Online Multi-Task Learning for Policy Gradient Methods

Haitham Bou Ammar HAITHAMB@SEAS.UPENN.EDU
Eric Eaton EEATON@CIS.UPENN.EDU
University of Pennsylvania, Computer and Information Science Department, Philadelphia, PA 19104 USA

Paul Ruvolo PAUL.RUVOLO@OLIN.EDU
Olin College of Engineering, Needham, MA 02492 USA

Matthew E. Taylor TAYLORM@EECS.WSU.EDU
Washington State University, School of Electrical Engineering and Computer Science, Pullman, WA 99164 USA

Abstract

Policy gradient algorithms have shown considerable recent success in solving high-dimensional sequential decision making tasks, particularly in robotics. However, these methods often require extensive experience in a domain to achieve high performance. To make agents more sample-efficient, we developed a multi-task policy gradient method to learn decision making tasks consecutively, transferring knowledge between tasks to accelerate learning. Our approach provides robust theoretical guarantees, and we show empirically that it dramatically accelerates learning on a variety of dynamical systems, including an application to quadrotor control.

1. Introduction

Sequential decision making (SDM) is an essential component of autonomous systems. Although significant progress has been made on developing algorithms for learning isolated SDM tasks, these algorithms often require a large amount of experience before achieving acceptable performance. This is particularly true in the case of high-dimensional SDM tasks that arise in robot control problems. The cost of this experience can be prohibitively expensive (in terms of both time and fatigue of the robot's components), especially in scenarios where an agent will face multiple tasks and must be able to quickly acquire control policies for each new task. Another failure mode of conventional methods is that when the production environment differs significantly from the training environment, previously learned policies may no longer be correct.

Proceedings of the 31^{st} *International Conference on Machine Learning*, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

When data is in limited supply, learning task models jointly through *multi-task learning* (MTL) rather than independently can significantly improve model performance (Thrun & O'Sullivan, 1996; Zhang et al., 2008; Rai & Daumé, 2010; Kumar & Daumé, 2012). However, MTL's performance gain comes at a high computational cost when learning new tasks or when updating previously learned models. Recent work (Ruvolo & Eaton, 2013) in the supervised setting has shown that nearly identical performance to batch MTL can be achieved in online learning with large computational speedups. Building upon this work, we introduce an online MTL approach to learn a sequence of SDM tasks with low computational overhead. Specifically, we develop an online MTL formulation of policy gradient reinforcement learning that enables an autonomous agent to accumulate knowledge over its lifetime and efficiently share this knowledge between SDM tasks to accelerate learning. We call this approach the *Policy Gradient Efficient Lifelong Learning Algorithm* (PG-ELLA)—the first (to our knowledge) online MTL policy gradient method.

Instead of learning a control policy for an SDM task from scratch, as in standard policy gradient methods, our approach rapidly learns a high-performance control policy based on the agent's previously learned knowledge. Knowledge is shared between SDM tasks via a latent basis that captures reusable components of the learned policies. The latent basis is then updated with newly acquired knowledge, enabling *a*) accelerated learning of new task models and *b*) improvement in the performance of existing models without retraining on their respective tasks. The latter capability is especially important in ensuring that the agent can accumulate knowledge over its lifetime across numerous tasks without exhibiting negative transfer. We show that this process is highly efficient with robust theoretical guarantees. We evaluate PG-ELLA on four dynamical systems, including an application to quadrotor control, and show that PG-ELLA outperforms standard policy gradients both in the initial and final performance.

2. Related Work in Multi-Task RL

Due to its empirical success, there is a growing body of work on transfer learning approaches to reinforcement learning (RL) (Taylor & Stone, 2009). By contrast, relatively few methods for *multi-task* RL have been proposed.

One class of algorithms for multi-task RL use nonparametric Bayesian models to share knowledge between tasks. For instance, Wilson et al. (2007) developed a hierarchical Bayesian approach that models the distribution over Markov decision processes (MDPs) and uses this distribution as a prior for learning each new task, enabling it to learn tasks consecutively. In contrast to our work, Wilson et al. focused on environments with discrete states and actions. Additionally, their method requires the ability to compute an optimal policy given an MDP. This process can be expensive for even moderately large discrete environments, but is computationally intractable for the types of continuous, high-dimensional control problems considered here. Another example is by Li et al. (2009), who developed a model-free multi-task RL method for partially observable environments. Unlike our problem setting, their method focuses on off-policy batch MTL. Finally, Lazaric & Ghavamzadeh (2010) exploit shared structure in the value functions between related MDPs. However, their approach is designed for on-policy multi-task policy evaluation, rather than computing optimal policies.

A second approach to multi-task RL is based on Policy Reuse (Fernández & Veloso, 2013), in which policies from previously learned tasks are probabilistically reused to bias the learning of new tasks. One drawback of Policy Reuse is that it requires that tasks share common states, actions, and transition functions (but allows different reward functions), while our approach only requires that tasks share a common state and action space. This restriction precludes the application of Policy Reuse to the scenarios considered in Section 7, where the systems have related but not identical transition functions. Also, in contrast to PG-ELLA, Policy Reuse does not support reverse transfer, where subsequent learning improves previously learned policies.

Perhaps the approach most similar to ours is by Deisenroth et al. (2014), which uses policy gradients to learn a single controller that is optimal on average over all training tasks. By appropriately parameterizing the policy, the controller can be customized to particular tasks. However, this method requires that tasks differ only in their reward function, and thus is inapplicable to our experimental scenarios.

3. Problem Framework

We first describe our framework for policy gradient RL and lifelong learning. The next section uses this framework to present our approach to online MTL for policy gradients.

3.1. Policy Gradient Reinforcement Learning

We frame each SDM task as an RL problem, in which an agent must sequentially select actions to maximize its expected return. Such problems are typically formalized as a Markov decision process (MDP) $\langle \mathcal{X}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is the (potentially infinite) set of states, $\mathcal{A} \subseteq \mathbb{R}^m$ is the set of possible actions, $P : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \mapsto [0,1]$ is a state transition probability function describing the system's dynamics, $R : \mathcal{X} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function measuring the agent's performance, and $\gamma \in [0,1)$ specifies the degree to which rewards are discounted over time. At each time step h, the agent is in state $\boldsymbol{x}_h \in \mathcal{X}$ and must choose an action $\boldsymbol{a}_h \in \mathcal{A}$, transitioning it to a new state $\boldsymbol{x}_{h+1} \sim p(\boldsymbol{x}_{h+1} \mid \boldsymbol{x}_h, \boldsymbol{a}_h)$ as given by P and yielding reward $r_{h+1} = R(\boldsymbol{x}_h, \boldsymbol{a}_h)$. A policy $\pi : \mathcal{X} \times \mathcal{A} \mapsto [0,1]$ is defined as a probability distribution over state-action pairs, where $\pi(\boldsymbol{a} \mid \boldsymbol{x})$ represents the probability of selecting action $\boldsymbol{a}$ in state $\boldsymbol{x}$. The goal of an RL agent is to find an optimal policy $\pi^\star$ that maximizes the expected return. The sequence of state-action pairs forms a trajectory $\boldsymbol{\tau} = [\boldsymbol{x}_{0:H}, \boldsymbol{a}_{0:H}]$ over a possibly infinite horizon H.

