第三讲：动态规划

主讲人 陈达贵

清华大学自动化系

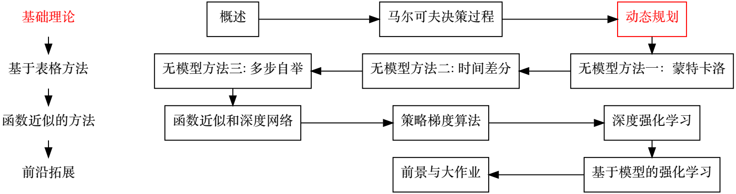
在读硕士

强化学习理论与实践

2018-12-21

目录

章节目录


基础理论
基于表格方法
函数近似的方法
前沿拓展
概述
马尔可夫决策过程
动态规划
无模型方法一： 蒙特卡洛
无模型方法二: 时间差分
无模型方法三: 多步自举
函数近似和深度网络
策略梯度算法
深度强化学习
基于模型的强化学习
前景与大作业

本章目录

什么是动态规划 (Dynamic Programming)?

- 之前提到解决序列决策问题有两种手段——学习与规划
- 当有一个精确的环境模型时，可以用动态规划去解
- 编程算法中也有动态规划的概念，与其相似
- 总的来说，就是将问题分解成子问题，通过解决子问题，来解决原问题
    - 动态: 针对序列问题
    - 规划: 优化，得到策略
- 贝尔曼方程是关键

动态规划可以解决什么问题？

动态规划是一种解决问题的方法，什么样的问题能使用动态规划去解决呢？这样的问题具有以下两种性质：

- 最优子结构
  - 满足**最优性原理**[1]
  - 最优的解可以被分解成子问题的最优解
- 交叠式子问题
  - 子问题能够被多次重复
  - 子问题的解要能够被缓存并再利用

恰好 MDPs 就满足这两个特性：

- 贝尔曼方程是递归的形式，把问题分解成子问题
- 值函数有效地存储了子问题的解，并能够再利用

[1] 多阶段决策过程的最优决策序列具有这样的性质：不论初始状态和初始决策如何，对于前面决策所造成的某一状态而言，其后各阶段的决策序列必须构成最优策略

强化学习中的动态规划

- 使用动态规划解决强化学习问题时，要求知道 MDPs 的所有元素
- 针对评价
  - 输入： MDP$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ 和策略 π
  - 或者: MRP$\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
  - 输出: 值函数 v_{π}
- 针对优化
  - 输入: MDP$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - 输出： 最优值函数 v_*
  - 和: 最优策略 π_*

动态规划的其他应用

动态规划不仅仅用来解决强化学习问题，是运筹学的一个分支。

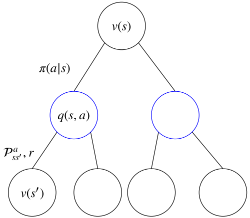
- Richard Bellman 在 1957 年出版作品《Dynamic Programming》
- 分类：线性动规，区域动规，树形动规，背包问题等
- 应用例子：最短路径问题，二分查找树，网络流优化问题等。

目录

策略评价问题

- **问题:** 给定一个策略 π，求对应的值函数 $v_\pi(s)(\text{or } q_\pi(s,a))$.
- 解决方法：
  - 直接解：$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} R^\pi$
    - 可以直接求得精确解
    - 时间复杂度 $O(n^3)$
  - 迭代解：$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$
    - 利用贝尔曼期望方程迭代求解
    - 同样可以收敛到 v_π

利用贝尔曼期望方程的迭代式策略评价



贝尔曼期望方程，表明了我们能够通过后继状态 s' 更新 s

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}(s,a) + \gamma\sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^{a} v_\pi(s')\right)$$

因此可以得到如下的迭代式子

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}(s,a) + \gamma\sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^{a} v_k(s')\right)$$

$$\boldsymbol{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma\boldsymbol{\mathcal{P}}^{\pi}\boldsymbol{v}^{k}$$

