Recurrent Multiagent Deep Deterministic Policy Gradient with Difference Rewards

Yathartha Tuladhar **Enna Sachdeva** **Manish Saroya**

1 Abstract

Deep Reinforcement Learning (DRL) algorithms have been successfully applied to a range of challenging simulated continuous control single agent tasks. These methods have further been extended to multiagent domains in cooperative, competitive or mixed environments. This paper primarily focuses on multiagent cooperative settings which can be modeled for several real world problems such as coordination of autonomous vehicles and warehouse robots. However, these systems suffer from several challenges such as, structural credit assignment and partial observability. In this paper, we propose Recurrent Multiagent Deep Deterministic Policy Gradient (RMADDPG) algorithm which extends Multiagent Deep Determinisitic Policy Gradient algorithm - MADDPG [5] by using a recurrent neural network for the actor policy. This helps to address partial observability by maintaining a sequence of past observations which networks learn to preserve in order to solve the POMDP. In addition, we use reward shaping through difference rewards to address structural credit assignment in a partially observed environment. We evaluate the performance of MADDPG and R-MADDPG with and without reward shaping in a Multiagent Particle Environment. We further show that reward shaped RMADDPG outperforms the baseline algorithm MADDPG in a partially observable environmental setting.

2 Introduction

Reinforcement learning (RL) has recently been applied to solve challenging problems in single agent domains, from game playing [7, 8], [10] to robotics [2, 4], where modelling or predicting the behaviour of other agents in the environment is largely unnecessary. However, there are a number of important applications that involve interaction between multiple agents, where emergent behavior and complexity arise from agents co-evolving together. Successfully scaling RL to multiple agent systems is crucial to building artificially intelligent systems that can productively interact with each other as well as humans. However, learning in multiagent systems leads several core challenges. Firstly, agents trying to learn a joint set of policies perceive the environment as non-stationary due to other agents trying to learn and adapt concurrently and the behaviour of other agents are changing in unpredictable ways. Further, multiple concurrently exploring agents adds noise to the agents' reward signal.

Secondly, with multiple agents in the environment, a particular agent's contribution to the global reward is dwarfed by the contribution of other agents, thereby leading to low signal to noise ratio [1]. This problem is known as structural credit assignment.

Furthermore, many real-world multiagent problems are partially observed which may be due to different sources including the need to remember information that is temporarily available, such as sensor limitations or noise, unobserved variations of plant under control, or state-aliasing due to function approximation. Partial observability also arises when the environment does not provide information about velocities, or due to bandwidth limitation of sensors. It then makes the states as non-markovian because the future game states (and rewards) not only depends on the agent's current state but previous states as well. This can be called as Partially-Observable Markov Decision Process

Preprint. Work in progress.

(POMDP). The efficient learners must learn to extract information from past observations in order to to accelerate learning and improve generalization to new tasks, to solve POMDPs.

Recently, Multiagent Deep-Deterministic Policy Gradient Algorithm (MADDPG) [5] has been introduced, which extends the actor-critic model for mixed cooperative-competitive environments. It adopts framework of centralized training with decentralized execution, allowing the policies to use extra information to ease training, so long as this information is not used at test time. The critic is augmented with extra information about the policies of other agents during training, while the actor only has access to local information. The primary motivation behind MADDPG is that, if the actions taken by all agents are known (to critic during training), the environment becomes stationary even when the policies of other agents change. The centralized critic will also help in assigning credit to each of the agent's actor policy. After training is completed, only the local observation information is used by each actor during execution phase, acting in a decentralized manner.

However, MADDPG is limited in the sense that it learns a mapping from a snapshot of states. Thus, it will be unable to master scenarios that require the agents to remember events from several time steps in the past. Learning with a sequence of past observations could help in multiagent cooperative and competitive scenarios because it provides more information about the behaviour of other agents. We hypothesize that MADDPG may be modified to better deal with POMDPs by leveraging advances in Recurrent Neural Networks (RNN).

