# CESMA: Centralized Expert Supervises Multi-Agents

Alex Tong Lin
Department of Mathematics
UCLA
atlin@math.ucla.edu

Mark J. Debord NAVAIR

Katia Estabridis NAVAIR Gary Hewer NAVAIR Stanley Osher UCLA

### **Abstract**

We consider the reinforcement learning problem of training multiple agents in order to maximize a shared reward. In this multi-agent system, each agent seeks to maximize the reward while interacting with other agents, and they may or may not be able to communicate. Typically the agents do not have access to other agent policies and thus each agent observes a non-stationary and partially-observable environment. In order to obtain multi-agents that act in a decentralized manner, we introduce a novel algorithm under the framework of centralized learning, but decentralized execution. This training framework first obtains solutions to a multiagent problem with a single centralized joint-space learner. This centralized expert is then used to guide imitation learning for independent decentralized multi-agents. This framework has the flexibility to use any reinforcement learning algorithm to obtain the expert as well as any imitation learning algorithm to obtain the decentralized agents. This is in contrast to other multi-agent learning algorithms that, for example, can require more specific structures. We present some theoretical error bounds for our method, and we show that one can obtain decentralized solutions to a multi-agent problem through imitation learning.

### 1 Introduction

Reinforcement Learning (RL) is the problem of finding an action policy that maximizes reward for an agent embedded in an environment [44]. It has recently has seen an explosion in popularity due to its many achievements in various fields such as, robotics [18], industrial applications [8], game-playing [25, 39, 37], and the list continues. However, most of these achievements have taken place in the single-agent realm, where one does not have to consider the dynamic environment provided by interacting agents that learn and affect one another.

This is the problem of Multi-agent Reinforcement Learning (MARL) where we seek to find the best action policy for each agent in order to maximize their reward. The settings may be cooperative, and thus they might have a shared reward, or the setting may be competitive, where one agent's gain is another's loss. Some examples of a multi-agent reinforcement learning problem are: decentralized coordination of vehicles to their respective destinations while avoiding collision, or the game of pursuit and evasion where the pursuer seeks to minimize the distance between itself and the evader while the evader seeks the opposite. Other examples of multi-agent tasks can be found in [29] and [20].

The key difference between MARL and single-agent RL (SARL) is that of interacting agents, which is why the achievements of SARL cannot be absentmindedly transferred to find success in MARL. Specifically, the state transition probabilities in a MARL setting are inherently non-stationary from the perspective of any individual agent. This is due to the fact that the other agents in the environment are also updating their policies, and so the Markov assumptions typically needed for SARL convergence are violated. This aspect of MARL gives rise to instability during training, where each agent is essentially trying to learn a moving target.

In this work, we present a novel method for MARL in the cooperative setting (with shared reward). Our method first trains a centralized expert with full observability, and then uses this expert as a supervisor for independently learning agents. There are a myriad of imitation/supervised learning algorithms, and in this work we focus on adapting DAgger (Dataset Aggregation) [34] to the multi-agent setting. After the imitation learning stage, the agents are able to successfully act in a decentralized manner. We call this algorithm Centralized Expert Supervises Multi-Agents (CESMA). CESMA adopts the framework of centralized training, but decentralized execution [16], the end goal of which is to obtain multi-agents that can act in a decentralized manner.

#### 2 Related works

The most straight-forward way of adapting single-agent RL algorithms to the multi-agent setting is by having agents be independent learners. This was applied in [46], but this training method gives stability issues, as the environment is non-stationary from the perspective of each agent [23, 3, 4]. This non-stationarity was examined in [28], and stabilizing experience replay was studied in [10]

Another common approach to stabilizing the environment is to allow the multi-agents to communicate. In [41], they examine this using continuous communications so one may backpropagate to learn to communicate. And in [9], they give an in-depth study of communicating multi-agents, and also provide training methods for discrete communication. In [31], they decentralize a policy by examining what to communicate and by utilizing supervised learning, although they mathematically solve for a centralized policy and their assumptions require homogeneous communicating agents.

Others approach the non-stationarity issue by having the agents take turns updating their weights while freezing others for a time, although non-stationarity is still present [7]. Other attempts adapt Q-learning to the multi-agent setting: Distributed Q-Learning [17] updates Q-values only when they increase, and updates the policy only for actions that are not greedy with respect to the Q-values, and Hysteretic Q-Learning [22] provides a modification. Other approaches examine the use of parameter sharing [13] between agents, but this requires a degree of homogeneity of the agents. And in [48], their approach to non-stationarity was to input other agents' parameters into the Q function. Other approaches to stabilize the training of multi-agents are in [40], where the agents share information before selecting their actions.

From a more centralized view point, [27, 32, 42] derived a centralized Q-value function for MARL, and in [49], they train a centralized controller and then sequentially select actions for each agent. The issue of an exploding action space was examined in [47].

A few works that follow the framework of centralized training, but decentralized execution are: RLar (Reinforcement Learning as Rehearsal) [16], COMA (Counterfactual Multi-Agent), and also [36, 11] – where the idea of knowledge-reuse is examined. In [6], they examine decentralization of policies from an information-theoretic perspective. There is also MADDPG [20] where they train in a centralized-critics decentralized-actors framework; after training completes, the agents are seperated from the critics and can execute in a fully distributed manner.

For surveys of MARL, see articles in [2, 30].

# 3 Background

In this section we briefly review the requisite material needed to define MARL problems. Additionally we summarize some of the standard approaches in general reinforcement learning and discuss their use in MARL.

**Dec-POMDP:** A formal framework for multi-agent systems is called a decentralized partiallyobservable Markov decision process (Dec-POMDP) [1]. A Dec-POMDP is a tuple observable Markov decision process (Dec-POMDP) [1]. A Dec-POMDP is a tuple  $(I, \mathcal{S}, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, P, R)$  where I is the finite set of agents indexed 1 to M, S is the set of states,  $\mathcal{A}_i$  is the set of actions for agent i, and thus  $\prod_{i=1}^M \mathcal{A}_i$  is the joint action space,  $\mathcal{O}_i$  is the observation space of agent i, and thus  $\prod_{i=1}^M \mathcal{O}_i$  is the joint observation space,  $P = P(\mathbf{s'}, \mathbf{o}|\mathbf{s}, \mathbf{a})$  (where  $\mathbf{o} = (o_1, \dots, o_M)$  and similarly for the others) is the state-transition probability for the whole system, and  $R : \mathcal{S} \times \prod_{i=1}^M \mathcal{A} \to \mathcal{R}$  is the reward. In the case when the joint observations  $\mathbf{o}$  equals the world state of the system, then we call the system a decentralized Markov decision process (Dec-MDP).

**DAgger:** The Dataset Aggregation (DAgger) algorithm [34] is an iterative imitation learning algorithm that seeks to learn a policy from expert demonstration. The main idea is to allow the learning policy to navigate its way through the environment, and have it query the expert on states that the it sees. It does this by starting with a policy  $\hat{\pi}_2$  which learns from the dataset of expert trajectories  $\mathcal{D}_1$  through supervised learning. Using  $\hat{\pi}_2$ , a new dataset is generated by rolling out the policy and having the expert provide supervision on the decisions that the policy made. This new dataset is aggregated with the existing set into  $\mathcal{D}_2 \supset \mathcal{D}_1$ . This process is iterated, i.e. a new  $\hat{\pi}_3$  is trained, another new dataset is obtained and aggregated into  $\mathcal{D}_3 \supset \mathcal{D}_2$  and so on. Learning in this way has been shown to be more stable and have nicer convergence properties as learning utilizes trajectories seen from the learner's state distribution, as opposed to only the expert's state distribution.