同步备份下的迭代式策略评价算法

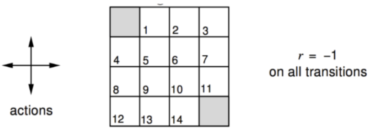
- 四个关键词
  - 备份（backup）：$v_{k+1}(s)$ 需要用到 $v_k(s')$，用 $v_k(s')$ 更新 $v_{k+1}(s)$ 的过程称为备份。更新状态 s 的值函数称为备份状态 s。备份图
  - 同步 (synchronous)：每次更新都要更新完所有的状态
  - 策略评价
  - 迭代式

算法 1 同步备份下的迭代式策略评价算法

```
for k = 1, 2, ··· do
    for 所有的状态 s ∈ S do
        使用迭代式更新值函数 v_{k+1}(s)
    end for
end for
```

注：异步的版本后面会有

策略评价例子



- 假设 $\gamma = 1$
- 14 个普通状态，2 个终止状态
- 走出边界的动作会导致状态不变
- 在走到终止状态前，任何动作都会导致-1 的奖励
- 给定一随机策略 $\pi(a|s) = 0.25, \forall s, a$

策略评价例子

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

通过贝尔曼期望方程验算:

- $-1.0 = -1 + \frac{1}{4} \times 0 + \frac{1}{4} \times 0 + \frac{1}{4} \times 0 + \frac{1}{4} \times 0$
- $-1.7 = -1 + \frac{1}{4} \times -1 + \frac{1}{4} \times -1 + \frac{1}{4} \times -1 + \frac{1}{4} \times 0$
- $-2.0 = -1 + \frac{1}{4} \times -1 + \frac{1}{4} \times -1 + \frac{1}{4} \times -1 + \frac{1}{4} \times -1$

策略评价例子

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

我们可以验证 $k = \infty$ 时，迭代收敛了

- $-14 = -1 + \frac{1}{4} \times 0 + \frac{1}{4} \times -18 + \frac{1}{4} \times -20 + \frac{1}{4} \times -14$
- $-18 = -1 + \frac{1}{4} \times -14 + \frac{1}{4} \times -14 + \frac{1}{4} \times -20 + \frac{1}{4} \times -20$
- $-20 = -1 + \frac{1}{4} \times -18 + \frac{1}{4} \times -18 + \frac{1}{4} \times -20 + \frac{1}{4} \times -20$
- $-22 = -1 + \frac{1}{4} \times -22 + \frac{1}{4} \times -22 + \frac{1}{4} \times -20 + \frac{1}{4} \times -20$

目录

怎么改进策略?

- 给定一个策略 π
  - 评价策略 π

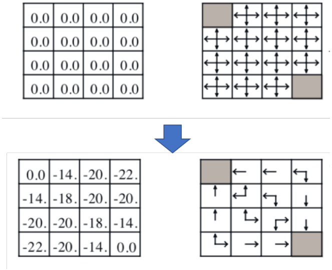
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s\right]$$

  - 在求得 v_π 之后，根据贪婪的动作改进策略

$$\pi' = \text{greedy}(v_\pi) \Leftrightarrow a' = \arg\max_a q_\pi(s,a)$$

- 可以证明 $\pi' \geq \pi$，即 $v_{\pi'}(s) \geq v_\pi(s), \forall s$
- 使得更新后的策略不差于之前的策略的过程称为策略提升
- 贪婪动作只是策略提升一种方式

策略提升



0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

1. 通过策略评价，和贪婪动作，策略从随机策略变成了最优策略 π^*
2. 上述的策略比较幸运，策略提升一次就到达了最优
3. 一般情况下，可能需要多次迭代（策略评价/策略提升）才能到达最优策略

策略提升定理

策略提升定理

对于两个确定性策略 π' 和 π，如果满足 $q_\pi(s, \pi'(s)) \geq v_\pi(s)$，那么我们可以得到

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

贪婪动作得到的策略是上述的特殊形式

$$q_\pi(s, \pi_{\text{贪婪}}) \geq q_\pi(s, \pi'), \quad \forall \pi'$$

证明

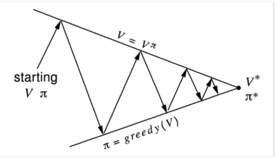
$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] \\
&\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2}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