Thus, the first contribution of this paper is the introduction of a Recurrent Neural Network in the actor's policy in MADDPG to address partial observability. We build our approach on top of MADDPG as it has been shown to outperform the state-of-the art single-agent algorithms that were applied to multiagent domain, and thus is considered as a baseline for evaluating multiagent algorithms [5]. We call our algorithm, Recurrent Multiagent Deep-Deterministic Policy Gradient Algorithm (R-MADDPG). Our second contribution is to use difference rewards in the R-MADDPG framework to achieve better credit assignment. Reward shaping using difference rewards (D) [9], has been shown to address structural credit assignment problem by assigning credit to each agent based on its contribution to system's performance, while reducing agent noise, and providing a more learnable signal compared to the global reward G. We evaluated R-MADDPG in the Multiagent Particle Environment [5], and compare it with MADDPG [5]. We also show the results for R-MADDPG and MADDPG when using a difference reward.

The rest of this paper is organized as follows. In Section 3, we provide the background details for Partially Observable Markov Decision Process (POMDP), Deep Deterministic Policy Gradient (DDPG), Multiagent Deep Deterministic Policy Gradient (DDPG), Reward shaping, and Recurrent Neural Network (RNN). Further, Section 4 describeS our methodology (R-MADDPG) along with difference rewards for reward shaping, in detail. in Section 5, we discuss the experimental setup and results and observations obtained on implementing our proposed approaches in multiagent cooperative environment. Section 6 discusses conclusions and future work.

3 Background

We consider a fully cooperative multiagent task environment in which N agents, each identified by $i \in \{1, ..., N\}$, coordinate together to spread and capture L landmarks/Points of Interests (POIs) $l \in \{1, ..., L\}$. The environment has a true state $s \in S$. At each time step, each agent simultaneously chooses an action $u^i \in U$, forming a joint action $u^n \in U^n$. All agents share the same reward function $r(s, u) : SXU \to \mathbb{R}$. The framework used for this settings is a Multiagent Particle Environment by OpenA, consisting of N agents and L landmarks inhabiting a two-dimensional world, as shown in Fig. 3. We consider a partially observable setting, in which agents draw observations $z \in Z$ according to the observation function $O(s, a) : SXA \to Z$. Each agent has an action-observation history $h_i \in H \equiv (ZXU)$.

3.1 Partially Observable Markov Decision Process

A partially observable Markov decision process (POMDP) is a generalization of a Markov decision process (MDP) in which it is assumed that the system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state. Instead, the agent maintains a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP. Therefore, the agent must make decisions under uncertainty of the true environment state.

At each time step t, the agent receives a state s_t and produces an action a_t using its policy π. The agent receives a scalar reward r_t and transitions to the next state s_{t+1}. This process continues until the agent reaches a terminal state marking the end of an episode. The return $R_t = \sum_{n=1}^{\infty} \gamma^k r_{t+k}$ is the total accumulated return from time step t with discount factor $\gamma \in (0, 1]$. The goal of the agent is to maximize the expected return. The state-value function $\mathcal{Q}^{\pi}(s, a)$ describes the expected return from state s after taking action a and subsequently following policy π.

3.2 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) [4] is a popular model-free RL algorithm for learning in continuous high dimensional actions spaces. DDPG uses an actor-critic architecture [11] maintaining a deterministic policy (actor) $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and an action-value function approximation (critic) $\mathcal{Q} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The critic approximates the actor's action-value function $\mathcal{Q}^{\pi}$. Both the actor and the critic are parameterized by (deep) neural networks with θ^{π} and $\theta^{\mathcal{Q}}$, respectively. A separate copy of the actor π' and critic $\mathcal{Q}'$ networks are kept as target networks for stability. These target networks parameters are updated periodically to match the actor π and critic networks $\mathcal{Q}$, modulated by a weighting factor τ.

