重庆邮电大学

计算机科学与技术学院

人工智能原理

强化学习

强化学习中的概念

智能体（agent）： 智能体是强化学习算法的主体，它能够根据经验做出主观判断并执行动作，是整个智能系统的核心。

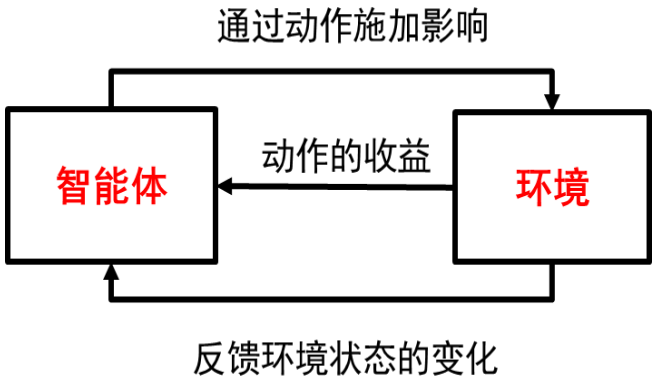
环境（environment）： 智能体以外的一切统称为环境，环境在与智能体的交互中，能被智能体所采取的动作影响，同时环境也能向智能体反馈状态和奖励。

状态（state）： 状态可以理解为智能体对环境的一种理解和编码，通常包含了对智能体所采取决策产生影响的信息。

动作（action）： 动作是智能体对环境产生影响的方式。

策略（policy）： 策略是智能体在所处状态下去执行某个动作的依据，即给定一个状态，智能体可根据一个策略来选择应该采取的动作。

奖励（reward）： 奖励是智能体序贯式采取一系列动作后从环境获得的收益。



强化学习的特点

	有监督学习	无监督学习	强化学习
学习依据	基于监督信息	基于对数据结构的假设	基于评估
数据来源	一次给定	一次给定	在时序交互中产生
决策过程	单步决策 (如分类和识别等)	无	序贯决策 (如棋类博弈)
学习目标	样本到语义标签的映射	数据的分布模式	选择能够获取最大收益的状态到动作的映射

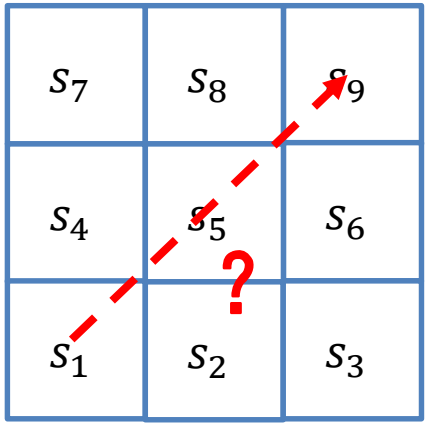
- **基于评估：**强化学习利用环境评估当前策略，以此为依据进行优化
- **交互性：**强化学习的数据在与环境的交互中产生
- **序列决策过程：**智能主体在与环境的交互中需要作出一系列的决策，这些决策往往是前后关联的

注：现实中强化学习问题往往还具有奖励滞后，基于采样的评估等特点

马尔可夫决策过程

（序列优化）问题：

- 在下图网格中，假设有一个机器人位于s_1，其每一步只能向上或向右移动一格，跃出方格会被惩罚（且游戏停止）
- 如何使用强化学习找到一种策略，使机器人从s_1到达s_9？



刻画解该问题的因素

智能主体	迷宫机器人
环境	3×3方格
状态	机器人当前时刻所处方格
动作	每次移动一个方格
奖励	到达s_9时给予奖励；越界时给予惩罚

离散马尔可夫过程（Discrete Markov Process）

- 一个随机过程实际上是一列随时间变化的随机变量，其中当时间是离散量时，一个随机过程可以表示为$\{X_t\}_{t=0,1,2,\cdots}$，其中每个X_t都是一个随机变量，这被称为离散随机过程
- 马尔可夫链（Markov Chain）：满足马尔可夫性（Markov Property）的离散随机过程，也被称为离散马尔科夫过程。

$$Pr(X_{t+1} = x_{t+1} | X_0 = x_0, X_1 = x_1, \cdots, X_t = x_t) = Pr(X_{t+1} = x_{t+1} | X_t = x_t)$$

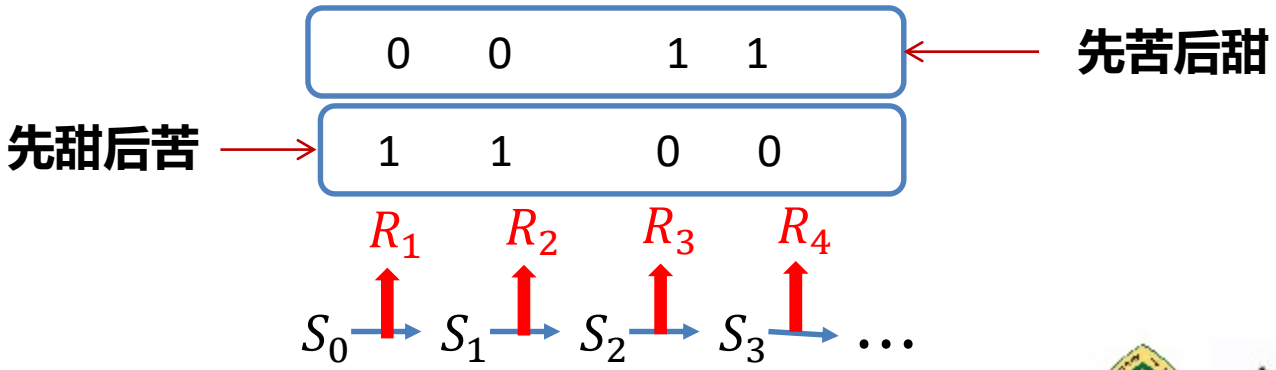
$t+1$时刻状态仅与t时刻状态相关

马尔可夫奖励过程（Markov Reward Process）：引入奖励

为了在序列决策中对目标进行优化，在马尔可夫随机过程框架中加入了奖励机制：

- 奖励函数$R: S \times S \mapsto \mathbb{R}$，其中$R(S_t, S_{t+1})$描述了从第$t$步状态转移到第 $t+1$步状态所获得奖励
- 在一个序列决策过程中，不同状态之间的转移产生了一系列的奖励$(R_1, R_2, \cdots)$，其中R_{t+1}为$R(S_t, S_{t+1})$的简便记法。
- 引入奖励机制，这样可以衡量任意序列的优劣，即对序列决策进行评价。

问题：给定两个因为状态转移而产生的奖励序列(1,1,0,0)和(0,0,1,1)，哪个序列决策更好?



马尔可夫奖励过程（Markov Reward Process）

问题：给定两个因为状态转移而产生的奖励序列(1,1,0,0)和(0,0,1,1)，哪个奖励序列更好?

为了比较不同的奖励序列，定义反馈（return），用来反映累加奖励：

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

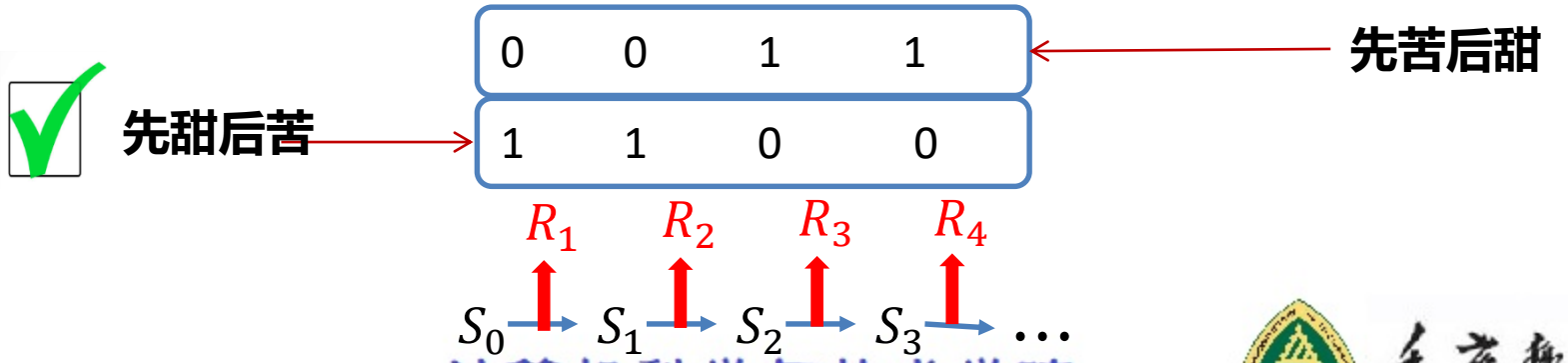
其中折扣系数（discount factor）$\gamma \in [0, 1]$

反馈值反映了某个时刻后所得到累加奖励，当衰退系数小于1时，越是遥远的未来对累加反馈的贡献越少

假设 $\gamma = 0.99$

(1,1,0,0)：$G_0 = 1 + 0.99 \times 1 + 0.99^2 \times 0 + 0.99^3 \times 0 = 1.99$

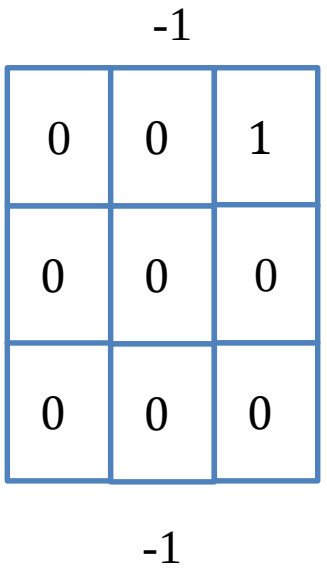
(0,0,1,1)：$G_0 = 0 + 0.99 \times 0 + 0.99^2 \times 1 + 0.99^3 \times 1 = 1.9504$



马尔可夫奖励过程（Markov Reward Process）

使用离散马尔可夫过程描述机器人移动问题

- 随机变量序列$\{S_t\}_{t=0,1,2,\cdots}$：S_t表示机器人第t步的位置，每个随机变量S_t的取值范围为$S = \{s_1, s_2, \cdots, s_9, s_d\}$
- 状态转移概率：$Pr(S_{t+1}|S_t)$满足马尔可夫性
- 定义奖励函数$R(S_t, S_{t+1})$：从S_t到S_{t+1}所获得奖励，其取值如图中所示
- 定义衰退系数：$\gamma \in [0, 1]$



