



高级人工智能

沈华伟

中国科学院计算技术研究所

2020.12.8


大纲

■ 策略学习

- 动态规划方法
- 蒙特卡洛方法
- 时序差分方法
- 参数近似方法

贝尔曼最优性方程

- 给定策略 π ，状态估值函数的贝尔曼方程


$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_{\pi}(s')], \quad \text{for all } s \in \mathcal{S}$$

- 状态估值函数的贝尔曼最优性方程

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

最优值

动态规划

- 给定策略下状态估值函数的更新规则

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_{\pi}(s')], \quad \text{for all } s \in \mathcal{S}$$

- 最优状态估值函数的更新规则

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

本图表示

注意：这里红色圆圈里的等号表示“赋值”，而不是贝尔曼方程里的“相等”

策略估值

- 给定策略 π ，该策略下的状态估值函数满足

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

- 假如环境 $p(s', r|s, a)$ 已知，状态估值函数的求解方式有
 - 求解线性方程组
 - 计算开销大
 - 寻找不动点——迭代策略估值

迭代策略估值

■ 迭代策略估值的更新规则

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

又被称为“期望更新”



Tip!

状态 s 新一轮的估值是基于 s 所有可能的下一状态 s' 的期望计算得到的（注意：不是基于某一次采样）

迭代策略估值的实现

■ 两种实现方式

- 同步更新：两个数组存放“新”和“旧”的状态估值
- 异步更新：收敛速度快一些，尽早利用了新信息
 - 仅使用一个数据组，“就地”更新（算法如下）

Iterative policy evaluation

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 $s \in \mathcal{S}$:

$v \leftarrow V(s)$

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

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

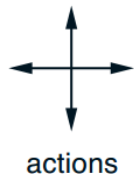
until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

迭代

连续两轮中同一状态估值的最大差

迭代策略估值的例子 李的月抄第4个例子



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

$R_t = -1$
on all transitions

- ✓ 每个状态下，等概率选择四个候选行动
- ✓ 出界时，移动后回到原位置（相当于撞墙）
- ✓ 每次行动获得奖励为-1

迭代策略估值过程

李的月抄

李的月抄

状态估值函数

对应的贪心策略

状态估值函数

对应的贪心策略

$k = 0$

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

	↔	↔	↔
↔	↔	↔	↔
↔	↔	↔	↔
↔	↔	↔	

$k = 3$

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

	←	←	↖
↑	↖	↖	↓
↑	↖	↖	↓
↖	→	→	

$k = 1$

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

	←	↔	↔
↑	↖	↖	↓
↔	↖	↖	↓
↔	↖	→	

$k = 10$

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

	←	←	↖
↑	↖	↖	↓
↑	↖	↖	↓
↖	→	→	

$k = 2$

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

	←	←	↖
↑	↖	↖	↓
↑	↖	↖	↓
↖	→	→	

$k = \infty$

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

	←	←	↖
↑	↖	↖	↓
↑	↖	↖	↓
↖	→	→	

optimal policy

策略提升

- 策略估值的目的是为了寻找更优的策略（策略提升）
 - 策略估值根据策略 π 计算其估值函数 v_π
- 策略提升
 - 根据当前策略的估值函数，寻找更优的策略（如果存在），逐步寻找到最优策略
 - 根据策略 π 的估值函数 v 寻找更优策略 π'
 - 提升方法
 - 给定一个确定策略 π ，在状态 s 下选择行为 a ，后续按照策略 π 行动所得的估值 $q_\pi(s, a)$ 是否高于完全按照策略 π 行动得到的估值 $v_\pi(s)$

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

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

期望

可以写

的公式

策略提升定理

- 对于两个确定式策略 π 和 π' ，如果对于所有状态 s 均满足

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

则策略 π' 优于策略 π ，即


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

下式一个简单例子

注意：上一页的策略提升方法是策略提升定理的一个特例

策略提升

- 给定策略 π ，按照贪心方式得到更优策略 π'

