Diff-DAC: Distributed Actor-Critic for Average Multitask Deep Reinforcement Learning

Sergio Valcarcel Macua
PROWLER.io
Cambridge, UK
sergio@prowler.io

Aleksi Tukiainen
PROWLER.io
Cambridge, UK
aleksi@prowler.io

Daniel García-Ocaña Hernández
Universidad Politécnica de Madrid
Madrid, Spain
d.garcia-ocana@alumnos.upm.es

David Baldazo
Universidad Politécnica de Madrid
Madrid, Spain
d.baldazo@alumnos.upm.es

Enrique Munoz de Cote
PROWLER.io
Cambridge, UK
enrique@prowler.io

Santiago Zazo
Universidad Politécnica de Madrid
Madrid, Spain
santiago@gaps.ssr.upm.es

ABSTRACT

We propose a fully distributed actor-critic algorithm approximated by deep neural networks, named *Diff-DAC*, with application to single-task and average multitask reinforcement learning (MRL). Each agent has access to data from its local task only, but it aims to learn a policy that performs well on average for the whole set of tasks. During the learning process, agents communicate their value-policy parameters to their neighbors, diffusing the information across the network, such that they converge to a common policy, with no need for a central node. The method is scalable, since the computational and communication costs per agent grow with its number of neighbors. We derive Diff-DAC's from duality theory and provide novel insights into the standard actor-critic framework, showing that it is actually an instance of the dual ascent method that approximates the solution of a linear program. Experiments suggest that Diff-DAC can outperform the only previous distributed MRL approach (i.e., Dist-MTLPS) and even the centralized architecture.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**; **Multitask learning**; *Cooperation and coordination*; *Markov decision processes*; Neural networks;

KEYWORDS

Reinforcement learning, Multitask learning, Distributed optimization, Primal dual, Saddle point, Actor critic, Deep learning.

1 INTRODUCTION

Within a decade, billions of interconnected devices will be processing and exchanging data throughout the global economy [28]. Centralized reinforcement learning (RL) architectures, where all devices interact with a central station, may be unfeasible. Fully distributed RL algorithms, where agents communicate only with their neighbors and without central control, offer a solution to this problem, since the communication cost per agent scales linearly with its number of neighbors. In this distributed approach, each agent learns by interacting with its own environment, but is able to cooperate and benefit from the learning process of the whole network. When all agents' environments are equal, they learn to perform a single task; when environments are different but related, they learn to generalize across all tasks [23]. The latter is known as the multitask reinforcement learning (MRL) problem. We propose an algorithm named *Diffusion-based Distributed Actor-Critic* (Diff-DAC) for both single-task and MRL problems.

Most previous MRL approaches assume access to data from all tasks [3, 16, 24]. But if the number of tasks is large and their data are geographically distributed, the communication cost of transmitting the data to a central station may be prohibitive.

The idea of making scalable MRL with distributed optimization was first proposed by [7] with the Dist-MTLPS method, which extended a distributed implementation of ADMM due to Wei and Ozdaglar [29]. Our work improves over Dist-MTLPS in a number of ways: *i)* Dist-MTLPS relies on linear function approximation, which requires finding salient features, and it only considers policies in the natural exponential family of distributions. Diff-DAC, on the other hand, uses deep learning architectures to avoid costly feature engineering, and is able to learn more expressive policies. *ii)* The distributed ADMM updates of the agents are done in sequential order, requiring finding a cyclic path that visits all agents, which is generally an NP-hard problem [10]. Diff-DAC uses a diffusion strategy [20], where each agent interacts with its neighbors, with no ordering, and possibly asynchronously [33]. *iii)* Sequential strategies are sensitive to agent or link failures, since they stop the information flow; while diffusion strategies are robust since the agents can still operate even if parts of the network become isolated.

To the best of our knowledge, all other previous works on distributed RL only considered tabular or linear function approximations (e.g., [9, 25, 27]), and do not apply immediately to expressive nonlinear approximations. In particular, reference [9] added a consensus rule to tabular Q-learning; a nonlinear extension raises questions like whether we should we add consensus to the target network updates, and would be an alternative contribution to our actor-critic approach. The Dist-GTD method due to [27] is for policy evaluation with linear approximation, and its extension to control and nonlinear approximations isn't trivial even for the single-agent GTD. Finally, reference [25] proposed a second order method, implying the inversion of the Hessian at every agent, which might be problematic for neural networks with hundreds of neurons. Other related works suffer from similar drawbacks.

Contributions. (1) We propose a fully distributed actor-critic deep reinforcement learning algorithm named Diff-DAC for the single and average multitask problem that scales gracefully to large number of tasks. (2) We re-derive the actor-critic framework from duality theory and show that it is an instance of *dual-ascent* to approximate the saddle-point of the Lagrangian of a linear program (LP). This derivation formalizes previous intuitions [17] and provides novel insights, like a policy gradient that includes the advantage function explicitly, rather than as a variance reduction technique. (3) Experimental results suggest that Diff-DAC outperforms Dist-MTLPS, and that it is more stable and achieves better local optima than the centralized approach, without replay memory or target networks.

2 PROBLEM FORMULATION

In this section, we formalize tasks as Markov decision processes (MDPs), define a family of tasks and introduce the multitask optimization problem.

Consider a parametric family of MDPs defined over finite[1] state-action sets, $\mathbb{S}$ and $\mathbb{A}$. Each MDP of the family has state transition distribution, $\mathcal{P}_\theta(s'|s,a)$, reward function, $r_\theta(s,a)$ and distribution over the initial state, $\mu_\theta(s)$, $\forall s, s' \in \mathbb{S}$, $a \in \mathbb{A}$, that depend on some parameter $\theta \in \Theta$, where Θ is a measurable compact set. The task family is given as a probability distribution over the parameter set, f, so that the parameter is a random variable[2]: $\boldsymbol{\theta} = \theta \sim f$. Let $\pi : \mathbb{S} \times \mathbb{A} \mapsto [0,1]$ be a stationary policy, such that $\pi(a|s)$ denotes the probability of taking action a at state s.

