2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561645

Coding for Distributed Multi-Agent Reinforcement Learning

Baoqian Wang[1] Junfei Xie[2] Nikolay Atanasov[3]

***Abstract*—This paper aims to mitigate straggler effects in synchronous distributed learning for multi-agent reinforcement learning (MARL) problems. Stragglers arise frequently in a distributed learning system, due to the existence of various system disturbances such as slow-downs or failures of compute nodes and communication bottlenecks. To resolve this issue, we propose a coded distributed learning framework, which speeds up the training of MARL algorithms in the presence of stragglers, while maintaining the same accuracy as the centralized approach. As an illustration, a coded distributed version of the multi-agent deep deterministic policy gradient (MADDPG) algorithm is developed and evaluated. Different coding schemes, including maximum distance separable (MDS) code, random sparse code, replication-based code, and regular low density parity check (LDPC) code are also investigated. Simulations in several multi-robot problems demonstrate the promising performance of the proposed framework.**

I. Introduction

Many real-life applications involve interaction among multiple intelligent systems, such as collaborative robot teams [1], internet-of-things devices [2], agents in cooperative or competitive games [3], and traffic management devices [4]. Reinforcement learning (RL) [5] is an effective tool to optimize the behavior of intelligent agents in such applications based on reward signals from interaction with the environment. Traditional RL algorithms, such as Q-Learning [6] and policy gradient [3], can be scaled to multiple agents by simultaneous application to each individual agent. However, learning independently for each agent performs poorly as the environment is non-stationary from the perspective of a single agent due to the actions of the other agents [3], [7]. Multi-agent reinforcement learning (MARL) [6] focuses on mitigating these challenges by adding other agents' policy parameters to the Q function [8] or relying on importance sampling [9]. Yang et al. [10] propose a mean-field Q learning algorithm, which uses Q functions defined only with respect to an agent's own action and those of its neighbors instead of all agent actions. The multi-agent deep deterministic policy gradient (MADDPG) [3] is an extension of the deep deterministic policy gradient (DDPG) algorithm [11] to a multi-agent setting. MADDPG uses a Q function that depends on all agent observations and actions but local control policies, defined over the observation and action of an individual agent. One key challenge faced by MARL approaches is that the training computational complexity scales with the number of agents in the environment. For large-scale MARL applications, the traditional centralized training mechanism that runs in a single compute node could thus be cost-prohibitive.

Distributing the training workload to multiple compute nodes is a promising solution for accelerating RL and MARL training, which has attracted a lot of attention recently. The first massively distributed architecture for deep RL is presented in [12]. It relies on an off-policy deep Q network algorithm that uses multiple actors and learners running in parallel. The asynchronous advantage actor-critic (A3C) algorithm [13] allows decorrelating the training data from multiple environments executed asynchronously and thus can be used for either on-policy or off-policy learning. Multiple variants of A3C have been developed, including GPU A3C (GA3C) [14], that allows A3C to be trained on a hybrid CPU-GPU architecture, and advantage actor-critic (A2C) [15], which achieves decorrelation with synchronous training. Distributed learning has received much less attention in the MARL setting. Simões et al. [16] extend the A3C framework to multi-agent systems by running multiple compute nodes in parallel to update the parameters asynchronously, with each node simulating multiple agents interacting with the environment. In [4], a multi-agent version of A2C with synchronous learning is investigated to address a traffic signal control problem.

In the aforementioned distributed MARL algorithms, learning is performed either synchronously or asynchronously, with a central controller distributing tasks and collecting the results. A major challenge for synchronous learning is its vulnerability to system disturbances, such as slow-downs or failures of individual learner nodes and communication bottlenecks in the network traffic, which are prevalent in mobile cloud computing and mobile fog/edge computing systems [17]. Learners that suffer from such issues become *stragglers* that can significantly delay the learning process. Asynchronous learning [18], [19] can help mitigate the impact of stragglers but suffers from other limitations, including slower convergence rate, lower accuracy, and more

This work is supported by the National Science Foundation (NSF) under grants 1953048, 1953049, and 2048266, and ARL DCIST CRA W911NF-17-2-0181.

[1] Baoqian Wang is with the Department of Electrical and Computer Engineering, University of California, San Diego and San Diego State University, La Jolla, CA, 92093 (e-mail: bawang@ucsd.edu).

[2] Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA, 92182 (e-mail: jxie4@sdsu.edu).

[3] Nikolay Atanasov is with the Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA, 92093 (e-mail: natanasov@ucsd.edu).

challenging analysis and debugging [6], [15].

We propose *coding theory* strategies [20], [21] to mitigate straggler effects in synchronous distributed learning for MARL problems. Coding techniques have been successful in improving the resilience of communication, storage, and cache systems to uncertain system disturbances [22]–[24]. Such techniques have recently been applied to speed up distributed computation tasks in the presence of stragglers, including matrix-vector [20], [25] and matrix-matrix [21] multiplication, linear inverse problems [26], multivariate polynomials [27], [28], convolution [29], and gradient descent [30]–[32]. Nevertheless, the merits of coding techniques remain to be explored in the RL and MARL settings.