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


对于策略 π , 进行策略估值

- 对于策略 π' ，进行策略估值

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

策略迭代

- 从初始策略 π_0 开始，迭代进行“策略估值(E)”和“策略提升(I)”，最终得到最优策略 π_*

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Policy iteration (using iterative policy evaluation)

1. Initialization

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

状态-动作对集合

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

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

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

until $\Delta < \theta$ (a small positive number)

不需要估值收敛

3. Policy Improvement

$policy_stable \leftarrow true$

For each $s \in \mathcal{S}$:

$old_action \leftarrow \pi(s)$

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

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

改进

不需要收敛迭代

策略迭代分析

- 策略迭代过程中，策略估值需要多次扫描更新状态估值，精确估计出当前策略的状态估值，耗费了大量计算时间
- 是否可以在不精确的状态估值下，进行策略提升呢？
 - 譬如：状态估值进行一轮扫描更新后，便进行策略提升

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

不显式给出当前的策略，而是直接根据当前估值按照贪心策略估计下一轮的值

估值迭代

公式得

- 从初始策略 v_0 开始，进行估值迭代，找到最优状态估值 v_* ，进而根据 v_* 按照贪心方式得到最优策略 π_*

Value iteration

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

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

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

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

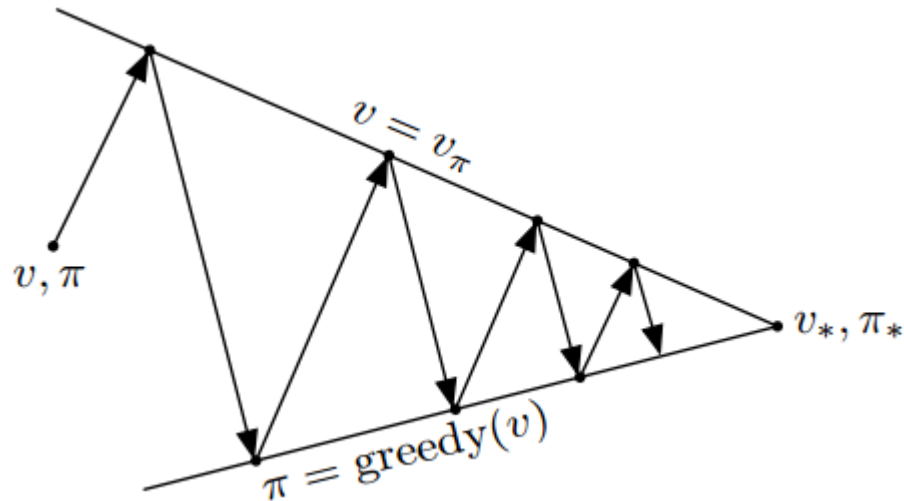
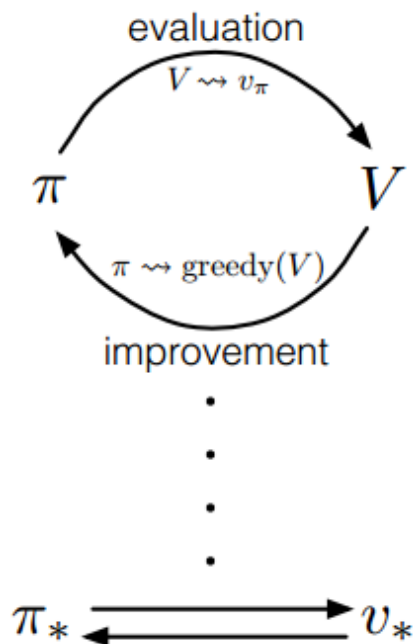
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that

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

广义策略迭代

- 迭代进行两个阶段：
 - 策略估值：让新的估值和当前的策略保持一致
 - 策略提升：根据当前估值，得到相应的贪心策略



动态规划方法小结

- 动态规划方法只不过是把贝尔曼方程转变为更新规则
 - 四个贝尔曼方程对应着四个更新规则 (v_π , v_* , q_π , q_*)
- 动态规划方法是一种“自举”(bootstrapping)方法
- 优势：
 - 动态规划方法计算效率高
- 缺点：
 - 动态规划方法需要知道关于环境的完整模型

