University of Tehran

Faculty of Engineering

School of Electrical and Computer Engineering

Distributed Optimization and Learning

Project 3

Distributed Cooperative Competitive Multi-Agent Reinforcement Learning in Markov Games

Mohammad

Mashreghi

810199492

Winter 2024

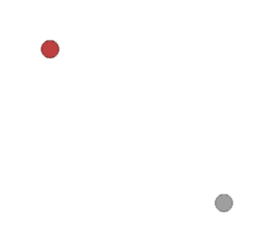
Abstract

In this project, we will consider distributed multi-agent reinforcement learning (MARL) in cooperative Markov games. We first consider a simple single-agent Q-learning algorithm to solve a single-agent MDP as a starting point after that we use actor critic to solve single agent model. Afterwards, we will implement four MARL algorithms to solve multi-agent tasks modeled by comparative Markov games. The first four algorithms are all value-based and include separated Q-learning ,Distributed Q-Learning, Minimax Q-Learning, and Belief based Q-Learning. The first two algorithms do not require communication among agents for convergence but only converge in deterministic MDPs, while Minimax and Belief based is communication-based and finally we test it in a continues mode with DDPG and MADDPG.

1. Probelem

Single agent problem:

In this project, we first attempt to use Q-learning and actor critic for single agent.



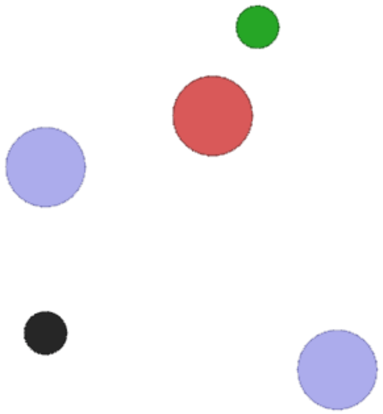
In this environment a single agent(grey) sees a landmark position(red) and is rewarded based on how close it gets to the landmark (Euclidean distance).

Our agent is in a continues state space but we change the continues state space to discrete.
we assume a 2*2 squre for our env and make 400 state every little house with 0.1*0.1 is a state. And if are agent is much far away from the landmark position which the agent can’t be seen in the environment, we assum the agent is in the margin.
For single agent our observation is agent_vel_x_axis, agent_vel_y_axis, relative_location_x, relative_location_y.
For actions we have 5 action which is : don’t’move, Up, down , right, left.
In the sigle agent decrease state space, we only use third and fourth observation or first and second one.

Multy agent Problem

Simple Adversary

In this environment, there is 1 adversary (red), N good agents (purple), N landmarks. All agents observe the position of landmarks and other agents. One landmark is the 'target landmark' (colored green). Good agents are rewarded based on how close the closest one of them is to the target landmark, but negatively rewarded based on how close the adversary is to the target landmark. The adversary is rewarded based on distance to the target, but it doesn't know which landmark is the target landmark. All rewards are unscaled Euclidean distance . This means good agents have to learn to 'split up' and cover all landmarks to deceive the adversary.



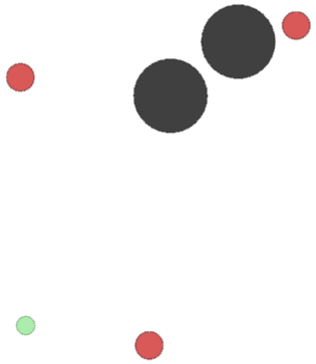
Agent observation

space: itself_loc , goal_landmark(only visiable for agent), other_fake_landmark(only visiable for agent), other_agent_or_adversery_loc

Agent and Adversary **action** is like before.

Simple tag

This is a predator-prey environment. Good agents (green) are faster and receive a negative reward for being hit by adversaries (red) (-10 for each collision). Adversaries are slower and are rewarded for hitting good agents (+10 for each collision). Obstacles (large black circles) block the way. By default, there is 1 good agent, 3 adversaries and 2 obstacles.



space: itself_loc and vel , goal_landmark(only visiable for agent), other_fake_landmark(only visiable for agent), other_agent_or_adversery_loc

Agent and Adversary **action** is like before.

2. Introduction

Unlike single-agent reinforcement learning (RL) where the state transition probabilities of the underlying Markov decision process (MDP) are determined by the actions of a single agent, in a multi-agent environment, the transitions are dependent on the actions of all agents that are present in the environment. Therefore, a natural framework for multi-agent reinforcement learning (MARL) is stochastic games (SG) [7, 8],

Definition 1. (Stochastic (Markov) game)**.** A stochastic (Markov) game is defined by extending the definition of MDP to a multi-player setting, and therfore can be described using a tuple of five key elements $(N, \mathcal{S}, \mathcal{A} = \{\mathcal{A}_i | i \in N\}, \mathcal{P}, R = \{R_i | i \in \mathcal{N}\})$, where,

- $\mathcal{N} = \{1,2,3, ..., n\}$ is a set of n players.
- $\mathcal{S}$ is a set representing the different states of the game, and is shared by all players.
- $\mathcal{A}_i$ is a set of m_i possible actions that player i could play at each state.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a probability mapping that at time step $t \in \mathbb{N}$ gives the transition probabilities of the agents going from state $s \in \mathcal{S}$ to the next state $s' \in \mathcal{S}$ at time step $t+1$ given the action profile $(a_1, ..., a_n) \in \mathcal{A}$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$ is the reward function that returns a bounded scalar in the range $[-R_{max}, R_{max}]$ to each agent as a result of the action profile $(a_1, ..., a_n)$ taken in state s and the transition to s'.

In general, in a MARL task, every agent will try to maximize her expected utility over the infinite horizon,

$$U_i(\pi^i, \pi^{-i}) = \mathbb{E}_{(a_t^i, a_t^{-i}) \sim \pi} \left\{ \sum_{t=0}^{\infty} \gamma^t R^i_{s_t, s_{t+1}}(a^i, a^{-i}) \right\}$$

In this project, we will be considering Markov games where the agents' interests are aligned with each other. An example of such a game is identical-interest (team) stochastic games,

Definition 2. (Team Stochastic Game)**.** An SG is a team game if for each action profile, all players receive an equal reward. In other words,

$$r_i(s, a_i, a_{-i}) = r_j(s, a_i, a_{-i}), \quad \forall i, j \in \mathcal{N}, s \in \mathcal{S}.$$

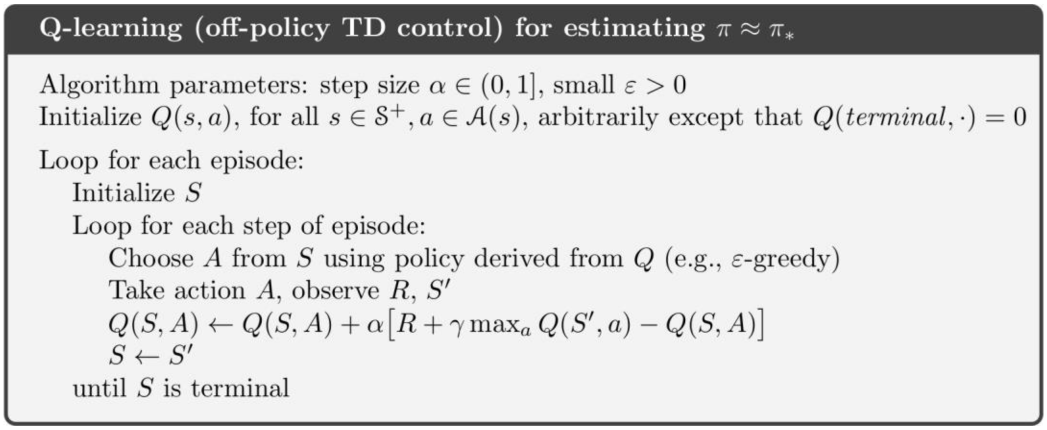
Therefore, in a stochastic team game, the incentive of all agents is fully-aligned. The first two MARL algorithms that we implement in this project (Distributed Q-Learning and Team Q-

Learning) have convergence guarantees only in identical-interest Markov games. The third and fourth algorithms which are QD-Learning and Networked Actor-Critic are cooperative algorithms that optimize the sum of expected returns of all agents by leveraging a communication graph.

3. Algorithms

3.1. Single-Agent Q-Learning

In our first experiment, we will consider a single-agent MDP with discrete state and action spaces, and implement the single-agent Q-learning,



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Single-agent Q-learning will always converge to an optimal policy (which is known to be deterministic for a finite MDP) if the agent follows a greedy in the limit with infinite exploration (GLIE) policy, such as epsilon greedy.

3.2. Actor critic

In a simple term, Actor-Critic is a Temporal Difference (TD) version of Policy gradient. Actor-critic algorithms combine the advantages of value-based and policy-based methods. The actor is a policy network that outputs a probability distribution over actions, while the critic is a value network that estimates the expected return for each state or state-action pair. The actor and the critic are trained jointly by using the critic's output as a baseline or a target for the actor's gradient update. This way, the actor can learn from both its own experience and the critic's feedback.

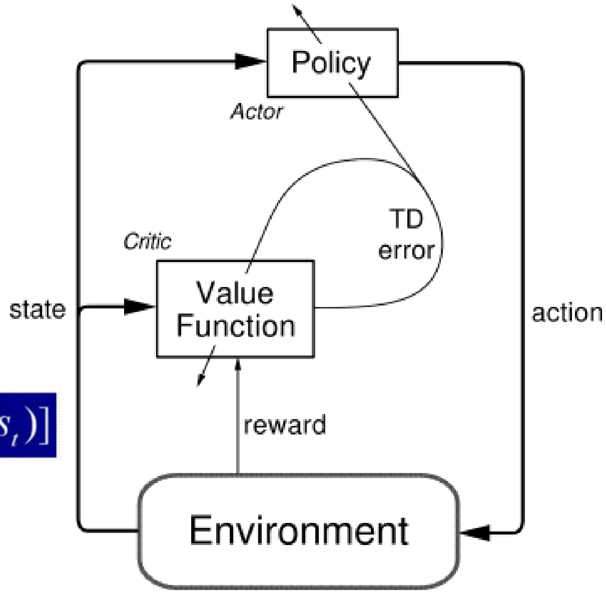
$$\delta_t = r_{t+1} + \gamma . V(s_{t+1}) - V(s_t)$$

$$\pi_t(s,a) = \Pr\{a_t = a \mid s_t = s\} = \frac{e^{P(s,a)}}{\sum_b e^{P(s,b)}}$$

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

$$V_{t+1}(s_t) = V_t(s_t) + \alpha\, \delta_t = V_t(s_t) + \alpha .[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)]$$

$$\pi(s,a) = \frac{e^{\frac{Q(a)}{\tau}}}{\sum_{b \in A(s)} e^{Q(b)/\tau}}$$

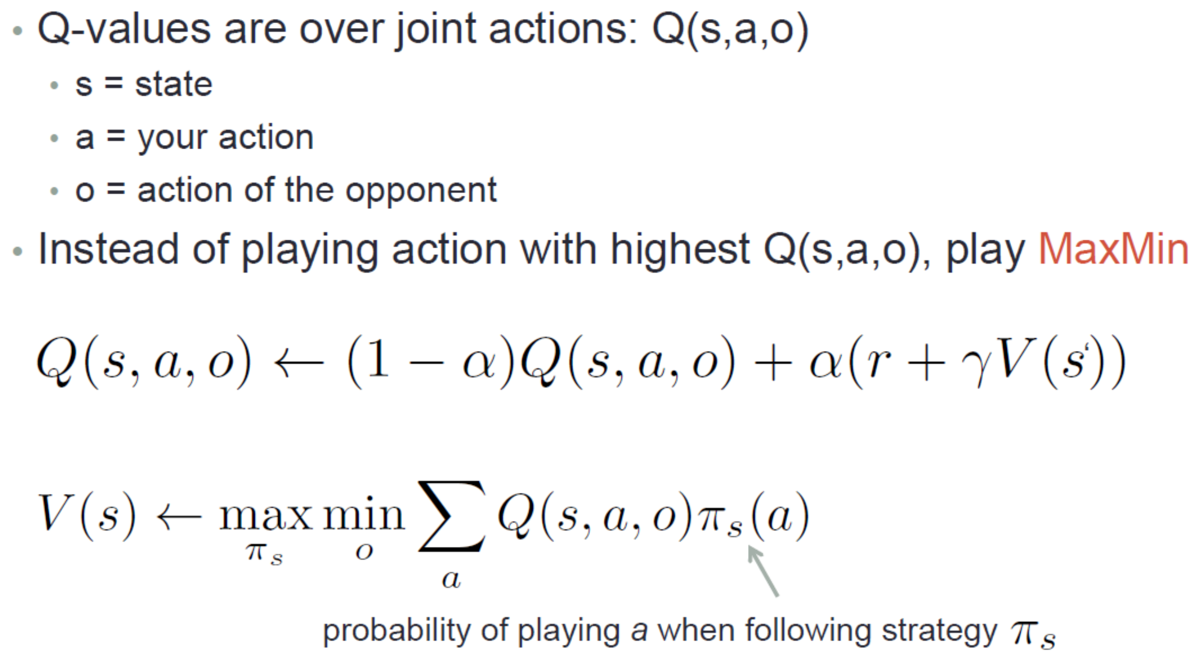


Actor-critic algorithms have several benefits over vanilla policy gradients. First, they can reduce the variance of the policy gradient by using the critic's value function as a baseline, which means they need fewer samples to converge. Second, they can incorporate temporal difference learning, which allows them to bootstrap from the critic's estimates and update the policy without waiting for the end of an episode. Third, they can use different types of critics, such as state-value functions, action-value functions, or advantage functions, to suit different problems and objectives.