The main *contribution* of this work is a coded distributed learning framework that can be applied with any policy gradient method to solve MARL problems efficiently despite possible straggler effects. As an illustration, we apply the proposed framework to create a coded distributed version of MADDPG [3], a state-of-the-art MARL algorithm. Furthermore, to gain a comprehensive understanding of the benefits of coding in distributed MARL, we investigate various codes, including the maximum distance separable (MDS) code, random sparse code, replication-based code, and regular low density parity check (LDPC) code. Simulations in several multi-robot problems, including cooperative navigation, predator-prey, physical deception and keep away tasks [3], indicate that the proposed framework speeds up the training of policy gradient algorithms in the presence of stragglers, while maintaining the same accuracy as a centralized approach.

II. Multi-Agent Reinforcement Learning

This section introduces the MARL problem and reviews the class of policy gradient algorithms for MARL problems.

A. Problem Formulation

In MARL, multiple agents learn to achieve specific goals by interacting with the environment. Let M be the number of agents. Denote the state and action of agent $i \in [M] := \{1, \dots, M\}$ at time t by $s_{i,t} \in \mathcal{S}_i$ and $a_{i,t} \in \mathcal{A}_i$, respectively, where $\mathcal{S}_i$ and $\mathcal{A}_i$ are the corresponding state and action spaces. Let $\mathbf{s}_t := (s_{1,t}, \dots, s_{M,t}) \in \mathcal{S} := \prod_{i\in[M]} \mathcal{S}_i$ and $\mathbf{a}_t := (a_{1,t}, \dots, a_{M,t}) \in \mathcal{A} := \prod_{i\in[M]} \mathcal{A}_i$ denote the joint state and action of all agents. At any time t, a joint action $\mathbf{a}_t$ applied at state $\mathbf{s}_t$ triggers a transition to a new state $\mathbf{s}_{t+1} \in \mathcal{S}$ according to an (unknown) conditional probability density function (pdf) $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. After each transition, each agent i receives a reward $r_i(\mathbf{s}_t, \mathbf{a}_t) \in \mathbb{R}$, where the reward functions r_i may be different for different agents.

Given a joint state $\mathbf{s}$, the objective of each agent i is to choose a stochastic policy, specified by a pdf $\pi_i(a_i|\mathbf{s})$ over the agent's action space $\mathcal{A}_i$, such that the expected cumulative discounted reward:

$$V_i^{\boldsymbol{\pi}}(\mathbf{s}) := \mathbb{E}_{\substack{\mathbf{s}_t\sim p\\ \mathbf{a}_t\sim\boldsymbol{\pi}}}\left[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}_t, \mathbf{a}_t)|\mathbf{s}_0 = \mathbf{s}\right] \tag{1}$$

is maximized. In (1), $\gamma \in (0, 1]$ is a discount factor, $\boldsymbol{\pi} := (\pi_1, \dots, \pi_M)$ denotes the joint policy of all agents. The function $V_i^{\boldsymbol{\pi}}(\mathbf{s})$ is known as the value function of agent i associated with joint policy $\boldsymbol{\pi}$. Alternatively, an optimal policy π_i^* for agent i can be obtained by maximizing the action-value function:

$$Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{\substack{\mathbf{s}_t\sim p\\ \mathbf{a}_t\sim\boldsymbol{\pi}}}\left[\sum_{t=0}^{\infty} \gamma^t r_i(\mathbf{s}_t, \mathbf{a}_t)|\mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right]$$

and setting $\pi_i^*(a_i|\mathbf{s}) \in \arg\max_{a_i} \max_{\mathbf{a}_{-i}} Q_i^*(\mathbf{s}, \mathbf{a})$, where $Q_i^*(\mathbf{s}, \mathbf{a}) := \max_{\boldsymbol{\pi}} Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a})$ and $\mathbf{a}_{-i}$ denotes the actions of all agents except i.

B. Policy Gradient Methods

There are two major classes of MARL algorithms: *value-based* and *policy-based*. Value-based algorithms aim to approximate the optimal action-value function $Q_i^*(\mathbf{s}, \mathbf{a})$. Examples include extensions of Q-learning [33], DQN [34], and SARSA [5] to multi-agent settings, such as Independent Q-learning [6], Inter-Agent Learning [35], and multi-agent SARSA [36]. Policy-based algorithms directly optimize over the space of policy functions. A general approach, known as *policy gradient* [5], is to define parametric policy functions $\pi_i(a_i|\mathbf{s}; \theta_i)$, using linear feature or neural network parameterization, and update the parameters along the gradient of the value function: $\theta_i \leftarrow \theta_i + \alpha\nabla_{\theta_i} V_i^{\boldsymbol{\pi}}(\mathbf{s})$. The value function gradient is obtained via the policy gradient theorem [5]:

$$\nabla_{\theta_i} V_i^{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}_{\substack{\mathbf{s}_t\sim p\\ \mathbf{a}_t\sim\boldsymbol{\pi}}}\left[\nabla_{\theta_i} \log \pi_i(a_{i,t}|\mathbf{s}_t; \theta_i) Q_i^{\boldsymbol{\pi}}(\mathbf{s}_t, \mathbf{a}_t)\right].$$