**Policy Gradients (PG):** One approach to RL problems are policy gradient methods [45]: instead of directly learning state-action values, the parameters  $\theta$  of the policy  $\pi_{\theta}$  are instead adjusted to maximize the objective,

$$J(\theta) = \mathbb{E}_{s \sim p^{\pi}, a \sim \pi_{\theta}} \left[ \nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a) \right]$$

where  $p^{\pi}$  is the state distribution from following policy  $\pi$ . The gradient of the above expression can written as [45, 44]:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s \sim p^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s|a) Q^{\pi}(s, a)]$$

 $\nabla_{\theta}J(\theta) = \mathbb{E}_{s \sim p^{\pi}, a \sim \pi_{\theta}}[\nabla_{\theta}\log \pi_{\theta}(s|a)Q^{\pi}(s,a)]$  Many policy gradient methods seek to reduce the variance of the above gradient estimate, and thus many study how one estimates  $Q^{\pi}(s,a)$  above [35]. For example, if we let  $Q^{\pi}(s,a)$  be the sample return  $R^t = \sum_{i=t}^T \gamma^{i-t} r_i$ , then we get the REINFORCE algorithm [14]. Or one can choose to learn  $Q^{\pi}(s,a)$  using temporal-difference learning [43, 44], and would obtain the Actor-Critic algorithms [44]. Other policy gradients algorithms are: DPG [38], DDPG [19], A2C and A3C [24], to name a

Policy Gradients have been applied to multi-agent problems; in particular the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [20] uses an actor-critic approach to MARL, and this is the main baseline we test our method against. Another policy gradient method is by [12] called Counterfactual Multi-Agent (COMA), who also uses an actor-critic approach.

#### 4 Methods

In this section, we explain the motivation and method of our approach: Centralized Expert Supervises Multi-Agents (CESMA).

# 4.1 Treating a multi-agent problem as a single agent problem

Intuitively, an optimal strategy of a multi-agent problem could be found by a centralized expert with full observability. This is because the centralized controller has the most information available about the environment, and therefore would not pay a high of cost of partial-observability that independent learners might. This is discussed more in 5.

To find this centralized expert, we treat a multi-agent problem as a single agent problem in the joint observation and action space of all agents. This is done by concatenating the observations of all agents into one observation vector for the centralized expert, and the expert learns outputs that represent the joint actions of the agents.

Our framework does not impose any other particular constraints on the expert. Any expert architecture that outputs an action that represents the joint-actions of all of the agents may be used. Due to that, we are free to use any standard RL algorithm for the expert such as DDPG, DQN, or potentially even analytically derived experts.

#### 4.2 Curse of dimensionality and some reliefs

When training a centralized expert, both the observation space and action space can grow exponentially. For example, if we use a DQN for our centralized expert then the number of output nodes will typically grow exponentially. This is due to each output needing to correspond to an element in the joint action space  $\prod_{i=1}^{M} A_i$ .

This problem has been studied by QMIX [32] and VDNs (Value-Decomposition Networks) [42], where exponential scaling of the output space is solved by having separate Q values for each agent and then using the sum as a system Q. Other techniques such as action branching [47] have also been considered.

In our experiments, we use DDPG (with Gumbel-Softmax action selection if the environment is discrete, as MADDPG does also) to avoid the exploding input nodes of the observation space, as well as exploding output nodes of the action space. Under this paradigm, the input and output nodes only grow linearly with the number of agents.

#### 4.3 CESMA for supervised learning on the multi-agents

In order to perform imitation learning to decentralize the centralized expert policy, we adapt DAgger to the multi-agent setting. There are many ways DAgger can be applied to multi-agents, but we implement a method that best allows the theoretical analysis from [34] to apply: Namely after training the expert, we do supervised learning on a single neural network with disconnected components, each of which corresponds to a multi-agent.

In more detail, after training a centralized expert  $\pi^*$ , we initialize the M agents  $\pi_{\theta_1},\ldots,\pi_{\theta_M}$ , and initialize the dataset  $\mathcal{D}$ . The agents then step through the environment, storing each observation the multi-agents encounter along with its action label:  $(\mathbf{o}, \mathbf{a}_{\text{expert}})$ , where  $\mathbf{a}_{\text{expert}} = \pi^*(\mathbf{o})$ . After  $\mathcal{D}$  has reached a sufficient size, at every kth time step (chosen be practitioner; we used k=1 in our experiments), we sample a batch from this dataset  $\{(\mathbf{o}^{(b)}, \mathbf{a}_{\text{expert}}^{(b)})\}_{b=1}^B$ , and then distribute observation  $o_i^{(b)}$  and action-label  $a_{\text{expert},i}^{(b)}$  to agent i. The observation and action label is then used to perform supervised learning on the agents. Having a shared dataset of trajectories in this way allows us to view  $(\pi_{\theta_1},\ldots,\pi_{\theta_M})$  as a single neural-network with disconnected components, and thus the error bounds from [34] directly apply, as discussed in Section 5.

See Figure 1 for a diagram. The appendix contains pseudo-code for our method.

The aformentioned procedure is sufficient when the agents do not have to communicate in order to solve the environment (e.g. they all have full observability, or their observations are isomorphic to full observability, or the partial observability is not too detrimental). When communication is involved, then we have to modify the above method.

# **4.4** CESMA for multi-agents that communicate

In order to train multi-agents that communicate, especially in an environment where communication is necessary (see Section 6.3 for two such cases), we have to slightly modify the above procedure. Training the centralized expert in a communication scenario follows the original procedure, but since the expert has full observability, it has no incentive to communicate with itself. Indeed, most of the time the reward func-

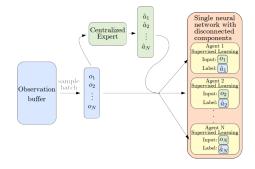


Figure 1: Diagram presenting how the centralized expert guides supervised learning for the multiagents. The multiagents can be thought of as making up the disconnected components of a single-agent learner.

tion is independent of the communication (although if one desired to act covertly, then this may not be the case). Thus any observations and actions regarding communication are not utilized when training the expert. In the next phase, the agents learn to communicate amongst themselves.

We leave the full details in the appendix, and opt here to provide an overview: In the environments we test, the communication action by an agent at time step t-1 will appear to other agents at the next time step t. Then in our dataset of observations, we store together the observations from time step t-1 and t.

The idea is to separate the losses of the agent actions into an *action loss* and a *communication loss*. In order to obtain the action loss, we do supervised learning on the observation and actions at time t where the input is the observation but with updated communications from the other agents, and the label is the centralized expert action label. The communication loss for agent i is computed by first obtaining agent i's output communication action at t-1. This is then combined with the observations and actions and of all other agents (at time t). We query the expert on the correct action for the other agents, and we use this supervised learning loss to backpropagate the errors through to agent i weights. Indeed, our main motivation was the chain rule.

By using this procedure we allow the decentralized agents to learn communications that help reduce the penalty associated with operating in a partially-observable environment. Section 5.1 discusses this in more detail.

In this way, we have alleviated a bit the issue of communication being a sparse reward, which is because even when an agent communicates to another agent, the receiving agent is still learning, so it is unclear if the broadcasting agent was correct, or the receiving agent was correct; we have a double ambiguity. With an expert supervisor, the correct action by the acting agent is clear.

# 5 Theoretical analysis

Although the proposed framework could handle a myriad of imitation learning algorithms, such as Forward Training [33], SMILe [33], SEARN [5], and more, we use DAgger in our experiments, and thus we follow its theoretical analysis, and provide some multi-agent extensions. And since our method can be viewed as using a single-agent learner with disconnected components so that we can trivially decompose it into multi-agents, then this gives us direct theoretical insights [34].