Actor-critic algorithms also have some drawbacks compared to vanilla policy gradients. First, they introduce a trade-off between bias and variance, as the critic's value function may be inaccurate or inconsistent with the actor's policy, which can affect the quality of the policy gradient. Second, they increase the complexity and computational cost of the algorithm, as they require two neural networks to be trained and updated simultaneously. Third, they may suffer from instability or divergence, especially when using function approximation, as the actor and the critic may interfere with each other's learning.

3.3. Minimax Q-learning

The First MARL algorithm considered is Minimax Q-Learning

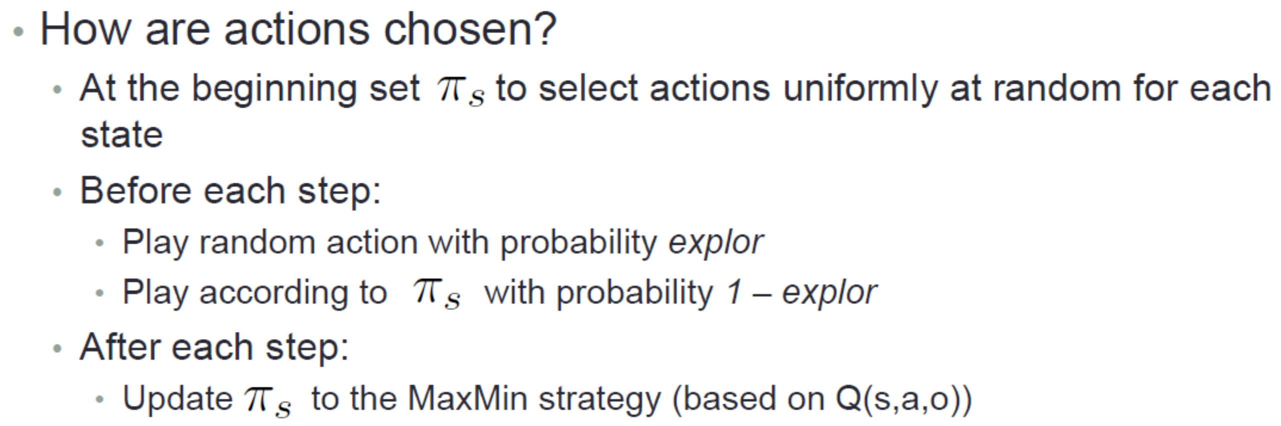


- Q-values are over joint actions: Q(s,a,o)
  - s = state
  - a = your action
  - o = action of the opponent
- Instead of playing action with highest Q(s,a,o), play MaxMin

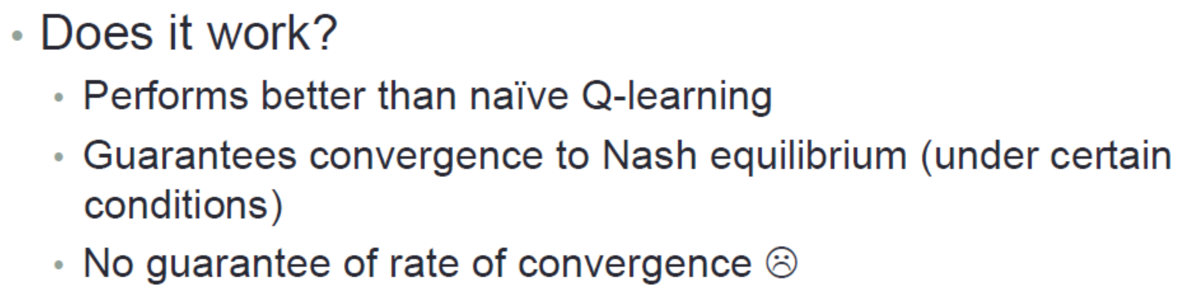
$$Q(s,a,o) \leftarrow (1-\alpha)Q(s,a,o) + \alpha(r + \gamma V(s'))$$

$$V(s) \leftarrow \max_{\pi_s} \min_{o} \sum_{a} Q(s,a,o)\pi_s(a)$$

probability of playing *a* when following strategy π_s



- How are actions chosen?
  - At the beginning set π_s to select actions uniformly at random for each state
  - Before each step:
    - Play random action with probability *explor*
    - Play according to π_s with probability *1 – explor*
  - After each step:
    - Update π_s to the MaxMin strategy (based on Q(s,a,o))



- Does it work?
  - Performs better than naïve Q-learning
  - Guarantees convergence to Nash equilibrium (under certain conditions)
  - No guarantee of rate of convergence ☹

3.4. Seperated Q_Learning

In this method we assume that agents try to maximum their reward by seeing only their self-actions and rewards and states.

3.5. Distributed Q-Learning

The second MARL algorithm considered is Distributed Q-Learning [5]. This algorithm is free of coordination, and the agents only need to store a local Q-table based on their own action. The update of Q-function is as follows:

$$Q_{i,k+1}(x_k, u_{i,k}) = \max\big\{Q_{i,k}(x_k, u_{i,k}), \\ r_{k+1} + \gamma \max_{u_i} Q_{i,k}(x_{k+1}, u_i)\big\}.$$

The local policy is only updated if the aforementioned update leads to an improvement in Q-values:

$$\bar{h}_{i,k+1}(x_k) = \begin{cases} u_{i,k} & \text{if } \max\limits_{u_i} \ Q_{i,k+1}(x_k, u_i) \\ & \quad > \max\limits_{u_i} \ Q_{i,k}(x_k, u_i) \\ \bar{h}_{i,k}(x_k) & \text{otherwise.} \end{cases}$$

Which ensures that the joint policy is always optimal with respect to global Q function.

This algorithm converges to an optimal policy in team Markov games with deterministic transitions when following a GLIE exploration strategy.

3.6. Belief based algorithm

The belief-based learning models are a class of models that intend to describe the way agents (such as humans) learn over time in strategic situations (games) that they play repeatedly. According to a belief-based learning model, agents form beliefs about the expected behavior of others, and choose actions based on how they would perform against opponents playing in this expected way. After playing the game, they update their beliefs based on the feedback they receive.

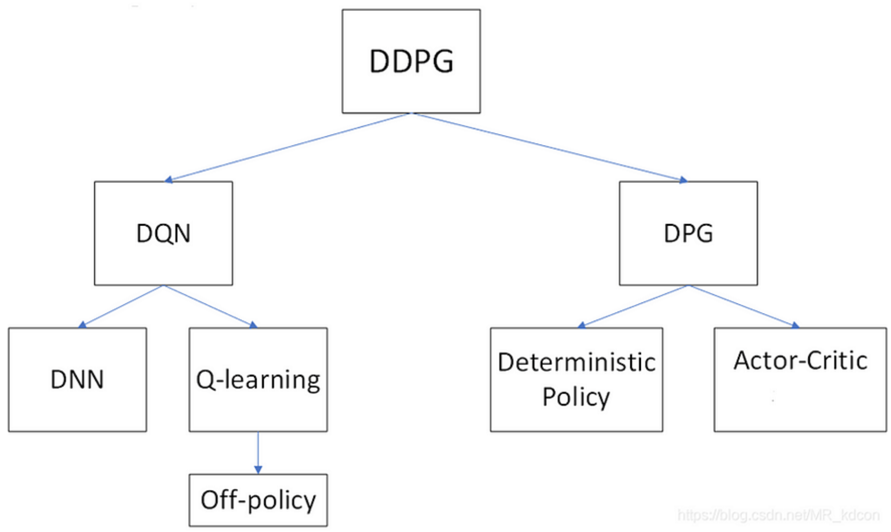
$$Q_{t+1}(s_t, a_t) \leftarrow (1-\alpha)Q_t(s_t, a_t) + \alpha_t(r(s_t, a_t) + \beta V_t(s_{t+1}))$$

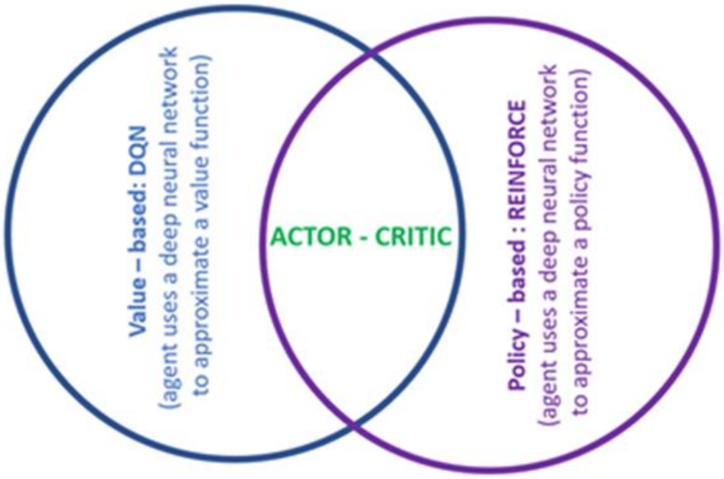
$$V_t(s) \leftarrow \max_{a_i} \sum_{a_{-i} \subset A_{-i}} Q_t(s, (a_i, a_{-i}))Pr_i(a_{-i})$$

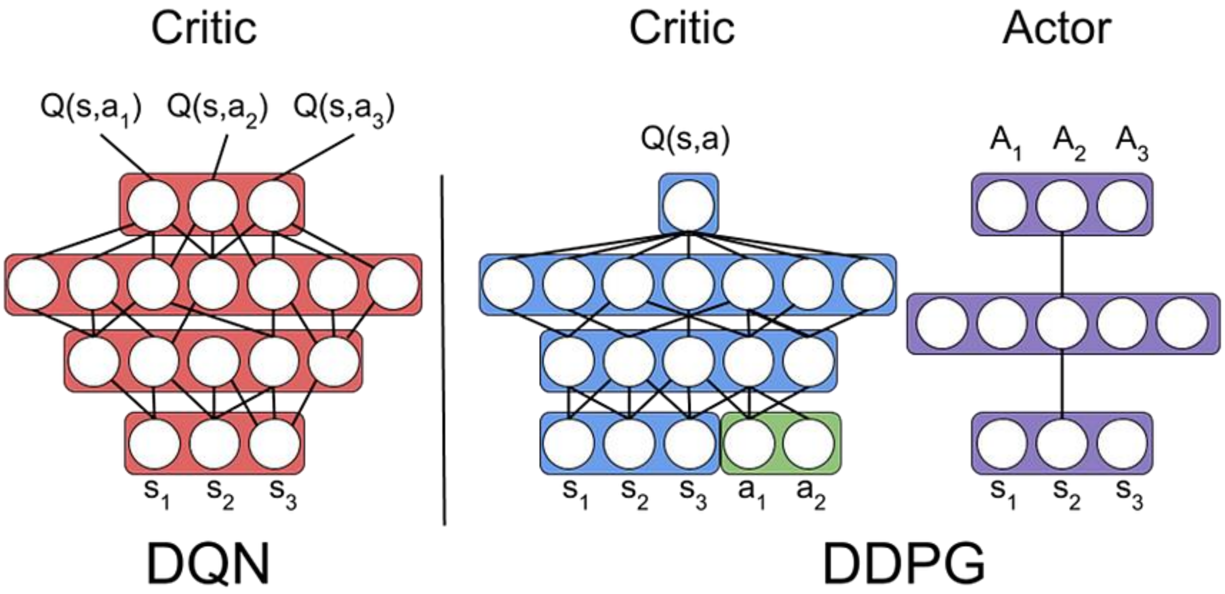
3.7. Deep Deterministic Policy Gradient

DDPG (Deep Deterministic Policy Gradient) is a model-free off-policy reinforcement learning algorithm for learning continuous actions. It combines ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). The development of deep deterministic policy gradient (DDPG) was inspired by the success of DQN and is aimed to improve performance for tasks that requires a continuous action space.

