Generalized Multi-Agent Reinforcement Learning in Cooperative and Competitive Environments

Diana Huang[1] Shalini Keshavamurthy[1] Nitin Viswanathan[1]

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 (Peng et al., 2017), multi-robot control (Matignon et al., 2012a), communication scenarios (Foerster et al., 2016), and even self-play (Sukhbaatar et al., 2017).

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 (Matignon et al., 2012b). 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 (Lowe et al., 2017).

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 (OpenAI, 2017). 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

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

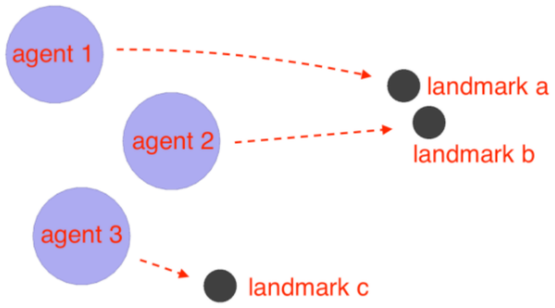

Figure 1. Cooperative navigation environment. Agents must learn to each navigate to a different landmark without colliding.

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.

At each timestep, all agents are rewarded based on the distance between landmarks and agents; the closer the agents are to separate landmarks, the higher the reward. If every agent was exactly on top of a different landmark, the reward at that timestep would be 0. 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.

We define an episode to be 25 timesteps long, as there is no defined terminal state in this environment.

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 (Lowe et al., 2017). MADDPG takes DDPG and tailors it to multi-agent environmnnts, 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 $\pi = \{\pi_1, ..., \pi_N\}$ be the associated set of agent policies that are parametrized by $\theta_1, ..., \theta_N$. Additionally let $s = \{s_1, ..., s_N\}$ 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:

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

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:

$$P(s_i'|s_i, a_1, ..., a_N, \pi_1, ..., \pi_N) = P(s_i'|s_i, a_1, ..., a_N)$$

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:

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

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

In order to evaluate the performance of our algorithms, every 250 training steps we do evaluation by running 40 episodes using the latest learned policies and calculate various metrics over them. Our primary evaluation metric for all experiments is 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, we also look at average distance between landmarks and agents, for more intuitive interpretation of the performance. Here we provide a detailed definition of average reward and average distance used in this paper, unless specified otherwise.

Average reward. Average reward is calculated by summing up rewards received from all agents, all timesteps within an episode, and then averaging across multiple episodes. Average rewards shown in plots are computed during dedicated evaluation runs whose data do not go into training set. The range of average reward is zero to negative infinity.

Average distance. Average distance is the sum of distances between each agent and its closest landmark, across all agents and all timesteps within an episode, and then averaged across episodes.

Besides the metrics defined above, each experiment may additional context-specific evaluation metric, which will be defined for that experiment.

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

REINFORCE and MADDPG. As shown in Figure 2 and Figure 3, REINFORCE has difficulties learning a good pol-

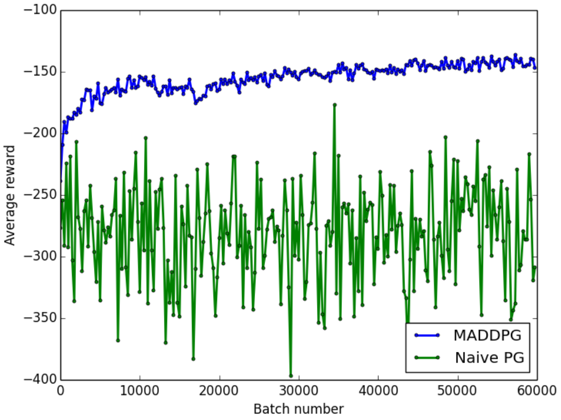

Figure 2. Average reward achieved by Naive PG (REINFORCE) algorithm and MADDPG algorithm.

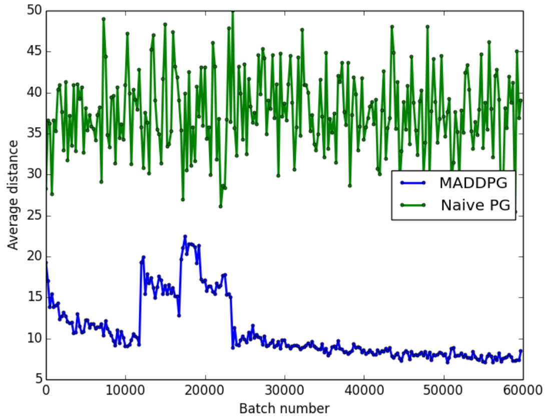

Figure 3. Average distance from landmarks with Naive PG algorithm and MADDPG algorithm.

