FoRL Spring 2023 Project Report: Adaptive Mechanism Design in Sequential Social Dilemmas

Guangda Ji
guanji@student.ethz.ch

Minxuan Qin
minqin@student.ethz.ch

Xuanang Lei
xualei@student.ethz.ch

Project TA: Vinzenz Thoma

Abstract

Social dilemmas are a type of multi-agent reinforcement learning (MARL) environment wherein agents face a difficult choice between acting in their self-interest or cooperating. Understanding social dilemmas is crucial as it offers valuable insights into the design of mechanisms for real-world scenarios. *Adaptive mechanism design* (AMD) is a MARL algorithm that has exhibited success in promoting cooperation within simple matrix games. This study aims to investigate the effectiveness of AMD in more complex games known as *sequential social dilemmas* (SSDs). SSDs simulate real-world scenarios with temporally extended decision-making processes, where elementary decisions do not directly reflect cooperativeness. In this project, we re-implement the AMD algorithm and evaluate its performance in two SSD environments, namely, *Wolfpack* and *Gathering*. Our results demonstrate that AMD is unable to effectively encourage cooperation in these complex games. The code for our project is accessible at `https://github.com/quantaji/AMD-SSD`.

1 Introduction

Social dilemmas describe the conflicts between individual and collective interests. Individuals' behavior usually follows the principle of self-interest, which will inevitably impair the interests of other people or groups. In the context of multi-agent reinforcement learning (MARL), each agent behaves myopically, and the system would end up in a bad equilibrium, where the total reward of all agents is not optimal. The concept of *mechanism design* emerges as an approach to alter systems in a way that promotes cooperation.

One notable algorithm in this domain is the *adaptive mechanism design* (AMD) proposed by Baumann et al.. In this algorithm, a *central planner* observes the states and actions of all agents and provides additional rewards to each agent, with higher rewards granted for cooperative actions. The effectiveness of this algorithm has been demonstrated in simple game environments, such as the *prisoner's dilemma* (PD).

However, real-world social dilemmas often extend temporally, where the cooperativeness is reflected in the overall policy rather than individual actions at each time step. This more complex scenario is referred to as a *sequential social dilemma* (SSD). Leibo et al. proposed two SSDs, namely *Wolfpack* and *Gathering*. Our primary objective is to investigate how the AMD algorithm promotes cooperation within these complex environments.

In this project, we aim to implement the AMD algorithm in the *Wolfpack* and *Gathering* SSDs. Our contributions can be summarized as follows:

- We re-implement the AMD algorithm using `ray.rllib`, an open-source industry-grade RL library. Unlike the original implementation limited to matrix games like the prisoner's dilemma, our implementation is compatible with any multi-agent environment. To the best of our knowledge, our code represents the first implementation to achieve this level of generality. We validate our implementation using the prisoner's dilemma and further modify the algorithm to accommodate complex environments.

- We provide open-source implementations of the *Wolfpack* and *Gathering* environments. To our knowledge, this is also the first publicly available implementation.
- Our experiments show negative results: AMD does not encourage cooperation in complex SSDs.

2 Related Works

In the context of RL algorithms that encourage cooperation, Foerster et al. first proposed *learning with opponent learning awareness* (LOLA). LOLA entails an agent being aware that its opponent is also undergoing parameter updates, and it updates with respect to the look-ahead parameter, $\Delta\theta_1 = \eta\nabla_1 V_1(\theta_1, \theta_2 + \Delta\theta_2)$ where $\Delta\theta_2 = \delta\nabla_2 V_2(\theta_1, \theta_2)$. The AMD algorithm [1] we implement in this project adapts a similar formulation, where a central planner knows the update of each individual agent. While LOLA has been demonstrated effective in matrix games, it fails to preserve stable fixed points when the LOLA update is turned off. Willi et al. proposed consistent LOLA (COLA) where the agent knows its opponent is also doing LOLA updates. The learning rule goes $\Delta\theta_1 = \eta\nabla_1 V_1(\theta_1, \theta_2 + \Delta\theta_2)$, $\Delta\theta_2 = \eta\nabla_2 V_2(\theta_1 + \Delta\theta_1, \theta_2)$. Instead of obtaining $\Delta\theta_{1,2}$ directly from gradients, COLA uses additional neural networks to approximate $\Delta\theta_{1,2}$.

In the context of sequential social dilemmas, Leibo et al. proposed the Wolfpack and Gathering environments. Wolfpack consists of two wolves and a prey, where the wolves can choose to capture the prey either together or individually. Gathering, on the other hand, is a game where two predators collect apples and can use a beam to eliminate opponents and prevent them from collecting apples. Additionally, Jaques et al. introduced two other SSDs: a public goods game *Cleanup*, and a public pool resource game *Harvest*.