Different approaches exist for estimating $Q_i^{\boldsymbol{\pi}}(\mathbf{s}_t, \mathbf{a}_t)$ in the above equation, such as the *REINFORCE* algorithm [37], which uses sample returns $\sum_{t=0}^{T} \gamma^t r_i(\mathbf{s}_t, \mathbf{a}_t)$ from several episodes of length T, critic methods [38] that parameterize and approximate $Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a})$ in addition to the policy, or the trust region policy optimization (TRPO) algorithm [39], which ensures that a policy improvement condition is satisfied.

III. Coded Distributed Learning for MARL

In this section, we introduce a coded distributed learning framework that can be incorporated with any general policy gradient method to solve the aforementioned MARL problem efficiently in the presence of stragglers. Before we describe this framework, we first overview a distributed learning framework for MARL without coding.

A. Uncoded Distributed Learning for MARL

A distributed learning system (see Fig. 1 for an illustration) for RL/MARL consists of a *central controller* and multiple *learners*. The central controller is a server that maintains all agent parameters, and uses them to construct policies for the agents. In each training iteration, it executes multiple episodes and sends both the collected environment data and the parameters of all agents to the learners. Once a learner receives these data, it updates the agent parameters and then sends the result back to the central controller.

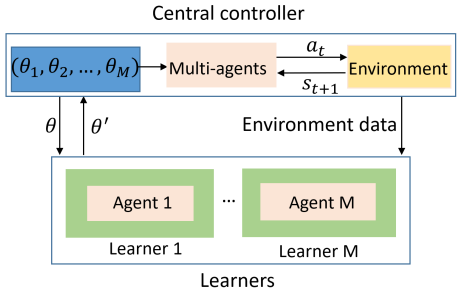


Fig. 1: Illustration of uncoded distributed learning for MARL.

Suppose there are N learners in the system, where $N \geq M$ and M is the number of agents. We use a matrix $\mathbf{C} \in \mathbb{R}^{N \times M}$ to describe the agent-to-learner assignment. In particular, learner j will update the parameters for agent i if the (j,i)-th entry of the assignment matrix $\mathbf{C}$, denoted as $c_{j,i}$, satisfies $c_{j,i} \neq 0$. Let θ_i' be the parameters for agent i after the update with gradient ascent along $\nabla_{\theta_i} V_i^{\boldsymbol{\pi}}(\mathbf{s})$. Each learner will send $y_j' = \sum_{i=1}^{M} c_{j,i}\theta_i'$ back to the central controller.

In an uncoded distributed learning system, each learner updates the parameters for a single agent. In other words, different learners are responsible for different agents and only M out of the N learners are used. The assignment matrix $\mathbf{C}_{Uncoded} \in \mathbb{R}^{N \times M}$ has entries:

$$c_{j,i} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{else.} \end{cases}$$

In this paper, we focus on synchronous learning systems. Therefore, the central controller does not update the agent policies until all updated parameters have been received. The learning efficiency is thus bounded by the slowest learner at each training iteration. Any node or link failure may affect the whole task. In the following subsection, we introduce a coded distributed learning framework to address this problem.

B. Coded Distributed Learning for MARL

To enhance the resilience of a distributed learning system to uncertain stragglers, our idea is to introduce redundancies into the computation by assigning one or more agents to each learner. This is achieved by applying coding schemes to construct an assignment matrix $\mathbf{C}$ that satisfies $rank(\mathbf{C}) = M$ and has one or more non-zero entries in each row.

In this coded learning framework, the central controller will be able to recover all updated parameters, denoted as $\boldsymbol{\theta}' = [\theta_1'^T, \ldots, \theta_M'^T]^T$ with results received from only a subset of the learners. Particularly, let $\mathcal{I} = \{j \mid y_j' \text{ is received}\}$ represent the set of learners whose results are received by the central controller by a certain time. Also let $\mathbf{C}_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}| \times M}$ be a submatrix of $\mathbf{C}$ formed by the j-th rows of $\mathbf{C}$, $\forall j \in \mathcal{I}$. Then $\boldsymbol{\theta}'$ can be recovered when $rank(\mathbf{C}_{\mathcal{I}}) = M$ via:

$$\boldsymbol{\theta}' = (\mathbf{C}_{\mathcal{I}}^T \mathbf{C}_{\mathcal{I}})^{-1} \mathbf{C}_{\mathcal{I}}^T \mathbf{y}_{\mathcal{I}}', \tag{2}$$

where $\mathbf{y}_{\mathcal{I}}'$ is the aggregate result derived by concatenating y_j', $\forall j \in \mathcal{I}$. In the following subsection, we introduce four different coding schemes and explain how they can be used to construct a coded assignment matrix $\mathbf{C}$.

