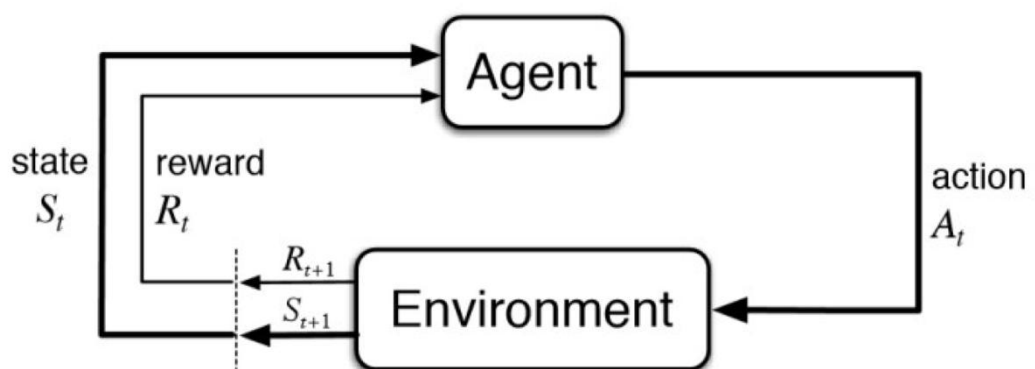


本周主要的理论工作是学习经典的强化学习方法，包括 model-based 的马尔可夫决策过程，model-free 的差分时序模型和 Q-learning。同时，查找并阅读有关基于 RL 的 deployment, consolidation, container migration 的论文。

学习笔记如下：

MDP(马尔可夫决策过程)

1. 基本概念



-
- 马尔可夫决策过程
- agent 在得到环境的状态过后，它会采取动作，它会把这个采取的动作返还给环境。环境在得到 agent 的动作过后，它会进入下一个状态，把下一个状态传回 agent。
- MDP 中，他的环境是全部可观测的
- 马尔可夫性质：从当前状态转移到下一个状态的概率，等于包含当前状态和之前所有状态的序列转移到下一个状态的概率
- 马尔可夫链：一系列从当前状态转移到下一状态的概率，对一系列转移进行采样就得到了轨迹 trajectory

2. 马尔可夫奖励过程：

- 马尔可夫奖励过程(Markov Reward Process, MRP) 是马尔可夫链再加上了一个奖励函数
- 奖励函数：奖励函数 R 是一个期望，就是说你到达某一个状态的时候，可以获得多大的奖励
- 理解：如果把 agent 看作纸船，纸船本身没有动力，放在河里随波逐流(追逐奖励)的过程就是马尔可夫奖励过程
- Horizon: 一个回合的长度（每个回合最大的时间步数），它是由有限个步数决定的。
- Return: 把奖励进行折扣后所获得的收益。Return 可以定义为奖励的逐步叠加，如下式所示：

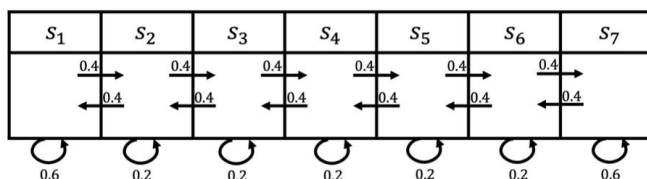
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots + \gamma^{T-t-1} R_T$$

- 价值函数:

$$\begin{aligned} V_t(s) &= E[G_t | s_t = s] \\ &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T | s_t = s] \end{aligned}$$

- 计算 return:

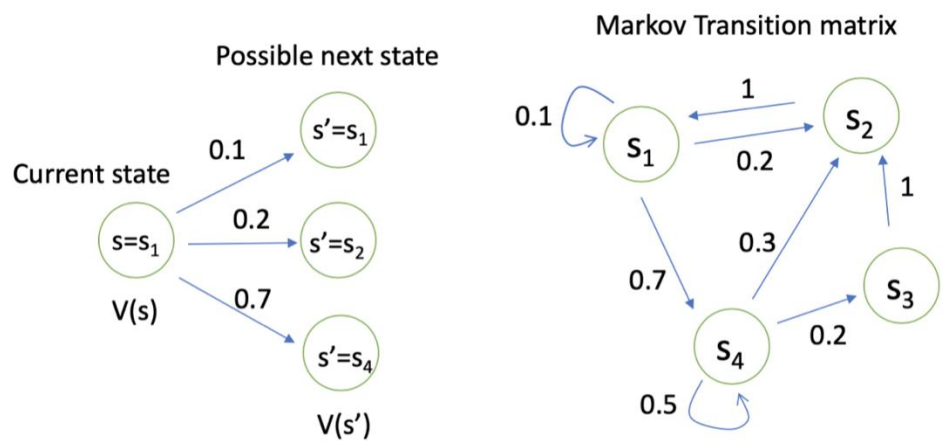
Example of MRP



- ① Reward: +5 in s_1 , +10 in s_7 , 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$
- ② Sample returns G for a 4-step episodes with $\gamma = 1/2$
 - ① return for s_4, s_5, s_6, s_7 : $0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 1.25$
 - ② return for s_4, s_3, s_2, s_1 : $0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 5 = 0.625$
 - ③ return s_4, s_5, s_6, s_7 : = 0
- ③ How to compute the value function? For example, the value of state s_4 as $V(s_4)$

- 在能计算 return 的情况下如何计算某特定状态对应的价值函数:
 - 对于当前状态后的所有 trajectory 的 return 进行采样累加(蒙特卡洛积分)
 - Bellman 等式: 当前状态的 reward 加上未来可能的状态 reward 乘以折扣因子累加

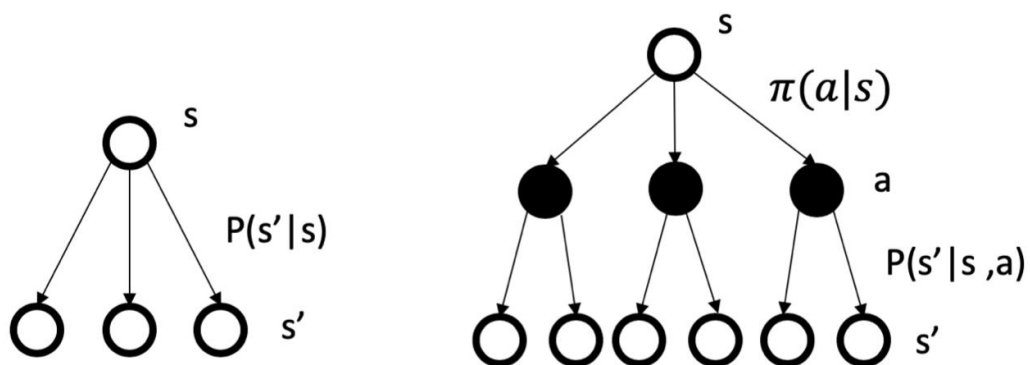
$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} P(s' | s) V(s')}_{\text{Discounted sum of future reward}}$$



-
-
- 计算所有的价值函数：根据 bellman 等式不断迭代直到状态收敛

3. 马尔可夫决策过程

- **状态转移函数**：在马尔可夫奖励过程的基础上多了行为 **action**，采取不同的行动未来的状态会不同
- **奖励**：在马尔可夫奖励过程的基础上多了行为 **action**，奖励根据当前状态当前行为决定
- **决策函数**：输入当前状态，得到当前行动的概率
- 马尔可夫决策过程转为**马尔可夫奖励过程**：
 - 状态转移函数为采取全部可能行动得到的状态概率积分
 - 奖励为采取全部可能行动得到的当前状态奖励均值




- 价值函数 Q-函数：

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} [G_t | s_t = s, A_t = a]$$
- 价值函数：价值函数就是所有动作可能性的 q 函数的加和

$$q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) \sum_{a' \in A} \pi(a' | s') q^{\pi}(s', a')$$

- 计算示例:

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

- ① $R = [5, 0, 0, 0, 0, 0, 10]$
- ② Practice 1: Deterministic policy $\pi(s) = \text{Left}$ with $\gamma = 0.5$ for any state s , then what are the state values under the policy?
- ③ Practice 2: Stochastic policy $P(\pi(s) = \text{Left}) = 0.5$ and $P(\pi(s) = \text{Right}) = 0.5$ and $\gamma = 0.5$ for any state s , then what are the state values under the policy?
- ④ Iteration t :

$$v_t^{\pi}(s) = \sum_a P(\pi(s) = a)(r(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) v_{t-1}^{\pi}(s'))$$