DDPG incorporates an actor-critic approach based on DPG. The algorithm uses two neural networks, one for the actor and one for the critic. Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy.




Value – based: DQN
(agent uses a deep neural network
to approximate a value function)
ACTOR - CRITIC
Policy – based : REINFORCE
(agent uses a deep neural network
to approximate a policy function)

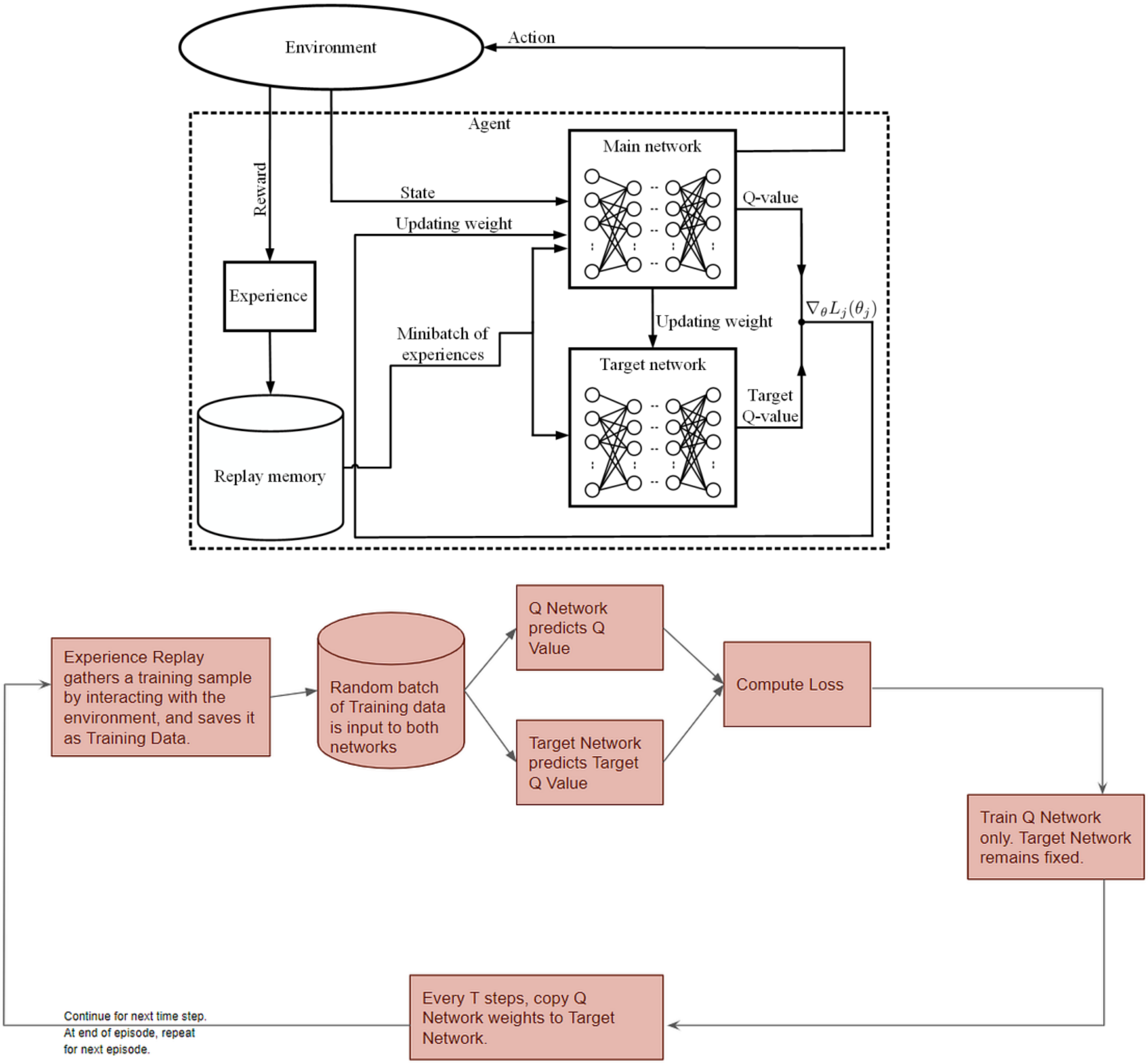


DDPG being an actor-critic technique consists of two models: Actor and Critic. The actor is a policy network that takes the state as input and outputs the exact action (continuous), instead of a probability distribution over actions like DQN.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma max Q\overbrace{(s',a')}^{\text{This depends Q function itself (at the moment it is being optimized)}} - Q(s,a)]$$

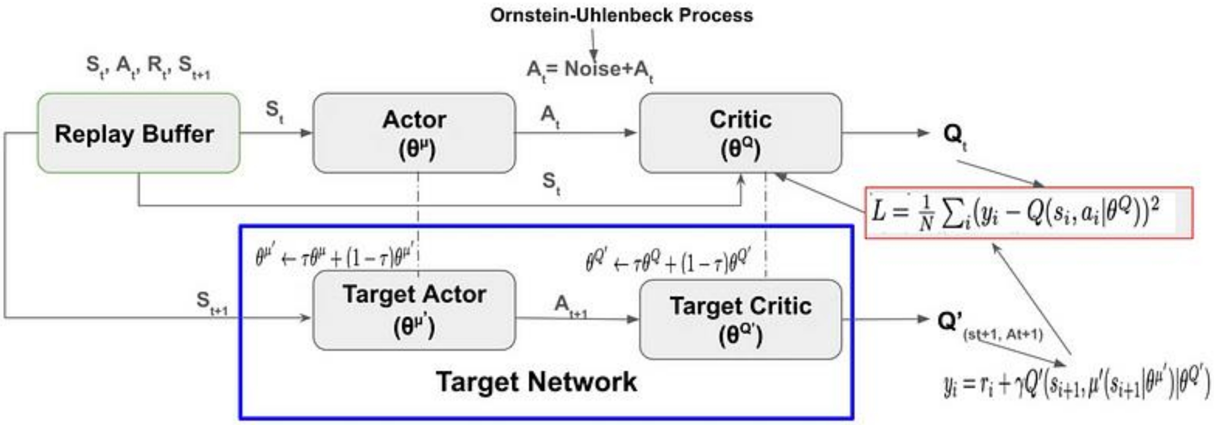
$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$



Environment
Action
Agent
Main network
Reward
State
Updating weight
Q-value
Experience
$\nabla_\theta L_j(\theta_j)$
Minibatch of experiences
Updating weight
Target network
Target Q-value
Replay memory
Experience Replay gathers a training sample by interacting with the environment, and saves it as Training Data.
Random batch of Training data is input to both networks
Q Network predicts Q Value
Target Network predicts Target Q Value
Compute Loss
Train Q Network only. Target Network remains fixed.
Every T steps, copy Q Network weights to Target Network.
Continue for next time step. At end of episode, repeat for next episode.

DQN

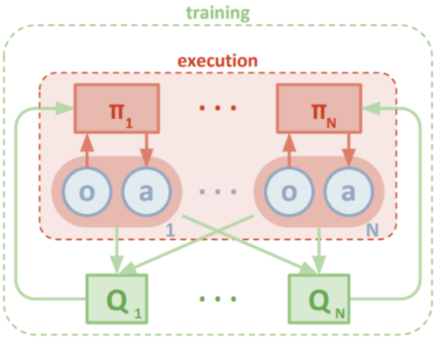


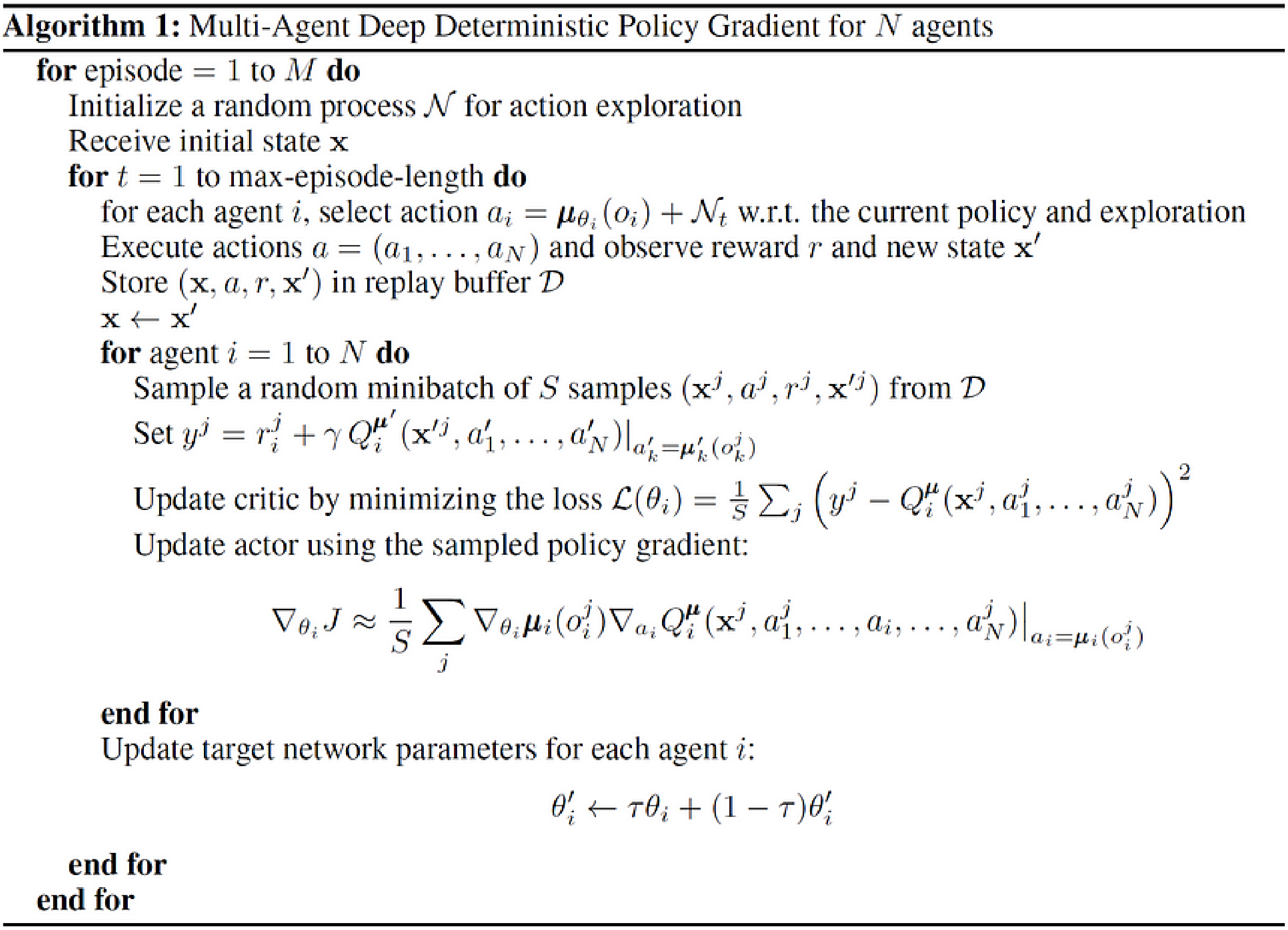
Ornstein-Uhlenbeck Process
S_t, A_t, R_t, S_{t+1}
A_t= Noise+A_t
Replay Buffer
S_t
Actor (θ^μ)
A_t
Critic (θ^Q)
Q_t
S_t
$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
S_{t+1}
Target Actor ($\theta^{\mu'}$)
A_{t+1}
Target Critic ($\theta^{Q'}$)
$Q'_{(st+1, At+1)}$
Target Network
$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

DDQN

3.8. Multi Agent Deep Deterministic Policy Gradient

MADDPG, or Multi-agent DDPG, extends DDPG into a multi-agent policy gradient algorithm where decentralized agents learn a centralized critic based on the observations and actions of all agents. It leads to learned policies that only use local information (i.e. their own observations) at execution time, does not assume a differentiable model of the environment dynamics or any particular structure on the communication method between agents, and is applicable not only to cooperative interaction but to competitive or mixed interaction involving both physical and communicative behavior. The critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. After training is completed, only the local actors are used at execution phase, acting in a decentralized manner.





