R-MADDPG for Partially Observable Environments and Limited Communication

Rose E. Wang 1  Michael Everett 2  Jonathan P. How 3

Abstract
There are several real-world tasks that would benefit from applying multiagent reinforcement learning (MARL) algorithms, including the coordination among multiple agents such as self-driving cars or autonomous delivery drones. Real-world conditions are a challenging environment for multiagent systems due to the environment’s partially observable, nonstationary nature. Moreover, if agents must share a limited resource (e.g., communication network bandwidth) they must all learn how to coordinate resource use. These aspects make learning very challenging. This paper introduces a deep recurrent multiagent actor-critic framework for handling multiagent coordination under partial observable settings and limited communication. We investigate the recurrency effects on the performance and communication use of a team of agents, and demonstrate that the resulting framework is capable of learning time-dependencies for not only sharing missing observations but also handling resource limitations. It gives rise to different communication patterns among agents, which still perform equivalently well as current multiagent actor-critic methods under fully observable settings.

1. Introduction
To apply reinforcement learning in real world settings, like collaboration among autonomous vehicles or between humans and machines, we must develop robust frameworks that explicitly address common real world challenges. Much of current RL research makes unrealistic assumptions, like full observability of the environment, one agent learning in isolation, or unlimited access to a communication network, none of which exist in the real world. Therefore, key remaining challenges in RL include learning in domains with: partial observability (agents must learn concise abstractions of history while learning to make good decisions); nonstationarity (introduced by multiple agents learning simultaneously); and limited communication between agents (constraints on sharing of beliefs and intents).

General multiagent reinforcement learning (MARL) methods either assume full observability and are less applicable to real world conditions (Peng et al., 2017; Kong et al., 2017), or handle partially observable environments by making assumptions on the types of policies learned, such as multiple agents developing homogeneous policies (Khan et al., 2018). Earlier works (Wu et al., 2009; Amato et al., 2015) model the multiagent learning problem as decentralized POMDPs (Dec-POMDPs), nonetheless the traditional search for an optimal policy requires knowledge about the transition function which agents typically do not have access to in the real world.

In MARL with communication, previous methods miss important elements of the real world: the architectures are designed specifically for communication and assume network parameter sharing (Foerster et al., 2016) or access to other agents’ hidden states (Singh et al., 2018; Sukhbaatar et al., 2016). Not only are these assumptions unrealistic for real world conditions, enforcing a specific communication architecture can limit the diversity of emergent communication protocols (Kottur et al., 2017). Moreover, since communication aids other objectives, an agent’s verbal policy should be modeled alongside other task policies (like physical motion), not learned separately (Khan et al., 2019).

This work proposes a new architecture for handling multiagent coordination under partially observable environments using only limited communication, and compares the proposed architecture’s performance against alternatives. Our method learns two policies in parallel—one for physical navigation and another for communication—as opposed to learning communication separately as in previously mentioned works. Specifically, we assume a multiagent actor-critic model and propose a model where both the actor and the...
critic are recurrent. This paper refers to the model as the recurrent multiagent deep deterministic policy gradient model (R-MADDPG). Alternative architectures include actor-critic models with only a recurrent actor or only a recurrent critic. Our experiments show that the fully recurrent actor-critic model learns with less variability in its mean and variance out of all architectures and that the recurrent critic is the crucial component that enables learning under real-world conditions (partial observations, limited communication, multiagent). The experiments also suggest a recurrent actor is insufficient by itself for partially observable domains. This work extends upon previous work, Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MADDPG) (Lowe et al., 2017).

Our contributions include: i) a demonstration of the failure of current MARL methods in a simple partially observable coordination task, which identifies a remaining gap between RL research and the real world; ii) the introduction of recurrent multiagent actor-critic architectures, with experiments showing successful learning under various communication and observability constraints; iii) empirical comparison between the proposed architectures that highlights the importance of a recurrent critic; and iv) an open-source implementation of R-MADDPG (to be released prior to workshop).

