University of Tehran

Faculty of Engineering

School of Electrical and Computer Engineering

Distributed Optimization and Learning

Project 3

Distributed Cooperative Multi-Agent Reinforcement Learning in Markov Games

Hafez Ghaemi

810199239

Winter 2022

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. Afterwards, we will implement four MARL algorithms to solve multi-agent tasks modeled by cooperative Markov games. The first three algorithms are all value-based and include Distributed Q-Learning, Team Q-Learning, and QD-Learning. The first two algorithms do not require communication among agents for convergence but only converge in deterministic MDPs, while QD-Learning is communication-based. This algorithm and the fourth algorithm which is a communication-based actor-critic algorithm for cooperative agents maximize the sum of expected return of all agents.

1. 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.

2. Algorithms

2.1. 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.

2.2. Team Q-Learning

The first MARL algorithm considered is Team Q-Learning [4] in which each agent has to observe the actions of all other agents, and also needs to store a Q-table with a dimensionality of $S \times A_1 \times \ldots \times A_n$ which makes it unfeasible for large action spaces. The update of Q-values is as follows:

$$\begin{aligned} Q_1[s, a_1, \ldots, a_n] &:= (1 - \alpha) Q_1[s, a_1, \ldots, a_n] \\ &\quad + \alpha(r_1 + \beta \mathrm{Val}_1(s, Q_1)). \end{aligned}$$

where,

$$\mathrm{Val}_1(s, Q_1) = \max_{a_1, \ldots, a_n} Q_1[s, a_1, \ldots, a_n].$$

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

2.3. 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:

$$\begin{aligned} Q_{i,k+1}(x_k, u_{i,k}) = \max\Big\{ & Q_{i,k}(x_k, u_{i,k}), \\ & r_{k+1} + \gamma \max_{u_i} Q_{i,k}(x_{k+1}, u_i)\Big\}. \end{aligned}$$

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.

2.4. QD-Learning

The QD-Learning algorithm [6] is a cooperative distributed MARL algorithm over a communication graph that minimizes the sum of expected costs of all agents, and the local Q values reach a consensus. The distributed Q update is as follows (this algorithm assumes the minimization of cost and not the maximization of reward):

$$
\begin{aligned}
Q^n_{i,u}(t+1) = Q^n_{i,u}(t) - \beta_{i,u}(t) \sum_{l \in \Omega_n(t)} \left(Q^n_{i,u}(t) - Q^l_{i,u}(t) \right) \\
+\alpha_{i,u}(t) \left(c_n(\mathbf{x}_t, \mathbf{u}_t) + \gamma \min_{v \in \mathcal{U}} Q^n_{\mathbf{x}_{t+1},v}(t) - Q^n_{i,u}(t) \right),
\end{aligned}
$$

Where the first term is for consensus and the second one is for learning the optimal policy. learning rates are:

$$
\beta_{i,u}(t) = \begin{cases} \frac{b}{(k+1)^{\tau_2}} & \text{if } t = T_{i,u}(k) \text{ for some } k \geq 0 \\ 0 & \text{otherwise,} \end{cases}
$$

$$
\alpha_{i,u}(t) = \begin{cases} \frac{a}{(k+1)^{\tau_1}} & \text{if } t = T_{i,u}(k) \text{ for some } k \geq 0 \\ 0 & \text{otherwise,} \end{cases}
$$

The Q values converge and reach a consensus for each state-action pair for all players, and the algorithm minimizes the sum of expected costs of all players (which shows the communication-based nature of the algorithm).

2.5. Networked Actor-Critic

The last algorithm considered is Distributed Actor-Critic for Networked Agents [1-3]. This algorithm also utilizes a communication graph and also works in continuous state-actions spaces due to its use of function approximation for Q values and parametrized policies.