Policy gradient methods (Sutton et al., 1999; Peters & Schaal, 2008; Peters & Bagnell, 2010) have shown success in solving high-dimensional problems, such as robotic control (Peters & Schaal, 2007). These methods represent the policy $\pi_\theta(\boldsymbol{a} \mid \boldsymbol{x})$ using a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ of control parameters. The goal is to determine the optimal parameters $\boldsymbol{\theta}^\star$ that maximize the expected average return:

$$\mathcal{J}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\, \mathfrak{R}(\boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau} \ , \tag{1}$$

where $\mathbb{T}$ is the set of all possible trajectories. The trajectory distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ and average per time step return $\mathfrak{R}(\boldsymbol{\tau})$ are defined as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = P_0(\boldsymbol{x}_0) \prod_{h=1}^{H} p(\boldsymbol{x}_{h+1} \mid \boldsymbol{x}_h, \boldsymbol{a}_h)\, \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_h \mid \boldsymbol{x}_h)$$

$$\mathfrak{R}(\boldsymbol{\tau}) = \frac{1}{H} \sum_{h=0}^{H} r_{h+1} \ ,$$

with an initial state distribution $P_0 : \mathcal{X} \mapsto [0,1]$.

Most policy gradient algorithms, such as episodic REINFORCE (Williams, 1992), PoWER (Kober & Peters, 2011), and Natural Actor Critic (Peters & Schaal, 2008), employ supervised function approximators to learn the control parameters $\boldsymbol{\theta}$ by maximizing a lower bound on the expected return of $\mathcal{J}(\boldsymbol{\theta})$ (Eq. 1). To achieve this, one generates trajectories using the current policy $\pi_{\boldsymbol{\theta}}$, and then compares the result with a new policy parameterized by $\tilde{\boldsymbol{\theta}}$. As described by Kober & Peters (2011), the lower bound on the expected return can be attained using Jensen's inequality and the concavity of the logarithm:

$$\begin{aligned}\log \mathcal{J}\left(\tilde{\boldsymbol{\theta}}\right) &= \log \int_{\mathbb{T}} p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau})\, \mathfrak{R}(\boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau} \\ &= \log \int_{\mathbb{T}} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\, p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau})\, \mathfrak{R}(\boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau} \\ &\geq \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\, \mathfrak{R}(\boldsymbol{\tau}) \log \frac{p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\, \mathrm{d}\boldsymbol{\tau} + \text{constant} \\ &\propto -\mathfrak{D}_{\mathrm{KL}}\big(p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\, \mathfrak{R}(\boldsymbol{\tau})\, \|\, p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau})\big) = \mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}\right) \quad ,\end{aligned}$$

where $\mathfrak{D}_{\mathrm{KL}}\big(p(\boldsymbol{\tau}) \,\|\, q(\boldsymbol{\tau})\big) = \int_{\mathbb{T}} p(\boldsymbol{\tau}) \log \frac{p(\boldsymbol{\tau})}{q(\boldsymbol{\tau})}\, \mathrm{d}\boldsymbol{\tau}$. We see that this is equivalent to minimizing the KL divergence between the reward-weighted trajectory distribution of $\pi_{\boldsymbol{\theta}}$ and the trajectory distribution $p_{\tilde{\boldsymbol{\theta}}}$ of the new policy $\pi_{\tilde{\boldsymbol{\theta}}}$.

3.2. The Lifelong Learning Problem

In contrast to most previous work on policy gradients, which focus on single-task learning, this paper focuses on the online MTL setting in which the agent is required to learn a series of SDM tasks $\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T_{\max})}$ over its lifetime. Each task t is an MDP $\mathcal{Z}^{(t)} = \langle \mathcal{S}^{(t)}, \mathcal{A}^{(t)}, P^{(t)}, R^{(t)}, \gamma^{(t)} \rangle$ with initial state distribution $P_0^{(t)}$. The agent will learn the tasks consecutively, acquiring multiple trajectories within each task before moving to the next. The tasks may be interleaved, providing the agent the opportunity to revisit earlier tasks for further experience, but the agent has no control over the task order. We assume that *a priori* the agent does not know the total number of tasks $T_{\max}$, their distribution, or the task order.

The agent's goal is to learn a set of *optimal* policies $\boldsymbol{\Pi}^\star = \left\{\pi^\star_{\boldsymbol{\theta}^{(1)}}, \ldots, \pi^\star_{\boldsymbol{\theta}^{(T_{\max})}}\right\}$ with corresponding parameters $\boldsymbol{\Theta}^\star = \left\{\boldsymbol{\theta}^{(1)\star}, \ldots, \boldsymbol{\theta}^{(T_{\max})\star}\right\}$. At any time, the agent may be evaluated on any previously seen task, and so must strive to optimize its learned policies for all tasks $\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T)}$, where T denotes the number of tasks seen so far ($1 \leq T \leq T_{\max}$).

4. Online MTL for Policy Gradient Methods

This section develops the Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA).

4.1. Learning Objective

To share knowledge between tasks, we assume that each task's control parameters can be modeled as a linear combination of latent components from a shared knowledge base. A number of supervised MTL algorithms (Kumar & Daumé, 2012; Ruvolo & Eaton, 2013; Maurer et al., 2013) have shown this approach to be successful. Our approach incorporates the use of a shared latent basis into policy gradient learning to enable transfer between SDM tasks.

PG-ELLA maintains a library of k latent components $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ that is shared among all tasks, forming a basis for the control policies. We can then represent each task's control parameters as a linear combination of this latent basis $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, where $\boldsymbol{s}^{(t)} \in \mathbb{R}^k$ is a task-specific vector of coefficients. The task-specific coefficients $\boldsymbol{s}^{(t)}$ are encouraged to be sparse to ensure that each learned basis component captures a maximal reusable chunk of knowledge.