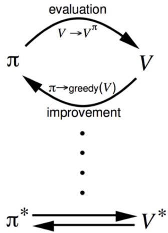
目录

策略迭代

通过不断地交替运行策略评价和策略提升，使策略收敛到最优的策略的过程即为策略迭代

- **策略评价**: 求 v_π。使用方法：迭代式策略评价
- **策略提升**: 提升策略 $\pi' \geq \pi$。使用方法: 贪婪策略提升





收敛证明

- 提升停止时

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- 此时满足了贝尔曼最优方程

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- 此时 $v_\pi(s) = v_*(s), \forall s \in \mathcal{S}$
- 此时 π 是一个最优策略

策略迭代算法（利用迭代式策略评价）

算法 2 策略迭代算法 (利用迭代式策略评价)

1: 随机初始化 $V(s)$ 和 $\pi(s)$
2: **repeat**
3: 对于当前策略 π，使用迭代式策略评价的算法估计 $v_\pi(s)$ 得到 $V(s)$
4: 使用贪婪策略提升得到 $\pi'(s)$
5: **until** 策略保持不变 $\pi'(s) = \pi(s), \forall s$

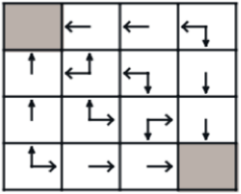
注：这里使用大写的 $V(s)$ 函数，它和小写的区别在于小写的 $v(s)$ 表示真实值，而大写的表示估计值

策略迭代的进一步思考

- 策略迭代分为两个步骤——策略评价和策略提升
- 一般策略评价需要迭代式求解。因此这里存在两个循环
- 策略评价一定要收敛到 v_π，才能进行策略提升吗？
- 我们是不是可以引入提前停止的规则？
  - 例如：值函数更新的 Δ 足够小则停止
  - 例如：限定迭代式策略评价只迭代 k 次
  - 策略评价只迭代一次，就策略提升？（k=1）值迭代

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



广义策略迭代

- 之前的策略迭代指定了策略评价（迭代式）和策略提升（贪婪）的方法。
- 广义策略迭代（Generalised Policy Iteration, GPI）不限定两者的方法，它包含
  - **策略评价**: 估计 v_π。任何策略评价方法均可
  - **策略提升**: 提升策略 $\pi' \geq \pi$。任何策略提升算法均可
- 几乎所有的强化学习算法都可以用 GPI 来描述
- 值函数只有在符合当前策略的情况下才稳定
- 策略只有在当前值函数下是贪婪的才稳定
- 因此稳态下，两者分别收敛到最优的 $v_*(s), \pi_*(s)$

目录

强化学习中的最优性原理

任何最优的策略都能被分解成两部分

- 最优的初始动作 A_*
- 从后继状态 S' 开始沿着最优策略继续进行

强化学习中的最优性原理

一个策略 $\pi(a|s)$ 能够实现从 s 开始的最优值函数，$v_\pi(s) = v_*(s)$，当且仅当

- 对于任何从状态 s 开始的后继状态 s'
- π 能实现从状态 s' 开始的最优值函数 $v_\pi(s') = v_*(s')$

值迭代

- 根据最优性原理，只要知道 $v_*(s')$，即可以知道 $v_*(s)$
- 我们只需要选择一步动作即可

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \left[\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} v_*(s') \right]$$

- 上式中 $[\cdot]$ 的部分表示进行了一步迭代式策略评价，$\max$ 操作符表示进行了一次策略提升
- 值迭代指的是利用上面的迭代式更新
- 相当于从最后的奖励函数出发，递归地求解之前的状态的值函数

例子——最短路径

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

值迭代算法

- 值迭代算法的两种理解方式：
  1. 策略迭代中，在策略评价阶段，只迭代一步
  2. 利用贝尔曼最优方程进行迭代
- 问题仍然为找到最优的策略 π
- 但是在更新的过程中并没有显式的策略

算法 3 同步备份下的值迭代算法

```
for k = 1, 2, ··· do
    for 所有的状态 s ∈ S do
        通过 v_k(s') 更新 v_{k+1}(s)
    end for
end for
```

- 与“同步备份下的迭代式策略评价算法”类似，但是有两点区别
  - 更新的公式不同
  - $v_k(s)$ 的意义不同

值迭代与策略迭代的对比

- 值迭代
  - $v_1 \to v_2 \to v_3 \to \cdots \to v_*$
  - 没有显式的策略
  - 迭代过程中的值函数可能不对应任何策略
  - 效率较高
  - 贝尔曼最优方程
- 策略迭代
  - $\pi_1 \to v_1 \to \pi_2 \to v_2 \to \cdots \to \pi_* \to v_*$
  - 有显式的策略
  - 迭代过程中的值函数对应了某个具体的策略
  - 效率较低
  - 贝尔曼期望方程 + 贪婪策略提升

