
Generalized Multi-Agent Reinforcement Learning in Cooperative and Competitive Environments

Diana Huang¹ Shalini Keshavamurthy¹ Nitin Viswanathan¹

Abstract

Multi-agent scenarios frequently come up in the real world, but traditional reinforcement learning algorithms do not perform well on them due to constantly changing environments from the perspective of any one agent. We replicate multi-agent deep deterministic policy gradients (MADDPG), an algorithm tailored to multi-agent scenarios, and evaluate its performance in a 3-agent cooperative navigation OpenAI environment. Additionally, we perform many experiments to explore how changing hyperparameters affects performance and attempt to address the main weakness of MADDPG, which is that it does not scale well to larger numbers of agents.

Our results confirm that MADDPG performs significantly better than a single-agent policy gradient approach and show the importance of batch size in performance and training stability.

1. Introduction

There are many applications where multiple agents need to learn how to act together, such as multiplayer games (?), multi-robot control (?), communication scenarios (?), and even self-play (?).

Traditional reinforcement learning approaches focus on training a single agent and as a result they do not work well on multi-agent problems because the environment for any single agent changes over time as other agents change their policies, leading to instability during training and high variance in results (?). Additionally, experience replay as used with Q-learning cannot be directly applied in multi-agent scenarios, significantly hampering stable learning.

Research into multi-agent reinforcement learning has attempted to develop new approaches specifically for multi-

agent scenarios. One such recently developed algorithm is multi-agent deep deterministic policy gradients algorithm (MADDPG), which obtains significantly improved results over other approaches by modifying DDPG to train agents while incorporating information from other agents (?).

We replicate the MADDPG algorithm and apply it to an OpenAI cooperative navigation environment, where agents must learn to each navigate to a different fixed landmark without colliding. We experiment with various hyperparameters and exploration vs. exploitation methodologies to show how MADDPG's performance is affected by them. We also generalize MADDPG to handle larger numbers of agents and explore how performance changes accordingly.

2. Related Work

2.1. Single-agent methods

TODO - prior work applying PG directly, Q-learning directly, etc.

2.2. Multi-agent methods

TODO - what we've read on multi-agent approaches including MADDPG

3. Approach

3.1. Environment

We use the cooperative navigation OpenAI Gym environment for our experiments (?). This is a 2-D world with N moving agents and N fixed landmarks ($N = 3$ by default), where the agents must learn to each move to a different landmark while avoiding collisions with each other.

Every agent has its own state that is an 18-dimensional vector containing its own position and velocity, relative distance to each of the landmarks, and relative distance to the other agents. The action that every agent can take is represented as a 5-dimensional vector, with 4 dimensions representing movement up/down/left/right and the last dimension representing a no-move action. This is a continuous action space, with the values for each action dimension representing acceleration in a given direction.

^{*}Equal contribution ¹Stanford University, Palo Alto, USA. Correspondence to: Diana Huang <hxydiana@stanford.edu>, Shalini Keshavamurthy <skmurthy@stanford.edu>, Nitin Viswanathan <nviswana@stanford.edu>.

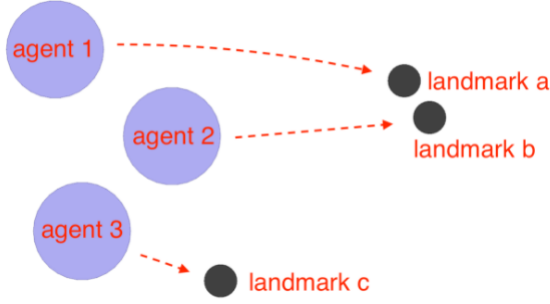


Figure 1. Cooperative navigation environment. Agents must learn to each navigate to a different landmark without colliding.

All agents are rewarded based on the distance between landmarks and agents (the closer the agents are to separate landmarks, the higher the reward). Agents have a non-zero size, so it is possible for them to collide in this environment. Agents receive a reward of -1 if there are collisions at any timestep. Therefore, in order to maximize reward, the agents need to each learn to move to a different landmark while also avoiding collisions with each other.

3.2. Policy Gradient

We implement a policy gradient algorithm as a baseline. Policy gradient algorithms seek to determine the optimal policy by directly adjusting the parameters θ of a policy π_θ in order to maximize the objective function $J^*(\theta) = \mathbb{E}_{\pi_\theta}[R]$. We also implemented a baseline and advantage for the policy gradient. Letting $\hat{A} = R_t - b(s_t)$ be the advantage estimate at time t , we can write the gradient of the policy as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s, \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) \hat{A}]$$

In the above equation, the expectation is taken over many sample trajectories through the environment using π_θ . Policy gradient algorithms are already susceptible to high variance, and executing them in a multi-agent environments only amplifies the problem. From a single agent’s perspective not only does the world change as other agents move, but the policies other agents follow change as well. Additionally, individual agents might not always get the correct gradient signal as an agent could take a good action but if other agents took bad actions, leading to overall decrease in reward, the gradient for the agent taking the correct action could still be negative.