大纲

■ 策略学习

- 动态规划方法
- 蒙特卡洛方法
- 时序差分方法
- 参数近似方法

蒙特卡洛方法的动机

- 动态规划方法需要知道关于环境的完整模型
- 大部分情况下没有关于环境的完整模型，或模型过于复杂
- 蒙特卡洛方法
 - 从真实或者模拟的经验(experience)中计算状态（行动）估值函数
 - 不需要关于环境的完整模型

蒙特卡洛方法

■ 状态估值

- 从某个状态 s 出发，使用当前策略 π 通过蒙特卡洛模拟的方式生成多个episode，使用这些episode的平均收益（return）近似状态估值函数 $v_\pi(s)$

First-visit MC prediction, for estimating $V \approx v_\pi$

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π

For each state s appearing in the episode:

$G \leftarrow$ the return that follows the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

蒙特卡洛模拟
求均值

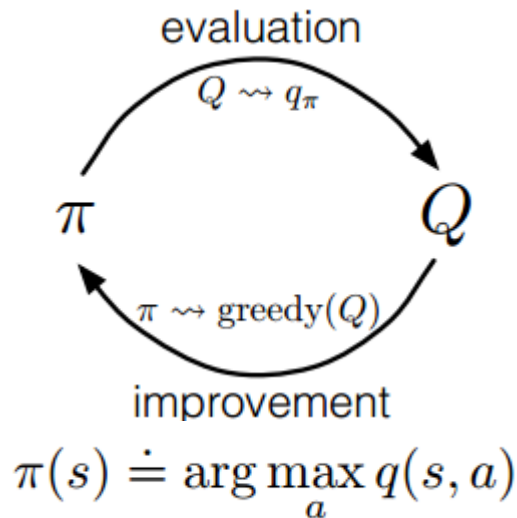
收敛速度大约为： $1/\sqrt{n}$ （ n 表示蒙特卡洛模拟次数）

蒙特卡洛方法的优势

- 直接根据真实经验或模拟经验计算状态估值函数
- 不同状态的估值在计算时是独立的
 - 不依赖于“自举”方法

基于蒙特卡洛方法的策略迭代

- 在完整的环境模型未知时，仅有状态估值 $v_\pi(s)$ 无法得出策略 π
- 大多数情况下，直接使用蒙特卡洛方法计算行为估值函数 $q_\pi(s, a)$ ，进而采用贪心方法得到策略 π



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

蒙特卡洛方法面临的问题

- 部分“状态-行为”在蒙特卡洛模拟中可能不出现
- 解决方法
 - Exploring Start: 每个“状态-行为”对都以一定的概率作为蒙特卡洛模拟的起始点

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

Repeat forever:

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0

Generate an episode starting from S_0, A_0 , following π

For each pair s, a appearing in the episode:

$G \leftarrow$ the return that follows the first occurrence of s, a

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

For each s in the episode:

$\pi(s) \leftarrow \arg\max_a Q(s, a)$

Exploring Start

摒弃 Exploring Start

■ 两种方法

□ On-policy方法：

- 在每个状态 s 下保持对所有行为 $\mathcal{A}(s)$ 进行探索的可能性，譬如采用 ε 贪心策略
 - 以 $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$ 选择贪心行为，以 $\frac{\varepsilon}{|\mathcal{A}(s)|}$ 的概率选择任意非贪心行为
- 缺点
 - 最终得到的最优策略仅仅是 ε 最优策略（ ε -soft policy）

□ Off-policy方法

- 使用两个策略：目标策略 π 和行为策略 b
- 目标策略是待优化的策略，以贪心方式进行
- 行为策略保证在每个状态下对所有行为进行探索的可能性

On-policy蒙特卡洛方法

■ Soft-policy:

On-policy first-visit MC control (for ε -soft policies), estimates $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

$\pi(a|s) \leftarrow$ an arbitrary ε -soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow$ the return that follows the first occurrence of s, a

