Multi-Agent Reinforcement Learning

Theory and Examples

Quantitative Life Sciences - ICTP 2024

Cristopher Erazo

July 1, 2024

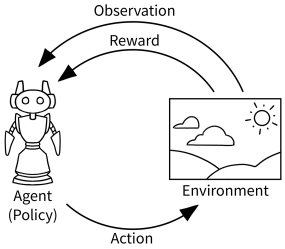
The Abdus Salam
International Centre
for Theoretical Physics

Table of Contents

List

Ingredients of Single Agent Reinforcement Learning
Markov Decision Process

- $\mathcal{S}$ finite set of environment states $s \in \mathcal{S}$.
- $\mathcal{A}$ finite set of actions $a \in \mathcal{A}$.
- $p(s'|s,a)$ state transition probability.
- $r(s,a)$ reward function.
- $\pi(a|s)$ policy.



At time t, the environment is at s_t. The agent observes the state and takes an action $a_t \sim \pi(\cdot|s_t)$. As a result: $s_{t+1} \sim p(\cdot|s_t, a_t)$ and the reward received is $r_{t+1} = r(s_t, a_t)$.

Goal of Reinforcement Learning

Value functions

The reward only evaluates the immediate effect of the action taken by the agent.

- Value Function: $V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) | S_0 = s, \pi\right]$
- State-Action Value Function: $Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) | S_0 = s, A_0 = a, \pi\right]$
- Objective Function: $G(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) | \pi, \rho_0\right]$ for $S_0 \sim \rho_0$

Goal

$$\pi^* = \arg\max_{\pi} G(\pi) \quad \Leftrightarrow \quad \pi^*(a|s) = \arg\max_{a} Q^*(s, a)$$

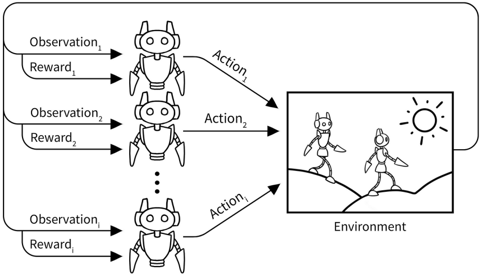
Table of Contents

List

Multi Agent Reinforcement Learning

Stochastic Games

- N agents.
- $\mathcal{S}$ finite set of environment states.
- $\mathcal{A}_n$ finite set of actions for agent n.
- $p(s'|s,\mathbf{a})$ state transition probability with $\mathbf{a} \in \prod_n \mathcal{A}_n$ being the joint action.
- $r_n(s,\mathbf{a})$ reward function for agent n that depends on the joint action taken.



The n-th policy is $\pi^n(a_n|s)$ with $a_n \in \mathcal{A}_n$ and the joint policy is $\boldsymbol{\pi}(\mathbf{a}|s) = \prod_n \pi^n(a_n|s)$.

Value Functions in MARL

Direct Generalization

For agent n we define:

- Value Function: $V_n^{\boldsymbol{\pi}}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, \mathbf{A}_t) | S_0 = s, \boldsymbol{\pi}\right]$
- State-Action Value Function: $Q_n^{\boldsymbol{\pi}}(s, \mathbf{a}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_n(S_t, \mathbf{A}_t) | S_0 = s, \mathbf{A}_0 = \mathbf{a}, \boldsymbol{\pi}\right]$
- Objective Function: $G_n(\boldsymbol{\pi}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_n(S_t, \mathbf{A}_t) | \boldsymbol{\pi}, \rho_0\right]$

The value functions and the objective are computed with the joint policy and therefore now the returns are correlated among all the agents.

Game Theory
Static Games

A static game has no states ($\mathcal{S} = \emptyset$). Is described only by the set of actions $\mathcal{A}_n$ and the reward functions $r_n(\mathbf{a})$. The policy becomes a strategy $\pi^n(a_n)$ as in Game Theory.

Expected utility: $u_n(\boldsymbol{\pi}) = \mathbb{E}\left[r_n(\mathbf{a})|\boldsymbol{\pi}\right]$ where $\boldsymbol{\pi}(\mathbf{a}) = \prod_n \pi^n(a_n)$ is the joint strategy.

Can be separated:

$$u_n(\boldsymbol{\pi}) = u_n(\pi^n, \boldsymbol{\pi}^{-n}) = \sum_{a_n, \mathbf{a}_{-n}} r_n(a_n, \mathbf{a}_{-n})\pi^n(a_n)\boldsymbol{\pi}^{-n}(\mathbf{a}_{-n}) \tag{1}$$

where $(-n)$ refers to all the other agents appart from n.

Why Game Theory?

Important Notions

Many MARL algorithms are designed for static games or work stage-wise.

Stage Game = Static Game that arises in a fixed state of the Stochastic Game.
Reward Functions = Q functions projected in action space at fixed state.
Strategy = Projection of the policy in action space at fixed state.

MARL learn strategies stage-wise and then aggregate them to obtain the overall policy.

