Hands-On Reinforcement Learning

Jiayu Xu

Shanghai University of Finance and Economics

jiayu.xu@163.sufe.edu.cn

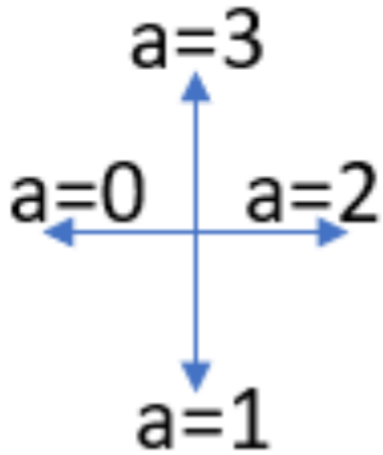
2025 年 5 月 12 日

Presentation Overview

1. Model-based
   - Frozen Lake
   - Policy Iter v.s. Value Iter
2. Model-free
   - Sarsa
   - Q-learning
   - Overestimation
3. Deep Reinforcement Learning
   - Snake
   - DQN
   - Replay Buffer
4. Extension
   - IM+RL
   - MARL
   - Offline RL

Frozen Lake

https://gym.openai.com/



Frozen Lake

规则介绍

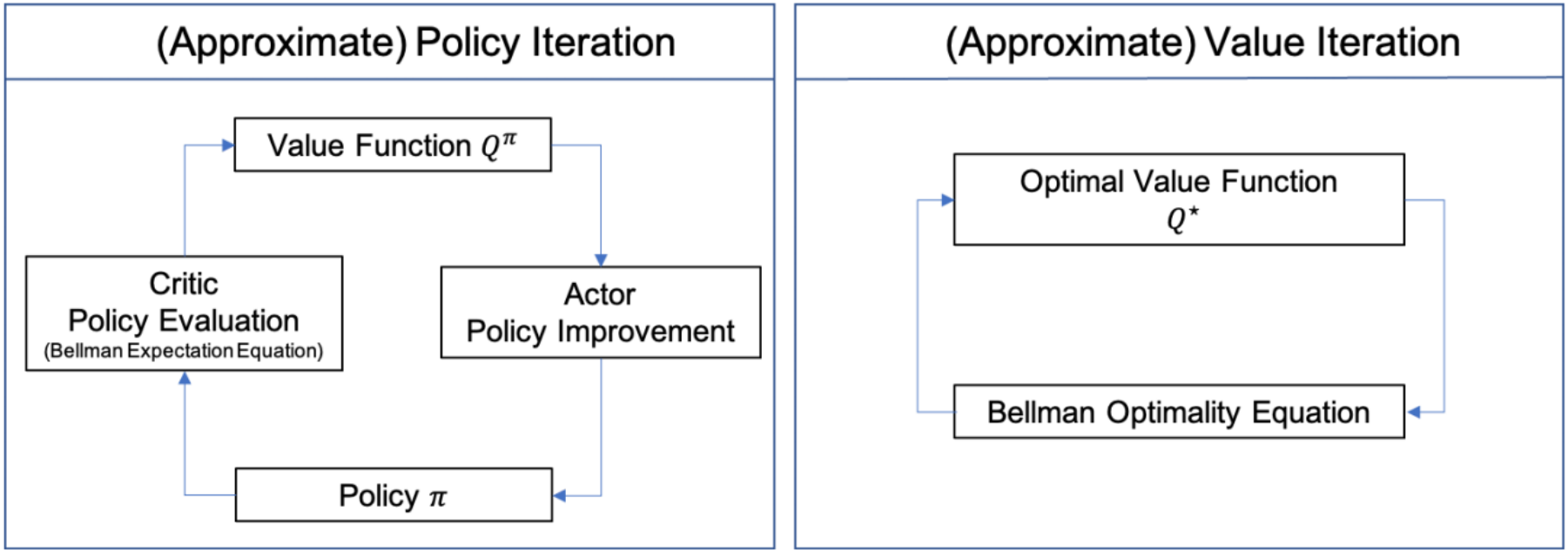
Frozen Lake （冰湖环境）是 Toy 环境的其中一个。它包括

- 离散动作空间 (action space): 长度为 4, 其中 0: 向左, 1: 向下, 2: 向右, 3: 向上
- 离散状态空间 (state space): 长度为 $num\_cols * num\_rows$，包括: 起点 (Start), 目标点 (Goal), 冰洞 (Hole), 冰格 (Frozen)
- 单步奖励 (reward): 到达 Goal: $+1$, 到达 Hole: $+0$, 到达 Frozen: $+0$
- 目标 (objective): 使得智能体 (agent) 从起点到目标点，而不落入任何冰洞中
- 状态转移 (transition): 由于冰冻湖泊的滑溜性质，agent 会以 $1/3$ 的概率向预定的方向移动，或者以 $1/3$ 的概率在两个垂直方向上移动