3 Theory and Framework

In this section, we will provide a detailed explanation of our implementation of the AMD algorithm. We will begin by discussing the general settings. Then, in sections 3.1 and 3.2, we will introduce the update rules for agents and the central planner, respectively. Lastly, in section 3.3, we will describe our efforts to improve the AMD algorithm.

In AMD, there are agents and an additional agent, *central planner*. Each agent with index i has an observation space $\mathcal{O}_i$ and an action space $\mathcal{A}_i$. The central planner observes a group of agents (with index $i = 1, \ldots, N$) that we wish to cooperate with. We let the central planner not able to observe all the agents because it is possible that agents fall into opposite groups, for example, groups of wolves and prey, and forcing the prey to give up its life to learn to cooperate makes no sense. Therefore, the central planner only functions within a custom group of agents among whom we desire cooperation.

The central planner observes the observations and actions of the agents in the group. In strict mathematical language, the observation space of the central planner is $\mathcal{O}^p = \mathcal{C}^a \times \mathcal{A}^a \times \mathcal{O}^a$, a denotes it comes from *agents*. $\mathcal{C}^a = \mathcal{C}_1 \times \cdots \times \mathcal{C}_N$ is the presence table of all agents. Here $\mathcal{C}_i = \{0, 1\}$. We introduce this term because in a practical environment, some agents can get killed and be absent at some time, and the central planner has to take this into account. $\mathcal{A}^p = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$ is the actions of the agents. $\mathcal{O}^a = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$ is the observations of the agents. When some agent is absent, its action and observation are filled with zeros. If the environment offers a more compact state $\mathcal{S}$ that gives information about all agents, our implementation also allows the central planner to switch to this observation, and $\mathcal{O}^a = S$ in this case.

At each time step, the central planner observes $o_t^p \in \mathcal{O}^p$ and delivers reward $\mathbf{r}_t^p = \mathbf{r}^p(o_t^p, \theta_p) = [r_1^p(o_t^p, \theta_p), \ldots, r_N^p(o_t^p, \theta_p)]^\top \in \mathbb{R}^N$. θ_p is the parameter of the central planner. In this project, $\mathbf{r}^p$ is deterministic, as it is in the original paper [1]. In the further, this can also be extended to a random reward, where the central planner learns a policy $\pi^p(\mathbf{r}^p | o_t^p, \theta_p)$

3.1 Update of Cooperating Agents

For each agent i in the group we would like them to cooperate, we assume it does some kind of policy gradient update with policy $\pi_i(a_i|o_i,\theta_i)$. It receives the actual reward $r_{i,t}$ from the environment, and also the reward from central planner $r^p_{i,t}$. Since the policy gradient is linear to the rewards, the actual policy gradient is also the sum of two policy gradient terms,

$$\Delta\theta_i = \eta_i \nabla_i J_i + \eta_i \nabla_i J^p_i. \tag{1}$$

The first part $\nabla_i J_i$ is the policy gradient of the back-end RL algorithm, it can be PPO or A3C, even with the help of a neural parameterized critic. We treat it as a black box and leave it unchanged. For the second part of the gradient, we do vanilla REINFORCE update,

$$\nabla_i J^p_i = \mathbb{E}\left[\left(\sum_{t=0}^{\infty} \nabla_i \log \pi_i(a_{i,t}|o_{i,t},\theta_i)\right) \times R^p_i \middle| s_0 \sim \mu_0, \substack{a_t\sim\pi(\cdot|s_t),\\ s_{t+1}\sim P(\cdot|s_t,a_t),\\ r_t=r(s_t,a_t),} \forall\, t \geq 0\right], \tag{2}$$

where $R^p_i = \sum_{t=0}^{\infty} \gamma^t r^p_{i,t}(o^p_{i,t},\theta_p)$ is the discounted accumulated rewards. For simplicity, we will abbreviate sampling trajectory with $\mathbb{E}_\tau\left[\cdot|\tau\right]$. From eq. (2) we know $\nabla_i J^p_i$ is a function of θ_p.

To sum it up, the (minimization) loss for each agent is

$$\mathcal{L}_i = \mathcal{L}_{i,\text{backend}} + \mathcal{L}_{i,\text{amd}}, \text{ where } \mathcal{L}_{i,\text{amd}} = -\mathbb{E}_\tau\left[\left(\sum_{t=0}^{\infty} \log \pi_i(a_{i,t}|o_{i,t})\right) \times R^p_i\right], \tag{3}$$

and $\mathcal{L}_{i,\text{backend}} = -J_i$ is the loss of the back-end algorithm.