Algorithm 1 The networked actor-critic algorithm based on action-value function

```
Input: Initial values of the parameters $\mu_0^i, \omega_0^i, \widetilde{\omega}_0^i, \theta_0^i, \forall i \in \mathcal{N}$; the initial state $s_0$ of the MDP, and
stepsizes $\{\beta_{\omega,t}\}_{t\geq 0}$ and $\{\beta_{\theta,t}\}_{t\geq 0}$.
Each agent $i \in \mathcal{N}$ executes action $a_0^i \sim \pi_{\theta_0^i}^i(s_0, \cdot)$ and observes joint actions $a_0 = (a_0^1, \ldots, a_0^N)$.
Initialize the iteration counter $t \leftarrow 0$.
Repeat:
    for all $i \in \mathcal{N}$ do
        Observe state $s_{t+1}$, and reward $r_{t+1}^i$.
        Update $\mu_{t+1}^i \leftarrow (1 - \beta_{\omega,t}) \cdot \mu_t^i + \beta_{\omega,t} \cdot r_{t+1}^i$.
        Select and execute action $a_{t+1}^i \sim \pi_{\theta_t^i}^i(s_{t+1}, \cdot)$.
    end for
    Observe joint actions $a_{t+1} = (a_{t+1}^1, \ldots, a_{t+1}^N)$.
    for all $i \in \mathcal{N}$ do
        Update $\delta_t^i \leftarrow r_{t+1}^i - \mu_t^i + Q_{t+1}(\omega_t^i) - Q_t(\omega_t^i)$.
        Critic step: $\widetilde{\omega}_t^i \leftarrow \omega_t^i + \beta_{\omega,t} \cdot \delta_t^i \cdot \nabla_\omega Q_t(\omega_t^i)$.
        Update $A_t^i \leftarrow Q_t(\omega_t^i) - \sum_{a^i \in \mathcal{A}^i} \pi_{\theta_t^i}^i(s_t, a^i) \cdot Q(s_t, a^i, a^{-i}; \omega_t^i)$,    $\psi_t^i \leftarrow \nabla_{\theta^i} \log \pi_{\theta_t^i}^i(s_t, a_t^i)$.
        Actor step: $\theta_{t+1}^i \leftarrow \theta_t^i + \beta_{\theta,t} \cdot A_t^i \cdot \psi_t^i$.
        Send $\widetilde{\omega}_t^i$ to the neighbors $\{j \in \mathcal{N} : (i,j) \in \mathcal{E}_t\}$ over the communication network $\mathcal{G}_t$.
    end for
    for all $i \in \mathcal{N}$ do
        Consensus step: $\omega_{t+1}^i \leftarrow \sum_{j \in \mathcal{N}} c_t(i,j) \cdot \widetilde{\omega}_t^j$.
    end for
    Update the iteration counter $t \leftarrow t + 1$.
Until Convergence
```

The Networked AC algorithm maximizes the sum of expected return for all agents in a collaborative fashion. To validate its results, we can use the centralized AC algorithm as below:

$$\mu_{t+1} = (1 - \beta_{\omega,t}) \cdot \mu_t + \beta_{\omega,t} \cdot r_{t+1}, \quad \omega_{t+1} = \omega_t + \beta_{\omega,t} \cdot \delta_t \cdot \nabla_\omega Q_t(\omega_t), \quad \theta_{t+1} = \theta_t + \beta_{\theta,t} \cdot A_t \cdot \psi_t,$$

And the TD-error is:

$$\delta_t = r_{t+1} - \mu_t + Q(s_{t+1}, a_{t+1}; \omega_t) - Q(s_t, a_t; \omega_t)$$

3. Results

3.1. Single-Agent Q-Learning

Simulation setup:

Number of states: 6

Number of actions per state: 5

Rewards generated from a uniform distribution from -0.5 to 0.5 multiplied by $e^{(s^2)}$ and then normalized between -1 and 1.

Ergodic MDP with random transitions

Discount = 0.8

1000 episodes and 100 iterations per episode.

