Multi-agent Policy Gradient Algorithms for Cyber-physical Systems with Lossy Communication

Adrian Redder[1][a], Arunselvan Ramaswamy[1][b] and Holger Karl[2][c]
[1]*Department of Computer Science, Paderborn University, Germany*
[2]*Hasso-Plattner-Institute, Potsdam University, Germany*

Keywords: Policy Gradient Algorithms, Multi-agent Learning, Communication Networks, Distributed Optimisation, Age of Information, Continuous Control.

Abstract: Distributed online learning over delaying communication networks is a fundamental problem in multi-agent learning, since the convergence behaviour of interacting agents is distorted by their delayed communication. It is a priori unclear, how much communication delay can be allowed, such that the joint policies of multiple agents can still converge to a solution of a multi-agent learning problem. In this work, we present the decentralization of the well known deep deterministic policy gradient algorithm using a communication network. We illustrate the convergence of the algorithm and the effect of lossy communication on the rate of convergence for a two-agent flow control problem, where the agents exchange their local information over a delaying wireless network. Finally, we discuss theoretical implications for this algorithm using recent advances in the theory of age of information and deep reinforcement learning.

1 INTRODUCTION

Centralized learning algorithms can usually be extended in a natural way to decentralized settings by replacing global information with information that has been communicated over a network. However, the communication network typically induces information delay, such that a centralized algorithm effectively uses data that has a certain *age of information* (AoI). In (Redder et al., 2021) practically verifiable sufficient conditions were presented, such that the AoI associated with data communicated over graph based networks behaves so that convergence of a distributed stochastic gradient descend scheme can be guaranteed.

In this paper, we go one step further and consider a multi-agent learning setting, where distributed agents seek to find local policies to solve a global learning problem. Specifically, we consider a multi-agent learning algorithm, where agents update their local policies conditioned on the policies of other agents with a certain AoI, i.e. policies that the other agents have used in the past.

Reinforcement learning (RL) problems pose additional challenges compared to stochastic optimisation problems. In RL one typically optimizes a policy in an online manner, where the interaction of the policy with an environment generates the observations from the environment. In stochastic optimisation on the other hand, one usually obtains observations of an environment that are independent from the optimisation variables of the problem. However, it was recently shown that the convergence of many RL algorithms can be analysed analogously to stochastic, iterative gradient descend algorithms (Ramaswamy et al., 2020). A natural question that therefore arises is, whether one can decentralize, with the help of communication, traditionally central RL algorithms to a multi-agent setting without losing asymptotic convergence guarantees.

In this paper, we therefore consider the decentralization of the deep deterministic policy gradient (DDPG) algorithm (Lillicrap et al., 2016) for the multi-agent learning setting. We achieve this, by allowing that the agents can frequently exchange their current local policy and their local state-action pairs over a communication network. At each local agent, training is then performed by conditioning a policy gradient step on the most recent available policy of the other agents. For the communication network, we

[a] https://orcid.org/0000-0001-7391-4688
[b] https://orcid.org/0000-0002-8343-6322
[c] https://orcid.org/0000-0001-7547-8111

Redder, A., Ramaswamy, A. and Karl, H.
Multi-agent Policy Gradient Algorithms for Cyber-physical Systems with Lossy Communication.
DOI: 10.5220/0010845400003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 1*, pages 282-289
ISBN: 978-989-758-547-0; ISSN: 2184-433X
Copyright © 2022 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

present a simple illustrative model and present associated conditions. We then show theoretically for this model, that the effect of lossy communication on the multi-agent learning algorithm will vanish asymptotically. Therefore, asymptotic guarantees for the algorithm with and without communication will be identical. Moreover, we also show that infinitely many global state-action pairs, will reach each agent.

We apply our algorithm to a simple but illustrative two-agent water filter flow control problem, where two agents have to control the in- and outflow to water filter. The agents have no information about the dynamics of the flow problem and have no prior information of the strategy of the other agents. But, the agents are allowed to communicate over a network, which therefore enables the use of our algorithm. Our simulations show how the communication network influences the rate of convergence of our multi-agent algorithm.

