Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Scalable Multitask Policy Gradient Reinforcement Learning

Salam El Bsat
Rafik Hariri University

Haitham Bou Ammar
American University of Beirut

Matthew E. Taylor
Washington State University

Abstract

Policy search reinforcement learning (RL) allows agents to learn autonomously with limited feedback. However, such methods typically require extensive experience for successful behavior due to their tabula rasa nature. Multitask RL is an approach, which aims to reduce data requirements by allowing knowledge transfer between tasks. Although successful, current multitask learning methods suffer from scalability issues when considering large number of tasks. The main reasons behind this limitation is the reliance on centralized solutions. This paper proposes to a novel distributed multitask RL framework, improving the scalability across many different types of tasks. Our framework maps multitask RL to an instance of general consensus and develops an efficient decentralized solver. We justify the correctness of the algorithm both theoretically and empirically: we first proof an improvement of convergence speed to an order of $\mathcal{O}\left(\frac{1}{k}\right)$ with k being the number of iterations, and then show our algorithm surpassing others on multiple dynamical system benchmarks.

Introduction

Reinforcement learning (RL) allows agents to solve sequential decision-making problems with limited feedback. Applications with these characteristics range from robotics control (Kober and Peters 2009) to personalized medicine (Murphy et al. 2007; Pineau et al. 2007). Though successful, typical RL methods require substantial experience before acquiring acceptable behavior. The cost of obtaining such experience can be prohibitively expensive in terms of time and data.

Transfer learning (Taylor and Stone 2009) and multitask learning (Lazaric and Ghavamzadeh 2010; Li, Liao, and Carin 2009) have been developed to remedy these problems by allowing agents to reuse knowledge across tasks. Unfortunately, both these techniques suffer from scalability problems as the number of tasks grows large.

Among the different methods proposed for handling scalability in supervised learning, two directions stand-out. First, data (i.e., trajectories or tasks)

are streamed sequentially online, leading to regret minimization games where the learner plays against an adversary. Second, decentralized solvers allow multiple processing units to share work. For example, a distributed version of multitask support vector machines can significantly outperform centralized solvers (Forero, Cano, and Giannakis 2010).

Centralized online solvers have been well studied under the multitask learning setting (Ammar, Tutunov, and Eaton 2015; Bou Ammar et al. 2015; Gheshlaghi Azar, Lazaric, and Brunskill 2013). For example, Bou Ammar *et al.* proposed PG-ELLA, an online multitask policy search algorithm. PG-ELLA decomposes a task's policy parameters into a shared latent repository, $\boldsymbol{L}$, and task specific coefficients, $\boldsymbol{s}_t$, one per task. It allows for knowledge transfer between tasks (using $\boldsymbol{L}$) while streaming problems online. The main drawback of PG-ELLA (and the like, e.g., (Kumar and Daumé III 2012), (Bou Ammar et al. 2015), and (Bou-Ammar et al. 2012)) is apparent when considering a large number of tasks or dimensions — determining the shared knowledge-base $\boldsymbol{L}$ becomes intractable due to two inefficiencies. The first inefficiency is that computing the expansion's operating point amounts to solving a local RL problem described by the current task's observed trajectories. These RL optima must be computed online at each round (i.e., time-step of the interaction between the agent and adversary) of PG-ELLA. This problem is remedied in the original paper by taking gradient steps in the local MDP's objective function, while reducing the number of trajectories used. Unfortunately, such a solution has multiple drawbacks.[1] The second inefficiency reducing PG-ELLA's scalability arises when updating the shared repository $\boldsymbol{L}$. The update involves an inversion of a $dp \times dp$ matrix, with d being the number of features. This leads to a complexity of $\mathcal{O}\left(d^3p^2\right)$ at each iteration of the algorithm. Consequently, such a method is not scalable when the policy parameterization or latent dimensions p becomes high dimensional.

Moreover, as the number of tasks grows large, these centralized methods (used in updating $\boldsymbol{L}$ or computing $\tilde{\boldsymbol{\theta}}_j^\star$) be-

[1] Crucial to the success of PG-ELLA are "good-enough" local policy parameters that are encoded by the shared repository. Following a policy gradient step, however, is not guaranteed to provide informative local parameters due to the

$\mathcal{O}(1/\sqrt{k})$ convergence rate of gradient-based methods, which is the fastest rate for gradient-based techniques (Wei and Ozdaglar 2012) known so far, reduction in the number of trajectories used, and the local minima problems inherit to such techniques.

come intractable due to computational and memory constraints.

This paper introduces the first framework for addressing the multitask RL problem in a distributed and scalable manner.

Our method maps multitask policy search to an instance of general consensus. We justify the correctness of our method both theoretically and empirically. First, we show linear convergence speeds in the order of $\mathcal{O}\left(\frac{1}{k}\right)$ with k being the iteration count. Second, we demonstrate that our algorithm can surpass state-of-the-art techniques on a variety of benchmark dynamical systems. This paper's contributions can be summarized as: *i)* developing the first scalable multitask policy search reinforcement learner, *ii)* formally describing the relation between RL and general consensus optimization, *iii)* developing a *distributed multidimensional* alternating direction method of multipliers (ADMM) solver for multitask RL, *iv)* proving linear convergence in the multidimensional setting, and *v)* empirically outperforming state-of-the-art techniques on five benchmark dynamical systems.

Reinforcement Learning

Reinforcement learning (RL) is a technique used to solve sequential decision making problems with limited feedback.

RL typically models such problems as a Markov decision process (MDP): $\langle \mathcal{X}, \mathcal{U}, \mathcal{P}, R, \gamma \rangle$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is the (potentially infinite) state-space, $\mathcal{U} \subseteq \mathbb{R}^c$ is the action space, $\mathcal{P} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0, 1]$ is the transition function describing the environment's dynamics, $R : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$ is the reward function quantifying the agent's behaviour, and $\gamma \in (0, 1]$ is the discount factor.

