TechX 深度强化学习课程（四）

何舜成

宽带网数字媒体实验室
清华大学自动化系

2018 年 7 月 20 日

目录

深入强化学习

深入强化学习

任何一种包含有**感知**、**决策**和**目标**的学习行为皆可称为强化学习。

1. 强化学习不同于监督学习：监督学习学到的是数据到标签的映射，并期望能泛化（推广）到训练样本之外的数据上。监督学习不存在交互行为。

2. 强化学习也不同于无监督学习：无监督学习学到的是数据内部结构，而强化学习的目的是得到更高的回报。

强化学习有独特的挑战，一方面需要从经验中**挖掘**（**exploit**）信息，但也要去**探索**（**explore**）未知的动作与状态，以期获得更高的回报。这两者同样重要，如何平衡二者到目前也未得到解决。这种权衡是以往监督学习和非监督学习所不具备的挑战。

强化学习的要素

一些 RL 样例：下棋、石油精馏设备控制、新生羚羊学习奔跑、扫地机器人的决策、优化做早餐的时间。

RL 要素：

- Agent
- Environment
- Policy
- Reward signal
- Value function
- Model (optional)

强化学习的要素

Policy 指 agent 的行为策略，即给定时间点，通过观测当前状态来计算要执行的动作。

Policy 对应神经科学里的**刺激-响应**规则或**关联**效应（这是什么?）。

Policy 可以是通过查表、搜索、计算等方式得到的。

Policy 可以是随机的。

强化学习的要素

Reward signal 是环境反馈给 agent 的即时信号，RL 算法的目标是最大化累积回报。

回报类似于人愉悦或痛苦的感觉。

回报通常是环境状态和 agent 动作的函数，具有随机性。

Value function 是 agent 对未来累积回报的预期，是一种长期而非即时的考量。

值函数是 RL 直接优化的目标。

值函数需要能准确估计未来的回报，否则无法起到作用。

强化学习的要素

Model 模拟了环境的“行为”。如果我们可以很好地给环境建模，我们就可以直接进行规划（planning）。

RL 算法依据是否有环境模型分为 model-based 和 model-free 方法。由于大多数情况下，环境模型未知，而学习环境模型又十分困难，因此当代 RL 算法 model-free 方法更为流行和有效。

Tabular Methods

多臂赌博机

多臂赌博机（Multi-armed Bandit）是一类简单而经典的 RL 问题。

一个赌博机有 k 个摇杆，每拨动一个摇杆，赌博机会打开相应的礼盒，礼盒里有不同的奖励。由于不同摇杆对应的奖励期望不同，而同一个摇杆对应的奖励也有随机性，因此无法只遍历一次就能确定哪个摇杆的收益最高。

要估计拨动某个摇杆的期望收益，也就是估计下式

$$q_*(a) = \mathbb{E}[R_t | A_t = a] \tag{1}$$

其中 A_t 表示 t 时刻的动作，R_t 表示 t 时刻的回报。假设 t 时刻时我们对 $q_*(a)$ 的估计为 $Q_t(a)$，希望 $Q_t(a)$ 能尽量接近 $q_*(a)$。

多臂赌博机

Exploitation vs. Exploration

- 从贪心角度看，我们要拨动目前收益的均值最高的摇杆，以获得当前状态下最高的回报
- 我们也需要去拨动其他摇杆，以找到潜在的回报更高的摇杆

二者显然是矛盾的，但又都是必须的。

Action-value 方法

如果我们完全知道 $q_*(a)$，那么我们每次选期望最大的摇杆即可，由于事先不知道 $q_*(a)$，我们只能用 $Q_t(a)$ 去估计。一个很自然的估计是将以往的回报平均起来。

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \mathbb{I}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{I}_{A_i=a}} \tag{2}$$