Let $v : \Pi \times \mathbb{S} \mapsto \mathbb{R}$ denote the value function, such that $v_\theta^\pi(s)$ is the value at state s when following policy π:

$$v_\theta^\pi(s) \triangleq \mathsf{E}_{\pi,\mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) \,\middle|\, \boldsymbol{s}_0 = s\right], \tag{1}$$

where $\mathsf{E}_{\pi,\mathcal{P}_\theta}[\cdot]$ is the expected value when $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t)$ and $\boldsymbol{s}_{t+1} \sim \mathcal{P}_\theta(\cdot|s_t, a_t)$; and $0 \leq \gamma < 1$ is the discount factor. Introduce the vector of values: $v_\theta^\pi \triangleq \left(v_\theta^\pi(s)\right)_{s\in\mathbb{S}} \in \mathbb{R}^{|\mathbb{S}|}$.

Suppose we have observed N tasks that correspond to parameters $\{\theta_k\}_{k=1}^N$. Our goal is to learn a stationary policy that maximizes the *global* average value:

$$\underset{\pi}{\text{maximize}} \quad \overline{\mu}^\top \left(\sum_{k=1}^{N} v_{\theta_k}^\pi\right), \tag{2}$$

where $\overline{\mu}$ is a vector of positive convex combination weights.

We assume bounded rewards for all $\theta \in \Theta$, such that:

$$|r_\theta(s,a)| \leq R_{\max} < \infty, \quad \forall (s,a) \in \mathbb{S} \times \mathbb{A}, \tag{3}$$

for some scalar $R_{\max}$. Under this assumption we can easily ensure existence of solution to (2). We define $\pi^\star$ as a policy that is a solution to (2), and $v^\star$ as its optimal value.

When all task parameters $\{\theta_k\}_{k=1}^N$ are equal, (2) is the single-task RL problem; when they differ, (2) becomes an MRL problem where we aim to learn a single policy that performs optimally in average for the whole set of tasks, although it might perform well for some tasks but poorly for others. Experiments show that our solution to (2) can outperform previous methods for more general formulations. Moreover, we consider this setting to be a stepping stone towards distributed task-dependent policies, as discussed in Sec. 7,

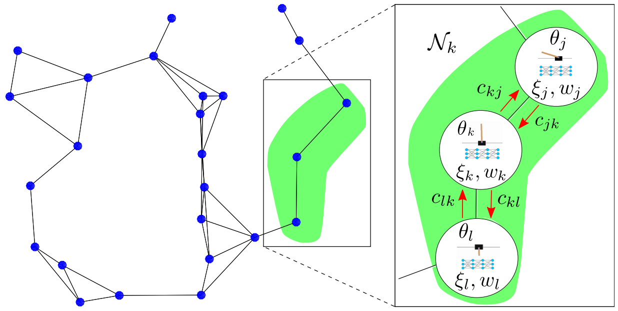

Figure 1: Example of network and detailed neighborhood. Blue nodes represent agents, and edges represent their connectivity. This network consists of $N = 25$ agents, with average neighborhood size: $|\overline{\mathcal{N}}| = \frac{1}{N}\sum_{k=1}^{N} |\mathcal{N}_k| = 4.2$. On the right, the figure zooms over neighborhood $\mathcal{N}_k$ (green area), where each agent k runs its own task instance of the swing-up cart-pole task (with different pole length and mass, defined by parameter θ_k). As explained in Sec. 5, agent k transmits its neural network weights, $\xi_{k,i}$ and $w_{k,i}$, to its neighbors j and l; and it receives their weights $\xi_{j,i}, w_{j,i}$ and $\xi_{l,i}, w_{l,i}$, and combines them with coefficients c_{jk} and c_{lk}, respectively.

3 NETWORKED MULTIAGENT SETTING

In this section we introduce the networked multiagent setting that learns in a fully distributed manner.

We have a network of N agents, which is expressed as a graph, $\mathcal{N}$. Each node, denoted $k = 1, \ldots, N$, corresponds to an agent that learns from data coming from its own task[3], with parameter $\theta_k \sim f$. The edges in the graph represent communication links. We assume that the graph is *connected* (i.e., there is at least one path between every pair of agents).

The graph can be represented by a non-negative matrix of size $N \times N$, denoted $C \triangleq (c_{lk})_{l,k=1}^N$, such that the element $c_{lk} \geq 0$ represents the weight given by agent k to information coming from l. Each agent k is only allowed to communicate within its own neighborhood, $\mathcal{N}_k$, which is defined as the set of agents to which it is directly connected, including k itself: $\mathcal{N}_k \triangleq \{l \in \{1, \ldots, N\} : c_{lk} > 0\}$. In order to ensure that the information flows through the network, we require the following standard conditions on the connectivity matrix C, which make it doubly-stochastic and primitive [20, 27]:

$$C^\top \mathbb{1} = \mathbb{1},\ C\mathbb{1} = \mathbb{1},\ \text{ and } c_{lk} \geq 0,\ k,l = 1, \ldots, N, \tag{4}$$

$$\operatorname{trace}[C] > 0, \tag{5}$$

where $\mathbb{1}$ is a vector of ones. Although conditions (4)–(5) seem restrictive, it turns out that there are procedures for every agent k

[1] The proposed Diff-DAC algorithm uses function approximation so that it is able to work in continuous state-action sets as well.

[2] We use boldface font to denote random variables and regular font to denote instances or deterministic variables.

[3] For simplicity, we assume that each agent is allocated with one task, similar to [7]. The extension to multiple tasks per agent is trivial.

to find the weights $\{c_{lk}\}_{l \in \mathcal{N}_k}$ in a fully distributed manner, such that C satisfies the required conditions. One of such procedures is the Hastings rule [32], [20, p.492].

4 MULTITASK ACTOR-CRITIC FROM DUALITY THEORY

In this section, we reformulate MRL as a linear program (LP), and show that by applying dual ascent to the Lagrangian, we get a tabular model-based actor-critic method that solves (2).

Introduce some average global variables:

$$\overline{v}^{\pi} \triangleq \frac{1}{N} \sum_{k=1}^{N} v_{\theta_k}^{\pi}, \tag{6}$$

$$\overline{r}(s,a) \triangleq \frac{1}{N} \sum_{k=1}^{N} r_{\theta_k}(s,a), \tag{7}$$

$$\overline{\mathcal{P}}(s'|s,a) \triangleq \frac{1}{N} \sum_{k=1}^{N} \mathcal{P}_{\theta_k}(s'|s,a). \tag{8}$$

The following lemma is key in our derivations, since it allows us to consider the multitask problem as a single MDP, with state-action reward and transitions given by $\overline{r}$ and $\overline{\mathcal{P}}$.