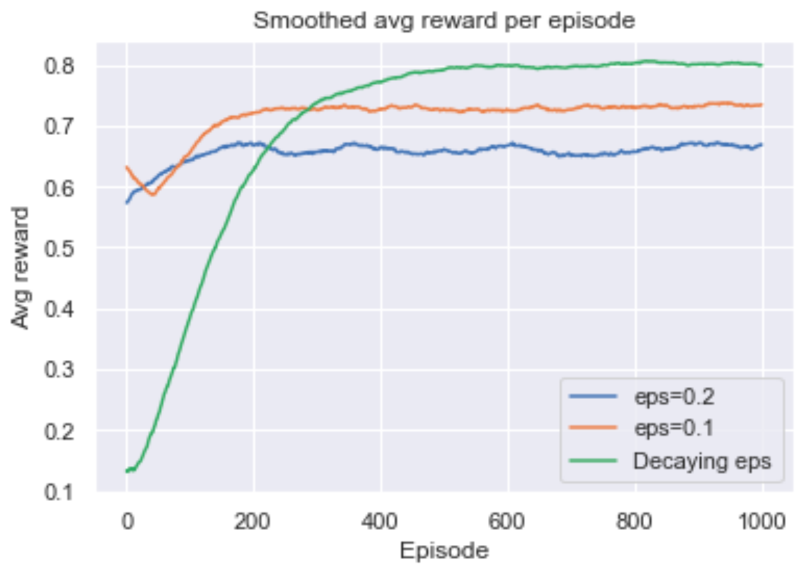
Lr = 0.1

Eps decay is from 1.0 with 0.99 rate.

Number of iterations per episode: 100

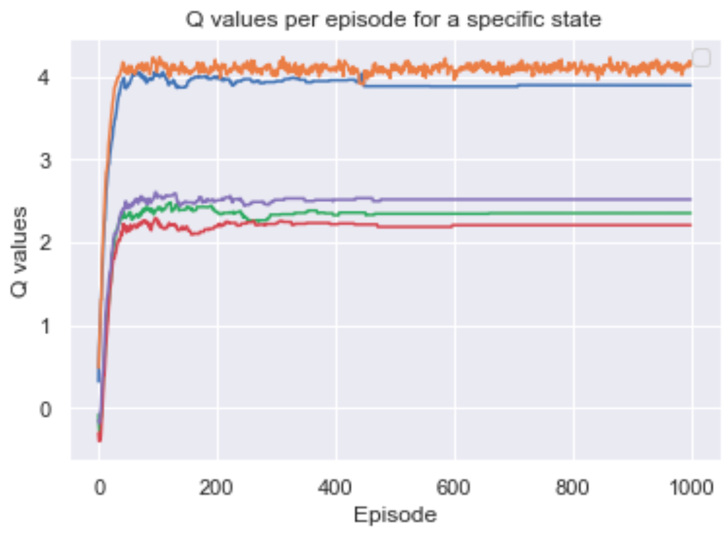
Number of episodes: 1000

We run the simulation for 1000 episodes and calculate the average return for each episode. The smoothed average reward is plotted for two different constant epsilon values and decaying epsilon.



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

The plot below shows the Q values for a specific state during learning. It can be seen that as the epsilon decays, mostly the Q value corresponding to the optimal action and not others are being updated.



3.2. Team Q-Learning

Simulation setup:

Number of states: 6

Number of actions per state: 5

Number of players: 4

Rewards generated from a uniform distribution between -0.5 and 0.5, multiplied by $e^{(s^2)}$ and then normalized between -1 and 1.

