1. 背景

问题 1：给定随机变量 X，并有对 X 的独立同分布的采样序列$\{x\}$，求解：

$$w = E(X)$$

- 构造函数$g(w) = w - E(X)$，则问题可以转换为：

$$g(w) = 0$$

- 带噪声的输出可以定义为：

$$\widetilde{g}(w,\eta) = w - x = \big(w - E(X)\big) + \big(E(X) - x\big) = g(w) + \eta$$

- 使用 RM 算法求解问题，求解步骤描述为：

$$w_{k+1} = w_k - \alpha_k \widetilde{g}(w,\eta) = w_k - \alpha_k(w_k - x_k)$$

问题 2：给定随机变量 X，并有对 X 的独立同分布的采样序列$\{x\}$，求解：

$$w = E\big(v(X)\big)$$

- 构造函数$g(w) = w - E\big(v(X)\big)$，则问题可以转换为：

$$g(w) = 0$$

- 带噪声的输出可以定义为：

$$\widetilde{g}(w,\eta) = w - v(x) = \Big(w - E\big(v(X)\big)\Big) + \Big(E\big(v(X)\big) - v(x)\Big) = g(w) + \eta$$

- 使用 RM 算法求解问题，求解步骤描述为：

$$w_{k+1} = w_k - \alpha_k \widetilde{g}(w,\eta) = w_k - \alpha_k\big(w_k - v(x_k)\big)$$

问题 3：给定随机变量 X，并有对 X 的独立同分布的采样序列$\{x\}$，求解：

$$w = E\big(R + \gamma v(X)\big)$$

- 构造函数$g(w) = w - E\big(R + \gamma v(X)\big)$，则问题可以转换为：

$$g(w) = 0$$

- 带噪声的输出可以定义为：

$$
\begin{aligned}
\widetilde{g}(w,\eta) &= w-[R+\gamma v(x)] \\
&= \Big(w-E\big(R+\gamma v(X)\big)\Big)+\Big(E\big(R+\gamma v(X)\big)-[R+\gamma v(x)]\Big) \\
&= g(w)+\eta
\end{aligned}
$$

- 使用 RM 算法求解问题，求解步骤描述为：

$$w_{k+1}=w_k-\alpha_k\widetilde{g}(w,\eta)=w_k-\alpha_k(w_k-[R+\gamma v(x_k)])$$

2. Temporal-Difference(TD) algorithm

TD 算法的目的是根据当前已有策略，估计出 state value，即完成 policy evaluation

TD 算法所需要的数据/经验：由给定的策略π产生的

$$(s_0,r_1,s_1,\ldots,s_t,r_{t+1},s_{t+1},\ldots)\ or\ \{(s_t,r_{t+1},s_{t+1})\}_t$$

用三元组来进行表示，其中s_t表示 t 时刻访问到的 state， r_{t+1}表示从s_t跳到s_{t+1}得到的 reward，则 TD 算法表示为：

$$
\begin{cases}
v_{t+1}(s_t)=v_t(s_t)-\alpha_t(s_t)\big[v_t(s_t)-[r_{t+1}+\gamma v_t(s_{t+1})]\big] \\
v_{t+1}(s)=v_t(s),\forall s\neq s_t
\end{cases}
$$

$v_t(s_t)$表示在 t 时刻访问到的 state 在 t 时刻的 state value 的估计值，$v_t(s)$表示集合空间中某个 state 在 t 时刻的 state value 的估计值，$\alpha_t(s_t)$表示在 t 时刻状态s_t的学习率；下面一个式子的意思是：在 t 时刻只访问了状态s_t，其他状态未访问，故其他状态的 state value 的估计值不更新

第一个式子的含义如下所示：

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}}=\underbrace{v_t(s_t)}_{\text{current estimate}}-\alpha_t(s_t)\Big[\overbrace{v_t(s_t)-\big[\underbrace{r_{t+1}+\gamma v_t(s_{t+1})}_{\text{TD target }\bar{v}_t}\big]}^{\text{TD error }\delta_t}\Big]$$

