

强化学习

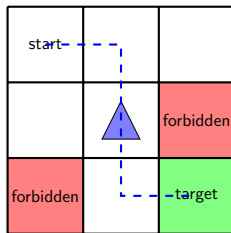
何家豪

hejiahao@ruc.edu.cn
中国人民大学信息学院

2025 年 4 月 11 日

目录

1. 基本概念
2. 贝尔曼公式
3. 最优策略和贝尔曼最优公式
4. 时序差分方法 (Q-Learning)
5. 值函数近似方法 (Deep Q-Learning)



- 格子：可通过/禁止通过/目标，边界

任务：

- 给定一个起始点，找出一条“好”的路径到终点。
- 怎么定义“好”：尽量避免 forbidden 的格子，少走重复的路，尽量不要撞到边界。

状态

状态：智能体在环境中被描述的状态。在网格世界中，智能体的位置就是他的状态。
网格世界中一共有 9 个可能的位置，所以智能体也就有 9 个可能的状态： s_1, s_2, \dots, s_9 。

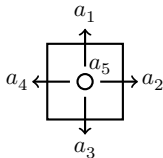
s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

状态空间：所有状态的集合 $\mathcal{S} = \{s_i\}_{i=1}^9$

动作

动作：对于每个状态，有 5 个可能的动作： a_1, a_2, \dots, a_5

- a_1 ：向上移动
- a_2 ：向右移动
- a_3 ：向下移动
- a_4 ：向左移动
- a_5 ：停留



s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

状态的动作空间：智能体在一个状态可以做的所有动作的集合。 $\mathcal{A}(s_i) = \{a_k\}_{k=1}^5$

问题：不同状态的动作空间一样吗？

状态转移

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

先关注 forbidden 的格子：如果在状态 s_5 选择了行动 a_2 会发生什么？

- 第一种情况：forbidden 的格子可以进入，但是会有惩罚。那么，

$$s_5 \xrightarrow{a_2} s_6$$

- 第二种情况：forbidden 的格子不能进入，那么，

$$s_5 \xrightarrow{a_2} s_5$$

之后我们基本上考虑的是第一种情况。

状态转移

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

我们可以用一个表格去形容状态转移。(只能表述确定性的情况)

	a_1	a_2	a_3	a_4	a_5
s_1	s_1	s_2	s_4	s_1	s_1
s_2	s_2	s_3	s_5	s_1	s_2
s_3	s_3	s_3	s_6	s_2	s_3
s_4	s_1	s_5	s_7	s_4	s_4
s_5	s_6	s_8	s_4	s_2	s_5
s_6	s_3	s_6	s_9	s_5	s_6
s_7	s_4	s_8	s_7	s_7	s_7
s_8	s_5	s_9	s_8	s_7	s_8
s_9	s_6	s_9	s_9	s_8	s_9

状态转移

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

状态转移概率：使用概率来描述状态转移。

- 如果我们在状态 s_1 选择动作 a_2 ，那么我们的下一个状态是 s_2 。
- 数学：

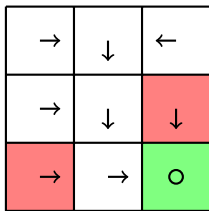
$$P(s_2|s_1, a_2) = 1$$

$$P(s_i|s_1, a_2) = 0, \quad \forall i \neq 2$$

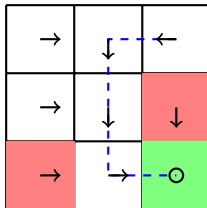
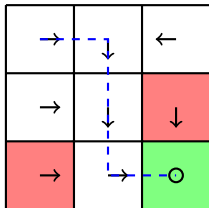
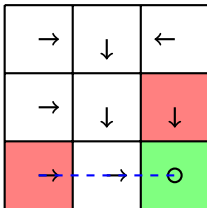
这里是**确定性**的情况，还有**不确定性**的情况（比如说会刮风）。

策略

策略告诉智能体在一个状态下应该选择什么样的动作。
用箭头表示策略。



根据这个策略，我们可以从不同的出发点开始得到不同的轨迹。



策略

→	↓	←
→	↓	↓
→	→	○

数学表示：用条件概率表示策略。

对于 s_1 而言：

$$\pi(a_1|s_1) = 0$$

$$\pi(a_2|s_1) = 1$$

$$\pi(a_3|s_1) = 0$$


$$\pi(a_4|s_1) = 0$$

$$\pi(a_5|s_1) = 0$$

这是一个确定性策略。

策略

随机策略：

	↓	←
→	↓	↓
→	→	○

在这个策略中，对于 s_1 而言：

$$\pi(a_1|s_1) = 0$$

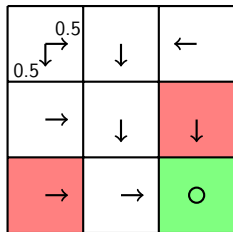
$$\pi(a_2|s_1) = 0.5$$

$$\pi(a_3|s_1) = 0.5$$

$$\pi(a_4|s_1) = 0$$

$$\pi(a_5|s_1) = 0$$

策略

用表格形式表示一个策略。

	a_1	a_2	a_3	a_4	a_5
s_1	0	0.5	0.5	0	0
s_2	0	0	1	0	0
...			...		
s_9	0	0	0	0	1

这样既可以表示确定性策略，也可以表示随机策略。

奖励

奖励是一个智能体做出动作之后得到的一个实数。

- 一个正数的奖励值鼓励智能体做这样的动作；
- 一个负数的奖励值惩罚智能体做这样的动作。

问题：

- 可以让正数代表惩罚，负数代表鼓励吗？
 - 可以。
 - 在这种情况下，奖励被称作代价。
- 奖励值为 0 会怎么样？
 - 只有相对奖励值才有意义，绝对的奖励值没有意义。
 - $r = \{+1, -1\}$ 变成 $r = \{+2, 0\}$ 并不会对策略有任何影响。

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

在网格世界中，我们可以这样设置奖励：

- 如果智能体撞墙了，那么奖励值 $r_{\text{bound}} = -1$ ；
- 如果智能体试图进入一个禁止通过的区域，那么奖励值 $r_{\text{forbidden}} = -1$
- 如果智能体到达终点，那么奖励值 $r_{\text{target}} = +1$
- 其他情况奖励值 $r = 0$

奖励值的存在可以让我们人类引导智能体按照我们想要的方式行动。

比如上述奖励的设置可以让智能体尽量不要撞墙或者试图进入禁止通过的区域。

奖励

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

用表格形式表示奖励机制。

	a_1	a_2	a_3	a_4	a_5
s_1	r_{bound}	0	0	r_{bound}	0
s_2	r_{bound}	0	0	0	0
...			...		
s_9	$r_{\text{forbidden}}$	r_{bound}	r_{bound}	0	r_{target}

只能用来表示确定性的奖励机制。

奖励

s_1	s_2	s_3
s_4	s_5	s_6
s_7	s_8	s_9

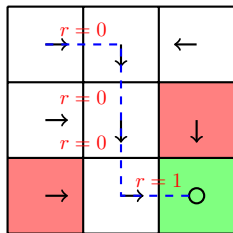
数学形式：条件概率

- 在 s_1 我们选择动作 a_1 奖励会是-1。
- $p(r = -1 | s_1, a_1) = 1$ 且 $p(r \neq -1 | s_1, a_1) = 0$

值得注意的是：

在一个确定的状态执行一个确定的动作，得到的奖励并不是确定的。比如努力学习一定会得到奖励，但是具体多少是不一定的。

轨迹和回报



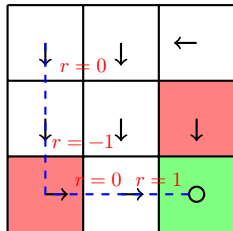
轨迹是一个状态-动作-奖励链。

$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9$$

回报是这一条轨迹上所有的奖励之和。

$$\text{return} = 0 + 0 + 0 + 1 = 1$$

轨迹和回报



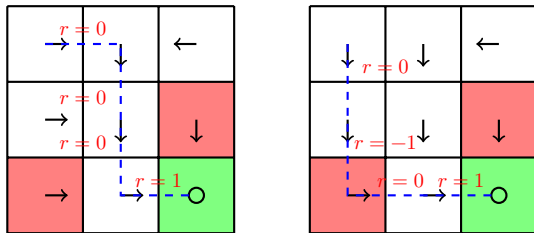
不同的策略会有不同的轨迹

$$s_1 \xrightarrow[r=0]{a_3} s_4 \xrightarrow[r=-1]{a_3} s_7 \xrightarrow[r=0]{a_2} s_8 \xrightarrow[r=1]{a_2} s_9$$

