TEAM POLICY LEARNING FOR MULTI-AGENT REINFORCEMENT LEARNING

Lucas Cassano[⋆†]*, Sulaiman A. Alghunaim*[⋆†] *and Ali H. Sayed*[†]

[⋆]Department of Electrical and Computer Engineering, University of California, Los Angeles
[†]School of Engineering, École Polytechnique Fédérale de Lausanne, Switzerland

ABSTRACT

This work presents a fully distributed algorithm for learning the optimal policy in a multi-agent cooperative reinforcement learning scenario. We focus on games that can only be solved through coordinated team work. We consider situations in which K players interact simultaneously with an environment and with each other to attain a common goal. In the algorithm, agents only communicate with other agents in their immediate neighborhood and choose their actions independently of one another based only on local information. Learning is done off-policy, which results in high data efficiency. The proposed algorithm is of the stochastic primal-dual kind and can be shown to converge even when used in conjunction with a wide class of function approximators.

***Index Terms*—** Reinforcement learning, multi-agent learning, off-policy, optimal policy, distributed algorithm

1. INTRODUCTION

We consider the problem of finding the optimal team policy in a multi-agent reinforcement learning (MARL) scenario in a fully distributed manner. The situation under consideration is one in which all agents form a team (to solve a common objective) and make decisions independently of one another based on their local information with the aim of maximizing the team's performance. There are various applications that follow this general framework. Some examples are teams of robots whose objective might be to put out a fire, to catch a criminal or to move a bulky item.

The contribution of this paper is the introduction of *Diffusion for Team Policy Optimization* (DTPO), an algorithm that achieves state of the art performance in team games. The algorithm is of the off-policy kind (similar to *Q-learning*) and has convergence guarantees (even when used in conjunction with a wide class of function approximators, although in this paper we only derive the tabular case due to length constraints).

Notation: Matrices and vectors are denoted by upper case and lower case letters, respectively. Bold font and calligraphic font are used to denote random variables and sets, respectively. $\mathbb{E}_g$ is the expected value with respect to distribution g.

1.1. RELATION TO PRIOR WORK

There is a considerable body of work in MARL. Some of the notable works include *Team Q-learning* [1], *Distributed Q-learning* [2], *FMQ* [3], *Hyper-Q learning* [4], *OAB* [5], *Hysteretic Q-learning* [6], and the two algorithms presented in [7]. All these works are based on Q-learning and unlike our algorithm can diverge when used in conjunction with function approximation (which is fundamental in real world applications). Moreover, as opposed to our work, these algorithms do not consider the possibility of communication among agents. Furthermore, references [1, 2, 3, 5, 6] only consider the situation in which the reward is equal for all agents. *Hyper-Q learning* also has the inconvenience that every agent needs to have an estimate of the parameterized and further discretized policies of all other agents. Works [2, 6] have the limitation that they only work in deterministic environments. The algorithms presented in [7] have two difficulties: in the first place, precise synchronization for the exploration periods among all agents is required, and secondly they converge to any Nash equilibrium instead of the optimal Nash equilibrium. Note that when there are multiple Nash equilibria, many of these can be highly suboptimal strategies. We illustrate this possibility in the simulation. The work [15] studies a similar problem to the one considered in this work but relies on the policy gradient theorem and on actor-critic algorithms. These schemes work on-policy and are data inefficient. More critically, the algorithms are only guaranteed to converge to equilibrium points that can be highly suboptimal in team games where coordination is necessary — see the example in the simulation section.

2. PROBLEM FORMULATION

We consider a team of K decision makers (also referred to as agents) that form a network. The network is represented by a graph in which the edges represent the communication links. Agent k communicates only with a subset of the agents in the network which we refer to as the neighborhood of k (denoted as $\mathcal{N}_k$). The topology of the network is determined by some combination matrix C whose kn-th entry (denoted by c_{kn}) is a scalar with which agent n weights information incoming from agent k (note that $c_{kn} \neq 0 \iff k \in \mathcal{N}_n$). We make the following assumption about the network structure (which will be used in Lemma 3).