2 BACKGROUND ON RL IN CONTINUOUS ACTION SPACES

In this section we recall preliminaries on RL in continuous action spaces. In RL an agent interacts with an environment $\mathcal{E}$ that is modeled as a Markov decision process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, Markov transition kernel P, scalar reward function $r(s,a)$. It does so by taking actions via a policy $\mu : \mathcal{S} \to \mathcal{A}$. The most common objective in RL is to maximize the discounted infinite horizon return $R^n = \sum_{t=n}^{\infty} r(s^t, a^t)$ with discount factor $\alpha \in [0,1)$. The associated action-value function is defined as $Q(s,a) = \mathbb{E}_{\mu,\mathcal{E}}[R_1 \mid s_1 = s, a_1 = 1]$. The optimal action-value function is characterized by the solution of the Bellman equation

$$Q^*(s,a) = \mathbb{E}_{\mathcal{E}}\left[r(s,a) + \alpha \max_{a' \in \mathcal{A}} Q^*(s',a') \;\middle|\; s,a \right] \quad (1)$$

Q-learning with function approximation seeks to approximate Q^* by a parametrized function $Q(s,a;\theta)$ with parameters θ. For an observed tuple (s,a,r,s'), this can be done by performing a gradient descent step to minimize the squared Bellman loss

$$\Big(Q(s,a,;\theta) - (r(s,a) + \alpha \max_{a' \in \mathcal{A}} Q(s',a';\theta)) \Big)^2. \quad (2)$$

When the action space $\mathcal{A}$ is discrete, $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a;\theta)$, $\forall s \in \mathcal{S}$ defines a greedy policy with respect to $Q(s,a,;\theta)$. When $\mathcal{A}$ is continuous this is not possible, but instead we can use $Q(s,a;\theta)$ to improve a parametrized policy $\mu(s;\phi)$ by taking gradient steps in the direction of policy improvement with respect to the Q-function:

$$\nabla_\phi \mu(s;\phi) \nabla_a Q(s,a;\theta)|_{a=\mu(s;\phi)} \quad (3)$$

In (Silver et al., 2014) it was shown that the expectation of (3) w.r.t the discounted state distribution induced by a behaviour policy[1] is indeed the gradient of the expected return of the policy μ. In practice, however, we usually do not sample from this discounted state distribution (Nota and Thomas, 2020), but still use (3) with samples from the history of interaction. When the parametrized functions in (2) and (3) are deep neural networks the corresponding algorithm is then known as the deep deterministic policy gradient algorithm (DDPG) (Lillicrap et al., 2016).

3 DECENTRALIZED POLICY GRADIENT ALGORITHM

This section describes the adaptation of the DDPG algorithm to the decentralized multi-agent setting.

3.1 Two-agent DDPG with Delayed Communication

To simplify the presentation, we consider only two agents 1 and 2. Consider an MDP as defined in the previous section and assume that agent 1 and 2 have local action spaces $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is the global action space. Further, both agents can observe local state trajectories $s_1^n \in \mathcal{S}_1$ and $s_2^n \in \mathcal{S}_2$, such that $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$ is the global state space. Finally, we consider a global reward signal r^n at every time step $n \geq 0$ that is observable by both agents.

The objective is that the agents learn local policies $\mu_1(s_1;\phi_1)$ and $\mu_2(s_2;\phi_2)$ parametrized by ϕ_1 and ϕ_2 to maximize the accumulated discounted reward. The global policy is therefore given by $\mu = (\mu_1, \mu_2)$. We propose to use the DDPG algorithm locally at every agent to train the policies μ_1 and μ_2. However, to execute the DDPG algorithm locally, each agent requires access to the parameters ϕ_1^n and ϕ_2^n at every time step $n \geq 0$. These parameter vectors are inherent local information of the agents and therefore are not a priori available at the corresponding other agent.

Let us therefore suppose that the agents use a communication network to exchange ϕ_1^n and ϕ_2^n. Since communication networks typically induce information delays, only $\phi_2^{n-\tau_1(n)}$ and $\phi_1^{n-\tau_2(n)}$ are available at agent 1 and 2, respectively. Here, $\tau_1(n)$ denotes the

[1] The behaviour policy is typically μ distorted by a noise process to encourage exploration.

AoI of $\phi_2^{n-\tau_1(n)}$ at agent 1 at every time step $n \geq 0$. We refer to $\tau_1(n)$ and $\tau_2(n)$ as the AoI random variables variables. The communication network will be discussed in the next section.

