Chapter 4 Dynamic Programming

假设：环境是有限MDP，即状态集合S，动作集合A和收益集合R是有限的。

DP 的核心思想是使用价值函数来结构化地组织对最优策略的搜索。本章主要讨论如何使用 DP作为工具 来计算第 3 章中定义的价值函数。

4.1 策略评估（预测）Policy Evaluation (Prediction)

考虑之前提到的状态价值函数 v

$$v_\pi(s) = E_\pi[G_t|S_t = s] = \Sigma_a\pi(a|s)\Sigma_{s',r}p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (4.4)$$

如果已知所有的环境动态特性，即已知所有的p，那么（4.4）就是一个有着|S|个未知数与|S|个等式联立线性方程组。理论上这个方程的解可以得到。

由于过程繁琐，使用迭代法。不妨假设我们迭代得到的价值函数列是

$v_0, v_1, v_2, \ldots,$ 初始 v_0 任意选取，后续的 v_k 则近似使用（4.4）更新。显然 $v_k = v_\pi$ 是这个更新规则的不动点。

事实上，这个序列满足 k → ∞ 收敛到 v_π. 为什么

这个迭代方法就是策略评估。

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Example 4.1

Exercise 4.2非常有趣，新加了一个15，但是15并不会影响12,14的状态价值函数，因此只需要解对13和15的状态函数

Exercise 4.3 这里不需要再对每个action去枚举rewards与states s’，因为一旦s与a确定，另外两个函数就会确定。（甚至四元组p会直接消失）

策略评估就是在更新 v

4.2 策略改进Policy Improvement

显然，我们期望能够找到一种方法来寻求更好的策略，那么我们需要先解决以下问题：

对某一状态 s，我们想知道是否应该选择一个不同于给定策略的动作 a。

一种解决方法是在状态 s 选择动作 a 后，继续遵循现有的策略 π。 这种方法的值为

$$q_\pi(s,a) = E_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a) = \\ \Sigma_{s'\in S(t+1), r\in R(t+1)} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

只需要比较 $q_\pi(s,a)$与 $v_\pi(s)$，即可判断动作a是否有效。

对于策略改进也有如下公式

$$\forall s \in S, q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

π'比 π 一样好或者更好，这样期望回报也会更好

D

当然，如果两个等式都是大于号，也成立。

证明过程如下：

$$\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] && \text{(by (4.6))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] && \text{(by (4.7))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \;\big|\; S_t = s\big] \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \;\big|\; S_t = s\big] \\
&\;\;\vdots \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \;\big|\; S_t = s\big] \\
&= v_{\pi'}(s).
\end{aligned}$$

当策略和状态价值函数已知时，评定某个特定动作改变产生的后果非常容易。（Example4.1）因此，得到以下新的贪心策略非常容易：

$$
\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned} \tag{4.9}
$$

前文考虑的都是确定性策略（$\pi = 1$的策略）但是实际上本节的策略很容易扩展到随机策略中。

4.3 策略迭代Policy Iteration

策略评估与策略迭代的图示

$$
\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,
$$

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

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

Example 4.2

Exercise 4.4

在策略改进的地方,

If old-action $\neq \pi(s)$, then policy-stable ←false

改为 $V(s) < \Sigma_{s',r} p(s',r|s,a)[r+\gamma V(s')]$

Exercise 4.5 ***非常重要

How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to that on page 80 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book

在step2中更新q时，使用

$$
\begin{aligned}
&Loop:\\
&\Delta \longleftarrow 0\\
&\ Loop\ for\ each\ s \in S:\\
&\ \ Loop\ for\ each\ a \in A:\\
&\ \ q \longleftarrow Q(s,a)\\
&\ \ Q(s,a) \longleftarrow \Sigma_{s',r} p(s',r|s,\pi(s))[r+\gamma\Sigma_{a'}\pi(a'|s')Q(s',a')]\\
&\ \ \Delta \longleftarrow max(\Delta, |q-Q(s,a)|)\\
&until \Delta < \theta
\end{aligned}
$$

Exercise 4.6

~~???What's ε-soft???~~

~~噢，记忆恢复大法，想起来了，Chapter2 的内容~~

Step 3改为 以 ε 的概率随机动作

Exercise 4.7 Codes?

周日讲完去写一写吧

4.4价值迭代Value Iteration

因为在有的例子中，迭代了前几步的时候，策略评估将不再影响后续的策略，我们不再需要每一次策略迭代后都更新 v

因此，考虑截断策略迭代中的策略评估步骤。

一种特殊的情况就是只更新一次状态。该算法被称为价值迭代。

$$
\begin{aligned}
&v_{k+1}(s) = max_a E[R_{t+1} + \gamma v_k(S_{t+1})|S_t = t, A_t = a] =\\
&max_a \Sigma_{s',r} p(s',r|s,a)[r+\gamma v_k(s')] \qquad (4.10)
\end{aligned}
$$

*与前文不同，这里需要求最大值

在 v_* 存在的条件下，可以证明 v_k 可以收敛到 v_*

算法

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop 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 $\Delta < \theta$

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

Example 4.3

Exercise 4.8

Because when capital = 50, $V(s) = max_a q(50, a) = q(50, 50) = 0.6 * (0 + V(0)) + 0.4 * (1) = 0.4$

when capital = 51, $V(s) = max_a q(51, a)$

and $q(51, 1) > q(51, 49)$.