回报是这一条轨迹上所有的奖励之和。

$$\text{return} = 0 + (-1) + 0 + 1 = 0$$

轨迹和回报

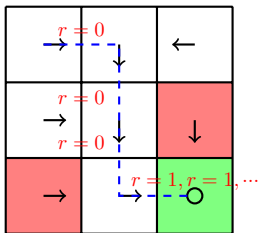


哪个策略更好一点？

- 第一个更好，因为它没有经过禁止通过的区域；
- 第一个更好，因为它的**回报**更高。

回报可以衡量一个策略的好坏。

折扣回报



轨迹可能是无限的：

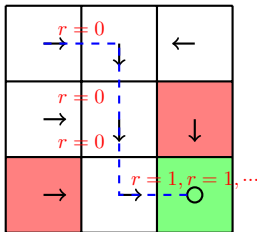
$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9 \xrightarrow[r=1]{a_5} s_9 \xrightarrow[r=1]{a_5} s_9 \dots$$

回报是：

$$\text{return} = 0 + 0 + 0 + 1 + 1 + \dots$$

这样的回报是发散的，是属于一种无效的定义。

折扣回报



需要引入一种折扣系数 $\gamma \in (0, 1)$:

$$\begin{aligned}\text{折扣回报} &= 0 + \gamma 0 + \gamma^2 0 + \gamma^3 1 + \gamma^4 1 + \gamma^5 1 + \dots \\ &= \gamma^3 (1 + \gamma + \gamma^2 + \dots) = \gamma^3 \frac{1}{1 - \gamma}\end{aligned}$$

γ 的作用: 1) 求和会变成一个有限的数; 2) 可以用来平衡短期收益和长期收益。

- 如果 γ 更接近 0, 那么折扣回报会更依赖于短期收益;
- 如果 γ 更接近 1, 那么折扣回报会更依赖于长期收益。

马尔可夫决策过程 (MDP)

MDP 的关键要素:

- 集合:
 - 状态: 状态集合 \mathcal{S}
 - 动作: 动作集合 $\mathcal{A}(s)$ 是与状态 $s \in \mathcal{S}$ 关联的
 - 奖励: 奖励集合 $\mathcal{R}(s, a)$
- 概率分布 (或被称为系统模型):
 - 状态转移概率: 在状态 s , 采取动作 a , 转移到状态 s' 的概率 $p(s'|s, a)$
 - 奖励概率: 在状态 s , 采取动作 a , 得到奖励 r 的概率 $p(r|s, a)$
- 策略: 在状态 s 采取动作 a 的概率 $\pi(a|s)$
- 马尔可夫性质: 无记忆性质

$$p(s_{t+1}|a_t, s_t, \dots, a_0, s_0) = p(s_{t+1}|a_t, s_t)$$

$$p(r_{t+1}|a_t, s_t, \dots, a_0, s_0) = p(r_{t+1}|a_t, s_t)$$

所有本节提到的概念都可以放入马尔可夫决策过程中理解。

本节通过网格世界的例子，阐述了以下概念：

- 状态
- 动作
- 状态转移，状态转移概率 $p(s'|s, a)$
- 奖励，奖励概率 $p(r|s, a)$
- 轨迹，回报，折扣回报
- 马尔可夫决策过程（MDP）。

1. 基本概念
2. 贝尔曼公式
3. 最优策略和贝尔曼最优公式
4. 时序差分方法 (Q-Learning)
5. 值函数近似方法 (Deep Q-Learning)

标注说明

考虑以下单步过程：

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1}$$

- $t, t+1$ ：离散时间序列
- S_t ：在时刻 t 的状态
- A_t ：在时刻 t 的动作
- R_{t+1} ：执行完 A_t 之后的奖励
- S_{t+1} ：执行完 A_t 之后的状态

注意 S_t, A_t, R_{t+1} 都是随机变量。

- $A_t \sim \pi(A_t | S_t = s)$
- $R_{t+1} \sim p(R_{t+1} | S_t = s, A_t = a)$
- $S_{t+1} \sim p(S_{t+1} | S_t = s, A_t = a)$

假设所有的上述概率分布（系统模型）都是已知的。

考虑以下轨迹：

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1} \xrightarrow{A_{t+1}} R_{t+2}, S_{t+2} \xrightarrow{A_{t+2}} R_{t+3}, \dots$$

其折扣回报为：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

- $\gamma \in (0, 1)$ ：折扣系数
- G_t ：也是一个随机变量，因为 R_{t+1}, R_{t+2}, \dots 都是随机变量

状态价值

G_t 的数学期望值被称为状态价值。

$$v_{\pi}(s) = \mathbb{E}[G_t | S_t = s]$$

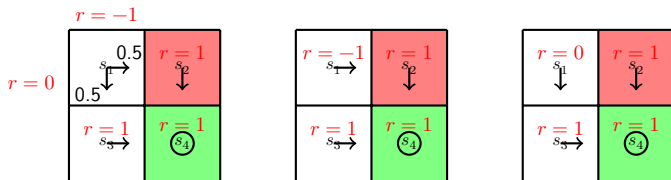
- 它是一个 s 的函数。它是当状态处于 s 时的条件期望。
- 它是一个 π 的函数。对于不同的策略，状态价值是不同的。

问题：状态价值和回报的关系是什么？

答：状态价值是从一个状态出发得到的回报的期望值。

状态价值

从 s_1 出发下面哪一个策略最好?



$$v_{\pi_1}(s_1) = 0.5 \left(-1 + \frac{\gamma}{1-\gamma} \right) + 0.5 \left(\frac{\gamma}{1-\gamma} \right) = -0.5 + \frac{\gamma}{1-\gamma}$$

$$v_{\pi_2}(s_1) = -1 + \gamma 1 + \gamma^2 1 + \dots = -1 + \frac{\gamma}{1-\gamma}$$

$$v_{\pi_3}(s_1) = 0 + \gamma 1 + \gamma^2 1 + \dots = \frac{\gamma}{1-\gamma}$$

贝尔曼公式

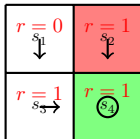
如何计算状态价值 $v_\pi(s)$?

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma \mathbb{E}[G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_r p(r|s, a)r + \gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)v_\pi(s') \\ &= \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s') \right], \quad \forall s \in \mathcal{S} \end{aligned}$$

- 以上方程被称作贝尔曼公式。它描述了不同状态价值之间的关系。
- 它由两项组成：一项是即时奖励，一项是长期奖励。
- 贝尔曼公式实际上是一组方程，每一个状态都有它对应的贝尔曼公式。

贝尔曼公式

考虑以下策略：



贝尔曼公式：

$$v_{\pi}(s) = \sum_a \pi(a|s) \left[\sum_r p(r|s, a) r + \gamma \sum_{s'} p(s'|s, a) v_{\pi}(s') \right], \quad \forall s \in \mathcal{S}$$

s_1 的状态价值：

- $\pi(a = a_3|s_1) = 1$ 且 $\pi(a \neq a_3|s_1) = 0$
- $p(s' = s_3|s_1, a_3) = 1$ 且 $p(s'|s_1, a \neq a_3) = 0$
- $p(r = 0|s_1, a_3) = 1$ 且 $p(r \neq 0|s_1, a_3) = 0$
- $v_{\pi}(s_1) = 0 + \gamma v_{\pi}(s_3)$

贝尔曼公式

贝尔曼公式:

$$v_{\pi}(s) = \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi}(s') \right], \quad \forall s \in \mathcal{S}$$

同理可得 (取 $\gamma = 0.9$):

$$v_{\pi}(s_1) = 0 + \gamma v_{\pi}(s_3) = \frac{\gamma}{1 - \gamma} = 9$$

$$v_{\pi}(s_2) = 1 + \gamma v_{\pi}(s_4) = \frac{1}{1 - \gamma} = 10$$

$$v_{\pi}(s_3) = 1 + \gamma v_{\pi}(s_3) = \frac{1}{1 - \gamma} = 10$$

$$v_{\pi}(s_4) = 1 + \gamma v_{\pi}(s_4) = \frac{1}{1 - \gamma} = 10$$

贝尔曼公式的矩阵-向量形式

每次计算状态价值都要为每个状态列一个方程求解，比较麻烦。可以采用贝尔曼公式的矩阵-向量形式简化。可以先将贝尔曼公式：

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

写为：

$$v_{\pi}(s) = r_{\pi}(s) + \gamma \sum_{s'} p_{\pi}(s'|s)v_{\pi}(s')$$

其中：

$$r_{\pi}(s) = \sum_a \pi(a|s) \sum_r p(r|s, a)r, \quad p_{\pi}(s'|s) = \sum_a \pi(a|s)p(s'|s, a)$$

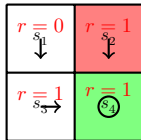
然后就可以得到矩阵-向量形式：

$$v_{\pi} = r_{\pi} + \gamma P_{\pi} v_{\pi}$$

贝尔曼公式的矩阵-向量形式

假设一共有 4 个状态, $v_\pi = r_\pi + \gamma P_\pi v_\pi$ 可以被写作:

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix} = \begin{bmatrix} r_\pi(s_1) \\ r_\pi(s_2) \\ r_\pi(s_3) \\ r_\pi(s_4) \end{bmatrix} + \gamma \begin{bmatrix} p_\pi(s_1|s_1) & p_\pi(s_2|s_1) & p_\pi(s_3|s_1) & p_\pi(s_4|s_1) \\ p_\pi(s_1|s_2) & p_\pi(s_2|s_2) & p_\pi(s_3|s_2) & p_\pi(s_4|s_2) \\ p_\pi(s_1|s_3) & p_\pi(s_2|s_3) & p_\pi(s_3|s_3) & p_\pi(s_4|s_3) \\ p_\pi(s_1|s_4) & p_\pi(s_2|s_4) & p_\pi(s_3|s_4) & p_\pi(s_4|s_4) \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}$$



$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \gamma \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}$$

从状态价值到动作价值：

- 状态价值：智能体从一个状态出发得到的回报的期望值
- 动作价值：智能体从一个状态出发执行了一个动作之后得到的回报的期望值

动作价值可以让我们知道在一个状态下采取哪个动作更好。

动作价值

定义：

$$q_{\pi}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

- q 是一个“状态-动作”对的函数。
- q 是一个 π 的函数。对于不同的策略，动作价值是不同的。

它服从条件期望的公式：

$$\mathbb{E}[G_t | S_t = s] = \sum_a \mathbb{E}[G_t | S_t = s, A_t = a] \pi(a | s) \quad (1)$$

所以：

$$v_{\pi}(s) = \sum_a \pi(a | s) q_{\pi}(s, a) \quad (2)$$

动作价值

贝尔曼公式：

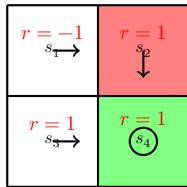
$$v_{\pi}(s) = \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi}(s') \right] \quad (3)$$

所以可以发现：

$$q_{\pi}(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi}(s') \quad (4)$$

- (2) 式表达了如何用动作价值计算状态价值
- (4) 式表达了如何用状态价值计算动作价值

动作价值



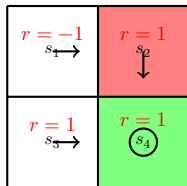
写出 s_1 的动作价值:

$$q_{\pi}(s_1, a_2) = -1 + \gamma v_{\pi}(s_2)$$

问题:

- $q_{\pi}(s_1, a_1), q_{\pi}(s_1, a_3), q_{\pi}(s_1, a_4), q_{\pi}(s_1, a_5)$ 分别是多少?

动作价值



- 动作价值比较关键的原因是我们最关心的其实是要做什么动作。
- 我们可以先把所有状态价值都算出来，然后再算所有的动作价值
- 即使没有模型，我们也可以估计动作价值。

关键概念和结论：

- 状态价值： $v_{\pi}(s) = \mathbb{E}[G_t | S_t = s]$
- 动作价值： $q_{\pi}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$
- 贝尔曼公式：

$$\begin{aligned} v_{\pi}(s) &= \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi}(s') \right] \\ &= \sum_a \pi(a|s) q_{\pi}(s, a) \end{aligned}$$

1. 基本概念
2. 贝尔曼公式
3. 最优策略和贝尔曼最优公式
4. 时序差分方法 (Q-Learning)
5. 值函数近似方法 (Deep Q-Learning)

最优策略

状态价值可以用来评估一个策略的优劣。如果：

$$v_{\pi_1}(s) \geq v_{\pi_2}(s), \forall s \in \mathcal{S}$$

则称策略 π_1 优于策略 π_2 。

定义

若一个策略 π^* 对于所有的 $s \in \mathcal{S}$ 和所有其他的策略 π 都满足 $v_{\pi^*}(s) \geq v_{\pi}(s)$ ，则称 π^* 为最优策略。

- 最优策略是否一定存在？
- 最优策略是否唯一？
- 最优策略是一个确定性策略还是一个随机策略？
- 如何得到最优策略？

使用**贝尔曼最优公式**求解。

贝尔曼最优公式

贝尔曼最优公式：

$$\begin{aligned} v(s) &= \max_{\pi} \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v(s') \right], \quad \forall s \in \mathcal{S} \\ &= \max_{\pi} \sum_a \pi(a|s) q(s, a), \quad \forall s \in \mathcal{S} \end{aligned}$$

- $p(r|s, a), p(s'|s, a), r, \gamma$ 都是已知的
- $v(s), v(s')$ 是未知的
- $\pi(s)$ 是已知还是未知的呢？

贝尔曼最优公式

贝尔曼最优公式：

$$v(s) = \max_{\pi} \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v(s') \right], \quad \forall s \in \mathcal{S}$$

例

根据下述方程求解 $x, a \in \mathbb{R}$ ：

$$x = \max_a (2x - 1 - 2a^2)$$

先看式子的右半部分，无论 x 的取值如何， $\max_a (2x - 1 - 2a^2)$ 的最大值都在 $a = 0$ 时取得，此时式子的右半部分为 $2x - 1$ 。然后再求解 $x = 2x - 1$ 解得 $x = 1$ 。综上： $x = 1, a = 0$ 时原方程的解。

贝尔曼最优公式

贝尔曼最优公式：

$$\begin{aligned}v(s) &= \max_{\pi} \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v(s') \right], \quad \forall s \in \mathcal{S} \\&= \max_{\pi} \sum_a \pi(a|s) q(s, a) \\&= \max_{a \in \mathcal{A}(s)} q(s, a)\end{aligned}$$

当达到最优时，只需要让每次智能体都选择动作价值最大的行动即可，即：

$$\pi(a|s) = \begin{cases} 1 & a = a^* \\ 0 & a \neq a^* \end{cases}$$

其中：

$$a^* = \arg \max_a q(s, a)$$

贝尔曼最优公式

确定了 π 之后，贝尔曼最优公式可以简化为：

$$v = f(v)$$

如何求解？

定理 (压缩映射定理)

对于任何具有 $x = f(x)$ 形式的方程，如果 f 是一个压缩映射，那么：

- 解存在性：一定存在一个 x^* 满足 $f(x^*) = x^*$
- 解唯一性： x^* 是唯一的
- 求解方式（值迭代法）：考虑一个数列 $\{x_k\}$ ，其中 $x_{k+1} = f(x_k)$ 。当 $k \rightarrow \infty$ 时 $x_k \rightarrow x^*$ 。

贝尔曼最优公式是一个压缩映射（证明略）。

- 例： $x = 0.5x$ ， $\{x_k\} = \{10, 5, 2.5, \dots\}$

贝尔曼最优公式

所以我们可以按照值迭代法的求解方式求出 v^* ，但它一定就是最高的状态价值吗？

定理 (贝尔曼最优定理)

假设 v^* 是 $v = \max_{\pi}(r_{\pi} + P_{\pi}v)$ 的唯一解，那么对于任何其他给定策略 π 的满足 $v_{\pi} = r_{\pi} + \gamma P_{\pi}v_{\pi}$ 的 v_{π} 而言都有：

$$v^* \geq v_{\pi} \quad \forall \pi$$

也就是说，对于贝尔曼最优公式这一个特殊的方程，**它的唯一解就是最优解！**

贝尔曼最优公式

那么当状态价值取到最优的 v^* 时，策略 π^* 是什么样的呢？

$$\pi^*(s) = \arg \max_{\pi} \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v^*(s') \right]$$

定理 (贝尔曼最优定理)

对于任何 $s \in \mathcal{S}$ ，下列确定性策略都是最优策略：

$$\pi^*(a|s) = \begin{cases} 1 & a = a^*(s) \\ 0 & a \neq a^*(s) \end{cases}$$

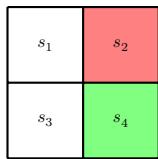
其中：

$$a^*(s) = \arg \max_a q^*(s, a), \quad q^*(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v^*(s')$$

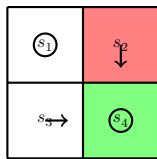
也就是说，每次选择动作价值最高的动作就是最优策略！

值迭代法

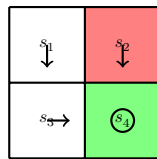
奖励设置为: $r_{\text{boundary}} = r_{\text{forbidden}} = -1, r_{\text{target}} = 1$ 。折扣系数为 $\gamma = 0.9$



(a)



(b)



(c)

$q(s, a)$:

动作价值	a_1	a_2	a_3	a_4	a_5
s_1	$-1 + \gamma v(s_1)$	$-1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$	$-1 + \gamma v(s_1)$	$0 + \gamma v(s_1)$
s_2	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_2)$	$1 + \gamma v(s_4)$	$0 + \gamma v(s_1)$	$-1 + \gamma v(s_2)$
s_3	$0 + \gamma v(s_1)$	$1 + \gamma v(s_4)$	$-1 + \gamma v(s_3)$	$-1 + \gamma v(s_3)$	$0 + \gamma v(s_3)$
s_4	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_4)$	$-1 + \gamma v(s_4)$	$0 + \gamma v(s_3)$	$1 + \gamma v(s_4)$