A noisy version of the policy: $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$ where $\mathcal{N}$ is temporally correlated noise generated using the Ornstein-Uhlenbeck process [15] is used for the training purpose. After each action, the tuple (s_t, a_t, r_t, s_{t+1}) containing the current state, actor's action, observed reward and the next state, respectively is saved into a **cyclic replay buffer** $\mathcal{R}$. The actor and critic networks perform a parameter update by randomly sampling mini-batches from $\mathcal{R}$. The critic is trained by minimizing the loss function:

$$L = \frac{1}{T} \sum_i (y_i - \mathcal{Q}(s_i, a_i | \theta^{\mathcal{Q}}))^2 \text{ where } y_i = r_i + \gamma \mathcal{Q}'(s_{i+1}, \pi'(s_{i+1} | \theta^{\pi'}) | \theta^{\mathcal{Q}'})$$

The actor is trained using the sampled policy gradient:

$$\nabla_{\theta^{\pi}} J \sim \frac{1}{T} \sum \nabla_a \mathcal{Q}(s, a | \theta^{\mathcal{Q}})|_{s=s_i, a=a_i} \nabla_{\theta^{\pi}} \pi(s | \theta^{\pi})|_{s=s_i}$$

The sampled policy gradient with respect to the actor's parameters θ^{π} is computed by backpropagation through the combined actor and critic network.

3.3 Multiagent Deep Determinsitic Policy Gradient

MADDPG is a simple extension of DDPG [10], where agents learn a centralized critic based on the observations and actions of all agents. This information is usually easily available at the training time, information from the simulator can be gathered easily during training phase. Since learning is centralised, the centralised critic conditions on the joint action and all available state information. However, due to the limitation of accessibility of the observations and states information of all agents in the environment, it considers decentralized execution where each agent's policy conditions only on its own local observation.

As the centralized critic explicitly uses the information of policies of other agents, this transforms the non-stationary environment to a stationary one, even if the policies of other agents change, since $P(s'|s, a_1, ..., a_N, \pi_1, ..., \pi_N) = P(s'|s, a_1, ..., a_N) = P(s'|s, a_1, ..., a_N, \pi'_1, ..., \pi'_N)$ for any $\pi_i \neq \pi'_i$. The method is briefly described as-

Consider N agents with continuous policies parameterized by $\mu_\theta = \mu_{\theta 1}, \mu_{\theta 2}, ..., \mu_{\theta N}$. Therefore, the expected return and its gradient for agent i, are given as J_{θ_i} and $\nabla_{\theta_i} J(\theta_i)$, respectively below-

$$J_{\theta_i} = \mathbb{E}[R_i]$$

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,a\ D}[\nabla_{\theta_i} \mu_i(a_i/o_i) \nabla_{a_i} l Q_i^{\mu}(x, a_1, ...a_N)|_{a_i=\mu_i(o_i)}]$$

$Q_i^{\pi}(x, a_1, ...a_N)$ is a centralized action-value function that takes as input the actions of all agents, $a_1, ..., a_N$, in addition to some state information x and outputs the Q-value of agent i. x could consist of observations of all agents or the additional state information, if available.

The replay buffer D contains the tuples $(x, x', a_1, ..., a_N, r_1, ..., r_N)$ recording experiences of all agents.

The action-value function Q_i^μ is updated as-

$$\mathcal{L}(\theta_i) = \mathbb{E}_{x,a,r,x'}[(Q_i^\mu(x, a_1, a_2, ..., a_N) - y)^2], \\ y = r_i + \gamma Q_i^{\mu'}(x', a'_1, ..., a'_N)|_{a'_j = \mu'_j(o_j)}$$

where, $\mu' = \mu'_{\theta_1}, ..., \mu'_{\theta_N}$ is the set of target policies with delayed parameters θ'_i.

Since, Q_i^μ is learned separately, agents can have different reward structures, such as conflicting rewards in a competitive setting. Therefore, the algorithm can be scaled to mixed cooperative-competitive settings very well.