3.2 Update of Central Planner

For the central planner, its update should maximize the total expected reward (or value) $V(\theta_{1:N}) := \sum_{i=1}^N V_i(\theta_{1:N})$. An intuitive idea is to maximize the value of the next time step $V(\theta_1+\Delta\theta_1,\ldots,\theta_N+\Delta\theta_N)$, where each $\Delta\theta_i$ depends on θ_p. To be clear from the beginning, we use the minimization notation and set the corresponding loss to be $\mathcal{L}^p(\theta_p) = -V(\theta_1+\Delta\theta_1,\ldots,\theta_N+\Delta\theta_N)$. Then, we perform first-order Taylor expansion approximation, $\mathcal{L}^p(\theta_p) = -\sum_{i=1}^N V_i(\theta_1+\Delta\theta_1,\ldots,\theta_N+\Delta\theta_N) \approx -\sum_{i,j=1}^N \nabla_j V_i(\theta_1,\ldots,\theta_N)^\top \Delta\theta_j - V(\theta_1,\ldots,\theta_N)$. We remove the last term because it does not depend on θ_p, then

$$\mathcal{L}^p(\theta_p) = -\sum_{i=1}^N \sum_{j=1}^N \nabla_j V_i(\theta_1,\ldots,\theta_N)^\top \Delta\theta_j \tag{4}$$

Here is where our project differs from the original paper. In [1], they chose a single-step prisoner dilemma, where V_i can be exactly calculated as a function of each agent's policy parameter. However, this is not the case for general environments. Therefore, we have to approximate $\nabla_j V_i$ with policy gradient (In the original paper, they refer to this approach as "estimate". They performed experiments on this approach, but this was not their focus.). The policy gradient approximation for $\nabla_j V_i$ is

$$\nabla_j V_i \approx \mathbb{E}_\tau\left[\sum_{t=0}^{\infty} \nabla_j \log \pi_j(a_{j,t}|o_{j,t})\Psi_{i,t} \middle| \tau\right], \tag{5}$$

where $\Psi_{i,t}$ can be all kinds of estimator, e.g., $A^{\pi,\gamma}(o_{j,t},a_{j,t})$ the biased GAE estimator, and $R_i = \sum_{t=0}^{\infty} r_{i,t}$ the total accumulated rewards. Then, $\nabla_j V \approx \mathbb{E}_\tau\left[\sum_t \nabla_j \log \pi_{j,t}\Psi_t|\tau\right]$, where $\Psi_t = \sum_i \Psi_{i,t}$ is the sum of advantage of all cooperating agents (we call $\Psi_{i,t}$ as *advantage* for simplicity.).

We add eqs. (1) and (5) into the loss defined in eq. (4), then it becomes

$$\mathcal{L}^p = -\sum_{i=1}^N \mathbb{E}_\tau\left[\sum_{t=0}^{\infty} \nabla_i \log \pi_i(a_{i,t}|o_{i,t})\Psi_t \middle| \tau\right]^\top \left(\eta_i \nabla_i J_i + \eta_i \nabla_i J^p_i\right). \tag{6}$$

Since only $\nabla_i J_i^p$ is related to θ_p, we can remove $\nabla_i J_i$ from this loss. Therefore,

$$\begin{aligned}\mathcal{L}^p &= -\sum_{i=1}^{N} \eta_i \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \nabla_i \log \pi_i(a_{i,t}|o_{i,t}) \Psi_t \right]^\top \nabla_i J_i^p \\ &= -\sum_{i=1}^{N} \eta_i \left\{ \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \nabla_i \log \pi_i(a_{i,t}|o_{i,t}) \Psi_t \right]^\top \left(\sum_{t'=0}^{\infty} \nabla_i \log \pi_i(a_{i,t'}|o_{i,t'}) \right) \right\} \times R_i^p.\end{aligned} \tag{7}$$

In our implementation, we define an intermediate variable, which we call *awareness*,

$$w_{i,t} := \mathbb{E}_\tau \left[\sum_{t'=0}^{\infty} \nabla_i \log \pi_i(a_{i,t'}|o_{i,t'}) \Psi_{t'} \right]^\top \nabla_i \log \pi_i(a_{i,t}|o_{i,t}), \tag{8}$$

because it contains the updated (next-step) information of each agent. This quantity is pre-calculated before the learning step. Then the loss for the central planner becomes a linear combination of all rewards,

$$\mathcal{L}^p := -\sum_{i=1}^{N} \eta_i \sum_{t=0}^{\infty} w_{i,t} R_i^p(\theta_p). \tag{9}$$

Except for this main loss, we can also add a regularization term $\|\mathbf{r}_t^p\|_q$, which is a q-norm, then the total minimization loss is

$$\mathcal{L}_{\text{central planner}} = \mathcal{L}^p + \alpha_{\text{reg}} \mathcal{L}_{\text{reg}} = -\sum_{i=1}^{N} \eta_i \sum_{t=0}^{\infty} w_{i,t} R_i^p(\theta_p) + \alpha_{\text{reg}} \sum_{t=0}^{\infty} \|\mathbf{r}_t^p\|_q. \tag{10}$$

Remark:

1. All environments may end or be truncated by actual implementation; the summation to infinity is only an abbreviation for this.
2. It is possible that some agent i is absent at time step t. Then the awareness is zero, $w_{i,t} = 0$, and also it will not appear in the sum of advantage, $\Psi_t = \sum_{j \neq i} \Psi_{j,t}$.
3. In actual implementation, the expectation is replaced with expectation w.r.t. the whole batch.
4. We use advantage function $\Psi_{i,t}$ in the approximation of real value function V_i. In actual implementation, this advantage might come from the critic. Therefore, we have to strictly separate the update of the critic from the central planner's rewards, so that the critic gives an unbiased estimation of the real value function.
5. It might seem non-intuitive that the central planner's loss is linearly dependent on its reward since originally we want to maximize the total value function, but this can be inferred from the Taylor expansion because $\Delta\theta_i$ are all linearly dependent on central planner's reward.
6. Awareness eq. (8) involves taking gradient w.r.t. agent's parameters, $\nabla_i \log \pi_i$. In actual neural networks, calculating this can be sophisticated and memory-consuming. Therefore, we offer another option, assuming softmax parameterization $\pi(a|s) = \frac{\exp(\theta_{a,s})}{\sum_{a'} \exp(\theta_{a',s})}$. We denote these two options as *neural* and *softmax*. (See appendix A)

3.3 Modification: Interpreting Rewards as Q-value Adjustment

In the current setting of AMD, a drawback is that in both agents' and the central planner's loss, all rewards appear in the accumulated form $R_i^p(\theta_p)$. Then each agent can only perceive the accumulated reward $R_i^p(\theta_p)$, but it cannot tell what is the central planner's reward at each time step. This means that the central planner's evaluation of the cooperativeness of the agent in an episode is only reflected by this single scalar, which is very coarse, but in actual cases, an agent can behave both cooperatively or defectively in a single episode.

Therefore, it would be nice if eq. (3) is in the form of $-\sum_t \log \pi_i(a_{i,t}|o_{i,t}) \times r_{i,t}^p$ and eq. (9) is in the form of $-\sum_t w_{i,t} \times r_{i,t}^p$, so that the multiplication comes before summation. Then both agents and the central planner know what is the planner's reward at each time step, and then they can do a more fine-grained update. To do this, we need a new interpretation of $r^p_{i,t}$ to make it logically correct.

From policy gradient theorem, we know that $\nabla_i J_i = \mathbb{E}_\tau \left[\nabla_i \log \pi_{i,t} Q_i(o_{i,t}, a_{i,t})\right]$, and we think of central planner delivers an adjustment $q_i(o_{p,t};\theta_p)$ of $Q(o_{i,t}, a_{i,t})$ to enhance cooperation,

$$\begin{aligned}\nabla_i J_{i,\text{new}} &= \mathbb{E}_\tau \left[\nabla_i \log \pi_{i,t} \left(Q_i(o_{i,t}, a_{i,t}) + q_i(o_{p,t};\theta_p)\right)\right] \\ &= \mathbb{E}_\tau \left[\nabla_i \log \pi_{i,t} Q_i(o_{i,t}, a_{i,t})\right] + \mathbb{E}_\tau \left[\nabla_i \log \pi_{i,t} q_i(o_{p,t};\theta_p)\right] \\ &= -\nabla_i \mathcal{L}_{i,\text{backend}} - \nabla_i \mathcal{L}_{i,\text{amd,new}}.\end{aligned}$$

In this interpretation, the $\mathcal{L}_{i,\text{amd}}$ in eq. (3) becomes

$$\mathcal{L}_{i,\text{amd,new}} = -\mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \nabla_i \log \pi_i(a_{i,t}|o_{i,t}) \times q^p_i(o_{p,t};\theta_p)\right], \tag{11}$$

and the $\mathcal{L}_p$ in eq. (9) becomes

$$\mathcal{L}_{p,\text{new}} = -\sum_{i=1}^{N} \eta_i \sum_{t=0}^{\infty} w_{i,t} q^p_i(o_{p,t};\theta_p). \tag{12}$$

We have now logically justified our modified version of AMD.
Remark: It might be physically unimaginable that the central planner delivers q-value adjustment, instead of actual rewards. However, the central planner in this case can be viewed as a reviewer that gives comments on how cooperative each agent's action is at each time step so that the agent can adjust their evaluation of their action.

4 Sequential Social Dilemma Environments: Wolfpack and Gathering

Both environments are grid worlds, and agents are represented as dots on the grid. All agents have an observation space $\mathcal{O} = \mathbb{R}^{16\times 21\times 3}$, which is an RGB image as shown in Figure 1, and have a basic action set $\mathcal{A} = \{$step forward, step right, step backward, step left, stand still, turn clockwise and turn counterclockwise$\}$. Black blocks are empty areas where agents can traverse, while grey blocks are barriers. The dark grey block beside each agent denotes its head orientation. We choose the `pettingzoo` library for its popularity and good compatibility with `rllib`.