Assumption 1. *We assume that the network is strongly-connected. This implies that there is at least one path from any node to any other node and that at least one node has a self-loop (i.e., at least one agent uses its own information). We further assume that the combination matrix C is symmetric and doubly-stochastic.*

The agents interact with an environment and with each other. We model their behavior as a Markov Decision Process (MDP), which is defined by the tuple $(\mathcal{S},\mathcal{A}^k,\mathcal{P},r^k)$. The symbol $\mathcal{S}$ is a set of global states shared by all agents of size $S = |\mathcal{S}|$ and $\mathcal{A}^k$ is the set of actions available to agent k of size $A^k = |\mathcal{A}^k|$. We refer to $\mathcal{A} = \prod_{k=1}^{K} \mathcal{A}^k$ as the set of team actions. Moreover, $\mathcal{P}(s'|s,a)$ specifies the probability of transitioning to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ having taken team action $a \in \mathcal{A}$ and $r^k : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function of agent k. Specifically, $r^k(s,a,s') = \mathbb{E}\,\boldsymbol{r}^k(s,a,s')$ is the expected reward of decision maker k when the team transitions to

Emails: {cassanolucas,salghunaim}@ucla.edu and ali.sayed@epfl.ch.

state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ having taken team action $a \in \mathcal{A}$. We clarify that we refer to the team's action (i.e., the collection of all individual actions) as a, while a^k refers to the individual action of agent k. It is important to note that the transition probabilities and the reward functions of the individual agents depend not only on their own actions but on the actions of all other agents. The goal of all agents is to maximize the aggregated return defined as:

$$J(\pi) = \sum_{t=0}^{\infty} \frac{\gamma^t}{K} \left(\sum_{k=1}^{K} \mathbb{E}_{\pi,\mathcal{P}} \left[\boldsymbol{r}^k(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}}) \right] \right) \tag{1}$$

where s_t and a_t are the state and actions at time t, respectively, $\pi(a|s)$ is the team's policy and $\gamma \in [0, 1)$ is the discount factor. Accordingly, the value function is defined by [8]:

$$v^{\pi}(s) \stackrel{\Delta}{=} \sum_{t=0}^{\infty} \frac{\gamma^t}{K} \sum_{k=1}^{K} \mathbb{E}_{\pi,\mathcal{P}} \left(\boldsymbol{r}^k(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{s_{t+1}}) \big| \boldsymbol{s_0} = s \right) \tag{2}$$

which satisfies the Bellman equation [8]:

$$v^{\pi}(s) = \mathbb{E}_{\boldsymbol{a}\sim\pi} \left[\frac{1}{K} \sum_{k=1}^{K} \boldsymbol{r}^k(s, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\pi}(\boldsymbol{s'}) \right] \tag{3}$$

where we defined $\boldsymbol{r}^k(s, a) = \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} \boldsymbol{r}^k(s, a, \boldsymbol{s'})$. The optimal value function and optimal policies[1] satisfy:

$$\pi^{\bullet}(a|s) = \arg\max_{\pi(a|s)} \mathbb{E}_{\boldsymbol{a}\sim\pi} \left[\frac{1}{K} \sum_{k=1}^{K} \boldsymbol{r}^k(s, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\bullet}(\boldsymbol{s'}) \right] \tag{4a}$$

$$v^{\bullet}(s) = \max_{\pi(a|s)} \mathbb{E}_{\boldsymbol{a}\sim\pi} \left[\frac{1}{K} \sum_{k=1}^{K} \boldsymbol{r}^k(s, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\bullet}(\boldsymbol{s'}) \right] \tag{4b}$$

