=c, S_{iT+t}=s\right]$$

$$= \sum_{c\in C}\Pr(C=c)\sum_{i=0}^{I}\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\Pr(S_{iT+t}=s|C=c,\mu)$$

$$\times \sum_{a\in\mathcal{A}}\Pr(A_{iT+t}=a|S_{iT+t})\,\mathbf{E}\left[R_t^i \middle| \mu, C=c, S_{iT+t}=s, A_{iT+t}=a\right]$$

$$= \sum_{c\in C}\Pr(C=c)\sum_{i=0}^{I}\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\Pr(S_{iT+t}=s|C=c,\mu)$$

$$\times \sum_{a\in\mathcal{A}}\Pr(A_{iT+t}=a|S_{iT+t}=s)$$

$$\times \sum_{s'\in\mathcal{S}}\Pr(S_{iT+t+1}=s'|S_{iT+t}=s, A_{iT+t}=a)R(s,a,s')$$

$$= \sum_{c\in C}\Pr(C=c)\sum_{i=0}^{I}\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\Pr(S_{iT+t}=s|C=c,\mu)$$

$$\times \sum_{a\in\mathcal{A}}\Pr(A_{iT+t}=a|S_{iT+t})\,\mathbf{E}\left[R_t^i \middle| \mu, C=c, S_{iT+t}=s, A_{iT+t}=a\right]$$

$$= \sum_{c\in C}\sum_{i=0}^{I}\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}}\Pr(C=c)\Pr(S_{iT+t}=s|C=c,\mu)$$

$$\times \Pr(A_{iT+t}=a|S_{iT+t}=s)$$

$$\times \Pr(S_{iT+t+1}=s'|S_{iT+t}=s, A_{iT+t}=a)R(s,a,s')$$

$$= \sum_{c\in C}\sum_{k=0}^{K}\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}}\Pr(C=c)\Pr(S_k=s|C=c,\mu)$$

$$\times \Pr(A_k=a|S_k=s)$$

$$\times \Pr(S_{k+1}=s'|S_k=s, A_k=a)R(s,a,s')$$

$$= \sum_{c\in C}\sum_{k=0}^{K}\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}}\sum_{x'\in\mathcal{X}}\Pr(C=c)\Pr(X_k=x|C=c,\mu)$$

$$\times \Pr(U_k=a|X_k=x)$$

$$\times \Pr(X_{k+1}=s'|X_k=s, U_k=a)R(x,u,x')$$

$$= \sum_{c\in C}\sum_{k=0}^{K}\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}}\sum_{x'\in\mathcal{X}}\Pr(C=c)\Pr(X_k=x|C=c,\mu)$$

$$\times \Pr(U_k=a|X_k=x)T(x,u,x')Y(x,u,x') \quad \text{(by Lemma 1)}$$

$$= \sum_{k=0}^{K}\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}}\sum_{x'\in\mathcal{X}}\Pr(X_k=x|\mu)$$

$$\times \Pr(U_k=a|X_k=x)T(x,u,x')$$

$$\times \mathbf{E}\left[Y_k|X_k=s, U_k=a, X_{k+1}=x', \mu\right]$$

$$= \sum_{k=0}^{K}\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}}\Pr(X_k=x|\mu)$$

$$\times \Pr(U_k=a|X_k=x)$$

$$\times \mathbf{E}\left[Y_k|X_k=x, U_k=a, \mu\right]$$

$$= \sum_{k=0}^{K}\sum_{x\in\mathcal{X}}\Pr(X_k=x|\mu)$$

$$\times \mathbf{E}\left[Y_k|X_k=x, \mu\right]$$

$$= \sum_{k=0}^{K}\mathbf{E}\left[Y_k|\mu\right]$$

$$= \mathbf{E}\left[\sum_{k=0}^{K} Y_k|\mu\right]$$

□

Since M_{meta} is an MDP for which an optimal exploration policy is an optimal policy, it follows that the convergence properties of reinforcement learning algorithms apply to the search for an optimal exploration policy. For example, in some experiments the advisor uses the REINFORCE algorithm [27], the convergence properties of which have been well-studied [17].

