Distributed Value-Function Learning with Linear Convergence Rates*

Lucas Cassano[1], Kun Yuan[1] and Ali H. Sayed[2]

***Abstract*— In this paper we develop a fully decentralized algorithm for policy evaluation with off-policy learning and linear function approximation. The proposed algorithm is of the variance reduced kind and achieves linear convergence with $O(1)$ memory requirements. We consider the case where a collection of agents have distinct and fixed size datasets gathered following different behavior policies (none of which is required to explore the full state space) and they all collaborate to evaluate a common target policy. The network approach allows all agents to converge to the optimal solution even in situations where neither agent can converge on its own without cooperation. We provide simulations to illustrate the effectiveness of the method in a Linear Quadratic Regulator (LQR) problem.**

I. INTRODUCTION

Traditionally, the reinforcement learning (RL) community has had a long standing interest in gradient algorithms, both for policy evaluation and control purposes. In the policy evaluation setup, the goal is to find the value function of a specific policy; while in the control case the goal is to find an optimal policy for a specific task usually modeled as a Markov Decision Process (MDP). Policy evaluation algorithms are important to study because they are often key components of policy optimization algorithms.

Recent years have seen a plethora of new gradient algorithms for policy evaluation like for example GTD [1], TDC [2], GTD2 [2] and True Online GTD(λ) [3]. All these algorithms have guaranteed stability (for small enough step-size) while combining off-policy learning and linear function approximation. When these algorithms are applied to problems with finite amounts of data, they have the drawback that they converge at a sub-linear rate because a decaying step-size is necessary to guarantee convergence to the minimizer. Leveraging recent developments in variance reduced algorithms (particularly SVRG [4] and SAGA [5]) the work [6] presented SVRG for Policy Evaluation and SAGA for Policy Evaluation. These algorithms can be seen as combinations of GTD2 with SVRG and SAGA, respectively, and they have the advantage over GTD2 (and the other gradient algorithms previously mentioned) that they have guaranteed linear convergence for the fixed data set case. Another interesting line of work is [7] and [8]; in these works the authors extend the TDC and GTD2 algorithms to the fully decentralized multi-agent case. These algorithms allow individual agents to converge at a sub-linear rate to the optimal solutions through collaboration even in situations where convergence is unfeasible without such collaboration.

The main contribution of this paper is to tackle the problem of Policy Evaluation in a situation where data is dispersed over a number of nodes and a fully distributed solution is preferable. To this end we present *Fast Diffusion for Policy Evaluation*, a fully decentralized multi-agent algorithm for policy evaluation with a guaranteed linear convergence rate to the global minimizer. The introduced algorithm combines off-policy learning, linear function approximation and has $O(1)$ memory requirements. In our distributed model a fusion center is not required and communication is only allowed between immediate neighbors. To the best of our knowledge, this is the first algorithm that combines all these features. The closest papers to the work we present here are [7] and [8], these papers consider the same distributed model that we do. Under this model each agent follows its own behavior policy and has access only to its own rewards and feature vector (which represents the state), which are independent of each other; collaboration is done only on a local basis (i.e., agents only share information with their immediate neighbors) and the agents have the goal of estimating the value function of a common target policy. This framework has several applications like for instance in what is known as collective robot learning (see for example [9], [10] and [11]). The algorithm we present is superior to the ones presented in [7] and [8] for the finite sample case because it converges with a linear rate to the minimizer, while the prior algorithms converge in a sub-linear fashion due to the necessity of a decaying step-size to guarantee convergence. This work was independently done from the recent work [12]. This paper also derives an algorithm for distributed policy evaluation but it has two main differences. In the first place, they only consider a scenario in which the state is global and known to all agents, and hence their algorithm is not suited for applications where agents have different states which are only known to themselves. Secondly, their algorithm is based on SAG [13] and therefore its memory requirements scale linearly with the amount of data $\big($i.e., $O(N)\big)$, while our algorithm's memory requirements are independent of the amount of data $\big($i.e., $O(1)\big)$.

Notation: Matrices are denoted by upper case letters, while vectors are denoted by lower case. Bold font and calligraphic font are used to denote random variables and sets, respectively. We denote the spectral radius of matrix A by $\rho(A)$. $\mathbb{E}_g$ is the expected value with respect to distribution

*This work was supported in part by NSF grants CCF-1524250 and ECCS-1407712.

[1]L. Cassano and K. Yuan are with Department of Electrical Engineering, University of California, Los Angeles, Los Angeles, CA 90095-1594, USA {cassanolucas,kunyuan}@ucla.edu

[2]Ali H. Sayed with the School of Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland CH-1015 ali.sayed@epfl.ch

g. $\|\cdot\|_D$ refers to the weighted matrix norm, where D is a diagonal positive definite matrix. And $\mathbb{R}$ denotes the set of real numbers.

II. PROBLEM SETTING

A. Markov Decision Processes and the Value Function

We consider the problem of policy evaluation within the traditional RL framework. As usual, we model this setting as a finite Markov Decision Process (MDP), with an MDP defined by the tuple ($\mathcal{S}$,$\mathcal{A}$,$\mathcal{P}$,r,γ), where $\mathcal{S}$ is a set of states of size $S = |\mathcal{S}|$, $\mathcal{A}$ is a set of actions of size $A = |\mathcal{A}|$, $\mathcal{P}(s'|s,a)$ specifies the probability of transitioning to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ having taken action $a \in \mathcal{A}$, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function (where $r(s,a,s') = \mathbb{E}\,\boldsymbol{r}(s,a,s')$ is the expected reward when the agent transitions to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ having taken action $a \in \mathcal{A}$) and $\gamma \in [0,1)$ is the discount factor.