2. Related Works

Recent work in reinforcement learning has demonstrated successful applications in game playing against real players, both in the single-agent (Mnih et al., 2015; Vinyals et al., 2019) and multiagent (OpenAI) space. However, to apply reinforcement learning to the real world, we should assume real-world conditions, such as partial observability, and allow for explicit communication among teams of agents.

Three key challenges in applying reinforcement learning to real life are: multiagent learning in fully and partially observable environments, multiagent learning for communication and/or communication protocols, and multiagent resource sharing. Most works below handle these challenges separately. This work is the first to handle all three of these challenges in one general framework.

Multiagent learning A well-known issue in multiagent learning is nonstationarity (Hernandez-Leal et al., 2017): Each agent simultaneously updates its policy during training, thus making each agent’s optimal policy a moving target. From the perspective of each of the agents, the updating policies of other agents are additional, unobservable states of the environment, making learning even more difficult. MADDPG (Lowe et al., 2017) combats nonstationarity by training the critic in a centralized manner, as in this work. This work distinguishes between centralized training (sharing experiences during network parameter updates) and communication messages (sharing observations/beliefs during task execution). Several single agent RL works address nonstationarity with experience replay (Mnih et al., 2015; Schaul et al., 2015). However, experience replay in multiagent setting introduces additional challenges, such as how to sample experiences in a synchronized fashion (Omidshafiei et al., 2017), and even conflicting information as to whether experience replay is helpful in multiagent settings (Foerster et al., 2016; Singh et al., 2018).

Communication and resource sharing MADDPPG’s method (Lowe et al., 2017) can handle cooperative tasks, however does not model explicit communication among agents and cannot handle partially observable environments and history-dependent decision making. (Khan et al., 2018) is similar to MADDPG, however is capable of scaling to more agents under the strong assumption that the agents’ policies can be approximated to a single policy. BiCNET (Peng et al., 2017) proposes an actor-critic model to learn long-term sequential strategies, however solves fully observable MDP problems because it conditions on complete information over the environment state. (Jiang & Lu, 2018) is similar to our learning environment, in that they want to learn how to conservatively use communication. They propose a central attentional unit in an actor-critic framework for learning when communication is needed and for integrating shared information. Nonetheless, they prioritize minimizing communication as much as possible, whereas this paper demonstrates that agents are capable limiting communication and adapting to any amount of resources.

CommNet (Sukhbaatar et al., 2016) and IC3NET (Singh et al., 2018) introduce a communication-specific network architecture for determining when to communicate, nonetheless relies on complete access to all agents’ hidden states for determining an agent’s communication action; thus agents execute communication actions in a centralized manner since their decision depends on the hidden states of others. RIAL and DIAL (de Freitas, 2016) are designed as communication-specific architectures that assume end-to-end differentiable communication between pairs of agents and parameter sharing across different agents. Their task setting only include tasks with verbal coordination, and not with parallel task coordination.

3. Background

3.1. Reinforcement Learning

In real world settings, agents make noisy observations of the true environment state to inform their action selection, typically modeled as a Partially Observable Markov decision process (POMDPs) (Kaelbling et al., 1998), or in its extended version with multiple agents, a Decentralized Par-
tially Observable Markov decision process (Dec-POMDPs) (Bernstein et al., 2002) defined as \((\mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R}, \gamma)\), where \(\mathcal{I} = \{1, \ldots, N\}\) is the set of \(N\) agents, \(\mathcal{S}\) is the set of states, \(\mathcal{A} = \times_i \mathcal{A}_i\) is the set of joint actions, \(\mathcal{T}\) is the transition probability function, \(\Omega = \times_i \mathcal{O}_i\) is the set of joint partial observations, \(\mathcal{O}\) is the observation probability function, \(\mathcal{R}\) is the reward function, and \(\gamma \in (0, 1)\) is the discount factor. At each timestep \(t\), agent \(i\) receives a partial observation \(o_i^t\) and takes action \(a_i^t\) according to policy \(\pi^i(h_i^t; \theta^i)\), where \(\theta^i\) is agent \(i\)'s policy parameters and \(h_i^t\) is agent \(i\)'s observation history. The current state of the Dec-POMDP \(s_t\) transitions to \(s_{t+1}\) according to the transition function with joint actions of the agents \(a_t = a_1^t \times \ldots \times a_N^t\), i.e. \(\mathcal{T}(s_{t+1}; s_t, a_t)\). The agents receive a shared team reward \(r_t = \mathcal{R}(s_t, a_t)\), and receive a new joint observation set \(o_{t+1} = \{o_{1,t+1}^t, \ldots, o_{N,t+1}^t\}\) after the state transition. The objective for each agent is to maximize its expected discounted reward \(\mathbb{E}[\sum_t r_t \gamma^t]\).