Frozen Lake

状态转移

Policy Iteration v.s. Value Iteration



- Bellman Expectation Equation

$$Q^\pi(s,a) = E[r + \gamma Q^\pi(s',a')]$$

- Bellman Optimally Equation

$$Q^*(s,a) = E[r + \gamma \max_{a'} Q^*(s',a')]$$

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow false$
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

如果从 policy iteration 角度看 value iteration ?

Sarsa

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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 $Q(\mathit{terminal}, \cdot) = 0$

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

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., ε-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

$$\hat{Q}^{\pi}(s,a) \leftarrow \hat{Q}^{\pi}(s,a) + \alpha\left(r + \gamma \hat{Q}^{\pi}(s', \pi(s')) - \hat{Q}^{\pi}(s,a)\right)$$

$$Q^{k+1}(s,a) \leftarrow Q^{k}(s,a) + \alpha\left(r + \gamma \max_{a'} Q^{k}(s', a') - Q^{k}(s,a)\right)$$

Overestimation

我们令 $x_1, x_2, \ldots, x_n$ 为真实值，添加均值为零的随机噪声给 $x_1, x_2, \ldots, x_n$，生成我们观测到的 $Q_1, Q_2, \ldots, Q_n$

- 均值为零的噪声不会导致均值被高估

$$\mathbb{E}[\text{mean}_i Q_i] \geq \text{mean}_i(x_i)$$

- 均值为零的噪声会导致最大值被高估

$$\mathbb{E}[\max_i Q_i] \geq \max_i(x_i)$$

Double Q-learning

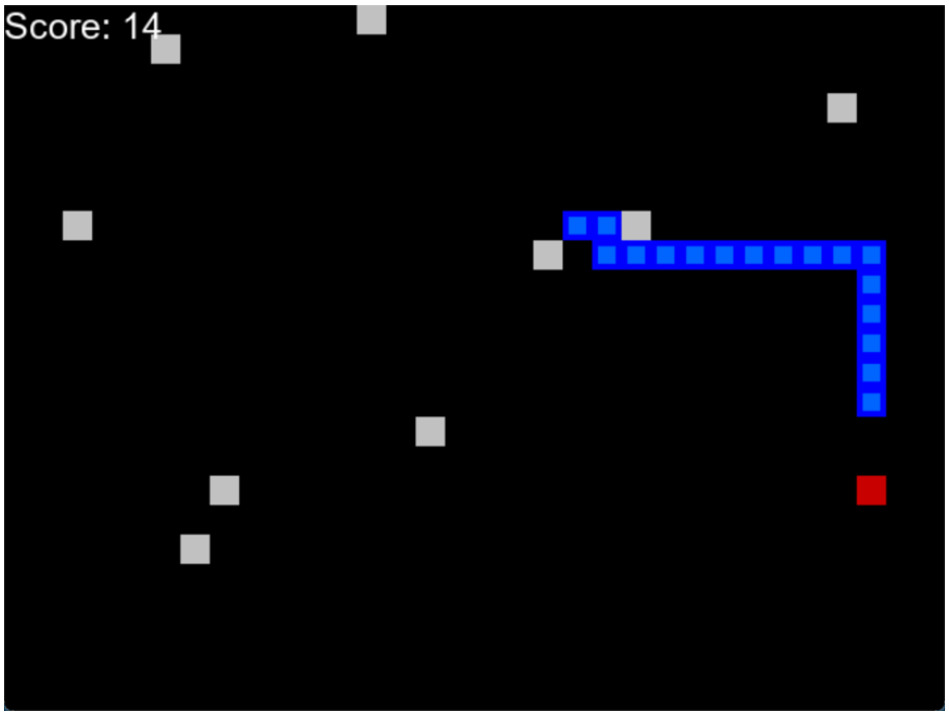
Algorithm 1 Double Q-learning

Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

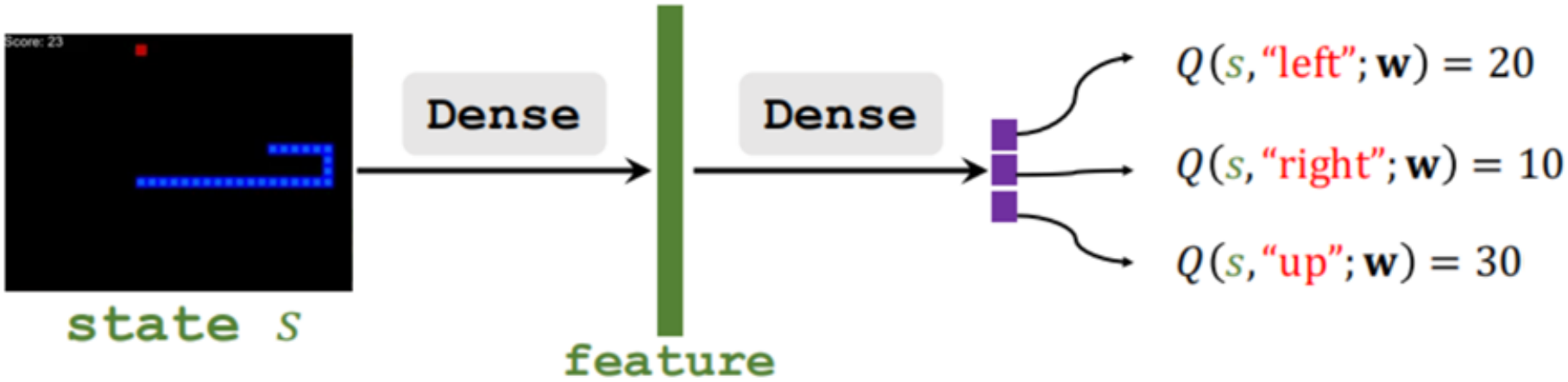
请试着实现这个算法！

贪吃蛇

- 游戏界面共有 32*24 个小格，游戏开始时，会随机生成 10 个障碍物
- 蛇的初始身长设定为 3 格，每吃到一个食物，蛇的长度就会增加一格，游戏分数加一分，且会重新生成一个食物随机出现在其他位置
- 当蛇头碰到自己、墙壁或者障碍物时游戏结束



DQN


Score: 23
Dense
Dense
$Q(s, \text{“left”}; \mathbf{w}) = 20$
$Q(s, \text{“right”}; \mathbf{w}) = 10$
$Q(s, \text{“up”}; \mathbf{w}) = 30$
state s
feature

DQN

- 动作空间：直走，左转，右转
- 状态：进行一定设计降低状态空间大小
  - 直行是否会发生碰撞、左行是否会发生碰撞、右行是否会发生碰撞
  - 当前行进方向
  - 食物相对蛇头的方向位置
- 奖励 (reward)：游戏的得分是一个天然的 reward，但这样的奖励较为稀疏，往往不利于智能体快速高效的学习。因此我们需要基于游戏的规则对 reward 进行设定，以帮助 agent 更快学到正确的行为模式：
  - 发生碰撞，游戏结束-10 分
  - 吃到食物 +10 分
  - 靠近食物 +0.1 分
  - 远离食物-0.2 分

