ruled algorithmhtbploa algorithmAlgorithm

Generalized Multi-Agent Reinforcement Learning in Cooperative and Competitive Environments

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

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 applied in multi-agent scenarios, significantly hampering stable learning.

In addition to applying traditional approaches, prior work on multi-agent reinforcement learning (MARL) has attempted to develop new approaches specifically for multi-agent scenarios. In particular, recent work has achieved significantly improved results by modifying existing RL algorithms to train agents while incorporating information from the other agents, creating multi-agent deep deterministic policy gradients (MADDPG) (**?**). However, one key limitation of MADDPG and other related approaches is that they do not generalize well to larger numbers of agents due to the growing computational complexity of incorporating information from all agents in training. Given the extensive training time necessary for even smaller scenarios, it is not feasible to retrain new networks for different possible numbers of agents.

We plan to develop a generalized training approach for multi-agent scenarios to find policies that scale well when applied in environments with varying numbers of agents. We do so by modifying the training process of MADDPG to train agents using only information from nearby agents instead of information from every agent, allowing agents to learn to make decisions based off of the location of a fixed number of nearby agents. As a baseline, we have implemented a REINFORCE policy gradient approach and will also re-implement MADDPG. We will compare these algorithms against our generalized algorithm to see how they perform in scenarios with various numbers of agents.

2. Related Work

All our read papers.

3. Approach

3.1. REINFORCE - Policy Gradient Method

As an initial baseline, we have implemented the REINFORCE algorithm for multi-agent scenarios. With REINFORCE, every agent trains its own independent policy network based on its observation of the environment and the possible actions it can take. Even after applying a baseline and advantage normalization we did not expect REINFORCE to perform well due to the potential for very large variance in the gradient. 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. Nevertheless, REINFORCE serves as a baseline for comparing against more sophisticated approaches.

3.2. Multi-Agent Actor Critic Deep Deterministic Policy Gradient

We are currently in the process of re-creating MADDPG from (**?**) as a second baseline. MADDPG takes DDPG and tailors it to multi-agent scenarios, resulting in significantly better performance than single-agent methods such as REINFORCE and DDPG. 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 $\theta_1, ..., \theta_N$. The gradient update for MADDPG for agent i, $\nabla_{\theta_i} J(\theta_i)$, is:

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

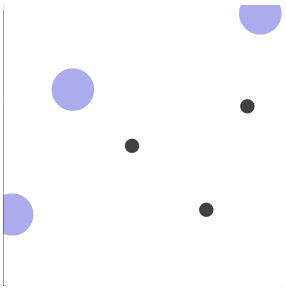

Figure 1. Example environment initialization. Blue circles are agents and black dots are target landmarks.

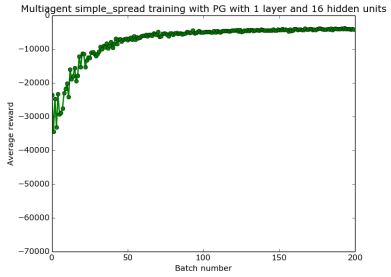

Figure 2. Average reward over time when training PG with 1 layer and 16 hidden units.

Where o_i is the local observation of the environment by agent i, x is the full state information across all agents, and Q_i^π is a centralized action-value function that takes as input the actions of all agents. By incorporating the actions of all agents into the action-value function, instead of only each individual agent, MADDPG produces better and more stable results than traditional policy gradient or Q-learning approaches.

After re-implementing MADDPG, we plan to modify the original MADDPG algorithm to train agents in a generalized way. More specifically, we will still maintain a centralized critic function for each agent, but instead of taking actions from all agents as input, only actions from neighboring agents would be considered. This is to make the number of parameters in the Q network constant, rather than linear to number of agents in the environment. Thus, the computational complexity for updating the Q network is also capped.

4. Experiments

4.1. Environment

We use an OpenAI Gym environment with a variety of multi-agent cooperative and competitive scenarios as well as environments that allow communication between agents (**?**). For our initial explorations, we focused on the cooperative navigation environment where there are N agents that can move in a 2-D world and N fixed landmarks. All agents are rewarded based on the collective distance between agents and landmarks, and punished for colliding with each other. In order to maximize reward, each agent needs to learn to move to a different landmark while avoiding collisions with other agents.

The observation space for the cooperative navigation environment consists of information about every agent's position and velocity, relative distance to all of the landmarks, and relative distance to every other agent. An agent acts by choosing to move either up, down, left, right, or taking a no-move action. During test time, an agent can only take actions based on it's own observations of the world, but during train time, MADDPG and other approaches tailored to multi-agent scenarios will allow agents to use observations from other agents as well.

For the milestone, we have been focusing on the cooperative navigation environment as mentioned previously. 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.

We will evaluate our model performance during both train and test time. During train time, we will see how quickly we are able to reach convergence and how stable the training process is across the different implemented methods. During test time, we will compare our different methods with the average reward over many trials as well as the variance, and also see how performance changes as we change the number of agents present to test generalization. The original authors of MADDPG only tested scenarios with a maximum of 3 agents , so we will try increasing this to 4 to 6 agents and seeing how performance changes.

For our re-created version of MADDPG, we expect to match results from the original MADDPG paper as closely as possible when evaluating on up to 3 agents. For more than 3 agents, we will compare results from our re-created MADDPG and the modified, generalized version of MADDPG. We aim to have the modified MADDPG perform at least as well as the re-created MADDPG but train faster than the re-created MADDPG.

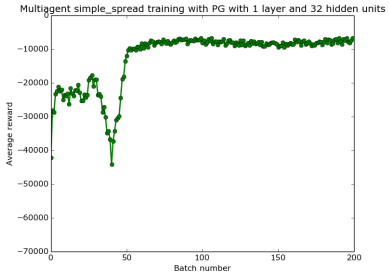

Figure 3. Average reward over time when training PG with 1 layer and 32 hidden units, gamma=0.95, learning rate=2e-3.

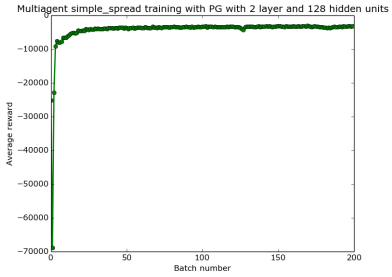

Figure 4. Average reward over time when training PG with 2 layers and 128 hidden units.

4.2. Results

Figures **??**, **??** and **??** 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.