4.1 Wolfpack

The *Wolfpack* game involves two predators, `wolf_1` (blue) and `wolf_2` (red) chasing the `prey` (orange).The prey is considered caught if it overlaps with either of the wolves, and the episode ends. To incentivize hunting, a starvation punishment of $r_{\text{starve}} = -0.05$ is assigned to the predators, while a reward of r_{live} is given to the prey if it survives the current time step. If a wolf captures the prey and the other wolf is within the prey an L_1 radius of $R = 6$, then both wolves are rewarded $r_{\text{team}} = 5$, otherwise, only the winning wolf will be rewarded $r_{\text{lone}} = 1$.

A cooperative policy would be: when a wolf observes the prey, it waits inside the cooperative radius R for the other wolf to join the capture together. On the other hand, a defective policy involves the wolf directly chasing the prey regardless of the other wolf. Cooperativeness is quantified as the percentage of episodes where wolves capture the prey together.

4.2 Gathering

In the *Gathering* game, the goal for the predators (blue and red) is to collect apples (green, maximum number is 3 in our setting). A reward of $r_{\text{apple}} = 1$ is assigned when a predator collects the apple, and it disappears in the grid world, and re-generates after $N_{\text{apple}} = 3$ frames in a random location. Similar to Wolfpack, predators get a starvation punishment of $r_{\text{starve}} = -0.05$ if it gets nothing in this round. Besides the basic actions, predators can also emit a beam (yellow) to wound their opponent. This will reduce one blood of the opponent. If a predator is out of blood, it will be removed from the environment and respawn after N_{tag} frames. In our current setting, the total blood of all agents is 1.

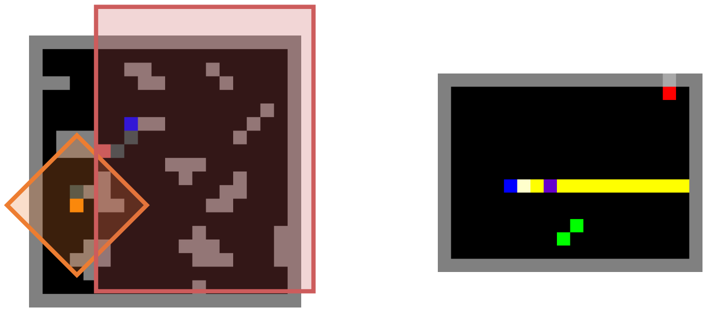


Figure 1: **Wolfpack** (left) and **Gathering** (right) environments. All agents have observation space of RGB image of size $16 \times 21 \times 3$, shown in the pink rectangle in the left figure. Predators are marked as blue and red. In Wolfpack, the orange grid denotes the prey. If both predators are within a capture radius (orange diamond shown on the left) when the prey is captured, they will be treated as cooperative and rewarded more. In Gathering, predators can fire yellow beams at their head orientation. Agents or apples hit by the beam will turn purple.

A defective policy would be aggressively emitting beams to kill its opponent and have all the apples to themselves. Conversely, a cooperative policy would never emit beams. Cooperativeness in this game is quantified by the percentage of time that each agent is alive in the game.

Remark 1 Note that cooperation tends to occur naturally when resources are abundant, such as when the resource parameter R is large in *Wolfpack* or when N_{apple} is large and N_{tag} is small in *Gathering*. However, our primary focus is to examine whether AMD will still encourage cooperation when resources become scarce.
Remark 2 Besides, we also implement PD for validation since it is easy to train. We use the same setting as [1], where the payoff matrix is $(C, C) = (3, 3)$, $(C, D) = (0, 4)$ and $(D, D) = (1, 1)$.

5 Implementation Details and Results

As previously mentioned, AMD is compatible with any policy gradient algorithm, we use vanilla actor-critic and PPO as back-end algorithms. The former is used for debugging while the latter one is for actual experiments.

For PD, the policy is either (1) a linear layer directly from input to output or (2) an MLP with two hidden layers of width 32 and ReLU activation. The central planner's policy is an MLP with two hidden layers of width 128 and ReLU activation. The batch size is 1024 and the learning rate is 10^{-4}.

The policy for *Wolfpack* and *Gathering* is a neural network that consists of (1) a convolution layer of kernel 3, stride 1 and 6 output channels (2) flattened vectors are passed to two fully-connected layers of width 32 with ReLU activation, (3) a 128-dim LSTM of 32-timestep window size (4) a linear layer for output of critic and actor's logits. LSTM is used to promote purposeful actions. Purely convolution networks fail to learn an effective policy. (See appendix C). The central planner's policy is a two-hidden layer MLP with widths of 256 and 128, followed by an LSTM of size 32. The batch size is 98304 and learning rate is 10^{-4}. The training takes in total 14 million timesteps. Each PPO update contains 10 SGD updates with $1/4$ the batch size.