We can then represent our objective of learning T stationary policies while maximizing the amount of transfer between task models by:

$$\boldsymbol{e}_{\mathrm{T}}(\boldsymbol{L}) = \frac{1}{T}\sum_{t=1}^{T} \min_{\boldsymbol{s}^{(t)}} \left[-\mathcal{J}\left(\boldsymbol{\theta}^{(t)}\right) + \mu \left\|\boldsymbol{s}^{(t)}\right\|_1\right] + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \;, \quad (2)$$

where $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, the L_1 norm of $\boldsymbol{s}^{(t)}$ is used to approximate the true vector sparsity, and $\|\cdot\|_{\mathsf{F}}$ is the Frobenius norm. The form of this objective function is closely related to other supervised MTL methods (Ruvolo & Eaton, 2013; Maurer et al., 2013), with important differences through the incorporation of $\mathcal{J}(\cdot)$ as we will examine shortly.

Our approach to optimizing Eq. (2) is based upon the Efficient Lifelong Learning Algorithm (ELLA) (Ruvolo & Eaton, 2013), which provides a computationally efficient method for learning $\boldsymbol{L}$ and the $\boldsymbol{s}^{(t)}$'s online over multiple tasks in the case of supervised MTL. The objective solved by ELLA is closely related to Eq. (2), with the exception that the $-\mathcal{J}(\cdot)$ term is replaced with a measure of each task model's average loss over the training data in ELLA. Since Eq. (2) is not jointly convex in $\boldsymbol{L}$ and the $\boldsymbol{s}^{(t)}$'s, most supervised MTL methods use an expensive alternating optimization procedure to train the task models simultaneously. Ruvolo & Eaton provide an efficient alternative to this procedure that can train task models consecutively, enabling Eq. (2) to be used effectively for online MTL. In the next section, we adapt this approach to the policy gradient framework, and show that the resulting algorithm provides an efficient method for learning consecutive SDM tasks.

4.2. Multi-Task Policy Gradients

Policy gradient methods maximize the lower bound of $\mathcal{J}(\boldsymbol{\theta})$ (Eq. 1). In order to use Eq. (2) for MTL with policy gradients, we must first incorporate this lower bound into our objective function. Rewriting the error term in Eq. (2) in terms of the lower bound yields

$$\boldsymbol{e}_{\mathrm{T}}(\boldsymbol{L}) = \frac{1}{T}\sum_{t=1}^{T} \min_{\boldsymbol{s}^{(t)}} \left[-\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right) + \mu \left\|\boldsymbol{s}^{(t)}\right\|_1\right] + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \;,$$

where $\tilde{\boldsymbol{\theta}}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$. However, we can note that

$$\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right) \propto -\int_{\boldsymbol{\tau} \in \mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\, \mathfrak{R}^{(t)}(\boldsymbol{\tau}) \log \left(\frac{p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\, \mathfrak{R}^{(t)}(\boldsymbol{\tau})}{p_{\tilde{\boldsymbol{\theta}}^{(t)}}(\boldsymbol{\tau})}\right) \mathrm{d}\boldsymbol{\tau} \,.$$

Therefore, maximizing the lower bound of $\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right)$ is equivalent to the following minimization problem:

$$\min_{\tilde{\boldsymbol{\theta}}^{(t)}}\left\{\int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\log\left(\frac{p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})}{p_{\tilde{\boldsymbol{\theta}}^{(t)}}(\boldsymbol{\tau})}\right)\mathrm{d}\boldsymbol{\tau}\right\} \ .$$

Substituting the above result with $\tilde{\boldsymbol{\theta}}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$ into Eq. (2) leads to the following *total* cost function for MTL with policy gradients:

$$\begin{aligned}\boldsymbol{e}_T(\boldsymbol{L}) = \frac{1}{T}\sum_{t=1}^{T}\min_{\boldsymbol{s}^{(t)}}\Bigg\{\Bigg[\int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\\ \log\left(\frac{p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})}{p_{\tilde{\boldsymbol{\theta}}^{(t)}}(\boldsymbol{\tau})}\right)\mathrm{d}\boldsymbol{\tau}\Bigg] + \mu\left\|\boldsymbol{s}^{(t)}\right\|_1\Bigg\} + \lambda\|\boldsymbol{L}\|_{\mathsf{F}}^2 \ .\end{aligned} \tag{3}$$

While Eq. (3) enables batch MTL using policy gradients, it is computationally expensive due to two inefficiencies that make it inappropriate for online MTL: *a*) the explicit dependence on *all* available trajectories through $\mathcal{J}(\boldsymbol{\theta}^{(t)}) = \int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\ \mathrm{d}\boldsymbol{\tau}$, and *b*) the exhaustive evaluation of a single candidate $\boldsymbol{L}$ that requires the optimization of all $\boldsymbol{s}^{(t)}$'s through the outer summation. Together, these aspects cause Eq. (3) (and similarly Eq. (2)) to have a computational cost that depends on the total number of trajectories and total number of tasks T, complicating its direct use in the lifelong learning setting.

We next describe methods for resolving each of these inefficiencies while minimizing Eq. (3), yielding PG-ELLA as an efficient method for multi-task policy gradient learning. In fact, we show that the complexity of PG-ELLA in learning a single task policy is independent of *a*) the number of tasks seen so far and *b*) the number of trajectories for all other tasks, allowing our approach to be highly efficient.

4.2.1. Eliminating dependence on other tasks

As mentioned above, one of the inefficiencies in minimizing $\boldsymbol{e}_T(\boldsymbol{L})$ is its dependence on all available trajectories for all tasks. To remedy this problem, as in ELLA, we approximate $\boldsymbol{e}_T(\boldsymbol{L})$ by performing a second-order Taylor expansion of $\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}(\tilde{\boldsymbol{\theta}}^{(t)})$ around the optimal solution:

$$\begin{aligned}\boldsymbol{\alpha}^{(t)} = \arg\min_{\tilde{\boldsymbol{\theta}}^{(t)}}\Bigg\{\int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\\ \log\left(\frac{p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})}{p_{\tilde{\boldsymbol{\theta}}^{(t)}}(\boldsymbol{\tau})}\right)\mathrm{d}\boldsymbol{\tau}\Bigg\} \ .\end{aligned}$$

As shown by Ruvolo & Eaton (2013), the second-order Taylor expansion can be substituted into the MTL objective function to provide a point estimate around the optimal solution, eliminating the dependence on other tasks.