In our setting, the centralized expert observes the joint observations of all agents, and thus it is a function  $\pi^*: \mathcal{O}_1 \times \cdots \times \mathcal{O}_M \to \mathcal{A}_1 \times \cdots \times \mathcal{A}_M$ , and we can decompose  $\pi^*$  into,

$$\pi^*(\mathbf{o}) = (\pi_1^*(\mathbf{o}), \dots, \pi_M^*(\mathbf{o}))$$

where  $\pi_i^*: \mathcal{O}_1 \times \cdots \times \mathcal{O}_M \to \mathcal{A}_i$ . In order to decentralize our policy, our goal is to find policies  $\pi_1, \dots, \pi_M$  such that,

$$\pi^*(\mathbf{o}) = (\pi_1^*(\mathbf{o}), \dots, \pi_M^*(\mathbf{o}))$$

$$\stackrel{\text{want}}{=} (\pi_i(o_1), \dots, \pi_M(o_M))$$

Note that  $\pi_i^*$  is able to observe the joint observations while  $\pi_i$  is only able to observe its own local observation  $o_i$ . This means we may encounter issues where

$$\pi_i^*(o_1,\ldots,o_{i-1},o_i,o_{i+1},\ldots,o_M) = a_i, \quad \text{but} \quad \pi_i^*(o_1',\ldots,o_{i-1}',o_i,o_{i+1}',\ldots,o_M') = a_i'$$
 by we want

$$\pi_i(o_i) = a_i \text{ or } a_i',$$

Thus the agent policy can act sub-optimally in certain situations, being unaware of the global observation. This unfortunate situation not only afflicts our algorithm, but any multi-agent training algorithm (and in general, any algorithm attempting to solve a POMDP).

We call this the *partial observability problem of decentralization* (note partial observability afflicts any algorithm trying to solve a POMDP (e.g. multi-agent systems), but here we examine from the viewpoint of decentralization).

More concretely, we can say there is is a partial observability problem of decentralization if there exists observations  $(o_1,\ldots,o_{i-1},o_i,o_{i+1},\ldots,o_M)$  and  $(o'_1,\ldots,o'_{i-1},o_i,o'_{i+1},\ldots,o'_M)$  such that  $\pi^*(o_1,\ldots,o_{i-1},o_i,o_{i+1},\ldots,o_M) \neq \pi^*(o'_1,\ldots,o'_{i-1},o_i,o'_{i+1},\ldots,o'_M)$ .

As it pertains to our algorithm, if we can bound the cost of this problem, then we can obtain an error bound. Then suppose  $\ell$  is the 0-1 loss of matching the expert policy  $\pi^*$ , and suppose the partial observability problem of decentralization is such that

$$\mathbb{E}_{\mathbf{o} \sim d_{(\pi_1^{(i)}, \dots, \pi_M^{(i)})}} \left[ \ell(\mathbf{o}, (\pi_1, \dots, \pi_M)) \right] \ge c_p$$

for all iterations i, observations  $\mathbf{o}$  and policies  $(\pi_1, \dots, \pi_M)$ . Then we have the following theorem (whose proof is in the appendix),

**Theorem 1.** If the number of iterations N is  $\tilde{O}(T)$ , and  $J(\pi)$  is the cost (negative of the reward) of executing policy  $\pi$ , then there exists a joint multi-agent policy  $\hat{\pi} \in {\{\hat{\pi}^{(i)}\}_{i=1}^{N}}$  such that

$$J(\hat{\pi}) \le J(\pi^*) + T\epsilon_N + O(1)$$
  
$$\le J(\pi^*) + Tc_n + O(1)$$

where,

$$\epsilon_N = \min_{(\pi_1, \dots, \pi_M) \in \Pi_1 \times \dots \times \Pi_M} \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{\boldsymbol{o} \sim d_{(\pi_1^{(i)}, \dots, \pi_M^{(i)})}} [\ell(\boldsymbol{o}, (\pi_1, \dots, \pi_M))]$$

Under these assumptions, this implies that decentralizing the policy always requires paying a cost of partial observability, independent of the number of iterations N. Although in our experiments, we sometimes find this cost is negligible.

#### 5.1 The need for communication

The most effective setting for our methods is when the multi-agents are all able to observe the full joint observation. From the perspective of each agent the only non-stationarity when learning comes from other agents' policies, as opposed to from the non-agent portion of the environment.

In the situation when each agent only has local observations, then to avoid the partial observability problem of decentralization, there is an incentive to communicate. Namely, we want for the multiagent policy  $(\pi_1, \ldots, \pi_M)$ ,

$$\pi^*(\mathbf{o}) = (\pi_1^*(\mathbf{o}), \dots, \pi_M^*(\mathbf{o}))$$

$$\stackrel{\text{want}}{=} (\pi_i(o_1, c_1), \dots, \pi_M(o_M, c_M))$$

where  $c_i$  is the communication from either all or only some of the other agents, to agent i. Note that we can view  $c_i$  as a function  $c_i : \mathcal{O}_1 \times \cdots \times \mathcal{O}_{i-1} \times \mathcal{O}_{i+1} \times \cdots \times \mathcal{O}_M \to \mathcal{C}_i$  (where  $\mathcal{C}_i$  is some communication action space). Then under the following conditions, we have the error bound:

**Theorem 2.** If the multi-agent communication  $c_i : \mathcal{O}_1 \times \cdots \times \mathcal{O}_{i-1} \times \mathcal{O}_{i+1} \times \cdots \times \mathcal{O}_M \to \mathcal{C}_i$  satisfies the following condition:

$$\pi^*(o_1, \dots, o_{i-1}, o_i, o_{i+1}, \dots, o_M) \neq \pi^*(o'_1, \dots, o'_{i-1}, o_i, o'_{i+1}, \dots, o'_M)$$
implies that  $c_i(o_1, \dots, o_{i-1}, o_{i+1}, \dots, o_M) \neq c(o'_1, \dots, o'_{i-1}, o'_{i+1}, \dots, o'_M)$ 

for all  $i=1,\ldots,M$ , then there is no partial observability cost of decentralization, and thus if the number of iterations N is of  $\tilde{O}(T)$ , then there exists a policy  $\hat{\pi}_{1,\ldots,M} \in \{\hat{\pi}_{1,\ldots,M}^{(i)}\}_{i=1}^N$  such that  $\mathbb{E}_{\boldsymbol{o} \sim d_{\hat{\pi}_1,\ldots,M}} \left[\ell(\boldsymbol{o},\hat{\pi}_{1,\ldots,M})\right] \leq \epsilon_N + O(1/T)$ . Then this implies,  $J(\hat{\pi}_{1,\ldots,M}) \leq J(\pi_{1,\ldots,M}^*) + \epsilon_N T + O(1)$ 

The proof is in the appendix. We remark that a naive communication protocol satisfying the above conditions is when  $c_i$  is the identity operator.

# 6 Experiments

Our experiments are conducted in the Multi-Agent Particle Environment [26, 20] provided by OpenAI, which has basic simulated physics (e.g. Newton's law) and multiple multi-agent scenarios.

#### 6.1 Preliminaries

In order to conduct comparisons to MADDPG, we also use the DDPG algorithm with the Gumbel-Softmax [15, 21] action selection for discrete environments, as they do. For the single-agent centralized expert neureal network, we always make sure the number of parameters matches (or is lower) than that of MADDPG's. For the decentralized agents, we use the same number of parameters as the decentralized agents in MADDPG (i.e. the actor part). We always use the discount factor  $\gamma=0.9$ ,

as that seemed to work best both for our centralized expert, and also MADDPG. Following their experimental procedure, we average our experiments over three runs. And for the decentralization, we trained three separate centralized experts, and used each of them to obtain three decentralized policies. Full details of our hyperparameters is in the appendix. Rewards are also averaged over 1,000 episodes. And we always use two-hidden layer neural networks. Brief descriptions of each environment are provided, and fuller descriptions and some example pictures are placed in the appendix.

