Reinforcement Learning: tabular method Note

FanmingL

2018 年 12 月 12 日

摘要

刚入门强化学习，总算把 Reinforcement Learing: An Introduction 的 Part I 看完了，现在重新过一遍，整理些笔记，以供以后的自己嘲笑。

1 表格方法

强化学习的表格方法 (tabular method) 即状态空间 s，动作空间 a 有限的。这种情况下，这些值函数如 Q(s,a)，可以用表格去表示。

1.1 多臂赌博机

多臂赌博机是一个最简单的强化学习问题，因为他只有一个 state。

1.1.1 多臂赌博机问题

你面前有个赌博机，它上面有 k 个手臂，每次摇动一个手臂你都可以得到一定的奖赏，每个臂的期望奖赏是固定的，也是我们不知道的，每个臂输出的奖赏，是以他的期望奖赏为期望的一个随机数值。这可以抽象成一个强化学习的问题，我们的动作空间是摇动不同的赌博臂，而对于每个动作，我们的期望回报可以写成式1。

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

若我们知道每个动作 a 对应的 $q_*(a)$，那么我们每次选择有最大 q_* 的动作，即可得到最大的回报，但是我们往往不知道 $q_*(a)$，我们只能够去估计它，不妨设每个时间点 t 的 $q_*(a)$ 的估计值为 $Q_t(a)$，我们希望 $Q_t(a)$ 逼近 $q_*(a)$。

在每个时间点，我们都会有对每个动作回报的估计，若我们直接选择使 $Q_t(a)$ 最大的动作的话，我们称这个动作为贪婪动作 (greedy action)，对应的其他动作为非贪婪动作 (nongreedy action)，若你选择 greedy action，我们称此操作为利用 (exploit)，反之，选择 nongreedy action，我们称之为探索 (explore)。利用在当前拥有的信息下，exploit 能够最大化当前的利益，但是，我们选择的 greedy action 的期望回报很有可能比其他的 action 小，而 explore 则能够让我们对其余 action 有更精准的估计，但是 explore 操作很有可能会带来段时间的比 exploit 更低回报，所以如何均衡它们，是我们重点要考虑的问题。

1.1.2 估计动作值函数

上一节提到了要对动作的回报进行估计，这一节，我们主要讨论如何去估计，换句话说，如何估计动作值函数，$Q_t(a)$。

对于一个随机过程，样本均值是对期望的无偏估计，由此我们可以得出 $Q_t(a)$ 的一个计算方法–取样本平均

$$Q_t(a) = \frac{\sum_{i=0}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=0}^{t-1} \mathbf{1}_{A_i=a}} \tag{2}$$

其中 $\mathbf{1}_{predict}$ 指，当 $predict$ 语句为真时 $\mathbf{1}_{predict}$ 的值为 1，否则为 0。

所以式2的意思就是，对每个动作的单步报酬取平均。exploit 过程即取

$$A_t = \arg\max_a Q_t(a)$$

作为当前步的 action。

而实际上，我们在进行 k 臂赌博机的过程中，$Q_t(a)$ 往往都是执行完一个 action 并得到一个 reward 之后，便立即进行更新，所以我们可以把式2写成在线更新的形式，假设当前时刻，取了动作 a_t，并得到单步回报 R，那么假设从开始到现在，有 n 次取了动作 a_t，那么，对动

作 a_t 的期望回报的第 $n+1$ 次估计 Q_{n+1} 可写为

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

为了更好的估计每个动作的值函数，我们不应该每次都取 greedy-action，我们可以选取一个几率 $\varepsilon \in [0,1)$，并以 ε 的概率去做随机的 explore。写成算法即算法1

Algorithm 1 赌博机算法

for $a = 1$ to k **do**
  $Q(a) = 0$
  $N(a) = 0$
end for
while True **do**
  if rand$< \varepsilon$ **then**
    A = random action
  else
    A = $\arg\max_a Q(a)$
  end if
  $R = bandit(A)$
  $N(A) = N(A) + 1$
  $Q(A) = Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$
end while