To compute the second-order Taylor representation, the first and second derivatives of $\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right)$ w.r.t. $\tilde{\boldsymbol{\theta}}^{(t)}$ are required. The first derivative, $\nabla_{\tilde{\boldsymbol{\theta}}^{(t)}}\ \mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right)$, is given by:

$$-\int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\ \nabla_{\tilde{\boldsymbol{\theta}}^{(t)}}\log p_{\tilde{\boldsymbol{\theta}}^{(t)}}(\boldsymbol{\tau})\,\mathrm{d}\boldsymbol{\tau}$$

with:

$$\begin{aligned}\log p_{\tilde{\boldsymbol{\theta}}^{(t)}}(\boldsymbol{\tau}) = \log p^{(t)}\left(\boldsymbol{x}_0^{(t)}\right) + \sum_{h=0}^{H^{(t)}} p^{(t)}\left(\boldsymbol{x}_{h+1}^{(t)} \mid \boldsymbol{x}_h^{(t)}, \boldsymbol{a}_h^{(t)}\right)\\ + \sum_{h=0}^{H^{(t)}} \log\pi_{\tilde{\boldsymbol{\theta}}^{(t)}}\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right) \ .\end{aligned}$$

Therefore:

$$\begin{aligned}\nabla_{\tilde{\boldsymbol{\theta}}^{(t)}}\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right) &= -\int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\\ &\qquad\left[\sum_{h=1}^{H^{(t)}}\nabla_{\tilde{\boldsymbol{\theta}}^{(t)}}\log\pi_{\tilde{\boldsymbol{\theta}}^{(t)}}\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right)\right]\mathrm{d}\boldsymbol{\tau}\\ &= -\mathbb{E}\left[\left(\sum_{h=1}^{H^{(t)}}\nabla_{\tilde{\boldsymbol{\theta}}^{(t)}}\log\pi_{\tilde{\boldsymbol{\theta}}^{(t)}}\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right)\right)\mathfrak{R}^{(t)}(\boldsymbol{\tau})\right] \ .\end{aligned}$$

Policy gradient algorithms determine $\boldsymbol{\alpha}^{(t)} = \tilde{\boldsymbol{\theta}}^{(t)\star}$ by following the above gradient. The second derivative of $\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right)$ can be computed similarly to produce:

$$\begin{aligned}\nabla^2_{\tilde{\boldsymbol{\theta}}^{(t)},\tilde{\boldsymbol{\theta}}^{(t)}}\ \mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right) = -\int_{\boldsymbol{\tau}\in\mathbb{T}^{(t)}} p_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{\tau})\,\mathfrak{R}^{(t)}(\boldsymbol{\tau})\\ \left[\sum_{h=1}^{H^{(t)}}\nabla^2_{\tilde{\boldsymbol{\theta}}^{(t)},\tilde{\boldsymbol{\theta}}^{(t)}}\log\pi_{\tilde{\boldsymbol{\theta}}^{(t)}}\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right)\right]\mathrm{d}\boldsymbol{\tau} \ .\end{aligned}$$

We let $\boldsymbol{\Gamma}^{(t)} = \nabla^2_{\tilde{\boldsymbol{\theta}}^{(t)},\tilde{\boldsymbol{\theta}}^{(t)}}\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right)$ represent the Hessian evaluated at $\boldsymbol{\alpha}^{(t)}$:

$$\boldsymbol{\Gamma}^{(t)} = -\mathbb{E}\left[\mathfrak{R}^{(t)}(\boldsymbol{\tau})\sum_{h=1}^{H^{(t)}}\nabla^2_{\tilde{\boldsymbol{\theta}}^{(t)},\tilde{\boldsymbol{\theta}}^{(t)}}\log\pi_{\tilde{\boldsymbol{\theta}}^{(t)}}\left(\boldsymbol{a}_h^{(t)}\middle|\boldsymbol{x}_h^{(t)}\right)\right]_{\tilde{\boldsymbol{\theta}}^{(t)}=\boldsymbol{\alpha}^{(t)}} .$$

Substituting the second-order Taylor approximation into Eq. (3) yields the following:

$$\begin{aligned}\hat{\boldsymbol{e}}_T(\boldsymbol{L}) = \frac{1}{T}\sum_{t=1}^{T}\min_{\boldsymbol{s}^{(t)}}\left[\left\|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}^{(t)}\right\|^2_{\boldsymbol{\Gamma}^{(t)}} + \mu\left\|\boldsymbol{s}^{(t)}\right\|_1\right]\\ + \lambda\|\boldsymbol{L}\|_{\mathsf{F}}^2 \ ,\end{aligned} \tag{4}$$

where $\|\boldsymbol{v}\|^2_{\boldsymbol{A}} = \boldsymbol{v}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{v}$, the constant term was suppressed since it has no effect on the minimization, and the linear term was ignored since by construction $\boldsymbol{\alpha}^{(t)}$ is a minimizer. Most importantly, the dependence on all available trajectories has been eliminated, remedying the first inefficiency.

4.2.2. Computing the Latent Space

The second inefficiency in Eq. (3) arises from the procedure used to compute the objective function for a single candidate $\boldsymbol{L}$. Namely, to determine how effective a given value of $\boldsymbol{L}$ serves as a common basis for all learned tasks, an optimization problem must be solved to recompute each of the $\boldsymbol{s}^{(t)}$'s, which becomes increasingly expensive as T grows large. To remedy this problem, we modify Eq. (3) (or equivalently, Eq. (4)) to eliminate the minimization over all $\boldsymbol{s}^{(t)}$'s. Following the approach used in ELLA, we optimize each task-specific projection $\boldsymbol{s}^{(t)}$ only when training on task t, without updating them when training on other tasks. Consequently, any changes to $\boldsymbol{\theta}^{(t)}$ when learning on other tasks will only be through updates to the shared basis $\boldsymbol{L}$. As shown by Ruvolo & Eaton (2013), this choice to update $\boldsymbol{s}^{(t)}$ only when training on task t does not significantly affect the quality of model fit as T grows large.

With this simplification, we can rewrite Eq. (4) in terms of two update equations:

$$\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \ell\left(\boldsymbol{L}_m, \boldsymbol{s}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\Gamma}^{(t)}\right) \tag{5}$$

$$\boldsymbol{L}_{m+1} \leftarrow \arg\min_{\boldsymbol{L}} \frac{1}{T}\sum_{t=1}^{T} \ell\left(\boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\Gamma}^{(t)}\right) + \lambda\|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \tag{6}$$

where $\boldsymbol{L}_m$ refers to the value of the latent basis at the start of the m^{th} training session, t corresponds to the particular task for which data was just received, and

$$\ell\left(\boldsymbol{L}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma}\right) = \mu\|\boldsymbol{s}\|_1 + \|\boldsymbol{\alpha} - \boldsymbol{L}\boldsymbol{s}\|_{\boldsymbol{\Gamma}}^2 \ .$$

To compute $\boldsymbol{L}_m$, we null the gradient of Eq. (6) and solve the resulting equation to yield the updated column-wise vectorization of $\boldsymbol{L}$ as $\boldsymbol{A}^{-1}\boldsymbol{b}$, where:

$$\boldsymbol{A} = \lambda \boldsymbol{I}_{d\times k, d\times k} + \frac{1}{T}\sum_{t=1}^{T}\left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{\Gamma}^{(t)}$$

$$\boldsymbol{b} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{vec}\left(\boldsymbol{s}^{(t)\top} \otimes \left(\boldsymbol{\alpha}^{(t)\top}\boldsymbol{\Gamma}^{(t)}\right)\right) \ .$$

For efficiency, we can compute $\boldsymbol{A}$ and $\boldsymbol{b}$ incrementally as new tasks arrive, avoiding the need to sum over all tasks.