其中 $\mathbb{I}_{condition}$ 是指示函数，只有当 condition 为真时等于 1，其他情况等于 0。

当分母项趋于无穷时，根据大数定律，$Q_t(a)$ 将收敛到 $q_*(a)$。而当分母项为 0 时，可以将 $Q_t(a)$ 设定为一个默认值，如 0。

Action-value 方法

贪心算法很容易推导，每次选择 $Q_t(a)$ 最大的即可。

$$A_t = \arg\max_a Q_t(a) \tag{3}$$

但由于我们需要进行探索，通常会选择一个概率 ϵ，即在 $1-\epsilon$ 的概率下选择 value 最大的动作，在 ϵ 的概率下从所有动作中等概率地选择一个执行。这种探索策略称为 $\epsilon - greedy$ 算法。

这是最简单的 RL 算法，实验留作课后练习。

提高计算效率

由于 $Q_t(a)$ 的计算用到了所有历史数据，为了程序能永久执行下去，我们需要设计增量型算法。

设 R_n 为第 n 次执行某动作后的回报，Q_n 为第 n 次执行某动作前对回报的估计，即

$$Q_n = \frac{R_1 + \cdots + R_{n-1}}{n-1} \tag{4}$$

那么通过计算递推式有

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\
&= Q_n + \frac{1}{n}(R_n - Q_n)
\end{aligned}$$

提高计算效率

类似这种更新的式子还会经常在 RL 算法中遇到：

$$NewEstimate = OldEstimate + StepSize[Target - OldEstimate] \tag{5}$$

非静态情形

以上讨论都建立在所有摇杆的回报期望值不随时间变化的基础上。假设期望是时变的，那么我们只能考虑最近几次的回报。迭代更新式需要变为

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n] \tag{6}$$

其中 $\alpha \in (0,1]$ 是一个常数步长，通过计算可以看出 Q_{n+1} 是初始值 Q_1 和历史回报的加权平均

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i
\end{aligned}$$

非静态情形

可以看到，越靠前的回报权重越低，越近期的回报权重越高，而且权重呈指数衰减趋势。

在非静态情形下，如果使用静态情形下的步长，会有怎样的结果呢？这项实验留作课后习题。

UCB 算法

为了保证算法能够有一定的探索性，之前使用了 $\epsilon - greedy$ 算法。然而我们做探索的目的是为了尝试有潜在的更大回报的动作，而不是无目的地从所有动作中随机选一个做探索。

一种叫 UCB（Upper-Confidence-Bound）的算法应运而生。由统计规律可知，当某一个摇杆选的次数越少，那么我们对它回报均值的估计不确定性越大，它回报均值可能的上界就越高。这是 UCB 算法的初衷，具体算法如下

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right] \tag{7}$$

其中 $N_t(a)$ 表示动作 a 在过去 t 步中被选中的次数。

梯度方法

类似于多分类问题，我们也可以选择输出一个所有摇杆上的概率分布。

首先定义偏好函数 $H_t(a)$，当偏好越大时，选择该摇杆的概率越大。选择概率可以仿照多分类的 Softmax 函数，即

$$\pi_t(a) = P(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \tag{8}$$

至于训练，可以考虑梯度上升的方式

$$H_{t+1}(a) = H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} \tag{9}$$

梯度方法

经过推导（留作习题）可以得到如下更新规则。对于 A_t：

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)) \tag{10}$$

对于其他 $a \neq A_t$：

$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) \tag{11}$$

其中 $\bar{R}_t$ 是 t 时刻以前（包括 t 时刻）的回报的均值。类似一种基线，高于基线时增加其概率，低于基线时减少其概率。

马尔科夫决策过程

Agent-Environment 接口

强化学习所区别于监督学习的地方在于 agent 与环境有相互作用，上一节提到的多臂赌博机问题中，我们的动作并不会影响环境，当环境受动作的影响时，问题会复杂很多。幸运的是，我们仍然可以有数学工具来描述这个过程，即**马尔科夫决策过程（MDP）**。

Agent 与环境的交互是一个离散的过程 $t = 0, 1, 2, \ldots$，每一时刻，agent 会得到当前环境的状态 $S_t \in \mathcal{S}$，而后根据状态选择动作 $A_t \in \mathcal{A}(s)$。执行一步之后，环境将提供一个即时回报 $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$，并形成新的状态 S_{t+1}。