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
```

1. Results

1.1. Single-Agent Q-Learning

Discount = 0.9,0.6,0.3

1000 episodes and 100 iterations per episode.

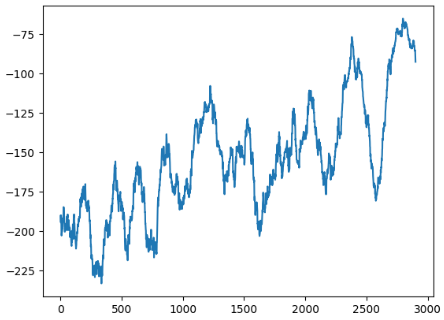
Lr = 0.01

Eps calculate each round with this

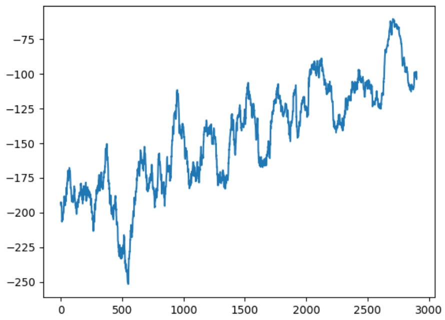
formula: $e^{-current\ episode*0.001}$

Number of iterations per episode: 25

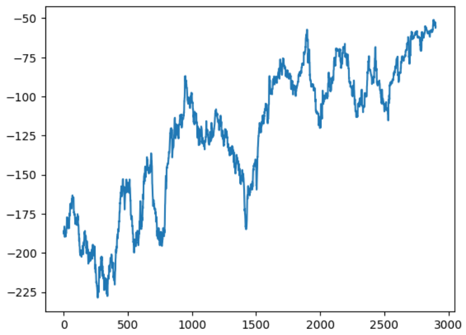
Number of episodes: 3000



Discount = 0.3



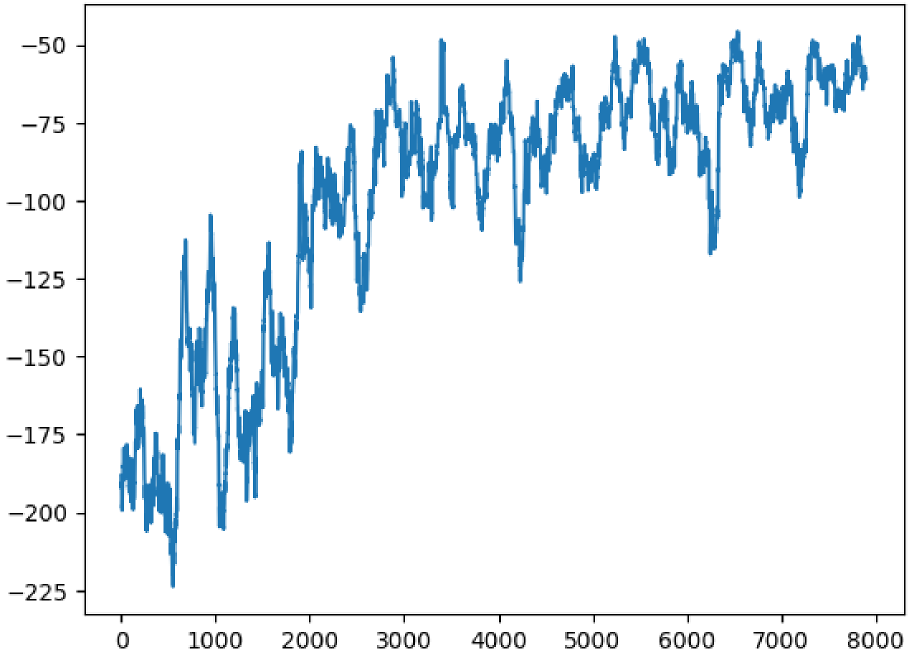
Discount = 0.6



Discount = 0.9

Intuitively, the decaying epsilon results in a better final performance as the exploration rate approaches zero.

If we choose Discount factor a high number it will converge sooner and get less confused (or even get confused check states more than other methods which take more episode but stand up with a better Q table value.



If you change the discount factor:

1. Higher Discount Factor (γ):

- If you increase the discount factor, the agent will prioritize long-term rewards more heavily. It will consider future rewards more significant compared to immediate rewards. This might lead to more far-sighted behavior, where the agent tries to maximize cumulative rewards over the long run, even if it means sacrificing immediate gains. However, this could also lead to slower convergence or instability in the learning process.

2. Lower Discount Factor (γ):

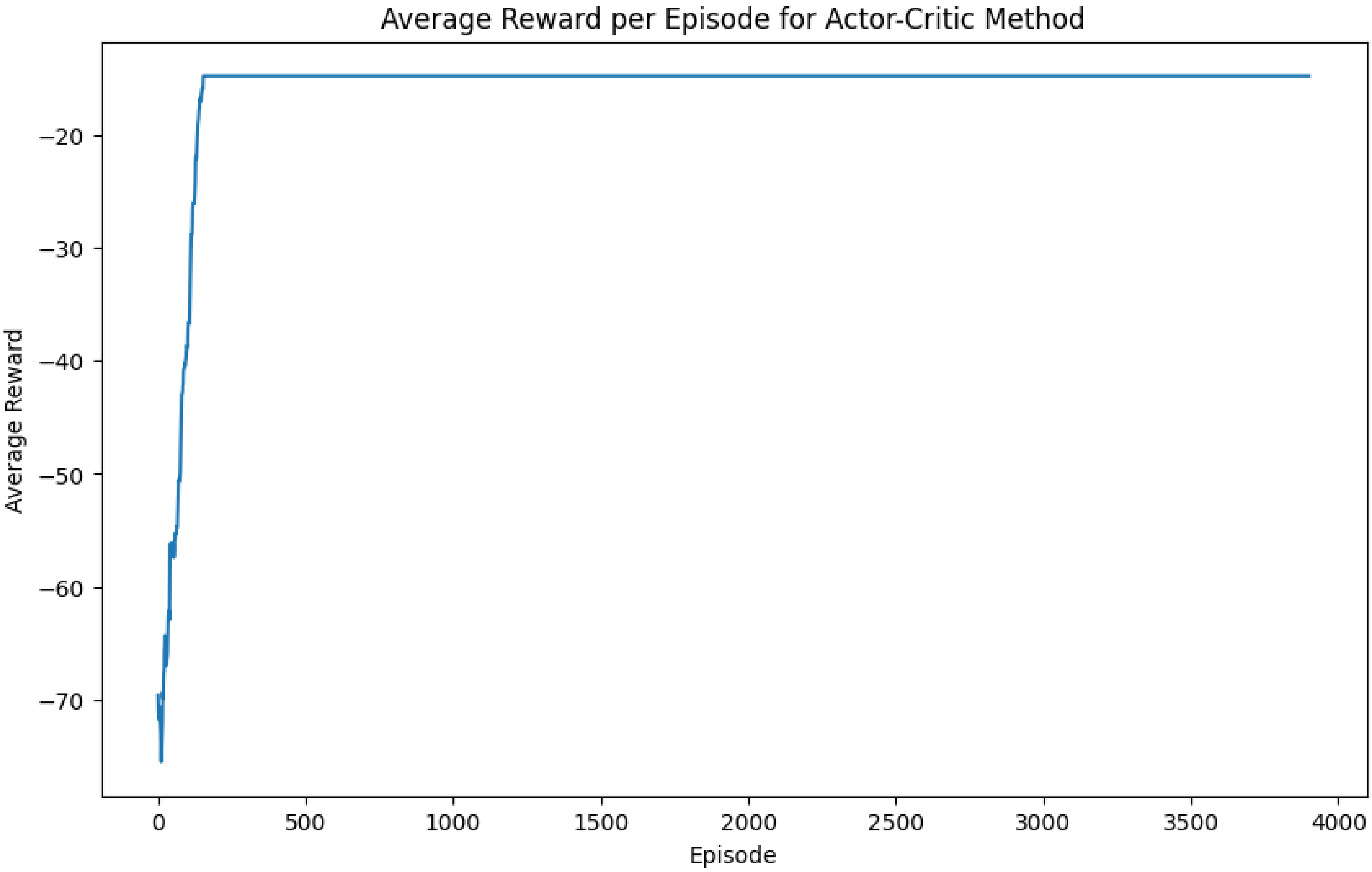
- Conversely, if you decrease the discount factor, the agent will focus more on immediate rewards. It will prioritize short-term gains over long-term benefits. This might lead to faster convergence in learning, especially in scenarios where immediate rewards are crucial for learning the optimal policy. However, the agent may overlook the long-term consequences of its actions.