实际上，式3的在线估计的形式非常常见。

$$NewEstimate = OldEstimate + StepSize\left[Target - OldEstimate\right] \tag{3}$$

若 StepSize 取 $\frac{1}{n}$，那么估计的便是整个随机过程的期望，但若该随机过程是非平稳的，即其期望是随时间变化的，那么 StepSize 取 $\frac{1}{n}$ 这种形式便是不合适的了，我们可以取 StepSize 为一常数 α，这样它就不是估计的从开始到现在的总的期望了，证明如下

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)(\alpha R_{n-1} + (1-\alpha)Q_{n-1}) \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^N Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n}\alpha(1-\alpha)^{n-i}R_i
\end{aligned} \tag{4}$$

而

$$\begin{aligned}\sum_{i=1}^{n}\alpha(1-\alpha)^{n-i} &= \alpha\sum_{i=0}^{n-1}(1-\alpha)^{i}\\ &= \alpha\frac{1-(1-\alpha)^{n}}{1-(1-\alpha)}\\ &= 1-(1-\alpha)^{n}\end{aligned}$$

所以 $\sum_{i=1}^{n}\alpha(1-\alpha)^{n-i}+(1-\alpha)^{n}=1$，所以式 $(1-\alpha)^{n}Q_1+\sum_{i=1}^{n}\alpha(1-\alpha)^{n-i}R_i$ 是一加权平均和，可以看出越新的 R（R_i 的下标越接近 n）的权重越大，所以 StepSize 为一常数的效果即对回报加权求平均，并能够跟踪非平稳过程的期望。

1.1.3 合适的初始值

从式4可以看出，对动作值函数的估计实际上是和初始值 Q_1 的选择有关的，初始值选的不好会引入偏差，当式3中 StepSize 为 $\frac{1}{n}$ 时，若每个动作都至少执行一次的话，这种偏差会消失，但若 StepSize 为常数 α 时，这种偏差会越来越小但是不会消失。

这个问题有好处也有坏处，坏处在于，初始值会作为一个参数让我们去调整，好处在于，我们可以通过这个参数提供先验的信息，同时，还能利用初始值去激励探索。

比如说 k 为 10 的 k 臂赌博机问题，若每个臂的期望奖赏都是 +5，那么我们不妨将初始值都设为 +5。结果表明，在 1000 步之前，纯粹的 greedy 算法甚至能够比初始值都设为 0，$\varepsilon=0.1$ 的 $\varepsilon-greedy$ 算法效果更好。

1.1.4 上限置信区间算法

对于均衡 explore 和 exploit 问题，除了 $\varepsilon-greedy$ 策略之外，还有上限置信区间 (Upper-Confidence-Bound Action Selection) 算法和梯度赌博机算法（Gradient Bandit Algorithms)。

在 UCB 算法中，我们对每一步都取 greedy action 只是，衡量指标变了，每次取

$$A_t=\arg\max_{a}\left[Q_t(a)+c\sqrt{\frac{\ln t}{N_t(a)}}\right] \tag{5}$$

其中 t 为从开始到现在经过的时间，$N_t(a)$ 是当前时刻取到 a 的次数。

这样一种方法引入了 $c\sqrt{\frac{\ln t}{N_t(a)}}$ 项，该项随着时间的推移，若 $N_t(a)$ 不变（未取到过该动作)，那么即使 $Q_t(a)$ 很低，也很有可能取到这个 action，实际的实验发现 UCB 的收敛速度会比 $\varepsilon-greedy$ 更快些。

1.1.5 梯度赌博机算法

梯度赌博机算法与上限置信区间算法恰恰相反，梯度赌博机每一步选择动作，都是随机采样得到的。每个动作被选择到的概率由 soft-max distribution 分布，即式6决定的。

$$Pr\{A_t=a\}=\frac{\exp(H_t(a))}{\sum_{b=1}^{k}\exp(H_t(b))}=\pi_t(a) \tag{6}$$

其中 $H_t(a)$ 是对每个动作 a 的一个偏好程度，这是我们需要学出来的，式6使用学出来的偏好给定每个动作的被选择到的概率 $\pi_t(a)$。$H_t(a)$ 的更新策略为，对于选择到的动作 A_t，更新公式为