Even though in this paper we analyze the multi-agent scenario, in this section we consider the single agent case for clarity of exposition. We consider an agent who wants to learn the value function, $v^\pi(s)$, for a target policy of interest $\pi(a|s)$ while following a potentially different behavior policy, $\phi(a|s)$. We recall that the notation $\pi(a|s)$ specifies the probability of selecting action a at state s. We also recall that the value function for a target policy π, starting from some initial state $s \in \mathcal{S}$ at time i, is defined as follows:

$$v^\pi(s) = \mathbb{E}\left[\sum_{t=i}^{\infty} \gamma^{t-i} \boldsymbol{r}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \Big| \boldsymbol{s}_i = s\right] \tag{1}$$

where $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ are the state and action at time t, respectively. Note that since we are dealing with a constant target policy π, the transition probabilities between states, which are given by $p^\pi_{s,s'} = \mathbb{E}_\pi \mathcal{P}(s'|s,\boldsymbol{a})$, are fixed and hence the MDP reduces to a Markov Rewards Process (MRP). In this case, the state evolution of the agent can be modeled as a Markov Chain with transition matrix P^π whose entries are given by $(P^\pi)_{ij} = p^\pi_{i,j}$.

Assumption 1: We assume that the Markov Chain induced by the behavior policy $\phi(a|s)$ is aperiodic and irreducible. Due to the Perron-Frobenius Theorem, this condition guarantees that the Markov Chain under $\phi(a|s)$ will have a steady state distribution in which every state has a strictly positive probability of visitation [14]. ■

Using the matrix P^π and defining:

$$v^\pi = [v^\pi(1), v^\pi(2), ..., v^\pi(|\mathcal{S}|)]^T \tag{2}$$

$$r^\pi(s) = \mathbb{E}[\boldsymbol{r}(s,\boldsymbol{a},\boldsymbol{s}')] \tag{3}$$

$$r^\pi = [r^\pi(1), r^\pi(2), ..., r^\pi(|\mathcal{S}|)]^T \tag{4}$$

we can rewrite (1) in matrix form as:

$$v^\pi = \sum_{n=0}^{\infty} (\gamma P^\pi)^n r^\pi = (I - \gamma P^\pi)^{-1} r^\pi \tag{5}$$

The value function also satisfies the Bellman equation:

$$v^\pi = r^\pi + \gamma P^\pi v^\pi \tag{6}$$

B. Definition of Cost Function

We are interested in applications where the state space is too large (or even infinite) and hence some form of function approximation is necessary to reduce the dimensionality of the parameters to be learned. As we anticipated in the introduction, in this work we use linear approximations. More formally, for every state $s \in \mathcal{S}$, we approximate $v^\pi(s) \approx x_s^T \theta^\star$ where $x_s \in \mathbb{R}^M$ is a feature vector corresponding to state s and $\theta^\star \in \mathbb{R}^M$ is a parameter vector, such that $M \ll S$. Defining $X = [x_1, x_2, \cdots, x_S]^T \in \mathbb{R}^{S \times M}$, we can write a vector approximation for v^π as $v^\pi \approx X\theta^\star$. We assume that X is a full rank matrix; this is not a restrictive assumption since the feature matrix is a design choice. It is important to note though that v^π need not be in the rangespace of X. If v^π is in the rangespace of X, an equality of the form $v^\pi = X\theta^\star$ holds exactly and the value of $\theta^\star$ is unique and given by $\theta^\star = (X^T X)^{-1} X^T v^\pi$. For the more general case where v^π is not in the rangespace of X, then $\theta^\star$ has to be defined through some cost function. One sensible choice for $\theta^\star$ is:

$$\theta^\star = \arg\min_\theta \|X\theta - v^\pi\|_D^2 = (X^T D X)^{-1} X^T D v^\pi \tag{7}$$

where D is some positive definite weighting matrix to be defined later. Note that $(X^T D X)^{-1}$ always exists and is positive definite because $X^T D X$ is positive definite (due to the fact that X is full rank and D is positive definite). Although (7) is a reasonable cost to define $\theta^\star$, it is not useful to derive a learning algorithm since v^π is not known beforehand and cannot be sampled. As a result, for the purposes of deriving a learning algorithm, another cost (whose gradient can be sampled) needs to be used as a surrogate for (7). We use the Mean Square Projected Bellman Error (MSPBE) (first introduced in [2]) with a regularization term:

$$S(\theta) = \frac{1}{2}\left\|\Pi\left[r^\pi + \gamma P^\pi X\theta\right] - X\theta\right\|_D^2 + \frac{\eta}{2}\|\theta - \theta_{\text{prior}}\|_U^2 \tag{8}$$

where $\Pi \in \mathbb{R}^{S \times S}$ is the weighted projection matrix onto the space spanned by X, i.e., $\Pi = X(X^T D X)^{-1} X^T D$, $\eta > 0$ is a regularization parameter, $U > 0$ is a symmetric positive-definite weighting matrix, and θ_{prior} reflects prior knowledge about θ. Two sensible choices for U are $U = I$ and $U = X^T D X$. The regularization term can be particularly useful when the policy evaluation algorithm is used as part of a policy iteration loop (since subsequent policies are expected to have similar value functions and the value of θ learned in one iteration can be used as θ_{prior} in the next iteration) like for example [15]. Equation (8) can be equivalently re-written in the following more compact form:

$$S(\theta) = \frac{1}{2}\|A\theta - b\|_{C^{-1}}^2 + \frac{\eta}{2}\|\theta - \theta_{\text{prior}}\|_U^2 \tag{9}$$

where

$$A \triangleq X^T D(I - \gamma P^\pi)X \tag{10a}$$

$$b \triangleq X^T D r^\pi \tag{10b}$$

$$C \triangleq X^T D X \tag{10c}$$

Remark 1: A is an invertible matrix.