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$

(with ties broken arbitrarily)

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

On-policy exploration

On-policy方式部分放弃了最优性来换取对策略的探索

Off-policy蒙特卡洛方法

- 保证寻找最优策略，在优化目标策略 π 时，用行为策略 b 进行策略探索
 - 行为策略需要确保对行为的覆盖度（coverage）
 - 对于所有 $\pi(a|s) > 0$ ，需要有 $b(a|s) > 0$
 - 缺点：方差比较大，收敛慢
 - 行为策略的选择影响收敛速度和方差
 - 通过重要度采样方式对蒙特卡洛模拟结果进行加权

样本重要度

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

一般的重要度采样

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

加权的重要度采样

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

蒙特卡洛方法小结

- 从经验中学习，不需要知道完整的环境模型
- 适用于环境模型未知或者环境模型复杂的情形
- 收敛性由大数定理保证
- On-policy和off-policy两种实现
 - 平衡exploitation和exploration

课间休息

大纲

■ 策略学习

- 动态规划方法
- 蒙特卡洛方法
- 时序差分方法
- 参数近似方法

时序差分方法

不好用

- 非平稳情形下的蒙特卡洛方法（恒定步长）

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$

蒙特卡洛方法

G_t 表示第 t 轮蒙特卡洛模拟的收益

时序差分

$V(S_t)$ 表示第 t 轮对状态 S_t 的估值

- 时序差分方法（Temporal Difference: TD）

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

不需要根据完整的episode计算 G_t ，只需要模拟几步（这里是1步）之后更新状态估值

时序差分方法的实现

■ 一步时序差分方法TD(0)的实现

Tabular TD(0) for estimating v_π

```
Input: the policy  $\pi$  to be evaluated
Initialize  $V(s)$  arbitrarily (e.g.,  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ )
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
     $A \leftarrow$  action given by  $\pi$  for  $S$ 
    Take action  $A$ , observe  $R, S'$ 
     $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
```

一种误差更新（参考：多臂赌博机）

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

注意：

这里的更新规则基于“**样本**”
的更新，而不是“**期望**”更新
(参考：动态规划方法)

时序差分方法分析

- 时序差分方法是强化学习中最核心的策略学习方法
- TD和蒙特卡洛方法的联系和区别
 - 联系：都是从经验中学习
 - 非平稳情形下的蒙特卡洛方法是TD的特例
 - 区别：蒙特卡洛方法需要episode完整的信息，TD只需要episode的部分信息
- TD和动态规划方法的联系和区别
 - 联系：TD和动态规划方法都采用自举的方法
 - 区别：动态规划方法依赖于完整的环境模型进行估计，TD依赖于经验进行估计

时序差分方法分析

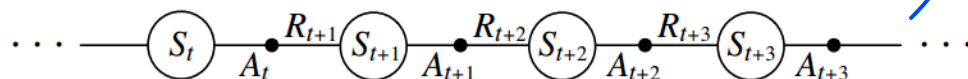
- 时序差分方法是 “learn a guess from a guess”
- 时序差分方法收敛吗？
 - 收敛
- 时序差分方法和蒙特卡洛方法收敛速度哪个快？

取决于问题

基于时序差分方法的On-policy策略优化

■ SARSA: 估计行为估值函数

□ TD所需的episode片段

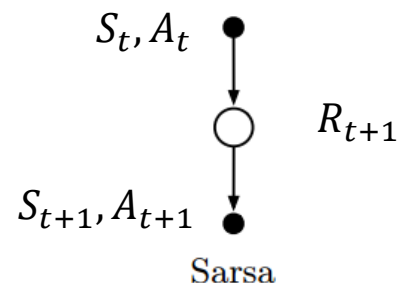


□ 行为估值函数的更新规则

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

□ 五元组

$$(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$$



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Repeat (for each step of episode):

Take action A , observe R, S'

Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$

until S is terminal

基于时序差分方法的Off-policy策略优化

■ Q-learning (Watkins, 1989)