1.2. Actor critic

Simulation setup: like before.

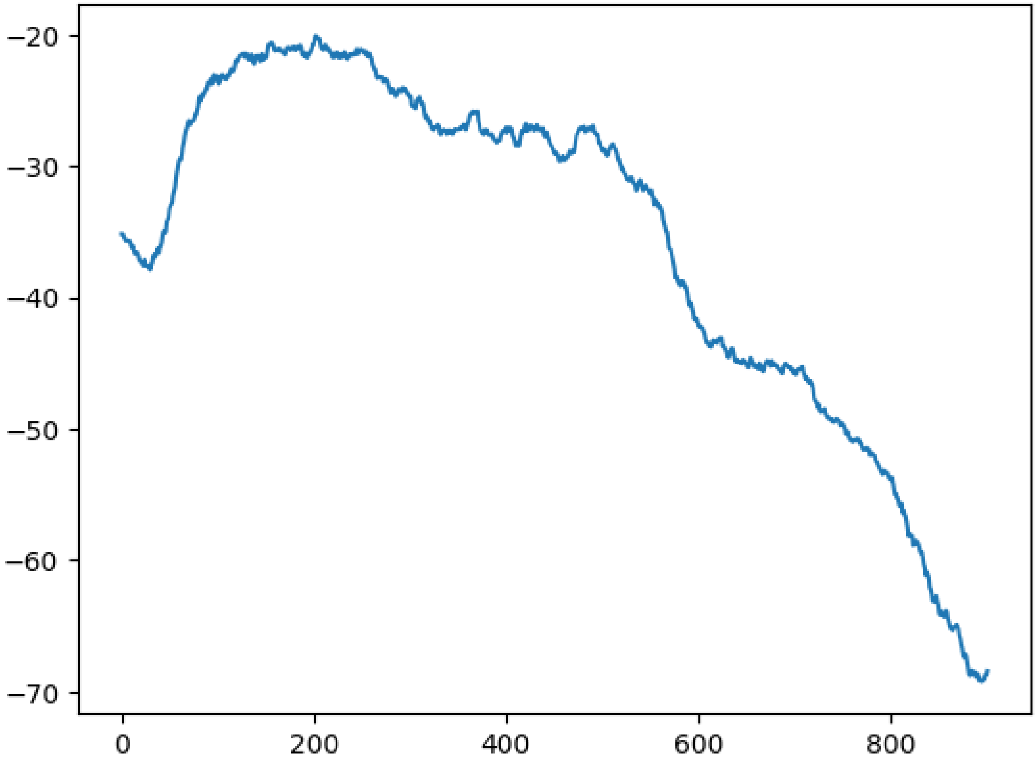
$$\boldsymbol{\tau = 1000\ for\ softmax}$$

$$\boldsymbol{actor_{lr} = 0.2}$$
$$\boldsymbol{critic_{lr} = 0.01}$$



It's converge soon because $\boldsymbol{\tau = 1000}$ for softmax if it was bigger, the agent would more explore and see the env.

We also implemented 2layer neural network for both actor and critic:



I have 2 geuss for not disconverg:

- First the agent when reach the landmark don’t get stop and get tiny neg reward after a while it disconverg.(we can stop agent when it reach the landmark)
- The env is slippy so when we tell agent do a action, it doesn’t change location exactly.

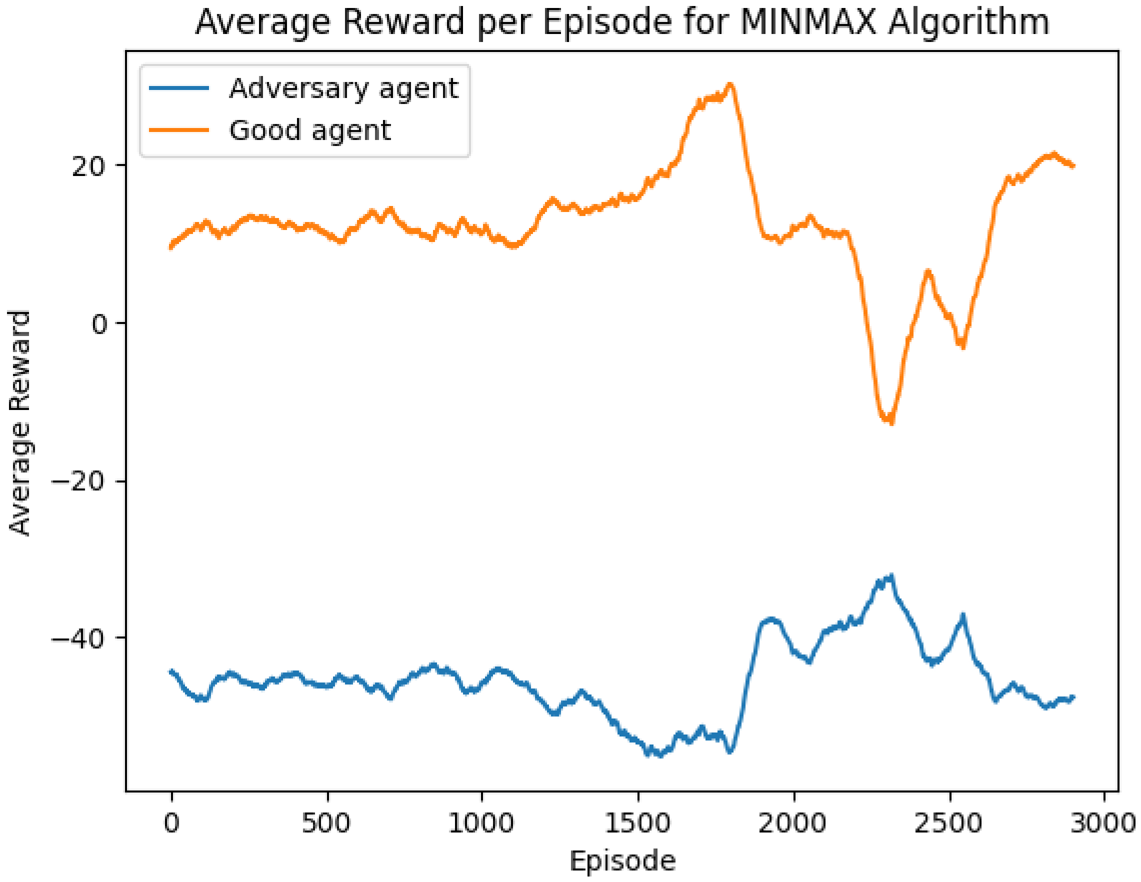
1.3. Minimax

Env = adversery

Num of good agent =1

Num of adversery agent =1

Simulation as before with two agent:



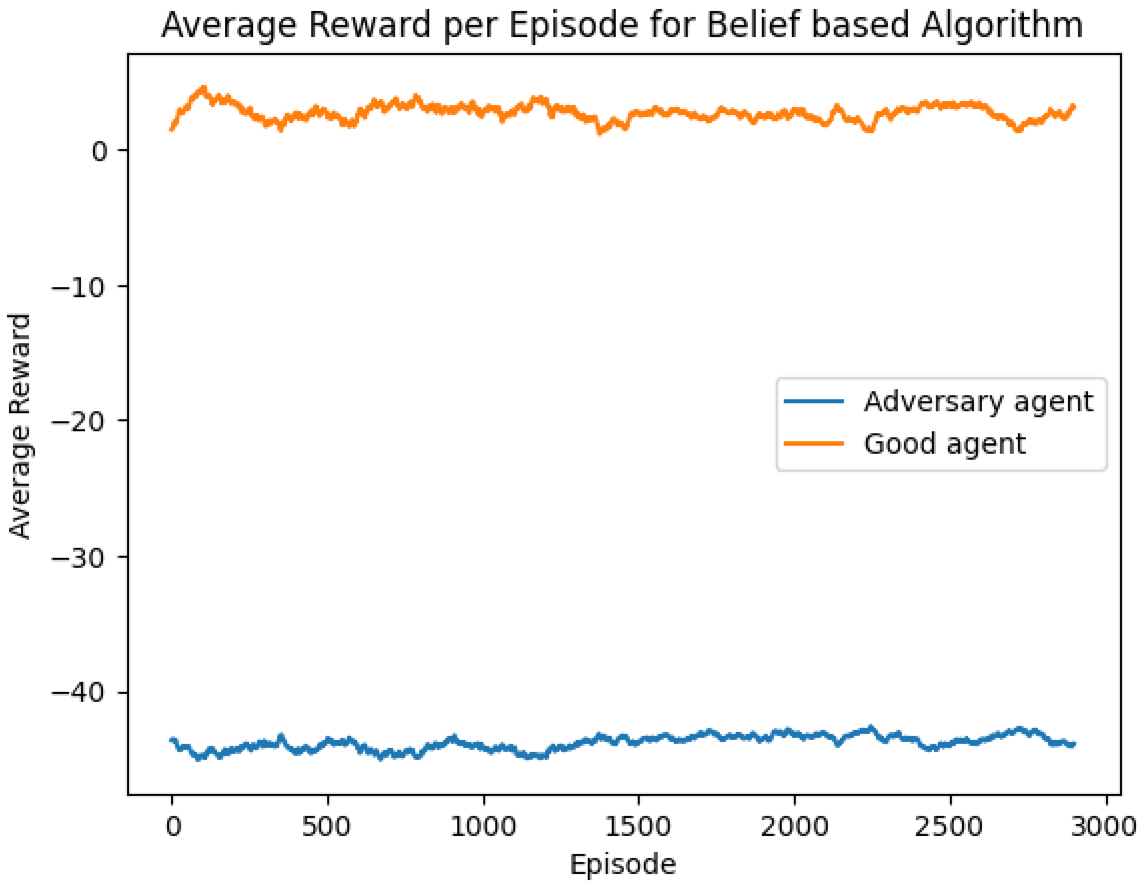
In here, looks like our adversery agent did well and follow the good agent.

1.4. Seperated Q-learning

Env = adversery

Num of good agent =1

Num of adversery agent =1

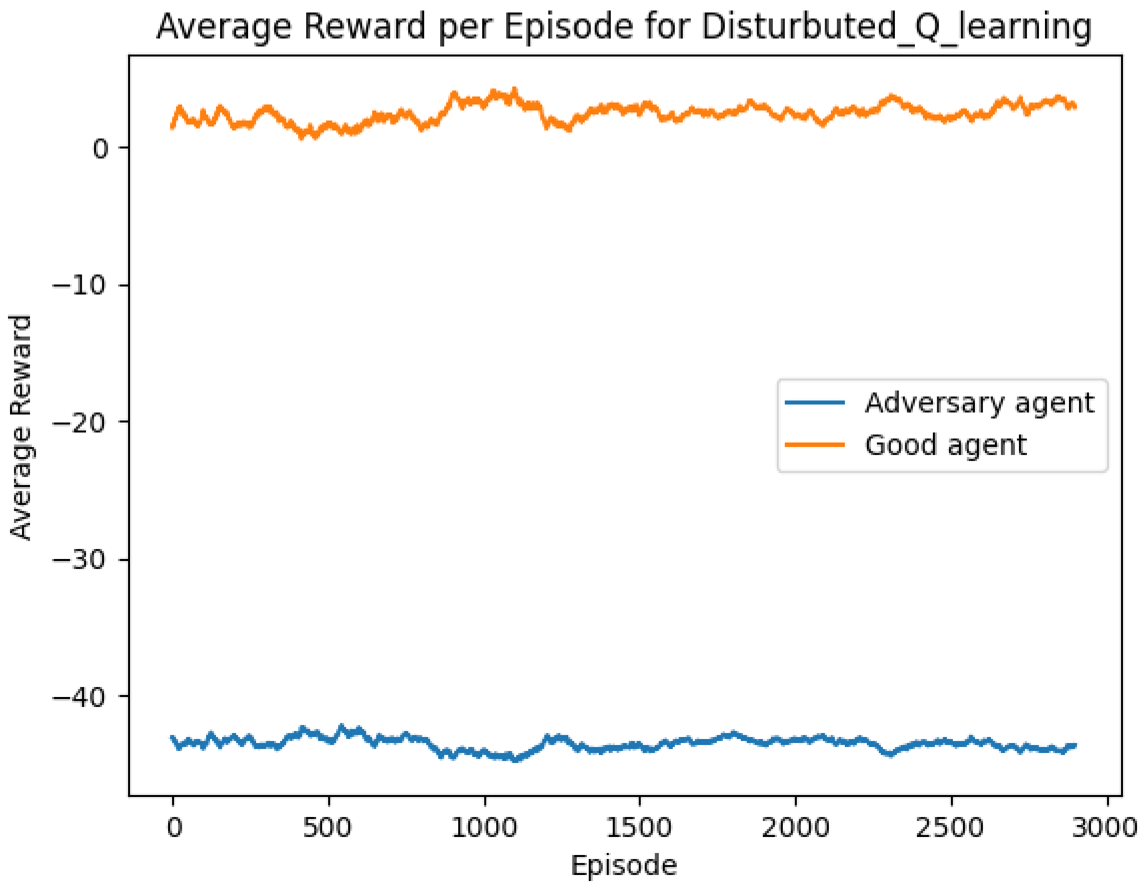


1.5. Distributed Q-Learning

Env = adversery

Num of good agent =1

Num of adversery agent =1

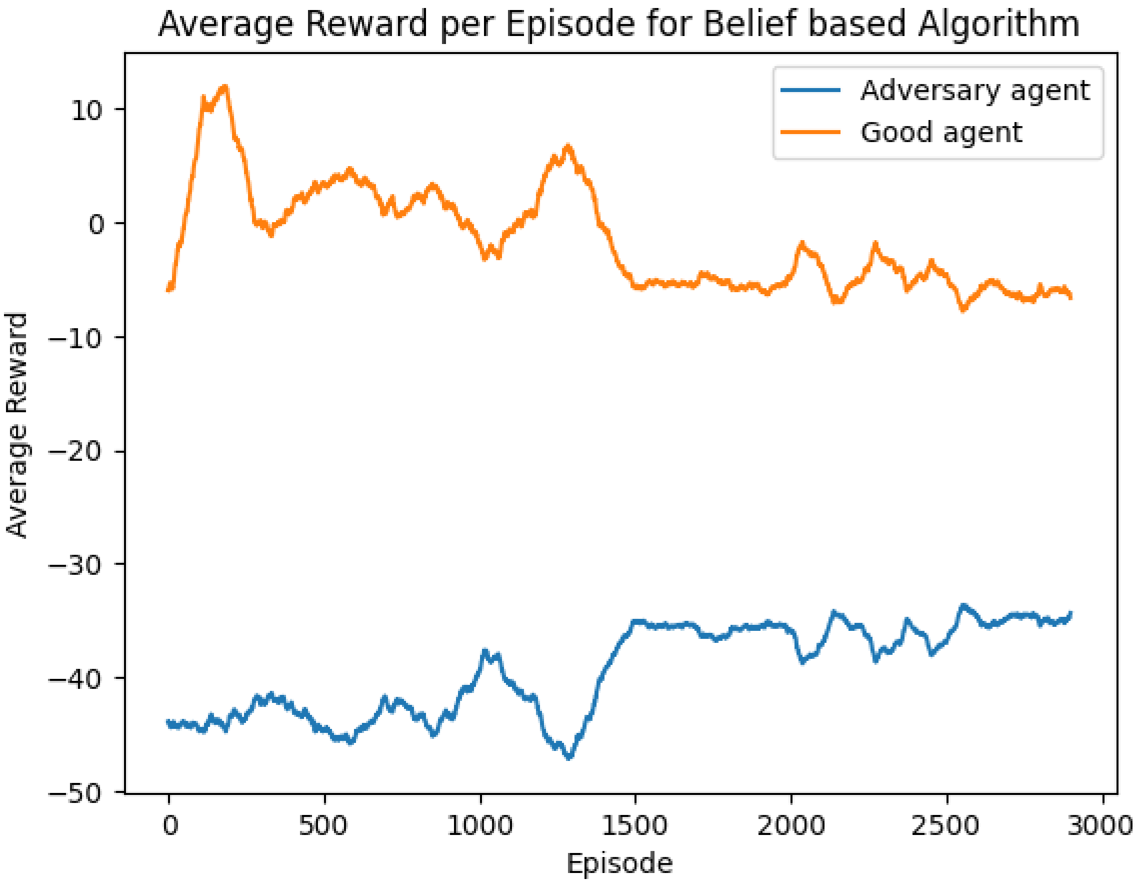


1.1. Belief Q-Learning

Env = adversery

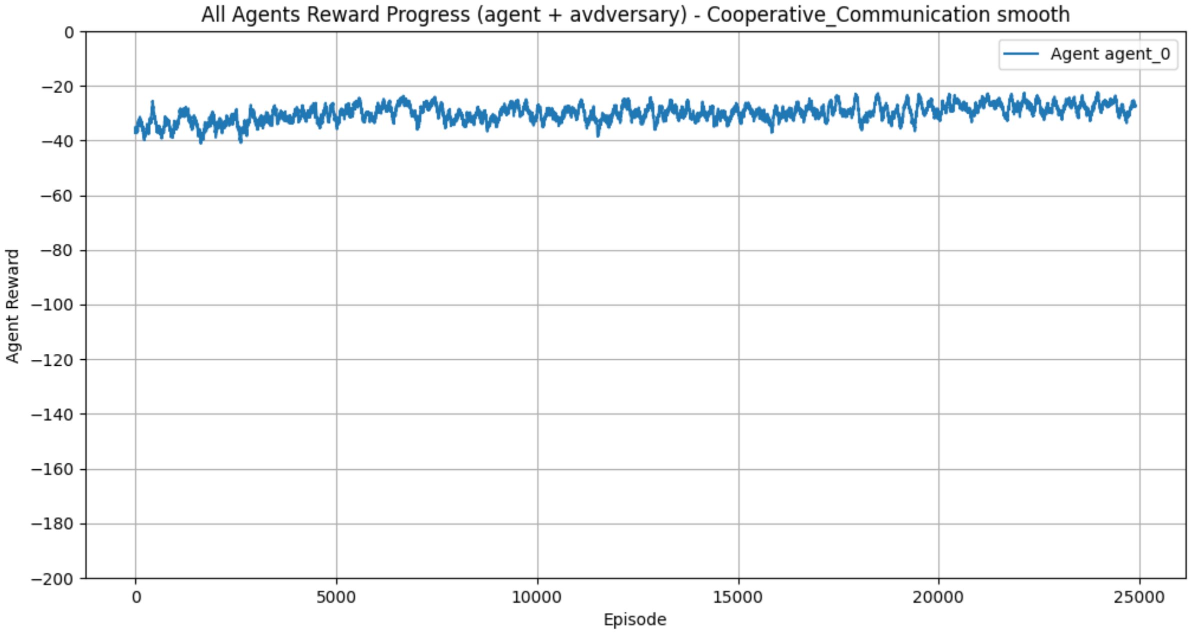
Num of good agent =1

Num of adversery agent =1



1.2. DDPG

For single agent:



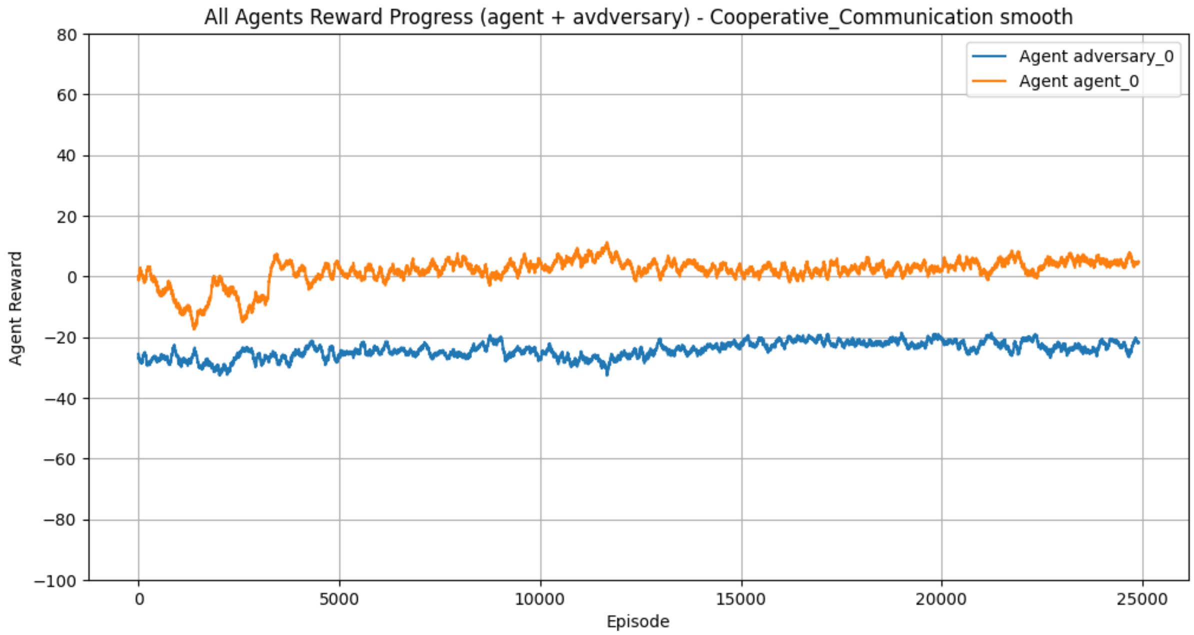
Simulation setup:

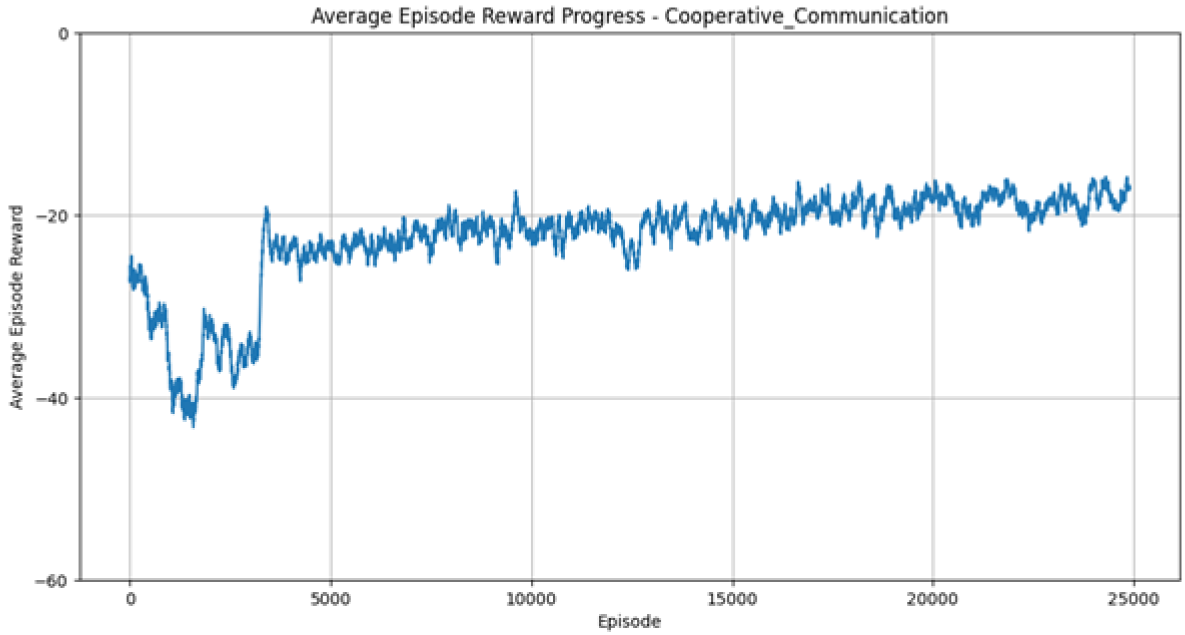
In this time, we simulate for 25000 step to learn interact we environment:

Env = adversery

Num of good agent =1

Num of adversery agent =1

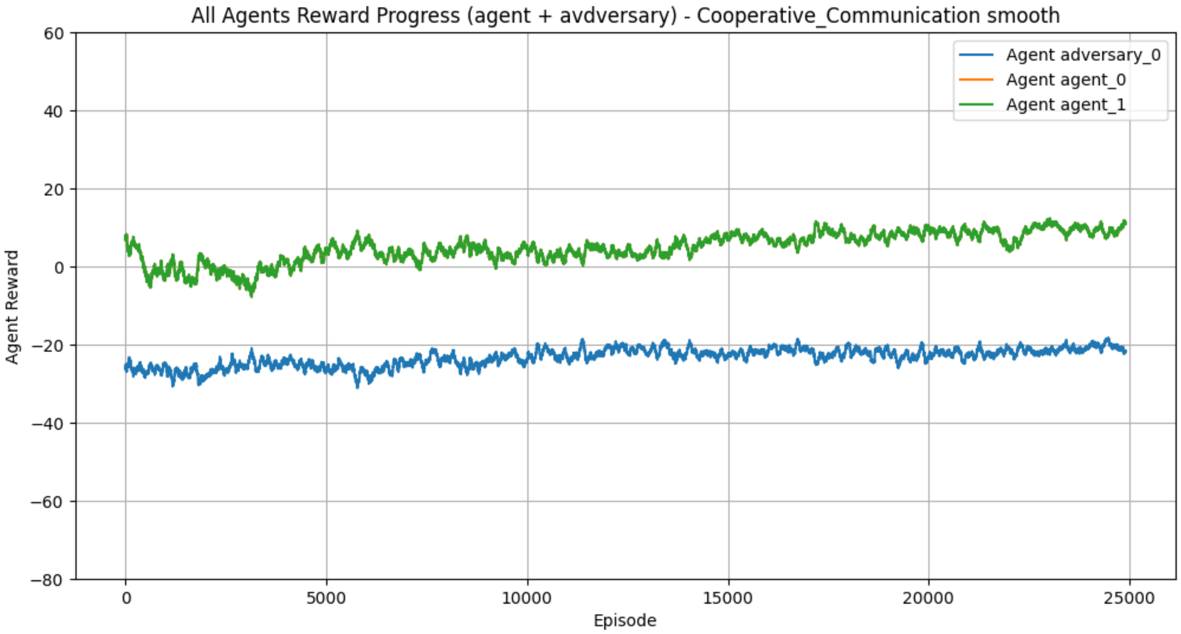


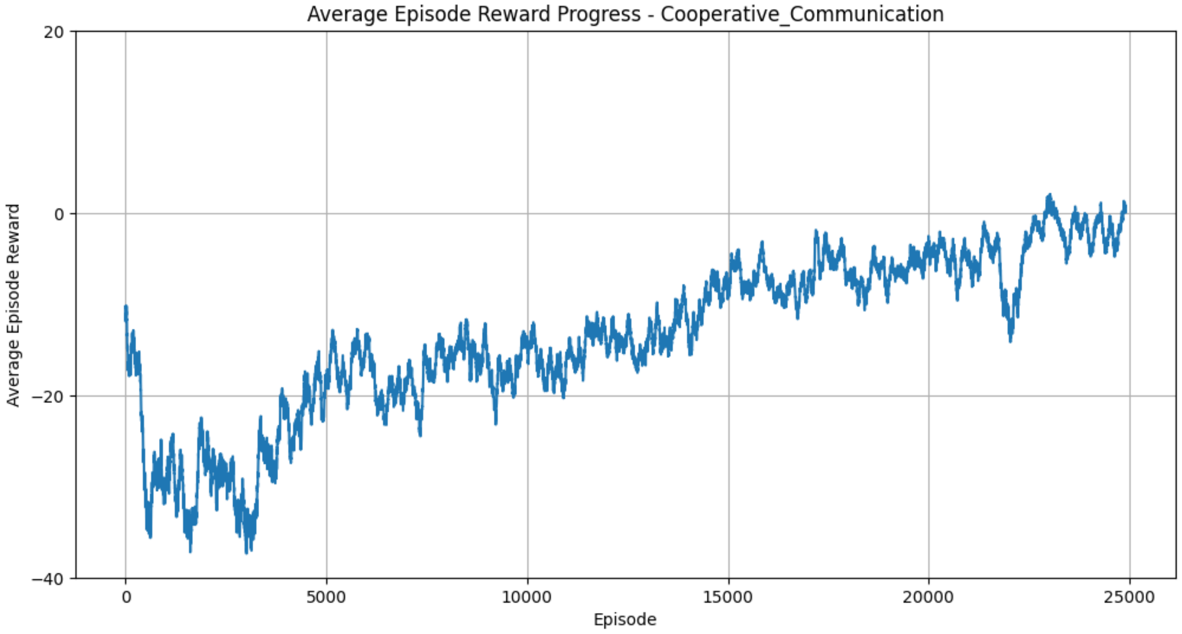