- $v_{t+1}(s_t)$表示对s_t的 state value 的新的估计值，$v_t(s_t)$表示s_t的 state value 的当前的估计值
- $\overline{v}_t=r_{t+1}+\gamma v_t(s_{t+1})$被称为 TD target
- $\delta_t=v_t(s_t)-[r_{t+1}+\gamma v_t(s_{t+1})]$被称为 TD error

为什么存在 TD target，它的作用是什么？

TD 算法的目的就是让$v(s_t)$朝着$\overline{v}_t$进行优化，推导过程：

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)[v_t(s_t) - \overline{v}_t]$$

两边同时减去一个$\overline{v}_t$，得到：

$$v_{t+1}(s_t) - \overline{v}_t = v_t(s_t) - \overline{v}_t - \alpha_t(s_t)[v_t(s_t) - \overline{v}_t]$$
$$v_{t+1}(s_t) - \overline{v}_t = [1 - \alpha_t(s_t)][v_t(s_t) - \overline{v}_t]$$

两边取绝对值得到：

$$|v_{t+1}(s_t) - \overline{v}_t| = |1 - \alpha_t(s_t)||v_t(s_t) - \overline{v}_t|$$

又由于 $0 < 1 - \alpha_t(s_t) < 1$，得到：

$$|v_{t+1}(s_t) - \overline{v}_t| \leq |v_t(s_t) - \overline{v}_t|$$

所以$v_{t+1}(s_t)$比$v_t(s_t)$更接近$\overline{v}_t$，意味着该算法是将 state value 估计值朝着$\overline{v}_t$进行优化的

那么 TD error 又如何解释？

TD error 用来描述两个相邻的时间步之间估计值的差异，它也用来描述我们要求的值v_π与当前估计值v_t之间的差异，当$\delta_t = 0$ 时，v_t应该是等于v_π的，证明如下：

$$\delta_t = v(s_t) - [r_{t+1} + \gamma v(s_{t+1})]$$

若在正确的 state value v_π下，它应该有

$$\delta_{\pi,t} = v_\pi(s_t) - [r_{t+1} + \gamma v_\pi(s_{t+1})]$$

则有：

$$E[\delta_{\pi,t}|S_t = s_t] = v_\pi(s_t) - E[R_{t+1} + \gamma v_\pi(s_{t+1})|S_t = s_t] = 0$$

所以当$v_t = v_\pi$时，应当有$\delta_t = 0$；反之，若$\delta_t \neq 0$，则$v_t \neq v_\pi$

所以 TD error 可以看作一种 innovation，它可以用来表示从(s_t, r_{t+1}, s_{t+1})得到的信息

TD 算法只能用来估计给定 policy 的 state value，无法估计 action value 以及搜索最佳策略。TD 算法是在没有模型的情况下来求解贝尔曼公式（给定策略π）

某个状态的 state value 定义为（详情可见第二章）：

$$v_\pi(s) = E[R + \gamma G|S = s], s \in S$$

又有：

$$E[G|S = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) v_\pi(s') = E[v_\pi(s')|S = s]$$

则可以将状态 s 的 state value 定义为：

$$v_\pi(s) = E[R + \gamma v_\pi(s')|S = s], s \in S$$

令$g\big(v(s)\big) = v(s) - E[R + \gamma v_\pi(s')|s]$，则可以表示为：

$$g\big(v(s)\big) = 0$$

实际上 TD 算法就是求解贝尔曼公式的一个 RM 算法，求能够满足等式的$v(s)$，即$v_\pi(s)$。带噪声的输出为：

$$\begin{aligned}\tilde{g}\big(v(s)\big) &= v(s) - [r + \gamma v_\pi(s')] \\ &= (v(s) - E[R + \gamma v_\pi(s')|s]) + (E[R + \gamma v_\pi(s')|s] - [r + \gamma v_\pi(s')]) \\ &= g\big(v(s)\big) + \eta\end{aligned}$$