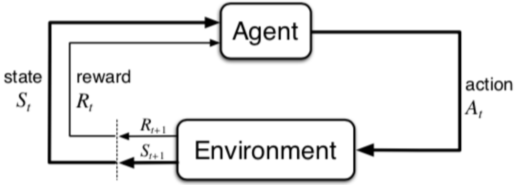

图 1: Agent-Env 接口

Agent-Environment 接口

如此执行下去可以得到一条**轨迹**:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{12}$$

在有限 MDP 情况下，状态集、动作集和回报集都是有限集合。这种情形下，随机变量 R_t 和 S_t 服从离散概率分布，且仅仅由前一时刻的状态与动作决定。此时 MDP 可以用转移概率来描述

$$p(s', r \mid s, a) = P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) \tag{13}$$

该函数 $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ 是四个变量的函数，由于条件概率定义，下式成立

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{14}$$

Agent-Environment 接口

通过计算边际分布还可以得到状态转移概率函数

$$p(s' \mid s, a) = P(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{15}$$

计算关于上一时刻状态与动作条件下回报的期望

$$r(s, a) = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{16}$$

以及后续状态变为 s' 的条件期望

$$r(s, a, s') = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} \tag{17}$$

Agent-Environment 接口

Agent 与 Environment 的界限在哪里?

- 机器人（或机械臂）
- 自动驾驶系统

取决于 agent 所能直接控制的量在哪。

累积回报

强化学习的目标并不是贪心地提高即时回报，而是要最大化累积回报的期望，累积回报定义为

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T \tag{18}$$

T 代表最后一步（the terminal step），由于这种终结，我们可以把 agent 与环境的交互分成一些子片段，叫 episode。

在某些任务里，agent 与环境的交互永远不会终结，如果还按上式定义累积回报，数值将达到无穷大。因此定义折扣累积回报

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{19}$$

其中 $0 \leq \gamma \leq 1$ 称为**折扣因子**。

累积回报

当 $\gamma < 1$ 时，若回报序列 $\{R_k\}$ 有界，那么累计回报也是有界的。

当 $\gamma = 0$ 时，累积回报退化成即时回报。

当 γ 值越大时，agent 就越有远见（考虑更靠后的回报）。

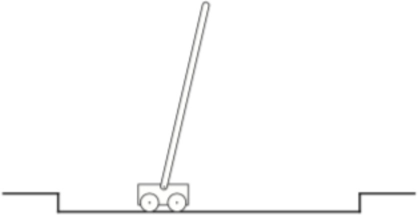

图 2: 平衡车

策略与值函数

由于我们无法获知环境会反馈什么样的回报，因此我们只能通过值函数的方式去估计当前状态（或状态-动作对）后的累积回报。值函数还与当前的策略有关，它所求的期望，是基于当前策略（policy）的，不同的策略显然会导致不同的累积回报以及值函数。

一个策略是指给定状态下，执行下一步动作的概率分布，记为 $\pi(a \mid s)$。

策略 π 下，状态值函数（state value function）定义为

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \quad (20)$$

策略与值函数

同样地可以定义动作值函数（action value function）

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]
\end{aligned} \tag{21}
$$

如果将 G_t 展开为 $R_{t+1} + \gamma G_{t+1}$，我们将得到一个递推式，称为 **Bellman 方程**。

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]
\end{aligned} \tag{22}
$$

最优策略与最优值函数

要定义最优策略，就应先定义策略上的一个“偏序”关系。如果对于任意 $s \in \mathcal{S}$ 有 $v_\pi(s) \geq v_{\pi'}(s)$，那么我们记 $\pi \geq \pi'$。可以证明，至少存在一个策略 π_* 使得它不劣于其他任何策略。那么这些策略称为**最优策略**。

在这种策略下，值函数也是最优的

$$
\begin{aligned}
v_*(s) &= \max_\pi v_\pi(s) \\
q_*(s, a) &= \max_\pi q_\pi(s, a)
\end{aligned}
$$

这两者还存在以下关系

$$
q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \tag{23}
$$

最优策略与最优值函数

最优值函数可以推导出 **Bellman 最优性方程**

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')]
\end{aligned}
\tag{24}
$$