□ 更新规则

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

□ Q-learning的实现

行为策略是 ϵ 贪心策略

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

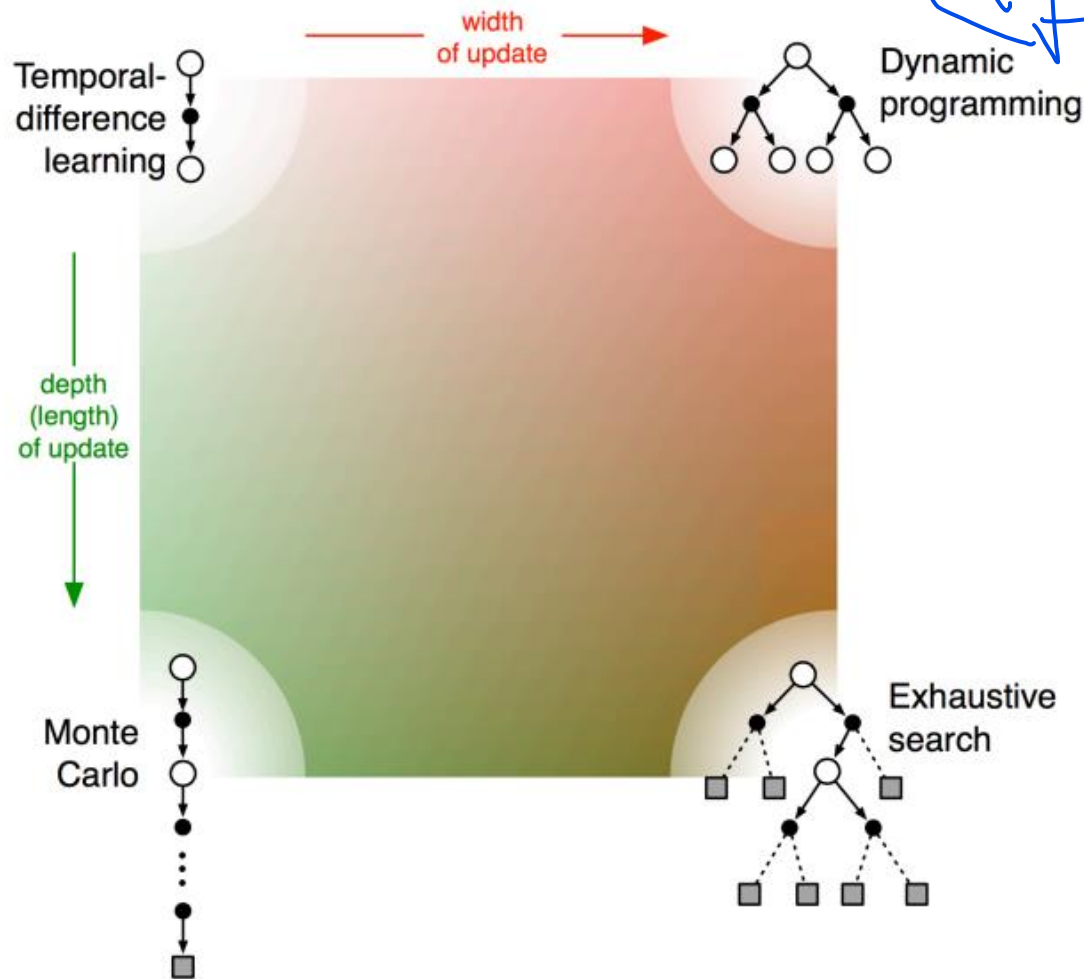
until S is terminal

目标策略是贪心策略

时序差分方法小结

- 一种在线的从经验中进行策略学习的方法
- 一般直接学习行为估值函数完成策略学习
- 适用于状态和行为空间比较小的问题

动态规划/蒙特卡洛/时序差分对比



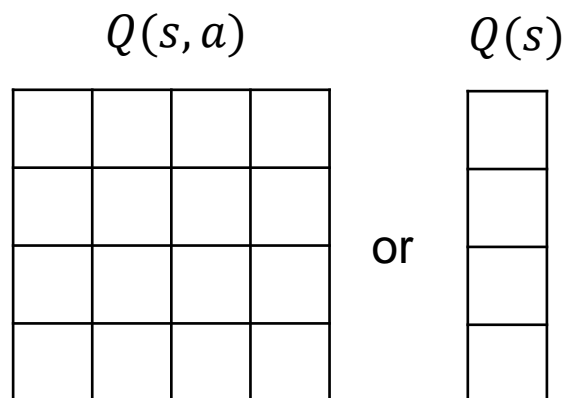
大纲

■ 策略学习

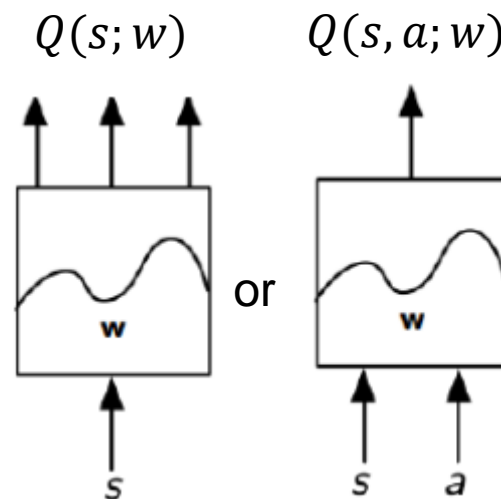
- 动态规划方法
- 蒙特卡洛方法
- 时序差分方法
- 参数近似方法

参数化近似方法

- 当状态空间或行为空间比较大时，采用表格方式存放状态估值或行为估值不可行
 - 需要对状态估值或行为估值进行参数化近似 (parametrized approximation)



表格化表示



参数化的函数式表示
(w 是参数)

参数化近似方法

■ 参数化的函数形式

□ 广义线性模型

- 参数是特征的权重

□ 决策树

- 参数是叶子节点的取值，和树节点分裂的阈值

□ 神经网络

- 每层的连接权重

■ 一般要求

- 参数个数要小于状态（或状态-行为）的个数

参数化近似方法的参数学习

■ 训练样本形式

□ 动态规划方法

$$s \mapsto \mathbb{E}_{\pi}[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) \mid S_t = s]$$

□ 蒙特卡洛方法

$$S_t \mapsto G_t$$

□ 时序差分方法

$$S_t \mapsto R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t)$$

■ 哪些监督学习方法适合于强化学习的参数学习呢？

- 能够进行在线训练，应对目标函数的不平稳或训练样本标注的不平稳
- 不能依赖于对训练样本的多次扫描

参数近似方法的损失函数

- 平均平方估值误差 (Mean Squared Value Error)

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_{\pi}(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