#### 6.2 Cooperative Navigation – Homogeneous and Nonhomogenous Agents

Here we examine the situation of N agents occupying N landmarks in a 2D plane, and the agents are either homogeneous or heterogeneous. The (continuous) observations of each agent are the relative positions of other agents, the relative positions of each landmark, and its own velocity. The agents do not have access to others' velocities so we have partial observability. The reward is based on how close each landmark has an agent near it, and the actions of each agent are discrete: up, down, left, right, and do nothing.

In Figure 2, in all cases the fully centralized expert is able to achieve a lower reward than MADDPG and DDPG. We are also able to decentralize the

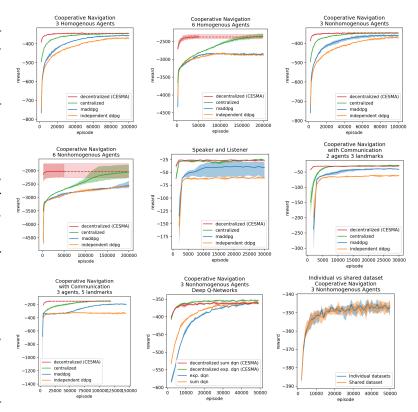


Figure 2: Reward curves for the various multi-agent environments. The dotted red lines for the decentralized curves represent when we stop the decentralization procedure, as the reward sufficiently matches the expert.

expert policy (which was chosen to be the one with lowest reward) so as to reach this same reward. And we remark that our method seems to work better with more agents.

The six nonhomogeneous case works as a good experiment to see what happens when we stop the centralized expert before it truly converges. In this case, decentralization to achieve the same reward as the expert is quickest and occurs within the first 5,000 episodes. Intuitively, it makes sense that a suboptimal solution allows faster convergence.

We observe our method improves sample efficiency over MADDPG: we picked centralized experts that achieved a reward of -350 (averaged over its past 1,000 episodes) which took the centralized expert around 50,000 episodes. Decentralizing these polices so that we obtain a reward higher than -355 took around 7,000 episodes, so in total it took 57,000 episodes to obtain decentralized policies achieving a reward higher than -355. But it takes MADDPG around 80,000 to sometimes more than 100,000 episodes to obtain rewards higher than -355.

We also examined the use of DQNs, one with an exponential number of output nodes, and a QMIX/Centralized VDN that sums the individual agents Q-values so we get linear growth in action. And we also test decentralization performance when each agent has its own dataset of trajectories, in

the 3 nonhomogeneous agents scenario. We hypothesized that the nonhomogeneity of the agents may have an effect on convergence, but this turned out not to be so. See Figure 2 for the reward curves.

# **6.3** Cooperative Communication

Here we we adapt CESMA to a task that involves communication. In this scenario, the communication action taken by each agent at time step t-1 will appear to the other agents at time step t. Although we require continuous communication to backprop, in practice we can use the softmax operator to provide the bridge between the discrete and continuous. And during decentralized execution, our agents are able to act with discrete communication inputs.

We examine three scenarios for CESMA that involve communication, and use the training scenario described in section 4.4. The first scenario called the "speaker and listener" environment has a speaker who broadcasts the correct goal landmark (in a "language" it must learn) out of a possible 3 choices, and the listener, who is blind to the correct goal landmark, must use this information to move there. Communication is a necessity in this environment. The second scenario is cooperative navigation with communication and here we have two/three agents whose observation space includes the goal landmark of the *other* agent, and not their own, and there are three/five possible goal landmarks.

We see in figure 2 that we achieve a lower reward and in a more sample efficient manner. For the speaker and listener environment, the centralized expert near-immediately converges, and same for the decentralization process. And MADDPG has a much higher variance in its convergence. In the cooperative navigation with communication scenarios, the story is similar, that the centralized expert quickly converges, and the decentralization process is near immediate.

#### 6.4 Reward vs. loss, and slow and fast learners

In our experiments with cooperative navigation, when using the cross entropy loss, we did not find an illuminating correlation between the reward and the loss. We reran the experiments in a truer DDPG fashion by solving a continuous version of the environment, and used the mean-squared error for the supervised learning. We examined the loss in the cooperative navigation task with 3 agents, both homogeneous and nonhomogeneous agents. We plot the figures in 6.4. We found that in these cases, the reward and loss were negatively correlated as expected, namely that we achieved a higher reward as the loss decreased. In the nonhomogeneous case, we plot each individual agents' reward vs its loss and found that the big and slow agent had the biggest loss, followed by the medium agent, and the small and fast agent being the quickest learner. This example demonstrates that in nonhomogeneous settings, some agents may be slower to imitate the expert than others.

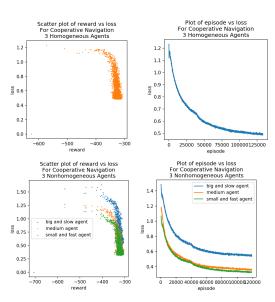


Figure 3: Reward vs. loss, and loss vs. episode.

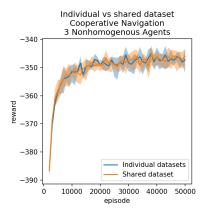
#### 7 Conclusion

We propose a MARL algorithm, called Central-

ized Expert Supervises Multiagents (CESMA), which takes the training paradigm of centralized training, but decentralized execution. The algorithm first trains a centralized expert policy, and then adapts DAgger to obtain decentralized policies that execute in a decentralized fashion.

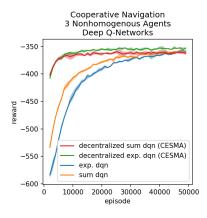
# A Experiment where each agent has its own dataset of trajectories

Here we describe in a little bit more detail our procedure for experimenting with individual dataset trajectories for each agent, vs a shared dataset. Namely, we plot the learning curves for decentralizing a policy in the two cases: (1) When each agent has its own dataset of trajectories, or (2) when there is a shared dataset of trajectories (which is the one we use in the experiments). We tested on the cooperative navigation environment with 3 nonhomogeneous agents. We hypothesized that the nonhomogeneity of the agents would have an effect on the shared reward, but this turned out not to be so. But it is interesting to note that in the main text, we found that the some agents had a bigger loss when doing supervised learning from the expert.



# B Experiment with DQNs

Here we examined decentralizing DQNs. We briefly described it in the main text, and give a bit more info here. We used the cross entropy loss for the supervised learning portion, and used the cooperative navigation environment with 3 nonhomogenous agents. The DQNs we used are: the exponential actions DQN, which is just a naive implementation of DQNs for the multi-agents, and a Centralized VDN where the system Q value is the sum of the individual agent Q values. We used a neural network with 200 hidden units, batch size 64, and for the exponential DQN, we used a learning rate and  $\tau$  of  $5 \times 10^{-4}$ , and for the QMIX/Centralized VDN DQN we used a learning rate and  $\tau$  of  $10^{-3}$ . We also used a noisy action selection for exploration.