综合以上信息，可用$MRP = \{S, Pr, R, \gamma\}$来刻画马尔科夫奖励过程

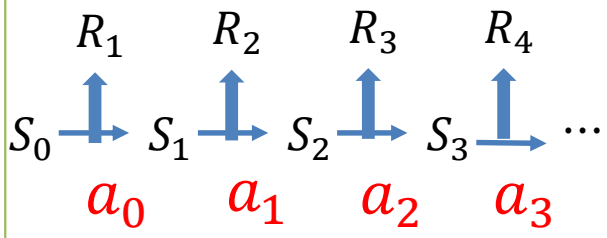
这个模型不能体现机器人能动性，仍然缺乏与环境进行进行交互的手段

马尔可夫决策过程（Markov Decision Process）：引入动作

在强化学习问题中，智能主体与环境交互过程中可自主决定所采取的动作，不同**动作**会对环境产生不同影响，为此：

- 定义智能主体能够采取的动作集合为A
- 由于不同的动作对环境造成的影响不同，因此状态转移概率定义为$Pr(S_{t+1}|S_t, a_t)$，其中$a_t \in A$为第t步采取的动作
- 奖励可能受动作的影响，因此修改奖励函数为$R(S_t, a_t, S_{t+1})$

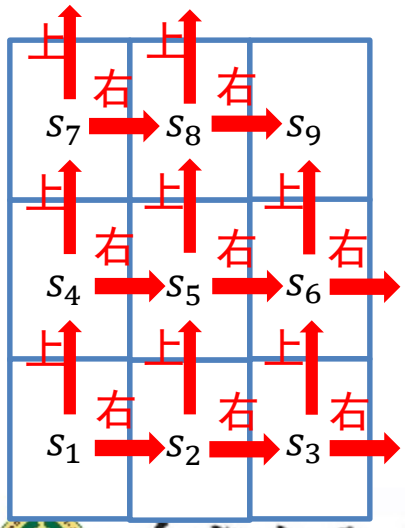
- 动作集合A可以是有限的，也可以是无限的
- 状态转移可是确定（deterministic）的，也可以是随机概率性（stochastic）的。
- 确定状态转移相当于发生从S_t到S_{t+1}的转移概率为1



马尔可夫决策过程（Markov Decision Process）

使用离散马尔可夫过程描述机器人移动问题

- 随机变量序列$\{S_t\}_{t=0,1,2,\cdots}$：S_t表示机器人第t步所在位置（即状态），每个随机变量S_t的取值范围为$S = \{s_1, s_2, \cdots, s_9, s_d\}$
- 动作集合：$A = \{$上，右$\}$
- 状态转移概率$Pr(S_{t+1}|S_t, a_t)$：满足马尔可夫性，其中$a_t \in A$。状态转移如图所示。
- 奖励函数：$R(S_t, a_t, S_{t+1})$
- 衰退系数：$\gamma \in [0, 1]$



综合以上信息，可通过$MDP = \{S, A, Pr, R, \gamma\}$来刻画马尔科夫决策过程

马尔可夫决策过程（Markov Decision Process）

- 马尔可夫决策过程$MDP = \{S, A, Pr, R, \gamma\}$是刻画强化学习中环境的标准形式
- 马尔可夫决策过程可用如下序列来表示：

$$S_0 \xrightarrow[a_0]{R_1} S_1 \xrightarrow[a_1]{R_2} S_2 \xrightarrow[a_2]{R_3} S_3 \xrightarrow[a_3]{R_4} \cdots$$

马尔科夫过程中产生的状态序列称为轨迹(trajectory)，可如下表示

$$(S_0, a_0, R_1, S_1, a_1, R_2, \cdots, S_T)$$

- 轨迹长度可以是无限的，也可以有终止状态S_T。有终止状态的问题叫做分段的（即存在回合的）（episodic），否则叫做持续的（continuing）

分段问题中，一个从初始状态到终止状态的完整轨迹称为一个片段或回合（episode）。如围棋对弈中一个胜败对局为一个回合。

马尔可夫决策过程（Markov Decision Process）

在机器人移动问题中：状态、行为、衰退系数、起始/终止状态、反馈、状态转移概率矩阵的定义如下

$S = \{s_1, s_2, \cdots, s_9, s_d\}$

$A = \{$上，右$\}$

$\gamma = 0.99$

起始状态：$S_0 = s_1$

终止状态：$S_T \in \{s_9, s_d\}$

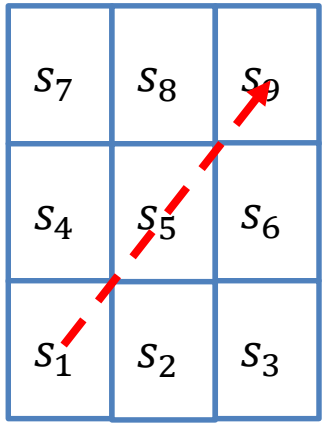
$$R(S_t, a_t, S_{t+1}) = \begin{cases} 1, \text{如果} S_{t+1} = s_9 \\ -1, \text{如果} S_{t+1} = s_d \\ 0, \text{其他情况} \end{cases}$$

$Pr(S_{t+1}|S_t, a_t = \text{右})$

S_t \ S_{t+1}	s_1	s_2	s_3	…	s_9	s_d
s_1	0	1	0	…	0	0
s_2	0	0	1	…	0	0
s_3	0	0	0	…	0	1
…						
s_8	0	0	0	…	1	0
s_9	0	0	0	…	0	1

$Pr(S_{t+1}|S_t, a_t = \text{上})$

S_t \ S_{t+1}	s_1	s_4	s_7	…	s_9	s_d
s_1	0	1	0	…	0	0
s_4	0	0	1	…	0	0
s_7	0	0	0	…	0	1
…						
s_6	0	0	0	…	1	0
s_9	0	0	0	…	0	1



如何从起始状态到终止状态？

强化学习中的策略学习

马尔可夫决策过程$MDP = \{S, A, Pr, R, \gamma\}$对环境进行了描述，那么

智能主体如何与环境交互而完成任务？需要进行策略学习

策略函数：

- 策略函数$\pi: S \times A \mapsto [0, 1]$，其中$\pi(s, a)$的值表示在状态$s$下采取动作$a$的概率。
- 策略函数的输出可以是确定的，即给定s情况下，只有一个动作a使得概率$\pi(s, a)$取值为1。对于确定的策略，记为$a = \pi(s)$。

强化学习中的策略学习

如何进行策略学习：一个好的策略是在当前状态下采取了一个行动后，该行动能够在未来收到最大化的反馈：

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

由马尔可夫性，未来的状态和奖励只与当前状态相关，与t无关。因此t取任意值该等式均成立，如“逢山开路，遇水搭桥”。

为了对策略函数π进行评估，定义

- **价值函数（Value Function）**$V: S \mapsto \mathbb{R}$，其中$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$，即在第$t$步状态为$s$时，按照策略$\pi$行动后在未来所获得反馈值的期望
- **动作-价值函数(Action-Value Function)** $q: S \times A \mapsto \mathbb{R}$，其中$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$表示在第$t$步状态为$s$时，按照策略$\pi$采取动作$a$后，在未来所获得反馈值的期望

这样，策略学习转换为如下优化问题：

寻找一个最优策略π^*，对任意$s \in S$使得$V_{\pi^*}(s)$值最大

价值函数与动作-价值函数的关系

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

$$= \mathbb{E}_{a\sim\pi(s,\cdot)}\left[\mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]\right]$$

$$= \sum_{a\in A} \underbrace{\pi(s,a)}_{\text{采取动作}a\text{的概率}}$$

$$V_\pi(s) = \sum_{a\in A} \pi(s,a) q_\pi(s,a)$$

可以用动作-价值函数来表达价值函数，即从状态s出发、采用策略π完成任务所得回报期望可如下计算：在状态s可采取的每个动作的概率值与采取这一动作而获得价值的乘积之和（即期望）。这里$\pi(s,a)$表示在状态s下采取动作a的概率、$q_\pi(s,a)$为在状态s采取动作a后的回报期望。也就是说，状态s的价值可用该状态下可采取所有动作而取得的期望价值来表述。

价值函数与动作-价值函数的关系

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$$
$$= \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s,a,s') + \gamma \mathbb{E}_\pi[R_{t+2} + \gamma R_{t+3} + \cdots | S_{t+1} = s']]$$

$$= \sum_{s' \in S} \underbrace{P(s'|s,a)}_{\text{在状态}s\text{采取行动}a\text{进入状态}s'\text{的概率}} \times [\underbrace{R(s,a,s')}_{\text{在}s\text{采取}a\text{进入}s'\text{得到的回报}} + \gamma \times \underbrace{V_\pi(s')}_{\text{在}s'\text{获得的回报期望}}]$$

⇩

$$q_\pi(s,a) = \sum_{s' \in S} P(s'|s,a)\,[R(s,a,s') + \gamma V_\pi(s')]$$

可知：可用价值函数来表示动作-价值函数，即在状态s采取动作a所取得价值如下计算：采取某个具体动作a进入状态s'的概率，乘以进入后续状态s'所得回报$R(s,a,s')$与后续状态s'价值函数的折扣值之和，然后对在状态s采取动作a后所进入全部状态的如上取值进行累加。这里$P(s'|s,a)$表示在状态s下采取动作a转移到状态s'的概率。也就是说，在某个状态下执行某一个动作所取得价值可以通过执行该动作之后进入的所有状态获得的瞬时奖励和后续状态可取得价值的期望来表示。

价值函数与动作-价值函数的关系：价值函数的贝尔曼方程

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s'\in S} P(s'|s,a)\left[R(s,a,s') + \gamma V_\pi(s')\right]$$

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)\sum_{s'\in S} P(s'|s,a)\left[R(s,a,s') + \gamma V_\pi(s')\right]$$
$$= \mathbb{E}_{a\sim\pi(s,\cdot)}\,\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s') + \gamma V_\pi(s')\right]$$

从这个公式可知，价值函数取值与时间没有关系，只与策略π、在策略π下从某个状态转移到其后续状态所取得的回报以及在后续所得回报有关。

进一步分析价值函数的贝尔曼方程，可见状态s可获得“好处”$V_\pi(s)$由两个部分构成：一个是在状态s执行当前动作所得到的瞬时奖励，另外一个是在后续状态所能得回报期望（价值）的折扣值。该式中出现了期望，是因为在状态s可以一定概率进入到多个后续状态，且从状态s进入某个后续状态均会有不同的价值。

价值函数与动作-价值函数的关系：价值函数的贝尔曼方程

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s'\in S} P(s'|s,a)\left[R(s,a,s') + \gamma V_\pi(s')\right]$$