This work focuses on using recurrent neural networks for learning representations capable of estimating the true state of the Dec-POMDP \(S\) from an agent’s local set of observations \(\Omega_i\). The recurrency in the network architecture therefore explicitly acts as a system mechanism for gathering partial observations so as to minimize the differences in system behavior with and without full observability of \(S\).

### 3.2. Q-learning

Q-learning and Deep Q-learning methods have been very popular in the context of Atari game playing. Q-learning is a model-free approach for determining the long-term expected return of executing an action \(a\) from a state \(s\), where it makes use of the action-value function under a given policy \(\pi\) (Sutton et al., 1998). In other words, Q is iteratively defined as,

\[
Q_\pi(s, a) = \mathbb{E}_{s', a', r} [r(s, a) + \gamma \mathbb{E}_{a'' \sim \pi} [Q_\pi(s', a')]].
\]

Deep Q-Learning methods approximate the Q-values by means of a neural network parameterized by the weight \(\theta\). It learns the values for \(Q^*\), where \(\bar{Q}^*\) is the target values, by minimizing the loss defined as:

\[
\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(Q^*(s, a; \theta) - (r + \gamma \max_{a'} \bar{Q}^*(s', a')))^2].
\]

Because the same network is used for generating next target values and for updating \(Q^*\), Deep Q-Learning demonstrates high variance in its learning trajectory for approximating action values. Thus, common techniques for facilitating learning stability include using experience replay (Mnih et al., 2015; Schaul et al., 2015) in a replay memory buffer sampled during training, and using a separate, target network \(\bar{Q}\) for generating the target values in the loss calculation. This target network is identical to the \(Q^*\) except that the target network is updated to match \(Q^*\) at a much slower rate (e.g. every thousand iterations) so as to stabilize the learning of \(Q^*\).

### 3.3. Policy Gradient Algorithms

Policy gradient methods are another way for maximizing expected reward for the agent by directly optimizing the policy. The policy is parameterized by weights \(\theta\). The objective is to maximize the score function

\[
J(\theta) = \mathbb{E}_{\pi_\theta} \left[ \sum_t R_t \right]
\]

where the gradient of the policy is defined by the Policy Gradient Theorem (Sutton et al., 2000) as:

\[
\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta (a \mid s) Q_\pi(s, a) \right].
\]

This paper uses the actor-critic framework, where a network, namely the critic, learns the approximation of \(Q_\pi(s, a)\) by temporal difference learning. To handle non-stationarity in the multiagent framework (Lowe et al., 2017), each agent’s critic uses all agents’ observations and actions for training. Thus, the loss with respect to agent \(i\)’s policy parameterization is:

\[
\nabla_\theta J = \mathbb{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta (a \mid o) Q_\pi(o_1, \ldots, o_N, a_1, \ldots, a_N) \right].
\]

### 4. Methods

This paper proposes three recurrent multiagent actor-critic models for partially observable and limited communication settings. The models only take in a single frame at each timestep. Because they cannot communicate all the time, they need a way to remember the last communication they received from their team, when they last transmitted a message and how their actions affect the communication budget over time. Recurrency acts as an explicit mechanism to do just that. Our models extend the multiagent actor-critic framework proposed by MADDPG to enable learning in a multiagent, partially observable, and limited communication domain.

