Distributed Multi-Agent Reinforcement Learning

Xinda Zhou xindaz@andrew.cmu.edu

Ketong Chen ketongc@andrew.cmu.edu

I. Abstract

Reinforcement learning (RL) has recently been applied to solve challenging problems, from game playing to robotics. However, a large number of real-world problems cannot be fully solved by a single active agent interacting with the environment. The solution is Multi-Agent Reinforcement Learning (MARL), where several agents learn concurrently how to solve a task by interacting with the same environment and with each other. RL problems are known to have huge amounts of data and extended training time, and training multiple agents at the same time adds extra complexity. Therefore, in this project we focus on applying the power of distributed computing in solving MARL problems. We based our work on a baseline model Multi-Agent Deep Deterministic Policy Gradient (MADDPG) and applied various distributed methods to speed up training time. In this project, we explored Simple Tag in the Multi Particle Environment for training, and greatly improved the training results from the original algorithm using about identical training time.

II. Related Work

The baseline model we will use is Multi-agent Actor-Critic Mixed Cooperative -Competitive Environments (MADDPG) by paper [1]. MADDPG applied an adaptation of actor-critic methods of centralized training with decentralized execution. Specifically, the actors act only according to local information during execution while the critics have access to other agents' policies, which could potentially lead to increasing interactions between agents. A more detailed version of the learning process can be visualized in Figure 1. The inner red region represents the actors, which only uses individual observation to generate their respective action. These actions are then sent to all the critics represented by the outer green region. Every critic has access to all policies as inputs to perform centralized training.

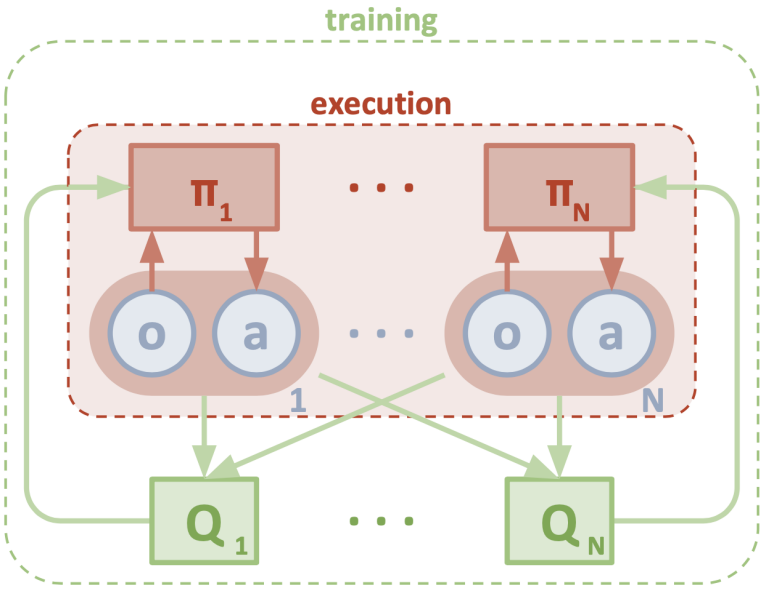


Figure 1. Visualization of multi-agent centralized critic and decentralized actor.

In this baseline model, Deep Deterministic Policy Gradient (DDPG) was used in this baseline model. The objective of the problem is to find the policy parameters to maximize the reward. The policy gradient for each agent i in the deterministic circumstance is:

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x},a\sim\mathcal{D}}[\nabla_{\theta_i}\boldsymbol{\mu}_i(a_i|o_i)\nabla_{a_i}Q_i^{\boldsymbol{\mu}}(\mathbf{x},a_1,...,a_N)|_{a_i=\boldsymbol{\mu}_i(o_i)}]$$

where $\boldsymbol{\mu}_i$ represents the policy of agent i, a_i and o_i represents the action of observation of agent i. $Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N)$ represents the critic's centralized action-value function that takes all agents' actions and all observations as input. This $Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N)$ action-value function is updated as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x},a_1,\ldots,a_N) - y)^2], \quad y = r_i + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}',a_1',\ldots,a_N')\big|_{a_j'=\boldsymbol{\mu}_j'(o_j)}$$

To increase efficiency, agents learn to infer other agents' policies online to reduce the communication. Specifically, each agent i can additionally maintain an approximation of other agents' policies, $\boldsymbol{\mu}_i^j$. This approximate policy is learned by maximizing the log probability of agent j's actions with an entropy regularizer.