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)\sum_{s'\in S} P(s'|s,a)\left[R(s,a,s') + \gamma V_\pi(s')\right]$$
$$= \mathbb{E}_{a\sim\pi(s,\cdot)}\,\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a,s') + \gamma V_\pi(s')\right]$$

由于在每个状态都有多种动作可供智能体选择，在每个状态施以某个具体动作后可按照一定概率转移到一个新的后续状态，因此相比于关心每个状态的价值，显然关心在当前状态下施以某个动作带来的价值更加直观实用。如果智能体已经知道在某个状态下所有可供选择动作带来的不同价值，那么智能体在当前状态就应该去选择能带来最大价值的对应动作去执行，这就是设计动作-价值函数的初衷。

价值函数与动作-价值函数的关系：价值函数的贝尔曼方程

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s'\in S} P(s'|s,a)\,[R(s,a,s') + \gamma V_\pi(s')]$$

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)\sum_{s'\in S} P(s'|s,a)\,[R(s,a,s') + \gamma V_\pi(s')]$$
$$= \mathbb{E}_{a\sim\pi(s,\cdot)}\,\mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma V_\pi(s')]$$

价值函数的贝尔曼方程

动作-价值函数中的回报与价值函数中的回报值不一样，动作-价值函数中的回报是在某个状态执行完某个具体动作之后取得回报的期望值，而价值函数中的回报值是在某个状态下选择所有动作执行后所得回报的期望值。

价值函数与动作-价值函数的关系：动作-价值函数的贝尔曼方程

$$V_\pi(s) = \sum_{a\in A}\pi(s,a)q_\pi(s,a)$$

$$q_\pi(s,a) = \sum_{s'\in S} P(s'|s,a)\left[R(s,a,s') + \gamma V_\pi(s')\right]$$

$$q_\pi(s,a) = \sum_{s'\in S} P(s'|s,a)\left[R(s,a,s') + \gamma \sum_{a'\in A}\pi(s',a')q_\pi(s',a')\right]$$

$$= \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi(s',\cdot)}[q_\pi(s',a')]]$$

动作-价值函数的贝尔曼方程

这个公式说明动作-价值函数取值同样与时间没有关系，而是与瞬时奖励和下一步的状态和动作有关。动作-价值函数表示在状态s采取了动作a后获得的“好处”，这也可以分为两部分：一是从状态s采取动作a后带来的瞬时奖励，二是进入后续状态后根据当前策略选择动作所得期望回报的折扣值。

价值函数与动作-价值函数的关系：动作-价值函数的贝尔曼方程

价值函数的贝尔曼方程

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s,\cdot)}\,\mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma V_\pi(s')]$$

动作-价值函数的贝尔曼方程

$$q_\pi(s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi(s',\cdot)}[q_\pi(s',a')]]$$

贝尔曼方程描述了价值函数或动作-价值函数的递推关系，是研究强化学习问题的重要手段。其中价值函数的贝尔曼方程描述了当前状态价值函数和其后续状态价值函数之间的关系，即当前状态价值函数等于瞬时奖励的期望加上后续状态的（折扣）价值函数的期望。而动作-价值函数的贝尔曼方程描述了当前动作-价值函数和其后续动作-价值函数之间的关系，即当前状态下的动作-价值函数等于瞬时奖励的期望加上后续状态的（折扣）动作-价值函数的期望。

价值函数与动作-价值函数的关系：动作-价值函数的贝尔曼方程

价值函数的贝尔曼方程

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s,\cdot)}\, \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma V_\pi(s')]$$

动作-价值函数的贝尔曼方程

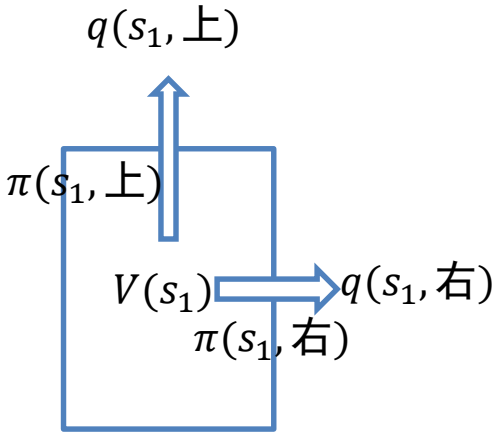
$$q_\pi(s,a) == \mathbb{E}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi(s',\cdot)}[q_\pi(s',a')]]$$

在实际中，需要计算得到最优策略以指导智能体在当前状态如何选择一个可获得最大回报的动作。求解最优策略的一种方法就是去求解最优的价值函数或最优的动作-价值函数（即基于价值方法，value-based approach）。一旦找到了最优的价值函数或动作-价值函数，自然而然也就是找到最优策略。当然，在强化学习中还有基于策略（policy-based）和基于模型（model-based）等不同方法。

价值函数与动作-价值函数的关系：以状态s_1的计算为例

$$V_\pi(s) = \sum_{a \in A} \pi(s,a) q_\pi(s,a)$$

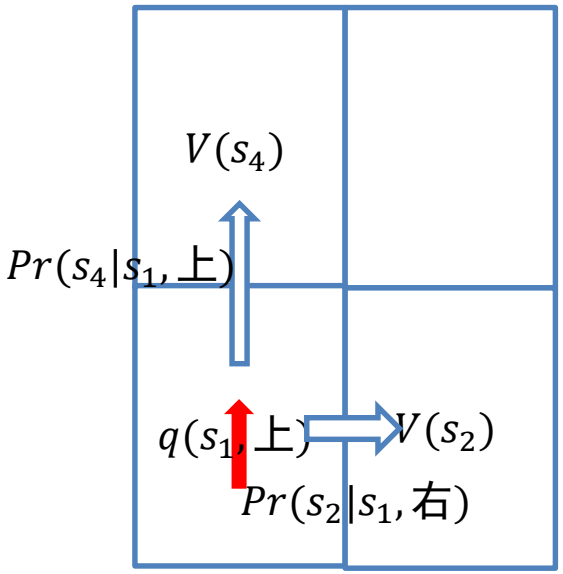
$$V_\pi(s_1) = \pi(s_1, 上) q_\pi(s_1, 上) + \pi(s_1, 右) q_\pi(s_1, 右)$$



不同动作下的反馈累加

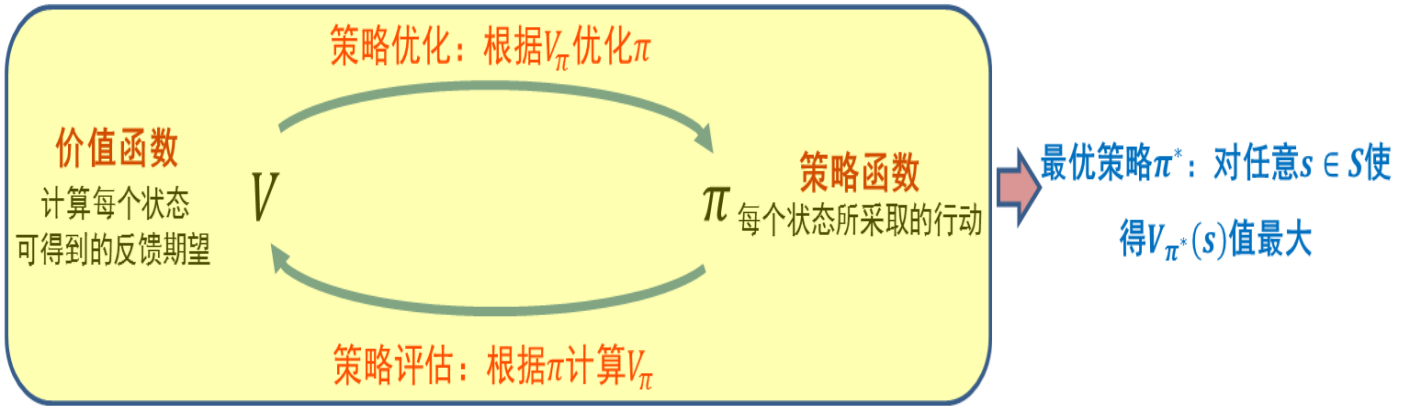
$$q_\pi(s,a) = \sum_{s' \in S} Pr(s'|s,a)\,[R(s,a,s') + \gamma V_\pi(s')]$$

$$q_\pi(s_1, 上) = Pr(s_4|s_1, 上)[R(s_1, 上, s_4) + \gamma V_\pi(s_4)]$$



动作确定时状态转移后的反馈结果

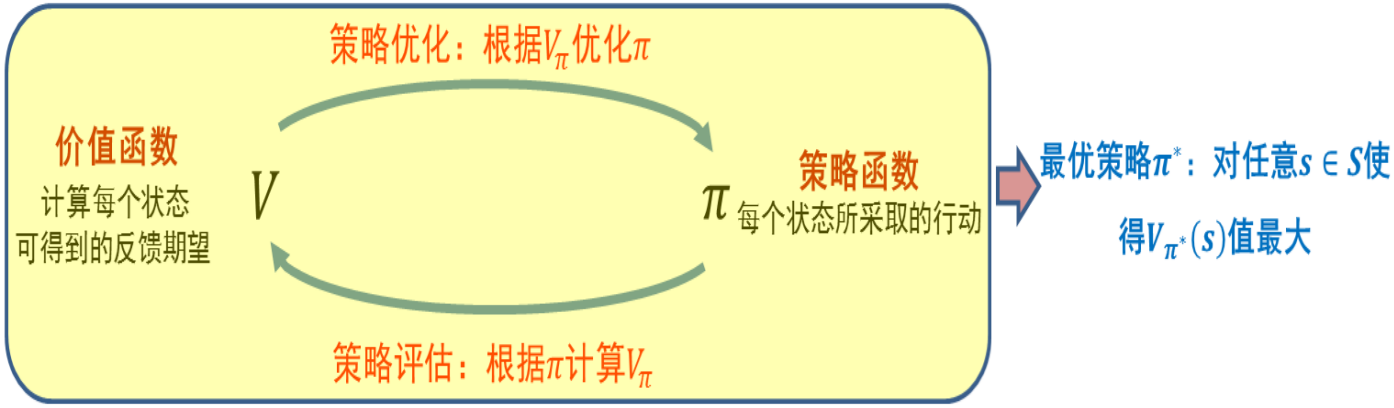
强化学习的问题求解



回顾强化学习问题的定义: 给定一个马尔可夫决策过程$MDP = (S, A, P, R, \gamma)$，

强化学习会寻找一个最优策略π^*，在策略π^*作用下使得任意状态$s \in S$对应的价值函数$V_{\pi^*}(s)$取值最大。

强化学习的问题与求解



为了求解最优策略π^*，图中展示了一种思路：从一个任意的策略开始，首先计算该策略下价值函数（或动作-价值函数），然后根据价值函数调整改进策略使其更优，不断迭代这个过程直到策略收敛。通过策略计算价值函数的过程叫做策略评估（policy evaluation），通过价值函数优化策略的过程叫做策略优化（policy improvement），策略评估和策略优化交替进行的强化学习求解方法叫做通用策略迭代（Generalized Policy Iteration，GPI）。