#### 4.1. Recurrent Multi Agent Actors

We perform the following updates using experience sampled \(\sim U(D)\). An agent \(i\)'s replay buffer \(D\) contains tuples of experiences, where an experience at time \(t\) contains:

\(\pi_{i,t}, a_{i,t}, a'_{i,t+1}, r_{i,t}, h^p_{i,t}, h^D_{i,t+1}\). \(o\) denotes agent \(i\)'s partial observations, \(a\) its action resulting from \(\pi_{i,t}(a_{i,t}, h_{i,t})\), \(r_{i,t}\) the agent’s reward, and \(h^p\) the hidden state of the actor network before and after the selected action. Let each agent have a continuous policy \(\mu = \mu_\theta\), and a target policy \(\mu' = \mu'_{\theta'}\). \(x \sim U(D)\) and \(a \sim U(D)\) are placeholders for
The action-value function \( Q^\pi_i \) is updated based on,
\[
\nabla_{\theta_i} J(\mu) = \mathbb{E}_{\sim U(D)} [\nabla_{\theta_i} \mu(a_{i,t}|o_{i,t}) \nabla_{a_{i,t}} Q^\pi_i(x,a,h^q_{i,t+1}) | a_{i,t} = \mu_i(o_{i,t})].
\]

The action-value function \( Q^\pi_i \) is updated based on,
\[
\nabla_{\theta_i} J(\mu) = \mathbb{E}_{\sim U(D)} [\nabla_{\theta_i} \mu(a_{i,t}|o_{i,t}, h^p_{i,t+1}) \nabla_{a_{i,t}} Q^\pi_i(x,a) | a_{i,t} = \mu_i(o_{i,t}, h^p_{i,t+1})].
\]

4.3. Recurrent Multi Agent Actors and Critics

We perform the following updates using experience sampled \( \sim U(D) \). An agent \( i \)'s replay buffer \( D \) tuples of experiences, where an experience at time \( t \) contains \( (o_{i,t}, a_{i,t}, r_{i,t}, h^q_{i,t}, h^q_{i,t+1}, h^p_{i,t}, h^p_{i,t+1}) \). We assume the same notation from before. The policy gradient is calculated as,
\[
\nabla_{\theta_i} J(\mu) = \mathbb{E}_{\sim U(D)} [\nabla_{\theta_i} \mu(a_{i,t} | o_{i,t}, h^p_{i,t+1}) \nabla_{a_{i,t}} Q^\pi_i(x,a,h^p_{i,t+1}) | a_{i,t} = \mu_i(o_{i,t}, h^p_{i,t+1})].
\]

The action-value function \( Q^\pi_i \) is updated based on,
\[
\nabla_{\theta_i} J(\mu) = \mathbb{E}_{\sim U(D)} [\nabla_{\theta_i} \mu(a_{i,t} | o_{i,t}, h^p_{i,t+1}) - Q^\pi_i(x,a,h^p_{i,t+1})^2].
\]

5. Experiments

This section shows that the recurrent critic is critical for agents to learn a good policy from their partially observable states and under limited communication settings. The recurrent actor alone is not able to discover the right policy, however combined with the recurrent critic it reduces the variance in the reward performance. Our experiments are based on a simultaneous arrival task; a navigation task (using OpenAI’s multiagent particle environments; an example video can be found here), where \( N \) agents must arrive at a goal location at the same time. In the fully observable environment, the agents know the positions of all the agents and the goal. In the partially observable environment, the agents only know of their position and the goal; only if an agent decides to communicate, do other agents know its position. The partially observable domain is especially difficult for MADDPG because it is unable to keep a history of its previous partial observations; this renders it almost impossible for MADDPG to estimate the underlying system state.

This environment allows us to focus on the analysis of time-wise coordination among agents and multi-timestep communication use under different recurrent architectures. We
investigate the effects of recurrency between MADDPG and R-MADDPG with the experiments below. For all the experiments we compare among regular MADDPG and these proposed networks from the Methods section. We both vary the observability (between full and partial observations) for the agents and vary the communication budget.