In addition, an ensemble of policies was introduced for agents to learn more robust policies with higher stability. K different sub-policies are trained at the same time. At each episode, one random sub-policy is selected to execute, and a collection of all K sub-policies' rewards are used as the objective.

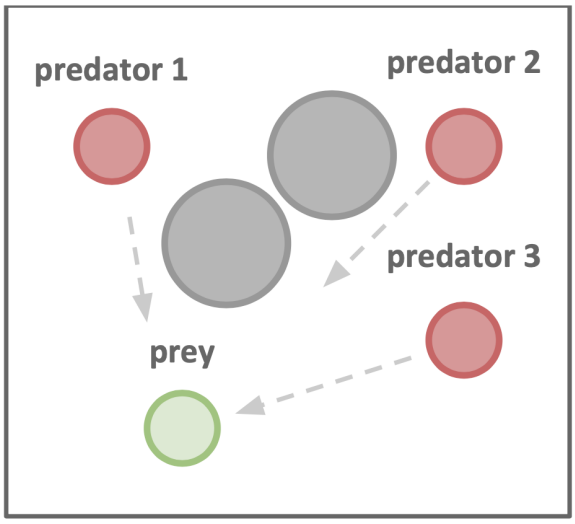


Figure 2. Simple Tag environment visualization.

III. Baseline

As baseline, we trained default MADDPG on Simple Tag under the Multiagent Particle Environment. Details of the algorithm and hyperparameter set can be found in the Appendix.

Simple Tag (Figure 2) is a predator-prey environment, where the prey (green) is faster and wants to avoid being hit by a group of predators (red). Obstacles (large black circles) block the way. In each timestep, each predator agent gains a reward of 10 for being in contact with the prey agent, while the prey agent receives a reward of -10 for being hit.

In our configuration of Simple Tag, we have 3 predator agents and 1 prey agent. We set maximum episode length to 50 timesteps, and each episode's maximum reward is 1500. We trained the 3 predators using MADDPG for 2 million timesteps against a weak pretrained prey, and we were only able to achieve at most 600 average reward (Figure 3). When we visualized our trained agents, we noticed that two of the three adversary agents are consistently following and trapping the good agent, while the remaining adversary almost always wanders off. With further training, it is possible that this third adversary will learn to cooperate with the other two, but we hope to find a way to improve training efficiency and speed up cooperation between agents so that average return converges faster and closer to the maximum.