The central planner's reward is deterministic. It is the $tanh$ of output logits multiplied by a factor of $R_{\max}$. In PD, $R_{\max} = 3.0$, while in *Wolfpack* and *Gathering*, $R_{\max} = 0.05$.

We choose `ray.rllib` as our reinforcement learning framework because it offers comprehensive RL algorithms. We fix our version to be `2.3.1` (due to rapid change in `ray`'s API, migrating to the next major version needs additional effort.). Also, we chose `Pytorch` as our deep learning framework because we are more familiar with it, and the original paper was implemented in `tensorflow` so it is more challenging. All of our experiments are run on an RTX3090 GPU of Euler.

5.1 Prisoner's Dilemma

We experiment on the AMD algorithm with both simple (*linear*) and complex (*MLP*) policy models, employing direct (referred to as *neural*) as well as *softmax* parameterization. Across all cases, while PPO converged to a defective Nash equilibrium, AMD successfully promoted cooperation (see Figure

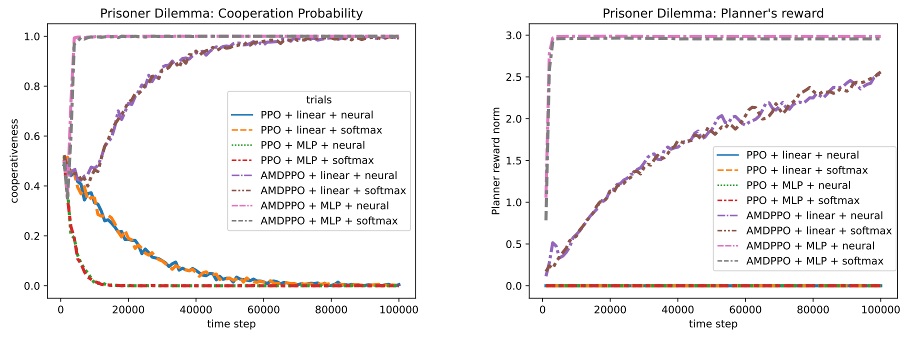


Figure 2: Training results of Prisoner Dilemma (PD). This figure **left** shows the probability of choosing cooperative action, while the **right** figure shows the reward delivered by the central planner. PPO/AMDPPO denotes the training algorithm. Linear/MLP denotes the agent's policy model. Neural/softmax denotes the parameter assumption in the AMD algorithm.

2). Notably, the MLP model converges faster, likely due to the adaptive nature of neural networks. These results affirm the effectiveness of our implementation. Moreover, we observed an increase in the cooperation probability, from 61.3% reported in the original paper to 100%. We attribute this to the increase in batch size, from 1 to 1024.

We observed an interesting behavior where the central planner consistently strives to the maximum reward $R_{\max}$, even when L_1 or L_2 regularization is applied. However, if the regularization factor is set too high, the central planner provides no reward, leading the system back to the Nash equilibrium. This behavior can be attributed to the linearity between the loss function eq. (9) and the reward. The loss function only achieves its minimum when r^p reaches $R_{\max}$. This drawback arises from the mismatch between the linear loss and the reward regularization term and roots from the first-order approximation in eq. (9). We leave it as a further direction to reconcile the main loss and the regularization.

5.2 Wolfpack and Gathering

The top left of fig. 3 demonstrates the effectiveness of the training process. Even the original PPO algorithm significantly reduced the average episode length from 200 to nearly 50. This indicates that the wolves are actively searching for prey rather than moving randomly. However, training using the AMD algorithm with accumulated reward (labeled as *R* in fig. 3) was unsuccessful. Training also failed when applying the AMD loss term later in the training stage. These results led us to explore the Q-value adjustment version of AMD described in section 3.3. Trials using this approach (denoted as **Q** in fig. 3) were compatible with PPO and did not result in training failure. In our experiments, trials with AMD do achieve higher rewards (fig. 3 top right). However, cooperativeness is indistinguishably the same for AMD and PPO (fig. 3 bottom left). Therefore, the difference in rewards can be attributed to randomness across trials. The results of Gathering (fig. 3 bottom right) also confirm the lack of effectiveness of AMD.

However, it is noteworthy that we also experimented with a variant of the model that excludes LSTM and utilizes only convolution layers. Naturally, this model yields poor performance. However, by incorporating the AMD term during training, we were able to improve its performance significantly (figs. 5 and 6). This suggests that while PPO may already be a sufficiently effective algorithm, AMD demonstrates its capabilities primarily when PPO falls short.

6 Discussion

Our experiment shows that AMD does not enhance cooperation in sequential social dilemmas. One possible reason for this could be the challenge of condensing the nuanced evaluation of cooperativeness into a single scalar reward, especially in complex real-world settings. The limited analytical abilities of the central planner may also contribute to this outcome. Furthermore, the effectiveness of PPO, which serves as a strong baseline, could overshadow any potential benefits of AMD. Nevertheless, our negative findings are valuable for future research.