The dynamics of an RL agent commences as follows. At each time step h, the agent resides in a state $\boldsymbol{x}_h \in \mathcal{X}$. Upon applying an action $\boldsymbol{u}_h \in \mathcal{U}$, the agent transitions to a new state $\boldsymbol{x}_{h+1} \in \mathcal{X}$ according to $\boldsymbol{x}_{h+1} \sim \mathcal{P}\left(\boldsymbol{x}_{h+1} | \boldsymbol{x}_h, \boldsymbol{u}_h\right)$ and receives a reward $r_{h+1} = R(\boldsymbol{x}_h, \boldsymbol{u}_h, \boldsymbol{x}_{h+1})$ quantifying such a transition. Repeating the aforementioned process for a horizon length H, the agent accumulates a sequence of state-action pairs, which we refer to as a trajectory denoted by $\boldsymbol{\tau} = [\boldsymbol{x}_{1:H}, \boldsymbol{u}_{1:H}]$.

The agent's goal is then defined as that of finding an action selection rule (i.e., a policy π) that maximizes the total accumulated rewards over multiple interactions with the environment.

Policy Search Reinforcement Learning

Policy search RL has shown successes in high-dimensional control problems (e.g., robotics control (Kober and Peters 2011)). In policy search, $\pi_{\boldsymbol{\theta}}$ is the policy, parameterized by a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. The agent's goal is to determine $\boldsymbol{\theta}$, maximizing:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\left[\mathcal{R}(\boldsymbol{\tau})\right] = \int_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \mathcal{R}(\boldsymbol{\tau}) d\boldsymbol{\tau}, \qquad (1)$$

where $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ and $\mathcal{R}(\boldsymbol{\tau})$ are the probabilities of acquiring a trajectory $\boldsymbol{\tau}$ and its total accumulated reward:

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \mathcal{P}_0(\boldsymbol{x}_0) \prod_{h=1}^{H} \mathcal{P}\left(\boldsymbol{x}_{h+1} | \boldsymbol{x}_h, \boldsymbol{u}_h\right) \pi_{\boldsymbol{\theta}}\left(\boldsymbol{u}_h | \boldsymbol{x}_h\right)$$
$$\mathcal{R}(\boldsymbol{\tau}) = \frac{1}{H} \sum_{h=1}^{H} r_{h+1},$$

with $\mathcal{P}_0 : \mathcal{X} \to [0, 1]$ being the initial state distribution.

Most policy gradient algorithms maximize a lower-bound to Equation 1 by generating trajectories using the current policy ($\pi_{\boldsymbol{\theta}}$) and then comparing the result with a new policy parameterized by $\underline{\boldsymbol{\theta}}$. As detailed elsewhere (Kober and Peters 2011), the expected return can be lower-bounded using Jensen's inequality and the concavity of the logarithm:

$$\begin{aligned} \log \mathcal{J}\left(\underline{\boldsymbol{\theta}}\right) &= \log \int p_{\underline{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \mathcal{R}(\boldsymbol{\tau}) d\boldsymbol{\tau} \qquad (2) \\ &\geq \int p_{\underline{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \mathcal{R}(\boldsymbol{\tau}) \log \left[\frac{p_{\underline{\boldsymbol{\theta}}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}\right] + \text{constant} \\ &\propto -\mathcal{D}_{\mathrm{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \mathcal{R}(\boldsymbol{\tau}) || p_{\underline{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(p_{\underline{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right), \end{aligned}$$

where $\mathcal{D}_{\mathrm{KL}}\left(p(\boldsymbol{\tau}) || q(\boldsymbol{\tau})\right) = \int p(\boldsymbol{\tau}) \log \frac{p(\boldsymbol{\tau})}{q(\boldsymbol{\tau})} d\boldsymbol{\tau}$. 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 $p_{\underline{\boldsymbol{\theta}}}$ of the new policy $\pi_{\underline{\boldsymbol{\theta}}}$.

Multi-Task Policy Search

Multitask policy search (MTPS) allows knowledge sharing and transfer across a group of domains. In MTPS, the agent is faced with a set of T tasks, each being an MDP denoted by $\langle \mathcal{X}_t, \mathcal{U}_t, \mathcal{P}_t, R_t, \gamma_t \rangle$. The goal of the agent is to learn a set of policies $\boldsymbol{\Pi} = \{\pi_1, \ldots, \pi_T\}$, one for each task.

Following policy gradients, we parameterize the policy for a task t by a vector $\boldsymbol{\theta}_t \in \mathbb{R}^d$, which must be found by maximizing the total expected return $\mathcal{J}_t\left(\boldsymbol{\theta}_t\right) = \int_{\boldsymbol{\tau}_t} p_{\boldsymbol{\theta}_t}(\boldsymbol{\tau}_t) \mathcal{R}_t(\boldsymbol{\tau}_t) d\boldsymbol{\tau}_t$. Consequently, when considering all tasks from $t = 1$ to $t = T$, the MTPS optimization problem is:

$$\min_{\boldsymbol{\theta}_1 : \boldsymbol{\theta}_T} -\sum_{t=1}^{T} \mathcal{J}_t(\boldsymbol{\theta}_t) + \mathrm{Reg}\left(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_T\right),$$

where $\mathrm{Reg}\left(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_T\right)$ is a regularization function that ensures improvement over single task learning, imposing shared structure between task policies. To allow for knowledge transfer between tasks, we assume a factored model represents the policy parameters, similar to the settings introduced elsewhere (Ammar, Tutunov, and Eaton 2015; Kumar and Daumé III 2012). Namely, we assume $\boldsymbol{\theta}_t = \boldsymbol{L}\boldsymbol{s}_t$. Here, $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ is a matrix used to represent k latent knowledge components, shared across all tasks. Furthermore, the task-specific coefficient vectors $\boldsymbol{s}_t \in \mathbb{R}^{k \times 1}$ "specialize" this knowledge to a task t by filtering over the shared repository $\boldsymbol{L}$. Given the above factored model, we rewrite the optimization problem of MTPS as:

$$\min_{\boldsymbol{L}, \boldsymbol{s}_1 : \boldsymbol{s}_T} -\sum_{t=1}^{T} \left[\mathcal{J}_t\left(\boldsymbol{L}\boldsymbol{s}_t\right) + \mu_1 ||\boldsymbol{s}_t||_1\right] + \mu_2 ||\boldsymbol{L}||_{\mathrm{F}}^2,$$

with $||\boldsymbol{A}||_{\mathsf{F}}$ denoting the Frobenius norm of a matrix $\boldsymbol{A}$, and $||\cdot||_1$ being the L_1 norm used to induce sparsity. The remaining ingredient needed to finalize the definition of MTPS is the usage of the lower-bound derived in Equation 2, instead of $\mathcal{J}_t(\cdot)$, leading to:

$$\min_{\boldsymbol{L},\boldsymbol{s}_1:\boldsymbol{s}_T} -\sum_{t=1}^{T}\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{\tau}_t)}\left[\mathcal{R}_t\left(\boldsymbol{\tau}_t\right)\log\left(p_{\boldsymbol{L}\boldsymbol{s}_t}\left(\boldsymbol{\tau}_t\right)\right)\right] + \mu_1\left|\left|\boldsymbol{s}_t\right|\right|_1 + \mu_2||\boldsymbol{L}||_{\mathsf{F}}^2, \tag{3}$$

where we wrote the optimization variable $\underline{\boldsymbol{\theta}_t} = \boldsymbol{L}\boldsymbol{s}_t$ to introduce transfer through $\boldsymbol{L}$, and

$$p_{\boldsymbol{L}\boldsymbol{s}_t}\left(\boldsymbol{\tau}_t\right) = \mathcal{P}_{0,t}(\boldsymbol{x}_{0,t})\prod_{h_t=1}^{H_t-1}\mathcal{P}_t\left(\boldsymbol{x}_{h+1,t}|\boldsymbol{x}_{h,t},\boldsymbol{u}_{h,t}\right) \times \pi_{\boldsymbol{L}\boldsymbol{s}_t}\left(\boldsymbol{u}_{h,t}|\boldsymbol{x}_{h,t}\right) \tag{4}$$

$$\mathcal{R}_t(\boldsymbol{\tau}_t) = \frac{1}{H_t}\sum_{h=1}^{H_t}R_t\left(\boldsymbol{x}_{h,t},\boldsymbol{u}_{h,t},\boldsymbol{x}_{h+1,t}\right). \tag{5}$$

Multi-Action RL & Gaussian Policies

Recent MTPS work typically focuses on systems exhibiting multidimensional state spaces and one dimensional actions.

We generalize this notion to the multidimensional action case by considering a matrix feedback controller. To do so, we assume a matrix $\boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right) \in \mathbb{R}^{c\times d}$ encodes multidimensional features of the state space for a task $t \in \{1,\ldots,T\}$ at time step $h \in \{1,\ldots,H_t\}$. Hence, a c-dimensional control vector $\boldsymbol{u}_{h,t}$ can be written as: $\boldsymbol{u}_{h,t} = \boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right)\boldsymbol{\theta}_t + \boldsymbol{\epsilon}_{h,t}$, where $\boldsymbol{\epsilon}_{h,t} \in \mathbb{R}^c$ is a c-dimensional Gaussian noise used for exploration. Consequently, the stochastic multidimensional policy is:

$$\pi_{\boldsymbol{\theta}_t}\left(\boldsymbol{u}_{h,t}|\boldsymbol{x}_{h,t}\right) = \mathcal{N}\left(\boldsymbol{\Phi}_t\boldsymbol{\theta}_t, \boldsymbol{\Sigma}_{h,t}^{-1}\right), \tag{6}$$

where $\boldsymbol{\Sigma}_{h,t}^{-1} \in \mathbb{R}^{c\times c}$ is a time-dependent covariance matrix. Substituting Equations 6 and 4 in 3 finalizes the MTPS problem:

$$\min_{\boldsymbol{L},\boldsymbol{s}_1:\boldsymbol{s}_T}\sum_{t=1}^{T}\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{\tau}_t)}\left[\mathcal{R}_t\left(\boldsymbol{\tau}_t\right)\sum_{h=1}^{H_t-1}\left|\left|\boldsymbol{u}_{h,t} - \boldsymbol{\Phi}_{h,t}\underbrace{\boldsymbol{L}\boldsymbol{s}_t}_{\boldsymbol{\theta}_t}\right|\right|_{\boldsymbol{\Sigma}_{h,t}^{-1}}^2 + \mu_1\left|\left|\boldsymbol{s}_t\right|\right|_1\right] + \mu_2||\boldsymbol{L}||_{\mathsf{F}}^2. \tag{7}$$

Equation 7 can also be explained intuitively. In the second term, the agent is solving an extended (over the Horizon of the trajectory) least squares problem with trajectory rewards acting as weights. In other words, a sample point (being a trajectory in our setting) is given high values if the reward attained is high and its significance is reduced in case of low values. Hence, when a "good" trajectory is observed, the agent tries to replicate the chosen actions by minimizing the error-norm between its model (i.e., $\boldsymbol{\Phi}_{h,t}\boldsymbol{L}\boldsymbol{s}_t$) and the real action $\boldsymbol{u}_{h,t}$. These norms are then summed over the horizon and a total number of exploratory trajectories (i.e., the expectation) to ensure successful learning. Finally, the outer summation (i.e., $\sum_t$) ensures a "good-enough" repository over all tasks $t \in \{1,\ldots,T\}$.

Drawbacks of Current Solvers: Solving the problem in Equation 7 appears difficult due to the non-convexity imposed by the bi-product of $\boldsymbol{L}$ and $\boldsymbol{s}_t$. In the case of bi-products, standard solutions from general optimization recognize that Equation 7 is convex in each of the variables while fixing the others and consider a two-step alternating optimization approach. In the first step, the repository is determined while fixing $\boldsymbol{s}_1,\ldots,\boldsymbol{s}_T$, while in the second step, task coefficients are computed given an updated $\boldsymbol{L}$. Though successful, such methods typically assume a centralized approach.