Proof: Since $\gamma < 1$ and $\rho(P^\pi) = 1$ (because P^π is right stochastic) the spectral radius of γP^π is strictly smaller than one, and hence $I - \gamma P^\pi$ is invertible. Noting that by assumption X and D are full rank matrices concludes the proof. ∎

The minimizer of (9) is given by:

$$\theta^o = (A^T C^{-1} A + \eta U)^{-1}(\eta U \theta_{\text{prior}} + A^T C^{-1} b) \tag{11}$$

where the inverse $(A^T C^{-1} A + \eta U)^{-1}$ exists and hence θ^o is always well defined. This is because ηU and $A^T C^{-1} A$ are positive-definite matrices.

At this point, all that is left to fully define the surrogate cost function $S(\theta)$ is to choose the positive definite matrix D. The algorithm that we derive in this paper is of the stochastic gradient type. With this in mind, we shall choose D such that the quantities A, b and C turn out to be expectations that can be sampled from data realizations. Thus, we start by setting D to be a diagonal matrix with positive entries; we collect these entries into the vector d^ϕ and write D^ϕ instead of D, i.e. $D = D^\phi = \text{diag}(d^\phi)$. We shall select d^ϕ to correspond to the steady state distribution of the Markov chain induced by the behavior policy, $\phi(a|s)$. This choice for D not only is convenient in terms of algorithm derivation, it is also physically meaningful; since with this choice for D, the estimation error at states which are visited more often is weighted more heavily than the error at states which are rarely visited. As a consequence of Assumption 1 and the Perron-Frobenius Theorem [14], the vector d^ϕ is guaranteed to exist and all its entries will be strictly positive and add up to one. Moreover, this vector satisfies ${d^\phi}^T P^\phi = {d^\phi}^T$ where P^ϕ is the transition probability matrix defined in a manner similar to P^π. Using this choice for D, the matrices A, b and C can be written as expectations as follows:

$$\begin{aligned}
A &= X^T D(I - \gamma P^\pi)X = \sum_{s_t \in \mathcal{S}} d^\phi(s_t) x_{s_t}\big(x_{s_t} - \gamma p^\pi_{s_t, s_{t+1}} x_{s_{t+1}}\big)^T \\
&= \mathbb{E}_{d^\phi, \mathcal{P}, \pi}\left[\boldsymbol{x}_t \left(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1}\right)^T\right] \\
&= \mathbb{E}_{d^\phi, \mathcal{P}, \phi}\left[\frac{\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)} \boldsymbol{x}_t \left(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1}\right)^T\right] \qquad (12) \\
b &= X^T D r^\pi \\
&= \sum_{s_t \in \mathcal{S}} d^\phi(s_t) x_{s_t} \sum_{a \in \mathcal{A}} \sum_{s_{t+1} \in \mathcal{S}} \pi(a|s_t) \mathcal{P}(s_{t+1}|s_t, a) r(s_t, a, s_{t+1}) \\
&= \mathbb{E}_{d^\phi, \mathcal{P}, \pi}\left[\boldsymbol{x}_t \boldsymbol{r}_t\right] = \mathbb{E}_{d^\phi, \mathcal{P}, \phi}\left[\frac{\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)} \boldsymbol{x}_t \boldsymbol{r}_t\right] \qquad (13) \\
C &= X^T D X = \sum_{s_t \in \mathcal{S}} d^\phi(s_t) x_{s_t} x_{s_t} = \mathbb{E}_{d^\phi}\left[\boldsymbol{x}_t \boldsymbol{x}_t^T\right] \qquad (14)
\end{aligned}$$

where, with a little abuse of notation, we define $\boldsymbol{x}_t = \boldsymbol{x}_{\boldsymbol{s}_t}$ and $\boldsymbol{r}_t = \boldsymbol{r}^\pi(\boldsymbol{s}_t)$, where $\boldsymbol{s}_t$ is the state visited at time t. We derive expressions with expectations taken with respect to ϕ to derive an algorithm suitable for both on-policy and off-policy operation.

C. Optimization Problem

Since the signal distributions are not known beforehand and we are working with a finite amount of data, say, of size N, we need to rely on empirical approximations to estimate the expectations in $\{A, b, C\}$. We thus let $\widehat{A}$, $\widehat{b}$, $\widehat{C}$ and $\widehat{U}$ denote estimates for A, b, C and U from data and replace them in (9) to define the following empirical optimization problem:

$$\min_\theta J_{\text{emp}}(\theta) \triangleq \frac{1}{2}\left\|\widehat{A}\theta - \widehat{b}\right\|^2_{\widehat{C}^{-1}} + \frac{\eta}{2}\left\|\theta - \theta_{\text{prior}}\right\|^2_{\widehat{U}} \tag{15}$$

Note that whether an empirical estimate for U is required depends on the choice for U. For instance, if $U = I$ then obviously no estimate is needed. However, if $U = X^T D X = \mathbb{E}_{d^\phi}\left[\boldsymbol{x}_t \boldsymbol{x}_t^T\right] = C$ then an empirical estimate is needed (for this particular choice, $\widehat{U} = \widehat{C}$).

To fully characterize the empirical optimization problem, expressions for the empirical estimates still need to be provided. For the general off-policy case the following expressions provide unbiased estimates of the desired quantities:

$$\widehat{A} \triangleq \frac{1}{N-1} \sum_{n=1}^{N-1} \underbrace{\xi_n x_n \left(x_n - \gamma x_{n+1}\right)^T}_{\triangleq \widehat{A}_n} \tag{16a}$$

$$\widehat{b} \triangleq \frac{1}{N-1} \sum_{n=1}^{N-1} \underbrace{\xi_n x_n r_n}_{\triangleq \widehat{b}_n} \tag{16b}$$

$$\widehat{C} \triangleq \frac{1}{N-1} \sum_{n=1}^{N-1} \underbrace{x_n x_n^T}_{\triangleq \widehat{C}_n} \tag{16c}$$

where we defined the importance sampling weights:

$$\xi_n \triangleq \pi(a_n|s_n)/\phi(a_n|s_n) \tag{17}$$

III. FAST DIFFUSION FOR POLICY EVALUATION

In this section we present the distributed framework we consider and later we derive *Fast Diffusion for Policy Evaluation*. The purpose of this algorithm is to deal with situations where data is dispersed among a number of nodes and the goal is to solve the policy evaluation problem in a fully decentralized manner. The algorithm combines two important tools for inference from data: diffusion strategies [16], [14] and amortized variance-reduced techniques [17].

