Project 3: Multi-Agent Proximal Policy Optimization for Overcooked Game

Jiawei Zhan, *Georgia Institute of Technology*

April 20, 2024

GitHub last commit hash: **672f755577b2bee5d1a0e11d2c05ab62876ec385**

1 Project Overview

In a multi-agent Overcooked environment, two chefs in a resturant need to collaborate to cook onion soups. My goal is to come up with a reinforcement learning method to maximaize the number of soups delivered within an episode on a variety of layouts. To solve this problem, I proposed to use multi-agent Proximal Policy Optimization (MAPPO) with centralizing training and decentralizing execution techinics. I also adopted reward shapeing and multiprocessing. I showed that such implementation achieve high effectivess in the Overcooked game.

2 Theory

2.1 Central Mechanism: Proximal Policy Optimization

Proximal Policy Optimization (PPO)[1] is trying to solve a question: how can we take the biggest possible improvement step on a policy without stepping so far from the old, using the data that are currently available. This's achieved by clipping in the objective function to prevent the new policy to get far from the old policy. Specifically, PPO update polices via

$$\theta_{k+1} = \text{argmax}_\theta E_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)], \tag{1}$$

by taking multiple steps of gradient decent. Here L is given by:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right), \tag{2}$$

where hyperparameter ϵ controls how far away the new policy is allowed to go from the old. The advantage function $A^{\pi_\theta}(s,a)$ is approximated via Generalized Advantage Estimation[2]:

$$A_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\left(r_{t+l} + \gamma V^{\pi,\gamma}(s_{t+l+1}) - V^{\pi,\gamma}(s_{t+l})\right). \tag{3}$$

Within the advantage function is the value function $V^{\pi,\gamma}(s_t)$ approximated by another model, which is fitted by regression on mean-squared error:

$$\psi = \text{argmin}_{\phi}||V_{\psi}^{\pi,\gamma}(s_t) - \sum_{l=0}^{\infty}\gamma^l r_{t+l}||^2, \tag{4}$$

through gradient descent algorithm.

2.2 Centralizing Training and Decentralizing Execution

The overcooked game is a cooperative, multi-agent games that may be challenging for traditional reinforcement learning approaches such as Q-Learning or policy gradien. Here, I decided to adopt centralizing training and decentralizing execution technics[3] to achieve cooperation amoung multiple-agent in the game. More concretely, for a game with N agents with policies $\boldsymbol{\pi} = \{\pi_1, ..., \pi_N\}$ parameterized by $\boldsymbol{\theta} = \{\theta_1, ..., \theta_N\}$. The gradient of expected return $J(\theta_i) = E[R_i]$ for agent i is:

$$\nabla_{\theta_i}J(\theta_i) = E_{o_i,a_i\sim\pi_i}[\nabla_{\theta_i}\text{log}\pi_i(a_i|o_i)A_i^{\boldsymbol{\pi}}(\mathbf{x}, a1, a2, ..., a_N)], \tag{5}$$

where o_i is the local observation of agent i, and $\mathbf{x}$ is the global state of the game, including $\{[\textbf{player i features}, \textbf{player i position}]\}$. In other word, during the training procedure, the advantage function is approximated given the full information of the game and agents' strategies are evolved based on that. Hopefully, agents can learn to coorperate and make decisions based on their local observation.

3 Implementation Details

The backbone PPO follows the implementation in the original PPO paper[1], utilizing a 4-layer policy model with 96, 64, 64, and 6 neurons, and a separate 4-layer value function with 48, 64, 64, and 1 neurons. This section discusses additional implementation aspects, including centralized training and decentralized execution, and outlines strategies to tackle challenges associated with sparse rewards and time-intensive online data collection.

3.1 Shared Policy Model and Symmetric Value Function

The idea of shared policy is motivated by the pursuit of AGI in the whold AI field as well as the intuition of 'I should be able to know what to do if I'm in his position.' In that case, both agent in the Overcooked game will share a single policy game. And two agents are trying to optimized the unified strategy together:

$$\theta_{k+1} = \text{argmax}_{\theta}E_i\left[E_{s_i,a_i\sim\pi_{\theta_k}}[L(o_i, a_i, \theta_k, \theta)]\right] \tag{6}$$

, where i is the agents' index.