Env = adversery

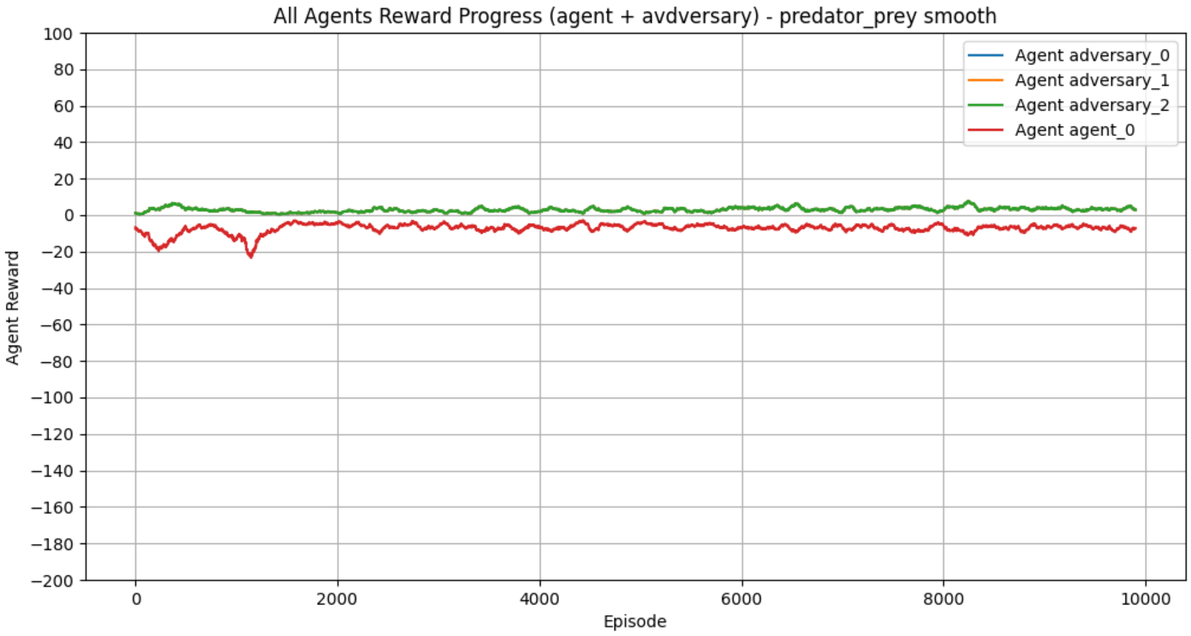
Num of good agent =2

Num of adversery agent =1

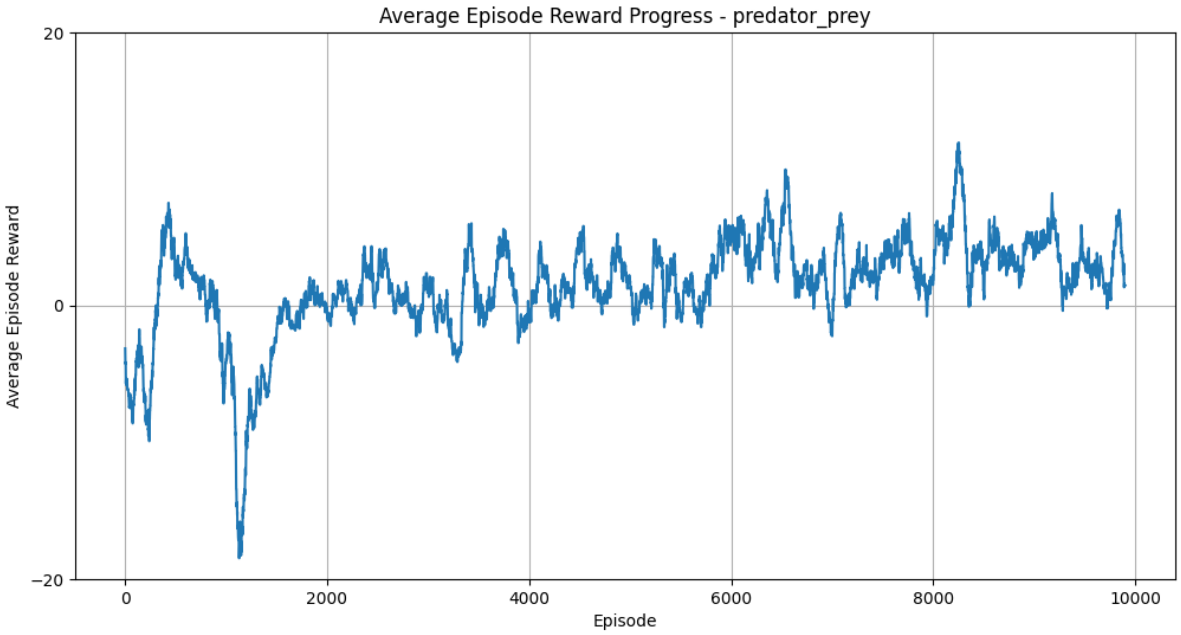




Simple Tag



All Agents Reward Progress (agent + avdversary) - predator_prey smooth
Agent Reward
Episode
100
80
60
40
20
0
−20
−40
−60
−80
−100
−120
−140
−160
−180
−200
0
2000
4000
6000
8000
10000
Agent adversary_0
Agent adversary_1
Agent adversary_2
Agent agent_0



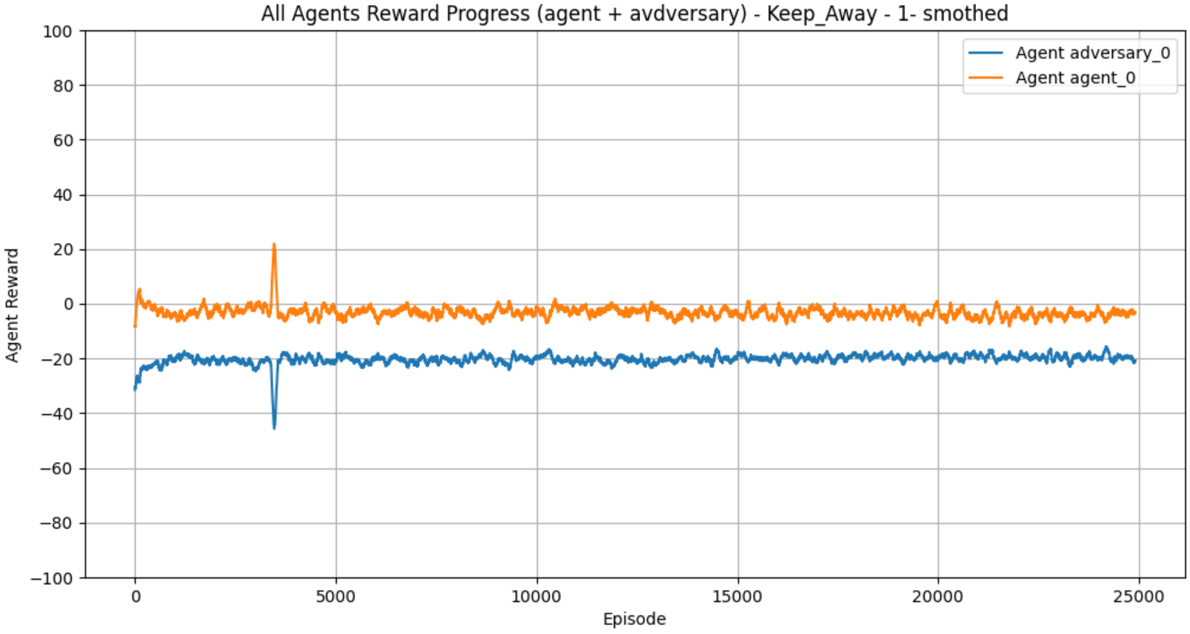
Average Episode Reward Progress - predator_prey
Average Episode Reward
Episode
20
0
−20
0
2000
4000
6000
8000
10000

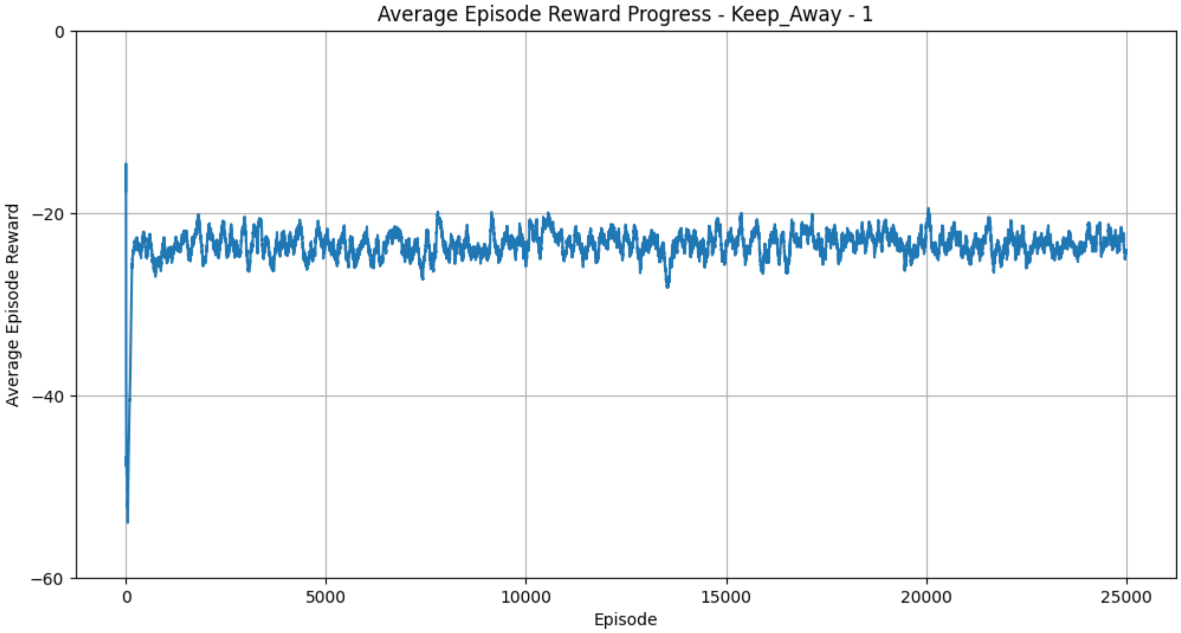
1.1. MADDPG

Env = adversery

Num of good agent =1

Num of adversery agent =1

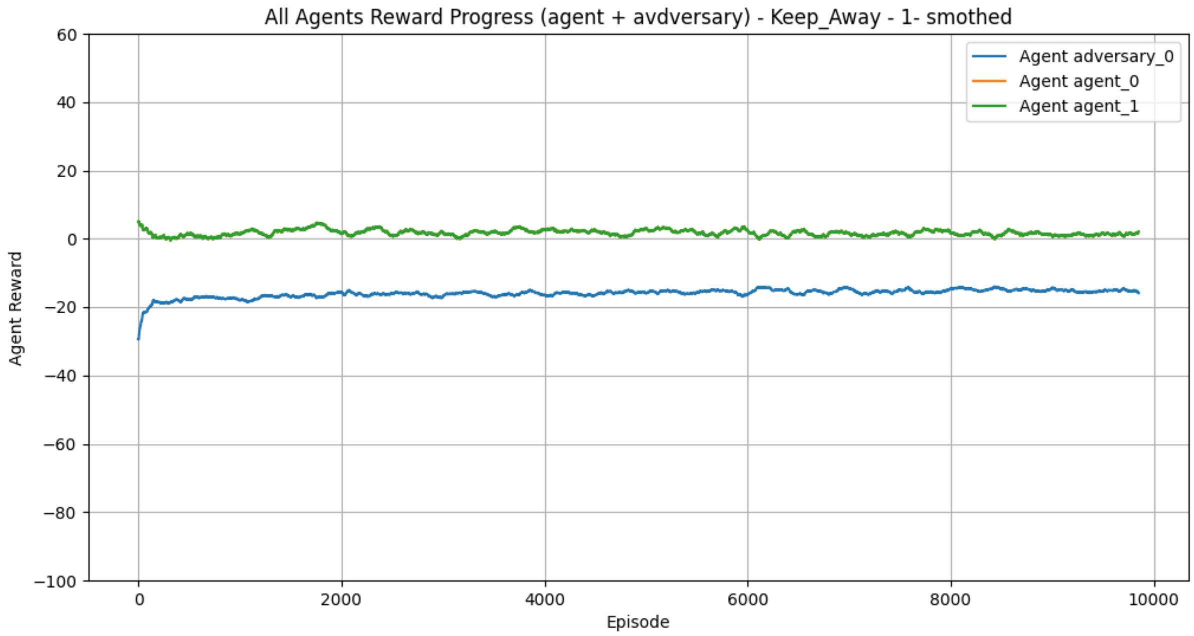


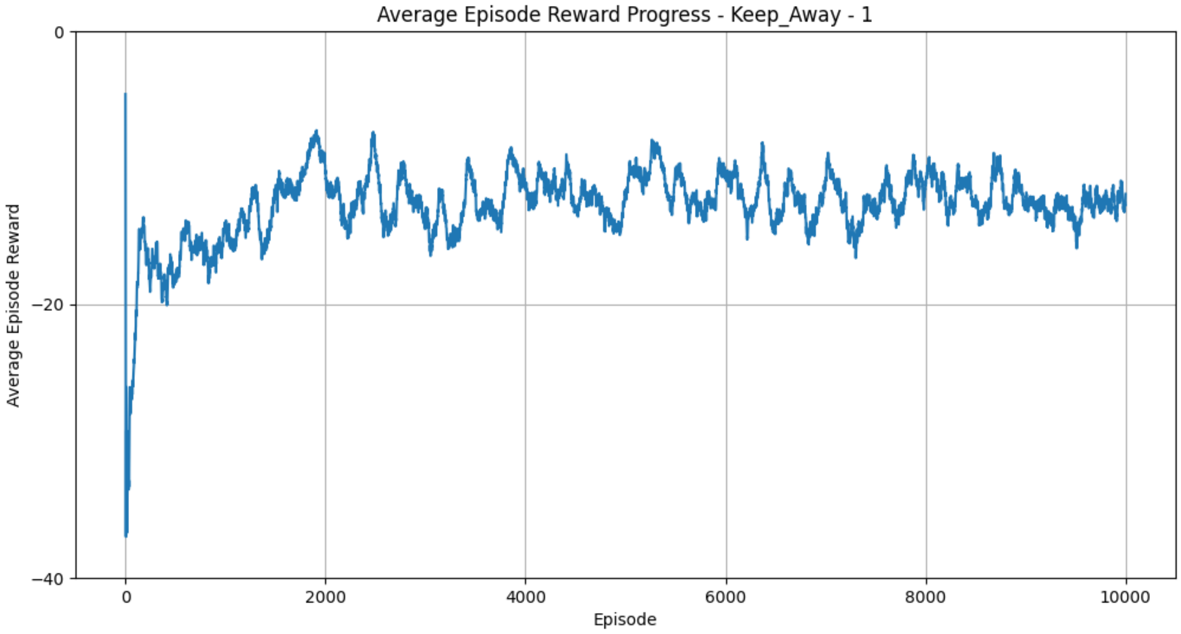


Env = adversery

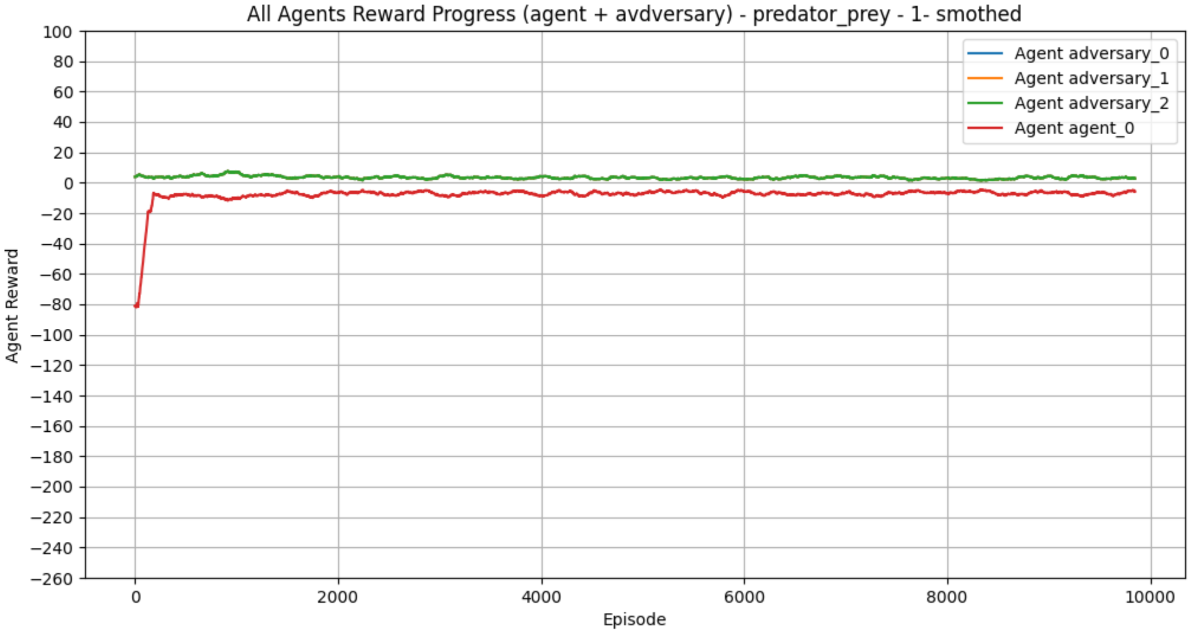
Num of good agent =2

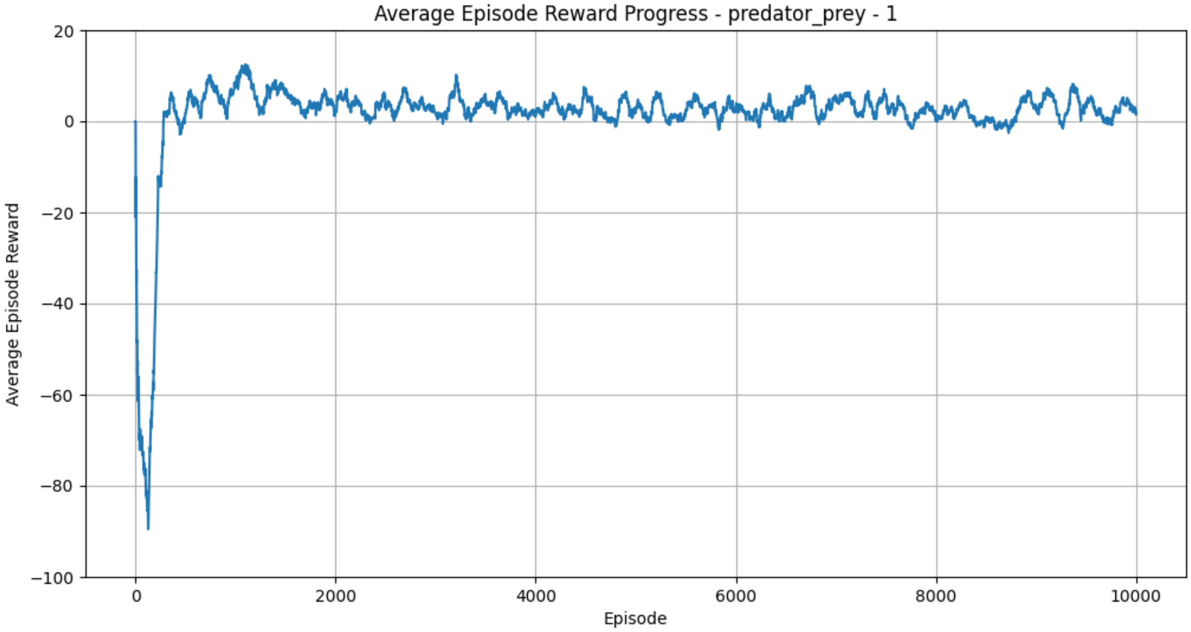
Num of adversery agent =1





Simple Tag


All Agents Reward Progress (agent + avdversary) - predator_prey - 1- smothed
Agent Reward
Episode
100
80
60
40
20
0
−20
−40
−60
−80
−100
−120
−140
−160
−180
−200
−220
−240
−260
0
2000
4000
6000
8000
10000
Agent adversary_0
Agent adversary_1
Agent adversary_2
Agent agent_0


Average Episode Reward Progress - predator_prey - 1
Average Episode Reward
Episode
20
0
−20
−40
−60
−80
−100
0
2000
4000
6000
8000
10000

2. Conclusion

In this project, we considered identical-interest Markov games and implemented some MARL algorithms to solve the corresponding multi-agent MDP. We also implemented Q-learning and actor critic to solve a single-agent MDP. Although the nature of the algorithms and their different simulation setups (we tried to use the setups from the corresponding paper of each algorithm to ensure convergence) may hinder direct comparison among their results, the following observations can be made,