A. Distributed Setting

We consider a situation in which there are K agents who want to evaluate a common target policy $\pi(a|s)$. Each agent has N samples which were collected following its own behavior policy ϕ_k (with steady state distribution matrix D^{ϕ_k}). Note that all behavior policies are assumed to be potentially different from each other. The goal for all agents is to estimate the value function of the target policy $\pi(a|s)$ leveraging all the data from all other agents in a fully decentralized manner.

To do this, they form a network in which each agent can only communicate with other agents in its immediate neighborhood. The network is represented by an undirected graph in which the nodes and edges represent the agents and communication links respectively. The topology of the graph is defined by a combination matrix L whose kn-th entry (i.e. l_{kn}) is a scalar with which agent n weights information incoming from agent k.

Assumption 2: 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. that at least one agent uses its own information). We further assume that the combination matrix L is symmetric and doubly-stochastic. ■

A combination matrix satisfying assumption 2 can be generated using the Laplacian rule, Maximum-degree rule, Metropolis rule or others (see Table 14.1 in [14]). Assumption 2 is necessary for the proof of convergence.

B. Algorithm Derivation

Mathematically the goal for all agents is to minimize the following aggregate cost:

$$S_{\text{multi}}(\theta)=\sum_{k=1}^{K}\tau_k\left[\frac{1}{2}\left\|\Pi\big(r^{\pi}+\gamma P^{\pi}X\theta\big)-X\theta\right\|_{D^{\phi_k}}^2+\frac{\eta}{2}\left\|\theta-\theta_{\text{prior}}\right\|_{U_k}^2\right] \tag{18}$$

The purpose of the nonnegative coefficients τ_k is to scale the costs of the different agents; this is useful since the costs of agents whose behavior policy is closer to the target policy might be assigned higher weights than those whose behavior policies deviate a lot from the target policy (and hence are subject to a higher variance). Defining:

$$D^M \triangleq \sum_{k=1}^{K}\tau_k D^{\phi_k} \qquad U^M \triangleq \sum_{k=1}^{K}\tau_k U_k \tag{19}$$

equation (18) becomes:

$$S_{\text{multi}}(\theta)=\frac{1}{2}\left\|\Pi\big(r^{\pi}+\gamma P^{\pi}X\theta\big)-X\theta\right\|_{D^M}^2+\frac{\eta}{2}\left\|\theta-\theta_{\text{prior}}\right\|_{U^M}^2 \tag{20}$$

Note that (20) has the same form as (9), the only difference is that in (20) the matrices D^M and U^M are defined by linear combinations of the individual matrices D^{ϕ_k} and U_k respectively. Note that the matrices D^{ϕ_k} are not required to be positive definite, only D^M is required to be a positive definite diagonal matrix. Since the matrices D^{ϕ_k} are given by the steady state probabilities of the behavior policies, this implies that each agent does not need to explore the entire state space by itself, but rather all the agents collectively need to explore the state space, this is one of the advantages of our multi-agent setting. In practice this could be useful since the agents can divide the entire state space in sections each of which can be explored by a different agent in parallel.

The empirical problem for the multi-agent case is given by:

$$\min_{\theta} J_{\text{emp}}(\theta) = \min_{\theta}\frac{1}{2}\left\|\widehat{A}\theta-\widehat{b}\right\|_{\widehat{C}^{-1}}^2+\frac{\eta}{2}\left\|\theta-\theta_{\text{prior}}\right\|_{\widehat{U}}^2 \tag{21}$$

where

$$\widehat{A}=\sum_{k=1}^{K}\tau_k\frac{1}{N-1}\sum_{n=1}^{N-1}\widehat{A}_{k,n} \tag{22a}$$

$$\widehat{b}=\sum_{k=1}^{K}\tau_k\frac{1}{N-1}\sum_{n=1}^{N-1}\widehat{b}_{k,n} \tag{22b}$$

$$\widehat{C}=\sum_{k=1}^{K}\tau_k\frac{1}{N-1}\sum_{n=1}^{N-1}\widehat{C}_{k,n} \tag{22c}$$

To solve problem (21) we shall judiciously combine the Exact Diffusion [18] and Amortized Variance Reduced Gradient (AVRG) [17] techniques. We do so in a similar manner as done in the Diffusion AVRG approach [19]. Exact Diffusion is a fully distributed algorithm that guarantees convergence to the global minimizer, while AVRG is a reduced-variance stochastic gradient algorithm that relies on random reshuffling. With this in mind we first note that Exact Diffusion is designed to solve problems of the following form:

$$\min_{\theta}\sum_{k=1}^{K}J_k(\theta) \tag{23}$$

where the summation runs across the K agents and $J_k(\theta)$ is the individual risk function of agent k. AVRG on the other hand tackles problems of the following form:

$$\min_{\theta}\sum_{n=1}^{N}Q_n(\theta) \tag{24}$$

where the summation runs across N samples and $Q_n(\theta)$ is some loss function evaluated at θ and the n-th data point. Hence to be able to apply both techniques we need our cost to have the following form:

$$\min_{\theta}\sum_{k=1}^{K}q_k\sum_{n=1}^{N}Q_{k,n}(\theta) \tag{25}$$

where $Q_{k,n}(\theta)$ is the loss function of the k-th agent evaluated for the n-th data sample and vector θ. However note that (21) does not have the form of (25). To circumvent this issue, we follow a similar approach as in [7] (see also [20] and [6]), which is to formulate (21) as an equivalent saddle-point problem. To this end, we first note that every quadratic function can be expressed in terms of its conjugate function as $\frac{1}{2}\|A\theta-b\|_{C^{-1}}^2=\max_{\omega}\left((A\theta-b)^T\omega-\frac{1}{2}\|\omega\|_C^2\right)$ (see [21]). Therefore, expression (21) can be rewritten as:

$$\min_{\theta}\max_{\omega}\left(\sum_{k=1}^{K}q_k\sum_{j=1}^{J}\frac{J}{N-H}\sum_{n\in\mathcal{J}_{kj}}F_{k,n}(\theta,\omega)\right) \tag{26}$$

$$F_{k,n}=\sum_{n\in\mathcal{J}_{kj}}\frac{\eta}{2}\|\theta-\theta_{\text{prior}}\|_{\widehat{U}_{k,n}}^2-\omega^T\big(\widehat{A}_{k,n}\theta-\widehat{b}_{k,n}\big)-\frac{1}{2}\|\omega\|_{\widehat{C}_{k,n}}^2 \tag{27}$$

where we defined $q_k \triangleq \tau_k/J$ and arranged the data in J mini-batches (where $\mathcal{J}_{kj}$ is the j-th mini-batch of the k-agent). Note that problem (26) has a similar form to (25) in the sense that it is an empirical average of loss functions. However there's a key difference, which is that (25) is a minimization problem while (26) is a saddle point problem. Hence, to be able to apply Exact Diffusion to (26) we modify the original formulation to make it suitable for our saddle-point problem. The modification we make is to change the gradient vector in the original formulation for another vector (which we refer to as β) in which the gradient with respect to the minimization variable is stacked with the negative gradient with respect to the maximization variable. We further apply the AVRG variance reduction scheme to the gradients of the primal and dual variables. We refer to this algorithm as *Fast Diffusion for Policy Evaluation* (see Algorithm 1). Note that this is not an application of

Algorithm 1: ***Fast Diffusion for Policy Evaluation*** **at node** k

Distribute the $N-H$ data points into J mini-batches of size $|\mathcal{J}_j|$; where $\mathcal{J}_j$ is the j-th mini-batch.

Initialize: $\theta_{k,0}^0$ and $\omega_{k,0}^0$ arbitrarily; let $\psi_{k,0}^0 = [\theta_{k,0}^{0\ T}, \omega_{k,0}^{0\ T}]^T$, $g_k^0 = 0$; $\beta_{k,n}(\theta_{k,0}^0, \omega_{k,0}^0) = 0$, $1 \le n \le N-H$

For $e = 0, 1, 2 \ldots$:

Generate a random permutation function of the mini-batches $\boldsymbol{\sigma}_k^e$

Set $\boldsymbol{g}_k^{e+1} = 0$

For $i = 0, 1, \ldots, J-1$:

Generate the local stochastic gradients:

$$j = \boldsymbol{\sigma}_k^e(i) \tag{28}$$

$$\beta_k(\theta_{k,i}^e, \omega_{k,i}^e) = g_k^e + \frac{1}{|\mathcal{J}_j|}\sum_{l\in\mathcal{J}_j}\big(\beta_{k,l}(\theta_{k,i}^e, \omega_{k,i}^e) - \beta_{k,l}(\theta_{k,0}^e, \omega_{k,0}^e)\big) \tag{29}$$

$$g_k^{e+1} = g_k^{e+1} + \frac{1}{N-H}\sum_{l\in\mathcal{J}_j}\beta_{k,l}(\theta_{k,i}^e, \omega_{k,i}^e) \tag{30}$$

Update $[\theta_{k,i+1}^e, \omega_{k,i+1}^e]^T$ with exact diffusion:

$$\psi_{k,i+1}^e = \begin{bmatrix} \theta_{k,i}^e \\ \omega_{k,i}^e \end{bmatrix} - q_k \begin{bmatrix} \mu_\theta & 0 \\ 0 & \mu_\omega \end{bmatrix} \beta_k(\theta_{k,i}^e, \omega_{k,i}^e) \tag{31}$$

$$\phi_{k,i+1}^e = \psi_{k,i+1}^e + \begin{bmatrix} \theta_{k,i}^e \\ \omega_{k,i}^e \end{bmatrix} - \psi_{k,i}^e \tag{32}$$

$$\begin{bmatrix} \theta_{k,i+1}^e \\ \omega_{k,i+1}^e \end{bmatrix} = \left(\phi_{k,i+1}^e + \sum_{n\in\mathcal{N}_k} l_{nk}\phi_{n,i+1}^e\right)/2 \tag{33}$$

$$\begin{bmatrix} \theta_{k,0}^{e+1} \\ \omega_{k,0}^{e+1} \end{bmatrix} = \begin{bmatrix} \theta_{k,J}^e \\ \omega_{k,J}^e \end{bmatrix} \tag{34}$$

Diffusion AVRG to (26), our algorithm finds the saddle point in a primal-dual formulation while Diffusion AVRG finds the minimizer of a strongly convex function. It cannot be assumed that the convergence properties of Diffusion AVRG will carry over to our saddle-point formulation, hence a new convergence theorem and proof are required.

In the above listing, we introduced $\boldsymbol{\sigma}_k^e$, $\mathcal{J}_j$ and $\beta_{k,j}(\theta,\omega)$, where $\boldsymbol{\sigma}_k^e$ indicates a random permutation of the J mini-batches of the k-th agent which is generated at the beginning of epoch e; $\mathcal{J}_j$ is the j-th mini-batch and $\beta_{k,l}(\theta,\omega)$ is defined as follows:

$$\beta_{k,l}(\theta,\omega) = \begin{bmatrix} \nabla_\theta F_{k,l}(\theta,\omega) \\ -\nabla_\omega F_{k,l}(\theta,\omega) \end{bmatrix} = \begin{bmatrix} \eta\widehat{U}_{k,l}(\theta-\theta_{\text{prior}}) - \widehat{A}_{k,l}^T\omega \\ \widehat{A}_{k,l}\theta - \widehat{b}_{k,l} + \widehat{C}_{k,l}\omega \end{bmatrix} \tag{35}$$

Note that the choice of the mini-batch size provides a communication-computation trade-off. As the number of mini-batches diminishes so do the communication requirements per epoch, however less updates are performed per epoch which might result in the need of more epochs to achieve a desired tolerance. Obviously the optimal amount of mini-batches J to minimize the overall time of the optimization process depends on the particular hardware availability of each implementation.