Deterministic transitions in the MDP

Deterministic transitions

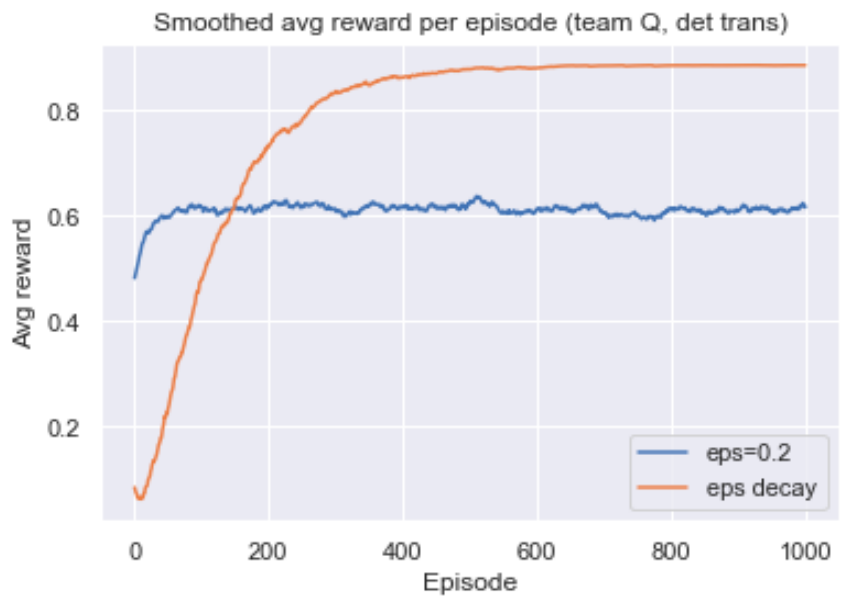
Discount = 0.8

1000 episodes and 100 iterations per episode.

Learning rate = 0.1

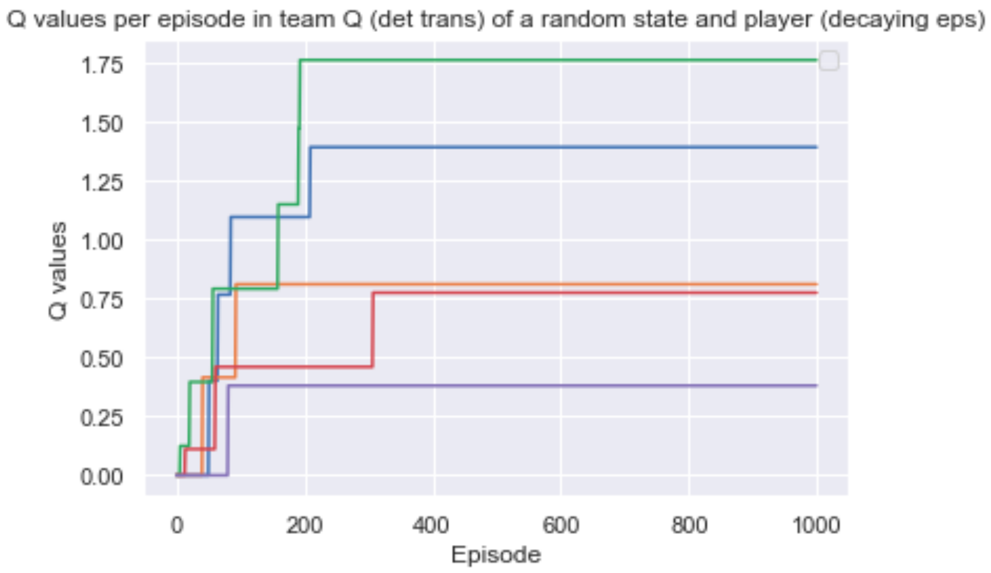
Eps decay is from 1.0 with 0.99 rate.

The smoothed average reward plot below shows that epsilon decay has an optimum performance in Team Q-Learning.



As the MDP is deterministic, the average return has converged to the maximum possible return in each state which is 1.

As shown in the the plot below, the global Q-values also converge, and their fluctuations are not noisy because of the deterministic MDP.