- The seperated and Distributed Q-learning algorithms have a very narrow scope and are only applicable to deterministic identically-interest MDPs. Furthermore.

- The actions space is continues and the obsev space is more than 4 data, if want to slve it descrete, we have cursed if dim, so we need to use better method.
- In all of the four MARL algorithms discussed, the agents need to observe the actions of other agents. Therefore, they cannot be called “fully-distributed”.

References

[1] Zhang, Kaiqing, Zhuoran Yang, and Tamer Başar. "Decentralized multi-agent reinforcement learning with networked agents: Recent advances." *Frontiers of Information Technology & Electronic Engineering* 22.6 (2021): 802-814.

[2] Zhang, Kaiqing, et al. "Fully decentralized multi-agent reinforcement learning with networked agents." *International Conference on Machine Learning*. PMLR, 2018.

[3] Zhang, Kaiqing, Zhuoran Yang, and Tamer Basar. "Networked multi-agent reinforcement learning in continuous spaces." *2018 IEEE conference on decision and control (CDC)*. IEEE, 2018.

[4] Lauer, Martin, and Martin Riedmiller. "An algorithm for distributed reinforcement learning in cooperative multi-agent systems." *In Proceedings of the Seventeenth International Conference on Machine Learning*. 2000.

[5] Littman, Michael L. "Value-function reinforcement learning in Markov games." *Cognitive systems research* 2.1 (2001): 55-66.

[6] Kar, Soummya, José MF Moura, and H. Vincent Poor. QD-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through Consensus + Innovations." IEEE Transactions on Signal Processing 61.7 (2013): 1848-1862

[7] Shapley, Lloyd S. "Stochastic games." Proceedings of the national academy of sciences 39.10 (1953): 1095-1100.