DQN pseudocode

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Replay Buffer

经验回放（Experience Replay）是强化学习中的一个技术，旨在改善学习的效率和稳定性

- 数据复用：同一个经验可以被多次用于训练，提高数据效率
- 去相关：随机抽样打破了数据之间的时间相关性，有助于训练稳定。

prioritized replay: 非均匀抽样，加大对训练重要的样本的抽样概率

- TD error 越大，被抽样到概率越大
- 越时新的 experience，被抽样到的概率越大

开源资料

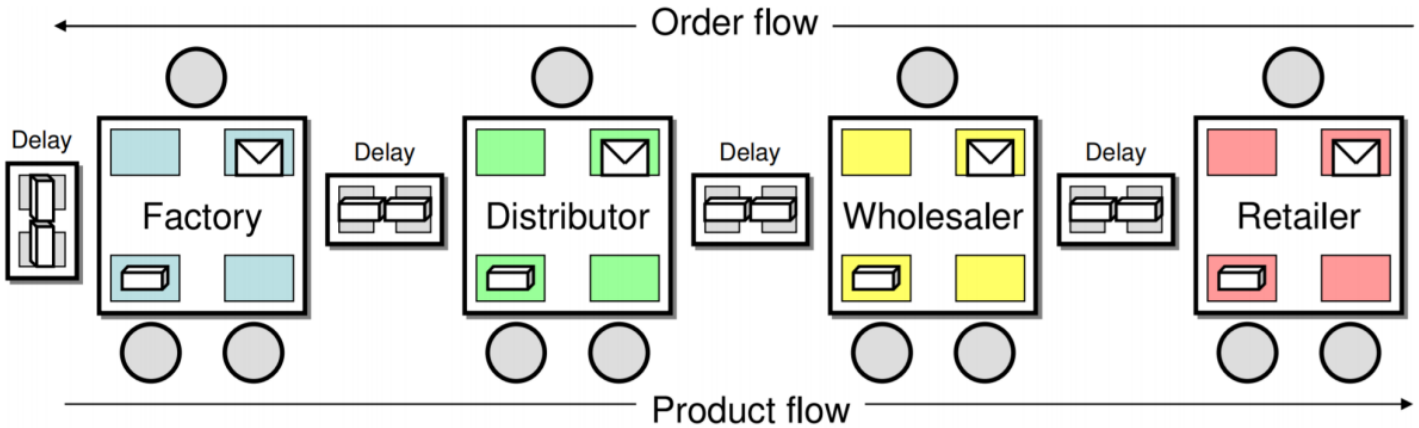
环境：

- openai gym: https://gym.openai.com/
- vizdoom: https://github.com/mwydmuch/ViZDoom
- CARLA: http://carla.org/

代码框架：

- RL-lib: https://docs.ray.io/en/latest/rllib/rllib-algorithms.html
- DRL with Pytorch: https://github.com/p-christ/Deep-ReinforcementLearning-Algorithms-with-PyTorch

IM+RL



- Deterministic transportation lead times l^{tr} and information lead times l^{in}
- In each period of the game, each agent chooses an order quantity q to submit to its predecessor (supplier) in an attempt to minimize the long-run system-wide costs:

$$\sum_{t=1}^{T}\sum_{i=1}^{4} c_h^i\left(IL_t^i\right)^+ + c_p^i\left(IL_t^i\right)^-$$

Huang

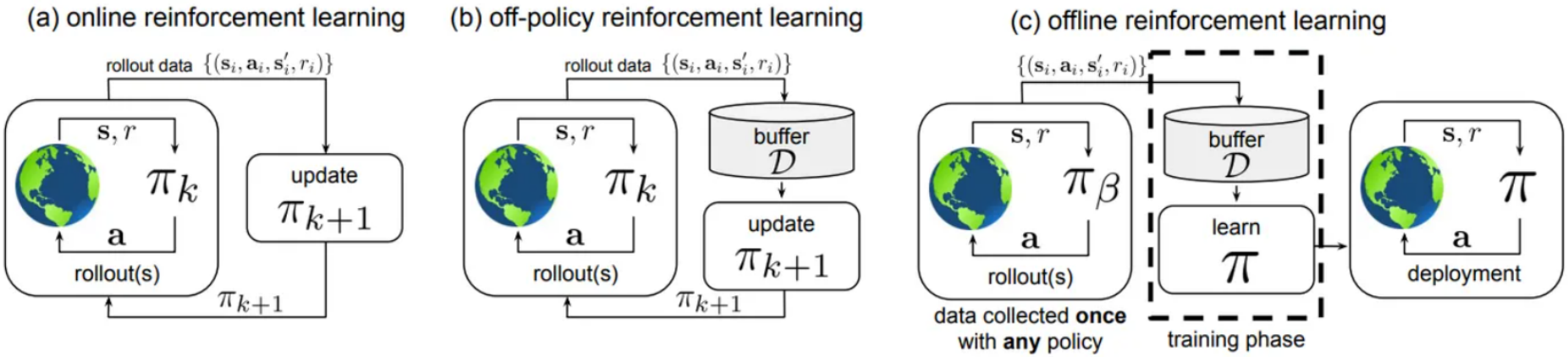
MARL

多个智能体之间的关系有：

- Fully cooperative 相互配合 e.g. 工业机器人
- Fully competitive 一方的收益是另一方的损失（捕猎者和猎物），如零和博弈，双方获得的奖励总和等于 0
- Mixed Cooperative competitive e.g. 足球机器人同队伍的机器人与不同队伍的机器人分别是合作与竞争关系
- Self-interested 利己主义，即每个 agent 只想要最大化自身利益如股票和期货的自动交易系统

主要挑战： non-stationary

Offline RL



online policy 主要分为两类：on-policy RL 和 off-policy RL

- 区别：用来收集数据的 policy(behavior policy) 和我们要优化的 target policy 是否一致

RL 可以分为 online RL 和 offline RL 两类：

- 区别：offline RL 不会与环境进行交互，目的是从一大堆现有的行为数据里面学习到一个更好的策略

主要挑战：distribution shift

The End