值迭代法

- $k = 0$: 令 $v_0(s_1) = v_0(s_2) = v_0(s_3) = v_0(s_4) = 0$

动作价值	a_1	a_2	a_3	a_4	a_5
s_1	-1	-1	0	-1	0
s_2	-1	-1	1	0	-1
s_3	0	1	-1	-1	0
s_4	-1	-1	-1	0	1

第一步：策略更新

$$\pi_1(a_5|s_1) = 1, \pi_1(a_3|s_2) = 1, \pi_1(a_2|s_3) = 1, \pi_1(a_5|s_4) = 1$$

第二步：值更新

$$v_1(s_1) = 0, v_1(s_2) = 1, v_1(s_3) = 1, v_1(s_4) = 1$$

如图 (b) 所示。

值迭代法

- $k = 1$: 因为 $v_1(s_1) = 0, v_1(s_2) = 1, v_1(s_3) = 1, v_1(s_4) = 1$, 所以:

动作价值	a_1	a_2	a_3	a_4	a_5
s_1	$-1 + \gamma 0$	$-1 + \gamma 1$	$0 + \gamma 1$	$-1 + \gamma 0$	$0 + \gamma 0$
s_2	$-1 + \gamma 1$	$-1 + \gamma 1$	$1 + \gamma 1$	$0 + \gamma 0$	$-1 + \gamma 1$
s_3	$0 + \gamma 0$	$1 + \gamma 1$	$-1 + \gamma 1$	$-1 + \gamma 1$	$0 + \gamma 1$
s_4	$-1 + \gamma 1$	$-1 + \gamma 1$	$-1 + \gamma 1$	$0 + \gamma 1$	$1 + \gamma 1$

第一步：策略更新

$$\pi_1(a_3|s_1) = 1, \pi_1(a_3|s_2) = 1, \pi_1(a_2|s_3) = 1, \pi_1(a_5|s_4) = 1$$

第二步：值更新

$$v_2(s_1) = \gamma 1, v_2(s_2) = 1 + \gamma 1, v_2(s_3) = 1 + \gamma 1, v_2(s_4) = 1 + \gamma 1$$

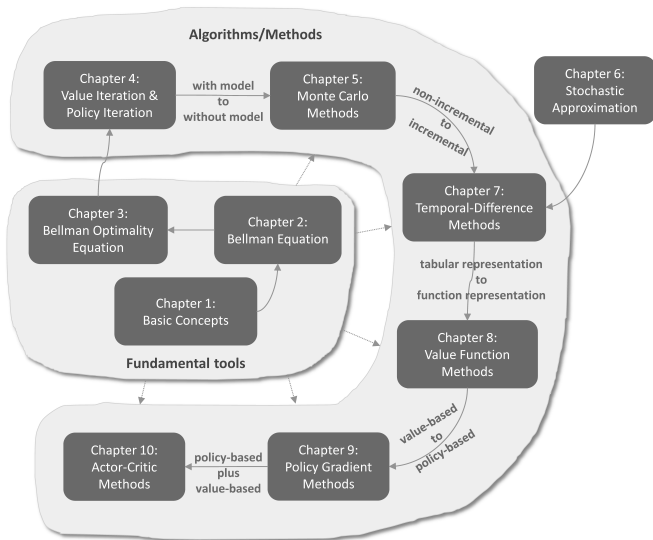
如图 (c) 所示（最优策略）

贝尔曼最优公式：

- 解存在性：根据压缩映射定理，一定存在解
- 解唯一性：根据压缩映射定理，解是唯一的
- 解最优性：根据最优策略定理，该解一定是最优解
- 求解方法：值迭代法
- 最优策略：最优状态价值对应的策略就是最优策略

1. 基本概念
2. 贝尔曼公式
3. 最优策略和贝尔曼最优公式
4. 时序差分方法 (Q-Learning)
5. 值函数近似方法 (Deep Q-Learning)

从有模型到无模型



从有模型到无模型

回顾贝尔曼公式：

$$v_{\pi}(s) = \sum_a \pi(a|s) \left[\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi}(s') \right], \quad \forall s \in \mathcal{S}$$

- 有模型： $p(r|s, a)$ 和 $p(s'|s, a)$ 已知；
- 无模型： $p(r|s, a)$ 和 $p(s'|s, a)$ 未知。

对于给定的策略，贝尔曼公式可以在有模型的情况下计算出状态价值。
无模型的情况下，如何估计状态价值——**时序差分方法**。

时序差分方法：TD

问题描述：

- 对于给定的策略 π ，估计状态价值 $\{v_\pi(s)\}_{s \in \mathcal{S}}$
- 输入：一系列采样 $(s_0, r_1, s_1, \dots, s_t, r_{t+1}, s_{t+1}, \dots)$ 或 $\{(s_t, r_{t+1}, s_{t+1})\}_t$

重要标注：

$$\begin{array}{c} v(s) \longrightarrow v_\pi(s) \\ \Downarrow \\ v(s_t) \longrightarrow v_\pi(s_t) \\ \Downarrow \\ v_t(s_t) \longrightarrow v_\pi(s_t) \end{array}$$

TD 算法：

$$v_{t+1}(s_t) = v_t(s_t) - \alpha[v_t(s_t) - (r_{t+1} + \gamma v_t(s_{t+1}))] \quad (1)$$

$$v_{t+1}(s) = v_t(s), \quad \forall s \neq s_t \quad (2)$$

其中 $t = 0, 1, 2, \dots$

此时， $v_t(s_t)$ 是 t 时刻对 $v_\pi(s_t)$ 状态价值的估计； α 是学习率。

- 在 t 时刻，只有路过的状态 s_t 的价值 $v_t(s_t)$ 会被更新，其他状态的价值与上一个时刻一致；
- 之后有时将省略 (2) 式。

时序差分方法：TD

TD 算法的性质：

- TD 算法只会估计对于给定策略的**状态价值**。
 - 它不会估计动作价值
 - 它不会寻找最优策略
- TD 算法可以很容易地拓展到可以估计动作价值和寻找最优策略的形式。
- 后续的时序差分算法都将以 TD 为基础。

问题：TD 算法在数学上解决了什么？

答：TD 算法可以在**无模型**但是**有样本**的时候求解以下贝尔曼方程：

$$v_{\pi}(s) = \mathbb{E}[R + \gamma v_{\pi}(S')|s], \quad \forall s \in \mathcal{S}$$

(还记得有模型怎么求解吗？——矩阵向量形式，值迭代法)

时序差分方法: Sarsa

- TD 算法只会估计对于给定策略的**状态价值**。
- 接下来介绍的 Sarsa 算法可以直接估计动作价值。
- Sarsa 算法也可以寻找最优策略。

时序差分方法: Sarsa

问题描述:

- 对于给定的策略 π , 估计所有的动作价值 $\{q_\pi(s, a)\}_{s \in \mathcal{S}, a \in \mathcal{A}}$
- 输入: 一系列采样 $(s_0, a_0, r_1, s_1, a_1, \dots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots)$ 或 $\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$

我们可以使用 Sarsa 算法估计动作价值:

$$q_{t+1}(s, a) = q_t(s, a) - \alpha[q_t(s, a) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))]$$
$$q_{t+1}(s, a) = q_t(s, a), \quad \forall s \neq s_t, a \neq a_t$$

其中 $t = 0, 1, 2, \dots$

- $q_t(s_t, a_t)$ 是在 t 时刻对 $q_\pi(s_t, a_t)$ 的估计。
- α 是学习率。

时序差分方法: Sarsa

- 为什么这个算法被称作 **Sarsa**? 因为这个算法的每一步都涉及到序列 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 。Sarsa 就是 state-action-reward-state-action 的简写形式。
- **Sarsa** 算法和 **TD** 算法之间是什么关系? 我们可以直接把 TD 算法中估计状态价值的 $v(s)$ 替换成估计动作价值的 $q(s, a)$ 。所以 Sarsa 算法是 TD 算法用于估计动作价值的一个拓展。
- **Sarsa** 算法在数学上解决了什么? Sarsa 算法可以在无模型但是有样本的时候求解以下贝尔曼方程:

$$q_{\pi}(s, a) = \mathbb{E}[R + \gamma q_{\pi}(S', A') | s, a], \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