Additionally, the centralized value function should be invarient to the switch of two agents, since the agents should be indistinguishable and the global state should only depends on where the agents are located in as well what they are doing, instead who is doing that. Motivated by this the invarient value function can be expressed by:

$$V_\psi(\mathbf{x}) = \boldsymbol{O}\{f_\psi(o_i)\}, \tag{7}$$

where $\boldsymbol{O}$ can be any invarient operator, such as mean and plus. In my implementation, I choose to use plus operator.

3.2 Reward Shaping

In the default setting, agents receive rewards only if they successfully deliver onion soup, which requires a sequence of actions including placing onions in a pot and picking up the soup. To promote efficient learning of these prerequisites for soup-making, rewarding intermediate achievements could be beneficial. This strategy helps overcome the challenge of sparse rewards throughout the game, potentially leading to highly inefficient exploration processes. The reward formula I employ is: $r = 20 \times \text{soup delivery} + 3 \times \text{PLACEMENT IN POT REWARD} + 3 \times \text{DISH PICKUP REWARD} + 5 \times \text{SOUP PICKUP REWARD}$ The coefficients in this formula represent the cumulative score all agents receive when one successfully completes an action. Due to the centralized training techniques I use, the rewards are shared among all agents.

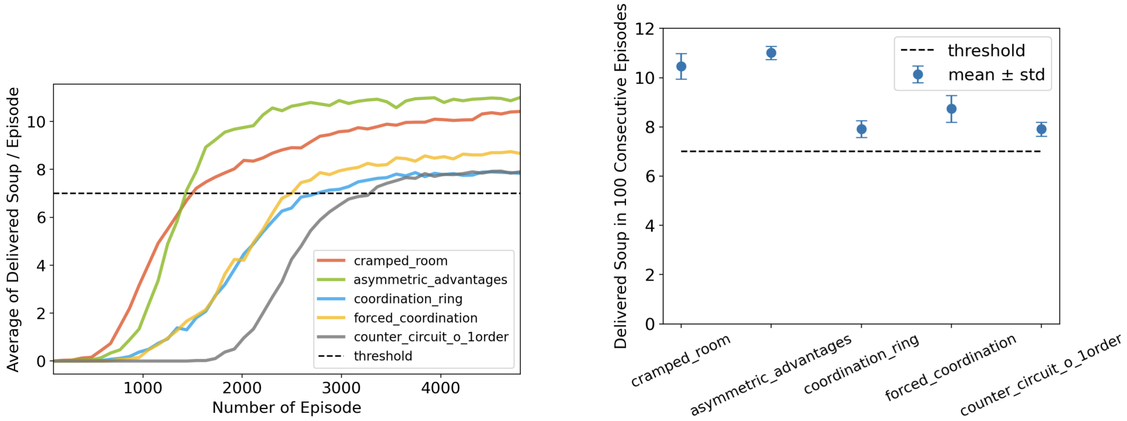
3.3 Efficient Exploration with Parallel Multiple Walkers

The exploration process significantly impacts the efficiency of learning in RL model training and can be time-consuming. To enhance efficiency, I have set up multiple independent environments allowing parallel exploration by creating several processes, each hosting an independent environment. These processes share the same policy model and value function. Gradients of weights are computed in parallel, then averaged and synchronized across processes. This setup enables all agents to explore the environments and collect data using the same strategy and value function.

4 Result

4.1 Efficiency and Effectiveness of MAPPO

By combining all technics mentioned above, I'm able to train MAPPO to deliver more than 7 onion soups for all 5 layouts in 400 step horizon. Fig.[1] shows the number of soups delivered per episode during the training procedure(a), as well as the reliability of trained model by plotting the mean and variance of occurance of delivering soup for 100 consecutive episodes(b). More than that, thanks to the parallel multi-walker exploration strategy enabled by multi-processing with 32 CPUs, the RL model can be trained within 5 minutes per layout.