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

其中 $\bar{R}_t$ 为基线 (baseline)，在这里指到现在为止的平均回报。为了保证 $\sum_a H_{t+1}(a)=\sum_a H_t(a)$ 令其余不等于 A_t 的 a 的偏好设为

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

首先我们初始化所有的 $H_1(a)$ 为 0，那么之后任意时刻，$\sum_a H_t(a)=0$。那么我们是如何得到 $H_t(A_t)$ 的更新公式的呢？证明如下。

当前步的期望回报为 $\mathbb{E}[R_t]=\sum_x \pi_t(x)q_*(x)$。令其对 $H_t(a)$ 求偏导数，即

$$
\begin{aligned}
\frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)} &= \sum_x q_*(x)\frac{\partial \pi_t(x)}{\partial H_t(a)} \\
&= \sum_x q_*(x)\frac{\partial \pi_t(x)}{\partial H_t(a)} + 0 \\
&= \sum_x q_*(x)\frac{\partial \pi_t(x)}{\partial H_t(a)} - \frac{\partial 1}{\partial H_t(a)} \\
&= \left(\sum_x q_*(x)\frac{\partial \pi_t(x)}{\partial H_t(a)}\right) - \frac{\partial \sum_x \pi_t(x)}{\partial H_t(a)} \\
&= \left(\sum_x q_*(x)\frac{\partial \pi_t(x)}{\partial H_t(a)}\right) - B_t\frac{\partial \sum_x \pi_t(x)}{\partial H_t(a)} \\
&= \sum_x (q_*(x)-B_t)\frac{\partial \pi_t(x)}{\partial H_t(a)} \\
&= \sum_x \pi_t(x)(q_*(x)-B_t)\frac{\partial \pi_t(x)}{\partial H_t(a)}/\pi_t(x) \\
&= \mathbb{E}\left[(R_t-\bar{R}_t)\frac{\partial \pi_t(A_t)}{\partial H_t(a)}/\pi_t(x)\right]
\end{aligned}
$$

其中

$$
\begin{aligned}
\frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)}\left[\frac{\exp(H_t(x))}{\sum_{b=1}^k \exp(H_t(b))}\right] \\
&= \frac{1}{(\sum_{b=1}^k \exp(H_t(b)))^2}\left[\left(\sum_{b=1}^k \exp(H_t(b))\right)\frac{\partial \exp(H_t(x))}{\partial H_t(a)} - \exp(H_t(x))\frac{\partial(\sum_{b=1}^k \exp(H_t(b)))}{\partial H_t(a)}\right] \\
&= \frac{1}{(\sum_{b=1}^k \exp(H_t(b)))^2}\left[\left(\sum_{b=1}^k \exp(H_t(b))\right)\frac{\partial \exp(H_t(x))}{\partial H_t(a)} - \exp(H_t(x))\exp(H_t(a))\right] \\
&= \frac{1}{\sum_{b=1}^k \exp(H_t(b))}\frac{\partial \exp(H_t(x))}{\partial H_t(a)} - \pi_t(x)\pi_t(a) \\
&= \frac{\mathbf{1}_{a=x}\exp(H_t(a))}{\sum_{b=1}^k \exp(H_t(b))} - \pi_t(x)\pi_t(a) \\
&= \mathbf{1}_{a=x}\pi_t(x) - \pi_t(x)\pi_t(a) \\
&= \pi_t(x)(\mathbf{1}_{a=x} - \pi_t(a))
\end{aligned}
$$

所以 $\frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)}$ 可以继续化简

$$
\begin{aligned}
\frac{\partial \mathbb{E}(R_t)}{\partial H_t(a)} &= \mathbb{E}\left[(R_t-\bar{R}_t)(\pi_t(x)(\mathbf{1}_{a=x}-\pi_t(a)))/\pi_t(x)\right] \\
&= \mathbb{E}\left[(R_t-\bar{R}_t)(\mathbf{1}_{a=x}-\pi_t(a))\right]
\end{aligned}
$$