In the supervised learning setting, two techniques help scalability. In the first technique, data is streamed sequentially online, leading to a regret-minimization game against an adversary. The field of online multitask RL is typically referred to as lifelong RL and techniques such as PG-ELLA (Bou-Ammar et al. 2014; Bou Ammar et al. 2015) have been developed to handle this problem. PG-ELLA (and others) still assume central computations and adopt slow first-order methods for computing latent models. Furthermore, due to the usage of approximations to the original loss function (e.g., second-order Taylor expansions), these methods can become inaccurate and restrictively slow, especially when the task distribution varies widely between domains.

The second technique relies on distributed optimization where multiple processing units are considered. Such a direction has not been well-explored yet in the multitask RL setting. Next, we present the first distributed solver for MTPS and demonstrate that our method improves the convergence speed of current techniques used for scalability (e.g., PG-ELLA) to $\mathcal{O}\left(\frac{1}{k}\right)$, with k being the iteration count.

Scalable Multitask Policy Search

We assume the processing units are connected by an undirected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, with a node set $\mathcal{V}$ and an edge-set $\mathcal{E}$. We assume the n nodes are connected via m edges, and we do not impose additional constraints on the topology of $\mathcal{G}$. We further assume a natural ordering among the nodes from $1,\ldots,|\mathcal{V}|$ and $e_{ij} \in \mathcal{E}$ to denote the edges between nodes i and j with $i < j$. We define a block matrix $\boldsymbol{A}$ to have $|\mathcal{E}|$ rows and $|\mathcal{V}|$ columns. Suppose edge e_{ij} is the k^{th} edge in $\mathcal{E}$ represented by the k^{th} row in matrix $\boldsymbol{A}$. Entry (k,i) of matrix $\boldsymbol{A}$ equals the identity matrix $\boldsymbol{I}$ (i.e., $\boldsymbol{A}(k,i) = \boldsymbol{I}$), entry $\boldsymbol{A}(k,j)$ of matrix $\boldsymbol{A}$ equals $-\boldsymbol{I}$ (i.e., $\boldsymbol{A}(k,j) = -\boldsymbol{I}$), and other entires are set to matrix 0. $\boldsymbol{A}(\boldsymbol{I})$ is the general edge-node incident matrix with identity matrix $\boldsymbol{I}$.

Each node $i \in \mathcal{V}$ is randomly assigned a set of tasks. Given a set of trajectories for all T tasks, the first step is to randomly assign chunks of data to each processor. The next step is to devise a procedure to determine the latent model based on partial information. The model we consider exhibits a product of two terms: 1) the shared repository $\boldsymbol{L}$, and 2) the task specific coefficient $\boldsymbol{s}_t$ for each task. Due to the non-convexity of the product, our technique adopts alternating optimization. When considering each of the alternating steps, computing task projections can be performed

fully distributed as long as each node is equipped with a Lasso solver. Updating the repository $\boldsymbol{L}$, on the other hand, requires more care due to its unifying nature across all tasks.

Our method will handle the repository while each node computes a separate update based on its assigned tasks. This can be achieved if we map the MTPS problem to distributed general consensus. To do so, we start by introducing $\text{vec}(\boldsymbol{L})$ to be a "column-wise unrolled" version of $\boldsymbol{L}$ — the operator $\text{vec}(\boldsymbol{L})$ vectorizes the $d \times k$ repository into a vector of size dk with the columns of $\boldsymbol{L}$ concatenated consecutively. Using $\text{vec}(\boldsymbol{L})$, it can be shown that the parameter model, $\boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right)\boldsymbol{L}\boldsymbol{s}_t$, can be written as: $\text{vec}\left(\boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right)\boldsymbol{L}\boldsymbol{s}_t\right) = \left(\boldsymbol{s}_t^{\mathsf{T}} \otimes \boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right)\right)\text{vec}(\boldsymbol{L})$, with $\otimes$ denoting the Kronecker product. Having updated the task coefficients, we can rewrite the optimization problem in Equation 7 in terms of $\text{vec}(\boldsymbol{L})$ as:

$$\min_{\text{vec}(\boldsymbol{L})} \sum_{t=1}^{T}\Bigg[\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{\tau}_t)}\Bigg[\mathcal{R}_t\left(\boldsymbol{\tau}_t\right) \sum_{h=1}^{H_t-1}\left|\left|\boldsymbol{u}_{h,t} - \left(\boldsymbol{s}_t^{\mathsf{T}} \otimes \boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right)\right)\text{vec}(\boldsymbol{L})\right|\right|^2_{\boldsymbol{\Sigma}^{-1}_{h,t}}\Bigg] + \mu_1\left|\left|\boldsymbol{s}_t\right|\right|_1\Bigg] + \mu_2||\text{vec}(\boldsymbol{L})||_2^2.$$

Having split data across n processing units, we can rewrite the MTPS in a equivalent distributed form as:

$$\min_{\text{vec}(\boldsymbol{L}_1):\text{vec}(\boldsymbol{L}_n)} \sum_{i=1}^{n} \mathcal{J}_i\left(\text{vec}\left(\boldsymbol{L}_i\right)\right) + \mu_2||\text{vec}\left(\boldsymbol{L}_i\right)||_2^2 \quad (8)$$
$$\text{s.t. } \text{vec}(\boldsymbol{L}_1) = \text{vec}(\boldsymbol{L}_2) = \cdots = \text{vec}(\boldsymbol{L}_n),$$

where $\mathcal{J}_i\left(\text{vec}\left(\boldsymbol{L}_i\right)\right)$ denotes the per-node loss function defined as:

$$\mathcal{J}_i\left(\text{vec}\left(\boldsymbol{L}_i\right)\right) = \sum_{t=1}^{T_i}\Bigg[\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{\tau}_t)}\Bigg[\mathcal{R}_t\left(\boldsymbol{\tau}_t\right) \sum_{h=1}^{H_t-1}\left|\left|\boldsymbol{u}_{h,t} - \left(\boldsymbol{s}_t^{\mathsf{T}} \otimes \boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}\right)\right)\text{vec}(\boldsymbol{L})\right|\right|^2_{\boldsymbol{\Sigma}^{-1}_{h,t}}\Bigg] + \mu_1||\boldsymbol{s}_t||_1\Bigg],$$

where T_i represents all tasks assigned to node $i \in \mathcal{V}$ such that $T_1 + T_2 + \cdots + T_n = T$. The distributed form is equivalent to the MTPS problem derived in Equation 7. The crucial difference, however, is that in distributed MTPS, each node $i \in \mathcal{V}$ updates a version of the shared repository based on only *partial* knowledge of the total number of tasks. These repositories are then unified using the consensus constraint (i.e., $\text{vec}(\boldsymbol{L}_1) = \cdots = \text{vec}(\boldsymbol{L}_n)$) to generate a single common knowledge base $\text{vec}(\boldsymbol{L}) \in \mathbb{R}^{dk}$.

Solving Distributed Multitask Policy Search: At this stage, any off-the-shelve distributed optimization package could be used for determining $\boldsymbol{L}$. Distributed first-order methods (e.g., distributed sub-gradients), however, exhibit slow convergence speeds leading to problems similar to these of PG-ELLA (Bou-Ammar et al. 2014). Our goal is an MTPS algorithm with linear convergence.

The alternating direction method of multipliers (ADMM) is an optimization technique that exhibits fast convergence speeds. ADMM solves constrained problems by relying on dual methods, where it decomposes the original problem to two subproblems. These are then solved by updating dual variables. In our setting, a decentralized ADMM solver is needed.

Wei and Ozdaglar (2012) have already proposed a distributed version of ADMM and showed linear convergence of the form $\mathcal{O}\left(\frac{1}{k}\right)$ with k being the total number of iterations. However, this technique is not readily applicable to our setting for two reasons. First, the method focuses on the univariate setting, which is limited by its one-dimensional setting. Second, the convergence proofs are rather restrictive. We therefore

1) generalize distributed ADMM to the multidimensional case, handling high-dimensional shared repositories, and 2) modify the convergence proofs to better suit RL.

The strategy we follow in deriving our algorithm relies on decomposition techniques using a Lagrangian augmented with a penalty term. To derive the corresponding form, we first introduce a generalized edge-node adjacency matrix $\tilde{\boldsymbol{A}} = \boldsymbol{A}(\boldsymbol{I}_{dk\times dk}) \in \mathbb{R}^{dkm\times dkn}$, where $\boldsymbol{I}_{dk\times dk}$ is a dk-dimensional identity matrix. Notice that $\tilde{\boldsymbol{A}}$ is just the original adjacency matrix, $\boldsymbol{A}$, but considers all the repository's dimensions. Further, we introduce a vector $\text{vec}\left(\tilde{\boldsymbol{L}}\right) = [\text{vec}(\boldsymbol{L}_1), \ldots, \text{vec}(\boldsymbol{L}_n)]^{\mathsf{T}} \in \mathbb{R}^{dkn}$ to be a vector concatenating all the repositories between the nodes of $\mathcal{G}$.

We can now rewrite Equation 8 in a more compact form as:

$$\min_{\text{vec}(\boldsymbol{L}_1):\text{vec}(\boldsymbol{L}_n)} \sum_{i=1}^{n} \mathcal{J}_i\left(\text{vec}\left(\boldsymbol{L}_i\right)\right) + \mu_2||\text{vec}\left(\boldsymbol{L}_i\right)||_2^2$$
$$\text{s.t. } \tilde{\boldsymbol{A}}\text{vec}\left(\tilde{\boldsymbol{L}}\right) = \boldsymbol{0}_{dkm}.$$

To solve the above constraint optimization problem, we then introduce a vector of Lagrange multiplies $\boldsymbol{\lambda} \in \mathbb{R}^{dkm}$ and write the augmented Lagrangian as:

$$\mathcal{L}_{\text{Aug}}\left(\text{vec}\left(\tilde{\boldsymbol{L}}\right), \boldsymbol{\lambda}\right) = \sum_{i=1}^{n} \mathcal{J}_i\left(\text{vec}\left(\boldsymbol{L}_i\right)\right) + \mu_2||\text{vec}\left(\boldsymbol{L}_i\right)||_2^2 - \boldsymbol{\lambda}^{\mathsf{T}}\tilde{\boldsymbol{A}}\text{vec}\left(\tilde{\boldsymbol{L}}\right) + \frac{\rho}{2}\left|\left|\tilde{\boldsymbol{A}}\text{vec}\left(\tilde{\boldsymbol{L}}\right)\right|\right|_2^2,$$

where $\rho > 0$ is a parameter of the penalty term. The augmented Lagrangian is the standard Lagrangian, typically used in constrained optimization, in addition to a penalty term of the form $\left|\left|\tilde{\boldsymbol{A}}\text{vec}\left(\tilde{\boldsymbol{L}}\right)\right|\right|_2^2$ used to ensure stability (e.g, see (Wei and Ozdaglar 2012; Boyd and Vandenberghe 2004)). To derive shared repository updates, we follow the standard strategy proposed in the original ADMM with crucial changes needed for the distributed setting. Contrary to (Wei and Ozdaglar 2012), we assume that each processor i keeps a multidimensional local decision estimate $\text{vec}(\boldsymbol{L}_i)$ and a vector of dual variables $\boldsymbol{\lambda}_{ki}$ with $k < i$. We define two sets for the neighbors of a node i, denoted by $\mathbb{P}_i$ and $\mathbb{S}_i$. Here, $\mathbb{P}_i$ collects all nodes having an index lower than i, i.e., $\mathbb{P}_i = \{j | e_{ij} \in \mathcal{E}, j < i\}$ and $\mathbb{S}_i$ all nodes with index higher than i, i.e., $\mathbb{S}_i = \{j | e_{ij} \in \mathcal{E}, i < j\}$. Our method determines $\text{vec}^{\star}\left(\tilde{\boldsymbol{L}}\right)$ using the instructions in Algorithm 1.