Let $\hat{\phi}_2^n := \phi_2^{n-\tau_1(n)}$ denote the latest available parameter vector of agent 2 at agent 1. Suppose, that agent 1 has access to a local approximation $Q_1(s,a;\theta_1^n)$ of the optimal Q-function $Q^*(s,a)$ parametrized by θ_1^n at time n. Further, suppose the agent can sample a realization of the global state $s = (s_1, s_2)$. Then, at time $n \geq 0$, agent 1 can calculate a local policy gradient with respect to Q_1 that is conditioned on the latest available policy of agent 2:

$$\nabla_{\phi_1}\mu_1(s_1;\phi_1^n)\nabla_{a_1}Q_1(s,a;\theta_1^n)\big|_{s=s,a_1=\mu_1(s_1;\phi_1^n),a_2=\mu(s_2;\hat{\phi}_2^n)}. \tag{4}$$

Equation (4) is therefore the ϕ_1 component (3), where ϕ_2^n is replaced by $\phi_2^{n-\tau_1(n)}$. In Algorithm 1, we use this policy gradient to train the local policy at agent 1. The algorithm of agent 2 is defined analogously. We let both agents train its own local Q-function using Deep Q-Learning. The agents might also cooperate to find better approximations of the Q-function (Mnih et al., 2016). Observe that Algorithm 1 requires complete global tuples (s,a,r,s') to execute the training steps. We therefore assume that the agents exchange their local states and actions over the communication network in addition to their local policy parametrizations. Ones a complete tuple is available, each agent stores it in a local replay memory. Then the local Q-function and local policy is updated using sampled batches from the replay memory.

We would like to point out that the difference compared to the central DDPG algorithm (Lillicrap et al., 2016) is that the Bellman loss gradient in line 12 and the policy gradient in line 13 of the algorithm take into account the most recent available parametrisation of the policy of agent 2. This implies that depending to this parametrization, i.e. the expected behaviour of agent 2, agent 1 will update its Q-function and its policy differently. For completeness, one should add target networks to the algorithm, since they are an important ingredient to stabilize RL with neural network function approximators (Lillicrap et al., 2016).

Algorithm 1: Algorithm at agent 1.

1 Randomly initialize critic and actor weights ;
2 Initialize replay memory $\mathcal{R}$;
3 **for** *the entire duration* **do**
4 Observe the current state s_1^n ;
5 Execute action $a_1^n = \mu_1(s_1^n;\phi_1^n)$;
6 Allocate s_1^n, a_1^n and θ_1^n for transmission to agent 2 ;
7 Exchange data with agent 2 ;
8 Store new tuples (s,a,r,s') in $\mathcal{R}$;
9 Sample a random batch of transitions (s^l,a^l,r^l,s^{l+1}) from $\mathcal{R}$;
10 Compute targets $y^l = r^l + \alpha Q_1(s^{l+1}, \mu_1(s_1^{l+1};\phi_1^n), \mu_2(s_2^{l+1};\hat{\phi}_1^n);\theta_1^n)$;
11 Update critic to minimize $(Q_1(s^l,s^l;\theta_1^n) - y^l)^2$ averaged over the sampled batch ;
12 Update μ_1 using (4) averaged over the sampled batch ;
13 **end**

3.2 Wireless Network Environment

Recall that Algorithm 1 requires that the agents do exchange their policy parameters ϕ_i^n as well as their local information (s_i^n, a_i^n), respectively. In this section we describe a simple network environment used by the two agents to exchange this information.

We represent the channel between the agents by a signal-to-interference-plus-noise-ratio (SINR) models

$$\text{SINR}_1^n = \frac{p_2}{I_1^n}, \qquad \text{SINR}_2^n = \frac{p_1}{I_2^n}. \tag{5}$$

At agent 1, SINR_1^n is the instantaneous SINR of a received signal transmitted with power p_2 during time slot n by agent 2; I_1^n is the experienced interference-noise power during time slot n with some unknown, but stationary distribution. For a time slot n, we assume that SINR_1^n and SINR_2^n are constant. Given SINR_1^n in (5), we assume an SINR-threshold β such that whenever the event $A_1^n := \{\text{SINR}_{1,n} \geq \beta\}$ occurs a transmission from node 2 to 1 can be successfully received; hence, errors are due to interference. The event A_2^n is defined analogously. The threshold β depends on the modulation, coding and path characteristics associated with each agent. The effective bit rate R for the event A_1^n is constraint by then Shannon bounded $R < B\log_2(1+\beta)$ for a given bandwidth B.

Observe that typically the dimension of ϕ^n is significantly larger that the dimension of (s_i^n, a_i^n), since the number of parameters of a function approximator (e.g. neural networks) would have to be increased to allow better approximation as the dimension of (s_i^n, a_i^n) increases. Lets assume that $\dim(\phi) = d_1$ and that $\dim((s_i,a_i)) = d_2$, with $d_1 >> d_2$.