Although the framework presented thus far is intuitive and results in nice theoretical properties (e.g., methods that guarantee convergence to at least locally optimal exploration policies), each episode corresponds to a new task $c \in C$ being sampled. This means that training the advisor may require to solve a large number of tasks (episodes of the meta-MDP), each one potentially being an expensive procedure. To address this issue, we sampled a small number of tasks $c_1, \ldots, c_n$, where each $c_i \sim d_C$ and train many episodes on each task in parallel. By taking this approach, every update to the advisor is influenced by several simultaneous tasks and results in an scalable approach to obtain a general exploration policy.

6 EMPIRICAL RESULTS

In this section we present experiments for discrete and continuous control tasks in the following problem classes: Pole-balancing, Animat, Hopper, and Ant, depicted in Figure 2. The implementations used for the discrete case pole-balancing and all continuous control problems, where taken from OpenAI Gym and Roboschool benchmarks, [3]. We demonstrate that: **1)** in practice the meta-MDP, M_{meta}, can be solved using existing reinforcement learning methods, **2)** the exploration policy learned by the advisor improves performance on existing RL methods, on average, and **3)** the exploration policy learned by the advisor differs from the optimal exploitation policy for any task $c \in C$, i.e., the exploration policy learned by the advisor is *not* necessarily a good exploitation policy.

To that end, we will first study the behavior of our method in two problem classes with discrete action-spaces: pole-balancing [21] and animat [23]. Figure 2a and 2b represent two variations of the pole-balancing problem class, where the height and mass of the pole differ significantly. Figure 2c and 2d represent two variations for animat, where the environment layout and goal locations can differ arbitrarily.

We chose these problems because there are easy-to-interpret behaviors in an optimal policy that are shared for any variation of the tasks. In pole-balancing, if a pole is about to fall to the right, taking an action that moves the pole further to the right will increase the odds of dropping the pole. In the animat problem class, there are actions that are not helpful for reaching any goal location. To meet our original criterion that returns are normalized between 0 and 1, we normalize the returns using estimates of the minimum and maximum possible expected returns for each task.

As a baseline meta-learning method, to which we contrast our framework, we chose the recently proposed technique called Model Agnostic Meta Learning (MAML), [5]. MAML was proposed as a general meta learning method for adapting previously trained neural networks to novel but related tasks. It is worth noting that the method was not specifically designed for RL, nonetheless, in their paper, the authors describe some promising results in adapting behavior learned from previous tasks to novel ones.

In the case of RL, MAML samples a batch of related tasks and maintains a global parameter for the meta-learner and a task-specific parameter for each task. The agent samples trajectories from each task, and each task parameter is updated according to its own specific objective. The global parameters are updated by following the sum of the gradients obtained from all tasks. In this manner, the global parameters are updated according to all training tasks in the batch. After training, when the agent faces a new task, it simply initializes its policy to that given by the global parameters.

There are few key differences between MAML and our method. Given that the global parameter are used to initialize the agents policy on novel tasks, it imposes a constraint that the policy of the meta-learner should have the same form as that of the agent. In contrast, we allow for different learning algorithms to be used for the advisor (the meta-learner) and the agent. Furthermore, the global policy learned by MAML is only used for initialization and it is updated thereafter. Since we focus in the RL setting, we specifically learn a policy suited for the problem class that the agent can call at any time.