Algorithm 1 consists of two steps. First, primal updates are performed by each node in $\mathcal{G}$ (line 3). Primal updates can be seen similar to standard ADMM with the crucial difference of being performed completely locally due to the introduction of $\mathbb{P}_i$ and $\mathbb{S}_i$. Second, given the primal, the dual variables are then computed as shown in line 4. Note that in the case of multidimensional Gaussian policies[2], the primal updates on line 3 of Algorithm 1 can be computed in closed form[3] after approximating the expectation by the sample mean over M_t trajectories for all tasks $t \in \{1, \ldots, T\}$:

$$\text{vec}\left(\boldsymbol{L}_i^{(k+1)}\right) = \left[\boldsymbol{P}_i + \frac{\rho \text{deg}_i}{2}\boldsymbol{I}_{dk\times dk}\right]^{-1}\Bigg(\boldsymbol{c}_i + \frac{\rho}{2}\Bigg(\sum_{j\in\mathbb{S}_i}\left[\text{vec}(\boldsymbol{L}_j^{(k)}) + \frac{1}{\rho}\boldsymbol{\lambda}_{ij}^{(k)}\right] + \sum_{j\in\mathbb{P}_i}\left[\text{vec}(\boldsymbol{L}_j^{(k+1)}) - \frac{1}{\rho}\boldsymbol{\lambda}_{ji}^{(k)}\right]\Bigg)\Bigg),$$

where

$$\boldsymbol{P}_i = \sum_{t=1}^{T_i}\frac{1}{M_t}\sum_{m=1}^{M_t}\mathcal{R}\left(\boldsymbol{\tau}_t^{(m)}\right)\sum_{h=1}^{H_{t-1}^{(m)}}\left(\boldsymbol{s}_t^{\mathsf{T}}\otimes\boldsymbol{\Phi}_{h,t}^{(m)}\left(\boldsymbol{x}_{h,t}^{(m)}\right)\right)^{\mathsf{T}} \times \boldsymbol{\Sigma}_{h,t}^{-1}\left(\boldsymbol{s}_t^{\mathsf{T}}\otimes\boldsymbol{\Phi}_{h,t}^{(m)}\left(\boldsymbol{x}_{h,t}^{(m)}\right)\right)$$

$$\boldsymbol{c}_i = \sum_{t=1}^{T_i}\frac{1}{M_t}\sum_{m=1}^{M_t}\mathcal{R}\left(\boldsymbol{\tau}_t^{(m)}\right)\sum_{h=1}^{H_{t-1}^{(m)}}\left(\boldsymbol{s}_t^{\mathsf{T}}\otimes\boldsymbol{\Phi}_{h,t}\left(\boldsymbol{x}_{h,t}^{(m)}\right)\right)^{\mathsf{T}} \times \boldsymbol{\Sigma}_{h,t}^{-1}\boldsymbol{u}_{h,t}^{(m)},$$

where deg_i is the degree of processing unit i.

Theoretical Guarantees: We now show the theorem that derives an improvement over other techniques $\Big($e.g., PG-ELLA with $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)\Big)$ to a linear convergence rate for our proposed method. Before detailing our theoretical result, however, we note the following standard assumptions (Boyd and Vandenberghe 2004; Wei and Ozdaglar 2012):

Assumption 1 (Saddle Point & Penalty Boundedness) *The augmented Lagrangian has a saddle point. That is, there is a solution* $\left(vec\left(\tilde{\boldsymbol{L}}^{\star}\right), \boldsymbol{\lambda}^{\star}\right)$ *such that:*

$$\mathcal{L}_{Aug}\left(\cdot\right) \leq \mathcal{L}_{Aug}\left(vec\left(\tilde{\boldsymbol{L}}^{\star}\right), \boldsymbol{\lambda}^{\star}\right) \leq \mathcal{L}_{Aug}\left(\cdot\right).$$

We further assume that $||\tilde{\boldsymbol{A}}\tilde{vec}(\boldsymbol{L})||_2^2 \leq \gamma$, *for* $\gamma > 0$.

Assuming the above, the appendix proves:

Theorem 1 *The sequence* $\{vec\left(\tilde{\boldsymbol{L}}^{(k)}\right), \boldsymbol{\lambda}^{(k)}\}_{k\geq 0}$, *converges linearly with a rate given by* $\mathcal{O}\left(\frac{1}{k}\right)$.

Experiments & Results

We empirically validate our method on five existing benchmark dynamical systems (Bou Ammar et al. 2015).

[2]This derivation can be generalized to any policy in the exponential family.

[3]The derivation can be found in the appendix.

Algorithm 1 Scalable Multitask Policy Search

1: Initialize $\text{vec}(\boldsymbol{L}_i)$, $\forall i \in \{1, \ldots, n\}$, and set $\rho > 0$.
2: **for** $k = 0, \ldots, K$ **do**
3: Each agent i updates, $\text{vec}(\boldsymbol{L}_i)$ in a sequential order from $i = 1, \ldots, |\mathcal{V}|$ using:

$$\text{vec}\left(\boldsymbol{L}_i^{(k+1)}\right) = \arg\min_{\text{vec}(\boldsymbol{L}_i)}\Bigg[\mathcal{J}_i\left(\text{vec}\left(\boldsymbol{L}_i\right)\right) + \mu_2||\text{vec}\left(\boldsymbol{L}_i\right)||_2^2 + \frac{\rho}{2}\sum_{l\in\mathbb{P}_i}\left|\left|\text{vec}\left(\boldsymbol{L}_i^{(k+1)}\right) - \text{vec}\left(\boldsymbol{L}_i^{(k)}\right) - \frac{1}{\rho}\boldsymbol{\lambda}_{li}^{(k)}\right|\right|_2^2 + \frac{\rho}{2}\sum_{l\in\mathbb{S}_i}\left|\left|\text{vec}\left(\boldsymbol{L}_i\right) - \text{vec}\left(\boldsymbol{L}_i^{(k)}\right) - \frac{1}{\rho}\boldsymbol{\lambda}_{il}^{(k)}\right|\right|_2^2\Bigg].$$