Now, observe that we are only interested in the most recent ϕ_i^n, while for training of the actor-critic algorithm we can use all samples (s_i^n, a_i^n). Therefore, since $d_1 >> d_2$, we may instead of transmitting a single update for ϕ_i transmit a large number of samples (s_i^n, a_i^n). However, its unclear how we should weight

the two data transmissions. For example, if $d_1 \approx kd_2$ for some $k \geq 0$ we could transmit k samples after every transmission of some ϕ_i^n and therefore give equal bandwidth usage to both. However, we may fix any ratio independent of d_1 and d_2.

We now consider the following communication protocol: Every agent keeps a backlog of samples that have not been transmitted to the other agent. Then, each agent transmits up to $k \geq 0$ samples after every complete transmission of its policy. Note that multiple events A_i^n might need to occur to exchange all components of some ϕ_i^n, since d_2 is potentially large. However, notice that the required number of events is fixed and determined by d_1 and d_2, the used coding scheme and the effective bit rate R. The following lemma shows that under suitable conditions the AoI variables $\tau_1(n)$ and $\tau_2(n)$ grow asymptotically slower than $\sqrt{n}$. Moreover, it shows that infinitely many global tuples reach each agent.

Lemma 1. *Suppose that $p_1 > \mathbb{E}[I_2^n]\beta$, $p_2 > \mathbb{E}[I_1^n]\beta$ and that the events A_i^n are i.i.d., then we have that*

$$\mathbb{P}\left(\tau_i(n) > \sqrt{n} \text{ infinitely often}\right) = 0 \tag{6}$$

for both $i \in \{1,2\}$. Moreover, there is sequence $\{n_i^k\}_{k\geq 0}$, such that all samples $(s_i^{n_i^k}, a_i^{n_i^k})$ have been received at agent i.

Proof. It follows from $p_1 > \mathbb{E}[I_2^n]\beta$, $p_2 > \mathbb{E}[I_1^n]\beta$ and (Redder et al., 2021, Lemma 3) that $\mathbb{P}((A_i^n)^c) > \varepsilon$, for some $\varepsilon > 0$. Here, $(A_i^n)^c$ denotes the complement of A_i^n. Recall, that a fixed finite number of events A_i^n guarantees successful exchange of some ϕ_i^n. Let $l \geq 0$ be the required number of events. Using that the events A_i^n are i.i.d., tt can then be shown that

$$\mathbb{P}(\tau_i(n) > m) \leq C\varepsilon^{m/l}, \tag{7}$$

with some constant c independent of n and m. It follows that all $\tau_i(n)$ are stochastically dominated by a random variable with finite second moment, which implies (6) using the Borel-Cantelli Lemma. In then follows from (6) that there exists a sample path dependent constant N, such that $\tau_i(n) \leq \sqrt{n}$ for $n \geq N$. □

Remark 1. *For Lemma 1 we considered that the interference-noise processes are stationary. However, one may also assume that the processes are non-stationary, but instead that one is able to track the processes mean asymptotically up some error $c \geq 0$. Then, assume $p_1 > (\mathbb{E}[I_2^n] + c)\beta$ and $p_2 > (\mathbb{E}[I_1^n] + c)\beta$ in Lemma 1. Furthermore, it is sufficient that these conditions hold merely periodically or with some positive probability. Then the agents may use an arbitrary medium access and power scheduling protocol at all other time instances. Finally, the i.i.d. assumption in Lemma 1 can be weakened to merely exponential dependency decay as discussed in (Redder et al., 2021), or even more generally to dependency decay with certain summability properties.*

3.3 Theoretical Implication of Lemma 1

A detailed theoretical analysis of Algorithm 1 is out of the scope for this work. However, we would like to explain the use Lemma 1.

Consider Algorithm 1, where policies and Q-functions are approximated by feed forward neural networks. It was recently shown in (Ramaswamy and Hullermeier, 2021) that for a large class of activation functions, including the Gaussian error linear unit (GELU), the gradient of the neural network with respect to its parameters is locally Lipschitz continuous in the parameter coordinate. Hence, $\nabla_{\phi_1}\mu_1(s;\phi_1)$ is locally Lipschitz in the ϕ_1 coordinate.