4.3. Data Generation & Model Update

Using the incremental form (Eqs. 5–6) of the policy gradient MTL objective function (Eq. 3), we can now construct an online MTL algorithm that can operate in a lifelong learning setting. In typical policy gradient methods, trajectories are generated in batch mode by first initializing the policy and sampling trajectories from the system (Kober & Peters, 2011; Peters & Bagnell, 2010). Given these trajectories, the policy parameters are updated, new trajectories are sampled from the system using the updated policy, and the procedure is then repeated. In this work, we adopt a slightly modified version of policy gradients to operate in the lifelong learning setting. The first time a new task is observed, we use a random policy for sampling; each subsequent time the task is observed, we sample trajectories using the previously learned $\boldsymbol{\alpha}^{(t)}$. Additionally, instead of looping until the policy parameters have converged, we perform only one run over the trajectories.

Algorithm 1 PG-ELLA (k, λ, μ)

```
$T \leftarrow 0$,   $\mathbf{A} \leftarrow \mathbf{zeros}_{k\times d, k\times d}$,
$\mathbf{b} \leftarrow \mathbf{zeros}_{k\times d, 1}$,   $\mathbf{L} \leftarrow \mathbf{zeros}_{d,k}$
while some task $t$ is available do
    if isNewTask($t$) then
        $T \leftarrow T + 1$
        $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow$ getRandomTrajectories()
    else
        $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow$ getTrajectories $\left(\boldsymbol{\alpha}^{(t)}\right)$
        $\mathbf{A} \leftarrow \mathbf{A} - \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
        $\mathbf{b} \leftarrow \mathbf{b} - \mathrm{vec}\left(\boldsymbol{s}^{(t)\top} \otimes \left(\boldsymbol{\alpha}^{(t)\top}\boldsymbol{\Gamma}^{(t)}\right)\right)$
    end if
    Compute $\alpha^{(t)}$ and $\boldsymbol{\Gamma}^{(\boldsymbol{t})}$ from $\left(\mathbb{T}^{(t)}, R^{(t)}\right)$
    $\mathbf{L} \leftarrow$ reinitializeAllZeroColumns($\mathbf{L}$)
    $\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \ell\left(\boldsymbol{L}, \boldsymbol{s}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\Gamma}^{(t)}\right)$
    $\mathbf{A} \leftarrow \mathbf{A} + \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
    $\mathbf{b} \leftarrow \mathbf{b} + \mathrm{vec}\left(\boldsymbol{s}^{(t)\top} \otimes \left(\boldsymbol{\alpha}^{(t)\top}\boldsymbol{\Gamma}^{(t)}\right)\right)$
    $\mathbf{L} \leftarrow \mathrm{mat}\left(\left(\frac{1}{T}\mathbf{A} + \lambda\mathbf{I}_{k\times d, k\times d}\right)^{-1} \frac{1}{T}\mathbf{b}\right)$
end while
```

Upon receiving data for a specific task t, PG-ELLA performs two steps to update the model: it first computes the task-specific projections $\boldsymbol{s}^{(t)}$, and then refines the shared latent space $\boldsymbol{L}$. To compute $\boldsymbol{s}^{(t)}$, we first determine $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$ using only data from task t. The details of this step depend on the form chosen for the policy, as described in Section 5. We can then solve the L_1-regularized regression problem given in Eq. (5)—an instance of the Lasso—to yield $\boldsymbol{s}^{(t)}$. In the second step, we update $\boldsymbol{L}$ by first reinitializing any zero-columns of $\boldsymbol{L}$ and then following Eq. (6). The complete PG-ELLA is given as Algorithm 1.

5. Policy Forms & Base Learners

PG-ELLA supports a variety of policy forms and base learners, enabling it to be used in a number of policy gradient settings. This section describes how two popular policy gradient methods can be used as the base learner in PG-ELLA. In theory, any policy gradient learner that can provide an estimate of the Hessian can be incorporated.

5.1. Episodic REINFORCE

In episodic REINFORCE (Williams, 1992), the stochastic policy for task t is chosen according $\boldsymbol{a}^{(t)} = \boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{x}_h^{(t)} + \boldsymbol{\epsilon}_h$, with $\boldsymbol{\epsilon}_h \sim \mathcal{N}(0, \sigma^2)$, and so $\pi\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right) \sim \mathcal{N}\left(\boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{x}_h^{(t)}, \sigma^2\right)$. Therefore,

$$\nabla_{\tilde{\boldsymbol{\theta}}^{(t)}}\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right) = -\mathbb{E}\left[\mathfrak{R}^{(t)}(\boldsymbol{\tau})\sum_{h=1}^{H^{(t)}}\sigma^{-2}\left(\boldsymbol{a}^{(t)} - \boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}_h^{(t)}\right)\right]$$

is used to minimize the KL-divergence, equivalently maximizing the total discounted pay-off. The second derivative for episodic REINFORCE is given by $\boldsymbol{\Gamma}^{(t)} = \mathbb{E}\left[\sum_{h=1}^{H^{(t)}}\sigma^{-2}\boldsymbol{x}_h^{(t)}\boldsymbol{x}_h^{(t)\mathsf{T}}\right]$.

5.2. Natural Actor Critic

In episodic Natural Actor Critic (eNAC), the stochastic policy for task t is chosen in a similar fashion to that of REINFORCE: $\pi\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right) \sim \mathcal{N}\left(\boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{x}_h^{(t)}, \sigma^2\right)$. The change in the probability distribution is measured by a KL-divergence that is approximated using a second-order expansion to incorporate the Fisher information matrix. Accordingly, the gradient follows: $\tilde{\nabla}_{\boldsymbol{\theta}}\mathcal{J} = \boldsymbol{G}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})$, where $\boldsymbol{G}$ denotes the Fisher information matrix. The Hessian can be computed in a similar manner to the previous section. For details, see Peters & Schaal (2008).