5.1. Experimental Setup

Let $s$ denote an agent’s fully observable state, containing this agent’s position, $(p_x, p_y)$, the goal position $(g_x, g_y)$, the communication message, $m$, is always the other agent’s position, and a communication budget $c$. A partially observable state, $s'$, contains the same state variables, however the communication message, $m'$ is either the other agent’s position if the other agent communicated that timestep, or $(-1, -1)$ otherwise. That is,

$$s = [p_x, p_y, g_x, g_y, m, c]$$

$$s' = [p_x, p_y, g_x, g_y, m', c]$$

At each time step, each agent selects two types of discrete actions, one physical, $a^p \in \{\text{none, north, east, west, south}\}$, and one verbal, $a^v \in \{\text{communicate, silent}\}$.

The joint team reward function,

$$R = \sum_{i} d(p_i, g) + \sum_{\text{pairs}(i,j)} |d(p_i, g) - d(p_j, g)|,$$

encourages agents to individually reach the common goal position $g = (g_x, g_y)$ through $R_{\text{dist}}$, and encourages simultaneity through $R_{\text{dij}}$, where $d$ is Euclidean distance. Throughout this section, we refer to $R_{\text{dist}}$ as the team distance and $R_{\text{dij}}$ as the difference in agents’ distances to goal. These measurements are used in evaluating performance respectively on the left and right hand columns of Figure 3 and Figure 4.

The communication budget is shared between agents, and a full communication budget is consider to be 1.0. If the communication budget is set to sending $x$ (total) messages, then the budget decreases by $\frac{x}{2}$ with every communication message. If no budget is given, i.e. no communication is allowed, the budget is set to 0.0. No agent is allowed to communicate once the budget reaches 0.0, and their messages are defaulted to a blank value $(-1, -1)$.

**Network architecture:** The networks contains three layers each with 64 units, where the first and last are fully connected layers and the middle layer is an LSTM layer. The first fully connected layer has an ReLU activation (Nair & Hinton, 2010).

**Hyperparameters:** The experiments assume an Adam Optimizer with a learning rate of 0.01, $\tau = 0.01$ for the target network updates, and $\gamma = 0.95$. The replay buffer size is $10^6$. We sample after every other 100 timesteps, and sample a batch size of 256 by episode. Training happens with 4 random seeds for all the experiments found above. All the hyperparameters will be set as the default in the open-source implementation of R-MADDPG, which will be released prior to the workshop.

5.2. Results

5.2.1. Observability

This section first explores whether the models are capable of learning in multiagent environments assuming complete observations, then learning in multi-agent environments assuming partial observations.

The experiments verify that both MADDPG and R-MADDPG variants perform equivalently well under fully observable settings in going to the goal (Figure 3a). R-MADDPG (in green, Figure 3b) does not converge as quickly in arriving simultaneously, and we hypothesize this is because it takes longer to learn if backpropagating through time in both the actor and critic.

Under partially observable settings, the experiments illustrate the importance of the recurrent critic for learning a policy from partial observations and under a limited communication budget that, at minimum requirement, moves the agents towards the goal (Figure 3c, Figure 3d). Furthermore, the figures illustrate that the recurrent actor and critic learns more stably than only the recurrent actor model; we define stable learning by the reward mean fluctuations and the reward variance, where these experiments assumed the same experiment stochasticity as described in 5.1.

The experiments demonstrate that recurrent actor by itself performs similarly to MADDPG. It is unable to learn from a sequence of partial observations, not only how to simultaneously arrive at the goal (Figure 3d), but to even to go
to the goal (Figure 3c). In other words, the recurrent actor provides insufficient information about the underlying task. We hypothesize this is because the the actor optimizes with respect to the critic and sampled actions; without a critic that understands the partial observable dynamics of the environment, the actor converges to a poor policy.

5.2.2. Communication Budget

This section investigates how well the models perform under different resource constraints by varying the communication budget shared by the agents. The communication budget dictates how many messages are allowed to be sent within a team of agents. We still assume the agents are in a partially observable environment, thus the agents must share information in order to arrive simultaneously at the goal. Video examples of R-MADDG, which illustrate the communication use and physical movements of the agents, can be found here.