# C Pseudo-algorithm of CESMA (without communication)

```
Algorithm 1 CESMA: Centralized Expert Supervises Multi-Agents
```

**Require:** A centralized policy  $\pi^*$  that sufficiently solves the environment.

**Require:** M agents  $\pi_{\theta_1}, \dots \pi_{\theta_M}$ , observation buffer  $\mathcal{D}$  for multi-agent observations, batch size B

1: **while**  $\pi_{\theta_1}, \ldots, \pi_{\theta_M}$  not converged **do** 

- 2: Obtain observations  $o_1, \ldots, o_M$  from the environment
- 3: Obtain agents' actions,  $a_1 = \pi_{\theta_1}(o_1), \dots, a_M = \pi_{\theta_M}(o_M)$
- 4: Obtain expert action labels  $e_i = \pi^*(o_1, \dots, o_M)_i$ , for  $i = 1, \dots, M$
- 5: Store the joint observation with expert action labels  $((o_1, e_1), \dots, (o_M, e_M))$  in  $\mathcal{D}$
- 6: **if**  $|\mathcal{D}|$  sufficiently large **then**
- 7: Sample a batch of B multi-agent observations  $\{((o_1^b, e_1^b), \dots, (o_M^b, e_M^b))\}_{b=1}^B$
- 8: Perform supervised learning for  $\pi_{\theta_i}$  where the inputs are  $\{o_i^b\}$  and the labels are  $\{e_i^b\}$ , for  $i=1,\ldots,M$ .
- 9: end if
- 10: end while

# D Detailed description of CESMA with communicating agents

The main intuition for our idea is very simple: the chain rule. In order to minimize notational burden, we first leave out some details, and then provide them after: If the supervised learning loss function is  $\ell$ , the expert is  $\pi^*$ , and if we have agents:  $\pi_1, \ldots, \pi_M$ , then in order to update the communication of agent i to agent j, then denoting  $c_{-i}$  the communications from all other agents to agent j excluding agent i, then we want to minimize:

$$\min_{\boldsymbol{\tau}} \ell(\pi^*(\mathbf{o})_j, \ \pi_j(o_j, c_{-i} \cup \pi_i(o_i)_{\text{comm}})))$$

and so we should backpropagate through other agents' actions (namely the physical actions which have the expert label, and not the communication action which does not) to update agent i's communication. We note both  $c_{-i}$  and  $\pi_i(o_i)_{\text{comm}}$  are communication actions that are obtained from the current policy of the agents, i.e. we re-query the agents for their communication actions.

And to update the action loss of agent i, we want to minimize

$$\min_{\pi_i} \ell(\pi^*(\mathbf{o})_i, \ \pi_i(o_i, \mathbf{c})_{\text{action}})$$

(note bold characters are vectors, e.g.  $\mathbf{o} = (o_1, \dots, o_M)$ ). We note that  $\mathbf{c}$  is an up-to-date communication action, which we obtain by re-querying the other agents for their communication actions.

The above leaves out some notation for ease of understanding and to get the main idea. The technically and notationally correct version is:

If

- the supervised learning loss function is  $\ell$ ,
- the expert is  $\pi^*$ ,
- the agents at the current iteration are:  $\pi_1, \ldots, \pi_M$ ,
- $o_i$  is the observation of agent i,
- $\hat{o}_i$  is the observation of agent i at the next timestep (i.e. after observing  $o_i$  and communications from other agents, and then taking a step in the environment, to get a new observation),
- $c_i$  is the communication action of other agents towards agent i that accompanies observation  $o_i$ ,
- $\hat{c}_i$  is the updated (i.e. using the most up-to-date policy) communication action of agents towards agent i, i.e.  $\hat{c}_i = (\pi_j(o_j, c_j)_{\text{comm},i})_{j \neq i}$

then we want to minimize:

$$\min_{\pi_i} \frac{1}{M-1} \sum_{j=1, j \neq i}^{M} \ell(\pi^*(\hat{\mathbf{o}})_j, \ \pi_j(\hat{o}_j, \ \hat{c}_j)_{\text{action}})$$
 (communication loss)

where we note  $\hat{c}_j$  contains the communication action from agent i to agent j, and thus we can backprop, and the subscript "action" means the physical action (and not the communication action). And for the action loss, we want to minimize:

$$\min_{\pi_i} \ell(\pi^*(\hat{\mathbf{o}})_i, \ \pi_i(\hat{o}_i, \ \hat{c}_i)_{\text{action}})$$
 (action loss)

where  $\hat{c}_i$  is the updated communications from all other agents, to agent i (i.e. we use the current most-up-to-date policy to form  $\hat{c}_i$ . The need to use an updated version of communication  $\hat{c}_i$  is because the language of the agents tends to change more drastically as training progresses. (One thing we did try was to set a low size limit on the trajectory dataset  $\mathcal{D}$ , but preliminary results showed it did not help much). But we do note that in order to construct  $\hat{c}_i$ , we need to use the observation/communications from time step t-1, so reliance on an older language is still there. One can imagine going all the way back to t=0 in order to update the communications of the whole trajectory, but for our experiments, going back one timestep sufficed.

We give a pseudocode in algorithm 2.

A diagram of the action loss and communication loss is given in figure 4 (action loss) and 5 (communication loss).

We remark that we also considered the case of a hybrid objective, where the actions are learned by supervised learning from the expert, and the communication is learned similar to a standard RL algorithm (e.g. the Q-values are communication actions). Preliminary results showed this did not work well.

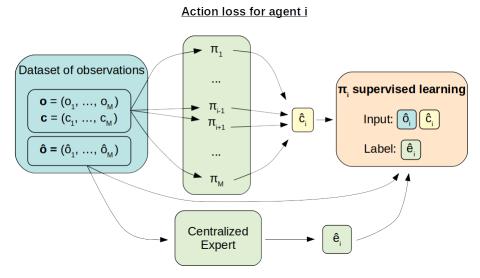


Figure 4: A diagram of the computation of the action loss for agent i. The  $\pi_j$ 's are the most up-to-date policies, and  $\hat{c}_i$  is an updated communication from the other agents.

# Communication loss for agent i

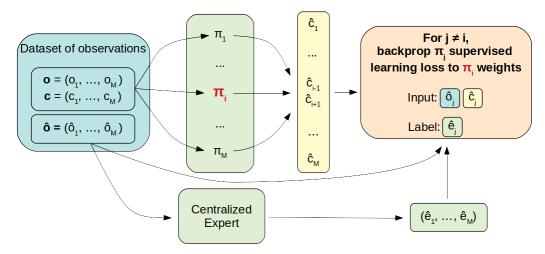


Figure 5: A diagram of the computation of the communication loss for agent i. The  $\pi_j$ 's are the most-up-to-date policies, and the  $\hat{c}_j$ 's are updated communications from the other agents. Note we calculate the action loss of agent  $j, j \neq i$ , and then backpropagate this loss to agent i's weights.

# Algorithm 2 CESMA: Centralized Expert Supervises Multi-Agents (Communicating Agents)

**Require:** A centralized policy  $\pi^*$  that sufficiently solves the environment.

**Require:** M initial agents  $\pi_1, \dots \pi_M$ , observation buffer  $\mathcal{D}$  for multi-agent observations, batch size

**Require:**  $\ell$ , the supervised learning loss

- 1: while  $\pi_1, \ldots, \pi_M$  not converged do
- Obtain the observations and communications  $\{(o_i, c_i)\}_{i=1}^M$  from the environment.
- With these observations, obtain actions and step through the environment, to get new observa-3: tions  $\{\hat{o}_i\}_{i=1}^M$ .
- Store the observations/communications together  $(((o_1, c_1), \hat{o}_1), \dots, ((o_M, c_M), \hat{o}_M))$  in  $\mathcal{D}$ 4:
- 5: if  $|\mathcal{D}|$  sufficiently large then
- a batch of B multi-agent observations  $\{((o_{1,b}, c_{1,b}), \hat{o}_{1,b}), \ldots, \}$ Sample  $\begin{array}{l} ((o_{M,b},c_{M,b}),\hat{o}_{M,b})\}_{b=1}^{B} \\ \text{for each agent } i=1 \text{ to } M \text{ do} \end{array}$
- 7:
- 8: Action loss:
- 9: Obtain the up-to-date communication actions from each agent, to agent i:  $\hat{c}_{i,b}$  $(\pi_j(o_{j,b},c_{j,b})_{\text{comm},i})_{j\neq i}$  (this can be pre-computed outside the for-loop)
- Obtain the action loss: 10:

$$\text{action loss} = \frac{1}{B} \sum_{b=1}^{B} \ell(\pi^*(\hat{\mathbf{o}}_b)_i, \ \pi_i(\hat{o}_{i,b}, \ \hat{c}_{i,b}))_{\text{action}})$$

where the subscript "action" denotes the physical action (and not the communication