Also, there is no need to bet 51.

Exercise 4.9 Codes

Exercise 4.10

$q_{k+1}(s, a) = \Sigma_{s', r} p(s', r|s, a)[r + \gamma max_a' q_k(s, a')$

4.5 异步动态规划Asynchronous Dynamic Programming

不知道这一章在讲什么，说了一堆ADP的优点，但是ADP是什么倒是只字不提

查了一下知乎发现两个写得还不错的帖子

强化学习（三）：动态规划 - 知乎

强化学习知识要点与编程实践（2）——动态规划寻找最优策略 - 知乎

总结一下就是三点思想

1. In-place dynamic programming 不再保留 v的备份
2. prioritised sweeping 对每个状态优先级分级，越高的优先级状态价值越先更新。评判标准是贝尔曼误差：

$$|max_{a\in A}(R_s^a + \gamma\Sigma_{s'\in S}p(s',r|s,a)v(s')) - v(s)|$$

3. real-time dynamic programming

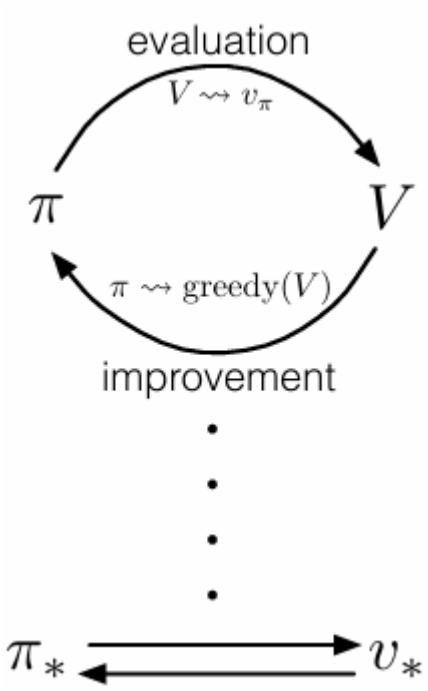
   没太懂这里在干嘛

7.1.3 实时动态规划

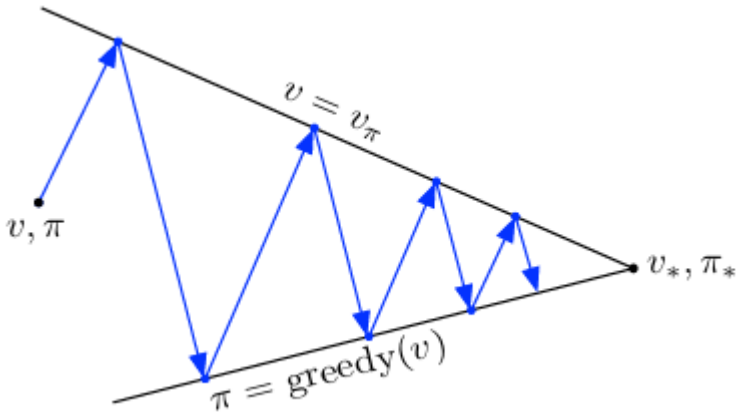
- 思想: **只使用和Agent相关的状态**
- 使用Agent的经验来进行状态的选择
- 在每个时间步 S_t, A_t, R_{t+1} 对状态 S_t 进行备份

$$v(S_t) \leftarrow \max_{a\in\mathcal{A}}\left(\mathcal{R}_{S_t}^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{S_t s'}^a v(s')\right)$$

4.6 广义策略迭代Generalized Policy Iteration



价值函数只有在与当前策略一致时才稳定，并且策略只有在对当前价值函数是贪心策略时才稳定。



可以将 GPI 的评估和改进流程看作竞争与合作。 竞争是指它们朝着相反的方向前进。让策略对价值函数贪心通常会使价值函数与当前策略不匹配，而使价值函数与策略一致通常会导致策略不再贪心。 然而，从长远来看，这两个流程会相互作用以找到一个联合解决方案最优价值函数和一个最优策略。

4.7 动态规划的效率Effciency of Dynamic Programming

如果我们分别用 n 和 k来表示状态和动作的数揖，则意味着一个 DP方法所需要的计算操作的数量少于 n 与 K 的某个多项式函数。 即使（确定）策略的总量达到 k^n ， DP 算法也能保
证在多项式时间内找到一个最优策略。

更适合解决大规模状态空间的问题。（过大时使用异步DP）