Suppose that ϕ_1^n is updated with a decreasing step size sequence $\gamma(n)$ with $\sum_{n\geq 0}\gamma(n) = \infty$, $\sum_{n\geq 0}\gamma(n)^2 < \infty$. Further, suppose that Algorithm 1 is stable, i.e. $\sup_{n\geq 0}\|\phi_1^n\| < \infty$ a.s. and $\sup_{n\geq 0}\|s^n\| < \infty$ a.s. Stability of the algorithm iterates can be enforced by projecting the iterates to sufficiently large compact set after each update. The stability of the state trajectory can be relaxed to non-compact sets like $\mathbb{R}^d$ see (Ramaswamy and Hullermeier, 2021).

Under these assumptions, we can restrict our attention to a compact set, where $\nabla_{\phi_1}\mu_1(s;\phi_1)$ is Lipschitz continuous in ϕ_1. One can then show that the error in Algorithm 1, when considering $\theta_2^{n-\tau_1(n)}$ instead of θ_2^n, satisfies

$$e_1^n \in o(\gamma(n)\tau_1(n)). \tag{8}$$

It now follows from the assumption that γ is decreasing with $\sum_{n\geq 0}\gamma(n)^2 < \infty$ that $\gamma(n) \in o(1/\sqrt{n})$. Using Lemma 1 we have that $\tau_1(n) \leq \sqrt{(}n)$ for $n \geq N$ and a sample bath dependent $N \geq 0$. It therefore follows that e_1^n vanishes asymptotically. Hence, Algorithm 1 with and without lossy communication behaves asymptotically identical.

4 MULTI-AGENT REINFORCEMENT LEARNING FOR FLOW CONTROL

In this section we introduce a two-agent flow control problem to illustrate the use of Algorithm 1. Flow control is an important problem in chemical industries, data centre cooling and water treatment plants. In our two-agent example each agent has to control one valve of a flow control problem. Specifically, we consider the water filter in Figure 1.

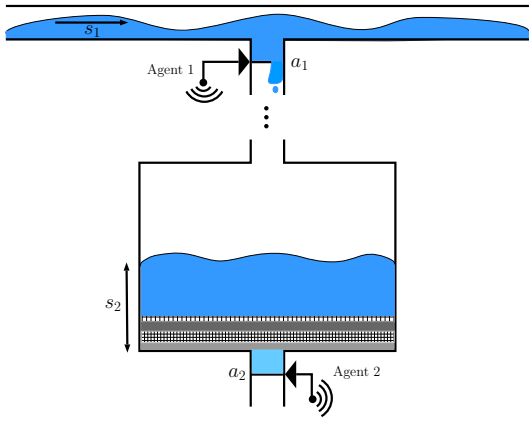

Figure 1: Illustration of the two agent water filter flow control problem.

4.1 Water Filter Flow Control Problem

In Figure 1 agent 1 has to control the inflow to the water filter from a main water line using valve a_1. Agent 2 has to control the outflow of the water filter using valve a_2. We consider that time is slotted into small intervals of constant length Δ. The discrete time steps are indexed by $n \in \mathbb{N}$. We refer to a time slot n as the time interval from time step $n-1$ to n. For both agents, we consider that the valves can be positioned continuously from close to open. Therefore, at every time step n the agents have to pick actions $a_1^n, a_2^n \in [0,1]$ for the associated time slot n. For every time slot n, we denote the sampled flow in the main line by $s_1^n \in S_1 \subset \mathbb{R}_{\geq 0}$ and the sampled water level in the water filter by $s_2^n \in S_2 \subset \mathbb{R}_{\geq 0}$. The problem is complicated, since the agents have no information about the dynamics of the main water line and the water filter. Additionally, the agents have no information about the policy of the other agent, i.e. the agents initially have no information about the strategy of the other agent to control its valve. However, we assume that the agents can use a wireless network to exchange information, which therefore enables the use of Algorithm 1.

Given s_1^n and a_1^n for some time step n, we assume a continuous function $f(s_1^n, a_1^n)$ that determines the inflow to the filter during time slot n. Equivalently, we can assume that agent 1 can measure the inflow to the filter. Other than that we make no additional assumptions. *The agents have no information about the dynamics of the filter or of the flow in the main water line.* We consider that the agents have to learn how to control the valves solely based on their local actions, observations and the information communicated with the other agent.

4.2 Partial Observability Issues

Algorithm 1 formulates a decentralized multi-agent solution based on locally observed states. The training procedure, however, can use delayed global information that has been communicated over a network. The solution therefore falls under the paradigm of decentralized training with central data, but decentralized execution. Depending on how the local state spaces are defined, agents may or may not be able to find good solutions for a problem, since the local policies should only use local states, such that inference is decentralized.