The $\max$ operator in equations (4) is inconvenient because it is non-differentiable and in this paper we are interested in deriving gradient algorithms. To circumvent this issue we use a differentiable approximation to the $\max$ and define the quasi-optimal value function as:

$$v^{\star}_{\lambda}(s) = \lambda(s) \log \left\{ \sum_{a} \exp \left(\frac{K^{-1} \sum_{k=1}^{K} r^k(s,a) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\star}_{\lambda}(\boldsymbol{s'})}{\lambda(s)} \right) \right\} \tag{5}$$

where $\lambda(s) > 0$ is a temperature parameter that controls the accuracy of the approximation. Note that $v^{\star}_{\lambda}(s)$ in (5) can equivalently be defined using the conjugate of the log-sum-exp function as [9]:

$$v^{\star}_{\lambda}(s) = \max_{\pi(a|s)} \mathbb{E}_{\boldsymbol{a}\sim\pi} \left[\sum_{k=1}^{K} \frac{r^k(s,\boldsymbol{a})}{K} - \lambda(s) \log \pi(\boldsymbol{a}|s) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\star}_{\lambda}(\boldsymbol{s'}) \right] \tag{6}$$

The $\max$ in (6) can easily be solved by differentiating and equating to zero. Doing so gives the policy corresponding to (6), which we refer to as the quasi-optimal policy:

$$\pi^{\star}_{\lambda}(a|s) = \frac{\exp \left[\lambda(s)^{-1} \Big(K^{-1} \sum_{k=1}^{K} r^k(s,a) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\star}_{\lambda}(s') \Big) \right]}{\sum_{a} \exp \Big[\lambda(s)^{-1} \Big(K^{-1} \sum_{k=1}^{K} r^k(s,a) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v^{\star}_{\lambda}(s') \Big) \Big]} \tag{7}$$

[1] Note that an MDP may have many optimal policies; at least one of which is deterministic and chooses an optimal action with probability one. Without loss of generality, we assume that there is a unique optimal policy.

Remark 1. *Note that* $\lim_{\lambda(s)\to 0} \pi^{\star}_{\lambda}(a|s) = \pi^{\bullet}(a|s)$ *and* $\lim_{\lambda(s)\to\infty} \pi^{\star}_{\lambda}(a|s) = 1/A$.

We defined $\lambda(s)$ as a function of the state because it provides an effective way of trading exploration and exploitation [8]. The parameter $\lambda(s)$ can be initialized at high values to encourage exploration (since for high values of $\lambda(s)$ expression (7) tends to the uniform distribution) in unexplored states and slowly decays as those states are explored.

Theorem 1. *For small enough* $\lambda(s)$ *it holds that:*

$$\arg\max_{a} \pi^{\star}_{\lambda}(a|s) = \arg\max_{a} \pi^{\bullet}(a|s) \tag{8}$$

Proof. The proof is omitted due to length restrictions. ■

Since our main goal is to obtain the optimal policy $\pi^{\bullet}(a|s)$, Theorem 1 is of fundamental importance. This is because the theorem guarantees that for small enough $\lambda(s)$, an optimal action at every state can be extracted from $\pi^{\star}_{\lambda}(a|s)$ (and hence an optimal policy $\pi^{\bullet}(a|s)$ can be extracted). The following lemma is an extension of [10] to the multi-agent setting:

Lemma 1. $v^{\star}_{\lambda}(s)$ *and* $\pi^{\star}_{\lambda}(a|s)$ *are the only pair that satisfy the following consistency equation:*

$$v(s) - \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v(\boldsymbol{s'}) = \frac{1}{K} \sum_{k=1}^{K} r^k(s,a) - \lambda(s) \log \pi(a|s) \tag{9}$$

Using (9) we can write the following quadratic optimization problem (whose solution is $\pi^{\star}_{\lambda}(a|s)$):

$$\min_{\pi, v} \left(\frac{1}{K} \sum_{k=1}^{K} r^k(s,a) - \lambda(s) \log \pi(a|s) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v(\boldsymbol{s'}) - v(s) \right)^2 \tag{10}$$