Theorem 1. *$\overline{\mathcal{P}}$ is a row-stochastic matrix.*

Proof. Stochastic matrices lie in a compact convex set [8, Th. 8.7], so that their convex combination lies in the same set [4, p.24]. More intuitively, note that $\overline{\mathcal{P}}$ is just a finite mixture of the distributions of all tasks. □

Thus, we can use standard (i.e., single-task) optimal control results [18, pp. 143–151] for our setting. Let $\mathbb{V}$ denote the set of bounded real functions on $\mathbb{S}$, with component wise partial order and norm $\|v\| \triangleq \sup_{s \in \mathbb{S}} |v(s)|$.

Corollary 1. *For any stationary policy π, $\overline{v}^{\pi}$ is the unique solution in $\mathbb{V}$, $\forall s \in \mathbb{S}$, of the Bellman equation:*

$$v(s) = \sum_{a \in \mathbb{A}} \pi(a|s) \left(\overline{r}(s,a) + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|s,a) v(s') \right). \tag{9}$$

The Bellman operator $\overline{T} : \mathbb{V} \mapsto \mathbb{V}$ for the new MDP defined by $\overline{r}$ and $\overline{\mathcal{P}}$ is given by:

$$\left(\overline{T}v\right)(s) \triangleq \max_{a \in \mathbb{A}} \left[\overline{r}(s,a) + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|s,a) v(s') \right]. \tag{10}$$

Corollary 2. *$\overline{T}$ is a contraction mapping with fixed point $v^{\star}$.*

Moreover, similar to single task optimal control theory [18, Sec. 9.1], we can reformulate (2) as an LP.

$$\begin{aligned} \underset{v \in \mathbb{R}^{|\mathbb{S}|}}{\text{minimize}} \quad & \overline{\mu}^{\top} v \\ \text{s.t.} \quad & v(s) \geq \overline{r}(s,a) + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|s,a) v(s') \\ & \forall (s,a) \in \mathbb{S} \times \mathbb{A}, \end{aligned} \tag{11}$$

From Corollary 2, we know that the feasible set of (11) consists of the single point $v^{\star}$.

Since problem (11) satisfies Slater condition, strong-duality holds [4, Sec. 5.2.3] and the primal and dual optimal values are attained and equal. The Lagrangian of (11) is given by:

$$\begin{aligned} L(v,d) = \overline{\mu}^{\top} v + \sum_{(s,a) \in \mathbb{S} \times \mathbb{A}} d(s,a) \Bigg(& \overline{r}(s,a) \\ & + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|s,a) v(s') - v(s) \Bigg), \end{aligned} \tag{12}$$

where the dual variable $d \triangleq (d(s,a))_{(s,a) \in \mathbb{S} \times \mathbb{A}} \in \mathbb{R}^{|\mathbb{S}||\mathbb{A}|}$ is a nonnegative vector of length $|\mathbb{S}||\mathbb{A}|$. Let $d^{\star}$ denote the optimal dual variable, which might not be unique. The saddle point condition of (12) is given by:

$$\min_{v} \max_{d} L(v,d) = L(v^{\star}, d^{\star}) = \max_{d} \min_{v} L(v,d). \tag{13}$$

There are multiple approaches to find a saddle point. We focus on the *dual-ascent* scheme [1], which consists in alternating between: *i)* Finding a primal solution, given the dual variable; and *ii)* ascending in the direction of $\nabla_d L(v,d)$, given the primal variable.

First, we show how to update the *primal* variable given d:

$$v \leftarrow \arg \min_{v \in \mathbb{R}^{|\mathbb{S}|}} L(v,d). \tag{14}$$

Recall that the Karush-Kuhn-Tucker (KKT) conditions are sufficient for optimality in convex problems that satisfy Slater's condition and have differentiable objective and constraints [4, Sec. 5.5.3]. KKT conditions include the gradient of the objective and constraints with respect to the primal variable, nonnegativity of the dual variable (for inequality constraints), the feasibility constraints, and *complementary slackness.* Since problem (11) is linear, first-order conditions do not depend on v, so that they hold $\forall v \in \mathbb{R}^{|\mathbb{S}|}$. Thus, the only condition that depends on v is *complementary slackness*:

$$\sum_{(s,a) \in \mathbb{S} \times \mathbb{A}} d(s,a) \left(\overline{r}(s,a) + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|s,a) v(s') - v(s) \right) = 0. \tag{15}$$

Similar to the standard single-task problem [18, Sec. 6.9], if we set $\overline{\mu} \triangleq \frac{1}{N} \sum_{k=1}^{N} \mu_{\theta_k}^{\top}$, it can be shown that our dual variable is an improper discounted state-action visitation distribution, so that we can obtain the policy from d:

$$\pi(a|s) = \frac{d(a,s)}{\rho(s)}, \quad \text{where} \quad \rho(s) \triangleq \sum_{a' \in \mathbb{A}} d(a', s). \tag{16}$$

Hence, the Bellman equation (9), typically used to derive the *critic* in actor-critic methods, is sufficient to guarantee (15).

Second, for the *dual* variable, we simply perform gradient ascent in the Lagrangian, yielding a recursion of the form:

$$d \leftarrow \left[d + \alpha \nabla_d L(v,d) \right]^{+}, \tag{17}$$

where α is the step-size, $[\cdot]^{+}$ denotes projection on the nonnegative quadrant, and the ∇_d denotes gradient w.r.t. dual variable d:

$$\nabla_d L(v,d) = \left(\frac{\partial L(v,d)}{\partial d(s,a)} \right)_{(s,a) \in \mathbb{S} \times \mathbb{A}}. \tag{18}$$

Interestingly, note that the partial derivatives of the Lagrangian in (18) are indeed the so named *advantage function* extended to our multitask problem:

$$A(s,a) \triangleq \overline{r}(s,a) + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|s,a) v(s') - v(s) = \frac{\partial L(v,d)}{\partial d(s,a)}. \tag{19}$$

If we learn $d^\star$, we can use (16) to obtain $\pi^\star$, so that recursion (17) can be seen as an *actor* update. Thus, (9) and (17) define a novel *tabular model-based actor-critic* method. In the following, we extend this approach to a model-free distributed actor-critic method with neural network approximations.