Now consider the flow control problem of the previous subsection. If agent 1 can only observe the state s_1 and never state s_2, then it can only learn to take very conservative actions, since it has no information about the current amount of water in the water filter. We expect better solutions, if both agents can observe the whole state space $s = (s_1, s_2)$. A system theoretic solution to this would now be to utilize a suitable local estimator and replace s_2^n by an estimate $\hat{s}_2^n$ at agent 1to execute a local policy that is functions of the global state space. Alternative, one could directly replace s_2^n by its delayed counterpart and utilize a recurrent architecture for the policies μ_1 and μ_2, see for example (Hausknecht and Stone, 2015).

Since our objective with this example is to illustrate the effect of lossy communication on the learning algorithm and not the effect on the quality inference, we assume here for simplicity that both agents can observe s_1 and s_2. Below, we therefore formulate the MDP for the two-agent flow control problem, where both agents observe $s = (s_1, s_2)$. Moreover, the agents therefore only need to communicate their local actions a_1^n and a_2^n, as well as their policy parametrizations ϕ_1^n and ϕ_2^n.

4.3 An MDP for Two-agent Flow Control

We consider the following high-level objective for the two-agent flow control problem:

(i) Maximize the through flow of the filter.

(ii) Avoid under- and overflows, i.e. $0 < s_2^n < 1$ for all $n \geq 0$.

(iii) Try to keep a reserve of $s_2^n \approx 1/2$ as good as possible for all $n \geq 0$.

The state space for both agents is defined as $\mathcal{S}_{1,2} := S_1 \times S_2$. The action spaces are defined as $\mathcal{A}_1 := [0,1]$ and $\mathcal{A}_2 := [0,1]$, respectively. To achieve the formulated objective, we consider a continuous reward

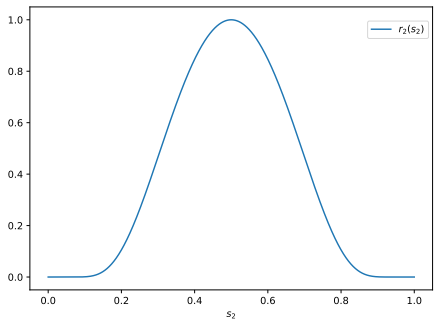

Figure 2: Plot of the second component, $r_2(s_2)$, of the reward function.

function $r(s_1,s_2,a_1) = f(s_1,a_1)c + r_2(s_2)$, with

$$r_2(s_2) = \exp\left(\frac{(2s_2-1)^2}{(s_2-1)s_2}\right), \tag{9}$$

and some constant $c > 0$ that may be used to weight the importance of (i) over (ii) and (iii). Recall that f is the function that determines the inflow to the water filter. The inflow can also be measured by agent 1 and communicated to agent 2 alongside a_1, since the reward is also only required for training and not for inference. Figure 2 shows the second component of the reward function. We choose this function to satisfy point (ii) and (iii) of our objective. Clearly, various other shapes for the reward function are possible.

5 NURMERICAL SIMULATION

In this section, we present the numerical evaluation of Algorithm 1 for the water flow control MDP of Section 4.3. Our goal is to illustrate the convergence of our algorithm while the two agents communicate over a lossy communication network. Most importantly, we illustrate for this example how the lossy communication influences the rate of convergence of Algorithm 1.

5.1 System and Algorithm Model

For our simulation we consider the following differential equation

$$A\frac{ds_2(t)}{dt} - K_1 s_1(t)a_1(t) + K_2(\rho g s_2(t))a_2(t) = 0 \tag{10}$$

to model the water filter. Here A is the area of the water filter, ρ is the density of water, g the acceleration of mass, and K_1 and K_2 are constants. The factor $\rho g s_2(t)$ is the so called differential pressure at time t (Rossiter, 2021). The second term is a bilinear model for the inflow of the system. The third term is a bilinear model for the outflow of the filter. The constant K_2 includes the resistance of the filter and the resistance of the outflow pipe. This is only an approximate model for "small" changes of the flow. However, it serves the purpose for our evaluation. For our simulation, we solve equation (10) numerically for time slots n of length $\Delta = 1s$, where we assume that s_1 and s_2 are approximately constant over each time slot. The agents then observe the sample of $s^n = (s_1^n, s_2^n)$ from the beginning of each time slot. For the constants in eq. (10), we use $K_1 = 0.1$, which corresponds to a maximum inflow of 10% of the main flow, and $K_2 = 10^{-5}$, which models the high resistance of the water filter. Finally, we use $c = 0.1$ to give high weight to balancing the water filter around $s_2 = 1/2$.