C. Coding Schemes

1) Replication-based Code: Using a replication-based coding scheme [40], agents are assigned to the learners in a round-robin fashion and each agent is assigned to at least $\lfloor \frac{N}{M} \rfloor$ learners. The (j,i)-th entry of its assignment matrix $\mathbf{C}_{Replication}$ is then given by

$$c_{j,i} = \begin{cases} 1, & \text{if } i = (j \bmod M) + M\mathbb{1}_{(j \bmod M)=0} \\ 0, & \text{else} \end{cases}$$

where $(j \bmod M)$ finds the remainder when j is divided by M, and $\mathbb{1}$ is the indicator function.

2) MDS Code: An MDS code [41] specifies the assignment matrix such that any M rows have full rank, by using, e.g., a Vandermonde matrix [42] as follows

$$\mathbf{C}_{MDS} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_M \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_M^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{N-1} & \alpha_2^{N-1} & \cdots & \alpha_M^{N-1} \end{bmatrix}$$

where α_i, $i \in [M]$, can be any non-zero real number. Note that all entries of $\mathbf{C}_{MDS}$ are non-zero, meaning that each learner needs to compute parameters for all the agents.

3) Random Sparse Code: A random sparse code [40] enables a sparser assignment matrix $\mathbf{C}_{Random}$ with (j,i)-th entry randomly generated from a Gaussian distribution $\mathcal{N}(0,1)$ with probability p_m, i.e., $\mathbb{P}(c_{j,i} = \epsilon) = p_m$, where $\epsilon \sim \mathcal{N}(0,1)$. Otherwise, $c_{j,i} = 0$ with $\mathbb{P}(c_{j,i} = 0) = 1 - p_m$. Note that, by choosing an appropriate p_m, we can control the sparsity of the assignment matrix $\mathbf{C}_{Random}$.

4) Regular LDPC Code: The regular LDPC based coding scheme [43] constructs an assignment matrix $\mathbf{C}_{LDPC}$ in three steps. The first step constructs a permutation matrix:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \in \mathcal{F}_2^{w \times w},$$

where $\mathcal{F}_2$ represents the binary field and w is a prime number that satisfies $\frac{N}{w} \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ represents positive integers. The second step constructs a parity check matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_w & \mathbf{I}_w & \mathbf{I}_w & \cdots & \mathbf{I}_w \\ \mathbf{I}_w & \mathbf{A} & \mathbf{A}^w & \cdots & \mathbf{A}^{w-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{I}_w & \mathbf{A}^{w-2} & \mathbf{A}^{2(w-2)} & \cdots & \mathbf{A}^{(w-2)(w-1)} \\ \mathbf{I}_w & \mathbf{A}^{w-1} & \mathbf{A}^{2(w-1)} & \cdots & \mathbf{A}^{(w-1)(w-1)} \end{bmatrix} \in \mathcal{F}_2^{Y \times N}$$

where Y satisfies $\frac{Y}{w} \in \mathbb{Z}^+$ and $Y \leq N$ and $\mathbf{I}_w \in \mathbb{R}^{w \times w}$ is an identity matrix. Finally, the assignment matrix is constructed by $\boldsymbol{C}_{LDPC} = [\mathbf{I}_M, \mathbf{P}]^T \in \mathcal{F}_2^{M \times N}$, where $\mathbf{P}$ is extracted from the parity check matrix with $\mathbf{H} = \left[-\mathbf{P}^\top, \mathbf{I}_{N-M}\right]$. A positive feature of this coding scheme is that it can be quickly decoded by an iterative algorithm introduced in [44]. The

complexity of this algorithm is $\mathcal{O}(M)$, while the decoding complexity of other schemes is $\mathcal{O}\left(M^3\right)$.

IV. Coded Distributed MADDPG

In this section, we apply the coded distributed learning framework to enhance the training efficiency of MADDPG [3] and its resilience to stragglers. Unlike the general formulation of MARL described in Sec. II, MADDPG adopts a deterministic policy $\pi_i(s_i)$ for each agent $i \in [M]$, which only depends on the local state s_i. The value function $Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a})$ of agent i still depends on the joint state $\mathbf{s}$ and joint action $\mathbf{a}$. Moreover, for each agent i, four neural networks are used to approximate its policy $\pi_i(s_i; \theta_{p,i})$, value function $Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}; \theta_{q,i})$, target policy $\hat{\pi}_i(s_i; \hat{\theta}_{p,i})$, and target value function $Q_i^{\hat{\boldsymbol{\pi}}}(\mathbf{s}, \mathbf{a}; \hat{\theta}_{q,i})$, respectively. $\hat{\boldsymbol{\pi}} = (\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_M)$ denotes the concatenated target policy. Therefore, $\theta_i = [\theta_{p,i}, \theta_{q,i}, \hat{\theta}_{p,i}, \hat{\theta}_{q,i}]$. To optimize these parameters using the coded distributed learning framework, a central controller and multiple learners need to be implemented, whose interactions are described in detail as follows.