时序差分方法: Sarsa

Sarsa 寻找最优策略

- 1: **for** 每一个轮次 **do**
- 2: 依据 $\pi_0(s_0)$ 从初始状态 s_0 开始选择动作状态 a_0
- 3: **if** $s_t (t = 0, 1, 2, \dots)$ 不是目标状态 **then**
- 4: 依据 (s_t, a_t) 收集样本 $(r_{t+1}, s_{t+1}, a_{t+1})$: 通过与环境交互获得 r_{t+1}, s_{t+1} ,
 根据 $\pi_t(s_{t+1})$ 获得 a_{t+1} 。
- 5: 更新 (s_t, a_t) 的动作价值:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha[q_t(s_t, a_t) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))]$$

- 6: 更新策略 $\pi_t(s_t)$:

$$\pi_{t+1}(s_t) = \arg \max_a q_{t+1}(s_t, a)$$

- 7: **end if**
- 8: **end for**

ϵ -greedy 策略

假设网格世界中从起点出发到终点有以下两种轨迹：

- 轨迹 A：很直接，容易发现，回报低（可能经过了一些禁止区域）
- 轨迹 B：需要绕来绕去，很难发现，但是回报高

但是根据上一页的算法，智能体就很容易就会陷入轨迹 A，永远探索不到轨迹 B。

ϵ -greedy 策略

$$\pi(a|s) = \begin{cases} 1 - \frac{\epsilon}{|\mathcal{A}|} (|\mathcal{A}| - 1), & a = \arg \max_a q(s, a) \\ \frac{\epsilon}{|\mathcal{A}|}, & a \neq \arg \max_a q(s, a) \end{cases}$$

- 例：在网格世界中， $\epsilon = 0.1$ ，则有 92% 的概率选择最优动作，2% 的概率随机选择其他动作中的任意一个。

时序差分方法: Sarsa

Sarsa 算法的特点:

- 状态 s_t 的策略在 $q(s_t, a_t)$ 更新之后立刻就更新了。所以它是一个增量更新的算法, 不需要收集整条轨迹才更新一次策略。
- Sarsa 算法需要使用 ϵ -greedy 策略来平衡探索和利用。

Sarsa 算法的核心:

- Sarsa 算法的核心很简单: 求解贝尔曼方程。

$$q_{\pi}(s, a) = \mathbb{E}[R + \gamma q_{\pi}(S', A') | s, a], \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

- Sarsa 算法有可能会看起来复杂, 是因为实际写成代码的时候需要考虑算法效率和更新策略。

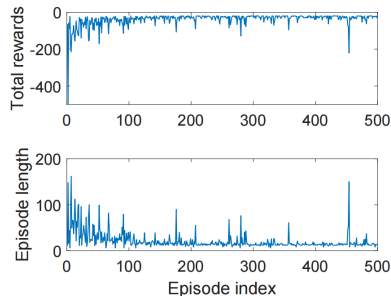
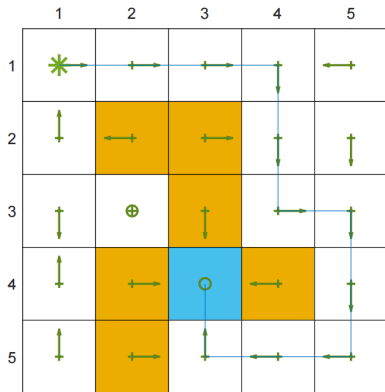
例子：

- 问题：找到从一个特定的起点走到终点的好路径。
 - 这个问题和之前问题的不同之处在于，我们并不需要找出所有状态的最优策略。
 - 每一条轨迹的起点都是左上角，终点都是 target
- $r_{\text{boundary}} = r_{\text{forbidden}} = -10, r_{\text{target}} = 0, r_{\text{other}} = -1, \gamma = 0.9, \alpha = 0.1, \epsilon = 0.1$

时序差分方法: Sarsa

结果:

- 左图是 Sarsa 找到的最优策略
 - 注意并不是所有状态都是最优策略。
- 右图是每一轮迭代的回报和轨迹长度。



时序差分方法：n 步 Sarsa

动作价值的定义：

$$q_{\pi}(s, a) = \mathbb{E}(G_t | S_t = s, A_t = a)$$

折扣回报 G_t 可以被写为：

$$\text{Sarsa} \leftarrow G_t^{(1)} = R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}),$$

$$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 q_{\pi}(S_{t+2}, A_{t+2}),$$

$$\vdots$$

$$\text{n 步 Sarsa} \leftarrow G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^n q_{\pi}(S_{t+n}, A_{t+n}),$$

$$\vdots$$

$$\text{Monte Carlo} \leftarrow G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

时序差分方法：n 步 Sarsa

Sarsa 要解的方程：

$$q_{\pi}(s, a) = \mathbb{E}[G_t^{(1)} | s, a] = \mathbb{E}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | s, a]$$

Monte Carlo 要解的方程：

$$q_{\pi}(s, a) = \mathbb{E}[G_t^{(\infty)} | s, a] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s, a]$$

n 步 Sarsa 要解的方程：

$$q_{\pi}(s, a) = \mathbb{E}[G_t^{(n)} | s, a] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n q_{\pi}(S_{t+n}, A_{t+n}) | s, a]$$

- 在 $n = 1$ 时，n 步 Sarsa 就是 Sarsa
- 在 $n \rightarrow \infty$ 时，n 步 Sarsa 就是 Monte Carlo

时序差分方法：n 步 Sarsa

- 输入：n 步 Sarsa 需要 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots, r_{t+n}, s_{t+n}, a_{t+n})$
- 在 t 时刻， $(r_{t+n}, s_{t+n}, a_{t+n})$ 还没有发生，所以我们无法在 t 时刻就立刻更新 $q(s_t, a_t)$ ，需要等到 $t+n$ 时刻才能更新 $q(s_t, a_t)$ ：

$$q_{t+n}(s_t, a_t) = q_{t+n-1}(s_t, a_t) - \alpha [q_{t+n-1}(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_{t+n-1}(s_{t+n}, a_{t+n})]]$$

- n 步 Sarsa 是 Sarsa 和 Monte Carlo 的折衷算法：
 - Sarsa 算法收敛过程波动很小，收敛结果偏差很大；
 - Monte Carlo 算法收敛过程波动很大，收敛结果偏差很小；
 - n 步 Sarsa 算法收敛过程的波动和收敛结果的偏差比较适中。

时序差分方法: Q-Learning

- 接下来我们介绍Q-Learning，它直到今天也是最常用的强化学习算法。
- Sarsa 可以对给定的策略 π 估计其相应的动作价值。它必须需要伴随着一个逐步迭代的路径来更新它的估计和策略。
- Q-Learning可以直接估计最优的动作价值和最优策略。

时序差分方法：Q-Learning

Sarsa 算法：

$$q_{t+1}(s, a) = q_t(s, a) - \alpha[q_t(s, a) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))]$$

Q-Learning 算法：

$$q_{t+1}(s, a) = q_t(s, a) - \alpha[q_t(s, a) - (r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a))]$$

Q-Learning 和 Sarsa 算法非常相似，只是它们的收敛目标不一样。

- Sarsa 算法中的收敛目标是 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$ 。
- Q-Learning 算法中的收敛目标是 $r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$ 。

Q-Learning 要解的方程：

$$q(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_a q(s_{t+1}, a) | S_t = s, A_t = a], \quad \forall s, a$$

这是一个贝尔曼**最优**方程用动作价值表达的形式。

所以 Q-Learning 可以直接估计出**最优（策略对应）的动作价值**，而不是某一特定策略的动作价值。

Q-Learning 寻找最优策略

- 1: **for** 对于收集到的每一个经验样本 $(s_t, a_t, r_{t+1}, s_{t+1})$ **do**
- 2: 更新 (s_t, a_t) 的动作价值:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha[q_t(s_t, a_t) - (r_{t+1} + \gamma \max_a q_t(s_{t+1}, a))]$$
- 3: 更新策略 $\pi_t(s_t)$ (greedy):

$$\pi_{t+1}(s_t) = \arg \max_a q_{t+1}(s_t, a)$$
- 4: 更新策略 $\pi_t(s_t)$ (ϵ -greedy):

$$\pi_{t+1}(a|s_t) = \begin{cases} 1 - \frac{\epsilon}{|\mathcal{A}|}(|\mathcal{A}| - 1), & a = \arg \max_a q_{t+1}(s_t, a) \\ \frac{\epsilon}{|\mathcal{A}|}, & a \neq \arg \max_a q_{t+1}(s_t, a) \end{cases}$$
- 5: **end for**

时序差分方法：Q-Learning

什么时候选择 greedy? 什么时候选择 ϵ -greedy?

输入的经验样本集合 $\{(s_t, a_t, r_{t+1}, s_{t+1})\}_n$ 来源于哪里?