5 DISTRIBUTED DEEP ACTOR-CRITIC

In order to use *diffusion* strategies [20] to derive a distributed optimization method, we have to express the global objective as a convex combination of each agent's local objective. Thus every agent can optimize its objective from local data; and by communicating with their neighbors, all agents converge to a common solution that optimizes the global objective. We do this for both critic and actor.

5.1 Distributed policy evaluation: Critic

When computing the critic for large (or continuous) state-action sets, it is common to approximate the value function with some parametric function $v_\xi(s) \approx v(s)$, where $\xi \in \mathbb{R}^{M_v}$ denotes the parameter vector of length M_v. We choose neural networks with multiple hidden layers (i.e., deep learning) as parametric approximators. Hence, we can learn the network weights, ξ, by transforming (9) into a nonlinear regression problem:

$$\underset{\xi \in \mathbb{R}^{M_v}}{\text{minimize}} \quad J(\xi) \triangleq \mathsf{E}\left[\left(v_\xi\left(\boldsymbol{s}_t\right) - \overline{\boldsymbol{y}}_t\right)^2\right], \tag{20}$$

where the target values are given by:

$$\overline{\boldsymbol{y}}_t \triangleq \overline{r}(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \sum_{s' \in \mathbb{S}} \overline{\mathcal{P}}(s'|\boldsymbol{s}_t, \boldsymbol{a}_t) v_\xi(s'). \tag{21}$$

In order to derive a diffusion-based distributed critic, we have to reformulate the problem as minimizing the convex combination of costs that depend only on a single task each. The cost for each individual task takes the form:

$$\widetilde{J}_k(\xi) \triangleq \mathsf{E}\left[\left(v_\xi\left(\boldsymbol{s}_t\right) - \boldsymbol{y}_{k,t}\right)^2\right], \ k = 1, \ldots, N, \tag{22}$$

where $\boldsymbol{y}_{k,t}$ is the target from task k at time t, given by

$$\boldsymbol{y}_{k,t} = r_{\theta_k}(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \sum_{s' \in \mathbb{S}} \mathcal{P}_{\theta_k}(s'|\boldsymbol{s}_t, \boldsymbol{a}_t) v_\xi(s'),$$

such that $\overline{\boldsymbol{y}}_t = 1/N \sum_{k=1}^N \boldsymbol{y}_{k,t}$. Now, in order to obtain a cost that is a combination of the individual costs, we can use Jensen's inequality to upper bound $J(\xi)$ by another function, $\widetilde{J}(\xi)$, and use this upper bound as surrogate cost:

$$\begin{aligned}\widetilde{J}(\xi) &\triangleq \frac{1}{N}\sum_{k=1}^N \widetilde{J}_k(\xi) = \frac{1}{N}\sum_{k=1}^N \mathsf{E}\left[\left(v_\xi\left(\boldsymbol{s}_t\right) - \boldsymbol{y}_{k,t}\right)^2\right] \\ &\geq \mathsf{E}\left[\left(\frac{1}{N}\sum_{k=1}^N \left(v_\xi\left(\boldsymbol{s}_t\right) - \boldsymbol{y}_{k,t}\right)\right)^2\right] = J(\xi).\end{aligned} \tag{23}$$

Now, we can apply *diffusion* stochastic-gradient-descent (SGD) strategies [20], which consist of two steps: *adaptation* and *combination.* During the *adaptation* step, each agent performs SGD on its individual cost, $\widetilde{J}_k$, to obtain some intermediate-parameter update. Then, each agent *combines* the intermediate-parameters from its neighbors. These two steps are described by the following updates, which run in parallel for all agents $k = 1, \ldots, N$:

$$\widehat{\xi}_{k,i+1} = \xi_{k,i} - \alpha_{i+1}\widehat{\nabla}_\xi \widetilde{J}_k(\xi_{k,i}), \tag{24a}$$

$$\xi_{k,i+1} = \sum_{l \in \mathcal{N}_k} c_{lk}\widehat{\xi}_{l,i+1}, \tag{24b}$$

where i is the iteration index; α_i is the step-size; and $\widehat{\nabla}_\xi \widetilde{J}_k(\xi_{k,i})$ is the stochastic gradient evaluated at $\xi_{k,i}$, estimated from samples $\left\{\left(s_{k,t}, a_{k,t}, r_{k,t+1}, s_{k,t+1}\right)\right\}_{t=0}^{T_{k,i}}$ of the i-th episode, of length $T_{k,i}$, gathered by the k-th agent. We use Monte Carlo estimates for the target $y_{k,t} = \sum_{j=t}^{T_{k,i}} \gamma^{j-t} r_{k,j+1}$ (for simplicity), where $r_{k,j+1} \triangleq r_{\theta_k}(s_{k,j}, a_{k,j})$ is a shorthand. Then, the stochastic gradient is given by:

$$\widehat{\nabla}_\xi \widetilde{J}_k(\xi_{k,i}) = \frac{1}{T_{k,i}} \sum_{t=0}^{T_{k,i}} \nabla_\xi v_{\xi_{k,i}}\left(s_{k,t}\right)\left(v_{\xi_{k,i}}\left(s_{k,t}\right) - y_{k,t}\right). \tag{25}$$

We remark that each agent learns from its current episode, without replay buffer, similar to A3C [14], but in a fully distributed fashion, as opposed as having multiple threads updating the same neural network at a single location.

5.2 Distributed policy gradient: Actor

For large state-action sets, it is convenient to approximate the policy with a parametric function. Again, we consider expressive deep neural networks for the policy. From (16), we can rewrite the Lagrangian as:

$$L(v, \pi, \rho) = \overline{\mu}^\top v + \sum_{(s,a) \in \mathbb{S} \times \mathbb{A}} \pi(a|s)\rho(s)A(s,a). \tag{26}$$

Let $\pi_w \approx \pi$ denote the parametric approximation of the actual policy, where $w \in \mathbb{R}^{M_\pi}$ is the parameter vector of length M_π. Replacing π with π_w in (26), we obtain an approximate Lagrangian, $\widetilde{L}(v, w, \rho) \approx L(v, \pi, \rho)$, of the form:

$$\widetilde{L}(v, w, \rho) = \overline{\mu}^\top v + \sum_{(s,a) \in \mathbb{S} \times \mathbb{A}} \pi_w(a|s)\rho(s)A(s,a). \tag{27}$$