3.3. Distributed Q-Learning

Simulation setup:

Number of states: 6

Number of actions per state: 5

Number of players: 4

Rewards generated from a uniform distribution between -0.5 and 0.5, multiplied by $e^{(s^2)}$ and then normalized between -1 and 1.

Deterministic transitions in the MDP

Deterministic transitions

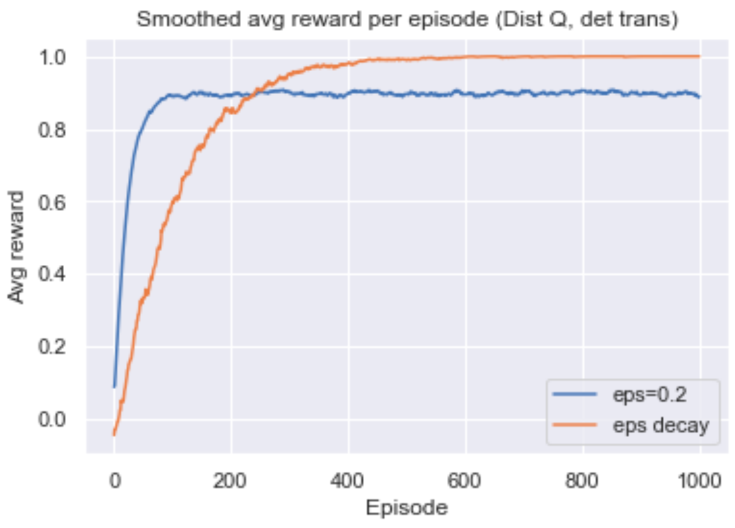
Discount = 0.8

1000 episodes and 100 iterations per episode.

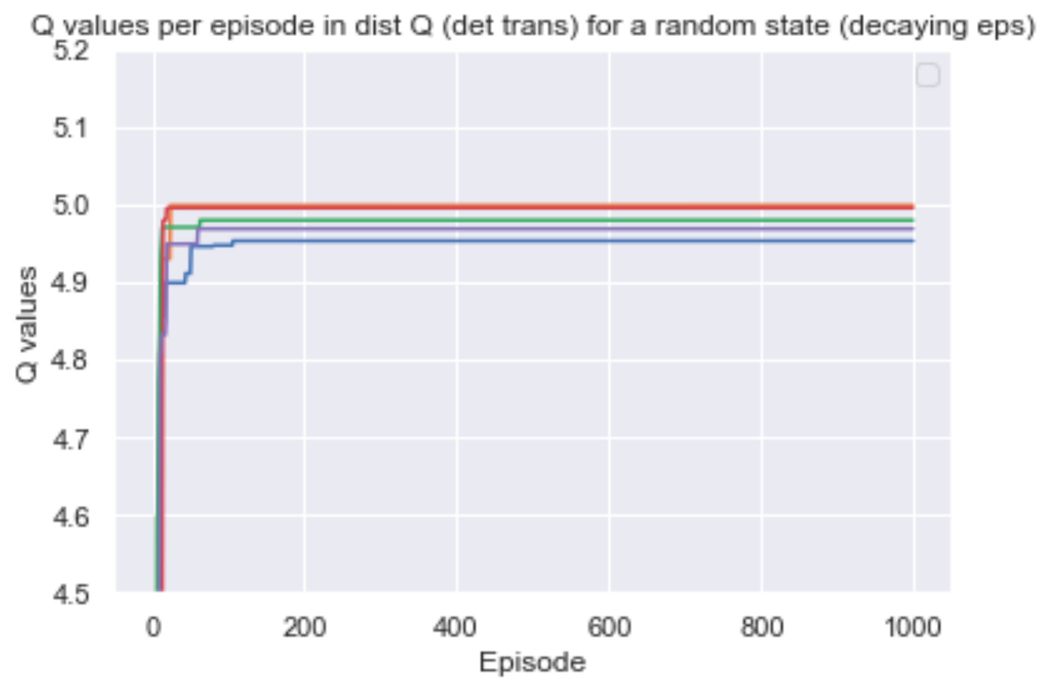
Learning rate = 0.1

Eps decay is from 1.0 with 0.99 rate.