3.4 Recurrent Neural Networks

Recurrent Neural Networks are useful when the decision making gets benefit from temporal information. In an application such as predicting the next word in a sentence, the next word has a very high correlation with the sequence of words that came before it. RNNs maintain a hidden state, also known as “memory”, which can capture information about its previous observations and actions. In the Multiagent Particle Environment setting, an RNN can help capture information about it’s previous observations and make a better decision when the environment is partially observable.

3.5 Related Work

The simplest approach to learning in multiagent settings such as the multiagent particle environment is to use independently learning agents. This was attempted with Q-learning in [13] but does not perform well in practice [6]. One issue is that each agent’s policy changes during training, resulting in a non-stationary environment and preventing the naïve application of experience replay. Previous work has attempted to address this by inputting other agent’s policy parameters to the Q function [14], explicitly adding the iteration index to the replay buffer, or using importance sampling [16]. Deep Q-learning approaches have previously been investigated in [12] to train competing Pong agents. Policy gradient methods with centralized critics have been explored, where there is either a single centralized critic [3] or a centralized critic for each agent [5]. Using difference rewards for credit assignment in the multiagent scenario has been explored in [9].

4 Methods

This section discusses our proposed algorithm: Recurrent Multiagent Deep Deterministic Policy Gradient (R-MADDPG) algorithm, and the use of difference rewards.

4.1 R-MADDPG: Recurrent Multiagent Deep Deterministic Policy Gradient

In order to address partial observability, we extend MADDPG using recurrent neural networks trained with backpropogation through time. In order to solve POMDP, the network learns to preserve the limited information about the past, using RNN. RNN captures the history of k observations and actions from current time step t to time step $t - k$ (where k is a hyper-parameter specified by the user and is domain dependent). The architecture of a simple RNN used for this research is described in Fig. 1, where s_t is the current hidden state, and h_t is the current observation. Mathematically, the hidden state is calculated as $s_t = f(Uh_t + Ws_{t-1})$. Each observation and the current "memory" are passed through feed-forward neural networks U and W respectively; they are then summed and passed through a non-linear function to create the final output action. This action is then used to take a step in the environment.

The architecture of our proposed algorithm R-MADDPG, is shown in Fig. 2, where the actors take actions based on the output generated by RNN, based on information of past states and observations.

Once we have the state, action, next-state, and reward based on the current policy, the actor and critic values are updated as following:

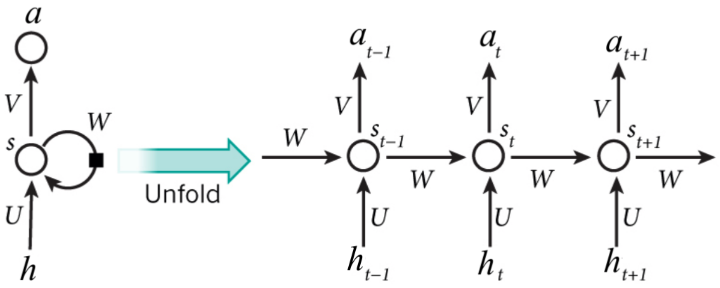

Figure 1: A simple Recurrent Neural Network structure used in the actor policy for Recurrent Multiagent Deep Deterministic Policy Gradient (R-MADDPG) algorithm. The observation history is h, and t represents the current time-step, s represents the current memory state, and a represents the action output.

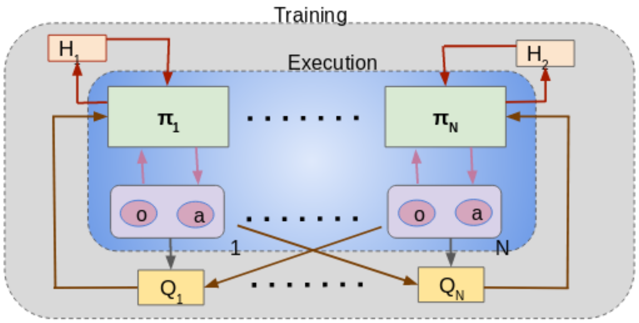

Figure 2: Architecture of the Proposed R-MADDPG where H_1 represents the RNN block for actor-1