- 11: Communication loss:
- For each other agent j, obtain the up-to-date communication  $\hat{c}_{j,b}$  (similarly to above for 12:  $\hat{c}_{i,b}$ ), which contains agent i's communication action to agent j so we can backprop to agent i's weights ( $\hat{c}_{i,b}$  can pre-computed outside the for-loop just like  $\hat{c}_{i,b}$  above)
- 13: Obtain the communication loss,

$$\text{communication loss} = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{M-1} \sum_{j=1, j \neq i}^{M} \ell(\pi^*(\hat{\mathbf{o}}_b)_j, \ \pi_j(\hat{o}_{j,b}, \ \hat{c}_{j,b})_{\text{action}})$$

where the subscript "action" denotes the physical action (and not the communication action).

- **Update:** 14:
- 15: Update the weights of  $\pi_i$  where the total loss equals the action loss plus the communication loss, to obtain  $\pi_i^{\text{new}}$ .
- 16:
- Set  $\pi_i \leftarrow \pi_i^{\text{new}}$ , for  $i = 1, \dots, M$ . 17:
- 18: end if
- 19: end while

# More detailed descriptions of the environments used in the experiments

#### E.1 Cooperative navigation

The goal of this scenario is to have N agents occupy N landmarks in a 2D plane, and the agents are either homogeneous or heterogeneous. The environment consists of:

• Observations: The (continuous) observations of each agent are the relative positions of other agents, the relative positions of each landmark, and its own velocity. Agents do not have access to other's velocities, and thus each agent only partially observes the environment (aside from not knowing other agents' policies).

• Reward: At each timestep, if  $A_i$  is the *i*th agent, and  $L_j$  the *j*th landmark, then the reward  $r_t$  at time t is,

$$r_t = -\sum_{j=1}^{N} \min \{ ||A_i - L_j|| : i = 1, \dots, N \}$$

This is a sum over each landmark of the minimum agent distance to the landmark. Agents also receive a reward of -1 at each timestep that there is a collision.

• Actions: Each agents' actions are discrete and consist of: up, down, left, right, and do nothing. These actions are acceleration vectors (except do nothing), which the environment will take and simulate the agents' movements using basic physics (i.e. Newton's law).

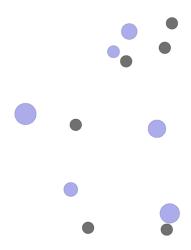


Figure 6: Example of cooperative navigation environment with 6 nonhomogeneous agents. The agents (blue) must decide how best to cover each landmark (grey).

#### E.2 Speaker listener

In this scenario, the goal is for the listener agent to reach a goal landmark, but it does not know which is the goal landmark. Thus it is reliant on the speaker agent to provide the correct goal landmark. The observation of the speaker is just the color of the goal landmark, while the observation of the listener is the relative positions of the landmark. The reward is the distance from the landmark.

- Observations: The observation of the speaker is the goal landmark. The observation of the listener is the communication from the speaker, as well as the relative positions of each goal landmark.
- Reward: The reward is merely the negative (squared) distance from the listener to the goal landmark.
- Actions: The actions of the speaker is just a communication, a 3-dimensional vector. The actions of the listener are the five actions: up, down, left, right, and do nothing.

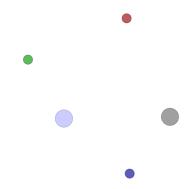


Figure 7: Example of the speaker and listener environment. The speaker (grey) must communicate to the agent which colored landmark to go towards (blue in this case).

# **E.3** Cooperative navigation with communication

In this particular scenario, we have one version with 2 agents and 3 landmarks, and another version with 3 agents and 5 landmarks. Each agent has a goal landmark that is only known by the other agents. Thus the each agent must communicate to the other agents its goal. The environment consists of:

- Observations: The observations of each agent consist of the agent's personal velocity, the relative position of each landmark, the goal landmark for the other agent (an 3-dimensional RGB color value), and a communication observation from the other agent.
- Reward: At each timestep, the reward is the sum of the distances between and agent and its goal landmark.
- Actions: This time, agents have a movement action and a communication action. The movement action consists of either not doing anything, or outputting an acceleration vector of magnitude one in the direction of up, down, left, or right; so do nothing, up, down, left right. The communication action is a one-hot vector; here we choose the communication action to be a 10-dimensional vector.

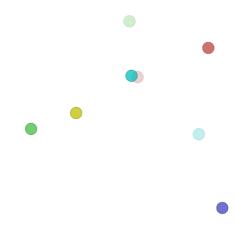


Figure 8: Example of cooperative navigation environment with communication. We have 3 agents and 5 landmarks. The lightly colored circles are agents and they must go towards their same-colored landmark.

# F Hyperparameters

- For all environments, we chose the discount factor  $\gamma$  to be 0.9 for all experiments, as that seemed to benefit both the centralized expert as well as MADDPG (and as well as independently trained DDPG). And we always used a two-hidden-layer neural network for all of MADDPG's actors and critics, as well as the centralized expert, and the decentralized agents. The training of MADDPG used the hyperparameters from the MADDPG paper [20], with the exception of having  $\gamma=0.9$  (instead of 0.95), as that improved MADDPG's performance.
- For the cooperative navigation environments with 3 agents, for both homogeneous and nonhomogeneous: Our centralized expert neural network was a two-hidden-layer neural network with 225 units each (as that matched the number of parameters for MADDPG when choosing 128 as their number of hidden units for each of their 3 agents), and we used a batch size of 64. The learning rate was 0.001, and  $\tau=0.001$ . We also clipped the gradient norms to 0.1. When decentralizing, each agent was a two-hidden-layer neural network with 128 units (as in MADDPG), where we trained with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 128 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.
- For the cooperative navigation with 6 agents, for both homogeneous and nonhomogeneous: Our centralized expert neural network was a two-hidden-layer neural network with 240 units each (as that matched the number of parameters for MADDPG when choosing 128 as their number of hidden units for each of their 3 agents' actor and critic), and we used a batch size of 32. The learning rate was 0.0001, and  $\tau = 0.0001$ . We also clipped the gradient norms to 0.1. When decentralizing, each agent was a two-hidden-layer neural network with 128 units (as in MADDPG), where we trained with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 128 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.
- For the speaker and listener environment: Our centralized expert neural network was a two-hidden-layer neural network with 64 units each (which gave a lower number of parameters than MADDPG when choosing 64 as their number of hidden units for each of their 2 agents' actor and critic), and we used a batch size of 32. The learning rate was 0.0001, and τ = 0.001. When decentralizing, each agent was a two-hidden-layer neural network with 64 units (as in MADDPG), where we trained with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 64 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.
- For the cooperative navigation with communication environment: Our centralized expert neural network was a two-hidden-layer neural network with 95 units each (which matched the number of parameters as MADDPG when choosing 64 as their number of hidden units for each of their 2 agents' actor and critic), and we used a batch size of 32. The learning rate was 0.0001, and  $\tau = 0.0001$ . When decentralizing, each agent was a two-hidden-layer neural network with 64 units (as in MADDPG), where we trained with a batch size of 32 and a learning rate of 0.001. In our experiment comparing with MADDPG, we use the cross entropy loss. The MADDPG and DDPG parameters were 64 hidden units, and we clipped gradients norms at 0.5, with a learning rate of 0.01.
- We also run all the environments for 25 time steps.