根据梯度下降法的准则，

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

得到

$$
H_{t+1}(a) = H_t(a) + \alpha\mathbb{E}\left[(R_t-\bar{R}_t)(\mathbf{1}_{a=x}-\pi_t(a))\right]
$$

但是式中的期望运算 $\mathbb{E}$ 非常不方便，不妨用当前值代替期望值，即随机梯度下降法 SGD

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

其余未选择到的 action 的 H_{t+1} 值则按照之前所说的，更新前后 $\sum_x H(x)$ 的值不变的准则更新。

1.1.6 上下文赌博机

之前我们提到了，多臂赌博机实际上是一个最简单的强化学习的任务，因为它只有一个 state，而上下文赌博机则有多个 state，用某种形式表示给 agent，如颜色的变化代表 state 的变化，我们需要对每个 state 都学一个 $Q_t(a)$，总的表示即学出 $Q_t(a,s)$ 来，这就是上下文赌博机，上下文赌博机更加贴近真实的强化学习的任务。

1.2 有限马尔可夫决策过程

这里我们将引入强化学习最经典的模型—有限马尔科夫决策过程 (Finite Markov Decision Processes)。

1.2.1 代理与环境

代理 (agents) 即学习与决策的智能体，agents 之外的都是环境。

agent 与环境会进行交互，在每个时间点，agent 收到环境当前状态 (state) 的描述，$S_t \in \mathcal{S}$，随后 agent 会依据某个策略选择一个 action，$A_t \in \mathcal{A}(s)$，下个时间点，agent 会得到一个报酬，$R_{t+1} \in \mathcal{R} \subset \mathbb{R}$，同时 agent 会进入到下一个 state，$S_{t+1}$

有限 MDP，states、actions、rewards 的集合元素都是有限的。所以任意一个 $s' \in \mathcal{S}, r \in \mathcal{R}$ 都会有一个概率，概率表示如式7

$$p(s',r|s,a) = Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \tag{7}$$

这就是 MDP 的 model。由于 $p(s',r|s,a)$ 是一个条件概率密度函数，所以对于所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s',r|s,a) = 1$$

同样我们可以求出边缘概率密度分布。

$$p(s'|s,a) = \sum_{r \in \mathcal{R}} p(s',r|s,a) \tag{8}$$

包括给定 s,a 时的期望报酬 $r(s,a)$

$$r(s,a) = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} rp(s',r|s,a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s',r|s,a) \tag{9}$$

同样我们能够得到，若当前状态为 s，并采取动作 a，下一步的状态为 s' 时，r 的概率为

$$p(r|s,a,s') = \frac{p(s',r|s,a)}{p(s'|s,a)} \tag{10}$$

所以此时，r 的期望为

$$r(r|s,a,s') = \sum_{r \in \mathcal{R}} r \frac{p(s',r|s,a)}{p(s'|s,a)} \tag{11}$$

1.2.2 回报

对于一个 MDP，t 时刻的回报 (return)G_t 定义如下

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T \tag{12}$$

其中 T 为最终的时间。所以每一个时刻的 G_t，即下个时刻到最后一个时刻的 reward 之和。显然，这适用于一轮 (episode) 一轮的 MDP，即当 agent 走到终止状态时，一轮结束，并重

新开始一轮。所以 $T < \infty$，但是对于连续的，没有终止状态的 MDP，$T = \infty$，我们需要对每个时刻的 reward 加权重，防止 G_t 趋于无穷大。即

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{13}$$

所以式12只是13的 $\gamma = 1$ 的特殊情况，且式13可以写成递归的形式

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{14}$$

现在可以证明只要 $R_{t+1} < \infty$，那么 $G_t < \infty$。所以无论是连续任务还是轮式任务，其回报 G_t 均可写为

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{15}$$

1.2.3 策略和值函数

几乎所有强化学习的算法都需要估计值函数（如状态值函数、动作值函数）去估计当前状态（或当前状态的执行某个动作）的好坏。

策略指一个状态到动作空间的概率密度的映射。若 t 时刻，agent 按照策略 π 选择动作，那么 $\pi(a|s)$ 就是在 $S_t = s$ 在 $A_t = a$ 下的概率。而强化学习正是描述如何根据 agent 的经验取调整策略的方法。