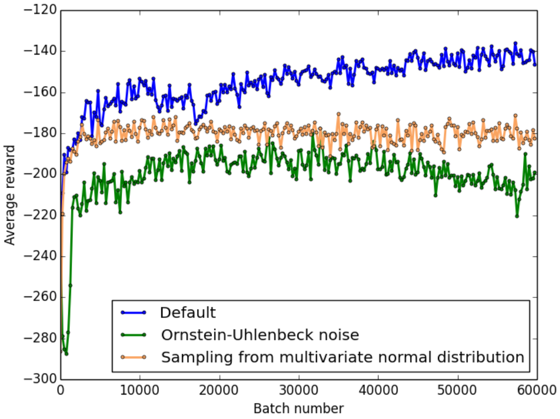

Figure 4. Average reward gained by using different exploration strategies.

icy as the world keeps changing from each agent's perspective. If other agents make a wrong move, it would reflect badly on this agent even though it might have picked a good action. In other words, the feedback signal is erroneous. Whereas with MADDPG, the centralized critic function takes into consideration other agents' observations and moves, so that the network could more accurately attribute rewards and penalties.

Exploration strategies. We experiment with 3 different ways of action exploration. First, we try re-implementing what Open AI does, which adds noise onto unnormalized policy network output, and then takes softmax to get the final action. We also try using Ornstein-Uhlenbeck process, with $\sigma = 0.3, \theta = 0.15$. Lastly, we experiment sampling from multivariate normal distribution with a trainable standard deviation.

We found that applying noise to unnormaized policy network output outperforms other strategies. Therefore we use that as our default exploration when conducting other experiments. To further evaluate, we compare the Euclidean distance between the effective displacement as a result of pre-noise action versus post-noise action. The Euclidean distances are then averaged across every 100 steps (that is the training frequency) across all agents. Figure 5 and 6 plot the range and standard deviation of the average Euclidean distances for the first 15000 batches. They suggest

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

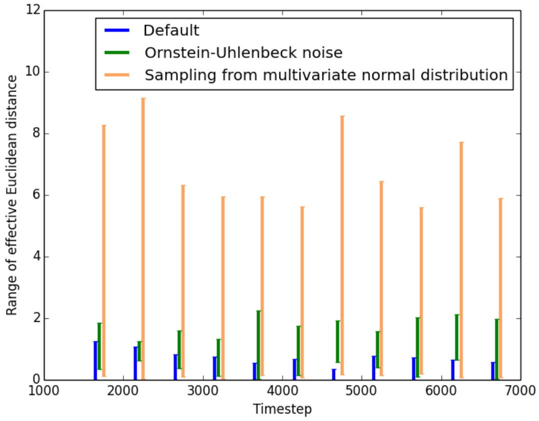

Figure 5. Average reward gained by using different exploration strategies.

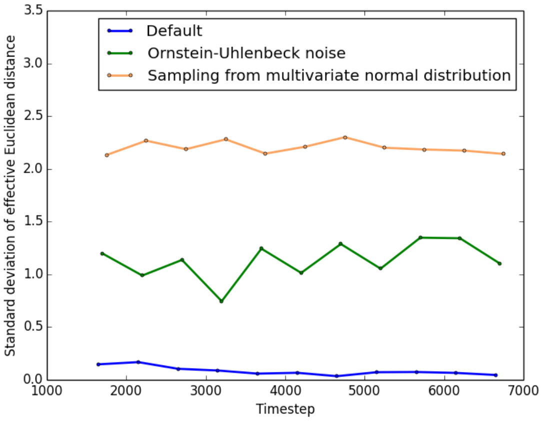

Figure 6. Average reward gained by using different exploration strategies.

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

We introduce the problem of multi-agent reinforcement learning and show why traditional single-agent approaches are ineffective on these problems. We implement a policy gradients as a baseline and also MADDPG, an algorithm designed for multi-agent scenarios, and apply them to the OpenAI cooperative navigation environment.

6. Future Work

As future work, we would like try our approach on other multi-agent environments, in particular adversarial ones, to compare how MADDPG performs vs. policy gradients. It would also be interesting to explore how performance changes if agents are trained using different policies - e.g. in an adversarial game, what would happen if one set of agents were trained with MADDPG and a competing set were trained with policy gradients.

Additionally, we would like to explore alternative forms of generalization as we feel this is key to supporting. Instead of Q_i^π looking at the K nearest neighbors to agent i for some fixed K, the Q-function could instead look at all neighbors within a particular distance to agent i. This way, the agents would have more information available to them when it is particularly important (the closer agents are to each other, the more likely it is to have a collision) while minimizing input space size when information about additional agents is less necessary.

Contributions

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

References

Foerster, J. N., Assael, Y. M., de Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning, 2016.

Lowe, Ryan, Wu, Yi, Tamar, Aviv, Harb, Jean, Abbeel, Pieter, and Mordatch, Igor. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017. URL http://arxiv.org/abs/1706.02275.

Matignon, L., Jeanpierre, L., and et al., A.-I. Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes, 2012a.

Matignon, L., Laurent, G. J., and Fort-Piat, N. Le. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems, 2012b.

OpenAI. Multi-agent particle environments. https://github.com/openai/multiagent-particle-envs, 2017.

Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., and Wang, J. Multiagent bidirectionallycoordinated nets for learning to play starcraft combat games, 2017.

Sukhbaatar, S., Kostrikov, I., Szlam, A., , and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play, 2017.