$\mu(s)$ 刻画状态 s 的重要程度（譬如状态被访问到的次数）

- ✓ 强化学习的目标是寻找最优策略
- ✓ 合适的损失函数是什么？VE损失函数？

模型训练

- 随机梯度下降 (Stochastic Gradient Descent)

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right]^2 \\ &= \mathbf{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

α 是学习步长

- 为何不完全利用梯度呢？步长 α 对收敛性的影响如何？
- 当目标值 $v_\pi(S_t)$ 观测不到、只观测到其近似值 U_t 时

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t)$$

随机梯度下降进行模型训练的例子

■ 蒙特卡洛方法下的随机梯度下降

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Initialize value-function weights \mathbf{w} as appropriate (e.g., $\mathbf{w} = \mathbf{0}$)

Repeat forever:

Generate an episode $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using π

For $t = 0, 1, \dots, T - 1$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

蒙特卡洛方法下的随机梯度，观测是 G_t

- ✓ G_t 是对 $v_\pi(S_t)$ 的无偏估计，因此具有收敛的保障
- ✓ 如果是一种“自举”方式的观测值？
 - 动态规划、时序差分

“半”随机梯度下降

- 在“自举”式观测值情况下
 - 以时序差分方法TD(0)为例，此时的观测值为

$$U_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$$

此时观测值也是关于参数 w 的函数

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Initialize value-function weights \mathbf{w} arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose $A \sim \pi(\cdot | S)$