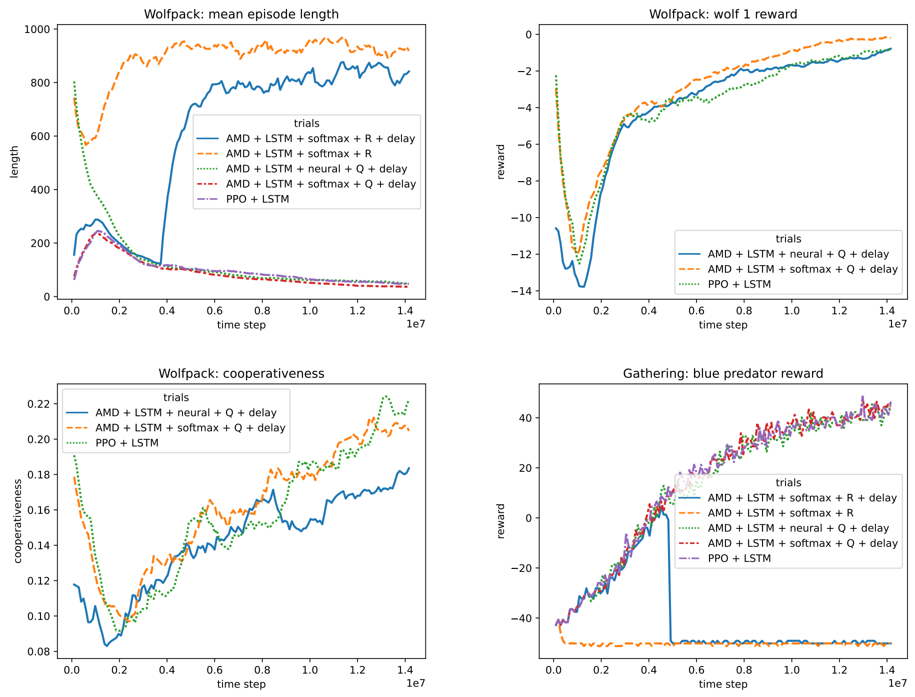


Figure 3: Training results of **Wolfpack** (Top left, top right and bottom left) and **Gathering** (bottom right). PPO/AMDPPO denotes the training algorithm. Linear/MLP denotes the agent's policy model. **Neural/softmax** denotes the parameter assumption in the AMD algorithm. **Q / R** denotes whether Q-adjustment or accumulated reward is used in the AMD algorithm. **Delay** means the AMD term is applied in the middle stage of training.

Regarding the coding aspect, none of our team members have prior experience in RL training. We encountered significant difficulties in comprehending the intricacies of the `ray.rllib` code base, consuming a substantial portion of our time. Despite these challenges, we successfully implemented the AMD algorithm, which exhibits generalizability to arbitrary multi-agent environments. We are satisfied with our accomplishments.

7 Conclusion

In conclusion, our project aimed to investigate the effectiveness of the adaptive mechanism design (AMD) algorithm in promoting cooperation within complex sequential social dilemma (SSD) environments, and our experiments revealed negative results, indicating that AMD does not encourage cooperation in such complex scenarios.

8 Contribution

- Guangda Ji implemented the main algorithm of AMD in `ray.rllib` and *Wolfpack* environment. He also conducted most of the experiments and writes the majority of this report.
- Minxuan Qin implemented the *Gathering* environment, ran some experiments under it, and helped by finishing up the poster and report.
- Xuanang Lei investigated the original implementation of AMD and gave valuable suggestions. He also made the majority part of the poster.

References

[1] Tobias Baumann, Thore Graepel, and John Shawe-Taylor. Adaptive mechanism design: Learning to promote cooperation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[2] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[3] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*, pages 3040–3049. PMLR, 2019.

[4] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

[5] Timon Willi, Alistair Hp Letcher, Johannes Treutlein, and Jakob Foerster. Cola: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, pages 23804–23831. PMLR, 2022.

A Choice of Parameterization

The calculation of awareness (eq. (8)) involves taking gradient w.r.t. agent's parameters, $\nabla \log \pi$. In our implementation, we offer two options, `param_assumption="neural|softmax"`. Option `neural` means gradient is calculated w.r.t. network parameter, and `softmax` means $\pi(a|s)$ are assumed to be the softmax of pseudo parameter, $\pi(a|s) = \frac{\exp(\theta_{a,s})}{\sum_{a'} \exp(\theta_{a',s})}$. Then the gradient is

$$\frac{\partial \log \pi(a|s)}{\partial \theta_{a',s'}} = \mathbb{1}\{s = s', a = a'\} - \pi(a'|s)\mathbb{1}\{s = s'\}, \tag{13}$$