A. Central Controller

In each training iteration, the central controller generates policies for the agents based on the current parameters $\boldsymbol{\theta}$. The agents then execute these policies for several episodes and store the transitions $\{(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}')\}$ into a replay buffer $\mathcal{D}$, where $\mathbf{r} = [r_1, r_2, \ldots, r_M]$. The central controller samples a random mini-batch $\mathcal{B}$ from the replay buffer $\mathcal{D}$ and broadcasts the mini-batch $\mathcal{B}$ and current parameters $\boldsymbol{\theta}$ to the learners. After that, it waits for the learners to update the parameters and return back the results. As soon as the central controller receives sufficient results, it recovers the updated parameters $\boldsymbol{\theta}'$, sends acknowledgements to the learners and then proceeds to the next iteration. This procedure is summarized in Algorithm 1 (Line 1-15).

B. Learners

When each learner j receives parameters $\boldsymbol{\theta}$ and mini-batch $\mathcal{B}$ from the central controller, it updates the parameters θ_i for each agent i assigned to it, where $i \in [M], c_{j,i} \neq 0$. Specifically, the value function parameters $\theta_{q,i}$ are updated by minimizing the temporal-difference error:

$$J(\theta_{q,i}) = \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{s},\mathbf{a},\mathbf{s}',r)\in\mathcal{B}} \left(L_i^{\hat{\boldsymbol{\pi}}}(\mathbf{s}', r_i) - Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}; \theta_{q,i})\right)^2 \quad (3)$$

where $L_i^{\hat{\boldsymbol{\pi}}}(\mathbf{s}', r_i) = r_i + \gamma Q_i^{\hat{\boldsymbol{\pi}}}(\mathbf{s}', \hat{\boldsymbol{\pi}}(\mathbf{s}'))$. The policy parameters $\theta_{p,i}$ are updated using gradient ascent, according to the policy gradient theorem [5]:

$$\nabla_{\theta_{p,i}} J(\theta_{p,i}) \approx \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{s},\mathbf{a},\mathbf{s}',r)\in\mathcal{B}} \nabla_{\theta_{p,i}} \pi_i(s_i; \theta_{p,i}) \nabla_{a_i} Q_i^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}). \quad (4)$$

The target policy and value function parameters are updated via Polyak averaging:

$$\begin{aligned} \hat{\theta}_{p,i} &\leftarrow \tau \hat{\theta}_{p,i} + (1-\tau)\theta_{p,i} \\ \hat{\theta}_{q,i} &\leftarrow \tau \hat{\theta}_{q,i} + (1-\tau)\theta_{q,i}, \end{aligned} \quad (5)$$

Algorithm 1: Coded Distributed MADDPG

```
// Central controller:
1  Initialize θ for all agents
2  for k = 1 : max_iteration do
3      for ℓ = 1 : max_episode_number do
4          for t = 1 : max_episode_length do
5              For each agent i, select a_{i,t}^ℓ = π_i(s_{i,t}^ℓ; θ_{p,i}).
6              Execute joint action a_t^ℓ and receive new
                state s'^ℓ_{t+1} and reward r_{t+1}^ℓ.
7              Store (s_t^ℓ, a_t^ℓ, r_{t+1}^ℓ, s'^ℓ_{t+1}) in replay buffer D.
8      Sample a random minibatch B from D.
9      Broadcast B and θ to the learners.
10     y' ← [ ]
11     do
12         Listen to the channel and collect results y'_j
            from the learners: y' ← [y'; y'_j], j ∈ [N]
13     while θ' is not recoverable
14     Send acknowledgements to learners.
15     Recover θ' using (2) and let θ ← θ'
// Learner j:
16 for k = 1 : max_iteration do
17     Listen to the channel.
18     if receiving B and θ from central controller then
19         y_j ← 0; i ← 1
20         while i ≤ M and no acknowledgement
            received do
21             if c_{j,i} ≠ 0 then
22                 Update θ_{p,i} using gradient ascent with
                    gradient computed using (4).
23                 Update θ_{q,i} by minimizing the loss in
                    (3) with gradient descent.
24                 Update θ̂_{p,i} and θ̂_{q,i} using (5).
                    y'_j ← y'_j + c_{j,i} θ_i
25             i ← i + 1
26         Send y'_j to the central controller.
```

where $\tau \in (0, 1)$ is a hyperparameter. Note that learner j does not update the parameters of the agents not assigned to it. In contrast, in the original MADDPG algorithm, each learner updates all agents' parameters at each training iteration. The procedure followed by the learners is summarized in Algorithm 1 (Line 16-26).