我们用 $v_\pi(s)$ 表示 π 的状态值函数，对于 MDP，我们可以定义 v_π 为

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{16}$$

表示使用策略 π 时，状态 s 下，回报的期望。

我们用 $q_\pi(s,a)$ 表示 π 的动作值函数，对于 MDP，我们可以定于 $q_\pi(s,a)$ 为

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{17}$$

即使用策略 π 时，在状态 s 下，选择动作 a 的回报的期望。

对 v_π 和 q_π 的估计可以直接使用模型，即式7，也可以用策略 π 进行采样然后取平均。

式16，可写成递归形式

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \mathbb{E}_\pi[\gamma G_{t+1}|S_t = s] \\ &= \left(\sum_r r p(r|S_t = s)\right) + \gamma\left(\sum_{s'} p(s'|S_t = s)\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right) \end{aligned} \tag{18}$$

下面计算 $p(r|S_t = s), p(s'|S_t = s)$，为方便首先计算 $p(r, s', a|S_t = s)$。

$$p(r, s', a|s) = p(r, s'|s, a)\pi(a|s) \tag{19}$$

所以很容易求出 $p(r|S_t = s), p(s'|S_t = s)$

$$\begin{aligned} p(r|S_t = s) &= \sum_a \sum_{s'} p(r, s', a|S_t = s) = \sum_a \sum_{s'} p(r, s'|s, a)\pi(a|s) \\ p(s'|S_t = s) &= \sum_a \sum_r p(r, s', a|S_t = s) = \sum_a \sum_r p(r, s'|s, a)\pi(a|s) \end{aligned} \tag{20}$$

所以式18可以继续化简

$$
\begin{aligned}
v_\pi(s) &= \left(\sum_r rp(r|S_t=s)\right) + \gamma\left(\sum_{s'} p(s'|S_t=s)\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\right) \\
&= \left(\sum_r r\sum_a\sum_{s'} p(r,s'|s,a)\pi(a|s)\right) + \gamma\left(\sum_{s'}\sum_a\sum_r p(r,s'|s,a)\pi(a|s)\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\right) \\
&= \sum_a\sum_{s'}\sum_r p(r,s'|s,a)\pi(a|s)\left[r+\gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\right] \\
&= \sum_a\sum_{s'}\sum_r p(r,s'|s,a)\pi(a|s)\left[r+\gamma v_\pi(s')\right] \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(r,s'|s,a)\left[r+\gamma v_\pi(s')\right]
\end{aligned}
\tag{21}
$$

同样我们可以把式17用 $v_\pi(s)$ 表示。

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

式21,22的结果分别是 v_π 的贝尔曼方程

$$
v_\pi(s) = \sum_a \pi(a|s)\sum_{s',r} p(r,s'|s,a)\left[r+\gamma v_\pi(s')\right] = \mathbb{E}_\pi\left[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s\right] \tag{23}
$$

和 q_π 的贝尔曼方程

$$
q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)(r+\gamma v_\pi(s')) = \mathbb{E}\left[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s,A_t=a\right] \tag{24}
$$

从式23和式24很容易看出

$$
v_\pi(s) = \sum_a \pi(a|s)q_\pi(s,a) \tag{25}
$$

对于每一个给定的策略 $\pi(a|s)$，我们可以由式23得到 $|\mathcal{S}|$ 个（状态空间的大小）线性方程，求解该线性方程我们便可以计算得到该策略下每个状态的值函数，再用该值函数用式24得出该策略下某状态中每个动作的值函数。