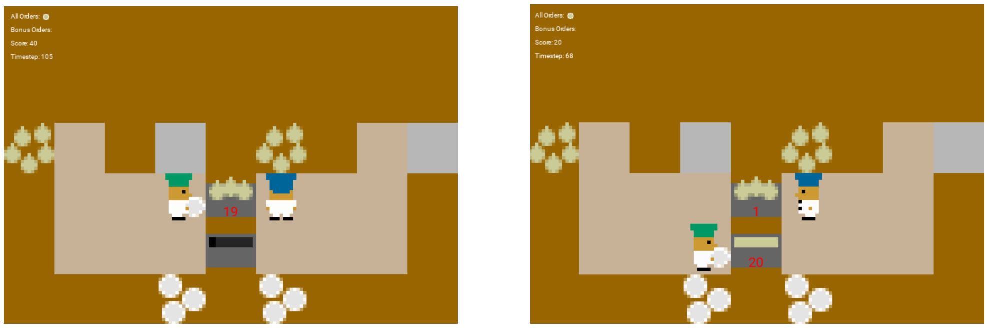
(a) Average delivered soup per episode during training.

(b) Average delivered soup in 100 consecutive episode(evaluation).

Figure 1: Efficiency and effectiveness of MAPPO in the Overcooked game.

4.2 Impact of Reward Shaping on Agent Behavior

Although the MAPPO model effectively learnd strategies to deliver 7 soups within 400 step horizon for each layout, there's some space for improvement. For example, with the default configuration mentioned in Sec.3, agents are not maximizing the utilization of multiple pots in some layouts, such as in the 'assymetric advantages' layout showed in Fig.[2.a]. This's probably because the constant reward shaping used during the training encourage picking up dished as much as possible, so that the agent is tracked at a suboptimal strategy.



(a) When using the default reward (non-annealing) mentioned in Sec.3, Agents are waiting by the pot while cooking.

(b) Agents can learn to cook with both pots when reward shaping is reduced to 0 (annealing) over the course of training.

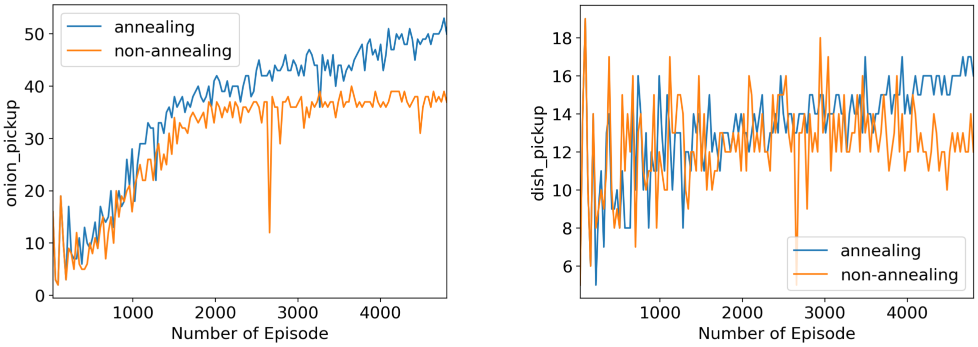
Figure 2: By annealing the reward shaping, the agents can learn to maximize the real goal: delivering soup.

To further improve the performance of agents on those layouts with multiple pots, I changed the reward shaping so that it will reduce to 0 over the couse of training (annealing).

Specifically, the reward is now:

$$r^{\text{annealing}} = 20 * \text{nsoup} + 0.5(\text{r\_shaped\_0} + \text{r\_shaped\_1})e^{-\text{nepisode}/10}. \tag{8}$$

Fig.[2.b] shows that with reward shaping annealing, agents now prioritize the main objectives—cooking and delivering soups—by picking up more onions rather than focusing on the less crucial, high-reward action of dish pickup. Fig.[3] illustrates these changes in behavior during training, highlighting the effectiveness of reward shaping.



(a) Agents would pick up more onion when using reward shaping annealing.

(b) Agents tend to pickup constant amount of dishes during training when usining non-annealing reward shaping.

Figure 3: By annealing the reward shaping, the agents can pick up more onions and cook more .

5 Discussion

I showed that MAPPO with centralizing training and decentralizing execution strategy achieves surprising effectiveness in cooperative, multi-agent Overcooked game. I also demonstrated the importance of reward structures in improving learning efficiency. Here are some additional explorations that I could possibly do in the future:

- Utilizing RNN for the policy and value functions.
- Robustness and adversarial testing.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[2] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[3] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020.