

强化学习基础篇（二十七） Model-free控制

终于推进到控制部分了，控制的问题才是核心。

1、预测与控制

预测与控制的区别在于：

- 预测问题中是输入一个MDP (S, A, P, R, γ) 以及一个策略 π ，然后输出基于当前策略 π 的价值函数 V_π 。
- 控制问题是MDP (S, A, P, R, γ) ，然后输出最优价值函数 V_* 以及最优策略 π_* 。

之前的内容主要讲了MC，TD， $TD(\lambda)$ 算法，这三个算法都是为了在给定策略下去估计价值函数 $V(s)$ 。其区别在于MC需要得到一个完整的episode才能进行一次价值函数的更新，而TD方法则可以没走一步就更新一次价值函数。但是我们的目标是要得到最优的策略，所以我们需要通过控制问题，在有MDP的情况下，通过价值函数去改进策略。不断得进行迭代改进，最终收敛到最优策略和最优价值函数。

控制的例子很多，比如控制一个机器人行走，让智能体学会玩围棋，开发一个交易管理的智能体等等。这些实际的问题在显示中可能有着两类问题：

- 一是，MDP模型未知，但是我们可以从现实中很容易进行采样收集数据。
- 二是，MDP模型已知，但是问题的规模太大了，完全没办法进行高效的计算，所以必须使用采样的方法。

Model-free控制就是专注于解决这类问题。

2、同轨策略与异轨策略（On and Off Policy）

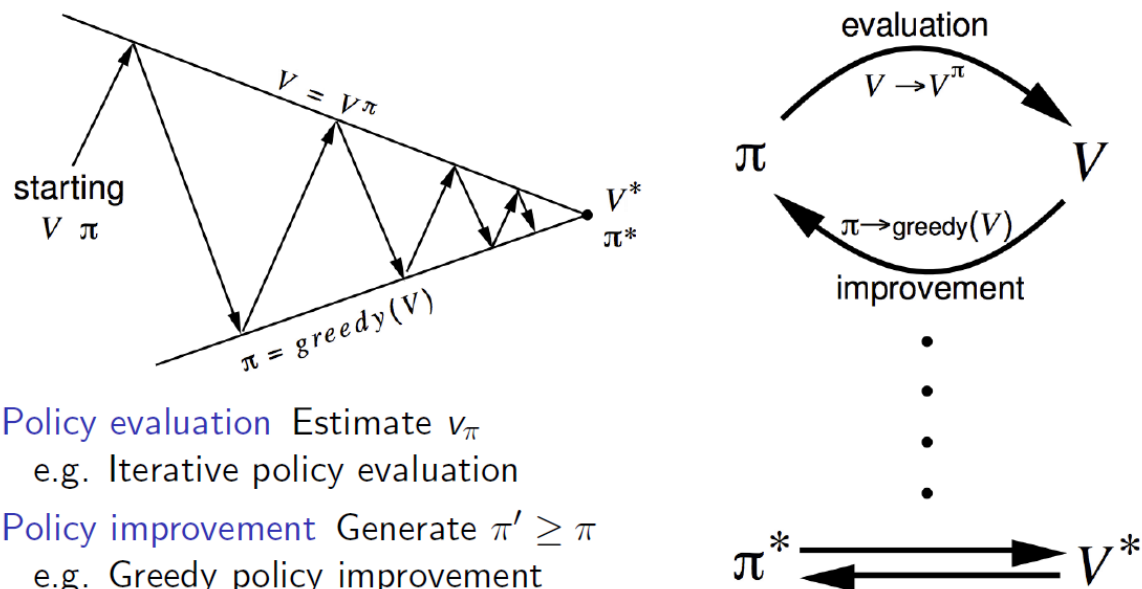
同轨策略学习（On policy learning）就是智能体已经有了一个策略 π ，并且基于该策略 π 进行采样，以得到的经验轨迹集合来更新价值函数。虽有采用策略评估和策略改进对给定策略进行优化，以获得最优策略。由于需要优化的策略 π 基于当前给定的策略 π ，所以称之为On Policy。

异轨策略学习（Off policy learning）是智能体虽然有一个策略 π ，但是并不基于该策略 π 进行采样，而是基于另一个策略 μ 进行采样。另一个策略 μ 可以是人类专家制定的策略等一些较为成熟的策略方法。由于优化的策略不完全基于当前策略，所以称为Off Policy。

3、同轨蒙特卡洛控制（On-policy Monte Carlo Control）

广义策略迭代（GPI）

回顾一下之前提到的广义策略迭代（Generalized Policy Iteration, GPI）模型是指让策略评估和策略改善交互的一般概念，它不依赖于两个过程的粒度（granularity）和其他细节。几乎所有强化学习方法都可以被很好地描述为GPI。



如果评估过程和改善过程都稳定下来，即不再发生变化，那么价值函数和策略必须都是最优的，如上图（右）所示。

人们还可以用两个目标来考虑GPI中评估和改善过程的相互作用，如上图（左）所示，上面的线代表目标价值函数 $V = V^\pi$ ，下面的线代表目标 $\pi = \text{greedy}(v)$ 。目标会发生相互作用，因为两条线不是平行的。从一个策略 π 和一个价值函数 v 开始，每一次箭头向上代表着利用当前策略进行价值函数的更新，每一次箭头向下代表着根据更新的价值函数贪婪地选择新的策略，说它是贪婪的，是因为每次都采取转移到可能的、状态函数最高的新状态的行为。最终将收敛至最优策略和最优价值函数。

基于MC的GPI

在之前的GPI方法中，策略评估用到的是贝尔曼方程，策略改进使用的是贝尔曼最优方程：

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1} \mid S_t = s, A_t = a)] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')] \end{aligned}$$

但是使用动态规划算法来改善策略是需要知道某一状态的所有后续状态及状态间转移概率，即：

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} (\mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s'))$$

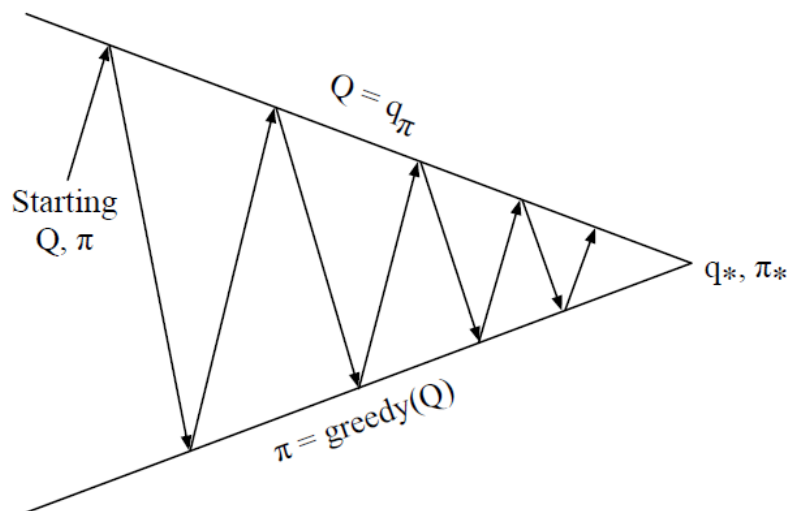
但是如果后续的转移概率未知，则在策略估计中就不能使用贝尔曼期望方程，而是变为sample方法，比如MC方法或TD方法。

在模型未知的时候，首先应该用状态行为对的价值 $Q(s, a)$ 来代替状态价值 $V(s)$ ：

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

这样做的目的是可以改善策略而不用知道整个模型，只需要知道在某个状态下采取什么样的行为价值最大即可。

所以基于 $Q(s, a)$ 的GPI可以为如下形式。



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

即使这样，至少还存在一个问题，即当我们每次都使用贪婪算法来改善策略的时候，将很有可能由于没有足够的采样经验而导致产生一个并不是最优的策略，我们需要不时的尝试一些新的行为，这就是探索（Exploration）。

探索的例子



"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

如上图，在你面前有两扇门，考虑如下的行为、奖励并使用贪婪算法改善策略：

- 你打开左侧门得到奖励为0: $V(left) = 0$

- 你打开右侧门得到奖励为1: $V(right) = +1$
- 使用greedy策略, 会继续去打开右侧门, 而不会打开左侧门, 假设得到奖励为+3:
 $V(right) = \frac{1+3}{2} = +2$
- 继续greedy策略, 打开右侧门, 假设得到奖励+2: $V(right) = \frac{1+3+2}{3} = +2$
- 如此一直循环下去, 会一直打开右侧门。

这种情况下, 打开右侧门是否就一定是最好的选择呢?

答案显而易见是否定的。因此完全使用贪婪算法改善策略通常不能得到最优策略。为了解决这一问题, 我们需要引入一个随机机制, 以一定的概率选择当前最好的策略, 同时给其它可能的行为一定的几率, 这就是 $\epsilon - greedy$ 探索。

$\epsilon - greedy$ 探索策略

$\epsilon - greedy$ 策略是一个最简单的探索策略, 其假设所有 m 个动作都有着非0的概率被执行, 在策略选择中以 $1 - \epsilon$ 的概率去选择贪婪动作, 并以 ϵ 的概率选择随机动作。其数学表达式为:

$$\pi(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

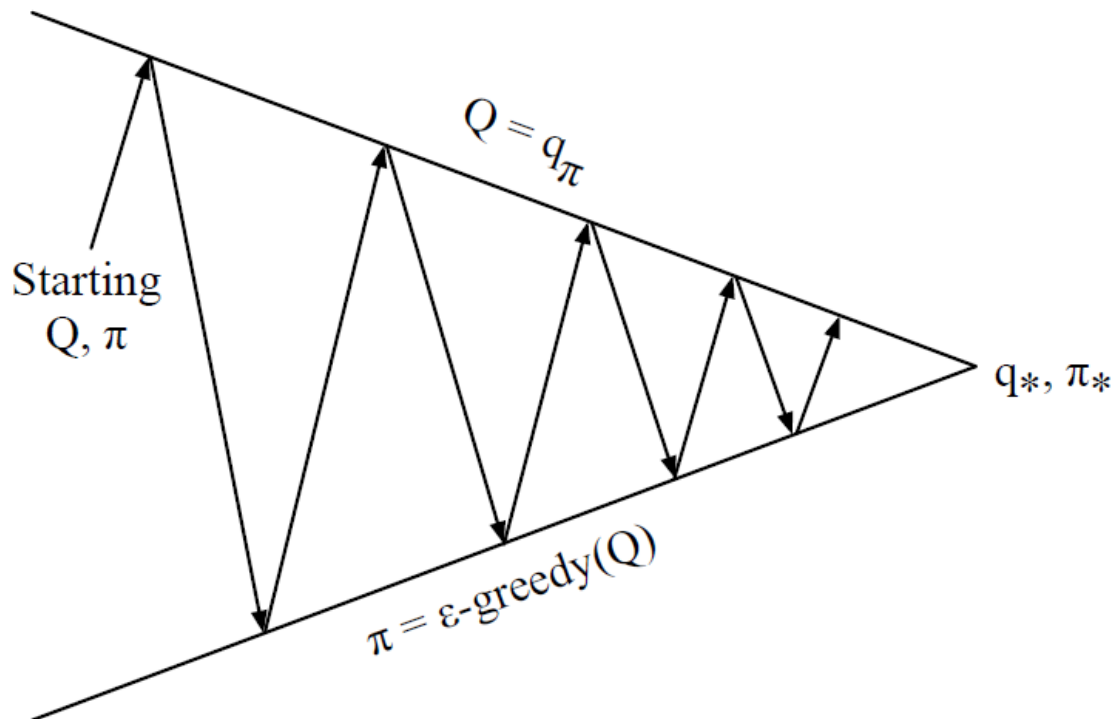
如果我们在GPI的策略改进部分使用 $\epsilon - greedy$ 探索策略, 那么我们会理论证明保障改进的策略可以单调递增的。

即对于任意的 $\epsilon - greedy$ 策略 π , 使用相应的 q_π 得到的 $\epsilon - greedy$ 策略 π' 是在 π 上的一次策略提升, 即 $v_{\pi'}(s) \geq v_\pi(s)$ 。

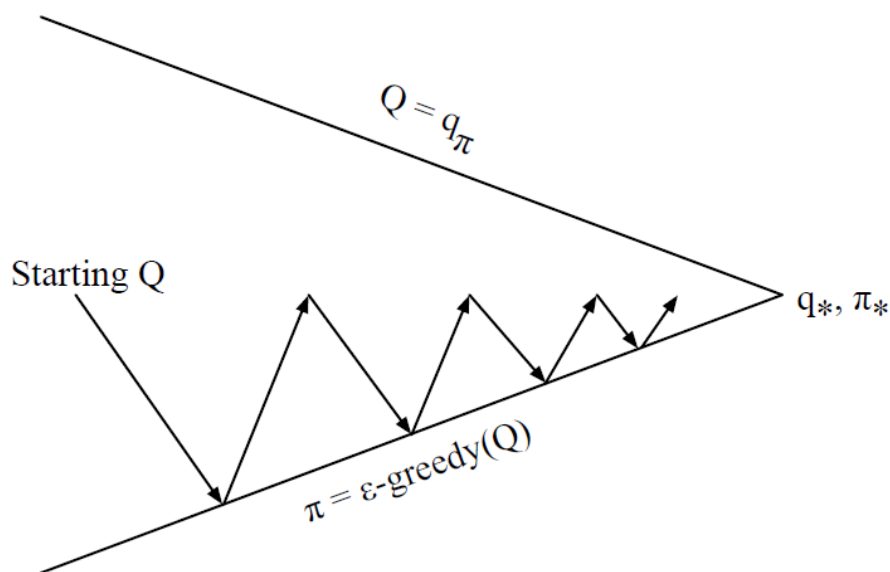
证明过程如下:

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a | s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a | s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a | s) q_\pi(s, a) = v_\pi(s) \end{aligned}$$

所以完整的MC策略迭代过程在引入了 $\epsilon - greedy$ 之后如下所示:



和之前讲到的策略迭代方法不一样，MC策略迭代在估计中用的是Q函数，在策略改进中用的是 $\epsilon - greedy$ 方法，在实际应用中，我们称之为蒙特卡洛控制，且更确切地给出其迭代示意图：



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

MC控制使用Q函数进行策略评估，使用 $\epsilon - greedy$ 探索改善策略。该方法最终可以收敛至最优策略。

图中每一个向上或向下的箭头都对应着多个episode。也就是说我们一般在经历了多个episode之后才进行依据Q函数更新或策略改善。

实际上我们也可以在每经历一个episode之后就更新Q函数或改善策略。但不管使用那种方式，在使用 $\epsilon - greedy$ 探索下我们始终只能得到基于某一策略下的近似Q函数，且该算法没有一个终止条件，因为它一直在进行探索。

GLIE

我们希望得到一个这样的学习方法：

- 1、在学习开始时有足够的探索：

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

- 2、最终得到的策略没有探索，是一个确定性的策略。

$$\lim_{k \rightarrow \infty} \pi_k(a | s) = \mathbf{1} \left(a = \operatorname{argmax}_{a' \in \mathcal{A}} Q_k(s, a') \right)$$

为此引入了另一个理论概念：GLIE(Greedy in the Limit with infinite Exploration), 如果 $\epsilon - greedy$ 策略能够使得 ϵ 在 $\epsilon_k = \frac{1}{k}$ 时候降低到0, 那么 $\epsilon - greedy$ 策略也是一个GLIE策略。其算法流程如下所示：

- 1. 首先从环境中使用策略 π 采样 k 个 episode: $S_1, A_1, R_2, \dots, S_T \sim \pi$
- 2. 对于在 episode 中的每个状态 S_t 以及动作 A_t , 进行如下增量更新：

$$\begin{aligned} N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t)) \end{aligned}$$

- 3. 基于新得到的Q函数更新策略：

$$\begin{aligned} \epsilon &\leftarrow 1/k \\ \pi &\leftarrow \epsilon - greedy(Q) \end{aligned}$$

在理论上GLIE MC控制的方法是可以收敛到最优动作值函数, $Q(s, a) \rightarrow q_*(s, a)$ 。

4、同轨时序差分 (On-policy TD) 学习

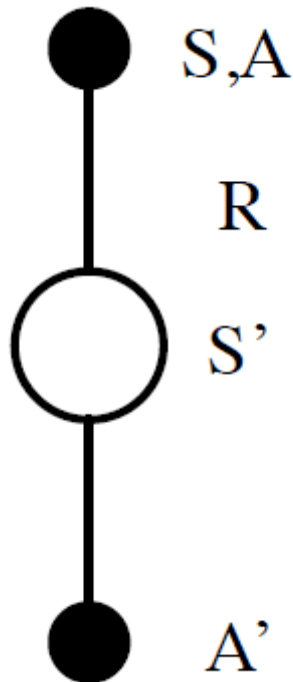
MC和TD控制的差别

时序差分 (Temporal-difference, TD) 学习方法相比于MC的方法有着几个优势：

- 低方差 (low variance)
- 可以在线实学习
- 可以学习不完整的episode

因此可以很自然的想到在控制的迭代中去使用TD方法代替MC方法。也就是下面提到的Sarsa算法。

Sarsa算法



Sarsa算法的名字就是来源于上图这个过程，在智能体处在某个状态 S 的时候按 $\epsilon - greedy$ 执行动作 A ，会得到一个即时奖励 R ，并在与环境交互中转移到下一个状态 S' ，再一次基于 $\epsilon - greedy$ 策略选择动作 A' 。

这个时候智能体不会去执行 A' ，而是通过自身当前的状态行为价值函数得到该 (S', A') 状态行为对的价值 $Q(S', A')$ ，同时结合 (S, A) 获得的奖励 R 来更新 $Q(S, A)$ 。所以通过公式很容易可以看出SARSA的更新规则：

$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

同轨策略的Sarsa算法描述如下：

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A';$ 
  until  $S$  is terminal
  
```

算法 Sarsa 算法

```
1: 初始化  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , 且  $Q(\text{终止状态}, \cdot) = 0$ 
2: repeat(对于每个片段)
3:   初始化状态  $S$ 
4:   根据  $Q$  选择一个在  $S$  处的动作  $A$ , (e.g. 使用  $\epsilon$ -贪婪策略)
5:   repeat(对于片段中的每一步)
6:     执行动作  $A$ , 观测  $R, S'$ 
7:     根据  $Q$  选择一个在  $S'$  处的动作  $A'$ , (e.g. 使用  $\epsilon$ -贪婪策略)
8:      $Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$ 
9:      $S \leftarrow S'; A \leftarrow A'$ 
10:  until  $S$  是终止状态
11: until 收敛
```

Sarsa的收敛性

Sarsa的收敛性是有定理支持的, 在满足如下两个条件时, Sarsa算法将收敛至最优行为价值函数。

- 条件1: 任何时候的策略 $\pi(a|s)$ 符合GLIE特性;
- 条件二: 步长系数 α_t 满足: $\sum_{t=1}^{\infty} \alpha_t = \infty$ 以及 $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

n-Step Sarsa

考虑从 $n = 1, 2, \dots, \infty$ 得到的n-step回报,

$$\begin{aligned} n = 1 \quad (\text{Sarsa}) \quad q_t^{(1)} &= R_{t+1} + \gamma Q(S_{t+1}) \\ n = 2 \quad q_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\ &\vdots \\ n = \infty \quad (MC) \quad q_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T \end{aligned}$$

我们可以得到n-step的Q-return:

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

所以接下来可以根据n-step的Q-return去更新Q函数:

$$Q(S, A) \leftarrow Q(S, A) + \alpha (q_t^{(n)} - Q(S, A))$$

前向观点的Sarsa(λ)

Sarsa(λ)

$1-\lambda$

$(1-\lambda) \lambda$

$(1-\lambda) \lambda^2$

\dots

s_t

s_T

λ^{T-t-1}

$\Sigma = 1$

Sarsa(λ)算法

除了状态价值函数 $Q(s, a)$ 的更新方式、超参数 λ ，以及资格迹 $E(s, a)$ 以为，Sarsa(λ)算法的思想和Sarsa是类似的，这里总结下算法流程：

```
Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
Repeat (for each episode):
     $E(s, a) = 0$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
    Initialize  $S, A$ 
    Repeat (for each step of episode):
        Take action  $A$ , observe  $R, S'$ 
        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
         $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$ 
         $E(S, A) \leftarrow E(S, A) + 1$ 
        For all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ :
             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$ 
             $E(s, a) \leftarrow \gamma \lambda E(s, a)$ 
         $S \leftarrow S'; A \leftarrow A'$ 
    until  $S$  is terminal
```

算法 Sarsa(λ) 算法

```
1: 对于所有的  $s \in \mathcal{S}, a \in \mathcal{A}(s)$  初始化  $Q(s, a)$ 
2: repeat (对于每一个片段)
3:     初始化  $E(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
4:     选择初始状态和初始动作  $S, A$ 
5:     repeat 对于片段中的每一步
6:         执行动作  $A$ , 观察到  $R, S'$ 
7:         根据  $Q$  选择一个在  $S'$  处的动作  $A'$  (e.g. 使用  $\epsilon$ -贪婪策略)
8:          $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$ 
9:          $E(S, A) \leftarrow E(S, A) + 1$ 
10:        for 对于所有的  $s \in \mathcal{S}, a \in \mathcal{A}(s)$  do
11:             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$ 
12:             $E(s, a) \leftarrow \gamma \lambda E(s, a)$ 
13:        end for
14:         $S \leftarrow S', A \leftarrow A'$ 
15:    until 直到终止状态
16: until 收敛
```

5、异轨策略学习 (Off-policy Learning)

异轨策略学习的目标是通过计算 $v_\pi(s)$ 或者 $q_\pi(s, a)$ 去评估一个目标策略 $\pi(a|s)$ ，但是会遵循另外一个行为策略 $\mu(s, a)$ 来进行。其采样可以是： $\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$

异轨策略这种方式之所以有效，主要有几个考虑的因素：

- 智能体可以不从自身的行为学习，而是从观测人类专家的行为或者观察其他智能体的行为中进行学习。
- 我们可以服用在运行中那些以前产生的旧策略进行学习，比如在更新过程中产生的各种策略 $\pi_1, \pi_2 \dots \pi_{t-1}$ 。
- 智能体可以在遵循一些探索策略的时候去学习最优策略。
- 智能体可以在只遵循一个策略的时候去学习多种策略。

重要性采样

这里要先介绍下Off-policy中重要的概念重要性采样 (Importance Sampling)，重要性采样就是我们要计算函数 $f(X)$ 在分布 P 下的期望时候，不好计算。那么我们可以转换下思路，去转化到计算函数在一个比较容易计算的分布下 Q 下的期望：

$$\begin{aligned}\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X) \frac{P(X)}{Q(X)} f(X) \\ &= \mathbb{E}_{X \sim Q} \left[\frac{P(X)}{Q(X)} f(X) \right]\end{aligned}$$

考虑 t 时刻之后的动作状态轨迹 $\rho_t = A_t, S_{t+1}, A_{t+1}, \dots, S_T$ ，可以得到该轨迹出现的概率为：

$$\mathbb{P}(\rho_t) = \prod_{k=t}^{T-1} \pi(A_k | S_k) \mathbb{P}(S_{k+1} | S_k, A_k)$$

因此可以得到相应的重要性权重为：

$$\eta_t^T = \frac{\prod_{k=t}^{T-1} \pi(A_k | S_k) \mathbb{P}(S_{k+1} | S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k | S_k) \mathbb{P}(S_{k+1} | S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k | S_k)}{\mu(A_k | S_k)}$$

即便是未知环境模型，也能得到重要性权重。

IS下的异轨MC

对于off-policy Monte-Carlo使用importance sampling：

- a、使用从MC的行为策略 μ 中得到的回报去评估策略 π ，
- b、得到的加权回报为：

$$G_t^{\pi/\mu} = \frac{\pi(A_t | S_t)}{\mu(A_t | S_t)} \frac{\pi(A_{t+1} | S_{t+1})}{\mu(A_{t+1} | S_{t+1})} \dots \frac{\pi(A_T | S_T)}{\mu(A_T | S_T)} G_t$$

- c、根据加权回报更新值函数：

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t^{\pi/\mu} - V(S_t))$$

这里要注意的是如果行为策略 μ 的概率为0，但是目标策略 π 的概率非0，就不能用了。

同时我们知道，MC方法的方差本来就很大，而重要性采样将会使得方差急剧增大，因此结合重要性采样的MC方法更不适用。

IS下的异轨TD

对于off-policy TD使用importance sampling，使用从TD方法在遵循行为策略 μ 的同时去评估策略 π ，其更新方式为：

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t | S_t)}{\mu(A_t | S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

采用TD的方式比MC的方式大大降低了方差。

Q – Learning算法

Q – Learning算法是不需要使用重要性采样的，其过程是这样的：

- 在 t 时刻与环境进行实际交互的行为 A_t 由一个 ϵ – greedy策略 μ 生成： $A_t \sim \mu(\cdot | S_t)$
- 在 $t + 1$ 时刻用来更新 Q 值的的行为 A'_{t+1} ，通过一个完全greedy的策略 π 产生： $A'_{t+1} \sim \pi(\cdot | S_{t+1})$

其动作状态值函数的更新方式为：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

其中的TD target是： $R_{t+1} + \gamma Q(S_{t+1}, A')$ ， $A' \sim \pi(\cdot | S_t)$

这里和之前的TD target还是有区别的： $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ ， $A_{t+1} \sim \mu(\cdot | S_t)$

Q – Learning进行异轨控制

我们已经知道 Q – Learning中目标策略 π 是一个关于 $Q(s, a)$ 的贪婪策略：

$$\pi(S_{t+1}) = \operatorname{argmax}_{a'} Q(S_{t+1}, a')$$

行为策略 μ 是一个关于 $Q(s, a)$ 的 ϵ – greedy策略，所以 Q – Learning的目标可以简化为：

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q\left(S_{t+1}, \operatorname{argmax}_{a'} Q(S_{t+1}, a')\right) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

其 Q 函数的更新方式为：

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

算法描述为：

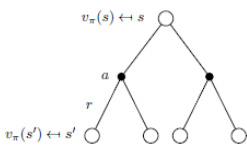

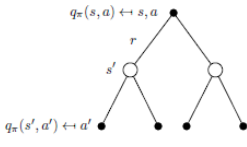
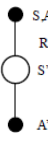
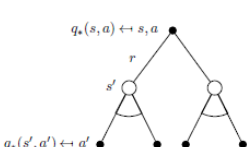
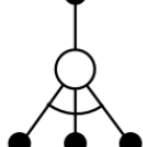
```
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ ;
  until  $S$  is terminal
```

算法 Q 学习算法

- 1: 初始化 $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 且 $Q(\text{终止状态}, \cdot) = 0$
- 2: **repeat**(对于每个片段)
- 3: 初始化状态 S
- 4: **repeat**(对于片段中的每一步)
- 5: 根据 Q 选择一个在 S 处的动作 A , (e.g. 使用 ε -贪婪策略)
- 6: 执行动作 A , 观测 R, S'
- 7: $Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$
- 8: $S \leftarrow S'$
- 9: **until** S 是终止状态
- 10: **until** 收敛

6、总结

下面两张图概括了各种DP算法和各种TD算法，同时也揭示了各种不同算法之间的区别和联系。总的来说TD是采样+有数据引导 (bootstrap)，DP是全宽度+实际数据。如果从Bellman期望方程角度看：聚焦于状态本身价值的是迭代法策略评估 (DP) 和TD学习，聚焦于状态行为对价值函数的则是Q-策略迭代 (DP) 和SARSA；如果从针对状态行为价值函数的Bellman优化方程角度看，则是Q-价值迭代 (DP) 和Q学习。

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $v_{\pi}(s)$	 <p>Iterative Policy Evaluation</p>	 <p>TD Learning</p>
Bellman Expectation Equation for $q_{\pi}(s, a)$	 <p>Q-Policy Iteration</p>	 <p>Sarsa</p>
Bellman Optimality Equation for $q_{*}(s, a)$	 <p>Q-Value Iteration</p>	 <p>Q-Learning</p>

<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	$V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$	$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$
Q-Value Iteration	Q-Learning
$Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$	$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$