1.2.4 最优策略和最优值函数

对于两个策略 π 和 π'，若对于每个状态 s，都满足 $v'_\pi(s) \le v_\pi(s)$，则我们可以认为策略 π 不比 π' 差，即 π 的效果比 π' 好或相等。对于有限 MDP，一定有一个最优策略不比任何策略差，令其为 π^*，它对应的最优状态值函数为 $v_*(s)$，最优动作函数为 $q_*(s\ a)$。数学表示分别为。

$$
v_*(s) = \max_\pi v_\pi(s)
$$

$$
q_*(s,a) = \max_\pi q_\pi(s,a)
$$

我们可以由式24得到

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] = \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$$

由式25可得

$$v_*(s) = \sum_a \pi^*(a|s) q_*(s,a)$$

显然 $v_*(s) \in [\min_a q_*(s,a), \max_a q_*(s,a)]$，其上界为 $\max_a q_*(s,a)$，所以我们很容易想到，要对任意 π, s，满足 $v_\pi(s) \le v_*(s)$，必定要使 $v_*(s) = \max_a q_*(s,a)$，否则，就会存在 π 使 $v_\pi(s) = \max_a q_*(s,a) > v_*(s)$。所以

$$\begin{aligned} v_*(s) &= \max_{a\in\mathcal{A}(s)} q_*(s,a) \\ &= \max_{a\in\mathcal{A}(s)} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_{a\in\mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s')) \end{aligned} \tag{26}$$

于是 $q_*(s,a)$ 为

$$\begin{aligned} q_*(s,a) &= \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s')) \\ &= \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_*(s',a')) \end{aligned} \tag{27}$$

式26和27为 v_* 和 q_* 的贝尔曼最优方程。

对于有限 MDP，v_π 的贝尔曼最优方程只有唯一解，该解是与策略无关的。同样对于 $|\mathcal{S}|$ 个未知数 $v_\pi(s)$，我们能够用式26列出同样个数的非线性方程，理论上只要我们知道了 MDP 的模型 $p(s',r|s,a)$，这些方程可以用很多方法解出。之后便可用式24解出 q_* 出来。用 q_* 可以很方便的得到策略，即直接采用贪心的方式，对于每个状态，取 $\arg\max_a q_*(s,a)$ 作为动作。

这个理论看起来很美，但是真实的世界中 $|\mathcal{S}|$ 往往非常大，并且我们也很难得到准确的 $p(s',r|s,a)$ 模型，这种方法只能用在小的已知模型中。

1.3 动态规划

DP 中的值迭代和策略迭代是非常标准的 model-based 的强化学习的方法。

1.3.1 策略评估

在研究 MDP 时，给定了一个策略 π 时，我们往往解若干个 v_π 的贝尔曼方程23求出每个 $v_\pi(s)$，但是当 $|\mathcal{S}|$ 非常大时，求解这个线性方程组会很麻烦。这里提出了一种迭代的方式去逼近 v_π，迭代公式就是贝尔曼方程，假设状态值函数迭代到了 v_k，那么 v_{k+1} 为

$$v_{k+1}(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1})\right] = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \tag{28}$$

这种迭代方式下，k 趋近于 ∞ 时，v_k 收敛于 v_π

可以看出，每次更新某个状态 s 的值的时候，我们要考虑到所有 s 可能单步转移到的状态 s'，我们称这样的更新方法为期望更新 (expected update)

在编写程序时，我们可以得到了 $v(s)$ 的值的时候，先把它保存到一个列表中，然后再用老的 $v(s)$ 的值去更新其他的的 $v(s)$，还有一种方法是更新了 $v(s)$ 时，直接把 $v(s)$ 的值改掉，其他状态的值函数更新时，就直接用更新后的 $v(s)$ 值去更新，这种方法叫原地更新 (in place)。所以我们可以总结出策略评估的迭代算法，算法2。

可以看出，算法2，是原地更新的算法。

Algorithm 2 迭代策略评估

输入需要评估的策略 π

设置参数 $\theta > 0$，这个值决定了最后值函数的精度

初始化状态值函数表 $V(s)$，对于终止状态，$V(s) = 0$，其他状态 $V(s)$ 任意给定。

repeat

  $\Delta = 0$

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

    $v = V(s)$

    $V(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

    $\Delta = \max(\Delta, |v - V(s)|)$

  end for

until $\Delta < \theta$

1.3.2 策略提升

若有两个确定性的策略 π 和 π' 若对于所有的状态

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

而对于 v_π，显然

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})]|S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})|S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3})|S_t = s] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2})|S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots|S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}$$