Thus, in order to approximate the saddle point condition (13), we can move in the ascent direction of the gradient of (27) w.r.t. the policy parameter, which is given by:

$$\begin{aligned}\nabla_w \widetilde{L}(v, w, \rho) &= \nabla_{\pi_w} \widetilde{L}(v, w, \rho)\nabla_w \pi_w \\ &= \left(\nabla_w \pi_w(a_1|s_1), \ldots, \nabla_w \pi_w(a_{|\mathbb{A}|}|s_{|\mathbb{S}|})\right)^\top \\ &\quad \left(\frac{\partial \widetilde{L}(v, w, \rho)}{\partial \pi_w(a|s)}\right)_{(s,a) \in \mathbb{S} \times \mathbb{A}} \\ &= \sum_{(s,a) \in \mathbb{S} \times \mathbb{A}} \nabla_w \pi(a|s)\frac{\partial L(v, w, \rho)}{\partial \pi_w(a|s)} \\ &= \sum_{s \in \mathbb{S}} \rho(s) \sum_{a \in \mathbb{A}} \nabla_w \pi_w(a|s)A(s,a) \\ &= \sum_{s \in \mathbb{S}} \rho(s) \sum_{a \in \mathbb{A}} \pi_w(a|s)\nabla_w \log \pi_w(a|s)A(s,a),\end{aligned} \tag{28}$$

where we used: $\nabla_w \pi_w(a|s) = \pi_w(a|s)\nabla_w \log \pi_w(a|s)$.

Since we have replaced d with two variables, π_w and ρ, we also require an optimality condition for ρ, which is given by:

$$\nabla_\rho \widetilde{L}(v, w, \rho) = \sum_{(s,a)\in\mathbb{S}\times\mathbb{A}} \pi_w(a|s)A(s,a) = 0 \tag{29}$$

Note that (29) is the same policy evaluation condition provided by the critic, so we do not need to compute it again.

Interestingly, (28) is similar to previous *policy gradient* theorems [22], with the important difference that it yields the advantage function explicitly; while previous works motivated the *baseline* mainly as a variance reduction technique [2, 30].

In order to derive a fully *distributed* actor, let us write the multitask advantage function (19) as the convex combination of advantage functions for the individual tasks:

$$A(s,a) = \frac{1}{N}\sum_{k=1}^{N} A_k(s,a), \tag{30}$$

$$A_k(s,a) \triangleq r_{\theta_k}(s,a) + \gamma \sum_{s'\in\mathbb{S}} \mathcal{P}_{\theta_k}(s'|s,a)v(s') - v(s). \tag{31}$$

Hence, we write the approximate Lagrangian for each task:

$$\widetilde{L}_k(v, w, \rho) \triangleq \mu_{\theta_k}^\top v + \sum_{(s,a)\in\mathbb{S}\times\mathbb{A}} \pi_w(a|s)\rho(s)A_k(s,a), \tag{32}$$

such that $\widetilde{L}(v,w,\rho) = \frac{1}{N}\sum_{k=1}^{N} \widetilde{L}_k(v,w,\rho)$.

Similar to the critic, once we have expressed the multitask approximate Lagrangian as the convex combination of the approximate Lagrangian of each individual task, we can apply diffusion SGD to perform the actor update, with smaller step-size, $\beta_{i+1} \leq \alpha_{i+1}$, to approximate convergence of the critic at every actor update:

$$\widehat{w}_{k,i+1} = w_{k,i} + \beta_{i+1}\widehat{\nabla}_w \widetilde{L}_k(v_{\xi_{k,i}}, w_{k,i}, \rho), \tag{33a}$$

$$w_{k,i+1} = \sum_{l\in\mathcal{N}_k} c_{lk}\widehat{w}_{l,i+1}, \tag{33b}$$

where each agent estimates its stochastic gradient as:

$$\widehat{\nabla}_w \widetilde{L}_k(v_{\xi_{k,i}}, w, \rho) = \frac{1}{T_{k,i}}\sum_{t=0}^{T_{k,i}} \nabla_w \log \pi_w(a_{k,t}|s_{k,t})\widehat{A}_{k,t}, \tag{34}$$

and $\widehat{A}_{k,t}$ can be any approximation of the advantage function [21]. We use the simple estimate:

$$\widehat{A}_{k,t} = \sum_{j=t}^{T_{k,i}} \gamma^{j-t} r_{k,j+1} - v_{\xi_{k,i}}(s_{k,t}). \tag{35}$$

Two remarks: *i)* In order to simplify the implementation, we set the target $y_{k,t}$ to be the empirical return, so that the stochastic gradient of the critic in (25) is the negative advantage estimate: $\widehat{\nabla}_\xi \widetilde{J}_k(\xi_{k,i}) = -\widehat{A}_{k,t}$. *ii)* Note that replacing $\xi_{k,i}$ with $\xi_{k,i+1}$ in (33a)–(33b) and (34)–(35), in a Gauss-Seidel fashion, might lead to faster convergence.

A detailed description of Diff-DAC is given in Algorithm 1.

6 NUMERICAL EXPERIMENTS

We evaluate the performance of Diff-DAC on three MRL problems of varying levels of difficulty. We use $\gamma = 0.99$ for all tasks:

Cart-pole balance: We use the OpenAI Gym [5] implementation, but with continuous force. The action follows a Gaussian distribution with mean in the interval $[-10, 10]$. The episode finishes when the pole is beyond 12 degrees from vertical, cart moves more than 2.4 units from the center, or run for 200 time-steps. The single task uses parameters $(0.1, 0.5, 1.0)$ for the pole mass, pole half-length and cart mass, respectively. The MRL problem consists of 25 tasks: pole mass in $\{0.1, 0.325, 0.55, 0.775, 1\}$, pole length in $\{0.05, 0.1625, 0.275, 0.3875, 0.5\}$, and cart mass 1.

Inverted pendulum: The pendulum consists of a rigid pole and an actuated joint, with maximum torque clipped to interval $[-2, 2]$. The pendulum starts at a random angle in $[-\pi, \pi]$, with uniform distribution. The goal is to take the pendulum to the upright position and balance. The MRL problem consists of 25 tasks with mass in $\{0.8, 0.9, 1.0, 1.1, 1.2\}$, and length in $\{0.8, 0.9, 1.0, 1.1, 1.2\}$. The single task pole mass and length are $(1.0, 1.0)$.

Cart-pole swing-up: We extend cart-pole balance to the case where the pole starts from the bottom and the task is to swing up the pole to the upright position and balance. The reward function

Algorithm 1: Diff-DAC. This algorithm runs in parallel at every agent $k = 1, \ldots, N$.

```
Input: Maximum number of episodes $E$, maximum number of steps per episode $T$, learning rate sequences $(\alpha_i, \beta_i)$.
Initialize critic, $v_{\xi_{k,0}}$, and actor, $\pi_{w_{k,0}}$, networks, $\forall k \in \mathcal{N}$.
Initialize episode counter, $i = 0$.
while $i < E$:
    Initialize empty trajectory, $\mathbb{M}_k = \{\}$.
    Initialize step counter: $t = 0$.
    Observe $s_{k,0}$.
    while $t < T$ and not terminal state:
        Select action $a_{k,t} \sim \pi_{w_{k,t}}(\cdot|s_{k,t})$.
        Execute $a_{k,t}$ and observe $r_{k,t+1}$ and $s_{k,t+1}$.
        Store tuple $(s_{k,t}, a_{k,t}, r_{k,t+1}, s_{k,t+1})$ in $\mathbb{M}_k$.
        Update step counter: $t \leftarrow t + 1$.
    end while
    for each sample $t \in \mathbb{M}_k$:
        Compute advantage function:
        $\widehat{A}_{k,t} = \sum_{j=t}^{|\mathbb{M}_k|} \gamma^{j-t} r_{k,j+1} - v_{\xi_{k,i}}(s_{k,t})$
    end for
    Compute distributed critic gradient:
        $\widehat{\xi}_{k,i+1} = \xi_{k,i} + \frac{\alpha_{i+1}}{|\mathbb{M}_k|}\sum_{t=0}^{|\mathbb{M}_k|} \nabla_\xi v_{\xi_{k,i}}(s_{k,t})\widehat{A}_{k,t}$
        $\xi_{k,i+1} = \sum_{l\in\mathcal{N}_k} c_{lk}\widehat{\xi}_{l,i+1}$
    Compute distributed actor update:
        $\widehat{w}_{k,i+1} = w_{k,i} + \frac{\beta_{i+1}}{|\mathbb{M}_k|}\sum_{t=0}^{|\mathbb{M}_k|} \nabla_w \log \pi_{w_{k,i}}(a_{k,t}|s_{k,t})\widehat{A}_{k,t}$
        $w_{k,i+1} = \sum_{l\in\mathcal{N}_k} c_{lk}\widehat{w}_{l,i+1}$
    Update episode counter: $i \leftarrow i + 1$.
end while
Return: Critic and actor weights: $\xi_{k,E}$, $w_{k,E}$.
```

is $r = \frac{2}{1+e^d} + \cos(\psi)$, where d is the Euclidean distance of the pole from the track center and upright position, and ψ is the pole angle. This is a much more difficult task than standard cart-pole and more difficult than the inverted pendulum due to more complex dynamics. The cart-pole swing-up MRL problem consists of 25 tasks, where pole mass is in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, pole half-length is in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, and cart mass is 0.5. The single task uses parameters $(0.5, 0.25, 0.5)$ for the pole mass, pole half-length and cart mass respectively.

We compare Diff-DAC with Dist-MTLPS for the MRL problem in the cart-pole balance environment. In particular, we compare against two variants of Dist-MTLPS, which consist in using two standard policy search methods, namely Reinforce [30] and PoWER [12], for solving the individual tasks. We only compare Diff-DAC with Dist-MTLPS in the cart-pole balance task, since the other two environments are uncontrollable with linear policies from raw data (since, e.g., the force with which the cart supports the pole is a nonlinear function of the system state) and nonlinear features are required [6, 19]. Our goal is to compare the performance of the single but expressive neural network policy provided by Diff-DAC with the task-specialized but less expressive linear policies provided by Dist-MTLPS.

We also compare Diff-DAC with Cent-AC, which has only one agent (central coordinator) that gathers and process samples from all the tasks synchronously and has the same neural network architectures and hyperparameters as Diff-DAC. We remark that we use two versions of exactly the same vanilla actor-critic algorithm, where their only difference is whether there is a single agent with access to all the data (Cent-AC) or multiple networked agents with access to local datasets (Diff-DAC).

Although experimenting with more environments and benchmarking with more algorithms would be helpful, testing in these simple environments is already useful, as they illustrate the behavior of the algorithms, without having to handle complex neural network architectures.

The network consists of $N = 25$ agents, randomly deployed in a 2D world, with average degree $|\overline{\mathcal{N}}| \triangleq \sum_{k=1}^{N} |\mathcal{N}_k| \approx 4.2$. In particular, the 2D world is a unit square; two agents are connected if their distance is smaller than $\sqrt{\left(|\overline{\mathcal{N}}|^\star + 1\right)/(N\pi)}$, where $|\overline{\mathcal{N}}|^\star$ is some target average degree that we want to approximate. We tried as many random deployments as needed until obtaining a connected network. To test the effect of the network topology on performance, we also include two additional networks of $N = 25$ and $|\overline{\mathcal{N}}| = 7.4$, and $N = 100$ and $|\overline{\mathcal{N}}| = 20$ (see results in Figure 5). Matrix C was obtained with the Hastigs-rule [20, p.492] in all cases, so that (4)–(5) hold. We remark that the network topologies used in the experiments are not related to any form of task similarity, but they reflect the sparse connectivity that appears naturally when agents and data are geographically distributed.

The critic and actor neural networks consist of 2 hidden layers of 400 neurons each with ReLu activation functions. The output layer for the critic network is linear. The output of the actor network includes a *tanh* activation function that determines the mean of a normal distribution, and a *Softplus* activation function that determines the variance for the normal distribution. We also included an extra penalty in the loss function equal to the entropy of the policy, with penalty coefficient 0.0005. Thus, both the mean and the variance are learned for the policy. We use ADAM optimizer [11], with learning rate 0.01 for critic and 0.001 for actor. Diff-DAC performs a learning step ($i \longleftarrow i + 1$) every fifth episode.

The return of the tasks is reported as the (undiscounted) total rewards every 20 episodes and is averaged over 10 test episodes at each point. Figures show the median and first and third quartiles of the distribution of the average return of all the tasks. Each epoch consists of 5 episodes per epoch and per agent in Diff-DAC, and $5N$ episodes in total per epoch for Dist-MTLPS and Cent-AC, so that the three algorithms simulate the same number of episodes. Every experiment was repeated at least 6 times.

In Figure 2 (bottom), we observe that Diff-DAC learns faster than Dist-MTLPS Reinforce and reaches better asymptotic performance. Dist-MTLPS PoWER converges faster than Diff-DAC, however the asymptotic performance of the latter is much better. This is remarkable since Dist-MTLPS learns one different policy per task, while Diff-DAC learns a single policy common to all tasks. We remark that both PoWER and Reinforce—used by Dist-MTLPS for solving the individual tasks—are able to achieve optimal asymptotic performance in the standard cart-pole balance single-task (similar to the results in Fig. 2-top).

In Figures 2–4, we also observe that Diff-DAC converges slower than the Cent-AC, which was expected since the latter can compute the gradients with data from all tasks at every iteration, while the former has to wait until the parameters are diffused across the network. However, Diff-DAC usually achieves similar or higher asymptotic performance and more stability, in both single task and multitask problems, which was also expected due to the already reported enhanced robustness against local optima of diffusion strategies for nonconvex optimization problems [26, Ch. 4]. This is shown in Figure 2 (top) and Figures 3 (top and bottom), where Cent-AC tends to reach the optimal faster, but is unstable; and in Figures 2 (bottom) and Figures 4 (top and bottom) where the asymptotic performance of Diff-DAC is always higher than Cent-AC. We guess that the instability of Cent-AC may be alleviated by adding a *replay buffer* [13, 15] or randomizing agents' samples to reduce their correlation, simulating asynchronicity (similar to asynchronous methods like A3C [14]). However, our goal with these experiments is not to compare with SOTA centralized algorithms—which use several advancements to stabilize or improve performance—, but to evaluate whether diffusion is not only a feasible distributed strategy but also a valid alternative to stabilize learning. Thus, we decided to use vanilla central actor-critic vs. vanilla distributed actor-critic, where their only difference is having a single agent with all the data vs. having multiple agents with local datasets.

Finally, Figure 5 shows a simple experiment that studies the influence of the network topology. We evaluate Diff-DAC for the single-task cart-pole balance problem and see that for the same network size, $N = 25$, a relatively sparse network, $|\overline{\mathcal{N}}| \approx N/6$, achieves performance similar to a more dense network, $|\overline{\mathcal{N}}| \approx N/3$. In addition, we see that larger number of agents $N = 100$ improves the asymptotic performance.

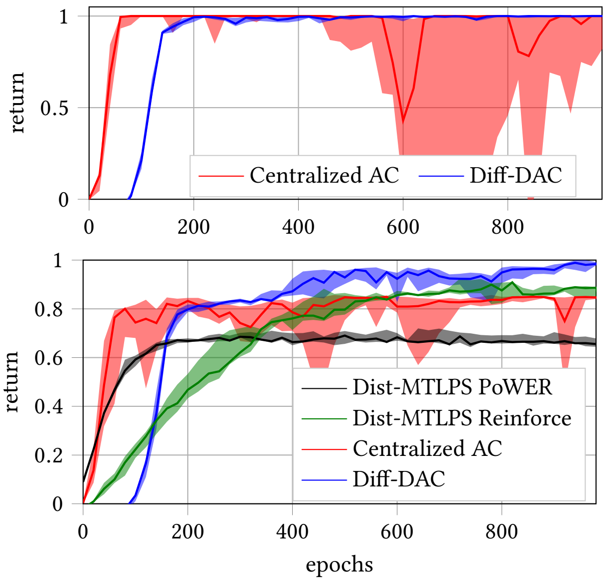

Figure 2: Cart-pole balance with continuous action for single-task (top) and multitask (bottom). Cent-AC is faster than distributed approaches, but Diff-DAC achieves the best asymptotic performance.

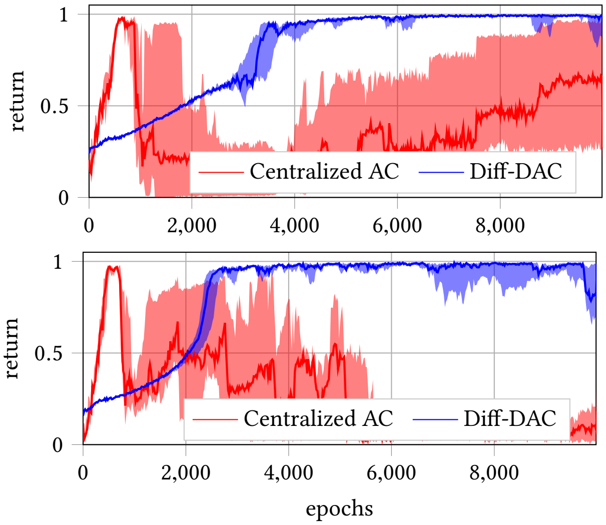

Figure 3: Inverted pendulum for single-task (top) and multitask (bottom). Diff-DAC learns in both the single-task and multitask robustly. The central method learns the task quickly but is unstable.

7 CONCLUSIONS

We considered MRL where tasks are parametrized MDPs with parameters drawn from some distribution, and we derived an algorithm that learns a policy that performs well on average for the observed set of tasks. We defined average global variables that allowed us to use standard optimal control theory and reformulate our MRL problem as an LP. From this LP, we derived an exact, model-based actor-critic algorithm as an instance of dual ascent for finding the saddle point of the Lagrangian. This saddle-point derivation is interesting in itself and provides novel insights in the actor-critic framework. By approximating the exact method with deep neural networks, we obtained the Diff-DAC algorithm, which can scale to large number of tasks.

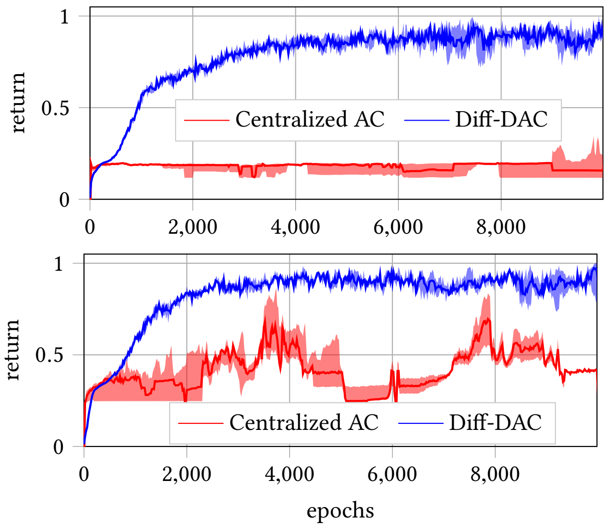

Figure 4: Cart-pole swing-up for single-task (top) and multitask (bottom). Diff-DAC learns to swing-up and balance the pole consistently, while Cent-AC achieves much inferior performance.