Update critic by minimizing the loss:

$$\mathcal{L}(\theta_i) = \frac{1}{s}\sum_t (y - Q_i^{\mu}(h_i^t, a_i|\theta^{\mathcal{Q}}))^2$$

Update actor using the samples policy gradient:

$$\nabla_{\theta_i} J(\mu_i) \approx \frac{1}{S}\sum_t \nabla_{\theta_i}\mu_i(h_i^t)\nabla_{a_i}Q_i^{\mu}(h_i^t, a_t^i)|_{a_i=\mu_i(h_i^t)}]$$

where h_i^{t+1} is the history of sates and observations of agent i from time t to $t-k$ The complete algorithm of R-MADDPG is discussed in Algorithm 1.

4.2 Reward Shaping

In order to address structural credit assignment problem in a multiagent settings, we extend MADDPG and R-MADDPG with reward shaping using difference rewards, in which each agent learns from the shaped reward rather than a global reward. Difference rewards are a powerful way to perform multiagent credit assignment, as they are more informative than the global reward because the second term in D removes much of the effect of other agents from agent $i's$ reward, and highlights each agent's contribution to the global performance. The second term does not depend on agent $i's$ actions, thus agents receive positive rewards for benefiting the system and negative for harming the system.

$$D_i = G(z) - G(z_{-i} + c_i)$$

where, $G(z)$: global reward received,
$G(z_{-i} + c_i)$: the system reward that would have been received if action of agent i is replaced by a default counterfactual action c_i. This causes the second term to be independent of i, and therefore D_i evaluates the agent $i's$ contribution directly to the global performance.

Algorithm 1 Recurrent Multiagent Deep Deterministic Policy Gradient (for N-agents)

```
for episode = 1 to M do
    Initialize a random process N for action exploration
    Receive initial state $x$
    for t = 1 to max-episode-length do
        for each agent $i$, select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}$ w.r.t the current policy and exploration
        Execute actions $a = (a_1, a_2, ..., a_N$ and observe reward $r$ and new state $x'$
        Store $(x, a, r, x')$ in replay buffer D
        $x \leftarrow x'$
        for agent $i = 1$ to $N$ do
            Sample a random minibatch of $S$ samples $(x^j, a^j, r^j, x'^j)$ from $\mathcal{D}$
            Construct histories $h_i^t = (o_t^i, o_{t-1}^i, a_{t-1}^i, ....., o_{t-K}^i, a_{t-K}^i)$, from time $t$ to time $t-K$
            Set target values for each episode using the recurrent neural networks:
            $y_t = r_i + \gamma \mathcal{Q}_i^{\mu'}(h_i^{t+1})|_{a'_k = \mu'_k(o_k)}$
            Update critic by minimizing the loss:
            $\mathcal{L}(\theta_i) = \frac{1}{s} \sum_t (y - Q_i^{\mu}(h_i^{t+1}, a_t^i))^2$
            Update actor using the samples policy gradient:
            $\nabla_{\theta_i} J(\mu_i) \approx \frac{1}{S} \sum_t \nabla_{\theta_i} \mu_i(h_i^t) \nabla_{a_i} Q_i^{\mu}(h_i^{t+1}, a_t^i)|_{a_i = \mu_i(h_i^t)}]$
        end for
        Update target network parameters for each agent i:
        $\theta_i^{'} \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$
    end for
end for
```

5 Experiments and Results

We tested our proposed methods on OpenAI multiagent cooperative domain- Simple Spread environment for cooperative navigation (Fig. 3), in a partially observed multiagent settings.