V. Experiments

We conduct a variety of experiments in the robotics environments proposed in [3] to evaluate the performance of our coded distributed MADDPG algorithm. We first describe the environments and then present the results.

A. Environments

The following four environments are considered. The agent interactions are either cooperative, competitive or mixed.

- **Cooperative navigation:** In this environment, M agents aim to cooperatively reach M landmarks, while avoiding collisions with each other. The landmarks are not assigned to the agents explicitly. Therefore, the agents must learn to reach all landmarks by themselves (see Fig. 2(a)).
- **Predator-prey:** In this environment, $M-K$ good agents with restricted speed of motion aim to cooperatively chase K faster adversary agents while avoiding static obstacles (see Fig. 2(b)).
- **Physical deception:** In this environment, any of $M-1$ good agents needs to reach a single known target among L landmarks. One adversary agent also desires to reach the target landmark but does not know which one is the target and must infer it from the movements of the good agents. Therefore, good agents must learn to spread out to cover all landmarks so as to confuse the adversary (see Fig. 2(c)).
- **Keep away:** This environment is similar to the physical deception, but with $M-K$ good agents and K adversary agents. Additionally, the adversaries can get in the way to prevent the good agents from reaching the target (see Fig. 2(d)).

B. Training Reward Comparison

To demonstrate the effectiveness of the proposed coded distributed learning framework, we compare the coded distributed MADDPG with the original centralized MADDPG. The total number of agents is set to $M = 8$. In the competitive environments, the number of adversary agents is set to $K = 4$. The cumulative rewards of all agents averaged over 250 training iterations for each environment is shown in Fig. 3(a)-3(d). As we can see, in all environments, the coded distributed MADDPG is able to generate the same quality of policies as the original MADDPG and converges within the same number of iterations.

C. Training Time Comparison

To evaluate the efficiency of the proposed coded distributed learning framework, we compare the training time of the coded distributed MADDPG with the uncoded distributed MADDPG. Different coding schemes, including the replication-based code, MDS code, random sparse code and regular LDPC are evaluated.

To implement the distributed learning systems, we used Amazon EC2 and chose m5n.large [45] instances to implement the central controller and learners. To simulate uncertain stragglers, we randomly pick k learners at each training iteration as stragglers, which delay returning the results for t_s seconds. The specific experimental settings adopted for each environment are summarized as follows:

- **Cooperative navigation:** $k \in \{0, 1, 2\}$, $t_s = 0.25s$.
- **Predator prey:** $k \in \{0, 2, 4\}$, $t_s = 1s$.
- **Physical deception:** $k \in \{0, 5, 8\}$, $t_s = 1s$.
- **Keep away:** $k \in \{0, 5, 8\}$, $t_s = 1.5s$.

Besides varying the value of k in each environment, we also vary the total number of agents M. Results obtained when $M = 8$ and $M = 10$ are shown in Fig. 4 and Fig. 5, respectively. Each value represents the training time of an algorithm averaged over 5 iterations, with the total number of iterations set to 50. In all experiments, the total number of learners is set to $N = 15$, and the parameter p_m in the random sparse code is set to 0.8.

Analyzing the two figures, we can make the following observations. First, when there are no stragglers or the straggler effect, indicated by the amount of delay t_s introduced by the stragglers, is small (e.g., in the cooperative navigation scenario), the uncoded distributed MADDPG achieves the best performance in all experiments. This demonstrates the good performance of the traditional uncoded distributed learning framework when the influence of stragglers is small.

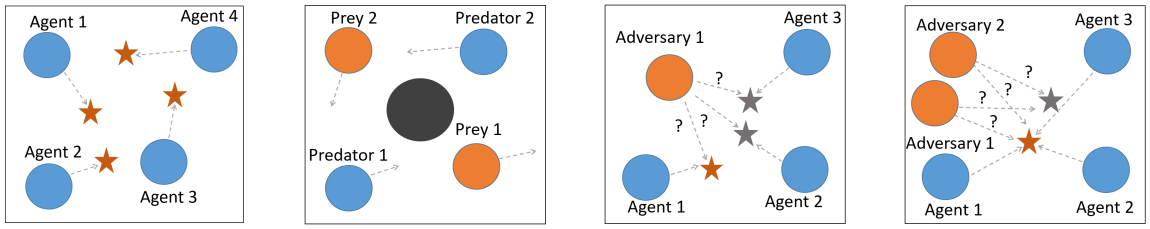
When the straggler effect is relatively large, we can observe that the performance of the uncoded scheme degrades significantly and achieves worse performance than most coding schemes. Furthermore, its performance remains stable as the number of stragglers increases. This is because each straggler is delayed by the same amount of time, t_s, such that each iteration is always delayed by t_s no matter how many stragglers are present. In contrast, the coding schemes are generally more robust to stragglers, except when the number of stragglers exceeds the limit that the coding schemes can tolerate.