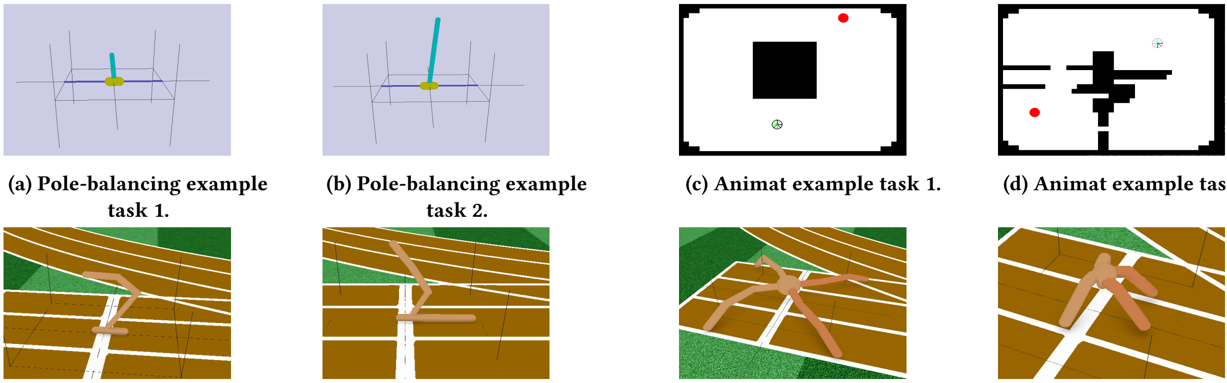
6.1 Pole Balancing Problem Class

In our first experiments on discrete action sets, we used variants of the standard pole-balancing (cart-pole) problem class. The agent is tasked with applying force to a cart to prevent a pole balancing on it from falling. The distinct tasks were constructed by modifying 4 variables: pole mass, m_p, pole length, l, cart mass, m_c, and force magnitude, f. States are represented by 4-D vectors describing the position and velocity of the cart, and angle and angular velocity of the pendulum, i.e., $s = [x, v, \theta, \dot{\theta}]$. The agent has 2 actions at its disposal: apply a force f in the positive or negative x direction.

Figure 3a, contrasts the cumulative return of an agent using the advisor for exploration (in blue) with the cumulative return obtained by an agent using ϵ-greedy random exploration (in red) during training over 6 training tasks. The exploitation policy, π, was trained using REINFORCE for $I = 1{,}000$ episodes and the exploration policy, μ, was trained using REINFORCE for 500 iterations. In the figure, the horizontal axis corresponds to iterations—episodes for the adviser. The horizontal red line denotes an estimate (with standard error bar) of the expected cumulative reward that an agent will obtain during its lifetime if it samples actions uniformly when exploring. Notice that this is not a function of the training iteration, as the random exploration is not updated. The blue curve (with standard error bars from 15 trials) shows how the expected cumulative reward that the agent will obtain during its lifetime changes as the advisor learns to improve its policy. Here the horizontal axis shows the number of training iterations—the number of episodes of the meta-MDP. By the end of the plot, the agent is obtaining roughly 30% more reward during its lifetime than it was when using a random exploration. To better visualize this difference, Figure 3b shows the mean *learning curves* (episodes of an agent's lifetime on the horizontal axis and average return for each episode on the vertical axis) during the first and last 50 iterations. The mean cumulative reward were 25,283 and 30,552 respectively. Notice that, although the final performance obtained is similar, using a trained advisor allows the agent to reach this level of performance faster; thus achieving a larger cumulative return.

6.2 Animat Problem Class

The following set of experiments were conducted in the *animat* problem class. In these environments, the agent is a circular creature that lives in a continuous state space. It has 8 independent actuators, angled around it in increments of 45 degrees. Each actuator can be



(a) Pole-balancing example task 1.

(b) Pole-balancing example task 2.

(c) Animat example task 1.

(d) Animat example task 2.

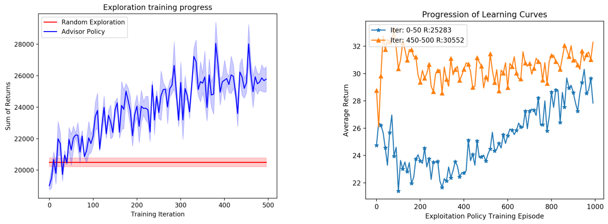
(e) Hopper example task 1.

(f) Hopper example task 2.

(g) Ant example task 1.

(h) Ant example task 2.

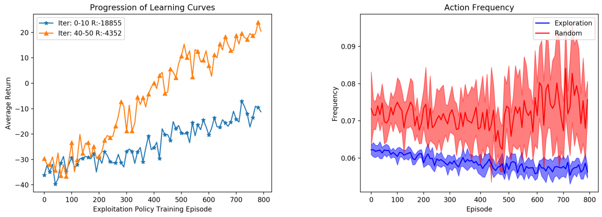
Figure 2: Example of task variations used in our experiments. The problem classes correspond to pole-balancing (top left), animat (top right), hopper (bottom left), and ant (bottom right)



(a) Performance curves during training comparing advisor policy (blue) and random exploration policy (red).