4: Each agent updates $\boldsymbol{\lambda}_{li}$ for $l \in \mathbb{P}_i$ as:

$$\boldsymbol{\lambda}_{li}^{(k+1)} = \boldsymbol{\lambda}_{li}^{(k)} - \rho\left(\text{vec}\left(\boldsymbol{L}_l^{(k)}\right) - \text{vec}\left(\boldsymbol{L}_l^{(k+1)}\right)\right).$$

5: **end for**

The *cart pole* (CP) system is controlled by the cart's mass m_c in kg, the pole's mass m_p in kg and the pole's length l in meters. The state is given by the cart's position and velocity v, as well as the pole's angle θ and angular velocity $\dot{\theta}$. The goal is to control the pole in an upright position.

The *double inverted pendulum* (DIP) is an extension of the cart pole system. It has one cart m_0 in kg and two poles in which the corresponding lengths are l_1 and l_2 in meters. We assume the poles have no mass and there are two masses m_1 and m_2 in kg on the top of each pole. The state consists of the cart's position x_1 and velocity v_1, the lower pole's angle θ_1 and angular velocity $\dot{\theta}_1$, as well as the upper pole's θ_2 and angular velocity $\dot{\theta}_2$. The goal is also to learn a policy to control the two poles in a specific state.

A linearized model of a *helicopter* (HC) assumes constant horizontal motion, characterized by two matrices $A \in \mathbb{R}^{4\times 4}$ and $B \in \mathbb{R}^{4\times 2}$. The main goal is to stabilize the helicopter by controlling the collective and differential rotor thrust.

The *simple mass* (SM) system is characterized by 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 x and the velocity, v, of the mass. The goal is to train a policy for guiding the mass to a specific state.

The *double mass* (DM) is an extension of the simple mass system with two masses m_1, m_2 (in kg), two springs (with spring constants k_1 and k_2 in N/m), and two damping constants (d_1 and d_2 in Ns/m). The state consists of the big mass' position x_1 and velocity v_1, as well as the small mass' position x_2 and velocity v_2. The goal is also to learn a policy to control the two mass in a specific state.

We generated 150 tasks for each domain by varying the dynamical parameters of each of the domains (for 750 in total). The cost was given by $\sqrt{||\boldsymbol{x}_m - \boldsymbol{x}_{\text{ref}}||_2^2}$, where $\boldsymbol{x}_{\text{ref}}$ was the goal state. We run each task for a total of 200 iterations. At each iteration, the learner observed a task through 50 trajectories of 150 steps and performed algorithmic updates. We used eNAC (Peters and Schaal 2008), a standard

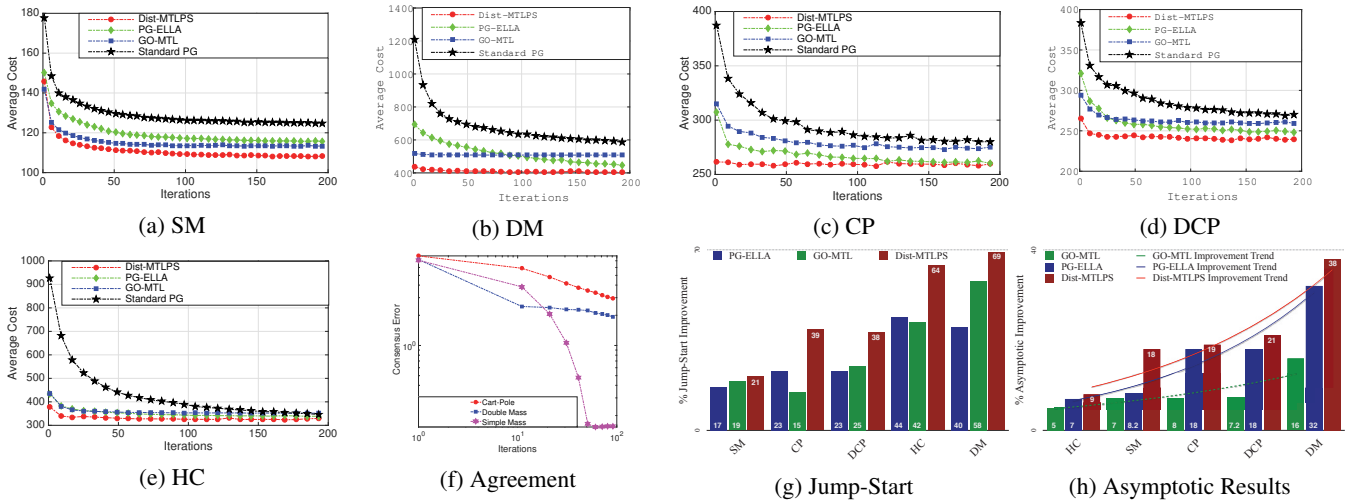

Figure 1: Figures (a)-(e) report average cost versus iterations and demonstrate that Dist-MTLPS is capable of outperforming other methods. Figure (f) shows the agreement error across processors on three sample systems. Figures (g) and (h) show that Dist-MTLPS also often outperforms the competition in the jumpstart and asymptotic performance.

PG algorithm, as the base learner. We compare our method (Dist-MTLPS) to standard PG, PG-ELLA (an online variant of multitask learning introduced to handle scalability to large number of tasks), and GO-MTL (Kumar and Daume 2012). For a fair comparison against PG-ELLA, we approximate the original loss of an RL problem by a second-order Taylor expansion around a local optimum computed at a given set of trajectories. Our method therefore solves the same problem as PG-ELLA. This comparison with PG-ELLA allows us to understand wether our method is capable of outperforming state-of-the-art techniques that adapt regret minimization games for scaling-up multitask reinforcement learning. We show in Figures (1a)-(1e) that our method can outperform all comparison methods in terms of the data complexity. One could worry that these improvements are only possible because of increased computation. However, Figure 2a shows that our technique actually achieves these data improvements while *improving* wall clock running times.