Next, we analyze the performance of different coding schemes. We can observe that the MDS code (orange bars) is very robust to stragglers when the number of stragglers k does not exceed the maximum tolerable number $N - M$. However, when $k > N-M$, the MDS performance degrades significantly, as shown in Fig. 4(c)-4(d) and Fig. 5(c)-5(d). Furthermore, we can observe that when $k < N-M$ and when the straggler effect is relatively large, MDS outperforms the uncoded scheme, as well as the replication-based and regular LDPC codes, as shown in Fig. 4(b)-4(d) and Fig. 5(b)-5(d). However, when the straggler effect is relatively small, the MDS code has the worst performance, as shown in Fig. 4(a) and 5(a), due to the high computational redundancies it introduces through the dense assignment matrix.

The random sparse code (green bars) shows a similar performance to the MDS code. This is because when its parameter p_m takes a large value ($p_m = 0.8$ in our experiments), the assignment matrix generated by this type of code has a similar density as the one generated by the MDS code.

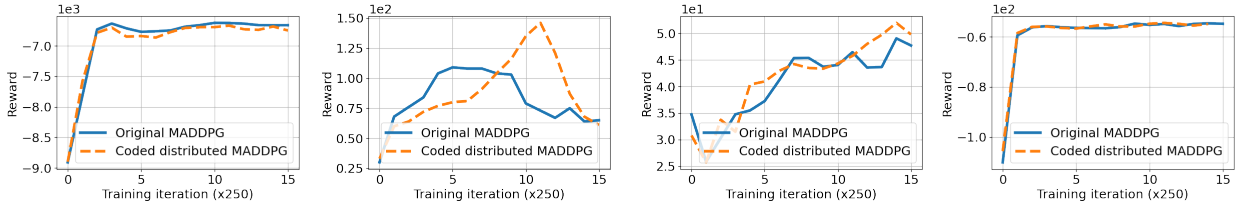
Finally, both the replication-based code (red bars) and regular LDPC code (purple bars) are more affected by an increase in the stragglers, as their assignment matrices are sparser. However, they achieve better performance than MDS and random sparse codes when the straggler effect is small, as shown in Fig. 4(a) and 5(a), and when there are many stragglers, as shown in Fig. 4(c)-4(d) and Fig. 5(c)-5(d).

The above studies suggest guidelines for selecting an appropriate coding scheme in different scenarios. If the strag-



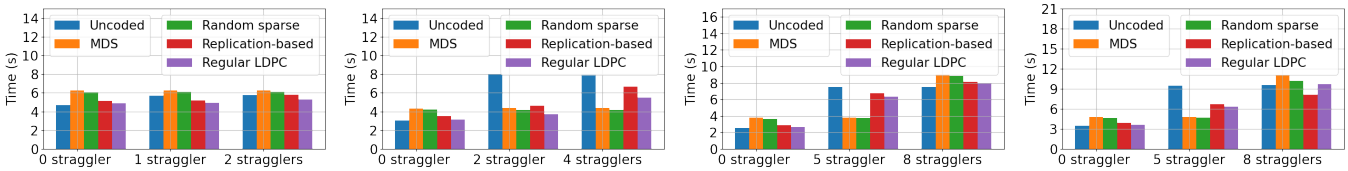
(a) (b) (c) (d)

Fig. 2: Illustration of the experimental environment for (a) cooperative navigation, (b) predator-prey, (c) physical deception, and (d) keep away.



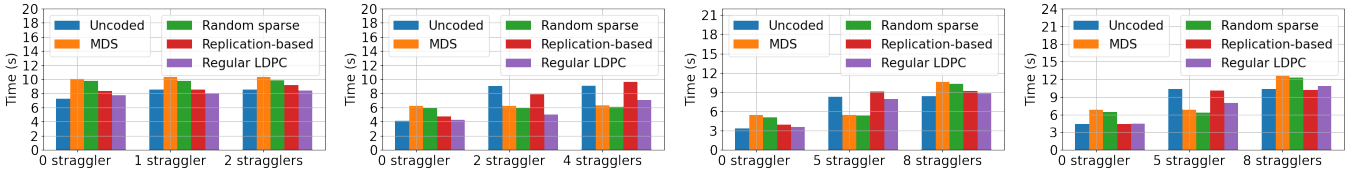
(a) (b) (c) (d)

Fig. 3: Average cumulative training reward on (a) cooperative navigation, (b) predator-prey, (c) physical deception, and (d) keep away.



(a) (b) (c) (d)

Fig. 4: Average training time on (a) cooperative navigation, (b) predator prey, (c) physical deception, and (d) keep away, when $M = 8, N = 15$.