6. Theoretical Results & Computational Cost

Here, we provide theoretical results that establish that PG-ELLA converges and that the cost (in terms of model performance) for making the simplification from Section 4.2.1 is asymptotically negligible. We proceed by first stating theoretical results from Ruvolo & Eaton (2013), and then show that these theoretical results apply directly to PG-ELLA with minimal modifications. First, we define:

$$\hat{g}_T(\boldsymbol{L}) = \frac{1}{T}\sum_{t=1}^{T}\ell\left(\boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\Gamma}^{(t)}\right) + \lambda\|\boldsymbol{L}\|_{\mathsf{F}}^2 \ .$$

Recall from Section 4.2.1, that the lefthand side of the preceding equation specifies the cost of basis $\boldsymbol{L}$ if we leave the $\boldsymbol{s}^{(t)}$'s fixed (i.e., we only update them when we receive training data for that particular task). We are now ready to state the two results from Ruvolo & Eaton (2013):

Proposition 1: The latent basis becomes more stable over time at a rate of $\boldsymbol{L}_{T+1} - \boldsymbol{L}_T = O\left(\frac{1}{T}\right)$.

Proposition 2:
1. $\hat{g}_T(\mathbf{L}_T)$ converges almost surely;
2. $\hat{g}_T(\mathbf{L}_T) - e_T(\mathbf{L}_T)$ converges almost surely to 0.

Proposition 2 establishes that the algorithm converges to a fixed per-task loss on the approximate objective function $\hat{g}_T$ and the objective function that does not contain the simplification from Section 4.2.1. Further, Prop. 2 establishes that these two functions converge to the same value. The consequence of this last point is that PG-ELLA does not incur any penalty (in terms of average per-task loss) for making the simplification from Section 4.2.1.

The two propositions require the following assumptions:

1. The tuples $\left(\boldsymbol{\Gamma}^{(t)}, \boldsymbol{\alpha}^{(t)}\right)$ are drawn $i.i.d.$ from a distribution with compact support (bounding the entries of $\boldsymbol{\Gamma}^{(t)}$ and $\boldsymbol{\alpha}^{(t)}$).
2. For all $\boldsymbol{L}, \boldsymbol{\Gamma}^{(t)}$, and $\boldsymbol{\alpha}^{(t)}$, the smallest eigenvalue of $\boldsymbol{L}_\gamma^\top\boldsymbol{\Gamma}^{(t)}\boldsymbol{L}_\gamma$ is at least κ (with $\kappa > 0$), where γ is the subset of non-zero indices of the vector $\boldsymbol{s}^{(t)} = \arg\min_{\mathbf{s}}\|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}\|_{\boldsymbol{\Gamma}^{(t)}}^2$. In this case the non-zero elements of the unique minimizing $\boldsymbol{s}^{(t)}$ are given by: $\boldsymbol{s}_\gamma^{(t)} = \left(\boldsymbol{L}_\gamma^\top\boldsymbol{\Gamma}^{(t)}\boldsymbol{L}_\gamma\right)^{-1}\left(\boldsymbol{L}_\gamma^\top\boldsymbol{\Gamma}^{(t)}\boldsymbol{\alpha}^{(t)} - \mu\boldsymbol{\epsilon}_\gamma\right)$, where $\boldsymbol{\epsilon}_\gamma$ is a vector containing the signs of the non-zero entries of $\boldsymbol{s}^{(t)}$.

The second assumption is a mild condition on the uniqueness of the sparse coding solution. The first assumption can be verified by assuming that there is no sequential dependency of one task to the next. Additionally, the fact that $\boldsymbol{\Gamma}^{(t)}$ is contained in a compact region can be verified for the episodic REINFORCE algorithm by looking at the form of the Hessian and requiring that the time horizon $H^{(t)}$ is finite. Using a similar argument we can see that the magnitude of the gradient for episodic REINFORCE is also bounded when $H^{(t)}$ is finite. If we then assume that we make a finite number of updates for each task model we can ensure that the sum of all gradient updates is finite, thus guaranteeing that $\boldsymbol{\alpha}^{(t)}$ is contained in a compact region.

Computational Complexity: Each update begins by running a step of policy gradient to update $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$. We assume that the cost of the policy gradient update is $O(\xi(d, n_t))$, where the specific cost depends on the particular policy algorithm employed and n_t is the number of trajectories obtained for task t at the current iteration. To complete the analysis, we use a result from Ruvolo & Eaton (2013) that the cost of updating $\boldsymbol{L}$ and $\boldsymbol{s}^{(t)}$ is $O(k^2d^3)$. This gives an overall cost of $O(k^2d^3 + \xi(d, n_t))$ for each update.

7. Evaluation

We applied PG-ELLA to learn control policies for the four dynamical systems shown in Figure 1, including three mechanical systems and an application to quadrotor control. We generated multiple tasks by varying the parameterization of each system, yielding a set of tasks from each domain with varying dynamics. For example, the simple mass spring damper system exhibits significantly higher oscillations as the spring constant increases. Notably, the opti-

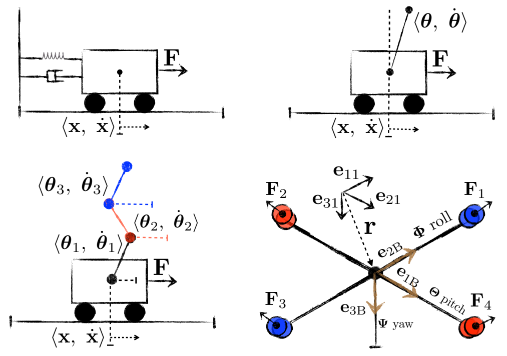


Figure 1. The four dynamical systems: *a*) simple mass spring damper (top-left), *b*) cart-pole (top-right), *c*) three-link inverted pendulum (bottom-left), and *d*) quadrotor (bottom-right).

mal policies for controlling these systems vary significantly even for only slight variations in the system parameters.

7.1. Benchmark Dynamical Systems

We evaluated PG-ELLA on three benchmark dynamical systems. In each domain, the distance between the current state and the goal position was used as the reward function.

Simple Mass Spring Damper: The simple mass (SM) system is characterized by three parameters: the spring constant k in N/m, the damping constant d in Ns/m, and the mass m in kg. The system's state is given by the position $\boldsymbol{x}$ and velocity $\dot{\boldsymbol{x}}$ of the mass, which vary according to a linear force $\boldsymbol{F}$. The goal is to design a policy for controlling the mass to be in a specific state $\boldsymbol{g}_{\text{ref}} = \langle \boldsymbol{x}_{\text{ref}}, \dot{\boldsymbol{x}}_{\text{ref}} \rangle$. In our experiments, the goal state varied from being $\boldsymbol{g}_{\text{ref}} = \langle 0,\ 0 \rangle$ to $\boldsymbol{g}_{\text{ref}}^{(i)} = \langle i,\ 0 \rangle$, where $i \in \{1, 2, \ldots, 5\}$.