则相对应的 RM 算法表示为：

$$v_{k+1}(s) = v_k(s) - \alpha_k \tilde{g}\big(v_k(s)\big) = v_k(s) - \alpha_k(v_k(s) - [r_k + \gamma v_\pi(s'_k)])$$

- 问题一：我们想求解这个问题，必须获得大量的经验，即多个$\{(s, r, s')\}$的集合，难道我们需要多次重复这个动作吗？

我们可以在每次 trajectory 中，如果恰巧访问到了 s，就对 s 更新一下；如果未访问到 s，对 s 的 state value 的估计值就保持不变。这样我们就可以通过多个 trajectory 来对所有状态的 state value 的估计值进行更新

- 问题二：每次要更新，我们需要$v_\pi(s')$的信息，但这个信息也是未知的，怎么办？

采用该状态的估计值$v_\pi(s'_k)$来进行更新，虽然可能不准确，但聊胜于无

TD 算法的收敛性定理:

当$t \to \infty$时，对任何$s \in S$，有$\sum_t \alpha_t(s) = \infty \ and \ \sum_t {\alpha_t}^2(s) < \infty$，则$v_t(s)$ with probability 1 收敛于$v_\pi(s)$

- 该定理想表示的是：通过 TD 算法一定能求解出给定策略π下的 state value
- $\sum_t \alpha_t(s) = \infty \ and \ \sum_t {\alpha_t}^2(s) < \infty$必须对所有$s \in S$生效。在第 t 个时间步，若$s = s_t$，则表示当前状态 s 被访问，$\alpha_t(s) > 0$；对于其他未被访问的状态，$\alpha_t(s) = 0$。但是要保证每个状态都被访问足够多（无限）次。
- 学习率α经常被设置为一个较小的常量，这样做会使得条件$\sum_t {\alpha_t}^2(s) < \infty$不再满足。但是在实际应用中，如果$\alpha \to 0$，越往后访问得到的经验作用越小，这不是我们希望看到的。

TD/Sarsa 算法与 MC 算法的区别:

TD/Sarsa learning	MC learning
Online: TD learning is online. It can update the state/action values immediately after receiving a reward.	**Offline:** MC learning is offline. It has to wait until an episode has been completely collected.
Continuing tasks: Since TD learning is online, it can handle both episodic and continuing tasks.	**Episodic tasks:** Since MC learning is offline, it can only handle episodic tasks that has terminate states.

- TD 算法是 online 的，因为在每次访问一个 state 后，都可以对该 state 的 state value 估计值进行更新；MC 算法是 offline 的，它只有在完整完成一个 episode 得到 return 值之后，才能对估计值进行更新
- 因此，TD 算法既可以处理 continuing task，也可以处理 episodic task；而 MC 算法只能处理有终端状态的 episodic task。从这两个方面看，TD 算法优于

MC 算法

TD/Sarsa learning	MC learning
Bootstrapping: TD bootstraps because the update of a value relies on the previous estimate of this value. Hence, it requires initial guesses.	**Non-bootstrapping:** MC is not bootstrapping, because it can directly estimate state/action values without any initial guess.
Low estimation variance: TD has lower than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$.	**High estimation variance:** To estimate $q_\pi(s_t, a_t)$, we need samples of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$. Suppose the length of each episode is L. There are $\lvert\mathcal{A}\rvert^L$ possible episodes.

- TD 算法是 bootstrapping 的，因为它每次对估计值的更新依赖于上一个时刻的估计值，因此我们需要提供一个初始值；而 MC 算法它直接利用每个 episode 的 return 来进行更新，不需要提供初始值
- TD 算法中涉及到的随机变量更少，一次更新只涉及到了$v_k(s)$、r_k和$v_\pi(s'_k)$，因此它的方差更小；而 MC 算法中，一次对估计值的更新涉及很多 episode，而一条 episode 又包含很多 state 和 reward，故其方差更大

3. Sarsa 算法

TD 算法的目的是根据当前已有策略，估计出 state value；而 Sarsa 算法的目的是根据当前已有策略，估计出 action value

Saras 算法所需要的数据/经验：由给定的策略π产生的

$$(s_0, a_0, r_1, s_1, a_1, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots)\ or\ \{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$$

用五元组来进行表示，其中s_t表示 t 时刻访问到的 state，a_t表示在s_t采取的 action，r_{t+1}表示从s_t跳到s_{t+1}得到的 reward，则 Sarsa 算法表示为：

$$\begin{cases} q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\big[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\big], \\ q_{t+1}(s, a) = q_t(s, a), \forall (s, a) \neq (s_t, a_t) \end{cases}$$

$q_t(s_t, a_t)$是对$q_\pi(s_t, a_t)$的估计值，$\alpha_t(s_t, a_t)$是在第 t 个时间步(s_t, a_t)的学习率。当此时访问到(s_t, a_t)，即对它的 action value 的估计值进行更新；否则不进行更新

Sarsa 算法的名称是 state-action-reward-state-action 首字母的缩写，它实际上是 TD 算法的求解 action value 的版本

Sarsa 算法实际上是求解另一种贝尔曼公式，该贝尔曼公式使用 action value 来进行表达：

$$q_\pi(s, a) = E[R + \gamma q_\pi(s', a')|s, a], \forall s, a$$

Sarsa 算法的收敛性定理：

当$t \to \infty$时，对任何(s, a)，有$\sum_t \alpha_t(s, a) = \infty \ and \ \sum_t {\alpha_t}^2(s, a) < \infty$，则$q_t(s, a)$ with probability 1 收敛于$q_\pi(s, a)$

- 该定理想表示的是：通过 Sarsa 算法一定能求解出给定策略下的 action value

我们知道强化学习的最终目的是求解最优策略，为了实现这个目标，我们将 Sarsa 算法和 policy improvement 步骤结合起来，结合后的新算法也被称为 Sarsa 算法，它的伪代码如下：

For each episode, do

If the current s_t is not the target state, do

Collect the experience $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$: In particular, take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}, and then take action a_{t+1} following $\pi_t(s_{t+1})$.

Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\Big]$$

Update policy:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}|}(|\mathcal{A}| - 1) \text{ if } a = \arg\max_a q_{t+1}(s_t, a)$$
$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}|} \text{ otherwise}$$

- 如果当前访问的状态s_t不是 target state，则根据当前策略π_t采取动作a_t，得到 reward r_{t+1}，跳到下一个状态s_{t+1}，然后在根据当前策略采取动作a_{t+1}，得到 Sarsa 算法所需要的经验
- 根据获得的经验，对访问到的(s_t, a_t)的 action value 估计值进行更新，然后根据当前估计值对当前访问到的状态s_t的策略进行更新（使用$\varepsilon - greedy$），然后再跳到第一步

这个算法与之前不同，之前的算法都是准确计算或近似估计出 action value 后，再进行策略更新；而在这个算法中，每对估计值进行一次更新就要对 policy 进行一次更新

4. Expected Sarsa 算法

Expected Sarsa 算法表示为：

$$\begin{cases} q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\big[q_t(s_t, a_t) - [r_{t+1} + \gamma E[q_t(s_{t+1}, A)]]\big], \\ q_{t+1}(s, a) = q_t(s, a), \forall (s, a) \neq (s_t, a_t) \end{cases}$$

$$E[q_t(s_{t+1}, A)] = \sum_a \pi_t(a|s_{t+1}) q_t(s_{t+1}, a) = v_t(s_{t+1})$$

它表示在策略π_t下，$q_t(s_{t+1}, A)$的期望值，即为$v_t(s_{t+1})$

与 Sarsa 算法相比：

- 它的 TD target 从 Sarsa 算法中的$r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$变成了$r_{t+1} + \gamma E[q_t(s_{t+1}, A)]$
- 为了求期望值，它需要的计算增多了；但是它的估计方差更小，因为在一次更新过程中，它涉及到的随机变量更少，在 Sarsa 中为$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$，而在 Expected Sarsa 为$(s_t, a_t, r_{t+1}, s_{t+1})$

Expected Sarsa 算法实际上是求解另一种贝尔曼公式，该贝尔曼公式表达式为：

$$q_\pi(s, a) = E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a], \forall s, a$$

5. n-step Sarsa 算法

action value 的定义为:

$$q_\pi(s,a) = E(G_t|S_t = s, A_t = a)$$

我们将G_t（注意：G_t只是一条 trajectory 的 discounted return）以不同方式分解,

可以表示为:

$$Sarsa \leftarrow G_t^{(1)} = R_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})$$

$$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(s_{t+2}, a_{t+2})$$

$$\ldots$$

$$n - step\ Sarsa \leftarrow G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^n q_\pi(s_{t+n}, a_{t+n})$$

$$\ldots$$

$$MC \leftarrow G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$

- Sarsa 要求解的贝尔曼公式表示为:

$$q_\pi(s,a) = E[G_t^{(1)}|s,a] = E[R_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})|s,a]$$

- MC 要求解的问题为:

$$q_\pi(s,a) = E[G_t^{(\infty)}|s,a] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots|s,a]$$

因为 MC 是使用一个多个 episode 的 return 的期望值来估计 action value，所以该

episode 得走到底，G_t被无穷展开

- 而进行折中展开的算法则被称为 n-step Sarsa，它要求解的贝尔曼公式表示为:

$$q_\pi(s,a) = E[G_t^{(n)}|s,a] = E[R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^n q_\pi(s_{t+n}, a_{t+n})|s,a]$$

则 n-step Sarsa 算法表示为:

$$\begin{aligned} q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) \\ &\quad - \alpha_t(s_t, a_t)\big[q_t(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^n q_t(s_{t+n}, a_{t+n})]\big] \end{aligned}$$

- 当$n = 1$ 时，n-step Sarsa 算法就变成 Sarsa 算法

- 当$n = \infty$时，取$\alpha_t = 1$，则$q_{t+1}(s_t, a_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$，n-step

Sarsa 算法就变成了蒙特卡洛算法

- n-step Sarsa 算法需要的数据为$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots, r_{t+n}, s_{t+n}, a_{t+n})$
- 因为在 t+n 时刻之前，我们都无法得到数据$(r_{t+n}, s_{t+n}, a_{t+n})$，因此我们无法在时刻 t 对访问到的$(s_t, a_t)$的 action value 估计值进行更新，但是我们可以在 t+n 时刻对其进行更新

$$\begin{aligned} q_{t+n}(s_t, a_t) = q_{t+n-1}(s_t, a_t) \\ - \alpha_{t+n-1}(s_t, a_t)[q_{t+n-1}(s_t, a_t) \\ - [r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^n q_{t+n-1}(s_{t+n}, a_{t+n})]] \end{aligned}$$

- Sarsa 算法和 MC 算法是 n-step Sarsa 算法的两个极端，当 n 比较大时，它更接近蒙特卡洛算法，具有比较大的 variance 和比较小的 bias；当 n 比较小时，它更接近 Sarsa 算法，具有比较小的 variance 和比较大的 bias（由于初始猜测造成，在多次更新后会慢慢减小）

6. Q-learning 算法

Q-learning 算法表示为：

$$\begin{cases} q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - \left[r_{t+1} + \gamma \max_{a \in A} q_t(s_{t+1}, a)\right]\right], \\ q_{t+1}(s, a) = q_t(s, a), \forall (s, a) \neq (s_t, a_t) \end{cases}$$

Q-learning 算法与 Sarsa 算法非常相似，唯一不同的是 TD target：在 Sarsa 算法中，TD target 为$r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$；在 Q-learning 算法中，TD target 为$r_{t+1} + \gamma \max_{a \in A} q_t(s_{t+1}, a)$

Q-learning 算法要求解的贝尔曼最优公式表示为：

$$q(s, a) = E\left[R_{t+1} + \gamma \max_a q(s_{t+1}, a) \,|s, a\right], \forall s, a$$

在强化学习中，存在两种 policy，分别为：

- behavior policy：用于与环境交互来产生经验
- target policy：是我们不断进行优化的策略，目标是最优策略

因此存在两种强化学习的算法：

- on-policy：behavior policy 与 target policy 一致，即我们用策略来与环境进行交互得到 experience，然后再用得到的 experience 来改进策略。我们前面所学的算法都是 on-policy 的
- off-policy：behavior policy 与 target policy 不一致，用 behavior policy 来与环境交互得到 experience，用这些 experience 去不断改进 target policy，target policy 会慢慢收敛到 optimal policy。Q-learning 是一种 off-policy

off-policy 的好处在于：它可以使用任何其他的 policy 所获得的 experience 来对自己的 target policy 进行改进，以达到 optimal policy。在这种情况下，我们可以将 behavior policy 设置地更加 exploratory，让它能够访问每个(s, a)对足够多次

如何判断一个算法是 on-policy 还是 off-policy？

（1） 从数学意义上看算法是在解决什么数学问题

（2） 在算法实施的过程中，需要哪些因素让算法运行起来

判断 Sarsa 算法是 on-policy 还是 off-policy：

（1） Sarsa 算法要求解的贝尔曼公式表示为：

$$q_\pi(s, a) = E[R + \gamma q_\pi(s', a')|s, a], \forall s, a$$

其中有$R \sim p(R|s, a), s' \sim p(s'|s, a), a' \sim \pi(a'|s')$

（2） Sarsa 算法的求解步骤表示为：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\big[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\big]$$

我们需要五元组$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$使算法运作起来：当$(s_t, a_t)$给定时，我们不需要依赖任何策略得到$r_{t+1}$和$s_{t+1}$（它们分别依赖于$p(r|s,a)$和$p(s'|s,a)$），但$a_{t+1}$是依赖于策略$\pi_t(s_{t+1})$的。所以在该算法中，我们既使用策略来与环境交互得到 experience，又对其进行更新，Sarsa 算法是 on-policy

判断 Q-learning 算法是 on-policy 还是 off-policy：

（1） Q-learning 算法要求解的贝尔曼公式表示为：

$$q(s,a) = E\left[R_{t+1} + \gamma \max_a q(s_{t+1}, a) \,|S_t = s, A_t = a\right], \forall s, a$$

（2） Q-learning 算法的求解步骤表示为：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - \left[r_{t+1} + \gamma \max_{a \in A} q_t(s_{t+1}, a)\right]\right]$$

我们需要四元组$(s_t, a_t, r_{t+1}, s_{t+1})$使算法运作起来：当$(s_t, a_t)$给定时，我们不需要依赖任何策略得到$r_{t+1}$和$s_{t+1}$（它们分别依赖于$p(r|s,a)$和$p(s'|s,a)$）

在 Q-learning 中，behavior policy 是用来从s_t产生a_t的，它可以是任何策略；target policy 是我们在更新得到 q 值后，要进行改进的策略

因为 Q-learning 是 off-policy 的，它既可以以 on-policy 形式实现也可以以 off-policy 形式实现。以 on-policy 形式实现的伪代码为：

For each episode, do

If the current s_t is not the target state, do

Collect the experience $(s_t, a_t, r_{t+1}, s_{t+1})$: In particular, take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}.

Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)]\Big]$$

Update policy:

$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}|}(|\mathcal{A}| - 1)$ if $a = \arg\max_a q_{t+1}(s_t, a)$

$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}|}$ otherwise

它与 Sarsa 算法非常类似，唯一不同的地方在于 q 值的更新

以 off-policy 形式实现的伪代码为：

For each episode $\{s_0, a_0, r_1, s_1, a_1, r_2, \ldots\}$ generated by π_b, do

For each step $t = 0, 1, 2, \ldots$ of the episode, do

Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q(s_t, a_t) - [r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)]\Big]$$

Update target policy:

$\pi_{T,t+1}(a|s_t) = 1$ if $a = \arg\max_a q_{t+1}(s_t, a)$

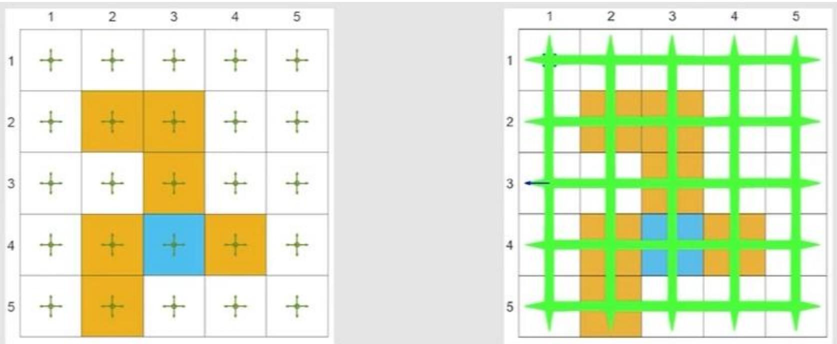
$\pi_{T,t+1}(a|s_t) = 0$ otherwise

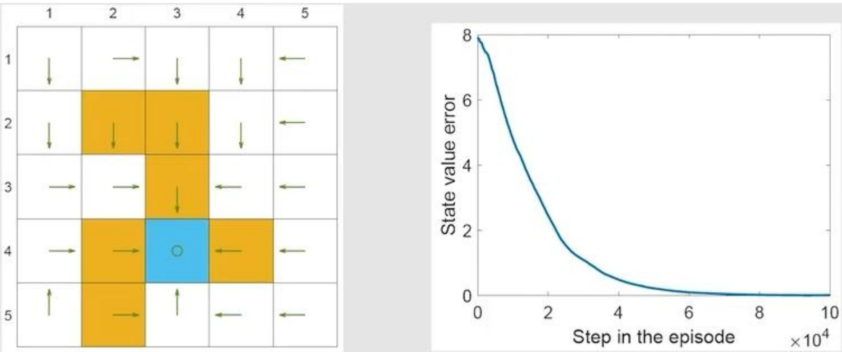
例子：

$$r_{boundary} = r_{forbidden} = -1, r_{target} = 1, \gamma = 0.9, \alpha = 0.1$$

观测不同的 behavior policy 产生的 episode 对 Q-learning 算法的影响：

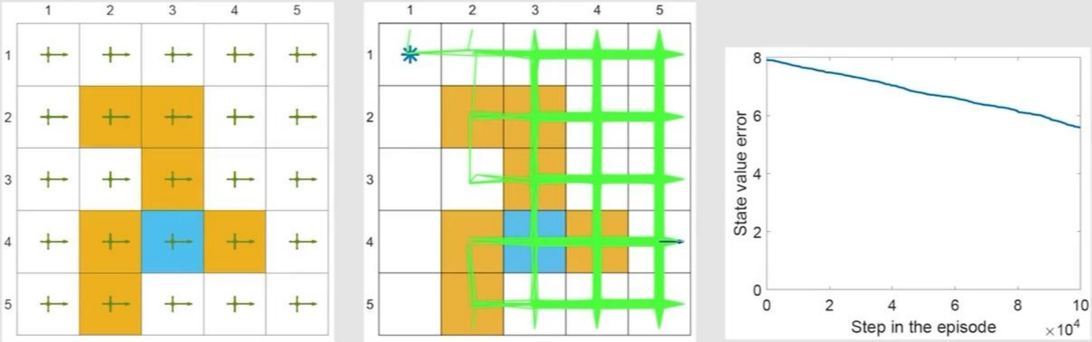
（1） 在该 behavior policy 下，每个 action 的概率是相同的