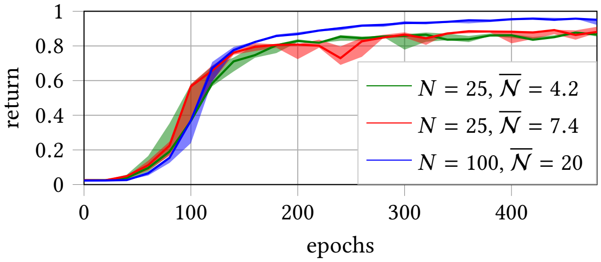

Figure 5: Influence of network topology in single-task cart-pole balance with continuous action. Diff-DAC combines the experience of all agents, relatively insensitive to network sparsity.

Simulation results showed that Diff-DAC can be faster and achieve higher asymptotic performance than the state of the art distributed algorithm for solving the MRL problem (i.e., Dist-MTLPS). This is a remarkable result since the Diff-DAC agents converge to a single common policy that behaves better than the task-dependent linear policies obtained by Dist-MTLPS. Moreover, Diff-DAC can solve complex problems that are uncontrollable from raw data by linear policies, while Dist-MTLPS requires (usually costly) feature engineering. Diff-DAC is also very stable and achieves similar or usually higher asymptotic performance than the centralized approach in both single and multitask problems. This suggests that the sparse connectivity among agents induces a regularization effect that helps them achieve better local optimum. We consider this form of regularization an interesting line of research.

Diff-DAC can also be extended to *zero-shot learning* by taking the task parameter θ_k as an additional input to each agent's value/policy networks, so that when a new task appears, it can input its parameter in the neural networks and adapt its behavior to this task without further training.

Finally, it would be interesting to apply the proposed framework to derive distributed variants of other algorithms like PPO [21] or ACKTR [31].

8 ACKNOWLEDGEMENTS

We thank Haitham Bou-Ammar for insightful discussions.

This work was partially supported by the Spanish Ministry of Science and Innovation under the grant TEC2016-76038-C3-1-R (HERAKLES) and the COMONSENS Network of Excellence TEC2015-69648-REDC.

REFERENCES

[1] K. J. Arrow, L. Hurwicz, and H. Uzawa. 1958. *Studies in Linear and Non-linear Programming*. Stanford University Press.

[2] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. 2009. Natural Actor-critic Algorithms. *Automatica* 45, 11 (Nov. 2009), 2471–2482.

[3] H. Bou-Ammar, E. Eaton, P. Ruvolo, and M. Taylor. 2014. Online multi-task learning for policy gradient methods. In *Proc. Int. Conf. on Machine Learning (ICML)*. 1206–1214.

[4] S.P. Boyd and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[6] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. 2015. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 2 (2015), 408–423.

[7] S. El Bsat, H. Bou-Ammar, and M. E. Taylor. 2017. Scalable Multitask Policy Gradient Reinforcement Learning.. In *AAAI Conf. on Artificial Intelligence (AAAI)*. 1847–1853.

[8] R.A. Horn and C.R. Johnson. 1990. *Matrix Analysis*. Cambridge University Press.

[9] S. Kar, J. M. F. Moura, and H. V. Poor. 2013. QD-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through Consensus + Innovations. *IEEE Transactions on Signal Processing* 61, 7 (2013), 1848–1862.

[10] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[11] DP Kingma and J. L. Ba. 2015. Adam: A method for stochastic optimization. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

[12] J. Kober and J. R. Peters. 2009. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems (NIPS)*. 849–856.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. Continuous control with deep reinforcement learning. (2015).

[14] V. Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[16] E. Parisotto, J. L. Ba, and R. Salakhutdinov. 2016. Actor-mimic: Deep multitask and transfer reinforcement learning. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

[17] D. Pfau and O. Vinyals. 2016. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945* (2016).

[18] M. L. Puterman. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (2nd ed.). John Wiley & Sons.

[19] Tapani Raiko and Matti Tornio. 2009. Variational Bayesian learning of nonlinear hidden state-space models for model predictive control. *Neurocomputing* 72, 16-18 (2009), 3704–3712.

[20] A. H. Sayed. 2014. Adaptation, learning, and optimization over networks. *Foundations and Trends in Machine Learning* 7, 4-5 (2014), 311–801.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[22] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*. 1057–1063.

[23] M. E. Taylor and P. Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, Jul (2009), 1633–1685.

[24] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. 2017. Distral: Robust Multitask Reinforcement Learning. *arXiv preprint arXiv:1707.04175* (2017).

[25] R. Tutunov, H. Bou-Ammar, and A. Jadbabaie. 2016. An exact distributed newton method for reinforcement learning. In *IEEE Conf. on Decision and Control (CDC)*. 1003–1008.

[26] Sergio Valcarcel Macua. 2017. *Distributed optimization, control and learning in multiagent networks*. Ph.D. Dissertation. Universidad Politécnica de Madrid.

[27] Sergio Valcarcel Macua, J. Chen, S. Zazo, and A. H. Sayed. 2015. Distributed Policy Evaluation Under Multiple Behavior Strategies. *IEEE Trans. Automat. Control* 60, 5 (May 2015), 1260–1274.

[28] R. van der Meulen. 2015. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. http://www.gartner.com/newsroom/id/3165317.

[29] E. Wei and A. Ozdaglar. 2012. Distributed alternating direction method of multipliers. In *IEEE Annual Conf. Decision and Control (CDC)*. 5445–5450.

[30] R. J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[31] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in neural information processing systems (NIPS)*. 5285–5294.

[32] X. Zhao and A. H. Sayed. 2012. Performance Limits for Distributed Estimation Over LMS Adaptive Networks. *IEEE Transactions on Signal Processing* 60, 10 (2012), 5107–5124.

[33] X. Zhao and A. H. Sayed. 2015. Asynchronous Adaptation and Learning Over Networks—Part I: Modeling and Stability Analysis. *IEEE Transactions on Signal Processing* 63, 4 (Feb 2015), 811–826.