5.1 Experimental Setup

In this setting, set of N agents cooperate through physical actions to reach a set of L landmarks. The action space of each agent is discretized with 5 actions- up, down, left, right and no-op. The goal for agents is to cover all landmarks. Here, we consider simple case of cooperative navigation with $N = 3$ agents and $L = 3$ landmarks, as shown in Fig. 3. Here, the partial observable environment means that each agent $i \in N \equiv \{1, 2, 3\}$ can observe the relative positions of other agents and landmarks $l \in L \equiv \{1, 2, 3\}$, but not the direction where agents' are heading. The system performance at every time step is the collective reward based on the proximity of any agent to each landmark, which is calculated as following-

$$G_t(z) = -\sum_{\substack{l=1 \\ \forall i}}^{L} min(d_{li})$$

$$l \in L, i \in N \equiv \{1, 2, 3\}$$

where, d_{li} is the euclidean distance of agent i from landmark l.

In the original setting, each agent receives the global reward $G_t(z)$ at every time step t, as shown in Fig. 4. In this paper, we consider reward shaping using zero counterfactual, i.e. when $c_i = 0$, and the world is without agent i. The global reward in this case is given as-

$$G_t(z_{-i}) = -\sum_{\substack{l=1 \\ \forall i' \neq i}}^{L} min(d_{li'})$$

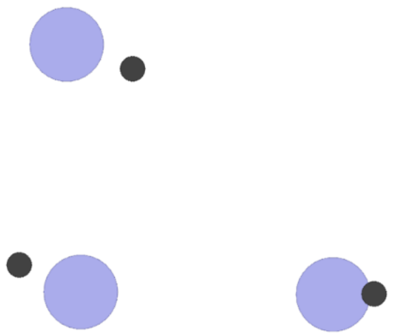

Figure 3: Cooperative Navigation scenario in the Multiagent Particle Environment. The agents (blue circles) learn to spread and cover as many Points Of Interests (POIs) (black dots) as possible. In the original settings, the agents are rewarded at each time-step based on the cumulative of the negative distance from each POI to the closest agent. The agents receive a negative reward for colliding into each other.

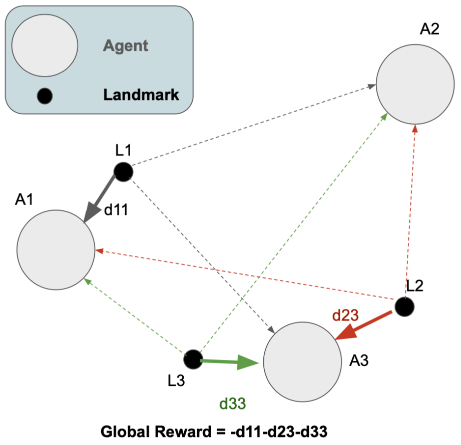

Figure 4: The global reward is the cumulative of the negative of the minimum distance between a Point of Interest (POI), and the nearest agent. For example, here the distance from POI 2 to agent 3 is denoted as $d23$. The problem with such a reward is that the contribution of an individual agent is hard to infer. As shown in figure, the agent A2 does not contribute to the system reward (as it is farthest from all POIs), but still gets a global reward.

Therefore, each agent gets a reward $(g(i))$ equal to the difference reward (D_i), which is estimated as following-

$$g_t(i) = D_t(z_i) = G_t(z) - G_t(z_{-i}), \forall i \in \{1,2,3\}$$

Further, the agents occupy significant physical space and are penalized when colliding with each other.

$$g_t(collision) = -1$$

Therefore, each agent gets the following reward at each time step t,

$$g_t(i) = \begin{cases} g_t(i) + g_t(collision) & \text{if collision} \\ g_t(i) & \text{if no-collision} \end{cases}$$

With this reward structure, each agent learns a policy using difference rewards (instead of global rewards) and thereby they all learn to capture the landmark they must cover, and move there while avoiding other agents.

For the RNN, we maintain a history based on past 5 observations, and use it to predict the next action. We believe that 5 observations would be enough to capture the directional movement of the agents, i.e the information of the heading angle of these agents with respect to any agent. For our experiments, the actor policy is a Recurrent Neural Network (RNN) with two layers with 64 hidden units each. The non-linearity function that we used is Rectified Linear Unit (ReLU). The environment takes the action logits and converts it into a one-hot vector, and takes the respective action in the environment. The critic network is also a two-layer feed-forward neural network with 64 hidden units, and the non-linearity used is ReLU.