至此我们得到一个重要的定理，策略提升定理：对于两个确定性的策略 π, π' 若对于每个状态 s，都满足

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

那么，对于每个状态 s，都满足

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

所以我们只需要选择一个贪心的策略 π'，即

$$\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s, a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned} \tag{29}$$

此时显然满足，对每个状态 s，

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \arg\max_a q_\pi(s, a) \\
&\geq v_\pi(s)
\end{aligned}
$$

所以

$$
\begin{aligned}
v_\pi(s) &\leq v_{\pi'}(s') \\
\pi &\leq \pi'
\end{aligned}
$$

当之前的策略 π 经过式29得到新的策略 π' 时，发现 π' 和 π 一样好，也就是说，对于每个状态 $s \in \mathcal{S}$

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

可以看出，这正是式26，即 v_π 的贝尔曼最优方程。所以此时，$v_{\pi'}$ 正是贝尔曼最优方程的解。所以我们只要循环：策略评估得到当前策略的状态值函数、策略提升得到一个贪心的解，策略就能够收敛到最优策略中去。所以我们可以写出算法3

Algorithm 3 策略迭代

```
对 V(s) ∈ ℝ 和 π(s) ∈ 𝒜(s) 任意初始化
repeat
  Δ = 0
  for 每个 s ∈ 𝒮 do
    v = V(s)
    V(s) = ∑_{s',r} p(s', r|s, π(s))[r + γV(s')]
    Δ = max(Δ, |v − V(s)|)
  end for
until Δ < θ
policy − stable = true
for 每个 s ∈ 𝒮 do
  old − action = π(s)
  π(s) = argmax_a ∑_{s',r} p(s', r|s, a)[r + γV(s')]
  if old − action ≠ π(s) then
    policy − stable = false
  end if
end for
if policy − stable then
  返回 V, π
else
  重复初始化之后的操作
end if
```

1.3.3 值迭代

策略迭代的策略评估一步，可能会多次遍历每个状态，颇为消耗时间，不如把这一步省去，实际上就是直接把贝尔曼最优方程，式26，写成一个更新的方式，即算法4。

Algorithm 4 值迭代

对终止状态的 s 的 $V(s) \in \mathbb{R}$ 初始化为 0，其余任意初始化。
repeat
  $\Delta = 0$
  for 每个 $s \in \mathcal{S}$ **do**
    $v = V(s)$
    $V(s) = \max_a \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$
    $\Delta = \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
输出策略 $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

由策略提升定理可以证明每一次迭代，$V(s)$ 对应的 π 都会变得比原来更好或相同，最终收敛时也满足贝尔曼最优方程，即也能收敛到贝尔曼方程的解。

1.3.4 异步动态规划

无论是策略迭代还是值迭代，每次迭代，都需要遍历每个状态，在状态空间非常大时，非常不合适，甚至不能完成一次迭代。

异步动态规划是指不对状态进行系统的遍历，这一类算法按照某些特定的顺序对状态进行更新，可能某些状态更新了很多次之后，某些状态还没有被更新过。但是为了正确的收敛，异步动态规划往往需要更新所有的状态，但是每个状态的权重不一样。

但是就算每次迭代不对每个状态进行遍历，在状态空间特别大的时候计算量还是会很大。

1.3.5 广义策略迭代

策略迭代的思路是：策略评估 + 策略提升，我们称由这两步构成的算法为广义策略迭代 (Generalized Policy Iteration)。

几乎所有强化学习算法都可以归纳为 GPI。在经过多次的迭代之后，最终值函数和策略都能够收敛到最优值函数和最优策略。因为算法收敛之时，值函数与策略相匹配，并且策略也是值函数的贪心策略，这与贝尔曼最优解的形式是一样的。