示例：Gridworld

Gridworld 的状态是当前位置，可选动作为上下左右。当处于 A 点时，无论往哪个方向移动，都会产生 +10 的回报，且下一时刻必定回到 A‘点，类似的，处于 B 点时，移动将产生 +5 回报，并强制回到 B’点。若移动后将出界，则会产生-1 的回报，且位置会保持不变。其余情况回报为 0。下图展示了最优值函数和最优策略。

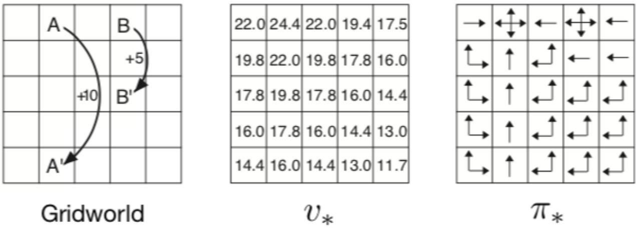

图 3: Gridworld 最优策略

动态规划

策略估值

理论上说，直接解 Bellman 方程可以解出 $v_\pi(s)$，但实际上非常耗费计算，而且需要精确知道环境模型。实际上会用迭代计算方法。

初始化一个值函数估计 v_0，例如将所有 $v_0(s)$ 设为 0，之后利用 Bellman 方程迭代更新

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \end{aligned} \tag{25}$$

由于 v_π 是该方程的不动点，序列 $\{v_k\}$ 将保证收敛到 v_π。

该算法本需要两个数组（新旧估值）来存储，但也可以使用即更即用的形式，只使用一个数组，按顺序更新，更新后的值直接覆盖原值。实验上可以观察到这种方式收敛速度更快。

策略估值

这种算法称为**迭代策略估值**

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

图 4: Alg. Iterative Policy Evaluation

示例：Gridworld

所有操作都会产生-1 的回报，直到行进到灰色方块，游戏终止。

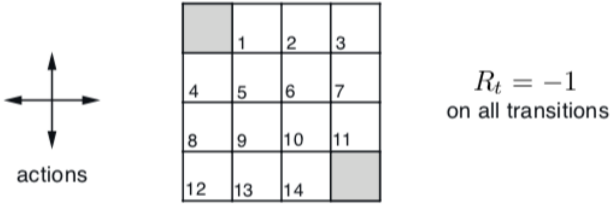

图 5: 新版 Gridworld

示例： Gridworld

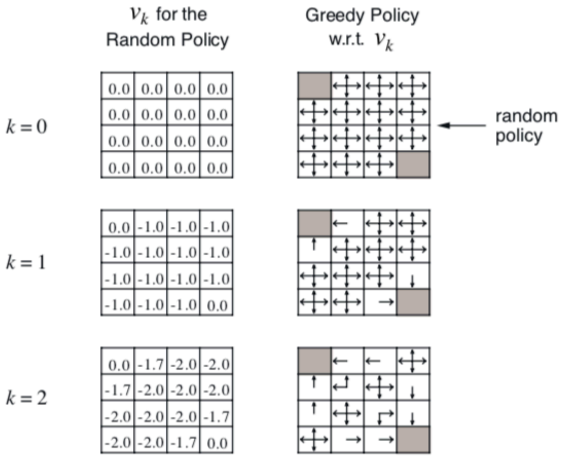

图 6: 策略估值迭代（1）

示例： Gridworld

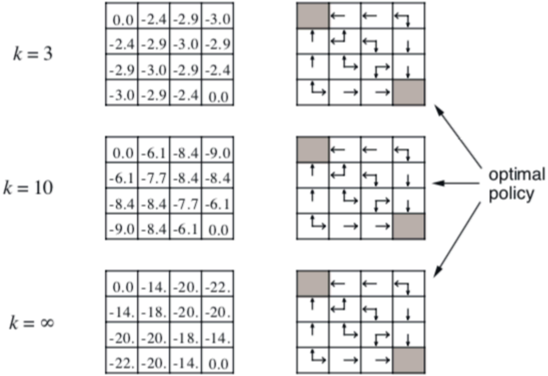

图 7: 策略估值迭代（2）

策略改进

当策略估值计算好之后，结合 q_π 与 v_π 之间的关系，我们是否能据此改进策略呢?

$$q_\pi(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{26}$$

若存在某个 s，和对应的 a，使 $q_\pi(s,a) > v_\pi(s)$，那么对于一个新的策略 π'，它和 π 只有一处不同，那就是每次遇到 s 都执行 a。这种情况下，$\pi' \geq \pi$ 是否成立呢?