Cart-Pole: The cart-pole (CP) system has been used extensively as a benchmark for evaluating RL algorithms (Buşoniu et al., 2010). CP dynamics are characterized by the cart's mass m_c in kg, the pole's mass m_p in kg, the pole's length l in meters, and a damping parameter d in Ns/m. The state is characterized by the position $\boldsymbol{x}$ and velocity $\dot{\boldsymbol{x}}$ of the cart, as well as the angle $\boldsymbol{\theta}$ and angular velocity $\dot{\boldsymbol{\theta}}$ of the pole. The goal is to design a policy capable of controlling the pole in an upright position.

Three-Link Inverted Pendulum: The three-link CP (3CP) is a highly nonlinear and difficult system to control. The goal is to balance three connected rods in an upright position by moving the cart. The dynamics are parameterized by the mass of the cart m_c, rod mass $m_{p,i}$, length l_i, inertia I_i, and damping parameters d_i, where $i \in \{1, 2, 3\}$ represents the index for each of the three rods. The system's state is characterized by an eight-dimensional vector, consisting of the position $\boldsymbol{x}$ and velocity $\dot{\boldsymbol{x}}$ of the cart, and the angle $\{\boldsymbol{\theta}_i\}_{i=1}^{3}$ and angular velocity $\{\dot{\boldsymbol{\theta}}_i\}_{i=1}^{3}$ of each rod.

Table 1. System parameter ranges used in the experiments.

SM	CP & 3CP	3CP	3CP
$k \in [1, 10]$	$m_c \in [0.5, 1.5]$	$l_1 \in [0.3, 0.5]$	$d_1 \in [0.1, 0.2]$
$d \in [0.01, 0.2]$	$m_p \in [0.1, 0.2]$	$l_2 \in [0.2, 0.4]$	$d_2 \in [0.01, 0.02]$
$m \in [0.5, 5]$	$l \in [0.2, 0.8]$	$l_3 \in [0.1, 0.3]$	$d_3 \in [0.1, 0.2]$
	$d \in [0.01, 0.09]$		$I_i \in [10^{-6}, 10^{-4}]$

7.1.1. Experimental Protocol

We first generated 30 tasks for each domain by varying the system parameters over the ranges given in Table 1. These parameter ranges were chosen to ensure a variety of tasks, including those that were difficult to control with highly chaotic dynamics. We then randomized the task order with repetition and PG-ELLA acquired a limited amount of experience in each task consecutively, updating $\boldsymbol{L}$ and the $\boldsymbol{s}^{(t)}$'s after each session. At each learning session, PG-ELLA was limited to 50 trajectories (for SM & CP) or 20 trajectories (for 3CP) with 150 time steps each to perform the update. Learning ceased once PG-ELLA had experienced at least one session with each task.

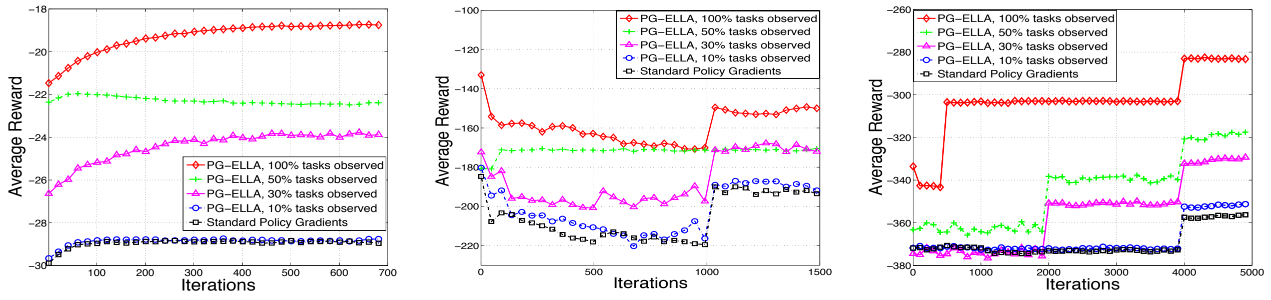
To configure PG-ELLA, we used eNAC (Peters & Schaal, 2008) as the base policy gradient learner. The dimensionality k of the latent basis $\boldsymbol{L}$ was chosen independently for each domain via cross-validation over 10 tasks. The stepsize for each task domain was determined by a line search after gathering 10 trajectories of length 150.

To evaluate the learned basis at any point in time, we initialized policies for each task using $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$ for $t = \{1, \ldots, T\}$. Starting from these initializations, learning on each task commenced using eNAC. The number of trajectories varied among the domains from a minimum of 20 on the simple mass system to a maximum of 50 on the quadrotors. The length of each of these trajectories was set to 150 time steps across all domains. We measured performance using the average reward computed over 50 episodes of 150 time steps, and compared this to standard eNAC running independently with the same settings.

7.1.2. Results on the Benchmark Systems

Figure 2 compares PG-ELLA to standard policy gradient learning using eNAC, showing the average performance on all tasks versus the number of learning iterations. PG-ELLA clearly outperforms standard eNAC in both the initial and final performance on all task domains, demonstrating significantly improved performance from MTL.

We evaluated PG-ELLA's performance on all tasks using the basis $\boldsymbol{L}$ learned after observing various subsets of tasks, from observing only three tasks (10%) to observing all 30 tasks (100%). These experiments assessed the quality of the learned basis $\boldsymbol{L}$ on both known as well as unknown tasks, showing that performance increases as PG-ELLA



(a) Simple Mass Spring Damper (b) Cart-Pole (c) Three-Link Inverted Pendulum

Figure 2. The performance of PG-ELLA versus standard policy gradients (eNAC) on the benchmark dynamical systems.

learns more tasks. When a particular task was not observed, the recent $\boldsymbol{L}$ with a zero initialization of $\boldsymbol{s}^{(t)}$ was used.

To assess the difference in total number of trajectories between PG-ELLA and eNAC, we also tried giving eNAC an additional 50 trajectories of length 150 time steps at each iteration. However, its overall performance did not change.

7.2. Quadrotor Control

We also evaluated PG-ELLA on an application to quadrotor control, providing a more challenging domain. The quadrotor system is illustrated in Figure 1, with dynamics influenced by inertial constants around $\boldsymbol{e}_{1,\text{B}}$, $\boldsymbol{e}_{2,\text{B}}$, and $\boldsymbol{e}_{3,\text{B}}$, thrust factors influencing how the rotor's speed affects the overall variation of the system's state, and the length of the rods supporting the rotors. Although the overall state of the system can be described by a nine-dimensional vector, we focus on stability and so consider only six of these state variables. The quadrotor system has a high-dimensional action space, where the goal is control the four rotational velocities $\{w_i\}_{i=1}^4$ of the rotors to stabilize the system. To ensure realistic dynamics, we used the simulated model described by Bouabdallah (2007), which has been verified on and used in the control of a physical quadrotor.