The experiments verify that the poor performance seen in Figure 3c and Figure 3d by MADDPG is due to the insufficient communication budget which prevents MADDPG from having complete observations over the environment at every timestep. Figure 4 fixes the best performing model from Figure 3c and Figure 3d, namely R-MADDG, and uses it as the best performing model. The graph increases the communication budget for MADDPG up to 200 messages, which means that every agent is allowed to communicate at each timestep of the episode. Only when this happens does the model’s performance closely match R-MADDG’s performance under partially observable conditions.

The paper also examines how communication is being used during an episode Figure 5, which identifies emergent coordination-communication behaviors that do not come across in the plotted aggregate statistics. The video format can be found on the paper’s website. Notably the scenarios illustrate that agents especially prefer to communicate closer to the goal and at the start of an episode. They tend to omit communicating on the way to the goal if initialized further away.

The experiments also vary the communication budget on R-MADDG to evaluate how adaptive and/or sensitive it is to increasing amounts of partial observability. We decrease the communication budget Figure 6 and note that agents can still learn how to move to the goal under decreased partial observability Figure 6a. Due to the reward scaling, a subtlety not quite well reflected in Figure 6b is that the agents fail to simultaneously arrive (the coordination task) with decreasing communication. We hypothesize this is because 1) reward signal $R_{diff}$ is dominated by $R_{dist}$, and 2) agents do need to communicate more in order to coordinate simultaneous arrival. If both agents decide not to communicate for longer periods of time there is more uncertainty about where the other agents are, whether they’ve directly gone to the goal, or whether they’re waiting for the agent some distance away from the goal. In cases of lower budget, this uncertainty results in one agent going directly to the goal, and the other waiting close to the goal.

6. Conclusions and Future Work

This paper proposes a recurrent multi-agent actor-critic model for coordination in partially observable, limited communication settings. This model is more applicable to real-world conditions since real-world settings are multiagent, partially observable and limited in communication. The experiments showed the recurrent critic is important for enabling R-MADDG to handle partially observable environments. They also showed shown that R-MADDG is capable of enabling coordination among agents in arriving simultaneously while varying the communication budget.

As future work, we hope to develop create more multi-agent coordination and communication scenarios and evaluate R-MADDG in more other environments, such as the environments used in (Mordatch & Abbeel, 2017) and DeepMind’s soccer environment from (Liu et al., 2019).

We also hope to expand it to coordination among heterogeneous agents and explore the effects of the replay buffer parameters/settings in the multi-agent environments.

7. Acknowledgments

The authors would like to thank Dong-Ki Kim and Macheng Shen for their feedback on the paper’s draft and interesting discussions.

References


Foerster, J. N., Assael, Y. M., de Freitas, N., and White-
Figure 3. Reward performance in observability experiments. Under fully observable settings (top row), both MADDPG (red) and recurrent variants (green, blue, orange) perform similarly. Under partially observable (bottom row) settings, the recurrent actor (orange) and MADDPG (red) are unable to learn how to simultaneously arrive (d), and even how to move towards the goal (c). This demonstrates the importance of the recurrent critic in partially observable settings. For partial observability, the communication budget is set to 20 messages, shared between 2 agents over $\sim 100$ timesteps per episode.
Figure 4. MADDPG’s performance depends on the degree of observability. Decreasing the communication budget dramatically worsens MADDPG’s performance in partially observable domain. These plots assume each episode is 100 timesteps. Thus, a shared communication budget of 200 messages means that both agents are able to communicate at every timestep during an episode. Yet, even with a 200 message budget that could enable full observability, MADDPG still performs worse than R-MADDPG which uses only 10% of the budget.

Figure 5. Example scenarios assuming R-MADDPG and shared communication budget of 50 messages (agents can only communicate \( \sim 25\% \) of timesteps). Here is a video including these examples. Agents prefer to communicate closer to the goal and at the start of an episode. They tend to omit communicating on the way to the goal if initialized further away. The agent initialized closer to the goal tends to move away from the goal and wait for the farther agent until they can simultaneously go to the goal.


Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay, 2015.