#### **G** Proofs of theorems

# **G.1** Mathmetical notation and preliminaries

For completeness, we provide here proofs of the theorems from the main text. The proofs heavily borrow from [34].

In order to reduce notational burden, we denote,

$$\pi_{1,\ldots,M}=(\pi_1,\ldots,\pi_M)$$

so that

$$\pi_{1,\ldots,M}(\mathbf{o}) = (\pi_1(o_1),\ldots,\pi_M(o_M))$$

where  $\mathbf{o} = (o_1, \dots, o_M)$  is the joint multi-agent observation. And we notate  $\hat{\pi}_{1,\dots,M}$  similarly.

We also denote  $d_{\pi_1,\dots,M}^t$  the distribution of states seen at time t, and  $d_{\pi_1,\dots,M} = \frac{1}{T} \sum_{t=1}^T d_{\pi_1,\dots,M}^t$  the average distribution of states encountered if we follow  $\pi_{1,\dots,M}$  for T time-steps.

And letting  $C(\mathbf{o}, \mathbf{a})$  be the cost of executing action  $\mathbf{a}$  in state  $\mathbf{o}$ , and letting  $C_{\pi_{1,...,M}}(\mathbf{o}) = \mathbb{E}_{\mathbf{a} \sim \pi_{1,...,M}(\mathbf{o})} [C(\mathbf{o}, \mathbf{a})]$ , then we let

$$J(\pi_{1,\dots,M}) = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{o} \sim d_{\pi_{1},\dots,M}^{t}} \left[ C_{\pi_{1},\dots,M}(\mathbf{o}) \right] = T \mathbb{E}_{\mathbf{o} \sim d_{\pi_{1},\dots,M}} \left[ C_{\pi_{1},\dots,M}(\mathbf{o}) \right]$$

We use the following lemma which gives a total variation bound:

**Lemma 3.** If  $d_{\hat{\pi}_{1,...,M}^{(i)}}$  is the average distribution of states encountered by  $\hat{\pi}_{1,...,M}^{(i)}$  (the policy that best mimics the expert on past trajectories at iteration i), and if  $d_{\pi_{1,...,M}}$  is the average distribution of states encountered by the policy  $\pi_{1,...,M}^{(i)}$  (which picks expert actions with probability  $\beta_i$  and  $\hat{\pi}_{1,...,M}^{(i)}$  otherwise), then we have  $\|d_{\pi_{1,...,M}^{(i)}} - d_{\hat{\pi}_{1,...,M}^{(i)}}\|_1 \leq 2T\beta_i$ .

*Proof.* Suppose d is the distribution of states encountered over T steps for a policy  $\pi_{1,\dots,M}^{(i)}$  that picks expert actions  $\pi^*$  at least once. Then since  $\pi_{1,\dots,M}^{(i)}$  picks actions from  $\hat{\pi}_{1,\dots,M}^{(i)}$  over T steps with probability  $(1-\beta_i)^T$ , then we have

$$d_{\pi_{1,\dots,M}^{(i)}} = (1 - \beta_i)^T d_{\hat{\pi}_{1,\dots,M}^{(i)}} + (1 - (1 - \beta_i)^T) d$$

Then,

$$\left\| d_{\hat{\pi}_{1,\dots,M}^{(i)}} - d_{\pi_{1,\dots,M}^{(i)}} \right\|_{1} = \left(1 - (1 - \beta_{i})^{T}\right) \left\| d - d_{\hat{\pi}_{1,\dots,M}^{(i)}} \right\|_{1} \le 2(1 - (1 - \beta_{i})^{T}) \le 2T\beta_{i}$$

Now we can prove:

**Theorem 4.** If the number of iterations N is  $\tilde{O}(T)$ , then there exists a policy  $\hat{\pi}_{1,...,M} \in \{\hat{\pi}_{1,...,M}^{(i)}\}_{i=1}^N$  such that  $\mathbb{E}_{\mathbf{0} \sim d_{\hat{\pi}_1,...,M}}[\ell(s,\hat{\pi}_{1,...,M})] \leq \epsilon_N + O(1/T)$  where

$$\epsilon_{N} = \min_{\pi_{1,...,M} \in \Pi_{1,...,M}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{o} \sim d_{\pi_{1,...,M}^{(i)}}} \left[ \ell(\boldsymbol{o}, \pi_{1,...,M}) \right]$$

*Proof.* Let  $\ell_{\max} = \sup_{\mathbf{0}, \pi_{1,...,M}} \ell(\mathbf{0}, \pi_{1,...,M})$ , and let  $\{\beta_i\}_{i=1}^N$  be a non-increasing sequence, and suppose  $n_\beta$  is the largest  $n \leq N$  where  $\beta_n > \frac{1}{T}$ .

Now, the lemma above implies,

$$\mathbb{E}_{\mathbf{0} \sim d_{\hat{\pi}_{1}^{(i)},\dots,M}} \left[ \ell(s,\hat{\pi}_{1,\dots,M}^{(i)}) \right] \leq \mathbb{E}_{\mathbf{0} \sim d_{\hat{\pi}_{1}^{(i)},\dots,M}} \left[ \ell(s,\hat{\pi}_{1,\dots,M}^{(i)}) \right] + 2\ell_{\max} \min(1,T\beta_{i})$$

Then let  $\gamma_N$  be a constant bounding (the no-regret bound),

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\mathbf{o}\sim\pi_{1,...,M}^{(i)}}\left[\ell(\mathbf{o},\pi_{1,...,M}^{(i)})\right]-\min_{\pi_{1,...,M}\in\Pi_{1,...,M}}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\mathbf{o}\sim\pi_{1,...,M}^{(i)}}\left[\ell(\mathbf{o},\pi_{1,...,M})\right]$$

Then we have the following set of inequalities,

$$\begin{split} \min_{\hat{\pi}_{1,...,M} \in \{\hat{\pi}_{1,...,M}^{(i)}\}_{i=1}^{N}} \mathbb{E}_{\mathbf{o} \in d_{\hat{\pi}_{1,...,M}}} \left[ \ell(s,\hat{\pi}_{1,...,M}) \right] &\leq \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\mathbf{o} \sim \hat{\pi}_{1,...,M}^{(i)}} \left[ \ell(\mathbf{o},\hat{\pi}_{1,...,M}^{(i)}) \right] \\ &\leq \frac{1}{N} \sum_{i=1}^{N} \left( \mathbb{E}_{\mathbf{o} \sim \pi_{1,...,M}^{(i)}} \left[ \ell(\mathbf{o},\hat{\pi}_{1,...,M}^{(i)}) \right] + 2\ell_{\max} \min(1,T\beta_{i}) \right) \\ &\leq \gamma_{N} + \frac{2\ell_{\max}}{N} \left[ n_{\beta} + T \sum_{i=n_{\beta}+1}^{N} \beta_{i} \right] \\ &+ \min_{\pi_{1,...,M} \in \Pi_{1,...,M}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\mathbf{o} \sim \pi_{1,...,M}^{(i)}} \left[ \ell(\mathbf{o},\pi_{1,...,M}) \right] \\ &= \gamma_{N} + \epsilon_{N} + \frac{2\ell_{\max}}{N} [n_{\beta} + T \sum_{i=n_{\beta}+1}^{N} \beta_{i}] \end{split}$$

So we have,

**Corollary 5.** If the number of iterations N is  $\tilde{O}(T)$ , then there exists a joint multi-agent policy  $\hat{\pi} \in {\{\hat{\pi}^{(i)}\}_{i=1}^{N}}$  such that

$$J(\hat{\pi}) \le J(\pi^*) + T\epsilon_N + O(1)$$

where,

$$\epsilon_N = \min_{\pi_{1,...,M} \in \Pi_{1,...,M}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{o} \sim d_{\pi_{1,...,M}^{(i)}}} [\ell(\boldsymbol{o}, \pi_{1,...,M})]$$