To produce multiple tasks, we generated 15 quadrotor systems by varying each of: the inertia around the x-axis $I_{xx} \in [4.5e^{-3}, 6.5e^{-3}]$, inertia around the y-axis $I_{yy} \in [4.2e^{-3}, 5.2e^{-3}]$, inertia around the z-axis $I_{zz} \in [1.5e^{-2}, 2.1e^{-2}]$, and the length of the arms $l \in [0.27, 0.3]$. In each case, these parameter values have been used by Bouabdallah (2007) to describe physical quadrotors. We used a linear quadratic regulator, as described by Bouabdallah, to initialize the policies in both the learning (i.e., determining $\boldsymbol{L}$ and $\boldsymbol{s}^{(t)}$) and testing (i.e., comparing to standard policy gradients) phases. We followed a similar experimental procedure to evaluate PG-ELLA on quadrotor control, where we used 50 trajectories of 150 time steps to perform an eNAC policy gradient update each learning session.

Figure 3 compares PG-ELLA to standard policy gradients (eNAC) on quadrotor control. As on the benchmark systems, we see that PG-ELLA clearly outperforms standard eNAC in both the initial and final performance, and this performance increases as PG-ELLA learns more tasks. The final performance of the policy learned by PG-ELLA after observing all tasks is significantly better than the policy learned using standard policy gradients, showing the benefits of knowledge transfer between tasks. Most importantly for practical applications, by using the basis $\boldsymbol{L}$ learned over previous tasks, PG-ELLA can achieve high performance in a new task much more quickly (with fewer trajectories) than standard policy gradient methods.

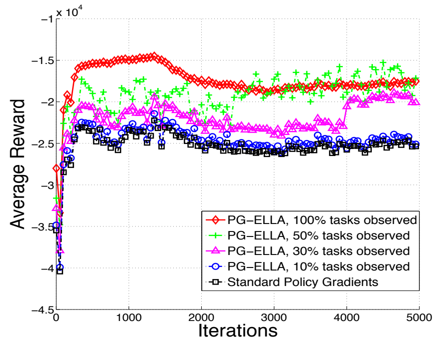


Figure 3. Performance on quadrotor control.

8. Conclusion & Future Work

PG-ELLA provides an efficient mechanism for online MTL of SDM tasks while providing improved performance over standard policy gradient methods. By supporting knowledge transfer between tasks via a shared latent basis, PG-ELLA is also able to rapidly learn policies for new tasks, providing the ability for an agent to rapidly adapt to new situations. In future work, we intend to explore the potential for cross-domain transfer with PG-ELLA.

Acknowledgements

This work was partially supported by ONR N00014-11-1-0139, AFOSR FA8750-14-1-0069, and NSF IIS-1149917. We thank the reviewers for their helpful suggestions.

References

Bócsi, B., Csato, L., and Peters, J. Alignment-based transfer learning for robot models. In *Proceedings of the 2013 International Joint Conference on Neural Networks* (IJCNN), 2013.

Bou-Ammar, H., Taylor, M.E., Tuyls, K., Driessens, K., and Weiss, G. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th Conference on Autonomous Agents and Multiagent Systems* (AAMAS), 2012.

Bouabdallah, S. *Design and control of quadrotors with application to autonomous flying*. PhD thesis, École polytechnique fédérale de Lausanne, 2007.

Buşoniu, L., Babuška, R., De Schutter, B., and Ernst, D. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida, 2010.

Daniel, C., Neumann, G., Kroemer, O., and Peters, J. Learning sequential motor tasks. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation* (ICRA), 2013.

Deisenroth, M.P., Englert, P., Peters, J., and Fox, D. Multi-task policy search for robotics In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation* (ICRA), 2014.

Fernández, F. and Veloso, M. Learning domain structure through probabilistic policy reuse in reinforcement learning. *Progress in AI*, 2(1):13–27, 2013.

Kober, J. and Peters, J. Policy search for motor primitives in robotics. *Machine Learning*, 84(1–2), July 2011.

Kumar, A. and Daumé III, H. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Conference on Machine Learning* (ICML), 2012.

Kupcsik, A.G., Deisenroth, M.P., Peters, J., and Neumann, G. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence* (AAAI), 2013.

Lazaric, A. and Ghavamzadeh, M. Bayesian multi-task reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning* (ICML), 2010.

Li, H., Liao, X., and Carin, L. Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research*, 10:1131–1186, 2009.

Liu, Y. and Stone, P. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the 21st National Conference on Artificial Intelligence* (AAAI), pp. 415–420, 2006.

Maurer, A., Pontil, M., and Romera-Paredes, B. Sparse coding for multitask and transfer learning. In *Proceedings of the 30th International Conference on Machine Learning* (ICML), 2013.

Peters, J. and Bagnell, J.A. Policy gradient methods. *Encyclopedia of Machine Learning*, pp. 774–776, 2010.

Peters, J. and Schaal, S. Applying the episodic natural actor-critic architecture to motor primitive learning. In *Proceedings of the 2007 European Symposium on Artificial Neural Networks* (ESANN), 2007.

Peters, J. and Schaal, S. Natural actor-critic. *Neurocomputing*, 71 (7-9):1180–1190, 2008.

Rai, P. and Daumé III, H. Infinite predictor subspace models for multitask learning. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence* (UAI), 2010.

Ruvolo, P. and Eaton, E. ELLA: An efficient lifelong learning algorithm. In *Proceedings of the 30th International Conference on Machine Learning* (ICML), 2013.

Sutton, R.S., McAllester, D.A., Singh, S.P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems* (NIPS), pp. 1057–1063, 1999.

Taylor, M.E., and Stone, P. Transfer learning for reinforcement learning domains: a survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Taylor, M.E., Whiteson, S., and Stone, P. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS), 2007.

Taylor, M.E., Kuhlmann, G., and Stone, P. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS), pp. 283–290, 2008.

Thrun, S. and O'Sullivan, J. Discovering structure in multiple learning tasks: the TC algorithm. In *Proceedings of the 13th International Conference on Machine Learning* (ICML), 1996.

Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. Multi-task reinforcement learning: a hierarchical Bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning* (ICML), pp. 1015–1022, 2007.

Zhang, J., Ghahramani, Z., and Yang, Y. Flexible latent variable models for multi-task learning. *Machine Learning*, 73(3):221–242, 2008.