(b) Average learning curves on training tasks over the first 50 advisor episodes (blue) and the last 50 advisor episodes (orange).

Figure 3: Advisor results on pole-balancing problem class.



(a) Average learning curves for animat on training tasks over the first 10 iterations (blue) and last 10 iterations (orange).

(b) Frequency of poor-performing actions in an agent's lifetime with learned (blue) and random (red) exploration.

Figure 4: Advisor results in the animat problem class.

either on or off at each time step, so the action set is $\{0, 1\}^8$, for a total of 256 actions. When an actuator is on, it produces a small force in the direction that it is pointing. The agent is tasked with moving to a goal location; it receives a reward of -1 at each time-step and a reward of +100 at the goal state. The different variations of the tasks correspond to randomized start and goal positions in different environments. The agent moves according to the following mechanics: let (x_t, y_t) define the state of the agent at time t and d be the total displacement given by actuator β with angle θ_β. The displacement of the agent for a set of active actuators, $\mathcal{B}$, is given by, $(\Delta_x, \Delta_y) = \sum_{\beta \in \mathcal{B}} (d \cos(\theta_\beta), d \sin(\theta_\beta))$. After taking an action, the new state is perturbed by 0-mean unit variance Gaussian noise.

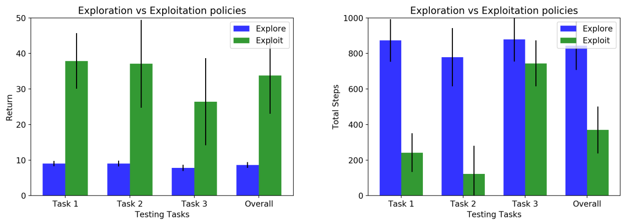
An interesting pattern that is shared across all variations of this problem class is that there are actuator combinations that are not useful for reaching the goal. For example, activating actuators at $\theta = 0°$ and $\theta = 180°$ would leave the agent in the same position it was before (ignoring the effect of the noise). Even though the environment itself might not provide enough structure for the advisor to leverage previous experiences, the presence of these poor performing actions provide some common patterns that can be leveraged.

Figure 4a shows the mean learning curves averaged over all training tasks, where the advisor was trained for 50 iterations. The curve in blue is the average curve obtained from the first 10 iterations of training the advisor and the curve in orange is the average obtained from the last 10 training iterations of the advisor. Each individual task was trained for $I = 800$ episodes. The figure shows a clear performance improvement on average as the advisor improves its policy.

To test our intuition that an exploration policy would exploit the presence of poor-performing actions, we recorded the frequency with which they were executed on unseen testing tasks when using the learned exploration policy after training and when using a random exploration strategy, over 5 different learned exploration policies. Figure 4b helps explain the difference in performance seen in Figure 4a. It depicts in the y-axis, the percentage of times these poor-performing actions were selected at a given episode, and in the x-axis the agent episode number in the current task. This shows that the agent using the advisor policy (blue) is encouraged to reduce the selection of known poor-performing actions, compared to a random action-selection exploration strategy (red).

Problem Class	R	R+Advisor	PPO	PPO+Advisor	MAML
Pole-balance (d)	20.32 ± 3.15	28.52 ± 7.6	27.87 ± 6.17	**46.29 ± 6.30**	39.29 ± 5.74
Animat	-779.62 ± 110.28	**-387.27 ± 162.33**	-751.40 ± 68.73	-631.97 ± 155.5	-669.93 ± 92.32
Pole-balance (c)	—	—	29.95 ± 7.90	**438.13 ± 35.54**	267.76 ± 163.05
Hopper	—	—	13.82 ± 10.53	**164.43 ± 48.54**	39.41 ± 7.95
Ant	—	—	-42.75 ± 24.35	83.76 ± 20.41	**113.33 ± 64.48**

Table 1: Average performance on discrete and continuous control unseen tasks over the last 50 episodes. In the cases where advisor performs best, the results are statistically significant. For the Ant domain, MAML appears to be better, although the high variance in returns makes this result *not* statistically significant



(a) Average returns obtained on test tasks when using the advisor's exploration policy (blue) and a task-specific exploitation (green)

(b) Number of steps needed to complete test tasks with advisor policy (blue) and exploitation (green).