Important Concepts in Game Theory

The **best response** for agent n is:

$$BR^n(\boldsymbol{\pi}^{-n}) = \arg\max_{\pi^n} u_n(\pi^n, \boldsymbol{\pi}^{-n})$$

Profile π^* is a **Nash Equilibrium** if:

$$\pi^n_* = BR^n(\boldsymbol{\pi}_*^{-n}) \; \forall n$$

Table of Contents

List

Benefits and Challenges

In a Nutshell

Benefits:

- Experience sharing can help agents with similar tasks to learn faster.
- MARL can be implemented with parallel computation when the task is decentralized.
- MARL is more robust because when one agent fails other one can take over some of the tasks.

Challenges:

- Course of dimensionality more severe because of the exponential growth in the state-action space with the number of agents.
- Specifying a good goal in MARL is difficult. The agent returns are correlated and cannot be maximized independently.
- Non-stationaty arises in MARL because all the agents learn simultaneously. Each agent faces a moving-target learning problem.

MARL Goal

In a Nutshell

Specifying a good general MARL goal is **challenging** because the agent's returns are different and correlated. In the literature, the goals are generally tracked within two edges: **Stability** and **Adaptation**.

- **Stability:** Convergence to equilibrium requirement. **Nash equilibrium** are the most frequently used, but there is not a consensus in weather it is useful. A normal objection is that the relation between stage-wise convergence and convergence in the stochastic game is still unclear.
- **Adaptation:** One criteria is rationality: convergence to a best response equilibrium if the others remain stationary. In this cases Nash equilibrium arises naturally combining convergence and rationality. Another technique is **no-regret** which means that the agent achieves a return that is at least as good as any stationary strategy.

MARL Goal
In a Nutshell

Some other properties can be related to **stability** and **adaptation**.

Opponent-independent algorithms converges to a stable strategies regardless of what the other agents are doing

Opponent-aware algorithms lean models of the other agents and and use them in some way of best response and is related to adaptation.

A good combination of both axis should be designed according to each problem. Stability is needed because stable agents are simpler to analyze and have more meaningful performance guarantees. Adaptation is needed because the behaviour of other agents is generally unpredictable.

MARL Algorithms
Classification

The classification of MARL algorithms encompasses several dimensions and in a simplified way can be can be separated by **task type**.

MARL Algorithms

Fully cooperative tasks

In this case $(r_1 = \cdots = r_n)$ and the goal is to maximize the common objective function. This is reduced to a MDP if a centralized controller is available $\pi(a_1, \cdots, a_n|s)$. Complications might arise because, multiple joint actions may be optimal. In absence of coordination mechanisms, the agents might pick sub-optimal joint action. Several solutions are proposed:

- **Team Q learning:** avoids coordination assuming that the optimal joint actions are unique (not always true).
- **Distributed Q learning:** solves cooperation without assuming coordination, only works in deterministic problems with non-negative rewards.
- **Coordination based:** when global Q functions can be decomposed additive into local Q functions of subset of agents.

MARL Algorithms

Fully competitive tasks

In this case (for $N = 2$ is equivalent to $r_1 = -r_2$) the minimax principle can be applied: maximize your benefit under the worst case assumption that the opponent will always try to minimize it. This priciple suggests a opponent-independent algorithm.

- **Minimax Q:** similar to Q learning introducing minimax operator.

$$\boldsymbol{m_1}(Q, s) = \max_{\pi_1(\cdot|s)} \min_{a_2} \sum_{a_1} \pi^1(a_1|s) Q_1(s, a_1, a_2) \tag{2}$$

The schematic update rule will be:

$$Q_1^{t+1} \leftarrow Q_1^t + \alpha_t (Q_1^t + \gamma \boldsymbol{m_1}(Q_1^t) - Q_1^t) \tag{3}$$

MARL Algorithms

Mixed Task

No constrains imposed on the reward functions of the agents. Self-interested, but not necessarily competing agents. Here the influence of Game Theory is the strongest.

- **Single - agent RL:** Algorithms for single agents like Q-learning can be applied directly. The non-stationarity of the problem invalidates most of the single-agent RL theoretical guarantees, nonetheless, it has a significant number of applications because of its simplicity.
- **Agent - independent methods:** Nash Q-learning modifies the Q-learning update rule to include a convergence to equilibrium (probably Nash) in each state.
- **Agent - tracking methods:** Estimate models of the other agents strategies or policies and act using some form of best response.
- **Agent - aware methods:** They target convergence as well as adaptation to other agents.