For Algorithm 1, we use the following set of parameters: Both agents approximate the Q-function by a two-layer neural network with 256 and 128 Gaussian error linear units (GELU's) as activation's. The final single neuron output layer is linear layer. Both policies μ_1 and μ_2 are also represented by two-layer neural networks, where each layer has 64 GELU's and a tanh output layer. We use $\alpha = 0.95$ and a training batch size of 128. We also use target networks and an Ornstein-Uhlenbeck noise process for exploration (Lillicrap et al., 2016).

5.2 Network Model

To emulate a delaying communication network, we merely consider two Bernoulli processes. We assume that $\mathbb{P}(A_i^n) = \lambda$ at every time step n for both agent 1 and 2. We then choose $\lambda \in \{1, 1/2, 1/4, 1/8, 1/16, 1/32\}$. Practically, this corresponds a suitable choice of the power schedule p_1 and p_2 for given i.i.d. interference-noise processes I_1^n and I_2^n. However, to make the communication model more realistic, we assume a narrowband wireless channel with $B = 10Mhz$ and a typical choice $\beta = 7$. Then, we can obtain an effective bitrate of $19^{Mbit}/_s$. Now, suppose we code every real number using 64 bits. Then, observe that each policy contains 4544 real-valued parameters and since we merely require the exchange of the actions a_i we only need to exchange one real number to obtain a sample (s_i, a_i). Therefore, we require at least 16 time slots of successful communication to transmit a single policy update, while in each time slot we can transmit more than 300 samples. For our experiments, we give roughly equal bandwidth usage to the parameter- and sample transmissions. Specifically, each agent transmits a single update to its parameter vector followed by the transmission of up to $300 \cdot 16$ samples or until its backlog is empty.

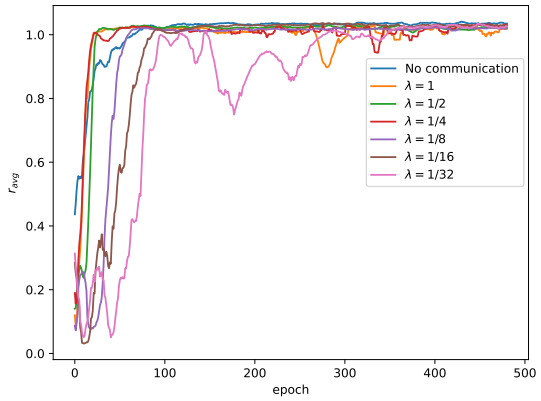

Figure 3: Average reward after each training epoch.

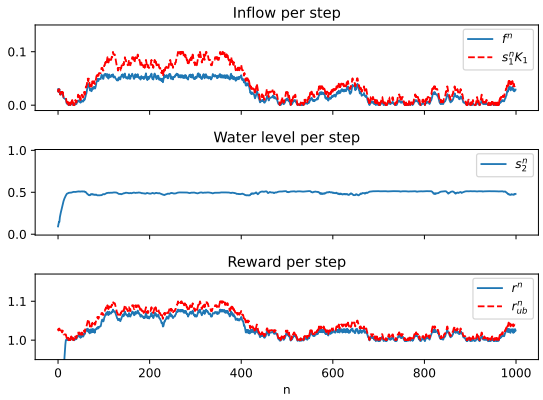

Figure 4: Main flow, inflow and water level during one trajectory after training.

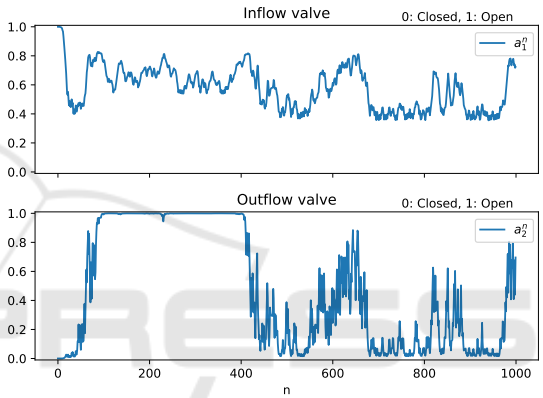

Figure 5: Actions associated with the simulated trajectory after training.