强化学习中的策略优化

策略优化定理：

对于确定的策略π和π'，如果对于任意状态$s \in S$

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

那么对于任意状态$s \in S$，有

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

即策略π'不比π差

注意，不等式左侧的含义是只在当前这一步将动作修改为$\pi'(s)$，未来的动作仍然按照π的指导进行

在讨论如何优化策略之前，首先需要明确什么是“更好”的策略。分别给出π和π'两个策略，如果对于任意状态$s \in S$，有$V_\pi(s) \leq V_{\pi'}(s)$，那么可以认为策略$\pi'$不比策略$\pi$差，可见“更优”策略是一个偏序关系。

强化学习中的策略优化

给定当前策略π、价值函数V_π和行动-价值函数q_π时，可如下构造新的策略π'，只要π'满足如下条件：

$$\pi'(s) = \text{argmax}_a q_\pi(s,a) \quad （对于任意s \in S）$$

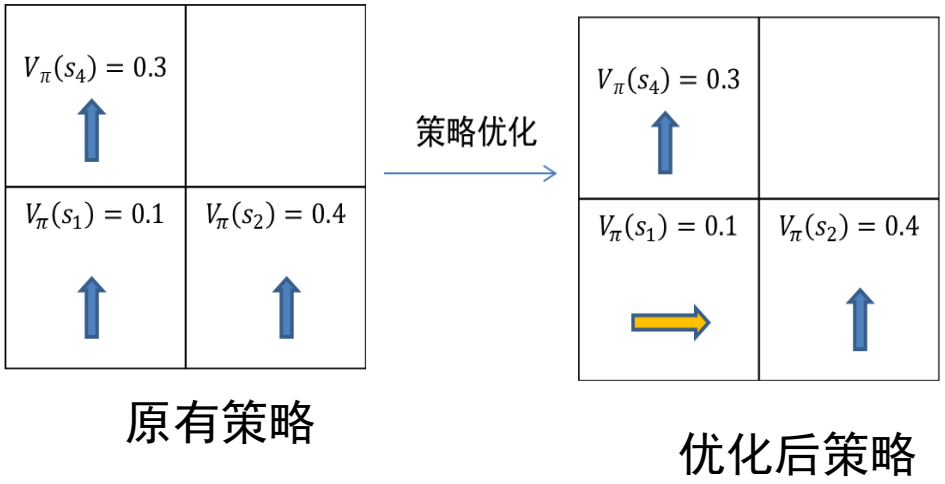
π'便是对π的一个改进。于是对于任意$s \in S$，有

$$q_\pi\big(s,\pi'(s)\big) = q_\pi\big(s,\text{argmax}_a\, q_\pi(s,a)\big)$$

$$= \max_a q_\pi(s,a)$$

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

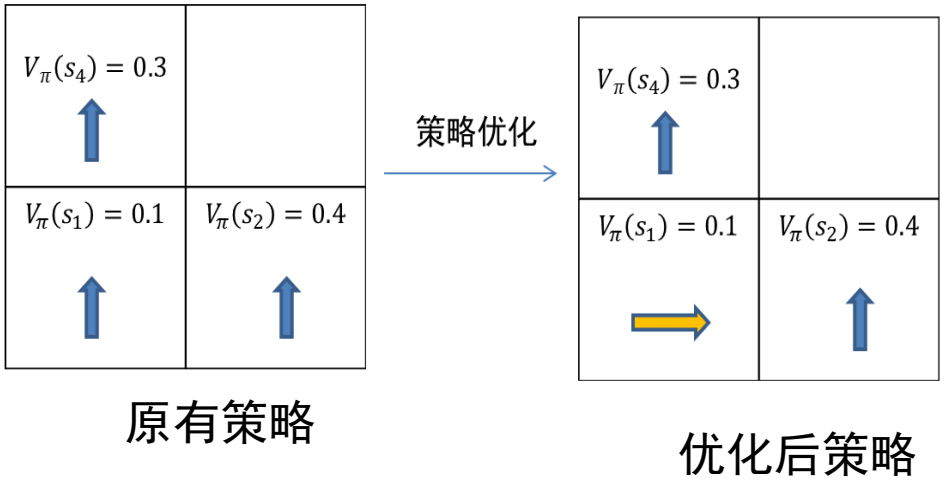
强化学习中的策略优化：机器人寻路问题为例子



原有策略　　优化后策略

左图给出了机器人寻路问题的原有策略及其对应价值函数。在左图中，根据原有策略，智能体位于状态s_1的价值函数取值为0.1。当智能体位于s_1状态时，智能体在原有策略指引下将选择向上移动一个方格的行动，从状态s_1进入状态s_4，状态s_4的价值函数取值为0.3。原有策略所给出的其他信息，可从左图周知。

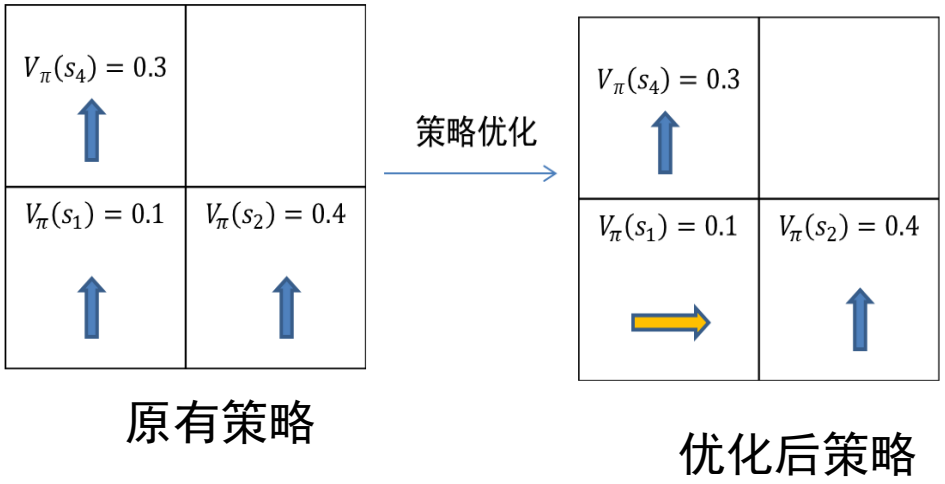
强化学习中的策略优化：机器人寻路问题为例子



$$q_\pi(s_1, \text{上}) = \sum_{s' \in S} \mathrm{P}(s'|s_1, \text{上}) \left[R(s_1, \text{上}, s') + \gamma V_\pi(s')\right]$$
$$= 1 \times (0 + 0.99 \times 0.3) + 0 \times \cdots = 0.297$$

下面来看智能体如何通过策略优化来改变在状态s_1所采取的行动。由于智能体在状态s_1能够采取“向上移动一个方格”或“向右移动一个方格”两个行动中的一个。首先计算状态s_1选择“向上移动一个方格”后所得动作-价值函数取值。

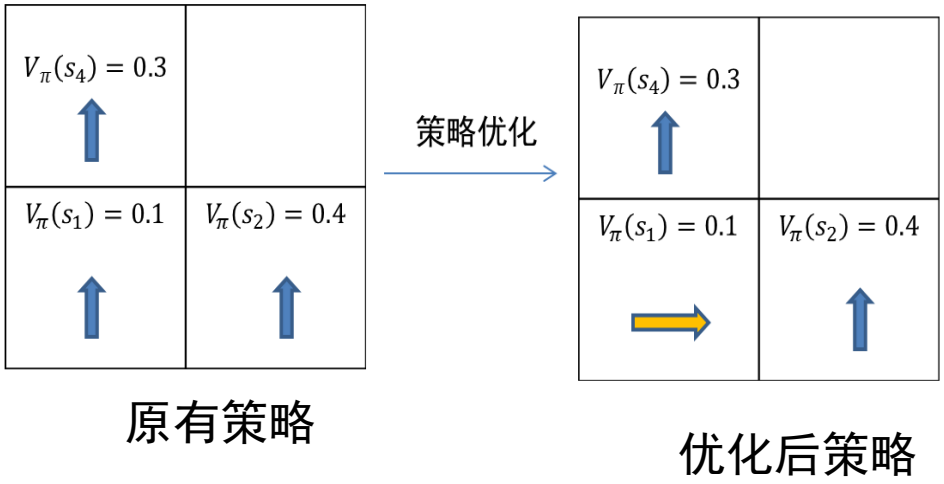
强化学习中的策略优化：机器人寻路问题为例子



$$q_\pi(s_1, \text{右}) = \sum_{s' \in S} \mathrm{P}(s'|s_1, \text{右}) \left[R(s_1, \text{右}, s') + \gamma V_\pi(s')\right]$$
$$= 1 \times (0 + 0.99 \times 0.4) + 0 \times \cdots = 0.396$$

接着计算状态s_1选择“向右移动一个方格”后所得动作-价值函数取值。

强化学习中的策略优化：机器人寻路问题为例子



$$q_\pi(s_1, 右) == 0.396 > q_\pi(s_1, 上) = 0.297$$

可见，智能体在状态s_1选择“向上移动一个方格”行动所得回报$q_\pi(s_1, 上)$值为0.297、选择“向右移动一个方格”行动所得回报$q_\pi(s_1, 右)$值为0.396。显然，智能体在状态s_1应该选择“向右移动一个方格”行动，这样能够获得更大的回报。于是，经过策略优化后，状态s_1处的新策略为$\pi'(s_1) = \text{argmax}_a q_\pi(s, a) = 右$，则将$s_1$处的策略从“上”更新为 “右”。其他状态的情况可用类似方法计算得到。