(a) (b) (c) (d)

Fig. 5: Average training time on (a) cooperative navigation, (b) predator prey, (c) physical deception, and (d) keep away, when $M = 10, N = 15$.

gler effect is small, uncoded, replication-based or a regular LDPC scheme should be preferred. If the straggler effect is relatively large but the number of stragglers is small, the regular LDPC or replication-based schemes are suggested. Finally, if the straggler effect is large and many stragglers are present, the MDS code or a random sparse code with p_m set to a large value should be chosen.

VI. Conclusion

This paper introduces a coded distributed learning framework for MARL, which improves the training efficiency of policy gradient algorithms in the presence of stragglers while not degrading the accuracy. We applied the proposed framework to a coded distributed version of MADDPG, a start-of-the-art MARL algorithm. Simulations on several multi-robot problems demonstrate the high training efficiency of the coded distributed MADDPG, compared with the traditional uncoded distributed learning approach.

The results also show that the coded distributed MADDPG generates policies of the same quality as the original centralized MADDPG and converges within the same number of iterations. Furthermore, we investigated different coding schemes including replication-based, MDS, random sparse, and regular LDPC codes. Simulation results show that the MDS and random sparse codes can tolerate more stragglers but introduce larger computation overhead. Additionally, the replication-based and regular LDPC codes produce less overhead but are more susceptible to stragglers.

REFERENCES

[1] F. Bonnet, L. Cazenille, A. Gribovskiy, J. Halloy, and F. Mondada, "Multi-robot control and tracking framework for bio-hybrid systems with closed-loop interaction," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4449–4456.

[2] A. Forestiero, "Multi-agent recommendation system in internet of things," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 772–775.

[3] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

[4] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[6] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[7] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems." *HAL*, 2012.

[8] G. Tesauro, "Extending q-learning to general adaptive multi-agent systems," in *Advances in neural information processing systems*, 2004, pp. 871–878.

[9] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1702.08887*, 2017.

[10] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," *arXiv preprint arXiv:1802.05438*, 2018.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *International Conference on Learning Representations (ICLR)*, 2016.

[12] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[14] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," *arXiv preprint arXiv:1611.06256*, 2016.

[15] "A2C." [Online]. Available: https://openai.com/blog/baselines-acktr-a2c/

[16] D. Simões, N. Lau, and L. P. Reis, "Multi-agent actor centralized-critic with communication," *Neurocomputing*, 2020.

[17] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[18] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[19] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[20] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proc. of IEEE ISIT 2016*. IEEE, aug 2016, pp. 1143–1147.

[21] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. of IEEE ISIT 2017*, 2017, pp. 2418–2422.

[22] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proc. of Allerton 2015*. IEEE, apr 2015, pp. 964–971.

[23] C. Liu, Q. Wang, X. Chu, Y.-W. Leung, and H. Liu, "Esetstore: An erasure-coded storage system with fast data recovery," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2001–2016, 2020.

[24] Y. Zhu, P. P. Lee, Y. Xu, Y. Hu, and L. Xiang, "On the speedup of recovery in large-scale erasure-coded storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1830–1840, 2013.

[25] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, mar 2018.

[26] Y. Yang, P. Grover, and S. Kar, "Coded Distributed Computing for Inverse Problems," *Advances in Neural Information Processing Systems*, pp. 710–720, 2017.

[27] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 1215–1225.

[28] M. Rudow, K. Rashmi, and V. Guruswami, "A locality-based approach for coded computation," *arXiv preprint arXiv:2002.02440*, 2020.

[29] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2403–2407.

[30] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *Proc. of the 34th International Conference on Machine Learning*. PMLR, 2017, pp. 3368–3376.

[31] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 857–866.

[32] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," *IEEE Journal on Selected Areas in Information Theory*, 2020.

[33] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[34] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep q-learning," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 486–489.

[35] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in neural information processing systems*, 2016, pp. 2137–2145.

[36] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, and A. Bouguettaya, "Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition," in *International Conference on Service-Oriented Computing*. Springer, 2014, pp. 154–168.

[37] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[38] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[40] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2413–2417.

[41] J. Lacan and J. Fimes, "Systematic mds erasure codes based on vandermonde matrices," *IEEE Communications Letters*, vol. 8, no. 9, pp. 570–572, 2004.

[42] A. Klinger, "The vandermonde matrix," *The American Mathematical Monthly*, vol. 74, no. 5, pp. 571–574, 1967.

[43] E. M. Gabidulin and M. Bossert, "On the rank of ldpc matrices constructed by vandermonde matrices and rs codes," in *International Symposium on Information Theory*, 2006, pp. 861–865.

[44] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[45] "Amazon EC2 m5n.large instance." [Online]. Available: https://aws.amazon.com/ec2/instance-types/m5/