- 依据 π_t 生成的。在 t 时刻想要更新状态 s_t 中某一个动作的动作价值，就把 s_t 输入到当前的 π_t 中，得到 a_t ，然后通过与环境交互得到 (r_{t+1}, s_{t+1}) ；—— ϵ -greedy
- 依据一个别的任意策略 π_b 生成的。在 t 时刻想要更新状态 s_t 中某一个动作的动作价值，就把 s_t 输入到一个固定的策略 π_b 中，得到 a_t ，然后通过与环境交互得到 (r_{t+1}, s_{t+1}) ；——greedy
- 随机生成的。在 t 时刻想要更新状态 (s_t, a_t) 的动作价值，就直接通过与环境交互得到 (r_{t+1}, s_{t+1}) ；——greedy

Q-Learning 可以从别的策略产生的经验中学习，这种特性叫做 off-policy。之前我们所学的 TD 和 Sarsa 都是 on-policy 的算法。

时序差分方法: Q-Learning

例子:

- 问题: 找到从所有的起点走到终点的最优路径。
- $r_{\text{boundary}} = r_{\text{forbidden}} = -1, r_{\text{target}} = 1, r_{\text{other}} = 0, \gamma = 0.9, \alpha = 0.1$

真实值:

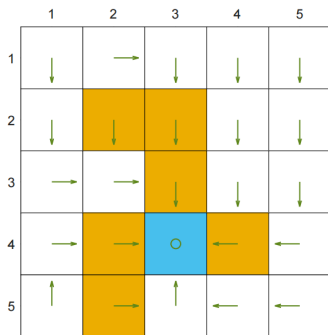


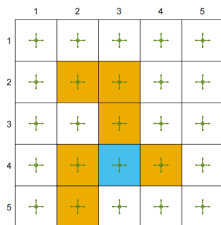
图: 最优策略

	1	2	3	4	5
1	5.8	5.6	6.2	6.5	5.8
2	6.5	7.2	8.0	7.2	6.5
3	7.2	8.0	10.0	8.0	7.2
4	8.0	10.0	10.0	10.0	8.0
5	7.2	9.0	10.0	9.0	8.1

图: 最优状态价值

时序差分方法: Q-Learning

生成策略、生成的经验样本通过 off-policy 方式的 Q-Learning 学习到的策略 (10^5 步):



图：生成策略

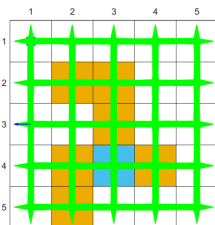


图: 生成策略的轨迹

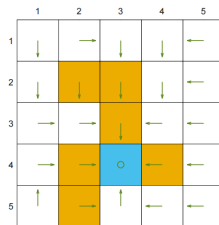


图: 学习策略

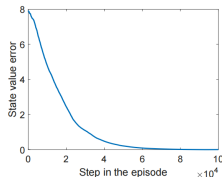


图: 状态价值误差

时序差分方法: Q-Learning

探索的重要性: 10^5 步

如果生成策略探索得不充分, 得到的经验样本就会很差。

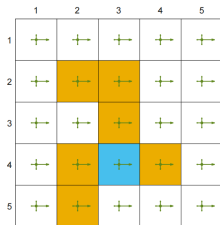


图: 生成策略 $\epsilon = 0.5$

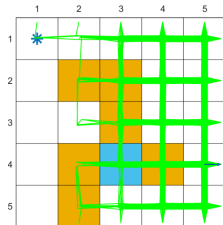


图: 生成策略的轨迹

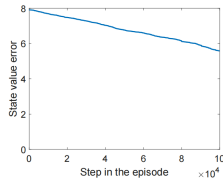


图: 状态价值误差

时序差分方法: Q-Learning

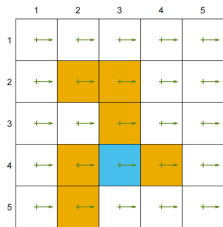


图: 生成策略 $\epsilon = 0.1$

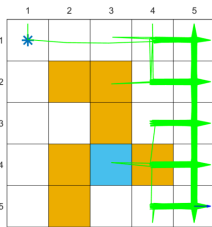


图: 生成策略的轨迹

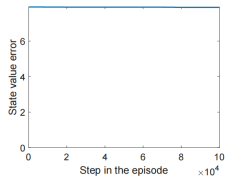


图: 状态价值误差

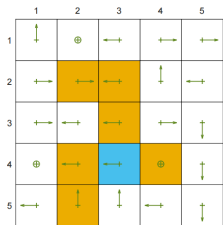


图: 生成策略 $\epsilon = 0.1$

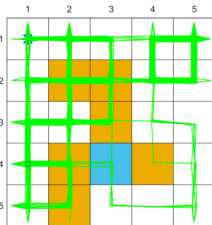


图: 生成策略的轨迹

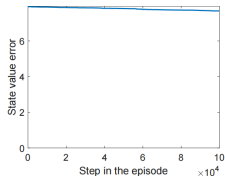


图: 状态价值误差

本节所有算法都可以归纳为以下形式：

$$q_{t+1}(s, a) = q_t(s, a) - \alpha[q_t(s, a) - q_{\text{target}}]$$

其中 q_{target} 是时序差分算法求解的目标。

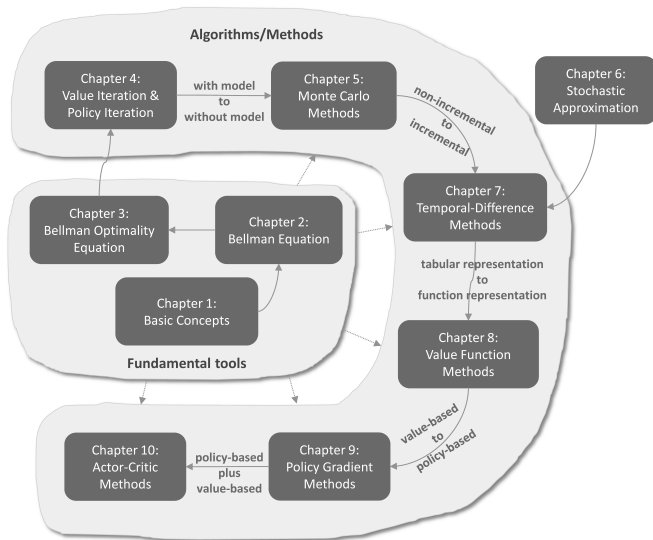
算法	q_{target}
Sarsa	$r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$
n 步 Sarsa	$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n}, a_{t+n})$
Monte Carlo	$r_{t+1} + \gamma r_{t+2} + \cdots$
Q-Learning	$r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$

所有时序差分算法都是解某种形式的贝尔曼公式的随机近似算法。

算法	求解的方程
Sarsa	BE: $q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) S_t = s, A_t = a]$
n 步 Sarsa	BE: $q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+1} + \dots + \gamma^n q_{\pi}(S_{t+n}, A_{t+n}) S_t = s, A_t = a]$
Monte Carlo	BE: $q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots S_t = s, A_t = a]$
Q-Learning	BOE: $q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_a q_{\pi}(S_{t+1}, a) S_t = s, A_t = a]$

1. 基本概念
2. 贝尔曼公式
3. 最优策略和贝尔曼最优公式
4. 时序差分方法 (Q-Learning)
5. 值函数近似方法 (Deep Q-Learning)

从表格表示到函数表示



从表格表示到函数表示

直到现在本节课涉及的所有动作价值和状态价值都是用表格表示的。

- 状态价值：

State	s_1	s_2	\cdots	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	\cdots	$v_\pi(s_n)$

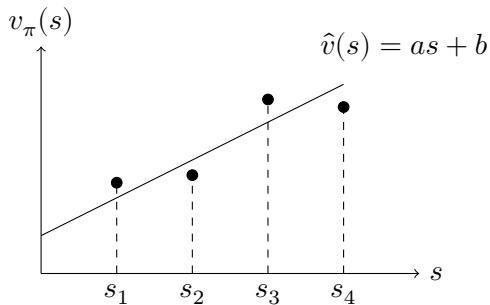
- 动作价值：

	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
\vdots			\vdots		
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

- 优势：直观，更容易理解分析
- 缺点：难以应对大且连续的状态空间。
主要体现在两点：1) 存储空间；2) 泛化能力。

从表格表示到函数表示

比如，我们可以用一条直线来拟合所有的状态价值：



比如这条直线的方程是：

$$\hat{v}(s, w) = as + b = [s, 1] \begin{bmatrix} a \\ b \end{bmatrix} = \phi^T(s)w$$

其中 w 是参数向量； $\phi(s)$ 是特征向量； $\hat{v}(s, w)$ 关于 w 是线性的。

从表格表示到函数表示

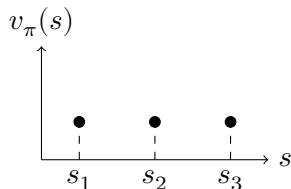
表格表示和函数表示有什么区别？

- 获取数据的方式不同：
 - 表格：将行列作为索引，直接去表中查得相应数据
 - 函数：需要输入特征向量计算： $s \rightarrow \phi(s) \rightarrow \phi^T(s)w = \hat{v}(s, w)$ 。
- 更新数据的方式不同：
 - 表格：将行列作为索引，直接去表中更新相应数据
 - 函数：通过更新 w 来间接更新相应数据 $\hat{v}(s, w)$ 。