Unfortunately (10) cannot be used to learn $\pi^{\star}_{\lambda}(a|s)$ because $r^k(s,a)$ and the transition kernel $\mathcal{P}$ are unknown and we want to derive a stochastic algorithm that learns from interactions with the environment. Before we proceed to propose an alternative cost function that relies on samples we note one key feature of (10), which is that there is no expectation taken with respect to the policy of any of the agents (the only expectation is taken with respect to $\mathcal{P}$, which depends on the MDP) and therefore transitions obtained following *any* policy are useful to learn $\pi^{\star}_{\lambda}(a|s)$ (this is what allows us to derive an off-policy algorithm). With this in consideration we propose the following cost:

$$\min_{\pi, v} \frac{1}{2} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\psi} \Big(\frac{1}{K} \sum_{k=1}^{K} r^k(\boldsymbol{s}, \boldsymbol{a}) - \lambda(\boldsymbol{s}) \log \pi(\boldsymbol{a}|\boldsymbol{s}) + \gamma \mathbb{E}_{\boldsymbol{s'}\sim\mathcal{P}} v(\boldsymbol{s'}) - v(\boldsymbol{s}) \Big)^2 \tag{11}$$

where the minimization variables π and v refer to $\pi(a|s)$ and $v(s)$ for all states and actions and ψ is some distribution under which (s, a) pairs are sampled. Distribution ψ is determined by two factors: the individual behavior policies of the agents under which data is collected and the experience selection strategy set for the replay buffers [11] of each of the agents. Replay buffers store past data collected by the agents which is later sampled by the learning algorithm and have been shown to be essential to achieve stable and fast learning [11], [12], [13]. The experience selection strategy refers to

the strategy that decides what data samples are stored in the buffer (not all data can be stored due to the finite memory availability of the buffer), and the strategy with which data is sampled from the buffer for training. A uniformly sampled First In First Out (FIFO) buffer is a popular experience selection strategy. An important characteristic of ψ is that the behavior policies used by the agents for data collection are potentially independent of one another, the only requirement for these policies is that every possible transition in the MDP has a positive probability of being sampled. This is a key feature for multi-agent learning because it means that data collection can be done in a fully distributed manner without any communication or synchronization requirements among the decision makers. Also different agents can have different replay buffer sizes with different experience selection strategies.

3. ALGORITHM DERIVATION

We now proceed to derive the algorithm. We start by expanding the square in (11):

$$\min_{\pi,v}\frac{1}{2}\mathbb{E}_{\psi}\Big[2\Big(\frac{1}{K}\sum_{k=1}^{K}r^k(\boldsymbol{s},\boldsymbol{a})\Big)\Big(\gamma\mathbb{E}_{\mathcal{P}}v(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi(\boldsymbol{a}|\boldsymbol{s})-v(\boldsymbol{s})\Big)$$
$$+\Big(\frac{1}{K}\sum_{k=1}^{K}r^k(\boldsymbol{s},\boldsymbol{a})\Big)^2+\Big(\gamma\mathbb{E}_{\mathcal{P}}v(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi(\boldsymbol{a}|\boldsymbol{s})-v(\boldsymbol{s})\Big)^2\Big] \quad (12)$$

Since the second term in (12) is independent of the optimization variables we can remove it without affecting the minimizer of the problem. To derive a distributed algorithm in which each agent uses only local information, we let $v^k(s)$ and $\pi^k(a|s)$ denotes the local copies of $v(s)$ and $\pi(a|s)$ available at k and instead solve the equivalent problem:

$$\min_{\pi^k,v^k}\mathbb{E}_{\psi}\Big[\sum_{k=1}^{K}r^k(\boldsymbol{s},\boldsymbol{a})\Big(\gamma\mathbb{E}_{\mathcal{P}}v^k(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi^k(\boldsymbol{a}|\boldsymbol{s})-v^k(\boldsymbol{s})\Big)$$
$$+\sum_{k=1}^{K}\frac{1}{2}\Big(\gamma\mathbb{E}_{\mathcal{P}}v^k(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi^k(\boldsymbol{a}|\boldsymbol{s})-v^k(\boldsymbol{s})\Big)^2\Big]$$
$$\text{s.t.}\qquad \pi^1=\cdots=\pi^K \qquad v^1=\cdots=v^K \quad (13)$$