The smoothed average reward plot below shows that epsilon decay has an optimum performance in Distributed Q-Learning.



Due to the nature of the update and the existence of the max operator, the converged Q-values have low differences.



We can see that although epsilon-greedy causes the average return to converge slowly, the optimal policy has been found after around 150 episodes.

3.4. QD-Learning

Simulation setup:

Number of states: 2

Number of actions per state: 2

Number of players: 10

Costs (the algorithm tries to minimize the cost) for each player and state action pair $c_n(i, a)$ come from a gaussian distribution with variance 10 and a mean uniformly sampled from [0, 100].

The action trajectories for the learning process are generated randomly over time. The state trajectories are determined by probabilities $p_{x_t}^{u_t}$.

Discount = 0.7

1000 timesteps

Learning rates:

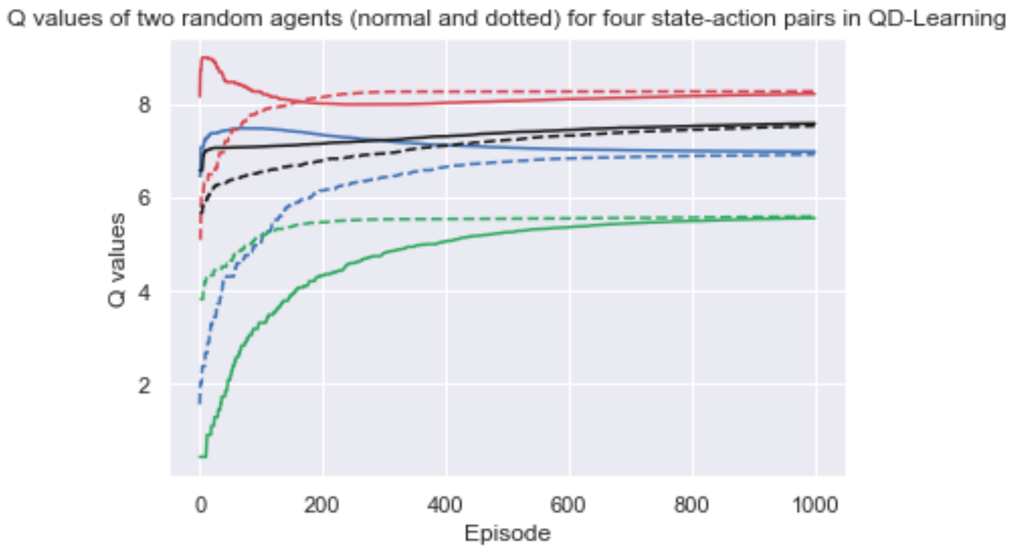
$$\beta_{i,u}(t) = \begin{cases} \frac{b}{(k+1)^{\tau_2}} & \text{if } t = T_{i,u}(k) \text{ for some } k \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

$$\alpha_{i,u}(t) = \begin{cases} \frac{a}{(k+1)^{\tau_1}} & \text{if } t = T_{i,u}(k) \text{ for some } k \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

where τ_1=1 and τ_2=0.2 and a=b=0.01.

The communication graph has a 0.5 probability of link erasure.

After running the simulation, we can observe both the convergence and consensus by plotting the four Q values of state-action pairs of two different agents (one indicated by normal lines and the other by dotted lines):



3.5. Multi-Agent Actor-Critic for Networked Agents

Simulation setup:

Number of states: 10

Number of actions per state: 2

Number of players (N): 10

Means of $R_i(s,a)$ are uniformly sampled from [0, 4] and varies for each agent. Then the instantaneous $R_i(s,a)$ is sampled uniformly from $[R_i(s,a) - 0.5, R_i(s,a) + 0.5]$.