强化学习中的策略评估

- 动态规划
- 蒙特卡洛采样
- 时序差分（Temporal Difference）

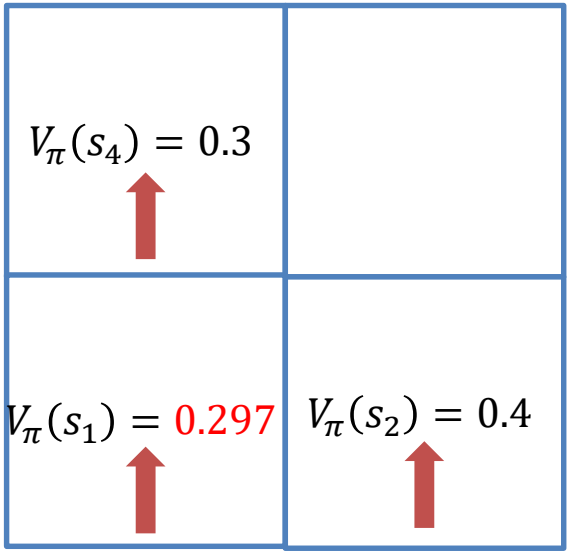
假定当前策略为π，策略评估指的是根据策略π来计算相应的价值函数V_π或动作-价值函数q_π。这里将介绍在状态集合有限前提下三种常见的策略评估方法，它们分别是基于动态规划的方法、基于蒙特卡洛采样的方法和时序差分（temporal difference）法。

强化学习中的策略评估：动态规划

基于动态规划的价值函数更新：使用迭代的方法求解贝尔曼方程组

初始化V_π函数
循环
　　枚举$s \in S$
　　$V_\pi(s) \leftarrow \sum_{a\in A}\pi(s,a)\sum_{s'\in S}Pr(s'|s,a)\left[R(s,a,s')+\gamma V_\pi(s')\right]$

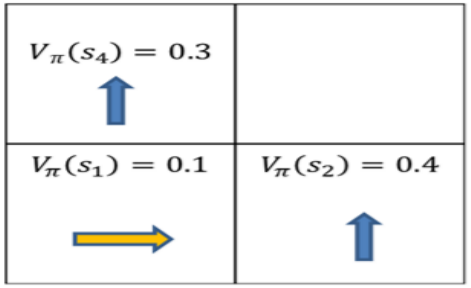
直到V_π收敛



更新$V_\pi(s_1)$的值：

$$q_\pi(s_1, 上) = 1 \times (0 + 0.99 \times 0.3) + 0 \times (0 + 0.99 \times 0.4)$$
$$+\ldots = 0.297$$

$$V_\pi(s_1) = 1 \times q_\pi(s_1, 上) + 0 \times q_\pi(s_1, 右) = 0.297$$



优化后策略

动态规划法的缺点：1) 智能主体需要事先知道状态转移概率;2) 无法处理状态集合大小无限的情况

强化学习中的策略评估：动态规划

基于蒙特卡洛采样的价值函数更新

选择不同的起始状态，按照当前策略π采样若干轨迹，记它们的集合为D

枚举$s \in S$

计算D中s每次出现时对应的反馈$G_1, G_2, \cdots, G_k$

$V_\pi(s) \leftarrow \frac{1}{k}\sum_{i=1}^{k} G_i$

假设按照当前策略可样得到以下两条轨迹

$(s_1, s_4, s_7, s_8, s_9)$

(s_1, s_2, s_3, s_d)

s_1对应的反馈值分别为

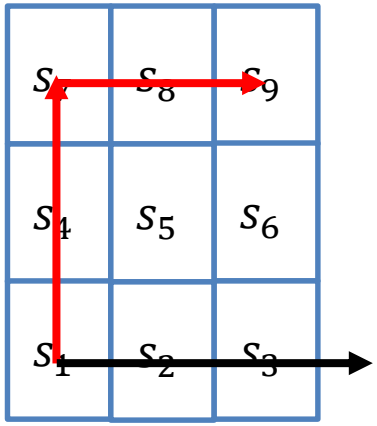
$$0 + \gamma \times 0 + \cdots + \gamma^3 \times 1 = 0.970$$

$$0 + \gamma \times 0 + \gamma^2 \times (-1) = -0.980$$

因此估计

$$V(s_1) = \frac{1}{2}(0.970 - 0.980) = -0.005$$

如果是确定的策略，每个起点只会产生一种轨迹



根据数理统计的知识，期望可以通过样本均值来估计的，这正是蒙特卡洛方法（Monte-Carlo method）的核心思想

强化学习中的策略评估：动态规划

基于蒙特卡洛采样的价值函数更新

选择不同的起始状态，按照当前策略π采样若干轨迹，记它们的集合为D
枚举$s \in S$
　　计算D中s每次出现时对应的反馈$G_1, G_2, \cdots, G_k$
　　$V_\pi(s) \leftarrow \frac{1}{k}\sum_{i=1}^{k} G_i$

按照这样的思路可使用蒙特卡洛方法来进行策略评估：给定状态s，从该状态出发不断采样后续状态，得到不同的采样序列。通过这些采样序列来分别计算状态s的回报值，对这些回报值取均值，作为对状态s价值函数的估计，从而避免对状态转移概率的依赖。

强化学习中的策略评估：动态规划

基于蒙特卡洛采样的价值函数更新

选择不同的起始状态，按照当前策略π采样若干轨迹，记它们的集合为D

枚举$s \in S$

计算D中s每次出现时对应的反馈$G_1, G_2, \cdots, G_k$

$V_\pi(s) \leftarrow \frac{1}{k}\sum_{i=1}^{k} G_i$

蒙特卡洛采样法的优点

- 智能主体不必知道状态转移概率
- 容易扩展到无限状态集合的问题中

蒙特卡洛采样法的缺点

- 状态集合比较大时，一个状态在轨迹可能非常稀疏，不利于估计期望
- 在实际问题中，最终反馈需要在终止状态才能知晓，导致反馈周期较长

强化学习中的策略评估：动态规划

基于时序差分（Temporal Difference）的价值函数更新

初始化V_π函数
循环
　　初始化s为初始状态
　　循环
　　　　$a \sim \pi(s,\cdot)$
　　　　执行动作a，观察奖励R和下一个状态s'
　　　　更新$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s') - V_\pi(s)]$
　　　　$s \leftarrow s'$
　　直到s是终止状态
直到V_π收敛

时序差分法可以看作蒙特卡罗方法和动态规划方法的有机结合。时序差分算法与蒙特卡洛方法相似之处在于，时序差分方法从实际经验中获取信息，无需提前获知环境模型的全部信息。时序差分算法与动态规划方法的相似之处在于，时序差分方法能够利用前序已知信息来进行在线实时学习，无需等到整个片段结束（终止状态抵达）再进行价值函数的更新。

强化学习中的策略评估：动态规划

基于时序差分（Temporal Difference）的价值函数更新

初始化V_π函数
循环
　　初始化s为初始状态
　　循环
　　　　$a \sim \pi(s,\cdot)$
　　　　执行动作a，观察奖励R和下一个状态s'
　　　　更新$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s') - V_\pi(s)]$
　　　　$s \leftarrow s'$
　　直到s是终止状态
直到V_π收敛

- 更新$V_\pi(s)$的值：$V_\pi(s) \leftarrow (1-\alpha)V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s')]$

过去的价值函数值　　**学习得到的价值函数值**

动态规划法根据贝尔曼方程迭代更新价值函数，要求算法事先知道状态之间的转移概率，这往往是不现实的。为了解决这个问题，时序差分法借鉴蒙特卡洛法思想，通过采样a和s'来估计计算$V_\pi(s)$

强化学习中的策略评估：动态规划

基于时序差分（Temporal Difference）的价值函数更新

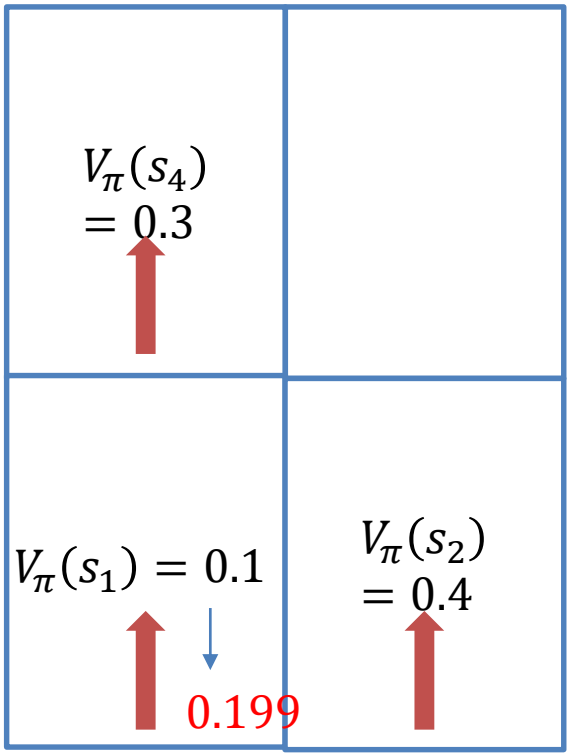
初始化V_π函数
循环
　　初始化s为初始状态
　　循环
　　　　$a \sim \pi(s,\cdot)$
　　　　执行动作a，观察奖励R和下一个状态s'
　　　　更新$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s') - V_\pi(s)]$
　　　　$s \leftarrow s'$
　　直到s是终止状态
直到V_π收敛

时序差分法和蒙特卡洛法都是通过采样若干个片段来进行价值函数更新的，但是时序差分法并非使用一个片段中的终止状态所提供的实际回报值来估计价值函数，而是根据下一个状态的价值函数来估计，这样就克服了采样轨迹的稀疏性可能带来样本方差较大的不足问题，同时也缩短了反馈周期。

强化学习中的策略评估：动态规划

基于时序差分（Temporal Difference）的价值函数更新

初始化V_π函数
循环
　　初始化s为初始状态
　　循环
　　$a \sim \pi(s,\cdot)$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s') - V_\pi(s)]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到V_π收敛

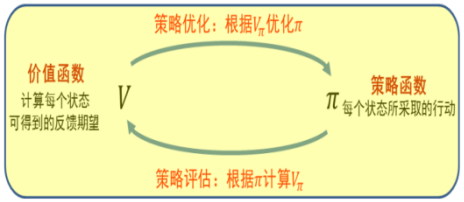


假设$\alpha = 0.5$，更新$V_\pi(s_1)$的值：
从$\pi(s_1,\cdot)$ 中采样得到动作$a =$ 上
从$Pr(\cdot\,|s_1, 上)$中采样得到下一步状态$s' = s_4$

$$V_\pi(s_1) \leftarrow V_\pi(s_1) + \alpha[R(s_1, 上, s_4) + \gamma V_\pi(s_4) - V_\pi(s_1)]$$
$$= 0.1 + 0.5 \times [0 + 0.99 \times 0.3 - 0.1] = 0.199$$