Table of Contents

List

Application in a Matrix Game

Natural Policy Gradient

N independent agents with action sets $\mathcal{A}_\backslash$. The reward functions are $r_n(\mathbf{a})$ and depends on the joint action $\mathbf{a}$. The policy is:

$$\pi^n_{\theta^n}(a) = \frac{e^{\theta^n_a}}{\sum_{a' \in \mathcal{A}_n} e^{\theta^n_a}} \text{ where } \theta^n = (\theta^n_a)_{a \in \mathcal{A}_n} \text{ is the set of parameters} \tag{4}$$

The goal for each agent is to maximize its expected utility:

$$u_n(\boldsymbol{\theta}) = u_n(\theta^n, \boldsymbol{\theta}^{-n}) = u_n(\pi^n_{\theta^n}, \boldsymbol{\pi}^{-n}_{\boldsymbol{\theta}^{-n}}) = \sum_{a_n, \mathbf{a}_{-n}} r_n(a_n, \mathbf{a}_{-n}) \pi^n_{\theta^n}(a_n) \boldsymbol{\pi}^{-n}_{\boldsymbol{\theta}^{-n}}(\mathbf{a}_{-n}) \tag{5}$$

Application in a Matrix Game

Natural Policy Gradient

The update rule for the parameters will follow the natural gradient:
$\theta_a^n \leftarrow \theta_a^n + \eta\tilde{\nabla}_a u_n(\theta^n)$ where $\tilde{\nabla} = \mathbb{J}^{-1}\nabla$ is the natural gradient and $\mathbb{J}$ is the Fisher matrix of the policy and η is the learning rate ($\tilde{\nabla}_a \equiv \tilde{\nabla}_{\theta_a^n}$)

If we apply the natural gradient to the expected utility we will obtain:

$$\tilde{\nabla}_a u_n(\theta^n) \sum_{a_n, \mathbf{a}_{-n}} r_n(a_n, \mathbf{a}_{-n}) \left[\tilde{\nabla}_a \pi_{\theta^n}^n(a_n)\right] \boldsymbol{\pi}_{\boldsymbol{\theta}^{-n}}^{-n}(\mathbf{a}_{-n})$$

The natural gradient of the policy as $\tilde{\nabla}_a \pi_{\theta^n}^n(a_n) = \delta_{a,a_n}$ and we obtain:

$$\tilde{\nabla}_a u_n(\theta^n) \equiv u^n(a, \boldsymbol{\pi}^{-n}) = \sum_{\mathbf{a}_{-n}} r_n(a, \mathbf{a}_{-n}) \boldsymbol{\pi}_{\boldsymbol{\theta}^{-n}}^{-n}(\mathbf{a}_{-n})$$

Application in a Matrix Game

Natural Policy Gradient

The great advantage of the softmax parametrization along with the natural gradient in this case is that now is very easy to construct an estimator of the gradient based on samples: if in a given step, agent n plays action A_n when the other agents play the joint action $\mathbf{A}_{-n}$, the estimator of the gradient will be:

$\widehat{\tilde{\nabla}_a u_n} = r_n(A_n, \mathbf{A}_{-n})$ and the update rule becomes:

$$\theta_a^n \;\leftarrow\; \theta_a^n + \eta\, r_n(A_n, \mathbf{A}_{-n})\delta(A_n = a)$$

Application in a Matrix Game

Replicator Dynamics

From a parallel point of view, we can check what is the update rule for the policy $\pi^n_{\theta^n}(a)$ instead of just its parameters.

$$\pi^n_{\theta^n}(a) \leftarrow \frac{e^{\theta^n_a + \eta\, u^n(a, \boldsymbol{\pi}^{-n})}}{\sum_{a' \in \mathcal{A}_n} e^{\theta^n_{a'} + \eta\, u^n(a', \boldsymbol{\pi}^{-n})}} \approx \frac{e^{\theta^n_a}\left(1 + \eta\, u^n(a, \boldsymbol{\pi}^{-n})\right)}{\sum_{a' \in \mathcal{A}_n} e^{\theta^n_{a'}}\left(1 + \eta\, u^n(a', \boldsymbol{\pi}^{-n})\right)} =$$

$$= \frac{e^{\theta^n_a}\left(1 + \eta\, u^n(a, \boldsymbol{\pi}^{-n})\right)}{\sum_{a' \in \mathcal{A}_n} e^{\theta^n_{a'}} + \eta \sum_{a' \in \mathcal{A}_n} e^{\theta^n_{a'}}\, u^n(a', \boldsymbol{\pi}^{-n})} = \pi^n_{\theta^n}(a) \frac{1 + \eta\, u^n(a, \boldsymbol{\pi}^{-n})}{1 + \eta\, u^n(\boldsymbol{\pi})} \approx$$

$$\pi^n_{\theta^n}(a)\left(1 + \eta\, u^n(a, \boldsymbol{\pi}^{-n}) - \eta\, u^n(\boldsymbol{\pi})\right)$$

Application in a Matrix Game

Replicator Dynamics

In summary, the update rule is:

$$\pi^n_{\theta^n}(a) \quad \leftarrow \quad \pi^n_{\theta^n}(a) + \eta \, \pi^n_{\theta^n}(a) \left[u^n(a, \boldsymbol{\pi}^{-n}) - u^n(\boldsymbol{\pi}) \right]$$

which in the continuous limit $\eta \to 0$ recovers the structure of the replicator dynamics:

$$\frac{d\pi^n(a)}{dt} = \pi^n(a) \left[u^n(a, \boldsymbol{\pi}^{-n}) - u^n(\boldsymbol{\pi}) \right]$$

that is a set of equations found in Evolutionary Game Theory that describes the evolution of a population of individuals with different finesses.

Table of Contents

List

Thank you for listening!
Any questions?