Stochastic transitions in the MDP (ergodic)

No discount factor.

500 time steps of simulation.

The vectors $q_{s,i}$ and θ_i are have a dimension of 5, and the feature vector for linear function approximation of Q_{s,a_i} has a dimension of 10.

The learning rates are $\beta_{\theta,t} = 1/t^{0.85}$ and $\beta_{w,t} = 1/t^{0.65}$.

The consensus weight matrix is initialized randomly and with a 4/N connectivity ratio, and it is normalized afterwards to become doubly stochastic.

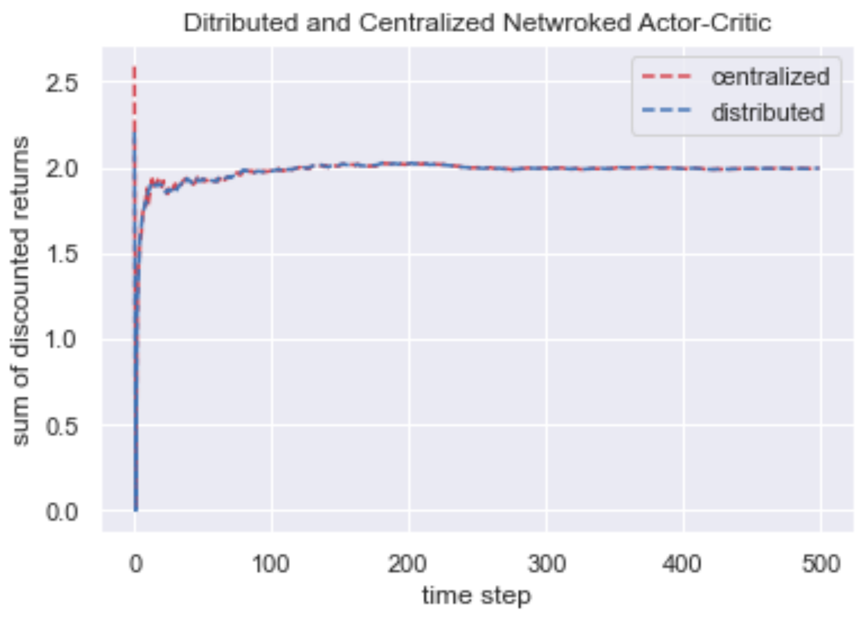
The policy is a Boltzman policy,

$$\pi^i_{\theta^i}(s,a^i) = \frac{\exp\left(q^\top_{s,a^i}\theta^i\right)}{\sum_{b^i \in \mathcal{A}^i} \exp\left(q^\top_{s,b^i}\theta^i\right)}$$

And therefore, its gradient is

$$\nabla_{\theta^i} \log \pi^i_{\theta^i}(s,a^i) = q_{s,a^i} - \sum_{b^i \in \mathcal{A}^i} \pi^i_{\theta^i}(s,a^i) q_{s,b^i}.$$

After running the distributed and centralized networked AC algorithms, we observe that the sum of discounted returns converges:



The advantage of the networked AC algorithm is that the number of time-steps required for convergence is small due to the communication-based nature of the algorithm.

4. Conclusion

In this project, we considered identical-interest Markov games and implemented four MARL algorithms to solve the corresponding multi-agent MDP. We also implemented vanilla Q-learning 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 Team Q-learning and Distributed Q-learning algorithms have a very narrow scope and are only applicable to deterministic identically-interest MDPs. Furthermore, Team Q-learning suffers from curse of dimensionality as we need to store a large Q-table considering all other agents' actions.
- QD-Learning and Networked AC have relatively fast convergence (AC is faster) and optimize the sum of expected return of all agents, but they require a communication graph to operate.
- Since Networked AC uses function approximation for Q function and parameterized policies, it can be easily extended to continuous action spaces [3].
- 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.