 Take action A , observe R, S'

$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})] \nabla \hat{v}(S, \mathbf{w})$

$S \leftarrow S'$

 until S' is terminal

半随机梯度不保障收敛，在线性函数下收敛性尚好

参数近似Q

- 半梯度Sarsa：一种on-policy的TD(0)

- 目标：

$$\hat{q}(s, a, \mathbf{w}) \approx q_*(s, a)$$

- 参数学习过程

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha [U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

$$\doteq \mathbf{w}_t + \alpha [R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

策略梯度

重点讲一下

与TD学习
区别

■ 策略梯度 (policy gradient)

$$\pi(a|s, \theta) = \Pr\{A_t = a \mid S_t = s, \theta_t = \theta\}$$

■ 一般情况下目标函数 $J(\theta)$ 定义为策略的表现性能

- 采用随机梯度上升法进行优化

$$\theta_{t+1} = \theta_t + \alpha \nabla \widehat{J}(\theta_t)$$

■ Actor-Critic方法

- 同时学习参数化的状态估值函数 $v(s, w)$ 和策略 $\pi(a|s, \theta)$

案例：AlphaGo Zero

■ 参数近似

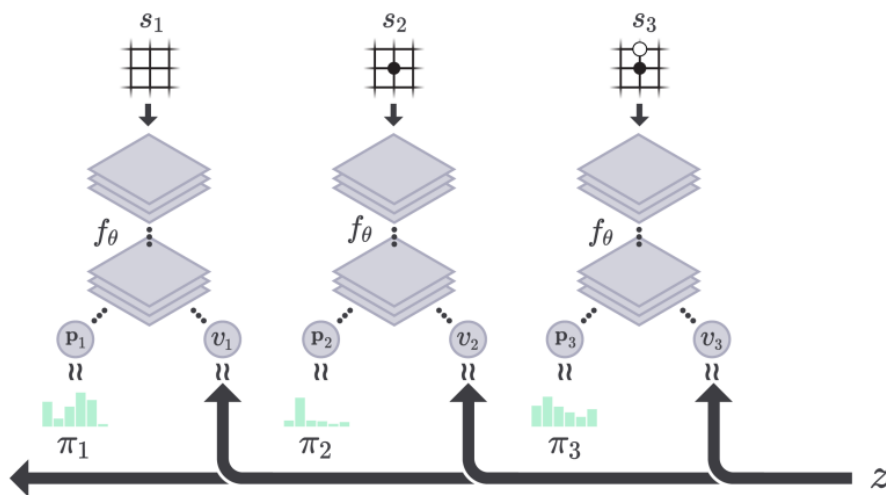
□ Policy gradient

$$(\mathbf{p}, v) = f_{\theta}(s)$$

□ 损失函数

$$l = (z - v)^2 - \boldsymbol{\pi}^{\top} \log \mathbf{p} + c \|\boldsymbol{\theta}\|^2$$