- 核心问题:
 - 预测问题 (prediction)
 - 给定一个 policy, 计算价值函数
 - 决策问题 (control)
 - 找到状态对应的最优价值函数和最优决策
- 可视化网站: https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Tabular Methods (表格型方法)

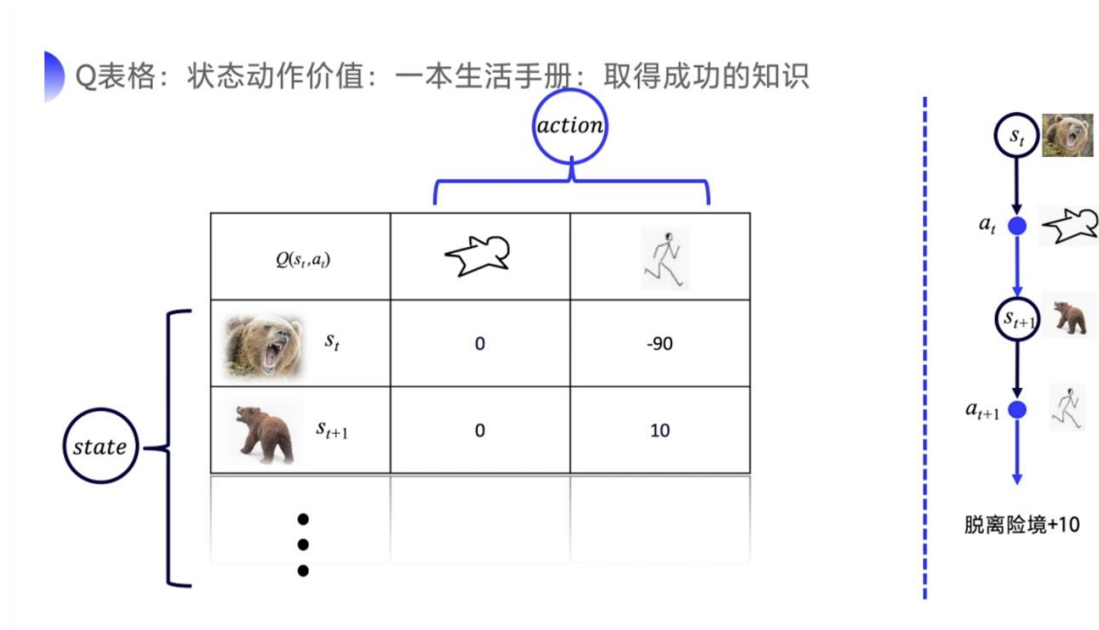
Model-based

- 之前学习的方法都是已知 reward, 状态转移函数的, 也就是能描述环境的, 存在模型的方法
- 存在模型的方法用之前的动态规划类的方法就可以解决

Model-free Prediction

- 状态转移函数和奖励函数未知
- agent 和环境交互, 采集到很多 trajectories, 通过 trajectories 改进策略

- 最终想要得到的是这样一张表格，由该表格来指引决策



- 强化就是我们可以用下一个状态的价值来更新当前状态的价值，其实就是强化学习里面 **bootstrapping** 的概念。在强化学习里面，你可以每走一步更新一下 Q 表格，然后用下一个状态的 Q 值来更新这个状态的 Q 值，这种单步更新的方法叫做时序差分。
- 方法：
 - Monte-Carlo Policy Evaluation
 - 生成很多轨迹，求轨迹 rewards 平均
 - 最终得到的是 **empirical mean return** 而不是 **expected reward**
 - 只能用在**有终止**的 MDP 上
 - 一些数学方法可以把 Monte-Carlo 转成增量型的迭代计算，但是上述三条性质不会改变
 - Temporal Difference Learning
 - 理解：肉给狗-->狗分泌唾液(无条件刺激) 铃声-->肉给狗--->狗分泌唾液(中性刺激) 铃声--->狗分泌唾液(产生无条件刺激)
 - **这种中性刺激跟无条件刺激在时间上面的结合，我们就称之为强化。**强化的次数越多，条件反射就会越巩固。
 - [TD Learning 可视化网站](#)
 - 上述 Demo 中小球对周围状态进行探索，最开始会发现一些能获得 reward 的格子，后来这些格子附近的格子的 value 也会被影响

Temporal-Difference (TD) Learning

- ① Objective: learn v_π online from experience under policy π
- ② Simplest TD algorithm: TD(0)
 - ① Update $v(S_t)$ toward estimated return $R_{t+1} + \gamma v(S_{t+1})$

$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_{t+1}) - v(S_t))$$

- ③ $R_{t+1} + \gamma v(S_{t+1})$ is called TD target
- ④ $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called the TD error
- ⑤ Comparison: Incremental Monte-Carlo
 - ① Update $v(S_t)$ toward actual return G_t given an episode i

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$

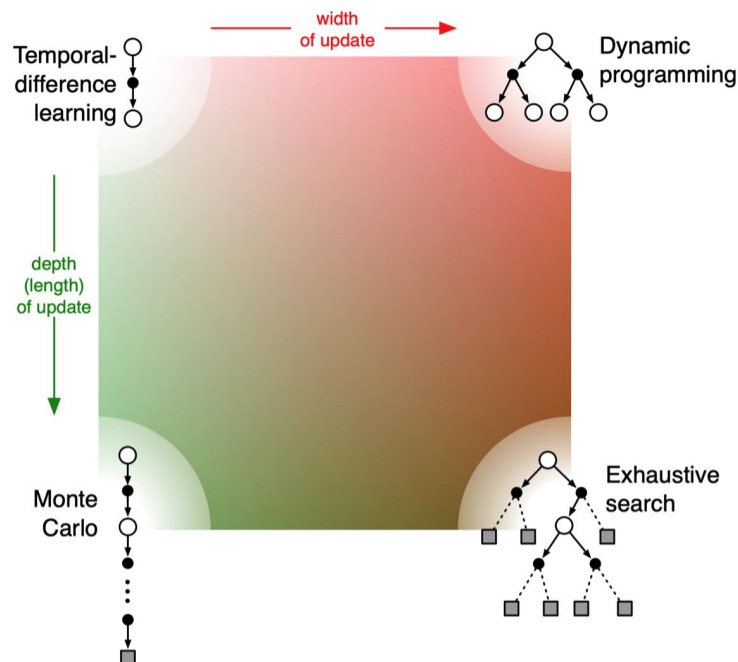
•
•

- 上述图片中 TD target 是对未来折扣 reward 的估计值
- 为什么是估计值？因为 TD target 是对期望折扣 reward 的采样，而且计算采用的是当前估计值 v ，非真实值

- TD vs MC:

- TD 走一步就可以更新估计值，不用等轨迹结束，更适合在线学习
- TD 可以从不完整序列上学习，MC 只能从完整序列上学习
- TD 可以在没终止状态的情况下学习，MC 只适用于有终止的情况
- 举例：
 - TD 会在路口 A 就开始更新预计到达路口 B、路口 C \cdots \cdots，以及到达公司的时间；
 - 而 MC 并不会立即更新时间，而是在到达公司后，再修改到达每个路口和公司的时间。