Remark 2: The saddle-point of (26) is given by

$$\begin{bmatrix} \theta_{\text{emp}}^o \\ \omega_{\text{emp}}^o \end{bmatrix} = \begin{bmatrix} \left(\widehat{A}^T\widehat{C}^{-1}\widehat{A} + \eta\widehat{U}\right)^{-1}\left(\eta\widehat{U}\theta_{\text{prior}} + \widehat{A}^T\widehat{C}^{-1}\widehat{b}\right) \\ \widehat{C}^{-1}\widehat{b} - \widehat{C}^{-1}\widehat{A}\theta_{\text{emp}}^o \end{bmatrix} \tag{36}$$

Proof: θ_{emp}^o and ω_{emp}^o are obtained by equating the gradient of (26) to zero and solving for θ and ω. ∎

Theorem 1: If Assumption 2 is satisfied and for $\eta > 0$ and small enough step-sizes μ_ω and μ_θ, the iterates $\boldsymbol{\theta}_{k,0}^e$ and $\boldsymbol{\omega}_{k,0}^e$ generated by *Fast Diffusion for Policy Evaluation* converge linearly to (36).

Proof: The argument is demanding and lengthy and involves several steps, we therefore omit the proof due to length restrictions and refer to the extended arXiv version [22]. ∎

IV. SIMULATION

In this section we illustrate a case application of *Fast Diffusion for Policy Evaluation* to a distributed Linear Quadratic Regulator (LQR) problem. The main point of this simulation is to validate our theoretical results and show the linear convergence rate of our algorithm versus existing approaches.

A. Linear Quadratic Regulators

We first briefly review the connection between LQR and reinforcement learning (RL). In a standard LQR problem, the state evolution of the system is dictated by a state equation:

$$s_{t+1} = Es_t + Ba_t \tag{37}$$

where s_t and a_t denote the state and input of the system at time t, respectively, and E and B are the state and input matrices. Matrices E and B are analogous to $\mathcal{P}$ in our RL formulation. The goal of the controller (which is what in the RL context is referred as the policy) is to minimize the following infinite horizon discounted quadratic cost:

$$J_1 = \sum_{t=0}^{\infty}\gamma^t\left(s_t^TQs_t + a_t^TRa_t\right) \tag{38}$$

or equivalently to maximize:

$$J_2 = \sum_{t=0}^{\infty}\gamma^t\underbrace{\left(-\left(s_t^TQs_t + a_t^TRa_t\right)\right)}_{\triangleq\, r_t(s_t,a_t)} = \sum_{t=0}^{\infty}\gamma^t r_t(s_t,a_t) \tag{39}$$

where Q and R are some positive semidefinite and positive definite matrices, respectively. Note that $-\left(s_t^TQs_t + a_t^TRa_t\right)$ is the reward at time t in our setting. It is well known that the feedback control law (optimal policy) that maximizes J_2 is linear in the state parameters:

$$a_t = Ps_t \tag{40}$$

where P is some matrix. The value function of some policy (controller) is defined as:

$$\begin{aligned} v(s_j) &= -\sum_{t=j}^{\infty}\gamma^{t-j}\left(s_t^TQs_t + a_t^TRa_t\right) = \sum_{t=j}^{\infty}\gamma^{t-j}r_t(s_t,a_t) \\ &= r_t(s_t,a_t) + \gamma v(s_{j+1}) \end{aligned} \tag{41}$$

Combining (37), (40) and (41) it can be shown that the value function can be written as a quadratic function of the state vector:

$$v(s_j) = s_j^TWs_j = \underbrace{(s_j\otimes s_j)^T}_{\triangleq\, x_t^T}\underbrace{\text{vec}(W)}_{\triangleq\,\theta} = x_t^T\theta \tag{42}$$

where we defined x_t (what we referred to in the previous sections as the feature vector) and θ in accordance with standard RL notation. Note that (42) justifies theoretically the use of linear function approximation for the value function. We further note that the value function matrix W is

some negative[1] semi-definite matrix and $\otimes$ is the Kronecker product. The value function matrix W corresponding to some policy ($a_t = Ps_t$) can be found by solving the following Lyapunov equation:

$$W = -Q - P^T R P + \gamma(E + BP)^T W (E + BP) \tag{43}$$

And the optimal policy $P^\star$ is given by:

$$P^\star = -\gamma(\gamma B^T W^\star B - R)^{-1} B^T W^\star E \tag{44}$$

where $W^\star$ is the corresponding value function matrix of $P^\star$ and is the solution to the following discrete time algebraic Riccati equation:

$$\begin{aligned} W^\star = &-(2\gamma - \gamma^2) E^T W^\star B (B^T W^\star B - R)^{-1} B^T W^\star E \\ &- Q + \gamma E^T W^\star E \end{aligned} \tag{45}$$

Naturally if the matrices E and B are known, the optimal controller can easily be found using (44) and (45) and hence RL techniques are not necessary. However, RL techniques are useful for cases in which those matrices are not known and a solution in which the optimal controller is learned from samples is preferable. Furthermore, RL techniques endow the controller with tracking abilities which are useful in time varying environments.