在对片段进行采样的同时，不断以上述方法更新当前状态的价值函数，不断迭代直到价值函数收敛为止。

强化学习中的策略评估：Q-learning



初始化q_π函数
循环
　　初始化s为初始状态
　　循环
　　~~$a \sim \pi(s,)$~~ ➡ $a = \text{argmax}_{a'} q_\pi(s,a')$
　　执行动作a，观察奖励R和下一个状态s'
　　~~更新$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R + \gamma V_\pi(s') - V_\pi(s)]$~~
　　➡ 更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

先前的策略优化：$\pi'(s) = \text{argmax}_a q_\pi(s,a)$

$$q_\pi(s,a) \leftarrow (1-\alpha) q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a')\right]$$

Q学习中直接记录和更新动作-价值函数q_π而不是价值函数V_π，这是因为策略优化要求已知动作-价值函数q_π，如果算法仍然记录价值函数V_π，在不知道状态转移概率的情况下将无法求出q_π。于是，Q学习中，只有动作-价值函数（即q函数）参与计算。

强化学习中的策略评估：Q-learning

s_d	s_7	s_8	s_9
	s_4	s_5	s_6
	s_1	s_2	s_3

初始化q_π函数
循环
　初始化s为初始状态
　循环
　　$a = \text{argmax}_{a'} q_\pi(s, a')$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　直到s是终止状态
直到q_π收敛

初始化q_π函数

在右图中，$^a/_b$表示$q_\pi(s, 上) = a, q_\pi(s, 右) = b$

所有终止状态的q函数值设为$^0/_0$，其余状态可随机初始化，此处设$^{0.2}/_0$

$^0/_0$	$^{0.2}/_0$	$^{0.2}/_0$	$^0/_0$
	$^{0.2}/_0$	$^{0.2}/_0$	$^{0.2}/_0$
	$^{0.2}/_0$	$^{0.2}/_0$	$^{0.2}/_0$

初始化s，s的值在右图中用黑框框出

强化学习中的策略评估：Q-learning

s_d	s_7	s_8	s_9
	s_4	s_5	s_6
	s_1	s_2	s_3

初始化q_π函数
循环
　　初始化s为初始状态
　　循环
　　$a = \text{argmax}_{a'} q_\pi(s, a')$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

$a = \text{argmax}_{a'} q_\pi(s_1, a') =$ 上

$R = 0,\ s' = s_4$

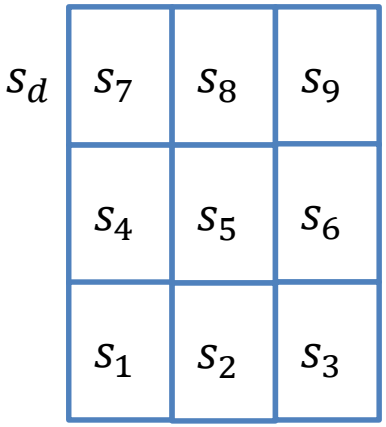
$q_\pi(s_1, \text{上}) \leftarrow 0.2 + 0.5 \times [0 + 0.99 \times \max\{0, 0.2\} - 0.2]$

$= 0.199$

$s \leftarrow s_4$

$0/0$	$0.2/0$	$0.2/0$	$0/0$
	$0.2/0$	$0.2/0$	$0.2/0$
	$0.199/0$	$0.2/0$	$0.2/0$

强化学习中的策略评估：Q-learning



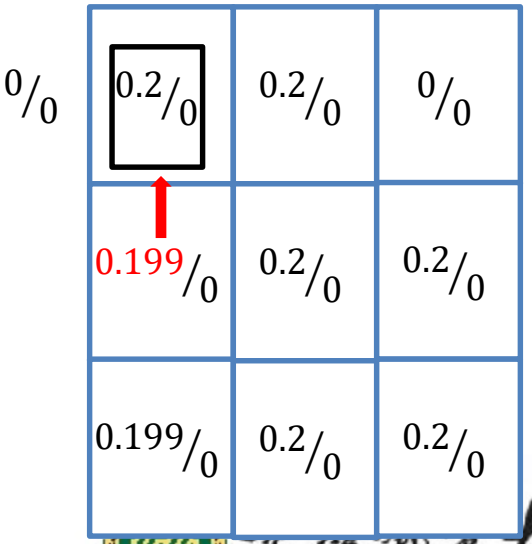
初始化q_π函数
循环
　　初始化s
　　循环
　　$a = \text{argmax}_{a'} q_\pi(s, a')$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

$a = \text{argmax}_{a'} q_\pi(s_4, a') =$ 上

$R = 0$，$s' = s_7$

$q_\pi(s_4, \text{上}) \leftarrow 0.2 + 0.5 \times [0 + 0.99 \times \max\{0, 0.2\} - 0.2] = 0.199$

$s \leftarrow s_7$



强化学习中的策略评估：Q-learning

s_d	s_7	s_8	s_9
	s_4	s_5	s_6
	s_1	s_2	s_3

初始化q_π函数
循环
　　初始化s
　　循环
　　$a = \text{argmax}_{a'} q_\pi(s, a')$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

$a = \text{argmax}_{a'} q_\pi(s_7, a') = $ 上
$R = -1$，$s' = s_d$
$q_\pi(s_7, \text{上}) \leftarrow 0.2 + 0.5 \times [-1 + 0.99 \times \max\{0,0\} - 0.2] = -0.4$
$s \leftarrow s_d$

因为s_d是终止状态，因此一个片段（episode）结束

$-0.4/0$	$0.2/0$	$0/0$
$0.199/0$	$0.2/0$	$0.2/0$
$0.199/0$	$0.2/0$	$0.2/0$

$0/0$

强化学习中的策略评估：Q-learning

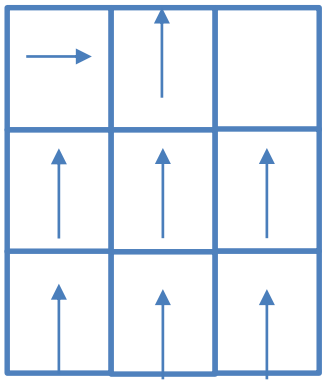
q函数

$-0.4/0$	$0.2/0$	$0/0$
$0.199/0$	$0.2/0$	$0.2/0$
$0.199/0$	$0.2/0$	$0.2/0$

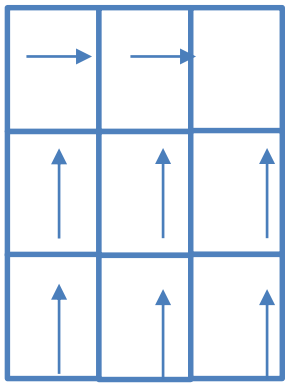
$-0.4/0.099$	$-0.4/0$	$0/0$
$0.100/0$	$0.2/0$	$0.2/0$
$0.198/0$	$0.2/0$	$0.2/0$

$-0.4/0.050$	$-0.4/0.5$	$0/0$
$0.099/0$	$0.2/0$	$0.2/0$
$0.148/0$	$0.2/0$	$0.2/0$

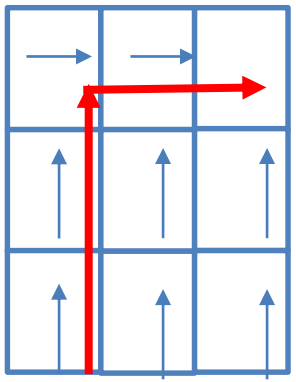
策略



第一个片段后



第二个片段后

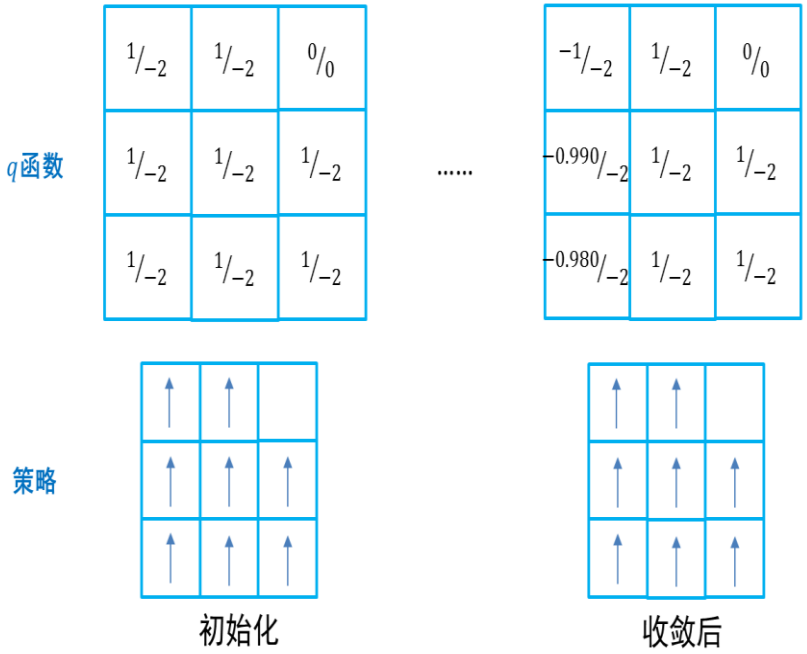


第三个片段后

强化学习中的策略评估：Q-learning

初始q函数

$^{1}/_{-2}$	$^{1}/_{-2}$	$^{0}/_{0}$
$^{1}/_{-2}$	$^{1}/_{-2}$	$^{1}/_{-2}$
$^{1}/_{-2}$	$^{1}/_{-2}$	$^{1}/_{-2}$

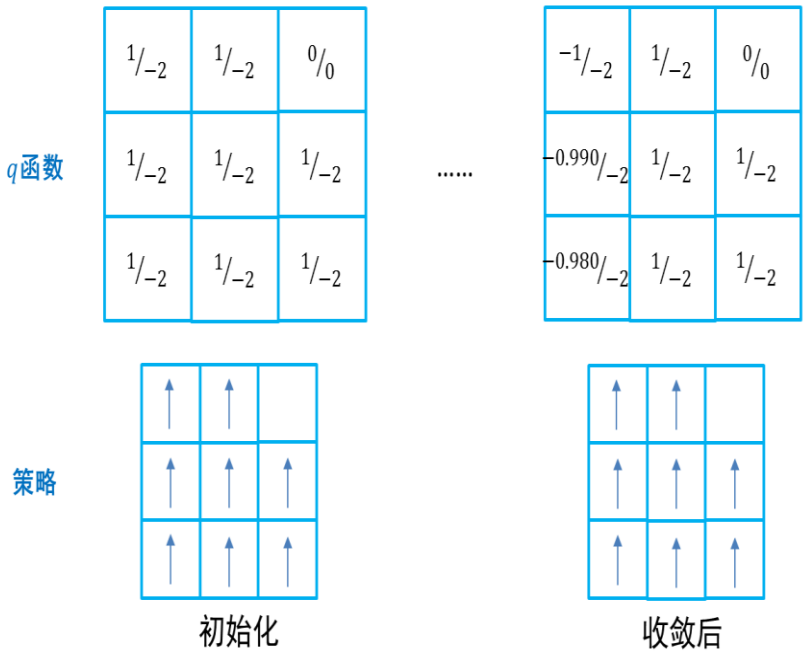


- 如果q函数的初始化为$^{1}/_{-2}$，在模型收敛后，策略仍无法使得智能主体找到目标状态
- 原因在于除了终止状态，对于每个状态，由于向上一个方格可得到1回报、而向右一个方格仅得到-2回报，因此智能体更倾向于向上移动一个方格而非向右一个方格。在这样初始策略下，即使在Q学习算法收敛以后，所得到的策略仍然不能使得机器人抵达目标状态s_9，而是沿着轨迹(s_1, s_4, s_7, s_d)导致机器人越界而被损坏。