And now we can prove the Theorem 1 from the main text:

**Theorem 1.** Let  $\ell = \ell(\boldsymbol{o}, \pi_{1,...,M})$  be the 0-1 loss of matching the expert policy. And suppose we have the cost of partial observability,

$$\mathbb{E}_{\boldsymbol{o} \sim d_{\pi_{1},\dots,M}^{(i)}} \left[ \ell(\boldsymbol{o}, \pi_{1,\dots,M}) \right] \geq c_{p}$$

If the number of iterations N is  $\tilde{O}(T)$ , then there exists a joint multi-agent policy  $\hat{\pi} \in {\{\hat{\pi}^{(i)}\}_{i=1}^{N}}$  such that

$$J(\hat{\pi}) \le J(\pi^*) + T\epsilon_N + O(1)$$
  
$$\le J(\pi^*) + Tc_p + O(1)$$

where,

$$\epsilon_N = \min_{(\pi_1, \dots, \pi_M) \in \Pi_1 \times \dots \times \Pi_M} \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{\boldsymbol{o} \sim d_{(\pi_1^{(i)}, \dots, \pi_M^{(i)})}} [\ell(\boldsymbol{o}, (\pi_1, \dots, \pi_M))]$$

*Proof.* The condition on  $c_p$  gives us that,

$$\epsilon_N = \min_{(\pi_1, \dots, \pi_M) \in \Pi_1 \times \dots \times \Pi_M} \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{\mathbf{o} \sim d_{(\pi_1^{(i)}, \dots, \pi_M^{(i)})}} [\ell(\mathbf{o}, (\pi_1, \dots, \pi_M))] \leq \frac{1}{N} N c_p = c_p.$$

And thus we have,

$$J(\hat{\pi}) \le J(\pi^*) + T\epsilon_N + O(1)$$
  
$$\le J(\pi^*) + Tc_p + O(1)$$

And now we prove Theorem 2 from the main text:

**Theorem 2.** If the multi-agent communication  $c_i : \mathcal{O}_1 \times \cdots \times \mathcal{O}_{i-1} \times \mathcal{O}_{i+1} \times \cdots \times \mathcal{O}_M \to \mathcal{C}_i$  satisfies the following condition:

$$\pi^*(o_1, \dots, o_{i-1}, o_i, o_{i+1}, \dots, o_M) \neq \pi^*(o'_1, \dots, o'_{i-1}, o_i, o'_{i+1}, \dots, o'_M)$$
implies that  $c_i(o_1, \dots, o_{i-1}, o_{i+1}, \dots, o_M) \neq c(o'_1, \dots, o'_{i-1}, o'_{i+1}, \dots, o'_M)$ 

for all  $i=1,\ldots,M$ , then there is no partial observability problem of decentralization, and thus if the number of iterations N is of  $\tilde{O}(T)$ , then there exists a policy  $\hat{\pi}_{1,\ldots,M} \in \{\hat{\pi}_{1,\ldots,M}^{(i)}\}_{i=1}^N$  such that  $\mathbb{E}_{\boldsymbol{o} \sim d_{\hat{\pi}_{1,\ldots,M}}} \left[\ell(\boldsymbol{o},\hat{\pi}_{1,\ldots,M})\right] \leq \epsilon_N + O(1/T)$ . Then this implies,  $J(\hat{\pi}_{1,\ldots,M}) \leq J(\pi_{1,\ldots,M}^*) + \epsilon_N T + O(1)$ 

*Proof.* We need only show that the there is no partial observability problem of decentralization when the above conditions hold. As a note: we now have that the observation of agent i is not just  $o_i$ , but now  $(o_i, c_i)$ .

Suppose for contradiction that there is a partial observability problem of decentralization, and thus there exists an agent i, observations  $\mathbf{o} = (o_1, \dots, o_{i-1}, o_i, o_{i+1}, \dots, o_M)$  and  $\mathbf{o}' = (o'_1, \dots, o'_{i-1}, o_i o'_{i+1}, \dots, o'_M)$  such that

$$\pi^*(\mathbf{o})_i = a_i, \quad \pi^*(\mathbf{o}')_i = a_i', \quad a_i \neq a_i'.$$

Denote  $\mathbf{o}_{-i}$  as the observation without  $o_i$  and similarly for  $\mathbf{o}'_{-i}$ . We then see we must have,

$$\pi_i(o_i, c_i(\mathbf{o}_{-i})) = \pi_i(o_i, c_i(\mathbf{o}'_{-i}))$$

for all possible policies  $\pi_i$ . But of course by the above conditions,  $c_i(\mathbf{o}_{-i}) \neq c_i(\mathbf{o}'_{-i})$ , and thus we can certainly construct a policy where the above does not hold true, a contradiction. And thus we must have that there is no partial observability problem of decentralization.

## References

- [1] Daniel S Bernstein, Eric A Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized pomdps. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI)*, pages 52–57, 2005.
- [2] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.
- [3] Lucian Busoniu, Robert Babuska, and Bart De Schutter. Multi-agent reinforcement learning: An overview. 2010.
- [4] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752, 1998.
- [5] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.
- [6] Roel Dobbe, David Fridovich-Keil, and Claire Tomlin. Fully decentralized policies for multi-agent systems: An information theoretic approach. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2941–2950. Curran Associates, Inc., 2017.
- [7] Maxim Egorov. Multi-agent deep reinforcement learning, 2016.
- [8] Richard Evans and Jim Gao. Deepmind ai reduces google data centre cooling bill by 40, 2016.
- [9] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.
- [10] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.
- [11] Jakob N. Foerster, Christian A. Schröder de Witt, Gregory Farquhar, Philip H. S. Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. *CoRR*, abs/1810.11702, 2018.
- [12] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *CoRR*, abs/1705.08926, 2017.
- [13] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.
- [14] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 09 1998.
- [15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [16] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82 94, 2016.
- [17] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.
- [18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
- [19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.
- [20] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.
- [21] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

- [22] L. Matignon, G. J. Laurent, and N. L. Fort-Piat. Hysteretic q-learning :an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 64–69, Oct 2007.
- [23] Laetitia Matignon, Guillaume j. Laurent, and Nadine Le fort piat. Review: Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *Knowl. Eng. Rev.*, 27(1):1–31, feb 2012.
- [24] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. CoRR, abs/1602.01783, 2016.
- [25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [26] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multiagent populations. arXiv preprint arXiv:1703.04908, 2017.
- [27] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.
- [28] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. *CoRR*, abs/1703.06182, 2017.
- [29] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, Nov 2005.
- [30] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.
- [31] James Paulos, Steven W. Chen, Daigo Shishika, and Vijay Kumar. Decentralization of multiagent policies by learning what to communicate. 2018.
- [32] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. CoRR, abs/1803.11485, 2018.
- [33] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.
- [34] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.
- [35] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. Highdimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.
- [36] Felipe Leno Da Silva, Matthew E. Taylor, and Anna Helena Reali Costa. Autonomously reusing knowledge in multiagent reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5487–5493. International Joint Conferences on Artificial Intelligence Organization, 7 2018.
- [37] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.
- [38] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning Volume 32*, ICML'14, pages I–387–I–395. JMLR.org, 2014.
- [39] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

- [40] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.
- [41] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016.
- [42] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Valuedecomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296, 2017.
- [43] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.
- [44] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 2018.
- [45] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.
- [46] Ming Tan. Readings in agents. chapter Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents, pages 487–494. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
- [47] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [48] Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems*, pages 871–878, 2004.
- [49] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. CoRR, abs/1609.02993, 2016.