B. Case Application

We consider a swarm of 100 Autonomous Underwater Vehicles (AUV's) running a digital control system. We assume that the state evolution matrices E and B are unknown and furthermore an adaptive solution is desirable to cope with a time varying environment (for instance, changing currents). Policy gradient algorithms constitute one possible solution for this problem. Policy gradient algorithms have the following general structure:

Basic Policy gradient loop

For e=0,1,... :

1- Collect batch of data following the current policy.
2- Estimate the value function (θ) of the current policy using some policy evaluation algorithm.
3- Use the value function to improve the policy.

Note that a distributed policy evaluation algorithm is necessary in order to implement the distributed policy gradient algorithm.

Each AUV can decide to run its own policy gradient algorithm and learn on its own without cooperation. However, cooperation is preferable for two main reasons. In the first place, different AUVs might be subject to slightly different dynamics (caused by differences in the electronics due to tolerances and also environmental differences due to the geographical distance between the agents, like stronger or weaker currents) hence, through cooperation, the agents can converge to more robust controllers than without cooperation. Secondly, through collaboration agents will converge faster than without cooperation due to the parallelism in data collection (step 1 in the basic policy gradient loop). Therefore, in this case, we assume that all agents in the network want to collaborate and do so by running a distributed policy gradient algorithm (for example [15]). In this simulation we will not run the entire policy gradient algorithm and only run step 3 to showcase the performance of our algorithm. We therefore assume the AUVs have a controller (policy) for which they want to estimate the value function (learn θ).

To model the dynamics of the AUVs we use a linearized and discretized version of one of the axis of motion of the model presented in [23]:

$$\begin{aligned} s_{t+1} = \begin{bmatrix} y_{t+1} \\ z_{t+1} \end{bmatrix} &= \begin{bmatrix} 1 & T \\ 0 & 1 + TN_r(I_z - N_{\dot{r}})^{-1} \end{bmatrix} \begin{bmatrix} y_t \\ z_t \end{bmatrix} \\ &+ \begin{bmatrix} 0 & 0 \\ 0 & T(I_z - N_{\dot{r}})^{-1} \end{bmatrix} \begin{bmatrix} 0 \\ a_t \end{bmatrix} \end{aligned} \tag{46}$$

where $T = 0.1$s, $N_r = -0.23\text{Kgms}^{-1}$, $I_z = 1.41\text{Kgm}^2$, $N_{\dot{r}} = -0.56\text{Kgm}^2$, s, y, z and a_t are the sampling period, linear drag coefficient, moment of inertia, added mass, state vector of the AUV, heading, angular speed and input torque respectively. The Q and R matrices are given by:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{47}$$

The goal of the controllers is to minimize $\sum_{t=0}^{\infty} \gamma^t \left(-(y_d - y_t)^2 - 0.01 a_t^2\right)$. The reward and feature vector at time t are given by:

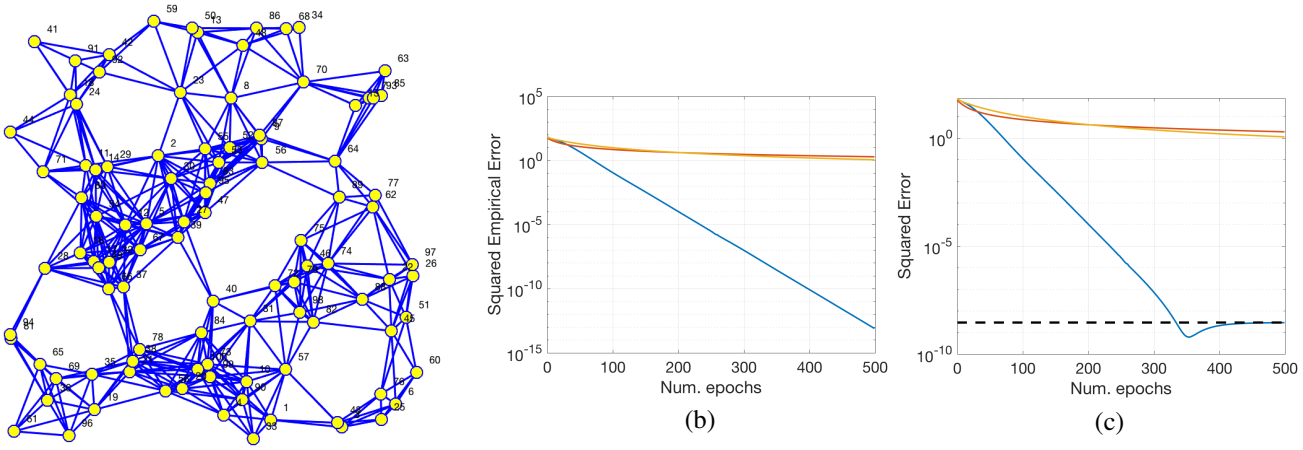
$$r_t(y, a) = -(y_d - y_t)^2 - 0.01 a_t^2 \tag{48}$$

$$x_t = \left[(y_d - y_t)^2, z_t^2, y_t z_t\right] \tag{49}$$

The controller implemented in each AUV is linear and given by $a_t = p(y_d - y_t + n_t^y) + d(z_t + n_t^z)$; where n_t^y and n_t^z are i.i.d Gaussian$\sim \mathcal{G}(\mu = 0, \sigma = 0.05)$ noise terms that account for normal observation noise, and p and d are the gains of the error and derivative of the error, respectively which define the controller. The p and d constants were sampled as $p \sim \mathcal{G}(\mu = p^\star, \sigma = 0.1)$ and $d \sim \mathcal{G}(\mu = d^\star, \sigma = 0.1)$ where $p = p^\star$ and $d = d^\star$ are the values of the optimal controller (obtained using (44)). Each AUV's initial heading is initialized randomly following a uniform distribution between -180^o and 180^o and the angular speed is initialized at zero. After initialization each AUV runs the control system for 6 seconds (i.e., $N = 60$), which is one second after the heading error reaches steady state and stores data samples. The network topology (Figure 1a) was generated randomly. To generate the network we distributed the agents uniformly in a unit square and then created connections betweens agents whose distance is closer than 0.2, this process was repeated until a connected network was created. The connection weights were determined using the Metropolis rule. The discount parameter was set to $\gamma = 0.99$.