□ 学习过程

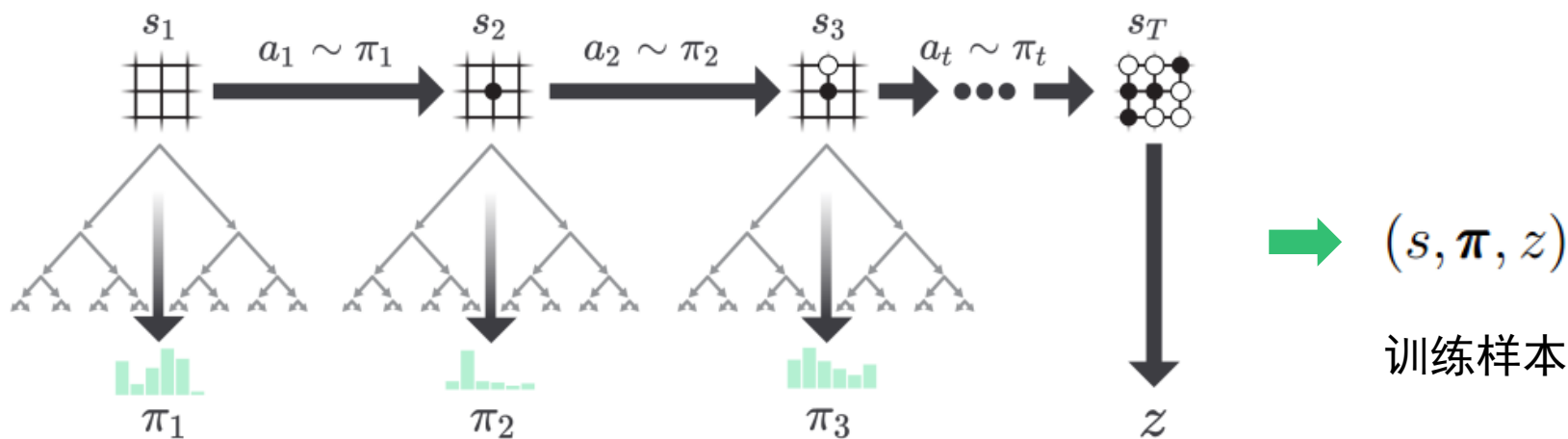


不停

案例：AlphaGo Zero

■ 蒙特卡洛方法获得模拟经验

□ Self-play



□ 用蒙特卡洛树搜索（MCTS）进行策略提升

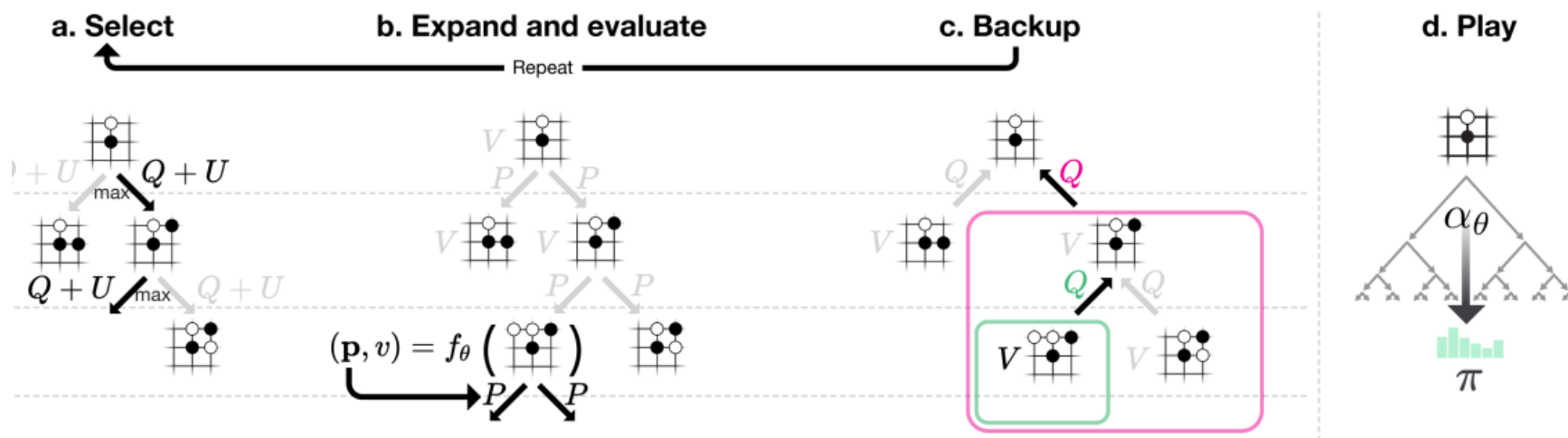
■ $p \rightarrow \pi$

□ 用蒙特卡洛模拟的结果进行策略估值

■ $v \rightarrow z$

案例：AlphaGo Zero

■ 蒙特卡洛树搜索



$$Q(s, a) + U(s, a) = Q(s, a) + \frac{P(s, a)}{1 + N(s, a)}$$

下课