As we mentioned in the previous section, we are interested in deriving a stochastic gradient algorithm compatible with function approximation. Problem (13) is not suitable for this because of the squared term. Due to such term, the gradient with respect to the variables that parameterize the value function is a product of expectations, and therefore a stochastic approximation of the gradient obtained with samples by removing such expectations would be biased. To address this issue we use the conjugate function [9] of the quadratic and obtain the following saddle-point formulation:

$$\min_{\pi^k,v^k}\sum_{k=1}^{K}\mathbb{E}_{\psi}\Big[r^k(\boldsymbol{s},\boldsymbol{a})\big(\gamma\mathbb{E}_{\mathcal{P}}v^k(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi^k(\boldsymbol{a}|\boldsymbol{s})-v^k(\boldsymbol{s})\big)$$
$$+\max_{\rho^k}\rho^k(\boldsymbol{s},\boldsymbol{a})\big(\gamma\mathbb{E}_{\mathcal{P}}v^k(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi^k(\boldsymbol{a}|\boldsymbol{s})-v^k(\boldsymbol{s})\big)-\frac{1}{2}\rho^k(\boldsymbol{s},\boldsymbol{a})^2\Big]$$
$$\text{s.t.}\qquad \pi^1=\cdots=\pi^K \qquad v^1=\cdots=v^K \quad (14)$$

Lemma 2. *The order of the expectation and* max *operator in* (14) *can be interchanged.*

Proof. The lemma can be proved by solving for both cases and checking that both solutions are equal. We omit the calculations due to length restrictions. ■

Algorithm 1: ***Distributed Team Policy Optimization*** at node k

Initialize: $(v^{k,0},\log\pi^{k,0},\rho^{k,0})$ and $\lambda(s)$ for all s.
For episodes $e=0,1,2\ldots$ **do**:
  For environment transitions $t=0,1,2\ldots,T$ **do**:
    Follow policy $\pi^{k,e}(a_t|s_t)$ and follow some store experience selection strategy to decide whether or not to store the t-th transition $[s_t,a_t,r^k(s_t,a_t),s'_t]$in the reply buffer $\mathcal{R}_k$.
    Diminish $\lambda(s_t)$
  For optimization iterations $i=0,1,2\ldots,I$ **do**:
    Arrange transitions from $\mathcal{R}_k$ into subsets $\boldsymbol{\sigma}_k(s,a)$ such that transition j belongs in $\boldsymbol{\sigma}_k(s,a)$ if $(s_j,a_j)=(s,a)$:
    For every (s,a) pair **do**:
      If $\boldsymbol{\sigma}_k(s,a)$ is not empty:

$$\nabla_{\rho^k(s,a)}L=\frac{1}{|\boldsymbol{\sigma}_k(s,a)|}\sum_{j\in\boldsymbol{\sigma}_k(s,a)}\Big[\gamma v^k(s'_j)-v^k(s_j)$$
$$-\lambda(s_j)\log\pi^k(a_j|s_j)-\rho^k(s_j,a_j)\Big]$$
$$\rho^{k,e+1}(s,a)=\rho^{k,e}(s,a)+\mu_\rho\nabla_{\rho^k(s,a)}L$$

    For every (s,a) pair **do**:
      If $\boldsymbol{\sigma}_k(s,a)$ is not empty:

$$\nabla_{\log\pi^k(a|s)}L=\frac{1}{|\boldsymbol{\sigma}_k(s,a)|}\sum_{j\in\boldsymbol{\sigma}_k(s,a)}\Big[r^k(s_j,a_j)+\rho^{k,e+1}(s_j,a_j)\Big]$$
$$\phi^{e+1}_{\pi,k}(a|s)=\log\pi^e_k(a|s)+\mu_\pi\nabla_{\log\pi^k(a|s)}L$$
$$\log\pi^{k,e+1}(a|s)=\sum_{n\in\mathcal{N}_k}c_{nk}\phi^{e+1}_{\pi,n}(a|s)$$