5.3 Training

We train the agents over 500 epochs with length $T = 100$. The Q- and policy networks are updated with the ADAM optimizer (Kingma and Ba, 2015) using Tensorflow 2. We use the following learning rate schedules

$$\gamma(n) = \frac{e^{-8}}{\frac{n}{1000}+1} + \frac{e^{-7}-e^{-8}}{\left(\frac{n}{1000}+1\right)^2}, \quad \delta(n) = \frac{e^{-8}}{\frac{n}{1000}+1} \tag{11}$$

These learning rates satisfy the standard assumptions from the stochastic approximation (SA) literature, i.e. they are not summable, but square summable. Moreover, they satisfy $\frac{\gamma(n)}{\delta(n)} \to 1$. Although non-standard in the SA literature, this assumption enables the use of tools developed in (Ramaswamy and Hullermeier, 2021) to show the convergence of Algorithm 1 (Redder et al., 2022).

Figure 3 shows the average reward at the end of each training epoch evaluated without exploration noise on a new trajectory of length $T = 1000$ for every λ. Observe that for all λ the algorithm converges to a policy of similar average reward compared to Algorithm 1 without communication and global information access. Moreover and as expected, as λ decreases the rate of convergence also decreases. However, for $\lambda \in \{1, 1/2, 1/4\}$ the effect of lossy communication seams to be minor.

After training, we evaluated the final policies on a new trajectory. For illustration we show the results for $\lambda = 1/16$. In Figure 4 we show the inflow to the water filter, the water level in the filter and the reward per step. In the water level plot, we see that the agents learned to quickly balance the water level around the desired height $s_2^n = 1/2$. The inflow is upper bounded by the maximal possible inflow for $a_1^n = 1$. The reward per step is upper bounded by 1 plus the aforementioned upper bound. However, the maximal possible inflow and therefore the maximal possible reward per step may not be obtainable, since opening the inflow valve fully may lead to an increase of the desired height even when the outflow valve is fully opened. We can observe this between time step $n = 50$ to $n = 400$. For this, consider the associated actions during the trajectory in Figure 5. We see that when the inflow is to large in $[50, 400]$, agent 1 has to reduce the inflow to the system, while agent 2 has to fully open the outflow. In this time interval the inflow part of the reward becomes significant and therefore the agents learned to maximize the inflow.

We can also observe that for the second halve of the trajectory the algorithm could potentially improve with longer training. Here, the agents mainly balance the water level at the desired height, since the inflow is relatively small and since we chose $c = 0.1$. However, opening the inflow valve more can still lead to a slightly larger reward per step. In summary, we have seen that agents are able to learn non-trivial policies for an unknown environment in the presence of highly uncertain communication.

6 CONCLUSIONS

We showed how the DDPG algorithm can be implemented in a distributed, multi-agent scenario using a communication network. We gave a theoretical discussion of the impact of the communication errors on the learning algorithm and validated the convergence of our algorithm for a two-agent example. For future work, we are interested in a fully end-to-end learning based solution, where agents use communicated, delayed state information as their local states. This can be done using recurrent architectures for the policies μ_1 and μ_2. The architecture would use the most recent available state and its corresponding age. The objective is that the recurrent architecture internally predicts the true current state, and then selects an action conditioned on the prediction error and the policy that one would associate with global state observations.

ACKNOWLEDGEMENTS

Adrian Redder was supported by the German Research Foundation (DFG) - 315248657 and SFB 901. We would also like to thank Eva Graßkemper for creating Figure 4.

REFERENCES

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR (Poster)*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

Nota, C. and Thomas, P. S. (2020). Is the policy gradient a gradient? In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 939–947.

Ramaswamy, A., Bhatnagar, S., and Quevedo, D. E. (2020). Asynchronous stochastic approximations with asymptotically biased errors and deep multi-agent learning. *IEEE Transactions on Automatic Control*.

Ramaswamy, A. and Hullermeier, E. (2021). Deep q-learning: Theoretical insights from an asymptotic analysis. *IEEE Transactions on Artificial Intelligence*.

Redder, A., Ramaswamy, A., and Karl, H. (2021). Practical sufficient conditions for convergence of distributed optimisation algorithms over communication networks with interference. *arXiv preprint 2105.04230*.

Redder, A., Ramaswamy, A., and Karl, H. (2022). Asymptotic convergence of deep multi-agent actor-critic algorithms. *Under review for AAAI 2022*.

Rossiter, A. (2021). Modelling, dynamics and control. https://controleducation.group.shef.ac.uk/chaptermodelling.html. Accessed: 25.05.2021.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS