强化学习引入

罗淦 北京大学 数学科学学院

李代波 北京大学 数学科学学院

目录

- 策略迭代(回顾)
- 价值迭代
- 异步DP
- Monte Carol
- 时序差分
- 基于价值的优化
- 基于策略的优化

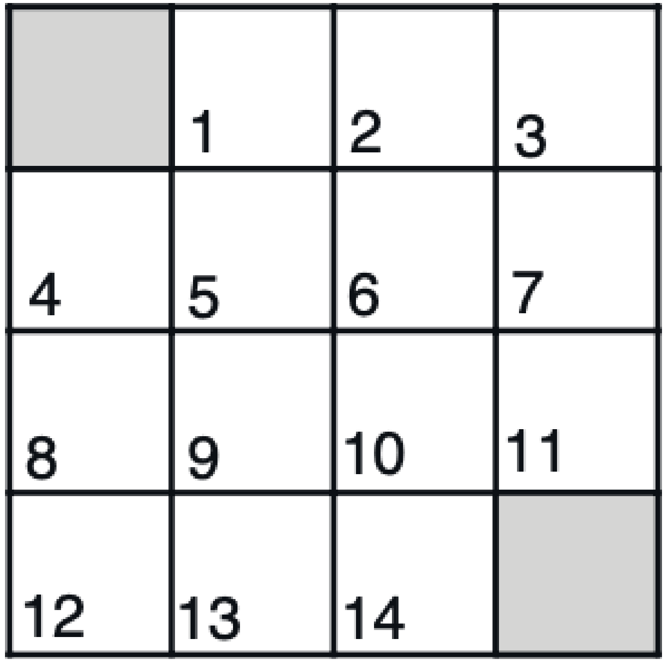
策略迭代(回顾)

策略估计

- 第一步：策略评估

$$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma v_\pi(S_{t+1})\,|S_t = s]$$

- 策略评估使用贝尔曼方程来计算状态价值
- 下面是一个经典的例子计算



策略估计

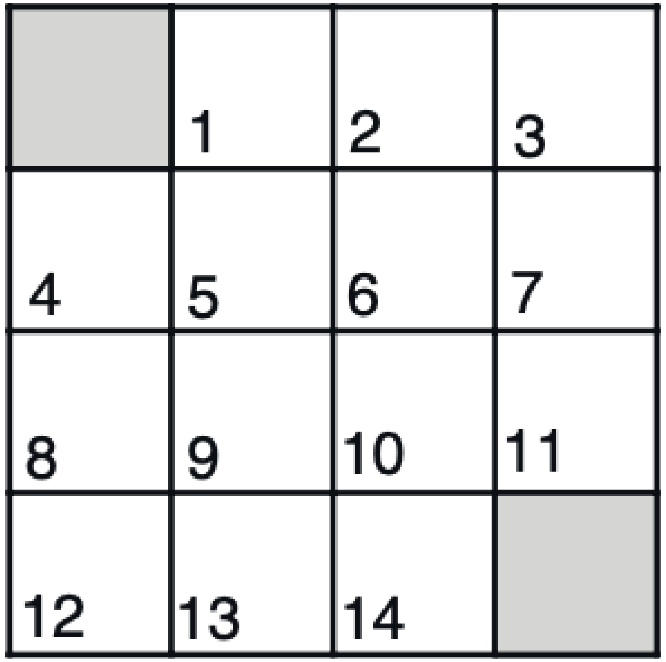
- 终止状态：0 和 15。
- 内部状态：5, 6, 9, 10（动作后确定转移）。
- 边缘状态：如 1, 2, 3, 4, 7, 8, 11, 12, 13, 14（可能因边界限制状态不变）。

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

- 动作：
- 东（+1），南（+4），西（-1），北（-4）。
- 转移规则：
- 内部状态：直接按动作转移。
- 边缘状态：如果超出边界，保持当前状态。
- 终止状态：无动作，价值固定为 0。

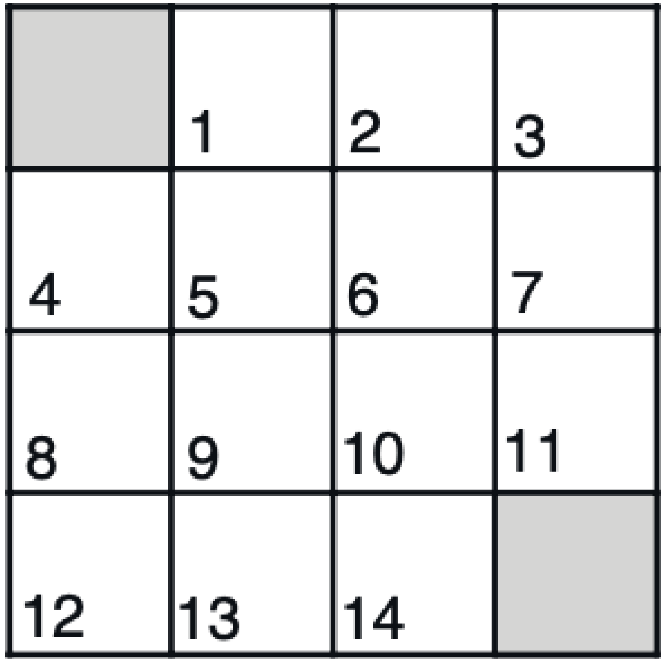
策略估计

- 任何在非终止状态间的转移得到的即时奖励均为-1，进入终止状态即时奖励为0。
- 策略：每个动作概率为 0.25



策略估计, 例子

```python
def policy_evaluation(policy, gamma=1.0, theta=1e-6):
    V = np.zeros(num_states)  # 初始化价值为 0
    while True:
        delta = 0  # 记录最大变化
        for s in range(num_states):
            if s in terminal_states:
                V[s] = 0  # 终止状态价值固定为 0
                continue
            v = V[s]  # 当前价值
            new_v = 0  # 新价值
            for a in range(num_actions):
                next_state = get_next_state(s, a)
                reward = get_reward(s, next_state)
                new_v += policy[s, a] * (reward + gamma * V[next_state])
            V[s] = new_v
            delta = max(delta, abs(v - new_v))
        if delta < theta:  # 收敛条件
            break
    return V
```



策略改进

- 策略改进基于当前评估得到的价值函数
- 如何改进？分析状态-动作价值函数，也就是，在当前状态s，执行动作a之后的期望收益
- 而状态-动作价值函数的计算依赖于之前“估计”的价值函数

策略改进

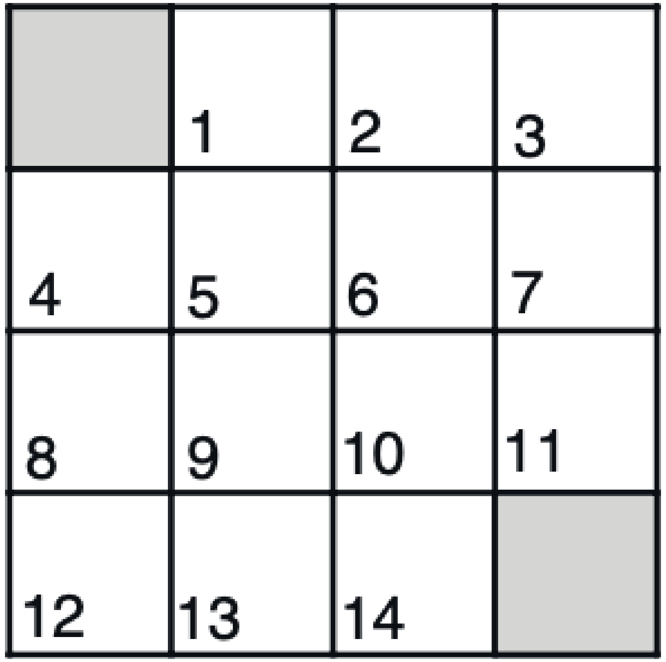
- 策略改进基于当前评估得到的价值函数
- 如何改进？分析状态-动作价值函数，也就是，在当前状态s，执行动作a之后的期望收益（下面是之前的例子中的状态-动作价值函数, 借助Grok）
-

$$Q(s,a) = R(s,a,s') + \gamma V(s')$$

  - s'：执行动作 a 后确定的下一状态。
  - $R(s,a,s')$：从状态 s 执行动作 a 到达 s' 的即时奖励。
  - γ：折扣因子，在您的问题中 $\gamma = 1$。
  - $V(s')$：下一状态 s' 的价值，由策略评估得到。

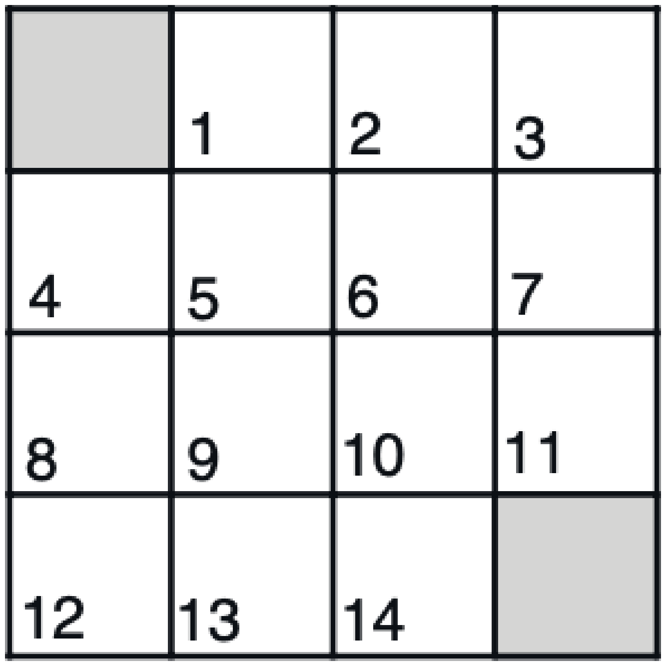
策略改进

- 策略改进基于当前评估得到的价值函数
- 对于状态-动作价值函数Q(s,a)，我们用贪心算法计算对每一个状态s使得Q(s,a)最大的动作a'，作为新的策略



策略改进，例子

```python
def policy_improvement(V, gamma=1.0):
    policy = np.zeros((num_states, num_actions))  # 初始化新策略
    for s in range(num_states):
        if s in terminal_states:
            continue  # 终止状态无动作
        # 计算每个动作的 Q(s, a)
        Q = np.zeros(num_actions)
        for a in range(num_actions):
            next_state = get_next_state(s, a)
            reward = get_reward(s, next_state)
            Q[a] = reward + gamma * V[next_state]
        # 找到最优动作（贪心选择）
        best_action = np.argmax(Q)
        policy[s, best_action] = 1.0  # 将概率设为 1
    return policy
```



价值迭代

价值迭代——解释

- 策略迭代的策略评估需要值函数完全收敛才进行策略提升的步骤(每一步)
- 在策略迭代中关注的是最优的策略，如果说我们找到一种方法，让最优值函数和最优策略同时收敛，那样我们就可以只关注值函数的收敛过程，只要值函数达到最优，那策略也达到最优，值函数没有最优，策略也还没有最优

- 方法：将策略改进视为值函数的改进，每一步都去求解最大的值函数
- 与策略迭代相比，没有等到状态价值收敛才去调整策略，而是随着状态价值的迭代，及时调整策略，这样就大大减少了迭代的次数

价值迭代——解释

- 方法：将策略改进视为值函数的改进，每一步都去求解最大的值函数
- 与策略迭代相比，没有等到状态价值收敛才去调整策略，而是随着状态价值的迭代，及时调整策略，这样就大大减少了迭代的次数

$$v^{(k+1)} = f\left(v^{(k)}\right) = \max_{\pi}\left(r_\pi + \gamma P_\pi v^{(k)}\right), \quad k = 1, 2, 3\ldots$$

- 策略更新（PU）：$\pi^{(k+1)} = \arg\max_\pi \left(r_\pi + \gamma P_\pi v^{(k)}\right)$
- 状态更新（VU）：$v^{(k+1)} = r_{\pi^{(k+1)}} + \gamma P_{\pi^{(k+1)}} v^{(k)}$

- 由于更新策略的时候选取了贪心的方法，所以可以直接使用最优的动作价值去更新状态价值

价值迭代

算法 2.7 价值迭代

为所有状态初始化 V

repeat

  $\delta \leftarrow 0$

  for $s \in \mathcal{S}$ **do**

    $u \leftarrow V(s)$

    $V(s) \leftarrow \max_a \sum_{r,s'} P(r,s'|s,a)(r+\gamma V(s'))$

    $\delta \leftarrow \max(\delta, |u - V(s)|)$

  end for

until δ 小于一个正阈值

输出贪心策略 $\pi(s) = \arg\max_a \sum_{r,s'} P(r,s'|s,a)(r+\gamma V(s'))$

价值迭代和策略迭代

- 策略迭代可以看作每个迭代步骤包含两步骤。第一步是评估当前的策略，第二步是改进当前的策略。其中评估当前的策略需要求解贝尔曼方程
- 如果我们不追求贝尔曼方程的解，仅仅迭代一次；然后就改进策略，那么就是价值迭代
- 并且，价值迭代的过程中没有显式记录每一步的策略

异步DP

异步DP

- 之前的DP都基于同步回溯：每个状态的价值需要回溯扫描计算，这些操作都发生在同一步中。有较高的计算代价(需要对每个状态回溯计算得到value)
- 因此考虑异步DP

异步DP算法

◦ 在位更新（In-Place Update）

同步价值迭代（**Synchronous Value Iteration**）存储价值函数 $V_{t+1}(\cdot)$ 和 $V_t(\cdot)$ 的两个备份：

$$V_{t+1}(s) \leftarrow \max_{a\in\mathcal{A}} R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V_t(s'). \tag{2.48}$$

在位价值迭代只存储价值函数的一个备份：

$$V(s) \leftarrow \max_{a\in\mathcal{A}} R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)V(s'). \tag{2.49}$$

异步DP算法

- 在位更新
- 在同一步，之前更新的s的价值对后面更新的s的价值有影响

同步价值迭代（Synchronous Value Iteration）存储价值函数 $V_{t+1}(\cdot)$ 和 $V_t(\cdot)$ 的两个备份：

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_t(s'). \tag{2.48}$$

在位价值迭代只存储价值函数的一个备份：

$$V(s) \leftarrow \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s'). \tag{2.49}$$

异步DP算法

- 优先扫描
- 误差大的状态优先处理

在异步 DP 中，另一个需要考虑的事情是更新顺序。给定一个转移 (s, a, s')，优先扫描将它的贝尔曼误差（Bellman Error）的绝对值作为它的大小：

$$\left| V(s) - \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \right) \right|. \tag{2.50}$$

它可以通过保持一个优先权队列来有效地实现，该优先权队列在每个回溯后存储和更新每个状态的贝尔曼误差。

异步DP算法

- 实时更新
- 立即更新当前状态的值的估计

3. 实时更新（Real-Time Update）

在每个时间步 t 之后，不论采用哪个动作，实时更新将只会通过以下方式回溯当前状态 S_t：

$$V(S_t) \leftarrow \max_{a \in \mathcal{A}} R(S_t, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|S_t, a) V(s'). \qquad (2.51)$$

它可以被视为根据智能体的经验来指导选择要更新的状态。

2.5 Monte Carol

Monte Carol

- 不需要知道环境的所有信息

- 在对环境只有很少的先验知识时从验中学习来取得很好的效果

Monte Carol 预测

算法 2.8 首次蒙特卡罗预测

输入：初始化策略 π
初始化所有状态的 $V(s)$
初始化一列回报：Returns(s) 对所有状态
repeat
  通过 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 生成一个回合
  $G \leftarrow 0$
  $t \leftarrow T - 1$
  for $t >= 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if $S_0, S_1, \cdots, S_{t-1}$ 没有 S_t **then**
      Returns(S_t).append(G)
      $V(S_t) \leftarrow$ mean(Returns(S_t))
    end if
    $t \leftarrow t - 1$
  end for
until 收敛

Monte Carol 预测

- 对具体策略产生的回报取平均值来从验中评估状态价值函数
- 首次 Monte Carol 和 每次 Monte Carol
-

算法 2.8 首次蒙特卡罗预测

输入：初始化策略 π
初始化所有状态的 $V(s)$
初始化一列回报：Returns(s) 对所有状态
repeat
  通过 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 生成一个回合
  $G \leftarrow 0$
  $t \leftarrow T-1$
  for $t >= 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $\boxed{\textbf{if } S_0, S_1, \cdots, S_{t-1} \text{ 没有 } S_t \textbf{ then}}$
      Returns(S_t).append(G)
      $V(S_t) \leftarrow$ mean(Returns(S_t))
    end if
    $t \leftarrow t-1$
  end for
until 收敛

Monte Carol 预测

- 对具体策略产生的回报取平均值来从验中评估状态价值函数
- 首次 Monte Carol 和 每次 Monte Carol
- 独立地对不同的状态值进行估算
- 它不用其他状态的估算来估算当前的状态值

动作价值的蒙特卡洛估计

- 如果模型可用(即指导转移概率和reward)，那么在得到状态价值函数的估计之后,就可以直接对策略进行评估
- 如果模型不可用,那么还需要估计动作价值（即状态-动作对的期望回报）特别有用，因为它允许智能体基于实际经验来评估不同动作的优劣。不需要环境模型，而是直接从与环境的交互中学习。
- 可能会有一些状态从来都没有被访问过，所以就没有回报。为了选择最优的策略，我们必须要探索所有的状态。一个简单的方法是直接选择那些没有可能被选择的状态-动作对来作为初始状态。这样一来，就可以保证在足够的回合数过后，所有的状态-动作对都是可以被访问的。我们把这样的一个假设叫作叫作探索开始（Exploring Starts）。

动作价值的蒙特卡洛估计

算法 2.9 蒙特卡罗探索开始

初始化所有状态的 $\pi(s)$
对于所有的状态-动作对，初始化 $Q(s,a)$ 和 Returns(s,a)
repeat
  随机选择 S_0 和 A_0，直到所有状态-动作对的概率为非零
  根据 π: $S_0, A_0, R_0, S_1, \cdots, S_{T-1}, A_{T-1}, R_t$ 来生成 S_0, A_0
  $G \leftarrow 0$
  $t \leftarrow T-1$
  for $t >= 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if $S_0, A_0, S_1, A_1 \cdots, S_{t-1}, A_{t-1}$ 没有 S_t, A_t **then**
      Returns(S_t, A_t).append(G)
      $Q(S_t, A_t) \leftarrow \text{mean}(\text{Returns}(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
    end if
    $t \leftarrow t-1$
  end for
until 收敛

蒙特卡洛控制

- 对状态-动作值使用贪心策略，在这种情况下不需要使用环境模型。贪心策略会一直选择在一个状态下有最高价值的动作

$$\pi(s) = \arg\max_a q(s, a)$$

- 而策略估计的方法和DP中完全一样

增量蒙特卡洛

- 通过逐步更新估计值来提高估计的准确性，而不是一次性计算整个

$$
\begin{aligned}
Q_{t+1} &= \frac{1}{t}\sum_{i=1}^{t} G_i \\
&= \frac{1}{t}\left(G_t + \sum_{i=1}^{t-1} G_i\right) \\
&= \frac{1}{t}\left(G_t + (t-1)\frac{1}{t-1}\sum_{i=1}^{t-1} G_i\right) \\
&= \frac{1}{t}(G_t + (t-1)Q_t) \\
&= Q_t + \frac{1}{t}(G_t - Q_t)
\end{aligned}
$$

增量蒙特卡洛

新估计值 ← 旧估计值 + 步伐大小 · (目标值 – 旧估计值)

$$
\begin{aligned}
Q_{t+1} &= \frac{1}{t}\sum_{i=1}^{t} G_i \\
&= \frac{1}{t}\left(G_t + \sum_{i=1}^{t-1} G_i\right) \\
&= \frac{1}{t}\left(G_t + (t-1)\frac{1}{t-1}\sum_{i=1}^{t-1} G_i\right) \\
&= \frac{1}{t}(G_t + (t-1)Q_t) \\
&= Q_t + \frac{1}{t}(G_t - Q_t)
\end{aligned}
$$

时序差分算法

状态	消耗的时间 (分钟)	估计剩余时间	估计总时间
周五晚上六点离开办公室	0	30	30
到车旁，开始下雨	5	35	40
下高速	20	15	35
在路上堵在卡车后面	30	10	40
开到居住的街道	40	3	43
到家	43	0	43

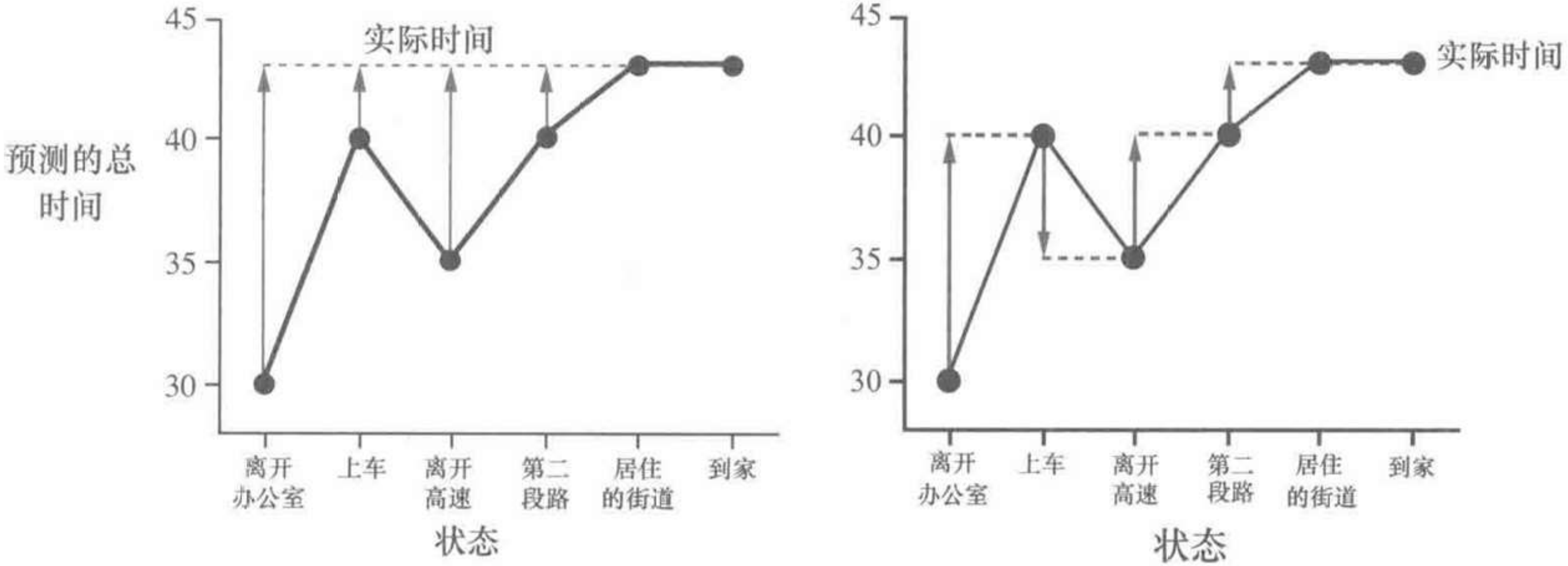


图 6.1 在开车回家这个例子中，蒙特卡洛方法 (左) 和 TD 方法 (右) 分别建议的改变

回顾: 贝尔曼方程

- 回顾贝尔曼方程

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] && (1)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] && (2)
\end{aligned}
$$

- 蒙特卡洛方法将 (1) 式的 $\mathbb{E}_\pi[G_t|S_t = s]$ 作为目标，走完整条路径采集所有奖励 $R_{t+1}, R_{t+2}, \cdots, R_T$ 来估计 G_t.
- 注意到 (2) 式只用到了 S_{t+1} 和 R_{t+1}，能否直接据此估计 G_t?

时序差分算法（Temporal-Difference）

- 这看似简单，直接采样计算 $R_{t+1} + \gamma v_\pi(S_{t+1})$ 就能得到 G_t 的估计?
- 注意 (2) 式的前提是 $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$，为了便于区分，我们将满足前提的这个价值函数记为 $v_\pi^*(s)$. 尽管我们可以采样出 R_{t+1} 和 S_{t+1}，但是 v_π^* 仍是未知的
- 时序差分算法采用**左脚踩右脚**的方式，先任意给定一个 $v_\pi(s)$，再利用

$$v_\pi(S_t) \leftarrow R_{t+1} + \gamma v_\pi(S_{t+1})$$

  不断更新 v_π. (注意：上式并非最终形式，省略了一些细节)
- 因此时序差分算法是**有偏**的。现在的问题是，v_π 会收敛到 v_π^* 吗?

回顾: 动态规划

- 回顾上周动态规划的策略迭代方法，其中策略评估阶段的公式为

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

可以定义一个算子 $\mathcal{T}^\pi(V) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$. $\mathcal{T}^\pi$ 是压缩算子，因为对任意的 s, 有

$$\begin{aligned}
|\mathcal{T}^\pi V(s) - \mathcal{T}^\pi V'(s)| &= \left| \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[\gamma V(s') - \gamma V'(s')] \right| \\
&\leq \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \gamma \|V - V'\|_\infty = \gamma \|V - V'\|_\infty
\end{aligned}$$

回顾: 动态规划

- 上述算子可以写成期望的形式并推广到更一般的情形

$$\mathcal{T}^{\pi}(v(x)) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v(S_{t+1})|S_t = x]$$

- 由压缩映射定理，如果不断进行 $v_{\pi}(s) \leftarrow \mathcal{T}^{\pi}(v_{\pi}(s))$ 的操作，$v_{\pi}(s)$ 最终会收敛到不动点 $v_{\pi}^{*}(s)$ 满足 $v_{\pi}^{*}(s) = \mathcal{T}^{\pi}(v_{\pi}^{*}(s))$
- 还有一个小细节，$\|\mathcal{T}^{\pi}v_1 - \mathcal{T}^{\pi}v_2\|_{\infty} \leq \gamma\|v_1 - v_2\|_{\infty}$ 压缩因子为 γ, 与 π 无关，这可能给策略提升的时候更新 π 提供了某种支持?

时序差分算法

- 我们已相信 $v_\pi(s) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$ 可以收敛到 $v_\pi^*(s)$，但如何估计这里的期望呢?
- 最直接的方法是对状态 S_t，采样多组 (R_{t+1}, S_{t+1})，分别计算 $R_{t+1} + \gamma v_\pi(S_{t+1})$ 并取平均，但需要保存所有数据
- 目前广泛采用的方法是，借鉴蒙特卡洛方法的增量式实现

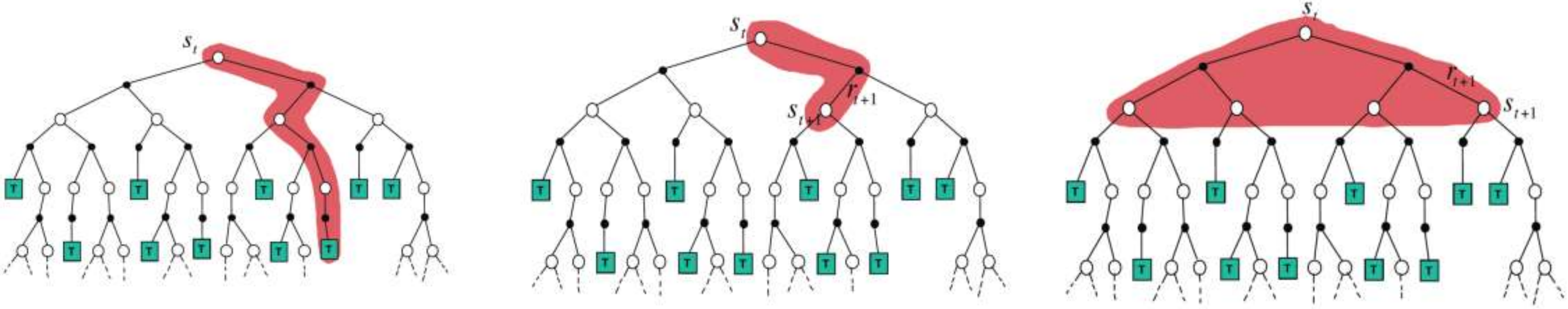
$$v_\pi(S_t) \leftarrow v_\pi(S_t) + \alpha[R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)]$$

算法 2.10 TD(0) 对状态值的估算

输入策略 π
初始化 $V(s)$ 和步长 $\alpha \in (0,1]$
for 每一个回合 **do**
  初始化 S_0
  for 每一个在现有回合的 S_t **do**
    $A_t \leftarrow \pi(S_t)$
    $R_{t+1}, S_{t+1} \leftarrow \text{Env}(S_t, A_t)$
    $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
  end for
end for

偏差（Bias）/方差（Variance）权衡

- 累计奖励 $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ 是 $V^{\pi}(S_t)$ 的无偏估计，收敛性质好
- 时序差分目标 $R_{t+1} + \gamma V(S_{t+1})$ 是 $V^{\pi}(S_t)$ 的有偏估计，存在初值敏感等问题
- 时序差分目标具有比累计奖励更低的方差
  - 累计奖励取决于多步随机动作，多步状态转移，多步奖励
  - 时序差分目标取决于单步动作，单步状态转移，单步奖励



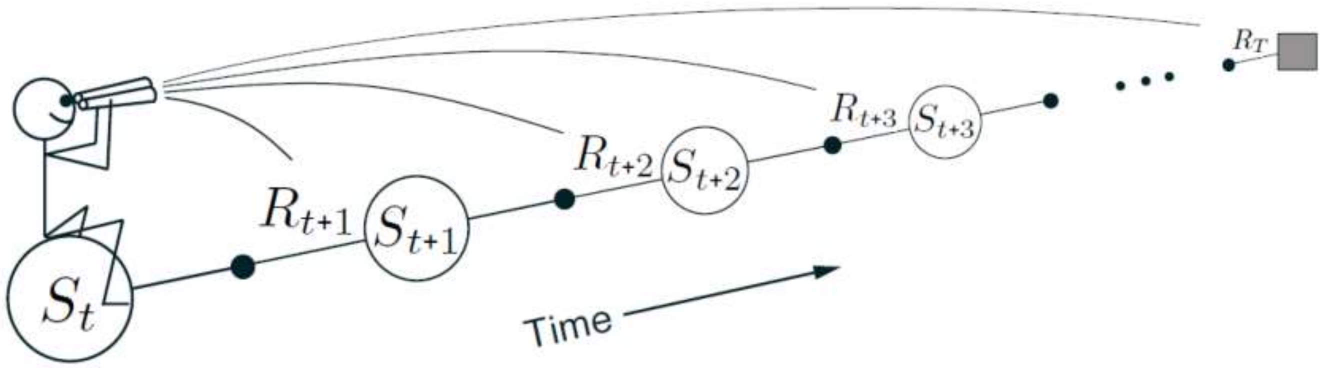
多步时序差分

- 定义n步累计奖励

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n步时序差分学习

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$



TD(λ)算法

- $TD(\lambda)$ 在多步时序差分的基础上使用了加权平均 ($\lambda \in [0,1]$):

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- 这里提一个有趣的现象，如果将 $R_t + \gamma V(S_t) - V(S_{t-1})$ 定义为 δ_t, 则

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+1}) = V(S_t) + \sum_{k=1}^{n} \gamma^{k-1} \delta_{t+k}$$

$$G_t^{\lambda} = G_t^{(1)} + \sum_{k=2}^{\infty} \lambda^{k-1} \left(G_t^{(k)} - G_t^{(k-1)}\right) = V(S_t) + \sum_{k=1}^{\infty} (\lambda\gamma)^{k-1} \delta_{t+k}$$

- 如果 $\lambda = 0$，退回到单步TD(0)，如果 $\lambda = 1$，则退化成蒙特卡洛

时序差分算法（动作价值函数版本）

- 动作价值函数的TD算法需要同时考虑状态-动作对的交替，Sarsa更新规则为

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

- 其中Sarsa 每一步只需给定五元组 $(S_t, A_t, S_{t+1}, A_{t+1}, R_{t+1})$ 即可更新 Q 函数
- Q-Learning 的更新规则如下

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

- Q-Learning 只需要四元组 $(S_t, A_t, S_{t+1}, R_{t+1})$ 即可完成更新

Sarsa 算法

算法 2.12 Sarsa （在线策略 TD 控制）

对所有的状态-动作对初始化 $Q(s, a)$
for 每一个回合 **do**
  初始化 S_0
  用一个基于 Q 的策略来选择 A_0
  for 每一个在当前回合的 S_t **do**
    用一个基于 Q 的策略从 S_t 选择 A_t
    $R_{t+1}, S_{t+1} \leftarrow \mathrm{Env}(S_t, A_t)$
    从 S_{t+1} 中用一个基于 Q 的策略来选择 A_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
  end for
end for

Q-Learning

算法 2.14 Q-Learning （离线策略 TD 控制）

初始化所有的状态-动作对的 $Q(s,a)$ 及步长 $\alpha \in (0,1]$
for 每一个回合 **do**
  初始化 S_0
  for 每一个在当前回合的 S_t **do**
    使用基于 Q 的策略来选择 A_t
    $R_{t+1}, S_{t+1} \leftarrow \mathrm{Env}(S_t, A_t)$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
  end for
end for

在线策略与离线策略

- 行为策略μ：用于与环境交互获取数据的策略
- 目标策略π：算法实际更新的策略
- 在线(同轨)策略方法：更新策略和行为策略必须相同，如 Sarsa
- 离线(离轨)策略方法：更新策略和行为策略可以不同，如 Q-Learning
- 例子1、经验回放: 策略更新后，之前用过的数据是否能反复使用？
  - Sarsa: 对于过去时刻t的数据 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$，假设时刻$t$的策略为 π_t，在时刻$T(> t)$的新策略为 π_T，由于选取A_t的行为策略和选取A_{t+1}的目标策略必须相同，都只能是 π_t
  - Q-Learning:对于过去时刻t的数据 $(S_t, A_t, R_{t+1}, S_{t+1})$，假设时刻$t$的策略为 π_t，在时刻$T(> t)$的新策略为 π_T，由于选取A_t的行为策略和实际更新的目标策略可以不同，可以让π_t充当行为策略，π_T充当目标策略继续学习
- 例子2、Q-Learning 的行为策略常常选用ϵ-贪婪策略进行学习

策略优化

策略优化算法

- **基于价值的优化：**通过优化动作价值函数来获得对动作选择的偏好，如Q-Learning
- **基于策略的优化：**根据采样的奖励值来直接优化策略，如 REINFORCE，交叉熵
- 二者的结合：**Actor-Critic** 方法

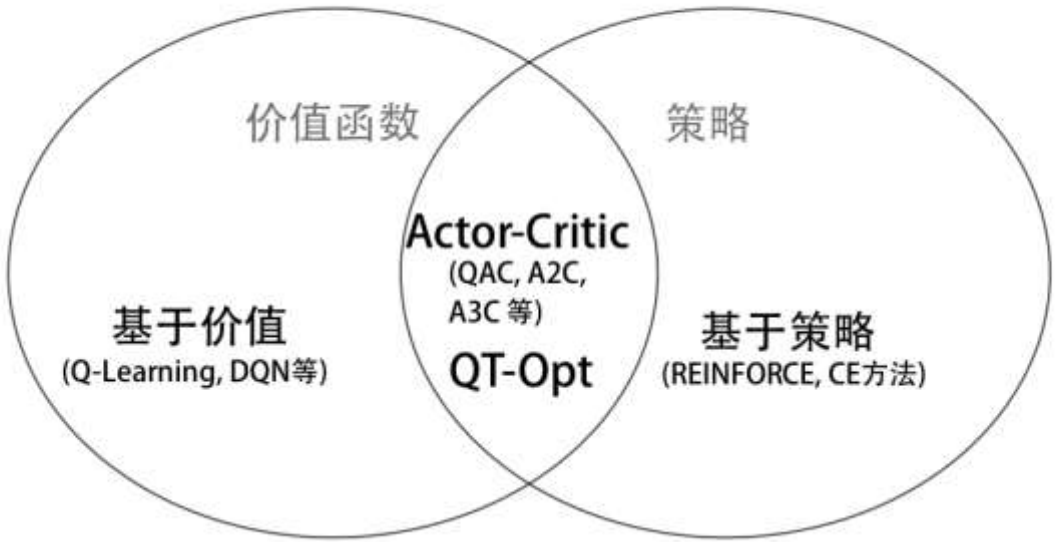
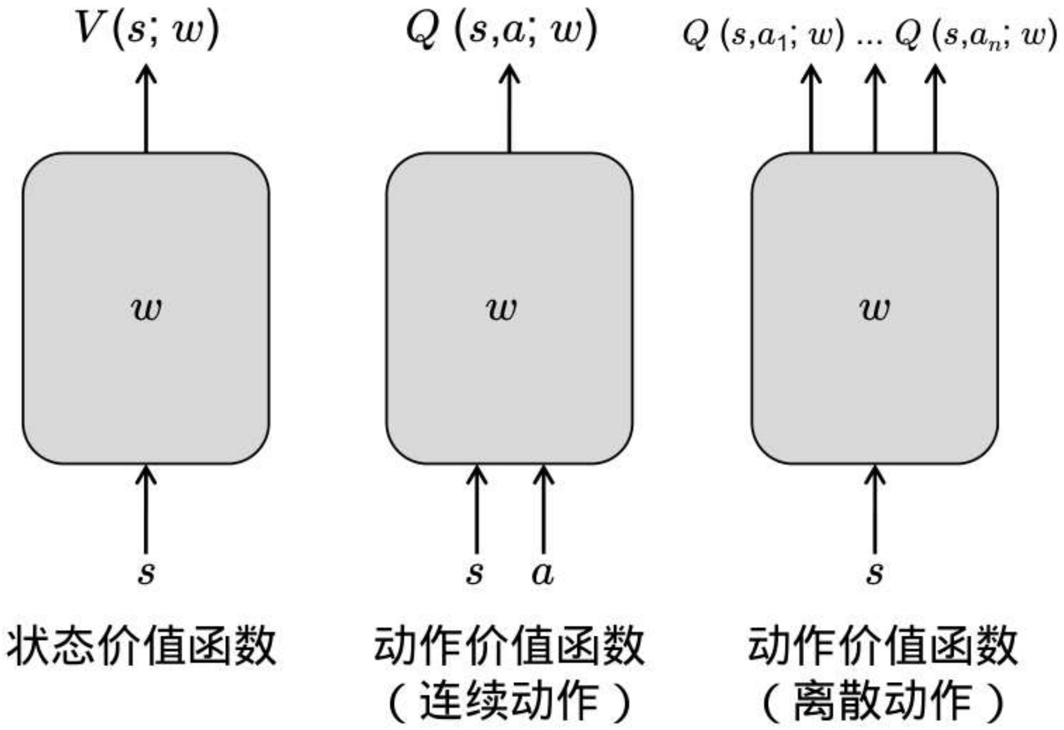


图 2.13 强化学习中策略优化概览

基于价值的优化

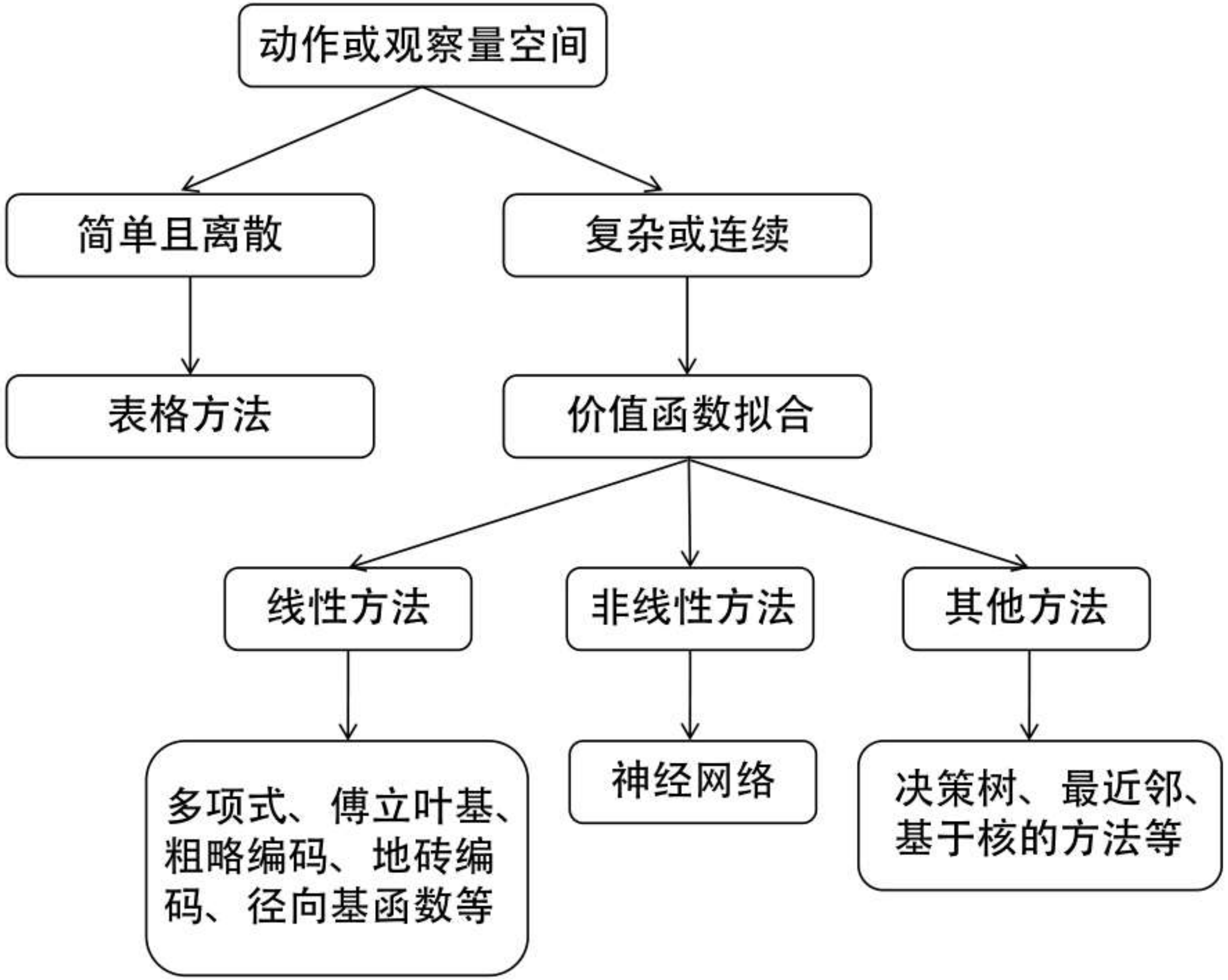
- 在 Q-Learning 中，我们使用传统查找表(Lookup Table)方法，为每个状态-动作对记录一个值，然而现实世界可能有更大和更复杂的状态动作空间，甚至可以是连续的，例如围棋有约 10^{170} 个状态
- 为了将基于价值的强化学习应用到大规模任务上，**函数拟合器**可用来应对上述限制条件，一般记函数拟合器的参数为 w, 例如 $V(s;w)$, $Q(s,a;w)$.



基于价值的优化

函数拟合器包括

- 线性方法：权重 $\boldsymbol{\theta}$ 和特征实数向量 $\phi(s) = (\phi_1(s), \phi_2(s), \cdots)$的线性组合
- 非线性方法：人工神经网络
- 其他方法：决策树，最近邻等



基于价值的优化

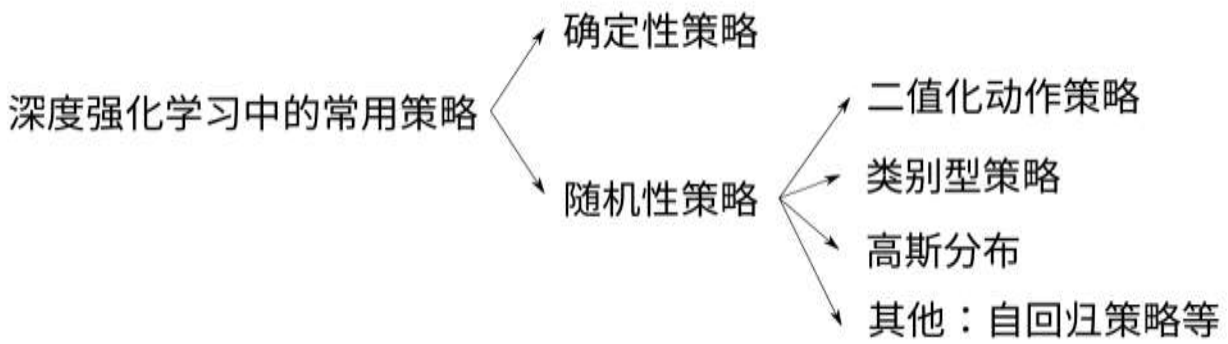
- 如何更新函数拟合器的参数 w?
- 对于函数拟合器 $V^{\pi}(s;w)$ 和 $Q^{\pi}(s,a;w)$, 假设真实目标分别为 $v_{\pi}(s)$ 和 $q_{\pi}(s,a)$, 则优化目标可以设置为二者的均方误差

$$J(w)=E_{\pi}\left[\left(V^{\pi}(s;w)-v_{\pi}(s)\right)^{2}\right]$$
$$J(w)=E_{\pi}\left[\left(Q^{\pi}(s,a;w)-q_{\pi}(s,a)\right)^{2}\right]$$

- 可以通过随机梯度下降法最小化该误差

$$\Delta w=\alpha\left(V^{\pi}(s;w)-v_{\pi}(s)\right)\nabla_{w}V^{\pi}(s;w)$$
$$\Delta w=\alpha\left(Q^{\pi}(s,a;w)-q_{\pi}(s,a)\right)\nabla_{w}Q^{\pi}(s,a;w)$$

- v_{π}和q_{π}可以是被估计的
  - 例如对于蒙特卡洛方法，$v_{\pi}(s)$ 是用采样回报 G_t 估计的
  - 对于时序差分， $v_{\pi}(s)$是用目标函数 $R_t+\gamma V\_\pi(S_{t+1};w_t)$ 估计的，例如后续的DQN

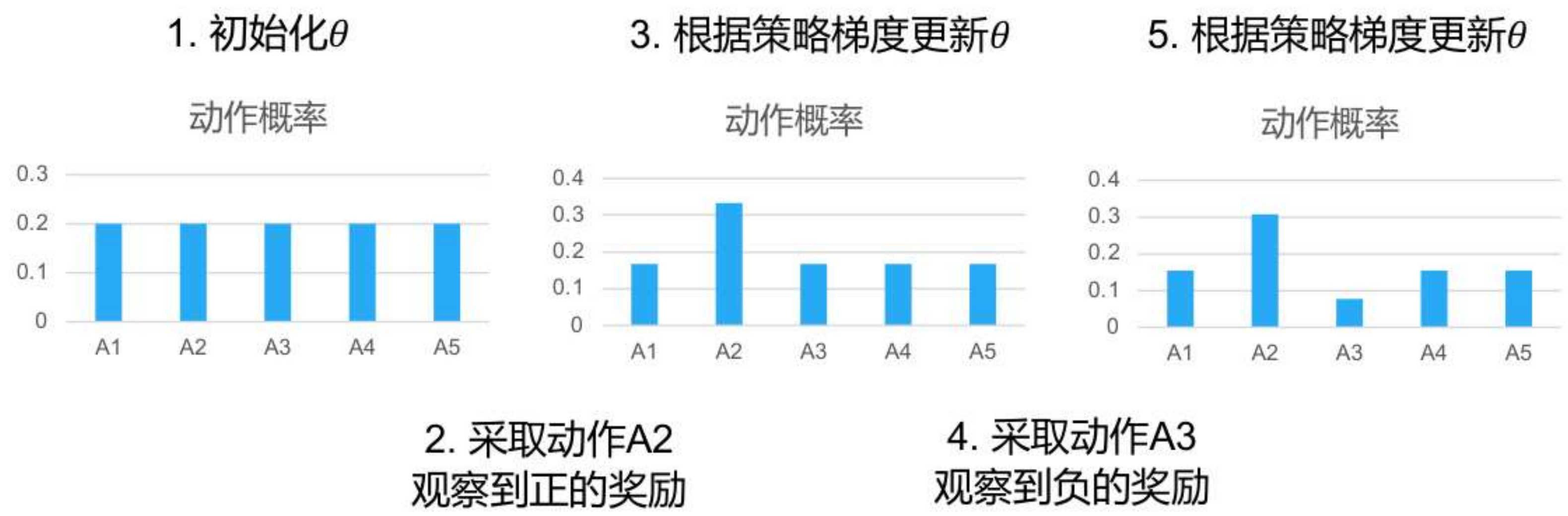


基于策略的优化

- 策略当然也可以参数化，我们之前接触的 ϵ-贪婪策略就含有参数 ϵ. 我们一般用 θ 表示策略的参数。策略可以分为**确定性策略** $A_t = \mu_\theta(S_t)$ 和**随机性策略** $A_t \sim \pi_\theta(\cdot\,|S_t)$.
- 确定性策略同样可以用线性方法，神经网络或者其他方法
- 随机性策略除神经网络外，还有如下方法
  - 二值化动作策略（如伯努利分布 $P(s;\theta) = \theta^s(1-\theta)^{1-s}, s \in \{0,1\}$）
  - 类别型策略（各种分类器，常用 softmax 激活函数）
  - 对角高斯策略（多变量高斯分布，但是协方差矩阵只有对角元非零，即不同动作维度不相关）
    - 除了直接建模 $\theta = (\boldsymbol{\mu}, \boldsymbol{\sigma})$ 以外还可以使用再参数化方法，令该多变量高斯分布的均值和方差为 $\boldsymbol{\mu}_\theta$ 和 $\boldsymbol{\sigma}_\theta$
  - 其他方法

基于策略的优化

- 如何更新策略 $\pi_\theta(a|s) = P(a|s;\theta)$ 的参数 θ 呢？
- 直觉上我们应该
  - 降低带来较低价值/奖励的动作出现的概率
  - 提高带来较高价值/奖励的动作出现的概率
- 一个离散动作空间维度为5的例子



基于策略的优化

- 考虑简单的单步马尔可夫决策过程
  - 起始状态 $s \sim d(s)$
  - 进行一步决策后结束，获得奖励 r_{sa}
- 策略的价值期望

$$J(\theta) = E_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) r_{sa}$$

- 直接求导

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \sum_{s \in S} d(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa} \\
&= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa} \\
&= E_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa}\right]
\end{aligned}$$

策略梯度定理

- 用长期的价值函数 $Q^{\pi_\theta}(s,a)$ 代替前面的瞬时奖励 r_{sa}

$$\frac{\partial J(\theta)}{\partial\theta} = E_{\pi_\theta}\left[\frac{\partial \log\pi_\theta(a|s)}{\partial\theta} Q^{\pi_\theta}(s,a)\right]$$

- 证明比较复杂，可以参考 Sutton 的《强化学习》第2版第13章
- 顺带一提，策略优化的本节虽然只是《深度强化学习》的一小节，但是 Sutton 的《强化学习》中花了整整5章、差不多1/3的篇幅来讲述，由此可见其内容非常丰富，远远不是十几页ppt能讲完的，因此感兴趣的话推荐读一下 Sutton 的这本书

结合基于策略和基于价值的方法

- 如果将价值函数 $V(s)$ 或 $Q(s,a)$ 和策略函数 π 都参数化，我们将会有两套参数 w 和 θ.
- 我们将 $\mathrm{V}^{\pi}(s;w)$ 或 $\mathrm{Q}^{\pi}(s,a;w)$ 称为行动者 Actor
- 将 $\pi_{\theta}(a|s)$ 称为批判者 Critic
- 这形成了一个非常重要的算法结构，叫做 Actor-Critic
- 典型的 Actor-Critic 方法包括
  - Q值 Actor-Critic
  - 深度确定性策略梯度 DDPG 等
- 可以按照前面的方式，交替更新 Actor 和 Critic

谢谢！