    For every s **do**:
      If $\boldsymbol{\sigma}_k(s)=\cup_{a\in\mathcal{A}}\boldsymbol{\sigma}_k(s,a)$ is not empty:

$$\nabla_{v(s)}L=\frac{1}{|\boldsymbol{\sigma}_k(s)|}\sum_{j\in\boldsymbol{\sigma}_k(s)}\Big[r^k(s_j,a_j)+\rho^{k,e+1}(s_j,a_j)\Big]$$
$$\phi^{e+1}_{v,k}(s)=v^{k,e}(s)+\mu_v\nabla_{v^k(s)}L$$
$$v^{k,e+1}(s)=\sum_{n\in\mathcal{N}_k}c_{nk}\phi^{e+1}_{v,n}(s)$$

Finally making use of Lemma 2 results in the following optimization problem:

$$\min_{\pi^k,v^k}\max_{\rho^k}\mathbb{E}_{\psi,\mathcal{P}}\sum_{k=1}^{K}\Big[r^k(\boldsymbol{s},\boldsymbol{a})\big(\gamma v^k(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi^k(\boldsymbol{a}|\boldsymbol{s})-v^k(\boldsymbol{s})\big)$$
$$+\rho^k(\boldsymbol{s},\boldsymbol{a})\big(\gamma v^k(\boldsymbol{s'})-\lambda(\boldsymbol{s})\log\pi^k(\boldsymbol{a}|\boldsymbol{s})-v^k(\boldsymbol{s})\big)-\frac{\rho^k(\boldsymbol{s},\boldsymbol{a})^2}{2}\Big]$$
$$\text{s.t.}\quad \pi^1=\cdots=\pi^K,\quad v^1=\cdots=v^K \quad (15)$$

Note that up to this point no approximations have been made and hence problems (15) and (11) are equivalent in the sense that the value of the optimizing variables π^k and π are the same. A similar saddle-point problem to (15) appears in [14], however, [14] deals with a single-agent scenario and $\lambda(s)=\lambda$ and hence cannot be used as an efficient exploration strategy.

Lemma 3. *We introduce the eigenvalue decomposition:*

$$0.5(I-C)=U\Sigma U^T \quad (16)$$

and define:

$$B\ \triangleq\ U\Sigma^{1/2}U^T \quad (17)$$
$$\bar{v}(s)\ \triangleq\ [v^1(s),\cdots,v^K(s)]^T \quad (18)$$
$$\bar{\pi}(a|s)\ \triangleq\ [\pi^1(a|s),\cdots,\pi^K(a|s)]^T \quad (19)$$

where U is an orthogonal matrix and $\Sigma^{1/2}$ is the element-wise square root of Σ, which is a diagonal matrix with non-negative entries, then under Assumption 1 it holds that [15]:

$$B\bar{v}(s) = 0 \iff v^1(s) = \cdots = v^K(s) \quad (21)$$

$$B \log \bar{\pi}(a|s) = 0 \iff \log \pi^1(a|s) = \cdots = \log \pi^K(a|s) \quad (22)$$

Finally, we approximate (15) by the following penalized formulation (for convenience we write it in vector form):

$$\min_{\pi^k, v^k} \max_{\rho^k} L(\pi^k, v^k, \rho^k)$$

$$L(\pi^k, v^k, \rho^k) = \mathbb{E}\Big[(\bar{r}(\boldsymbol{s},\boldsymbol{a}) + \bar{\rho}(\boldsymbol{s},\boldsymbol{a}))^T \big(\gamma \bar{v}(\boldsymbol{s'}) - \lambda(\boldsymbol{s}) \log \bar{\pi}(\boldsymbol{a}|\boldsymbol{s}) - \bar{v}(\boldsymbol{s})\big) - \frac{\|\bar{\rho}(\boldsymbol{s},\boldsymbol{a})\|^2}{2} + \eta_v \|B\bar{v}(\boldsymbol{s})\|^2 + \eta_\pi \|B \log \bar{\pi}(\boldsymbol{s},\boldsymbol{a})\|^2\Big] \quad (23)$$