强化学习中的策略评估：Q-learning

初始q函数

$^{1}/_{-2}$	$^{1}/_{-2}$	$^{0}/_{0}$
$^{1}/_{-2}$	$^{1}/_{-2}$	$^{1}/_{-2}$
$^{1}/_{-2}$	$^{1}/_{-2}$	$^{1}/_{-2}$



这一问题出现的原因概括来说就是$q_\pi(\cdot,右)$的初始值太小，导致机器人被损坏产生的-1奖励不足以推动智能体来改变策略。既然外部刺激不足以使机器人尝试新的策略，那么不妨从内部入手为智能体改变固有策略来添加一个探索的动力。

策略学习中探索（exploration）与利用（exploitation）的平衡

ϵ贪心（ϵ-greedy）策略

$$\epsilon\text{-}greedy_\pi(s) = \begin{cases} \text{argmax}_a q_\pi(s,a), & \text{以} 1-\epsilon \text{的概率} \\ \text{随机的} a \in A \ , & \text{以} \epsilon \text{的概率} \end{cases}$$

初始化q_π函数
循环
　　初始化s
　　循环

用ϵ贪心（ϵ-greedy）策略
代替$a = \text{argmax}_{a'} q_\pi(s,a')$

$a = \text{argmax}_{a'} q_\pi(s,a')$

执行动作a，观察奖励R和下一个状态s'

更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)\right]$

$s \leftarrow s'$

　　直到s是终止状态
直到q_π收敛

为此，在Q学习中引入探索（exploration）与利用（exploitation）机制。这一机制用用ϵ贪心（ϵ-greedy）策略来代替$a = \text{argmax}_{a'} q_\pi(s,a')$。用$\epsilon$贪心（$\epsilon$-greedy）策略定义如下：在状态$s$, 以$1-\epsilon$的概率来选择带来最大回报的动作，或者以$\epsilon$的概率来随机选择一个动作。

策略学习中探索（exploration）与利用（exploitation）的平衡

ϵ贪心（ϵ-greedy）策略

$$\epsilon\text{-}greedy_\pi(s)=\begin{cases}\text{argmax}_a q_\pi(s,a), & \text{以}1-\epsilon\text{的概率}\\ \text{随机的}a\in A\ , & \text{以}\epsilon\text{的概率}\end{cases}$$

初始化q_π函数
循环
　初始化s
　循环
　　用ϵ贪心（ϵ-greedy）策略
　　代替$a=\text{argmax}_{a'}q_\pi(s,a')$
　　$a=\text{argmax}_{a'}q_\pi(s,a')$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a)\leftarrow q_\pi(s,a)+\alpha\left[R+\gamma\max_{a'}q_\pi(s',a')-q_\pi(s,a)\right]$
　　$s\leftarrow s'$
　直到s是终止状态
直到q_π收敛

$-1/-2$	$1/-2$	$0/0$
$-0.990/-2$	$1/-2$	$1/-2$
$-0.980/-2$	$1/-2$	$1/-2$

ϵ 贪心策略的解释：大体上遵循最优策略的决定，偶尔（以 ϵ 的小概率）进行探索。如右图所示，如果能够偶尔在某些状态随机选择“向右移动一个方格”的动作，则可克服机器人无法走到终点s_9这一不足。

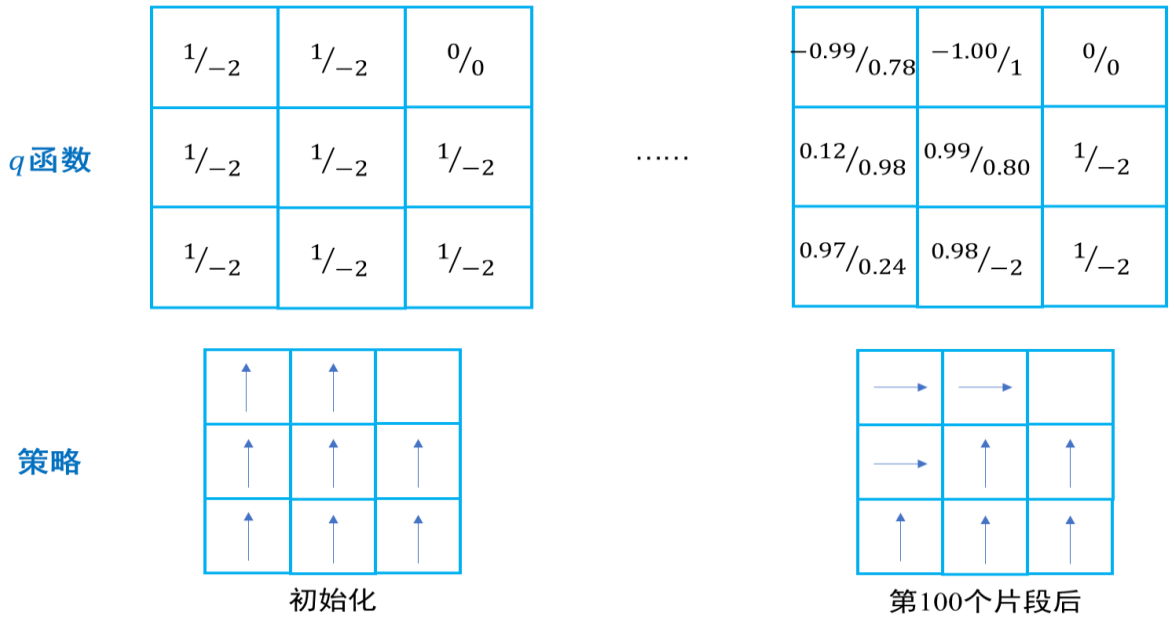
强化学习中的策略评估：使用ϵ贪心策略的Q学习

初始化q_π函数
循环
　　初始化s为初始状态
　　循环
　　采样$a \sim \epsilon - greedy_\pi(s)$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

采样策略与更新策略不同

- 将动作采样从“确定地选取最优动作”改为“按照ϵ贪心策略选取动作”
- 更新时仍保持用max操作选取最佳策略。像这样更新时的目标策略与采样策略不同的方法，叫做离策略（off-policy）方法

强化学习中的策略评估：使用ϵ贪心策略的Q学习



令$\epsilon = 0.1$，采用探索策略的Q学习，算法在执行了100个片段后，得到了经过轨迹$(s_1, s_4, s_5, s_8, s_9)$到达目标状态的策略。由于$\epsilon$贪心策略具有概率性，因此图中的结果并不是确定的，有可能经过100个片段仍不能得到一个合适的策略，且在图中也能发现存在还没有被访问过的状态s_3。但随着迭代次数的增加，算法探索到最优策略的可能性也会显著增加，通过适当增大参数ϵ的值也能够促进算法乐于探索。

基于策略的强化学习--策略梯度法

- 参数化策略方法（parameterized policy），该方法不再利用值函数，而是利用策略函数来选择动作，同时使用值函数来辅助策略函数参数的更新。
- 策略梯度法（policy gradient，PG）是参数化策略函数的一种常用算法。
- 根据策略类型的不同，PG可以分为：
  - ✓ 随机策略梯度（stochastic policy gradient，SPG）
  - ✓ 确定性策略梯度（deterministic policy gradient，DPG）

梯度上升方法

- 参数化策略不再是一个概率集合，而是一个可微的函数。策略函数 $\pi(a|s,\theta)$ 表示 t 时刻在状态 s 和参数 θ 下选择动作 a 的概率：

$$\pi(a|s,\boldsymbol{\theta}) = P\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}, \quad \boldsymbol{\theta} \in \mathrm{R}^{d'}$$

其中，θ 表示策略参数。参数化策略函数可以简记为 π_θ，这样 $\pi(a|s,\theta)$ 可以简记为 $\pi_\theta(a|s)$ 。

基于策略的强化学习--策略梯度法

- 参数化策略函数 $\pi(a|s,\theta)$ 可以看作概率密度函数，Agent按照该概率分布进行动作选择。
- 通常最直接的思想就是将目标函数定义为折扣回报的期望：

$$J(\theta) = \mathrm{E}_{\pi_\theta}[G] = \mathrm{E}_{\pi_\theta}\left[R_1 + \gamma R_2 + \gamma^2 R_2 + \cdots + \gamma^{n-1} R_n\right]$$

- 算法的目标是使得回报最大化，所以对参数采用梯度上升方法，该方法也被称为随机梯度上升（stochastic gradient-ascent，SGA）算法。

基于策略的强化学习--策略梯度法

- 基于**SGA**的参数更新方程为：

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla \hat{J}(\boldsymbol{\theta}), \quad \nabla \hat{J}(\boldsymbol{\theta}) \in \mathrm{R}^{d'}$$

其中，$\nabla \hat{J}(\boldsymbol{\theta})$ 为策略梯度函数的估计值，即近似策略梯度。

策略梯度法

➢**PG法的优点**

（1）平滑收敛：有很强的收敛性。

（2）处理连续动作空间任务：基于值函数的方法需要对比状态中的所有动作的价值，才能得到最优动作值函数，在处理大动作空间或连续状态动作空间任务时，难以实现。

➢**PG法的缺点**

（1）PG法通常只能收敛到局部最优解。

（2）PG法的易收敛性和学习过程平滑优势，都会使Agent尝试过多的无效探索，从而造成学习效率低，整体策略方差偏大，以及存在累积误差带来的过高估计问题。

REINFORCE

- **REINFORCE算法**采用**MC算法**来计算动作值函数，只考虑**Agent**在状态 S_t 下实际采取的动作 A_t：

$$\nabla J(\boldsymbol{\theta}) = \mathrm{E}_{S_t \sim \rho^{\pi_\theta}, A_t \sim \pi_\theta}\left[\nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta}) q_{\pi_\theta}(S_t, A_t)\right]$$
$$= \mathrm{E}_{S_t \sim \rho^{\pi_\theta}, A_t \sim \pi_\theta}\left[\nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta}) G_t\right] \qquad q_{\pi_\theta}(S_t, A_t) = \mathrm{E}_{\pi_\theta}\left[G_t \mid S_t, A_t\right]$$