Three algorithms were implemented to solve this problem, *Fast Diffusion for Policy Evaluation*, *Diffusion GTD2* [7] and *ALG2* [8] (the latter two with decaying step-sizes to guarantee convergence for a fair comparison). The parameters of the *Fast Diffusion for Policy Evaluation* implementation are: $\eta = 1 \times 10^{-4}$, $|\mathcal{J}| = 2$, $\mu_\theta = 3 \times 10^{-7}$, $\mu_\omega = 4 \times 10^{-7}$, $q_k = 1/K$ for all k, $U = I$, and θ_{prior} is the all zeros vector.

[1] W is negative semi-definite as opposed to positive semi-definite, as it is usually the case in optimal control, due to the fact that we are maximizing negative rewards as opposed to minimizing positive costs.



(a) (b) (c)

Fig. 1: The blue, yellow and red curves denote *Fast Diffusion for Policy Evaluation*, *Diffusion GTD2* and *ALG2* respectively. The dotted line in (c) depicts $\|\theta^o_{\text{emp}} - \theta^o\|^2$.

The parameters of the *Diffusion GTD2* implementation are: $\mu_\theta = 3.75 \times 10^{-7}(1 + 0.06e)^{-1}$, $\mu_\omega = 5 \times 10^{-7}(1 + 0.06e)^{-1}$, where e is the epoch number. And finally the parameters of the *ALG2* implementation are: $\mu_\theta = 5 \times 10^{-7}(1 + 0.06e)^{-1}$, $\mu_\omega = 5 \times 10^{-7}(1 + 0.06e)^{-1}$.

Figures 1b and 1c show the squared error ($\|\theta^e_0 - \theta^o\|^2$) and the Squared Empirical Error ($\|\theta^e_0 - \theta^o_{\text{emp}}\|^2$), respectively. Each curve depicts the error averaged over all agents. As predicted, *Fast Diffusion for Policy Evaluation* shows great improvement in terms of convergence speed over the other two algorithms due to the fact that it converges linearly.

REFERENCES

[1] R. S. Sutton, H. R. Maei, and C. Szepesvári, "A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation," in *Proc. Advances in Neural Information Processing Systems*, Vancouver, Canada, 2009, pp. 1609–1616.

[2] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proc. International Conference on Machine Learning*, Montreal, Canada, 2009, pp. 993–1000.

[3] H. van Hasselt, A. R. Mahmood, and R. S. Sutton, "Off-policy TD(λ) with a true online equivalence," in *Proc. Conference on Uncertainty in Artificial Intelligence*, Quebec City, Canada, 2014, pp. 330–339.

[4] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. Advances in neural information processing systems*, Lake Tahoe, USA, 2013, pp. 315–323.

[5] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. Advances in Neural Information Processing Systems*, Montreal, Canada, 2014, pp. 1646–1654.

[6] S. S. Du, J. Chen, L. Li, L. Xiao, and D. Zhou, "Stochastic variance reduction methods for policy evaluation," in *Proc. International Conference on Machine Learning*, Sydney, Australia, 2017, pp. 1049–1058.

[7] 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.

[8] M. S. Stanković and S. S. Stanković, "Multi-agent temporal-difference learning with linear function approximation: Weak convergence under time-varying network topologies," in *Proc. American Control Conference*, Boston, USA, 2016, pp. 167–172.

[9] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *Proc. International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 4263–4270.

[10] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[11] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE International Conference on Robotics and Automation*, Singapore, May 2017, pp. 3389–3396.

[12] H.-T. Wai, Z. Yang, P. Z. Wang, and M. Hong, "Multi-agent reinforcement learning via double averaging primal-dual optimization," *arXiv:1806.00877*, August 2018.

[13] N. L. Roux, M. Schmidt, and F. R. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," in in *Proc. Advances in Neural Information Processing Systems*, Lake Tahoe, Nevada, 2012, pp. 2663–2671.

[14] A. H. Sayed, "Adaptation, learning, and optimization over networks," *Foundations and Trends in Machine Learning*, vol. 7, 2014.

[15] S. V. Macua, A. Tukiainen, D. G.-O. Hernández, D. Baldazo, E. M. de Cote, and S. Zazo, "Diff-DAC: Distributed actor-critic for multitask deep reinforcement learning," *arXiv:1710.10363*, April 2018.

[16] A. H. Sayed, "Adaptive networks," *Proceedings of the IEEE*, vol. 102, no. 4, pp. 460–497, 2014.

[17] B. Ying, K. Yuan, and A. H. Sayed, "Convergence of variance-reduced learning under random reshuffling," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, Alberta, Canada, 2018, pp. 2286–2290.

[18] 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.

[19] K. Yuan, B. Ying, J. Liu, and A. H. Sayed, "Variance-reduced stochastic learning by networked agents under random reshuffling," *IEEE Transactions on Signal Processing*, vol. 67, no. 2, pp. 351–366, 2017.

[20] B. Liu, J. Liu, M. Ghavamzadeh, S. Mahadevan, and M. Petrik, "Finite-sample analysis of proximal gradient td algorithms." in *Proc. Conference on Uncertainty in Artificial Intelligence*, Amsterdam, Holland, 2015, pp. 504–513.

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

[22] L. Cassano, K. Yuan, and A. Sayed, "Multi-agent fully decentralized off-policy learning with linear convergence rates," *arXiv:1810.07792*, October 2018.

[23] C. D. Makavita, H. Nguyen, S. G. Jayasinghe, and D. Ranmuthugala, "Predictor-based model reference adaptive control of an unmanned underwater vehicle," in *Proc. International Conference on Control, Automation, Robotics and Vision*, Phuket, Thailand, 2016, pp. 1–7.