where η_π and η_v are two non-negative constants, the matrix B, $\bar{v}(s)$, and $\bar{\pi}(a|s)$ are defined in Lemma 3 and:

$$\bar{\rho}(s,a) \triangleq [\rho^1(s,a), \cdots, \rho^K(s,a)]^T \quad (24)$$

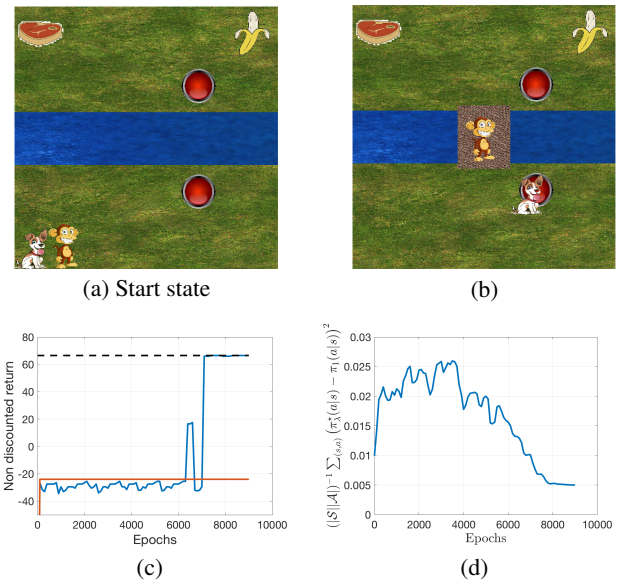
Using $2B^2 = I - C$ (from Lemma 3), choosing $\eta_\pi = \mu_\pi^{-1}$ and $\eta_v = \mu_v^{-1}$ (where μ_π and μ_v are two step-sizes that will be used in the update equations) and applying mini-batch stochastic gradient descent (and ascent) updates (incremental updates to π and v), we get our proposed algorithm (*see Algorithm 1*). Similar arguments to [16] can be used to prove convergence for small enough step-sizes μ_v, μ_π and μ_ρ, however due to length constraints the proof is omitted.

4. SIMULATION RESULTS

In this section we test the performance of DTPO and compare with [17, Algorithm 1], which we refer to as *Distributed Actor-critic* (DAC). Note that DAC is the only algorithm mentioned in the introduction suitable for this problem.

We consider a simple yet challenging team game. In this game, there are two agents (dog and monkey) who want to collect food (steak and banana) as fast as possible with as little movements as possible; to achieve this objective, they need to safely cross a river, which can only be done by collaboration. Figure 1 (a) illustrates the situation. Note that there are two buttons which when pressed a bridge to safely cross the river appears. In the optimal strategy, the dog presses the southern button for the monkey to cross the bridge which in turn presses the northern button so that the dog can cross (see Figure 1 (b)). Finally, each of the agents walks towards their respective food. At every time step, each of the agents has five possible actions: move north, east, south, west or stay (if an agent moves against an edge it just stays in place). When an agent falls into the river or collects its food it spawns back in the lower left corner. The reward structure for each of the agents is as follows: -1 is obtained at every time step, -0.5 every time the agent moves in any direction, -100 if it falls in the river and +50 if it collects its food (these rewards are additive, so if an agent moves and falls into the river it collects a rewards equal to -101.5). There are a total of 21 positions and 5 actions for each agent, hence the total state space for the team is $(21 \times 5)^2 = 11025$. The discount factor is set to $\gamma = 0.97$.

We implemented a game with only 2 agents because we wanted a game that could be implemented in tabular form (note that in this game, the state space grows exponentially with the number of agents, hence function approximation becomes necessary to reduce the dimensionality of the state space) so that the optimal policy would be attainable by the learning agents and could be calculated exactly to monitor the progress of the algorithm. Note that even with two agents, this game is extremely challenging to solve for fully distributed algorithms due to the fact that there is a highly suboptimal (yet easily attainable) Nash equilibrium, which is for both agents to stay still at all times in the lower left corner (which where they start). Therefore, algorithms that only guarantee convergence to equilibrium point are extremely likely to converge to this very poor strategy (which is what happens with DAC).