下面需要证明**策略改进定理**（policy improvement theorem）。

策略改进

定理：假设 π 和 π' 是两个确定性策略，对于任意 $s \in \mathcal{S}$，有

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{27}$$

那么 π' 至少与 π 同样好，或者更好，也就是说对于任意 $s \in \mathcal{S}$，有

$$v_{\pi'}(s) \geq v_\pi(s) \tag{28}$$

策略改进

证明：从条件出发，逐项展开可得

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(a)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}
\tag{29}
$$

策略改进

很容易能够想到一种贪心策略，即，在现有的 v_π 基础上，对于每一个 s，计算所有的 $q_\pi(s,a)$，选取 q 值最大的 a，此时得到了 π'：

$$\begin{aligned}\pi'(s) &= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \sum_{s',r} p(s',r \mid s,a)[r+\gamma v_\pi(s')]\end{aligned} \tag{30}$$

由于 $\max_a q_\pi(s,a) \geq v_\pi(s)$ 恒成立，因此新构造出的策略满足策略改进定理的条件，那么新策略至少不会差于旧策略。

策略改进

假设新策略 π' 不比 π 好，即 $v_{\pi'} = v_\pi$，那么可以推导出

$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\pi'}(s')] \end{aligned} \tag{31}$$

这是 Bellman 最优性方程，也就意味着 $v_{\pi'} = v_*$，π' 和 π 也都是最优策略。同时还说明若 π 不是最优策略，π' 将严格地比 π 好。

同样可以证明随机策略也存在改进定理。

策略迭代法

策略改进通常不会一步到最优，因此需要反复迭代，估值和改进轮流进行。这种算法称为**策略迭代法**。

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

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \mathrm{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ 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

图 8: Alg: Policy Iteration

值迭代法

策略迭代主要缺陷在于每次计算贪婪策略前，都要进行策略估值，估值本身就需要迭代很多步。从 Gridworld 的例子中可以看到，没等估值结束，贪婪策略就已是最优策略了。

问题：是否能截断策略估值呢?

极端情形下，策略估值只迭代一次，这种算法称为**值迭代法。**

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \end{aligned} \tag{32}$$

值迭代法可以视作 Bellman 最优性方程求不动点。

值迭代法

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

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

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

图 9: Alg: Value Iteration

示例：赌博游戏

庄家抛一枚硬币，赌博者根据自己的现有的现金数下注，如果硬币正面朝上，赌博者可以拿到该注等量的现金，否则失去下注的现金。一旦现金归零，游戏结束，一旦现金达到 100 元，赌博者离开赌局。

该游戏中，状态是赌博者当前现金量 $s \in \{1, 2, \ldots, 99\}$，动作是可下注的现金量 $a \in \{0, 1, \ldots, \min(s, 100 - s)\}$，当现金达到 100 元时，回报为 $+1$，否则均为 0。假设硬币正面朝上的概率已知 $p_h = 0.4$，求最优策略。

示例：赌博游戏

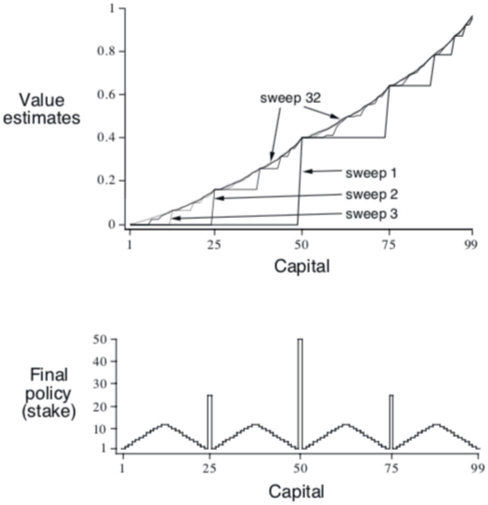

图 10: 值迭代与最优策略