Figure 5: Performance comparison of exploration and exploitation policies.

6.3 Is an Exploration Policy Simply a General Exploitation Policy?

One might be tempted to think that the learned policy for exploration might simply be a policy that works well in general. So how do we know that the advisor is learning a policy that is useful for exploration and not simply a policy for exploitation? To answer this question, we generated three distinct unseen tasks for both pole-balancing and animat problem classes and compare the performance of using only the learned exploration policy with the performance obtained by an exploitation policy trained to solve each specific task.

Figure 5 shows two bar charts contrasting the performance of the exploration policy (blue) and the exploitation policy (green) on each task variation. In both charts, the first three groups of bars on the x-axis correspond to the performance each test task and the last one to an average over all tasks. Figure 5a corresponds to the mean performance on pole-balancing and the error bars to the standard deviation; the y-axis denotes the return obtained. We can see that, as expected, the exploration policy by itself fails to achieve a comparable performance to a task-specific policy. The same occurs with the animat problem class, depicted in Figure 5b. In this case, the y-axis refers to the number of steps needed to reach the goal (smaller bars are better). In all cases, a task-specific policy performs significantly better than the learned exploration policy, indicating that the learned policy is useful for exploration, and *not* a general exploitation policy.

6.4 Performance Evaluation on Novel Tasks

In this section we examine the performance of our framework on novel tasks, and contrast our method to MAML trained using PPO. In the case of discrete action-sets, we trained each task for 500 episodes and compare the performance of an agent trained with REINFORCE (R) and PPO, with and without an advisor. In the case of continuous tasks, we restrict our experiments to an agent trained using PPO (since it was shown to perform well in continuous control problems), with and without an advisor after training for 500 episodes. In our experiments we set the initial value of ϵ to $\epsilon_0 = 0.8$, and defined the update after each agent episode to be $\epsilon_{i+1} = \max(0.1, 0.995\epsilon_i)$. The results shown in table 1 were obtained as follows. Each novel task was trained 5 times, and the average and standard deviation of those performances were recorded. The table displays the mean of those averages and the mean of the standard deviations recorded. In both the discrete and continuous case, there were 5 novel tasks. The problem classes "pole-balance (d)" and "animat" correspond to discrete actions spaces, while "pole-balance (c)", "hopper", and "ant" are continuous.

In the discrete case, we can see that for both pole-balancing and Animat, MAML showed a clear improvement over starting from a random initial policy. However, using the advisor with PPO resulted in a clear improvement in pole-balancing and, in the case of animat, training the advisor with REINFORCE led to an almost 50% improvement over MAML. In the case of continuous control, the first test corresponds to a continuous version of pole-balancing, where the different variations were obtained by modifying the length and mass of the pole, and the mass of the cart. The second and third set of tasks correspond to the "Hopper" and "Ant" problem classes, where the task variations were obtained by modifying the length and size of the limbs and body. In all continuous control tasks, both using the advisor and MAML led to an significant improvement in performance in the alloted time. In the case of pole-balancing using the advisor led the agent to accumulate almost twice as much reward as MAML, and in the case of Hopper, the advisor led to accumulating 4 times the reward. On the other had, MAML led to an higher average return than the advisor in the Ant problem class, but showing very high variance. An important takeaway from these results is that in all cases, using the advisor resulted in a clear improvement in performance over a limited number of episodes. This does not necessarily mean that the agent can reach a better policy over an arbitrarily long period of time, but rather that it is able to reach a certain performance level much quicker.

7 CONCLUSION

In this work we developed a framework for leveraging experience to guide an agent's exploration in novel tasks, where the *advisor* learns the exploration policy used by the *agent* solving a task. We showed that a few sample tasks can be used to learn an exploration policy that the agent can use improve the speed of learning on novel tasks.

