Generalized Multi-Agent Reinforcement Learning in Cooperative and Competitive Environments

Diana Huang [*1] **Shalini Keshavamurthy** [2] **Nitin Viswanathan** [*1 2]

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., 2012), communication scenarios (Foerster et al., 2016), and even self-play (Sukhbaatar et al., 2017). Existing efforts in Multi-Agent Reinforcement Learning (MARL) have looked at both existing reinforcement learning methodologies as well as modifying them to better support multiple agents.

Traditional reinforcement learning approaches, focused on training a single agent, do not work well on these 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 (cite). Furthermore, experience replay as designed for traditional Q-learning cannot be applied in multi-agent scenarios, significantly hampering stable learning.

Recent work has achieved better results in mutli-agent scenarios by modifying existing RL algorithms to specifically operate in this space, creating Multi-agent Deep Deterministic Policy Gradients (MADDPG) (Lowe et al., 2017). However, one key limitation of these 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 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 only train agents using information from nearby agents instead of information from every agent, allowing agents to train only based off of the location of a fixed number of nearby agents instead of every agent in the environment.

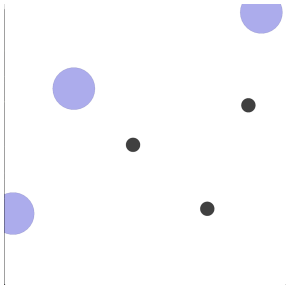

Figure 1. Example environment initialization. Blue circles are agents and black dots are target landmarks.

2. Approach

2.1. Environment

We use an OpenAI Gym environment with a variety of cooperative and competitive scenarios, as well as environments that allow communication between agents (OpenAI, 2017). 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. So in order to maximize reward the agents need to learn to each move on top of a different landmark and avoid running into each other.

The observation space for the cooperative navigation environments 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.

As part of our post-milestone investigations, we plan to apply our methods to other multi-agent environments as well as they should generalize.

2.2. Methods

As an initial baseline, we have implemented the REINFORCE algorithm for our multi-agent scenario. With this methodology, every agent trains it's own independent policy

network based on the observed environment and possible actions it can take. We did not expect REINFORCE to perform well on the environment due to how much it changes across time steps - 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.

We are currently in the process of re-creating MADDPG from (Lowe et al., 2017) 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 $\theta_1, ..., \theta_N$. MADDPG then writes the gradient update for agent i, $\nabla_{\theta_i} J(\theta_i)$, as:

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

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 instead of that from one specific agent. 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 modified 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.

2.3. Experiments

For the milestone, we have been focusing on the cooperative navigation environment as mentioned previously. Our primary evaluation metric so far 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 are evaluating our methodology both based on how often it converges quickly and how often it converges without errors.

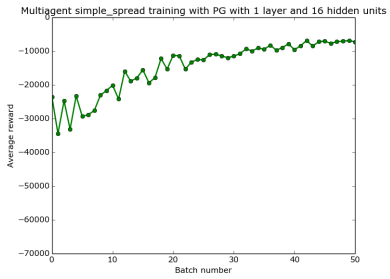

Figure 2. Average reward over time when training PG with 1 layer and 16 hidden units.

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 4-6 agents and seeing how performance changes.

For our re-created version of MADDPG model, 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.

2.4. Results

Figures 2 and 3 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

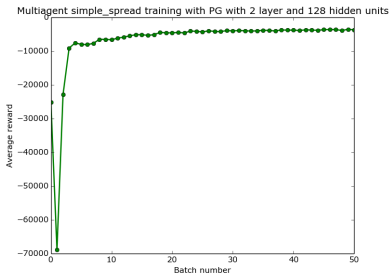

Figure 3. Average reward over time when training PG with 2 layers and 128 hidden units.

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.

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, 2012.

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.