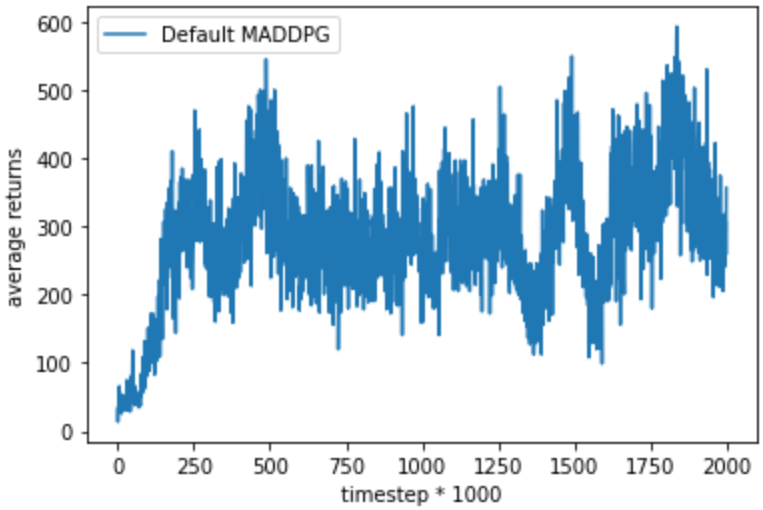


Figure 3. Baseline Training Results

IV. Distributed MADDPG

Our first goal of the project is to speed up the training of MADDPG by distributing computation on multiple workers running in parallel. Here we tried three different distributed MADDPG methods: (1) data parallelism; (2) model parallelism where each worker represents one agent; (3) model parallelism where each worker represents one agent and an additional worker construct replay buffer. For method (1), the same networks were duplicated on multiple workers and each worker is trained using different micro-batches of transitions. Gradient is calculated respectively on every worker, and sent to the server to aggregate and then sent back to each worker for the following update. This method allows the calculation of gradients on different data to be parallel. For method (2), we use multiple workers that each contain actor and critic networks of a single agent. The communication among workers is managed using Celery. After each iteration, the respective results, i.e. actions determined by the actors, are sent to a server and the server will send back to each node all actions to boost the centralized critic models. In this method, each worker acts as an agent so that training of multiple agents are in parallel. For method (3), the workers mentioned in the previous method are kept the same, while an additional worker is used to sample transitions from the environment and store them in the replay buffer. This parallelizes the process of generating transitions alongside the actual training, and could help reduce run time especially when batch size is large and the environment is very complicated to take actions and get results.

Using the three methods introduced above, we have made the MADDPG algorithm parallel in different ways with the cost of frequent communication between workers and server. These methods would in theory greatly reduce runtime and increase efficiency when model training and/or environment stepping are the bottlenecks. However, in our case the environment and model are both very simple. Specifically, our actor and critic models consist of 2 hidden

layers of 64 neurons each, and the dimensions of our state space and action space are 87 and 20 respectively. In addition, the training of our distributed MADDPG, similar to most RL problems, requires a large number of training steps to explore the environment and eventually converge. In our case, the agents take around 1 million to 2 million steps to train from scratch and reach good performance. Any of the three methods above requires multiple rounds of communication for every train step and results in huge communication overhead. For example, running method (3) on 5 AWS t2.2xlarge instances using Celery for communication is in fact 10 times slower than running default MADDPG on one instance.

Despite our failure to improve training efficiency with the current model and environment setup, our distributed MADDPG algorithms will provide a much greater ratio of runtime speedup when the model and environment scale up. In experiments where we doubled, tripled, and quadrupled the number of hidden layers and number of nodes per layer in the actor and critic models, total training time did not change significantly, meaning that the bottleneck remains to be communication latency. We also believe that in more complex environments where tens or even hundreds of agents are taking actions simultaneously, method (2) and method (3) will have significant speedup with enough compute resources. Given our limited time and resources for this project, we decided to pursue other means of enhancing MADDPG using distributed computing.

V. Distributed Genetic MADDPG

To increase the performance of the baseline algorithm while avoiding excessive communication, we created an algorithm that takes ideas from genetic algorithms and named it Distributed Genetic MADDPG. In this method, workers each run the whole MADDPG algorithm separately, and the server periodically selects the best worker based on their per episode average reward. The critic network parameters of the selected best worker will be used to update the critic networks on all other workers. Intuitively, the best performing worker shares its own interpretation of the environment to every other worker to improve the overall performance. Take note that actor networks never sync in this process to preserve unique policies learned by each worker. Since critic networks are synced periodically, minimal communication rounds are needed. The small size of our networks also ensures that sending network parameters does not create noticeable overhead. Using this genetic method, we tuned hyperparameters of both the actor and critic models and selected the hyperparameter set with the best performances (see Appendix). The algorithm runtime is identical to default MADDPG, but we are able to consistently achieve higher average rewards.

To assess our result, we have compared the default and improved version of MADDPG in two different situations.

Firstly we compare the results of default MADDPG with our algorithm when training against a weak pre-trained prey (Figure 4). Here we can see a distinct increase in improved MADDPG, indicating that it is continuously learning and improving in the process. However,

default MADDPG outperformed our algorithm during the first 500,000 timesteps. We believe that our algorithm's weak initial performance is due to the fact that periodically syncing networks hinders exploration, and exploration is especially important in early stages of training.