REFERENCES

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. *CoRR* abs/1606.04474 (2016). arXiv:1606.04474 http://arxiv.org/abs/1606.04474

[2] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. 2017. Minimax Regret Bounds for Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, Doina Precup and Yee Whye Teh (Eds.). PMLR, International Convention Centre, Sydney, Australia, 263–272.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. (2016). arXiv:arXiv:1606.01540

[4] Fernando Fernandez and Manuela Veloso. 2006. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*. ACM, New York, NY, USA, 720–727. https://doi.org/10.1145/1160633.1160762

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1126–1135. http://proceedings.mlr.press/v70/finn17a.html

[6] Aurélien Garivier and Eric Moulines. 2011. On Upper-confidence Bound Policies for Switching Bandit Problems. In *Proceedings of the 22nd International Conference on Algorithmic Learning Theory (ALT'11)*. Springer-Verlag, Berlin, Heidelberg, 174–188.

[7] Sandeep Goel and Manfred Huber. 2003. Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies.. In *FLAIRS Conference*, Ingrid Russell and Susan M. Haller (Eds.). AAAI Press, 346–350.

[8] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2016. Curiosity-driven Exploration in Deep Reinforcement Learning via Bayesian Neural Networks. *CoRR* abs/1605.09674 (2016). arXiv:1605.09674 http://arxiv.org/abs/1605.09674

[9] Romain Laroche, Mehdi Fatemi, Harm van Seijen, and Joshua Romoff. 2017. Multi-Advisor Reinforcement Learning. (April 2017).

[10] Bingyao Liu, Satinder P. Singh, Richard L. Lewis, and Shiyin Qin. 2012. Optimal rewards in multiagent teams. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EPIROB 2012, San Diego, CA, USA, November 7-9, 2012*. 1–8. https://doi.org/10.1109/DevLrn.2012.6400862

[11] Marlos C. Machado, Marc G. Bellemare, and Michael H. Bowling. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. *CoRR* abs/1703.00956 (2017). arXiv:1703.00956

[12] Sridhar Mahadevan. 2005. Proto-value Functions: Developmental Reinforcement Learning. In *Proceedings of the 22Nd International Conference on Machine Learning (ICML '05)*. ACM, New York, NY, USA, 553–560. https://doi.org/10.1145/1102351.1102421

[13] Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. 2017. Count-Based Exploration in Feature Space for Reinforcement Learning. *CoRR* abs/1706.08090 (2017). arXiv:1706.08090

[14] Amy Mcgovern and Andrew G. Barto. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. of ICML*.

[15] 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. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533.

[16] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven Exploration by Self-supervised Prediction. *CoRR* abs/1705.05363 (2017). arXiv:1705.05363 http://arxiv.org/abs/1705.05363

[17] V. V. Phansalkar and M. A. L. Thathachar. 1995. Local and Global Optimization Algorithms for Generalized Learning Automata. *Neural Comput.* 7, 5 (Sept. 1995), 950–973. https://doi.org/10.1162/neco.1995.7.5.950

[18] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. 2017. Routing Networks: Adaptive Selection of Non-linear Functions for Multi-Task Learning. *CoRR* abs/1711.01239 (2017). arXiv:1711.01239 http://arxiv.org/abs/1711.01239

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[20] Alexander L. Strehl. 2008. Probably Approximately Correct (PAC) Exploration in Reinforcement Learning. In *ISAIM*.

[21] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[22] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2016. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. *CoRR* abs/1611.04717 (2016). arXiv:1611.04717

[23] Philip S. Thomas and Andrew G. Barto. 2011. Conjugate Markov Decision Processes. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. 137–144.

[24] Harm van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. 2017. Hybrid Reward Architecture for Reinforcement Learning.

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

[26] Steven D. Whitehead. 1991. Complexity and Cooperation in Q-Learning. In *Proceedings of the Eighth International Workshop (ML91), Northwestern University, Evanston, Illinois, USA*. 363–367.

[27] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*. 229–256.