GPI 实际上可以看成是一种合作和竞争的关系，策略评估是求得和策略匹配的值函数，而策略提升采用了值函数的贪心策略，若算法还未收敛，那么值函数不再与策略匹配，于是继续进行策略评估，评估之后，策略对于值函数不再贪心，长此以往，算法收敛，也得到了最优的值函数和策略。

1.3.6 动态规划的效率

动态规划的时间复杂度与状态空间、动作空间大小的关系是多项式的关系，而直接求解贝尔曼最优方程的复杂度远远高于动态规划，相比之下，动态规划计算起来会简单很多。

1.3.7 自益

自益 (bootstrapping) 是指，基于其他状态的估计去更新当前状态的方法。

1.4 蒙特卡洛方法

可以看出，动态规划算法在有模型的情况下非常有效，但是没有模型的情况下该怎么办呢，该如何估计值函数呢？蒙特卡洛算法就是一种免模型的强化学习算法 (model-free algorithm)。他用样本均值去估计值函数。

蒙特卡洛算法只需要一个能够产生样本的模型，并不需要模型的具体分布，我们从样本中学出一个策略。蒙特卡洛就像在多臂赌博机中讨论的一样，对每个状态-动作对采样并取平均，唯一的不同是现在，状态个数不为 1 了，就像上下文赌博机一样，有多个状态。蒙特卡洛方法的思想就是上一章提出的 GPI 的思想，所以首先我们要解决的是策略评估。

1.4.1 蒙特卡洛预测

蒙特卡洛方法中的策略评估的方法与动态规划不一样，蒙特卡洛算法用样本均值去代替期望，即在给定的策略 π 下，采样很多轨迹，便可以计算出每个轨迹中每个点对应的状态的回报，每个状态对应的所有回报取个平均，便得到平均，即对状态值函数的估计。如算法5。由大数定律可得，随着轨迹数量的增加，均值将会收敛至其期望，且均值的方差会随着样本数量的增加，按 $\frac{1}{n}$ 的速度衰减。

Algorithm 5 first-visit 蒙特卡洛预测

```
对 V(s) 任意初始化
Returns(s) 设为空表
while true do
  按照 π 采样一段轨迹:S_0, A_0, R_1, S_1, A_1, ···
  G = 0
  for 轨迹的每一步 t = T − 1, T − 2, ··· do
    G = G + R_{t+1}
    if S_t 没有出现在 S_0, S_1, ··· , S_{t−1} 中 then
      在 Returns(S_t) 中插入 G
      V(S_t) = average(Return(S_t))
    end if
  end for
end while
```

算法5是 first-visit 版本的蒙特卡洛预测算法，对一个状态，只取每条轨迹该状态第一次出现时，对应的的 G_t，将其插入至 $Returns(S_t)$ 中，另一种版本是 every-visit，顾名思义，对于每个状态，这条轨迹上该状态每次出现，都将其对应的 G_t 插入至 $Returns(S_t)$ 中，最后取平均。

1.4.2 动作值函数的蒙特卡洛估计

在 GPI 中，策略提升的目标是找到与状态值函数对应的贪心策略，贪心策略是指每次选择能够使得动作值函数最大化的动作，而动作值函数需要用模型和状态值函数求得，但是我们没

有模型，不能够由状态值函数推出动作值函数，所以我们只有直接估计每个动作值函数，同样是用样本均值去估计期望。

问题在于，若我们的策略是一个确定性策略，那么我们采样到的轨迹必然会有很多状态-动作对是无法采样到的，对于一个随机过程，用样本均值代替期望的一个关键在于遍历性，于是我们又碰到了和多臂赌博机中一样的问题—探索和利用的均衡，为了能够遍历所有状态-动作对，我们能够改变的是一个轨迹的起始位置，我们需要保证每个状态-动作对都有可能被选为起点，那么无限次采样之后，我们能够保证遍历性，这就是探索起始的假设。

实际上在我们实际的应用中并不能满足探索起始的假设，因为有的状态-动作对是很难作为起始的。另一个想法是抛弃确定性策略的个前提，让这个策略在所有状态下，每个动作被执行的概率都大于零。

1.4.3 蒙特卡洛控制