[8] Littman, Michael L. "Markov games as a framework for multi-agent reinforcement learning." Machine learning proceedings 1994. Morgan Kaufmann, 1994. 157-163.

[9] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8 (1992): 279-292.

[10] https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14

[11] https://medium.com/intro-to-artificial-intelligence/deep-deterministic-policy-gradient-ddpg-an-off-policy-reinforcement-learning-algorithm-38ca8698131b

[12] https://medium.com/@amaresh.dm/how-ddpg-deep-deterministic-policy-gradient-algorithms-works-in-reinforcement-learning-117e6a932e68

[13] https://medium.com/data-science-in-your-pocket/deep-q-networks-dqn-explained-with-examples-and-codes-in-reinforcement-learning-928b97efa792

[14] https://medium.com/geekculture/introduction-to-deterministic-policy-gradient-dpg-e7229d5248e2

[15] http://proceedings.mlr.press/v32/silver14.pdf

[16] https://medium.com/geekculture/introduction-to-deterministic-policy-gradient-dpg-e7229d5248e2

[17] https://www.youtube.com/watch?v=tZTQ6S9PfkE

[18] https://arxiv.org/abs/1706.02275

[19] https://medium.com/machine-intelligence-and-deep-learning-lab/a-tutorial-on-maddpg-53241ae8aac

[20] https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

[21] https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network#:~:text=Here%20are%20the%20steps%20of,buffer%20to%20store%20past%20experiences.

[22] https://www.youtube.com/watch?v=ljHwbY9QrJU

[23] https://medium.com/@arshren/step-by-step-guide-to-implementing-ddpg-reinforcement-learning-in-pytorch-9732f42faac9

[24] https://medium.com/data-science-in-your-pocket/deep-deterministic-policy-gradient-ddpg-explained-with-codes-in-reinforcement-learning-5825fbdc77b2