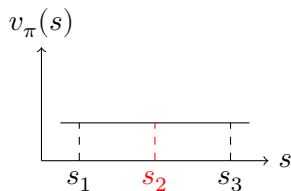
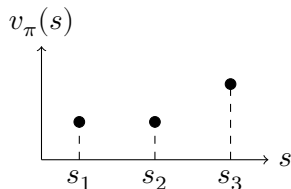
优点：我们不需要再存储所有的 $|S|$ 个状态价值，只需要存储 w 中的参数。

从表格表示到函数表示

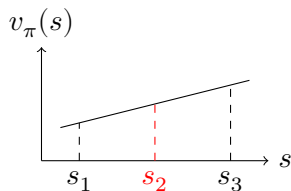
更新函数表示：



更新 $\hat{v}(s_3)$



更新 w



优点：泛化性增强。在上面的例子中，我们实际上是为了让直线更好的拟合 s_3 的状态价值去改变 w ，但是 s_2 的状态价值也跟着变化了。

从表格表示到函数表示

获得这些优点是有代价的。代价就是状态价值无法准确的被表示出来，这也是为什么这样的方法被称为值函数近似方法。

我们可以通过更高维度的曲线来拟合状态价值：

$$\hat{v}(s, w) = as^2 + bs + c = [s^2, s, 1] \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \phi^T(s)w$$

在这种情况下：

- 随着 w 和 $\phi(s)$ 维度的增加，状态价值可以被拟合更精确。
- 虽然 $\hat{v}(s, w)$ 对于 s 是非线性的，但是对于 w 是线性的。非线性体现在 $\phi(s)$ 中。

从表格表示到函数表示

快速小结：

- 核心：使用参数化的函数来近似状态价值和动作价值： $\hat{v}(s, w) \approx v_{\pi}(s)$ 其中 $w \in \mathbb{R}^m$ 是参数向量
- 关键区别：如何获取和更新 $v(s)$
- 优点：
 - 存储： w 的维度明显低于 $|S|$
 - 泛化性：当更新状态 s 的价值时，参数 w 会更新。所以一些其他状态的价值也会随着更新。

目标函数

正式定义问题：

- $\pi(s)$ 是状态价值的**真实值**； $\hat{v}(s, w)$ 是状态价值的**估计值**。
- 我们的目标是找到一个最优的参数 w ，使得 $\hat{v}(s, w)$ 尽可能的接近 $\pi(s)$ 。

为了找到最优的 w ，我们需要两步：

- 第一步：定义一个目标函数（损失函数）来量化 $\hat{v}(s, w)$ 和 $\pi(s)$ 之间的差距。
- 第二步：通过优化算法来最小化目标函数。

目标函数：

$$J(w) = \mathbb{E}[(v_{\pi}(S) - \hat{v}(S, w))^2]$$

有了目标函数，下一步就是使用优化算法来优化它：

- 为了最小化目标函数 $J(w)$ ，我们可以使用**梯度下降法**。

$$w_{k+1} = w_k - \alpha_k \nabla_w J(w_k)$$

其中梯度：

$$\begin{aligned}\nabla_w J(w) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] \\ &= \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S, w))^2] \\ &= 2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w))(-\nabla_w) \hat{v}(S, w)] \\ &= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w)]\end{aligned}$$

计算梯度需要涉及计算状态的期望。状态的分布是什么？

我们可以使用随机采样梯度来代替真实梯度：

$$\begin{aligned}w_{k+1} &= w_k + \alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w)] \\&\Downarrow \\w_{t+1} &= w_t + \alpha_t (v_\pi(s_t) - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)\end{aligned}$$

其中 s_t 是 S 的一个样本。在此已经把 $2\alpha_t$ 简化为 α_t 。

- 这些采样应当是按照状态的分布采样得来的，实际上可能不是。
- 我们需要知道 $v_\pi(s_t)$ 的值，实际上我们不知道。
- 但是我们可以通过一些估计值来代替 $v_\pi(s_t)$ ，至少让算法先可以实现。

具体而言：

- 使用值近似的 Monte Carlo 算法：

令 g_t 为从 s_t 出发的轨迹的折扣回报。那么 g_t 就可以用来估计 $v_\pi(s_t)$ 。

$$w_{t+1} = w_t + \alpha_t(g_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

- 使用值近似的 TD 算法：

按照 TD 算法的思路，我们也可以用 $r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t)$ 来估计 $v_\pi(s_t)$

$$w_{t+1} = w_t + \alpha_t(r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

使用值近似的 TD 算法

- 1: 初始化：一个对 w 可微的函数 $\hat{v}(s, w)$ 。初始化 $w = w_0$ 。
- 2: 目标：学习到策略 π 的真实状态价值。
- 3: **for** 每一个策略 π 产生的轨迹 $\{(s_t, r_{t+1}, s_{t+1})\}_t$ **do**
- 4: **for** 每一个经验样本 (s_t, r_{t+1}, s_{t+1}) **do**
- 5: 更新参数 w (一般情况):

$$w_{t+1} = w_t + \alpha_t(r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)$$
- 6: 更新参数 w (线性情况):

$$w_{t+1} = w_t + \alpha_t(r_{t+1} + \gamma \phi^T(s_{t+1})w_t - \phi^T(s_t)w_t) \phi(s_t)$$
- 7: **end for**
- 8: **end for**

虽然这个算法只能用来估计某个策略下的状态价值，但是它依然是后续介绍的算法的基础。

关于值近似 TD 算法的快速小结：

1) 从目标方程出发：

$$J(w) = \mathbb{E}[(v_{\pi}(S) - \hat{v}(S, w))^2]$$

2) 梯度下降法：

$$w_{t+1} = w_t + \alpha_t (v_{\pi}(s_t) - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)$$

3) 将真实的价值函数用估计值代替：

$$w_{t+1} = w_t + \alpha_t (r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)$$

数学上并不严谨，有兴趣的同学可以自行推导一下。问题出在 3) 当我们用估计值代替真实的价值之后，我们实际上优化的目标方程并不是 1)。

值近似 Sarsa 算法

目前为止我们主要考虑的是状态价值的估计，即：

$$\hat{v}(s) \approx v_{\pi}(s), \quad \forall s \in \mathcal{S}$$

为了找最优策略，我们需要估计动作价值。

值近似 Sarsa 算法：

$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$

和之前的值近似 TD 算法几乎一样，只是把 \hat{v} 替换成了 \hat{q} 。

值近似 Sarsa 算法

- 1: 初始化: 一个对 w 可微的函数 $\hat{q}(s, a, w)$ 。初始化 $w = w_0$ 。一个初始策略 π_0 。
- 2: 目标: 找到一个让智能体从一个特定的起点 s_0 出发, 到达终点的最优策略。
- 3: **for** 每一次生成轨迹 **do**
- 4: $a_0 \leftarrow \pi_0(s_0)$
- 5: **while** s_t 不是目标状态 **do**
- 6: 根据 (s_t, a_t) 收集经验样本 $(r_{t+1}, s_{t+1}, a_{t+1})$
- 7: 更新动作价值:
 $w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$
- 8: 更新策略:

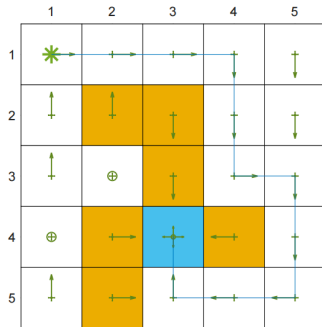
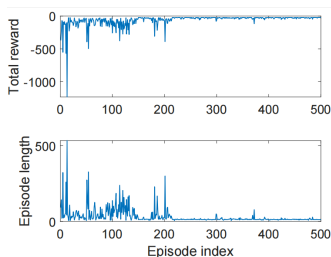
$$\pi(a|s_t) = \begin{cases} 1 - \frac{\epsilon}{|\mathcal{A}|} (|\mathcal{A}| - 1), & a = \arg \max_a \hat{q}(s_t, a, w_{t+1}) \\ \frac{\epsilon}{|\mathcal{A}|}, & a \neq \arg \max_a \hat{q}(s_t, a, w_{t+1}) \end{cases}$$

- 9: $s_{t+1} \leftarrow s_t, a_{t+1} \leftarrow a_t$
- 10: **end while**
- 11: **end for**

值近似 Sarsa 算法

例子：

- 使用线性函数进行值近似的 Sarsa: $\hat{q}(s, a, w) = \phi^T(s, a)w$
- $\gamma = 0.9, \epsilon = 0.1, r_{\text{boundary}} = r_{\text{forbidden}} = -10, r_{\text{target}} = 1, \alpha = 0.001$