Unified View of Reinforcement Learning



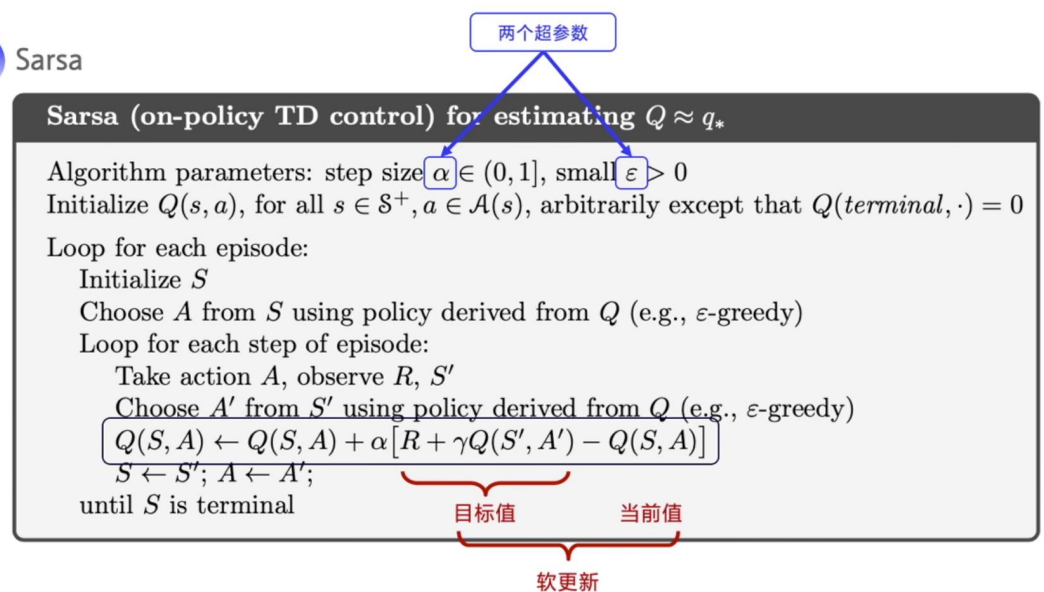
-
-

Model-free control

- 前一章说的 policy iteration 适用于解决已知状态转移函数和 Reward 的 MDP
- 在不知道的情况下，只需要将 policy iteration 的第一步 policy evaluation 用相应的 model-free prediction 方法计算即可
- 贪心系数:
 - ϵ 为 10% 代表 agent 有 90% 的概率根据 q 函数选下一个 action，有 10% 的概率执行随机的动作
 - ϵ 越大，agent 的行为越偏向探索； ϵ 越小，agent 的行为越偏向利用现有知识
 - 往往在 agent 学习到后期时需要减小 ϵ ，减小 random 行为
- Sarsa(On-policy TD control):
 - Sarsa 做出的改变就是把原来 TD 算法里的计算 V 变成了计算 Q
 - 根据 Q -Table 下一步状态下一步 action 和 Reward 去更新当前状态当前 action 的 Q
 - 该算法由于每次更新值函数需要知道当前的状态(state)、当前的动作(action)、奖励(reward)、下一步的状态(state)、下一步的动作(action)，即 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ 这几个值，由此得名 Sarsa 算法。它走了一步之后，拿到了 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ 之后，就可以做一次更新。

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Sarsa



-
-
- 在上述 sarsa 算法的基础上，计算 t+1 到 t+n 目标值的总和就是 n-step sarsa 算法，在 n-step 目标值前乘以衰减因子 lambda，就是 lamda sarsa 算法
- Q-learning (off-policy TD control)
 - sarsa 是一种 on-policy 策略，优化的是实际执行的策略，直接拿执行下一步的 action 去优化 Q-table；即用来选取 action 的策略和用来优化的策略是同一种策略
 - off-policy 在学习过程中，有两种策略：
 - target policy，一般用 pi 表示，是我们要去学习的策略，它根据自己的经验学习最优的策略，而不与环境做交互
 - behavior policy，一般用 mu 来表示，mu 可以去探索所有可能的轨迹，并采集轨迹采集数据，把数据喂给 target policy 去学习
 - off-policy 的好处
 - 可以利用 behavior policy 来学到一个最佳的策略，学习效率高；
 - 可以学习其他 agent 的行为，模仿学习，学习其他 agent 产生的轨迹；
 - 重用老的策略产生的轨迹，可以节省资源。

Q-learning vs 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 $Q(\text{terminal}, \cdot) = 0$
Loop for each episode:
 Initialize S ③
 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)
 $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ ① A' 为下一个state的实际action
 $S \leftarrow S'; A \leftarrow A'$; ②
 until S is terminal

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 $Q(\text{terminal}, \cdot) = 0$
Loop for each episode:
 Initialize S ③
 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'
 $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
 $S \leftarrow S'$ ②
 until S is terminal ①默认A为最优策略选的动作

- Sarsa 在更新 Q 表格的时候，它用到的 A' 。我要获取下一个 Q 值的时候， A' 是下一个 step 一定会执行的 action。这个 action 有可能是 ε -greedy 方法采样出来的值，也有可能是 max Q 对应的 action，也有可能是随机动作，但这是它实际执行的那个动作。
 - 但是 Q-learning 在更新 Q 表格的时候，它用到这个的 Q 值 $Q(S', a)$ 对应的那个 action，它不一定是下一个 step 会执行的实际的 action，因为你下一个实际会执行的那个 action 可能会探索
- Sarsa 是 Q-learning 的改进，Q-learning 因为没有考虑实际的行为而是直接用最大值计算，它的表现会比 Sarsa 大胆
- Q-learning 悬崖寻路算法示例：


```
print("Episode:{}/{}: reward:{:.1f}".format(i_ep+1, cfg.train_eps,ep_reward))
```