5.2 Compared Baselines

We compare the performance of MADDPG-difference rewards and MADDPG-global rewards. Further, we also compare the performance of RMADDG vs MADDPG with difference rewards. All these algorithms are evaluated on the Multiagent simple spread environment, presented in Fig. 3. The hyperparameters for all the algorithms along with the actors and critic neural networks are kept same for all the conducted experiments.

5.3 Methodology for Reported Metrics

We experimented with 2 approaches with some constraints to observe the difference in the behaviour of the learned policy, as following- a) all agents and landmarks initialized randomly in the environment during training as well testing; b) all 3 agents initialized in one of the four random quadrants of the environment, and each of 3 POIs randomly initialized in each of the remaining 3 quadrants, during learning, while both initialized randomly during testing. However, not much difference was observed in the behaviour of learned policies in both these cases. Therefore, for all the future subsections, we will report results corresponding to the (a) case, when both agents and POIs are randomly initialized during training as well as testing. While learning, 25 steps were allowed during each episode, and each step corresponds to an instance when the agent takes an action and gets a reward $(g_t(i))$ from the environment. We train our models for each agent until 500 episodes, and then evaluate them by averaging rewards obtained from 15 statistical runs, with error bars logging the standard errors.

5.4 Results

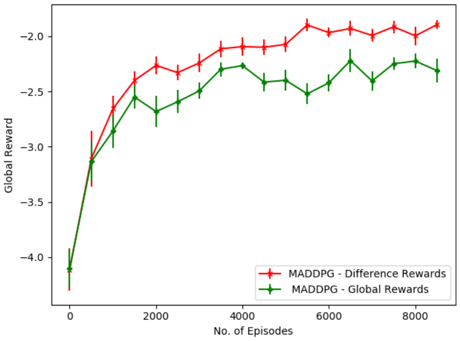

Figure 5: Performance of MADDPG with and without difference rewards. Our results show that MADDPG with difference rewards significantly outperforms the baseline MADDPG algorithm.

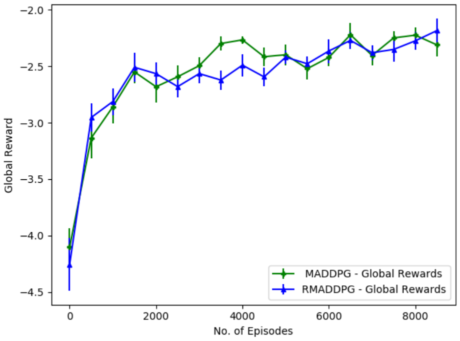

Figure 6: Performance of MADDPG vs R-MADDPG. Our results show that R-MADDPG performs similar to the MADDPG.

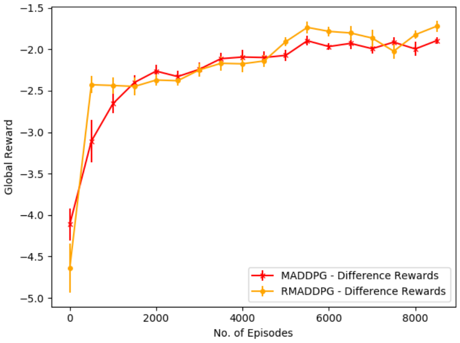

Figure 7: Performance of MADDPG and R-MADDPG with difference rewards. The R-MADDPG performs a little better when finally trained.

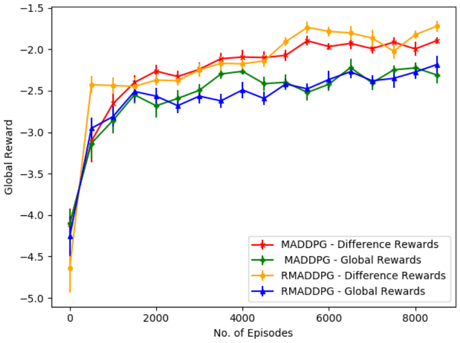

Figure 8: Overall, we see that using difference rewards tend to increase the performance of the system.