and the awareness is,

$$\begin{aligned} w_{i,t} &= \sum_{t',a',s'} \Psi_{t'} \mathbb{1}\{s_{t'} = s'\} \left(\mathbb{1}\{a_{t'} = a'\} - \pi(a'|s_{t'})\right) \times \mathbb{1}\{s_t = s'\} \left(\mathbb{1}\{a_t = a'\} - \pi(a'|s_t)\right) \\ &= \sum_{t',a'} \Psi_{t'} \mathbb{1}\{s_{t'} = s_t\} \left(\mathbb{1}\{a_{t'} = a'\} - \pi(a'|s_{t'})\right) \left(\mathbb{1}\{a_t = a'\} - \pi(a'|s_t)\right). \end{aligned}$$

We denote $\mathbf{g}_t = \mathbf{E}_{a_t} - \pi(\cdot|s_t) \in \mathbb{R}^{|\mathcal{A}|}$. Then, the awareness is,

$$w_{i,t} = \sum_{t'} \Psi_{t'} \mathbb{1}\{s_{t'} = s_t\} \mathbf{g}_{t'}^\top \mathbf{g}_t. \tag{14}$$

Note that, even in this simplified softmax parameter assumption, computing all the awareness requires more than $O(T^3)$) time complexity and $O(T^2)$ memory. Therefore, AMD does not scale very well in this assumption.

B Central Planner's reward in both games

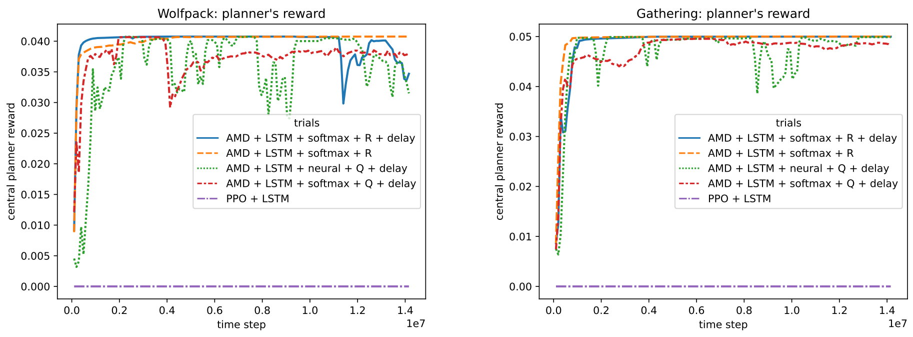


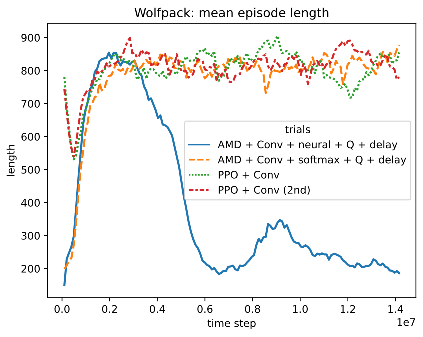
Figure 4: Central planner's reward of both games.

C Training results with only Convolution neural network

Our pure convolution network model consists of

- Conv Layer of kernel 4, stride 2, 16 output channels,
- Conv Layer of kernel 4, stride 2, 32 output channels,
- Conv Layer of kernel (4, 6), stride 1, 64 output channels,
- two fully connected layer of width 32 and ReLU activation,
- a linear layer to critic and policy's logits.

All of our convolution network models fails to learn a working policy. Either the episode length is extremely long, meaning it stays still or runs stupidly, or its reward is extremely low. However, with the AMD algorithm and Q-value adjustments, the agents start to learn.



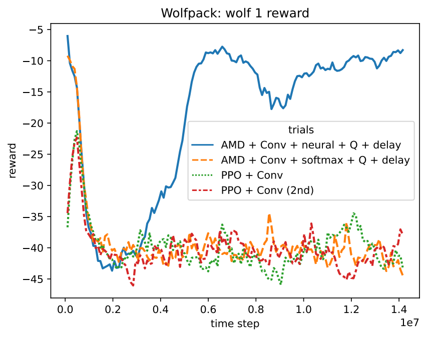


Figure 5: Training results of **Wolfpack** using only convolution neural network without LSTM.

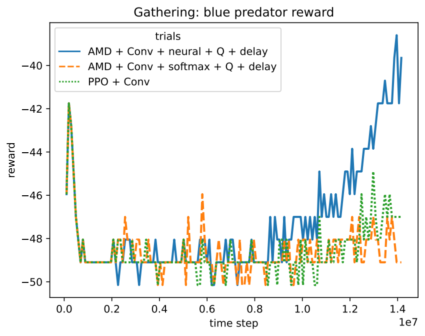


Figure 6: Training results of **Gathering** using only convolution neural network without LSTM.