To distribute our computations, we made use of MATLAB's parallel pool running on 10 nodes. For Dist-MTLPS, we assigned 150 tasks evenly across 10 agents. The edges between agents were generated randomly to increase the likelihood of expander graphs, which provide improved practical consideration (e.g., low condition numbers leading to increased computational stability). To make sure every node in the graph has at least one predecessor and successor, we connect every node i to node $i+1$ (and wrap around to node 1 for node 10). We then randomly assigned 20 additional edges to the graph.

Figures 1(a)-1(e) report the average cost on the different benchmark in terms of iterations. In all systems, our method outperforms the others in total cost incurred. As we consider constraint optimization, it is crucial for Dist-MTLPS to acquire a solution that satisfies the constraints. Figure 1(f) shows that our method is capable of acquiring feasible solutions by reporting the consensus error on three example systems — our method not only acquires good repositories, but is also capable of converging to an agreement between the processing units. Interestingly, when interpreted differently, this result can pave the way for considering safe-MTPS (which we leave as a direction for future work).

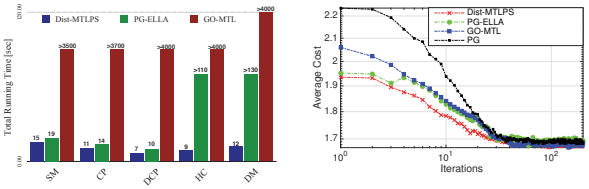

Figure 2: (a) shows Dist-MTLPS takes less time to optimize the multitask learning objective function (seconds) and (b) shows our method can generalize to novel tasks.

Figures 1(g) and 1(h) demonstrate the our algorithm is capable of outperforming PG-ELLA and GO-MTL in terms of jumpstarts and asymptotic performance. Figure 2(a) demonstrates that our method takes less wall clock time to optimize the objective function. As tasks become more complex, the advantage of using our distributed method increases. For instance, Dist-MTLPS can solve a problem with 50 HC tasks in about 9 seconds, compared to $\sim$ 200 seconds for PG-ELLA and over 4000 seconds for GO-MTL.

Multitask learning provides significant advantages when the agent faces a novel task domain. To evaluate this, we chose the most complex of the task domains (HC) and trained the multitask learner, using Dist-MTLPS, on 149 tasks to yield an effective shared knowledge base for each of the algorithms. Then, we evaluated the ability to learn a new task from the helicopter domain, comparing the benefits of Dist-MTLPS transfer from $\boldsymbol{L}_{\text{Dist-MTLPS}}$ with PG-ELLA

(from $\boldsymbol{L}_{\text{PG-ELLA}}$), GO-MTL (from $\boldsymbol{L}_{\text{GO-MTL}}$), and PG. Figure 2(b) depicts the result of learning on a novel domain, averaged over HC tasks, showing that Dist-MTLPS converges fastest in this scenario.

Conclusions and Future Work

This paper introduces the first distributed, scalable multi-task policy search framework. We show our method achieves linear convergence speeds, a significant improvement over gradient-based methods. We further assess our technique empirically and demonstrated superiority to state-of-the-art methods on a variety of benchmark dynamical systems. Future work will include targeting the more general cross-domain multitask learning setting and running our method on real-world multiple-robotics applications.

Acknowledgements

The authors would like to thank Yusen Zhan for the help in the experimental results and for the interesting discussions. Further, the authors would like to thank the anonymous reviewers who helped better-shape this paper.

References

Ammar, H. B.; Tutunov, R.; and Eaton, E. 2015. Safe policy search for lifelong reinforcement learning with sublinear regret. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2361–2369.

Bou-Ammar, H.; Tuyls, K.; Taylor, M. E.; Driessen, K.; and Weiss, G. 2012. Reinforcement Learning Transfer via Sparse Coding. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Bou-Ammar, H.; Eaton, E.; Ruvolo, P.; and Taylor, M. 2014. Online Multi-task Learning for Policy Gradient Methods. In Jebara, T., and Xing, E. P., eds., *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. JMLR Workshop and Conference Proceedings.

Bou Ammar, H.; Eaton, E.; Luna, J. M.; and Ruvolo, P. 2015. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-15)*.

Boyd, S., and Vandenberghe, L. 2004. *Convex Optimization*. New York, NY, USA: Cambridge University Press.

Forero, P. A.; Cano, A.; and Giannakis, G. B. 2010. Consensus-Based Distributed Support Vector Machines. *Journal of Machine Learning Research* 11:1663–1707.

Gheshlaghi Azar, M.; Lazaric, A.; and Brunskill, E. 2013. Sequential Transfer in Multi-armed Bandit with Finite Set of Models. In Burges, C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc.

Kober, J., and Peters, J. R. 2009. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, 849–856.

Kober, J., and Peters, J. 2011. Policy Search for Motor Primitives in Robotics. *Machine Learning* 84(1-2).

Kumar, A., and Daumé III, H. 2012. Learning Task Grouping and Overlap in Multi-Task Learning. In *International Conference on Machine Learning (ICML)*.

Kumar, A., and Daume, H. 2012. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 1383–1390.

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

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

Murphy, S. A.; Oslin, D. W.; Rush, A. J.; and Zhu, J. 2007. Methodological challenges in constructing effective treatment sequences for chronic psychiatric disorders. *Neuropsychopharmacology* 32(2):257–262.

Peters, J., and Schaal, S. 2008. Natural Actor-Critic. *Neurocomputing* 71.

Pineau, J.; Bellemare, M. G.; Rush, A. J.; Ghizaru, A.; and Murphy, S. A. 2007. Constructing evidence-based treatment strategies using methods from computer science. *Drug and alcohol dependence* 88:S52–S60.

Taylor, M. E., and Stone, P. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10:1633–1685.

Wei, E., and Ozdaglar, A. 2012. Distributed alternating direction method of multipliers. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, 5445–5450. IEEE.