同步备份下的三种算法总结

问题	贝尔曼方程	算法
评价	贝尔曼期望方程	迭代式策略评价
优化	贝尔曼期望方程 + 贪婪策略提升	策略迭代
优化	贝尔曼最优方程	值迭代

- 算法都是基于状态值函数的（V 函数）
- 如果一共有动作 m 个，状态 n 个，每次迭代的复杂度为 $O(mn^2)$
- 上述算法也可以拓展到状态动作值函数（Q 函数）
- 运用到 Q 函数时，每次迭代的复杂度为 $O(m^2n^2)$

目录

动态规划引申

- 异步动态规划
- 全宽备份和样本备份
- 压缩映射

异步动态规划

- 之前的动态规划方法都使用了同步规划
- 也就是说，每次迭代都会同时保存所有状态的值函数
- 异步动态规划以某种顺序单独考虑每一个状态
- 能够大大减少计算量
- 只要所有的状态都能被持续的选择到，收敛性能够保证
- 常用的三种形式：
  - 就地动态规划
  - 优先清理
  - 实时动态规划

就地 (In-Place) 动态规划

- 同步值迭代存储了值函数的两个副本
  对于每一个 $s \in \mathcal{S}$

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} v_{old}(s') \right)$$

$$v_{old} \leftarrow v_{new}$$

- 就地动态规划仅仅存储一个副本
  对于每一个 $s \in \mathcal{S}$

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

注：可以看出就地动态规划，每一次的更新与值遍历的顺序有关系

优先清理 (Prioritised Sweeping)

- 使用贝尔曼误差的大小来指导状态的选择

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

- 只备份当前贝尔曼误差最大的状态
- 每次备份完之后，更新受到影响的状态的贝尔曼误差
- 受到影响的状态分两类：1，更新的状态作为 s, 2，更新的状态作为 s'
- 第一种情况，贝尔曼误差变为 0，第二种情况要求知道逆运动学 (前驱状态)
- 在编程上，可以通过维护一个优先级队列来完成
- 可以保证每个状态都能被遍历, 因此能收敛

实时动态规划

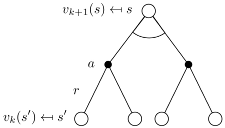
- 之前的动态的规划考虑的是状态粒度的操作
- 实时动态规划考虑了时间粒度的操作
- 仅仅和智能体相关的状态会被备份
- 用智能体的经验去指导状态的选择
- 在每个时间步 t 上，智能体与环境交互了 S_t, A_t, R_{t+1}
- 备份状态 S_t

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

- 并不能保证每个状态被遍历，需要结合一定的探索方法

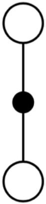
全宽备份

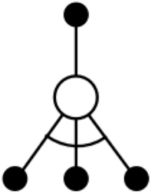
- 动态规划用的是全宽备份
- 全宽备份表示对每一次备份都要考虑到每一个后继状态以及每一个动作
- 要求知道奖励函数 $\mathcal{R}$ 和状态转移函数 $\mathcal{P}$
- 当状态数量较少（数百万）时，动态规划很有效
- 当状态数量太多（维度灾难）时，即使是每一次备份都会需要很久的时间



样本备份

- 强化学习中主要使用样本备份——后面的章节
- 直接通过采样得到转移记录 (transition)$\langle S, A, R, S' \rangle$
- 通过采样替代总体
- 优点：
  - 无模型（model-free）：无需要知道 $\mathcal{R}$ 和 $\mathcal{P}$
  - 通过采样打破了维度灾难
  - 备份的时间复杂度固定，和状态的数量无关





压缩映射 (contraction mapping)

证明了以下的问题

- 值迭代收敛到 v_*?
- 迭代式策略评价收敛到 v_π?
- 策略迭代收敛到 v_*?
- 解是唯一的吗？
- 算法收敛速度？

https://zhuanlan.zhihu.com/p/39279611