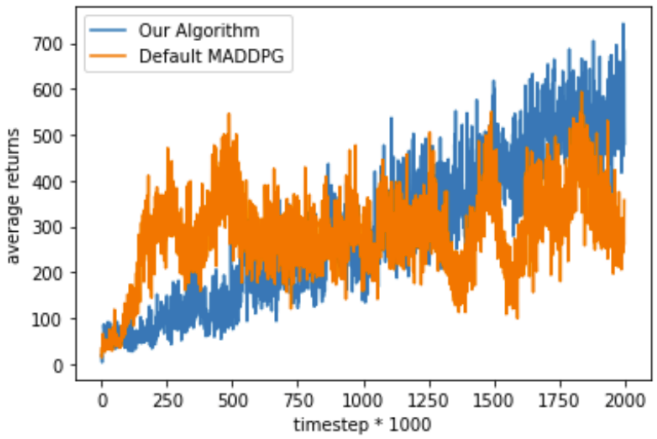


Figure 4. Default VS improved MADDPG when training predators to play against weak prey

Next we compared our algorithm's performance against default MADDPG when training predators against a strong prey pre-trained for 2 million timesteps (Figure 5). We can see that both models have shown only a slight increase in rewards across the training, since the prey is already very good at avoiding the predators. Our distributed genetic MADDPG has consistently achieved higher rewards than the original algorithm across the board.

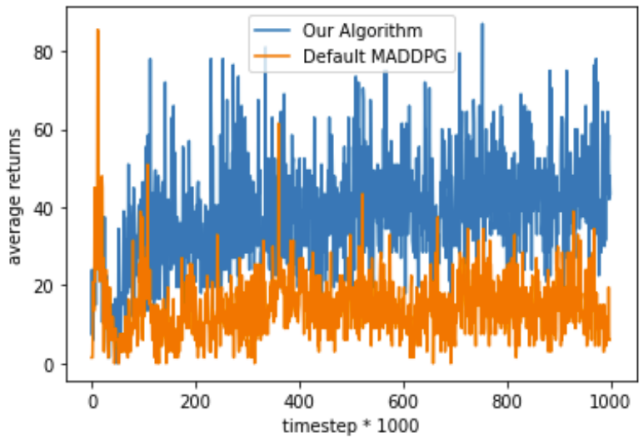
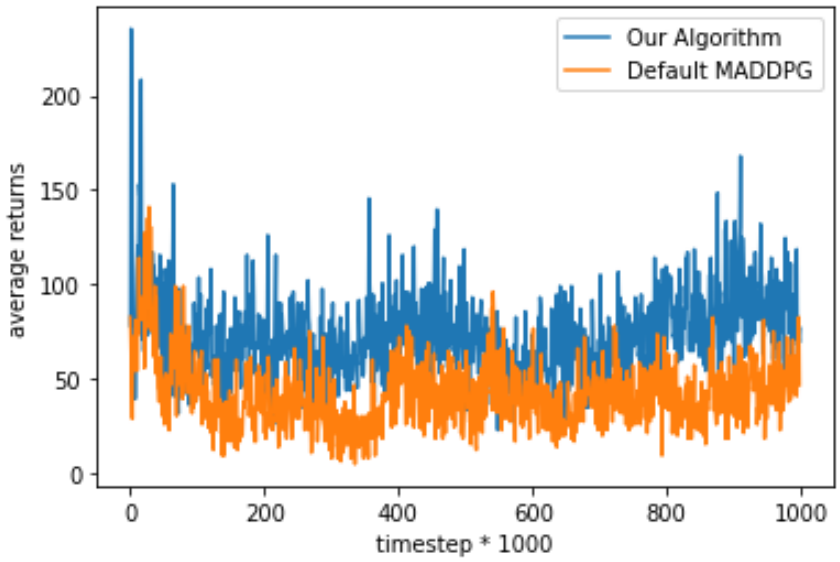


Figure 5. Default VS improved MADDPG when training predators to play against well-trained prey

Lastly, we compared our algorithm against default MADDPG when training both predators and prey from scratch simultaneously (Figure 6). We can see that our algorithm

consistently achieves higher average rewards even when the prey is constantly adapting their policy against policies of predators.