3.3. MADDPG

TODO - diagram/visualization of how MADDPG works

We recreate MADDPG as our main algorithm for the cooperative navigation task (?). MADDPG takes DDPG and tai-

lors it to multi-agent environments, resulting in significantly better performance than single-agent methods. MADDPG trains separate policy networks for every agent, but uses a centralized action-value function that incorporates information across all agents to ensure a more stable learning process.

Consider a game with N agents, and let $\pi = \{\pi_1, \dots, \pi_N\}$ be the associated set of agent policies that are parametrized by $\theta_1, \dots, \theta_N$. Additionally let $s = \{s_1, \dots, s_N\}$ represent the set of states observed by each agent. When using MADDPG to train agents, every agent additionally maintains a critic function that outputs a Q-value given the states and actions of all agents:

$$Q_i^\pi(s, a_1, \dots, a_N)$$

Standard Q-learning techniques are used to update the critic function. Note that for the cooperative navigation environment that we run our experiments on, the Q-values will be the same for all agents as the reward for all agents is the same. However, this Q-function formulation where every agent learns its own Q_i^π allows MADDPG to generalize to environments where agents have different reward functions (e.g. adversarial games).

During training, agents now have information not just about the locations of other agents, but about the actions they will take as well. Therefore, the environment is stationary for any agent as it does not depend on other agent policies:

$$P(s'_i | s_i, a_1, \dots, a_N, \pi_1, \dots, \pi_N) = P(s'_i | s_i, a_1, \dots, a_N)$$

The policy network updates for an agent are determined by maximizing the agents centralized Q-values. The gradient update is similar to the update used in the policy gradient, except that it uses the centralized Q-value instead of only looking at the returns across trajectories:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s, \pi_i} [\nabla_{\theta_i} \log \pi_i(a_i | o_i) Q_i^\pi(x, a_1, \dots, a_N)]$$

By incorporating the actions of all agents into the action-value function during training instead of only each individual agent, MADDPG produces better and more stable results than traditional policy gradient or Q-learning approaches. During policy execution, agents only use their learned policy networks and therefore do not use any information about other agents; MADDPG is an enhancement to the training process while leaving the execution phase unchanged.

3.4. More scalable MADDPG

In addition to implementing MADDPG as described in the previous section, we further modify it to scale to better support large numbers of agents. The Q function described above increases linearly in input size as more agents are added to the scenario, as it has to take every agent’s observation and action as an input. In order to reduce the

additional computation time needed as N grows, we modify MADDPG to only look at the K nearest neighbors during training, where $K < N - 1$. The mathematical formulation of the algorithm remains the same, but the inputs to Q_i^π vary based on which agents are closest to i at each timestep.

4. Experiments and Results

4.1. Evaluation Metrics

TODO - update

Our primary evaluation metric has been average reward, which in this environment corresponds to the distance between agents and the target landmarks while also accounting for collisions that occur along the way. In addition to average reward, in the future we will also look at success rate, where we define success as agents successfully reaching the target landmarks with 0 collisions, no matter how long it took for this to occur. This way, we will evaluate the policy that our algorithm converges to not only based on how quickly it leads agents to reach the landmarks, but how often it leads agents to the landmarks.

4.2. Experiments

TODO - include all of the various experiments we ran, and an analysis of each of them

TODO - also include a table with our main hyperparameters (copy from poster) and explain what they are for in text

Figures X,Y,and Z show our initial results from training with different policy network sizes. We can see that although there is a significant improvement in average reward from the start of training, the average reward is still well below 0, indicating that the agents are not able to find an optimal strategy to cooperate and cover all of the target landmarks. This result is to be expected as policy gradient will be very unstable due to the non-stationary environment from the point of view of any given agent during training. When we increased the number of layers and units per layer we saw the network reach convergence quicker, but it was still not able to improve substantially, indicating that the algorithm itself could be a bottleneck on performance as opposed to the size of the policy networks.

5. Conclusion

TODO - update

6. Future Work

TODO - update

Apply our work to other OpenAI environments (e.g. com-

munication and adversarial ones), explore using a modular Q function to generalize instead of a fixed number of neighbors, etc.

Contributions

Diana wrote the initial implementation of MADDPG, Shalini wrote our logging/plotting code, and Nitin wrote the initial implementation of REINFORCE. All of us iterated on the code with bugfixes and enhancements and ran experiments using our implementations.

References