5.4.1 Performance comparison of MADDPG with R-MADDPG

The network, and parameters for the MADDPG agents are exactly the same as in [5]. Fig. 6 shows the comparison of system performance with R-MADDPG and MADDPG and the results show that both performs almost similar. We hypothesized that using an RNN would help alleviate partial observability and thus result in better performance. However we see that there is no improvement in system performance when we add an RNN to the actor policy of MADDPG. We believe that this might be due to the global reward used in these settings, which discourages agents to learn any better policies as it adds noise (by providing information of global reward and not individual's own contribution) and even using an RNN is unable to improve the performance deteriorated by the noise from global rewards.

5.4.2 Performance comparison of MADDPG with Global and Difference Rewards

Here, we compare the performance of MADDPG (which uses cumulative "Global" reward of all agents), with a shaped difference reward. Fig. 7 shows the comparison of MADDPG with reward shaping (difference rewards) and MADDPG (with global reward). We plot the learning curves over 9000 episodes for various approaches in Fig. 7. It shows that the MADDPG with difference reward outperforms MADDPG and is able to learn faster too. This confirms our hypothesis that shaping rewards using difference rewards can result in learning signal that provides a better feedback for individual agents and therefore, addresses structural credit assignment problem.

5.4.3 Performance Comparison of R-MADDPG with reward shaping with the Baseline MADDPG

Though R-MADDPG performs equally well as MADDPG with global reward, as explained above, it outperforms the baseline algorithm MADDPG, when implemented with reward shaping using difference rewards. We infer that the reward shaping plays a more crucial role that RNN in improving the performance of the RMADDPG with reward shaping. The availability of individual's reward information to each agent (in the form of difference rewards) boosts R-MADDPG performance. This is shown in Fig. 7 and Fig. 8. The drop in performance of R-MADDPG with the difference reward might be due to the reason that the agents while learning, repeatedly see same levels (steps) as training goes on and could start overfitting, thereby attaining high rewards in the learning curves, while performing poorly in the evaluation steps.

6 Conclusion

In this paper, we introduced an algorithm- Recurrent Multiagent Deep Deterministic Policy Gradients (R-MADDPG) to address the problem of partial observability. Additionally, we also introduce the use of reward shaping using difference rewards to address structural credit assignment problem. We compared our algorithm with MADDPG, which is the current state-of-the-art baseline multiagent coordination algorithm. Our results show that R-MADDPG achieves the same performance when compared to MADDPG when implemented with global rewards and we infer that noisy information due to global reward does not encourage RNN to perform any better than system without RNN. Further, our results show that MADDPG with difference rewards perform significantly better than just MADDPG, , which proves our hypothesis that reward shaping improves the system performance and enables agents to learn better coordination in a cooperative environment. For future work, we want to make the environments harder by introducing adversarial agents in a competitive environment. We also look forward to implement the R-MADDPG with reward shaping in a tightly coupled environment. Furthermore, coming up with different counterfactual scenarios for difference rewards needs to be investigated.

References

[1] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[3] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[5] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[6] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[9] S. Proper and K. Tumer. Modeling difference rewards for multiagent learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1397–1398. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[11] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[12] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4): e0172395, 2017.

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

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

[15] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36 (5):823, 1930.

[16] S. Whiteson. Stabilising experience replay for deep multi- agent reinforcement learning. 2017.

7 Appendix

Table 1: Individual's contribution to Project

Metric	**Yathartha**	**Enna**	**Manish**
Organization	33.33%	33.33%	33.33%
Technical Contribution	40.00 %	30.00%	30.00%
Coding	45.00%	20.00%	35.00%
Writing	30.00%	45.00%	25.00%