VI. Future Work

Periodically syncing networks is likely to restrict exploration, which greatly affects the initial stages of training agents from scratch. In the future, we plan on experimenting with an adaptive critic sync rate that decreases as time step increases.

In MARL environments, cooperation is essential for agents to achieve high rewards. When we visualize our agent policies during intermediate train steps, one out of the three predators often wanders off in one direction without cooperating with other predators to catch the prey. In the future, we hope to improve training efficiency by incentivizing cooperation between agents during training. Although we tried adding a discounted team reward to each agent's individual rewards, it did not lead to performance increase. As discussed in paper [7], a latent variable inducing nonzero mutual information between actions can be added to maximize correlation of agents' behaviors. We plan on experimenting with similar strategies to find out whether there will be increases in agent cooperation and overall performance.

Another direction we hope to explore in the future is min-max optimization of agent policies. As discussed in paper [8], agents' policies can be very sensitive to their learning partner's policy, and for the purpose of learning robust policies, it is helpful to update policies under the assumption that all other agents are acting adversarially. This approach may lead to better and more stable performance when we are simultaneously training both predators and prey in an adversarial manner.

VII. Conclusion

In this paper, we proposed Distributed Genetic MADDPG, a multi-agent deep deterministic policy gradient algorithm that utilizes distributed computing to train multiple workers in parallel and periodically sync best performing critic networks. Empirically, Distributed Genetic MADDPG consistently outperforms MADDPG in multiple training settings under the Simple Tag environment.

At the same time, we also explored other methods of distributing MADDPG using data and model parallelism. Due to the nature of Reinforcement Learning problems, the environments and training networks suitable for the scale of our project are generally simple and small in size, while requiring a great amount of training timesteps for convergence. In this situation, distributed computing does not bring about significant runtime speedups and often suffers from huge communication overhead. However, for tackling MARL problems at greater scale, it will be interesting to examine whether our proposed distributed MADDPG algorithms will improve training efficiency significantly.

VIII. References

1. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (n.d.). *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. https://doi.org/10.48550/arXiv.1706.02275
2. Buşoniu, L., Babuška, R., & De Schutter, B. (2010). Multi-agent Reinforcement Learning: An overview. *Innovations in Multi-Agent Systems and Applications - 1*, 183–221. https://doi.org/10.1007/978-3-642-14435-6_7
3. Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., & Spanò, S. (2021). Multi-agent Reinforcement Learning: A review of challenges and applications. *Applied Sciences*, *11*(11), 4948. https://doi.org/10.3390/app11114948
4. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., & Kavukcuoglu, K. (2018, June 28). *Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures*. arXiv.org. Retrieved February 19, 2022, from https://arxiv.org/abs/1802.01561
5. *Blog - new environments and algorithm for multi-agent RL*. Autonomous Agents Research Group. (n.d.). Retrieved February 19, 2022, from https://agents.inf.ed.ac.uk/blog/new-environments-algorithm-multiagent-rl/
6. Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., … Petersen, S. (2016, December 13). *Deepmind lab*. arXiv.org. Retrieved February 19, 2022, from https://arxiv.org/abs/1612.03801
7. Woojun Kim, Whiyoung Jung, Myungsik Cho, and Youngchul Sung. A maximum mutual information framework for multi-agent reinforcement learning. arXiv preprint arXiv:2006.02732, 2020.
8. Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., & Russell, S. (2019). Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 4213-4220. https://doi.org/10.1609/aaai.v33i01.33014213

Appendix

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Hyperparameter Set:

actor network learning rate: $1e^{-3}$
critic network learning rate: $1e^{-4}$
ϵ: 0.1
γ: 0.95
τ: 0.01
noise rate: 0.1
buffer size: $5e^5$
batch size: 256
soft update rate: 10
critic sync rate: 5000

Implementation of Distributed MADDPG and Distributed Genetic MADDPG can be found at *https://github.com/xindazz/DistributedMADDPG*.