值近似 Q-Learning 算法

与 Sarsa 类似，Q-Learning 也可以通过值近似推广。
值近似 Q-Learning 算法：

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

和之前的值近似 Sarsa 算法几乎一样，只是把 $\hat{q}(s_{t+1}, a_{t+1}, w_t)$ 替换成了 $\max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t)$ 。

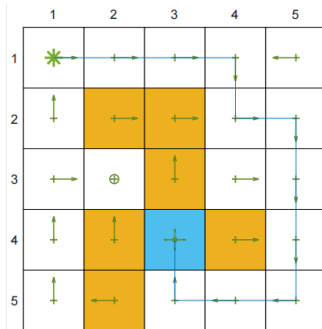
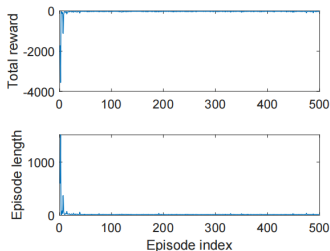
值近似 Q-Learning 算法

- 1: 初始化: 一个对 w 可微的函数 $\hat{q}(s, a, w)$ 。初始化 $w = w_0$ 。一个初始策略 π_0 。
- 2: 目标: 找到一个让智能体从一个特定的起点 s_0 出发, 到达终点的最优策略。
- 3: **for** 每一次生成轨迹 **do**
- 4: $a_0 \leftarrow \pi_0(s_0)$
- 5: **while** s_t 不是目标状态 **do**
- 6: 根据 (s_t, a_t) 收集经验样本 (r_{t+1}, s_{t+1})
- 7: 更新动作价值:
$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$
- 8: 更新策略:
$$\pi(a|s_t) = \begin{cases} 1 - \frac{\epsilon}{|\mathcal{A}|} (|\mathcal{A}| - 1), & a = \arg \max_a \hat{q}(s_t, a, w_{t+1}) \\ \frac{\epsilon}{|\mathcal{A}|}, & a \neq \arg \max_a \hat{q}(s_t, a, w_{t+1}) \end{cases}$$
- 9: $s_{t+1} \leftarrow s_t, a_{t+1} \leftarrow a_t$
- 10: **end while**
- 11: **end for**

值近似 Q-Learning 算法

例子:

- 使用线性函数进行值近似的 Q-Learning: $\hat{q}(s, a, w) = \phi^T(s, a)w$
- $\gamma = 0.9, \epsilon = 0.1, r_{\text{boundary}} = r_{\text{forbidden}} = -10, r_{\text{target}} = 1, \alpha = 0.001$



- 最早将神经网络与强化学习的结合的算法之一（不是第一）。
- 其中神经网络的作用是作为一个非线性方程的拟合器
- 与下列算法不同：

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

因为训练神经网络的方式与之前的方法不同。

Deep Q-Learning 的目标函数（损失函数）：

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

\Downarrow

$$J(w) = \mathbb{E} \left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w) \right)^2 \right]$$

其中 (S, A, R, S') 都是随机变量。

Deep Q-Learning

如何最小化这个目标函数？梯度下降法！

- 如何计算目标函数的梯度？
- 目标函数：

$$J(w) = \mathbb{E} \left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w) \right)^2 \right]$$

其中参数 w 不但出现在了 $\hat{q}(S, A, w)$ 中，还出现在了

$$y \doteq R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w)$$

- 但是

$$\nabla_w y \neq \gamma \max_{a \in \mathcal{A}(S')} \nabla_w \hat{q}(S', a, w)$$

- 为了解决这个问题，我们可以假设在 y 中的 w 是不变的（至少对于一段短时间来说，比如 5-10 个经验样本）。

Deep Q-Learning

为了解决这个问题，我们可以引入两个神经网络：

- 一个是主网络 $\hat{q}(s, a, w)$
- 另一个是目标网络 $\hat{q}(s, a, w_T)$

如此，我们可以将目标函数可以退化为：

$$J = \mathbb{E} \left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w) \right)^2 \right]$$

其中 w_T 是目标网络中的参数。

当 w_T 固定时， J 的梯度就可以按照以下公式计算了：

$$\nabla_w J = -2\mathbb{E} \left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w) \right) \nabla_w \hat{q}(S, A, w) \right]$$

Deep Q-Learning 的基本思想就是使用梯度下降法最小化目标函数。

改进点 1: 双网络，一个目标网络和一个主网络

实现方式:

- 令 w 和 w_T 分别是主网络和目标网络的参数，将它们初始化为一样的值。
- 每一轮迭代时，从经验池中取出一小批经验样本 $\{(s, a, r, s')\}$
- 对于其中的每一个经验样本 (s, a, r, s') 计算目标值:

$$y_T \doteq r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$$

- 得到一小批数据: $\{(s, a, y_T)\}$
- 用 $\{(s, a, y_T)\}$ 训练主网络最小化 $(y_T - \hat{q}(s, a, w))^2$

改进点 2: 经验重放

什么是经验重放?

实现方式:

- 在我们采集了足够多的经验样本之后, 我们并不按照采集他们的顺序去使用它们。
- 而是把它们存在一个叫做经验池的集合中 $\mathcal{B} \doteq \{(s, a, r, s')\}$
- 每当我们需要一小批数据的时候, 我们就从这个集合中进行随机采样
- 我们的采样应该遵循均匀分布。(?)

Deep Q-Learning

- 1: 初始化：两个具有相同初始参数的网络 w 和 w_T 。
- 2: 目标：从一个任意策略 π_b 出发学习出所有的最优动作价值。
- 3: 将 π_b 产生的经验样本存入经验池 $\mathcal{B} = \{(s, a, r, s')\}$
- 4: **for** 每一次迭代 **do**
- 5: 从 \mathcal{B} 中按照均匀分布采样出一小批经验样本
- 6: **for** 每一个经验样本 (s, a, r, s') **do**
- 7: 使用目标网络计算最优动作价值 $y_T = r + \gamma \max_{a \in \mathcal{A}(s_{S'})} \hat{q}(s', a, w_T)$
- 8: **end for**
- 9: 使用这一小批数据更新主网络，使得 $(y_T - \hat{q}(s, a, w))^2$ 最小
- 10: 每过 k 轮将 w_T 设置为 w 。
- 11: **end for**
 - 为什么没有策略更新环节？
 - 与论文中的输入输出有所不同。

例子：

- 只用了一条轨迹来生成经验样本。
- 这条轨迹有 1000 步。
- 产生这条轨迹的策略就是在每一个状态都均匀的随机选择一个动作。
- 使用的神经网络只有一层隐藏全连接层 + 非线性激活函数用来拟合 $\hat{q}(s, a, w)$ ，这一层有 100 个神经元。

Deep Q-Learning

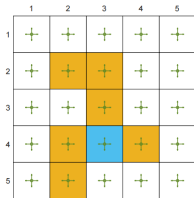


图: 行为策略

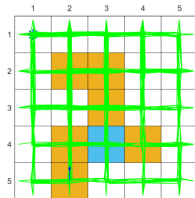


图: 轨迹

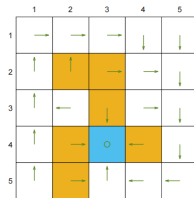


图: 习得策略

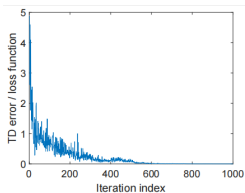


图: TD Error

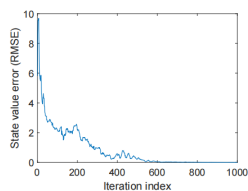
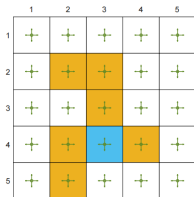


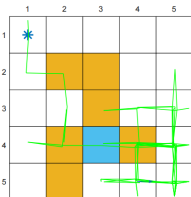
图: 状态价值误差

Deep Q-Learning

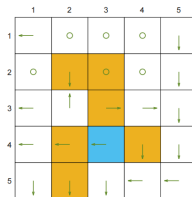
如果只有 100 步呢？



图：行为策略



图：轨迹



图：习得策略

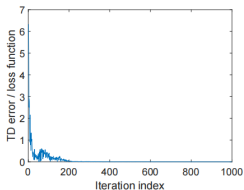


图: TD Error

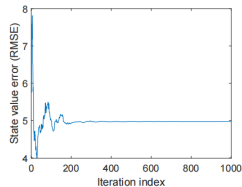


图: 状态价值误差

值近似方法：

- 值近似 TD 算法

$$w_{t+1} = w_t + \alpha_t (r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)$$

- 值近似 Sarsa 算法

$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$

- 值近似 Q-Learning 算法

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

- Deep Q-Learning
 - 双网络
 - 经验重放

The End