(a) Start state (b) (c) (d)

Fig. 1. In (c), the dashed line is the performance of the optimal policy, blue is DTPO and red is DAC.

For the DAC implementation we used $\beta_\omega = 10^{-3}$ and $\beta_\theta = 10^{-4}$. The hyper-parameters of the DTPO implementation are as follows. We set $\mu_\rho = 1$, $\mu_V = 1$, $\mu_\pi = 0.01$ and $\lambda(s) = 20$ for every state and subtracted 0.05 every time the state was visited until $\lambda(s) = 0.5$. Since there are two agents, all combination weights are given by $c_{nk} = 0.5$. The constants T and I in Algorithm 1 were set to 1200 and 1 respectively. In our experience selection strategy transitions were included in the replay buffer if the state-action was selected for the first time and the replay buffers of all the agents were big enough to hold all these transitions.

The results of DTPO and DAC are shown in Figures 1 (c) and (d). The former shows the performance of the greedy policy with respect to the learned one for each of the algorithms for 12 time-steps (enough to collect the food). Figure 1 (d) shows the mean square difference between (7) (for $\lambda(s) = 0.5$ for all states) and $\pi_1(a|s)$. Note that DTPO learns the optimal strategy while DAC converges to the suboptimal Nash equilibrium. A video showing the evolution of the policy during learning is available online[2].

5. REFERENCES

[1] M. L. Littman, "Value-function reinforcement learning in markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[2]Video available at EPFL's Adaptive Systems Lab website https://asl.epfl.ch/conferences/.

[2] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. International Conference on Machine Learning* (ICML), Palo Alto, USA, 2000, pp. 535–542.

[3] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," in *Proc. AAAI/IAAI*, Alberta, Canada, 2002, pp. 326–331.

[4] G. Tesauro, "Extending Q-learning to general adaptive multi-agent systems," in *Proc. Advances in Neural Information Processing Systems*, Vancouver, Canada, 2004, pp. 871–878.

[5] X. Wang and T. Sandholm, "Reinforcement learning to play an optimal nash equilibrium in team markov games," in *Proc. Advances in Neural Information Processing Systems*, Vancouver, Canada, 2003, pp. 1603–1610.

[6] L. Matignon, G. Laurent, and N. Le Fort-Piat, "Hysteretic Q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams." in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, USA, 2007, pp. 64–69.

[7] G. Arslan and S. Yüksel, "Decentralized Q-learning for stochastic teams and games," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1545–1558, 2017.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[9] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[10] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," in *Proc. Advances in Neural Information Processing Systems*, Long Beach, USA, 2017, pp. 2775–2785.

[11] L. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[12] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proc. Advances in Neural Information Processing Systems*, Long Beach, USA, 2017, pp. 5048–5058.

[13] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "Experience selection in deep reinforcement learning for control," *Journal of Machine Learning Research*, vol. 19, no. 9, pp. 1–56, 2018.

[14] B. Dai, A. Shaw, L. Li, L. Xiao, N. He, Z. Liu, J. Chen, and L. Song, "Sbeed: Convergent reinforcement learning with nonlinear function approximation," in *Proc International Conference on Machine Learning* (ICML), Stockholm, Sweden, 2018, pp. 1133–1142.

[15] K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact diffusion for distributed optimization and learning—part I: Algorithm development," *IEEE Transactions on Signal Processing*, vol. 67, no. 3, pp. 708–723, 2018.

[16] S. V. Macua, J. Chen, S. Zazo, and A. H. Sayed, "Distributed policy evaluation under multiple behavior strategies," *IEEE Transactions on Automatic Control*, vol. 60, no. 5, pp. 1260–1274, 2015.

[17] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Başar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. International Conference on Machine Learning*, Stockholm, Sweden, 2018, pp. 5867–5876.