(a) Behavior policy (b) Generated episode



(a) Estimated policy (b) State value error

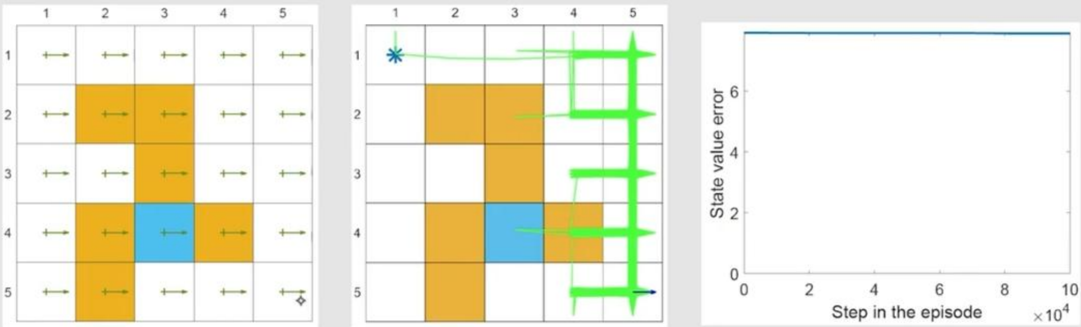
（2） 在该 behavior policy 中，设置$\varepsilon = 0.5$，使得某个 action 的概率略大于其他 action



(a) Behavior policy (b) Generated episode (c) Q-learning result

$\epsilon = 0.5$

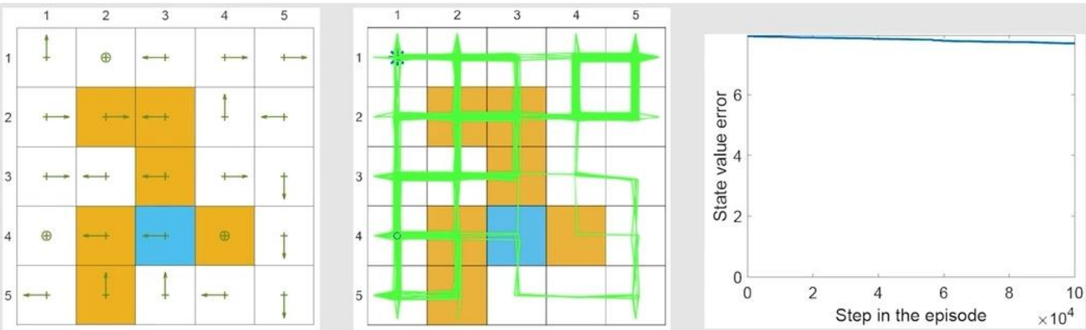
（3） 在该 behavior policy 中，设置$\varepsilon = 0.1$，使得某个 action 的概率明显大于其他 action



(a) Behavior policy (b) Generated episode (c) Q-learning result

$\epsilon = 0.1$

（4） 在该 behavior policy 中，设置设置$\varepsilon = 0.1$，并使每个 state 的 greedy action 可以不同



(a) Behavior policy (b) Generated episode (c) Q-learning result

$\epsilon = 0.1$

7. 总结

以上所有算法的求解步骤都可以被表示为：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - \bar{q}_t]$$

$\bar{q}_t$表示 TD target，不同算法的 TD target 的情况如下所示：

Algorithm	**Expression of $\bar{q}_t$**
Sarsa	$\bar{q}_t = r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$
n-step Sarsa	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n}, a_{t+n})$
Expected Sarsa	$\bar{q}_t = r_{t+1} + \gamma \sum_a \pi_t(a \vert s_{t+1}) q_t(s_{t+1}, a)$
Q-learning	$\bar{q}_t = r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$
Monte Carlo	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \ldots$

并且这些算法都是在求解贝尔曼公式或者贝尔曼最优公式，不同算法求解的问题情况如下所示：

Algorithm	**Equation aimed to solve**
Sarsa	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \vert S_t = s, A_t = a]$
n-step Sarsa	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(s_{t+n}, a_{t+n}) \vert S_t = s, A_t = a]$
Expected Sarsa	BE: $q_\pi(s, a) = \mathbb{E}\big[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}}[q_\pi(S_{t+1}, A_{t+1})] \big\vert S_t = s, A_t = a\big]$
Q-learning	BOE: $q(s, a) = \mathbb{E}\left[R_{t+1} + \max_a q(S_{t+1}, a) \middle\vert S_t = s, A_t = a\right]$
Monte Carlo	BE: $q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots \vert S_t = s, A_t = a]$