- 由于采用**MC**算法，所以这是一种对策略梯度 $\nabla J(\theta)$ 的无偏估计。 **REINFORCE**算法的策略参数 θ 更新方程为：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \log \pi(A_t \mid S_t, \boldsymbol{\theta}_t) G_t$$

蒙特卡洛策略梯度法

该方法可以从理论上保证策略参数 θ 的收敛性，最大化 $J(\theta)$：

- 梯度增量 $\Delta\theta$ 正比于回报 G_t，使得策略参数 θ 向着能够产生最大回报的动作的方向更新；
- 梯度增量 $\Delta\theta$ 反比于迹向量，能够减少被频繁选择的动作。

蒙特卡洛策略梯度法

REINFORCE

输　入：

可微策略函数 $\pi(a \mid s, \boldsymbol{\theta})$，折扣因子 γ，学习率 $\{\alpha_k\}_{k=0}^{\infty} \in [0,1]$

初始化：

1. 初始化策略参数 $\boldsymbol{\theta} \in \mathbb{R}^{d'}$

2. **repeat** 对每个情节 $k = 0,1,2,\cdots$

3. 根据策略 $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$ 生成一个情节 $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

4. $G \leftarrow 0$

5. **repeat** 对每个时间步 $t = T-1, T-2, \ldots, 0$

6. $G \leftarrow \gamma G + R_{t+1}$

7. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_k \gamma^t \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta}) G$

行动者-评论家

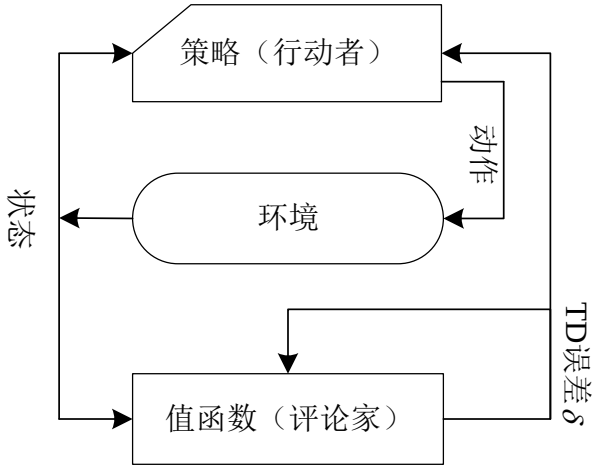
行动者-评论家（actor–critic，AC）算法将PG法和值函数相结合，同时学习策略和值函数，实现实时、在线地学习：

- 行动者（actor）依赖于评论家（critic）的值函数，利用PG法更新策略参数，学习（改进）策略 ；
- 评论家依赖于行动者策略得到的经验样本，更新值函数。

行动者-评论家

对于AC过程可以直观来理解：

- Agent根据任务的当前状态选择一个动作（基于当前策略或初始化策略）；
- 评论家根据当前状态-动作对，针对当前策略的表现打分；



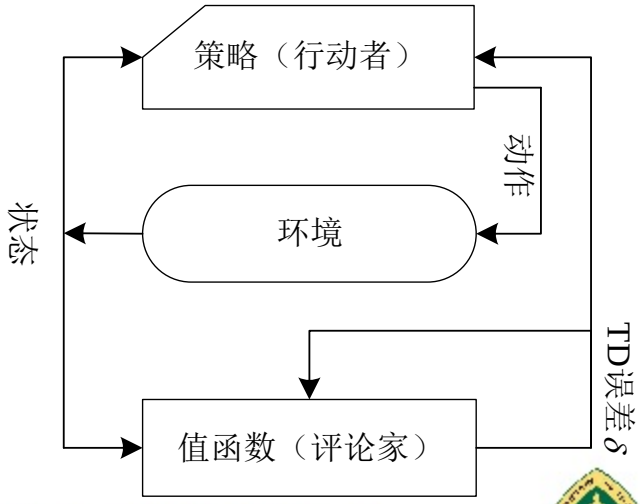
AC基本框架图

行动者-评论家

- 行动者依据评论家的打分，改进策略（调整策略参数）；
- 评论家根据环境返回的奖赏，改进策略打分方式（调整值函数）；
- 利用更新后的策略在下一状态处选择动作，重复以上过程。
- 最初行动者随机选择动作，评论家随机打分。但由于环境返回的奖赏，评论家的评分会越来越准确，行动者会选择到更好的动作。

行动者-评论家

- 行动者-评论家方法属于一种TD方法；
- 用两个独立的存储结构分别表示策略和值函数：
  - ✓ 行动者：表示策略函数，根据评估的TD误差选择动作；
  - ✓ 评论家：表示估计的值函数，通过计算值函数来评价行动者的好坏。



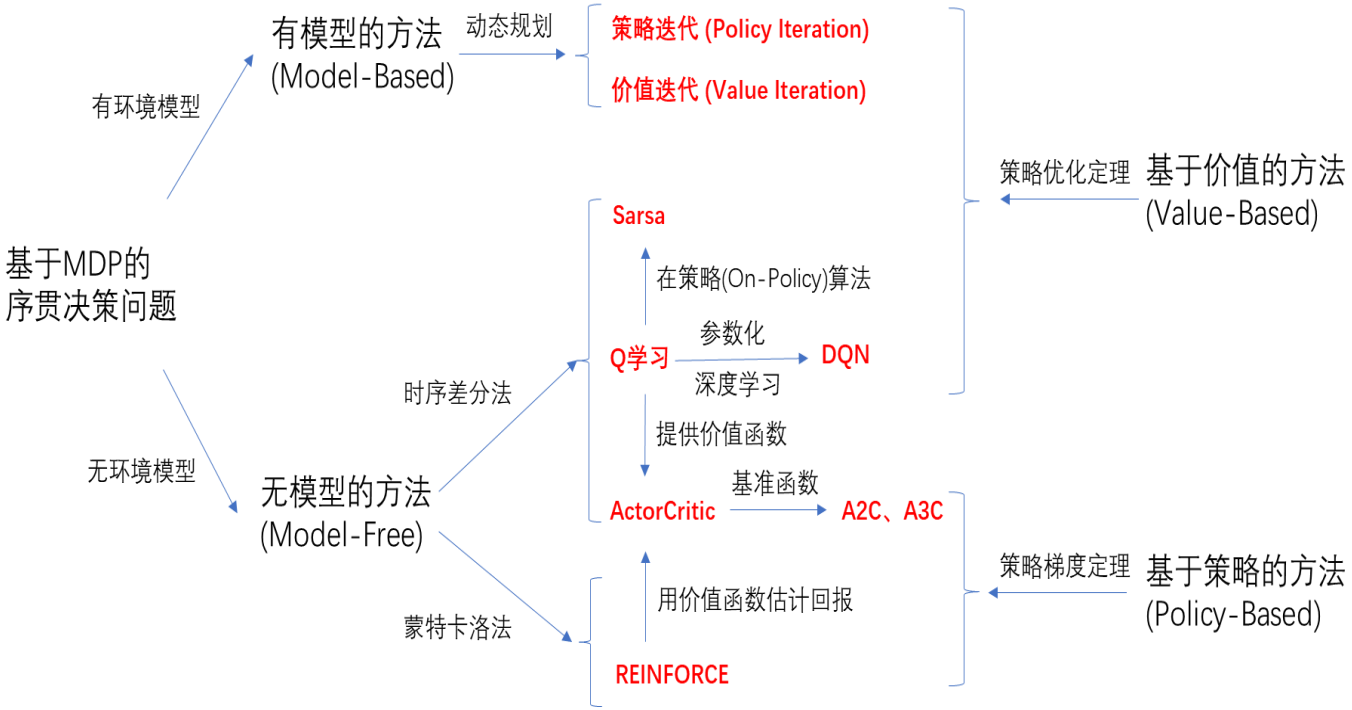
行动者-评论家

- **行动者-评论家TD误差：**
  - ✓ **TD误差表示的是当前状态的1-步回报或者*n*-步回报与其值函数之间的差值，TD误差的计算公式如下：**

$$\delta_t = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}) - V(s_t)$$

  - ✓ **TD误差可以用来评估当前正在被选择动作a_t的好与坏。当TD误差为正时，表明未来选择动作a_t的趋势应该加强；当TD误差为负时，表明未来选择动作a_t的趋势应该减弱。**

强化学习的分类



图中从两个角度对强化学习算法做了分类，其中依靠对环境（即马尔可夫随机过程）的先验知识或建模的算法称为基于模型（Model-based）的方法，反之称为无模型的方法（Model-free）；只对价值函数建模并利用策略优化定理求解的方法称为基于价值（Value-based）的方法，对策略函数建模并利用策略梯度定理求解的方法称为基于策略（Policy-based）的方法。

深度强化学习

- 深度强化学习将深度学习的感知能力与强化学习的决策能力相结合，利用深度神经网络有效识别高维数据的能力，使得强化学习算法在处理高维度状态空间任务中更加有效。
- **2013年DeepMind团队首次提出了将强化学习中的Q学习与深度神经网络中的卷积神经网络相结合的深度Q网络算法。**
- **2015年该团队对DQN算法进一步完善，使得DQN模型在Atari 2600的大部分游戏中能够取得超越人类玩家水平的成绩。**

深度强化学习

- 自DQN算法提出以来，深度强化学习逐步成为机器学习的研究热点，相关技术也广泛应用于游戏、机器人控制、自动驾驶、自然语言处理、计算机视觉等领域。在网络结构和算法理论方面也出现了大量的研究成果。

DQN算法

核心思想

- **DQN是一种经典的基于值函数的深度强化学习算法，它将卷积神经网CNN与Q-Learnig算法相结合，利用CNN对图像的强大表征能力，将视频帧数据视为强化学习中的状态输入网络，然后由网络输出离散的动作值函数，Agent再根据动作值函数选择对应的动作。**

用神经网络拟合（行动）价值函数：Deep Q-learning

- 状态数量太多时，有些状态可能始终无法采样到，因此对这些状态的q函数进行估计是很困难的
- 状态数量无限时，不可能用一张表（数组）来记录q函数的值

使用ϵ贪心策略的Q学习

初始化q_π函数
循环
　　初始化s为初始状态
　　循环
　　采样$a \sim \epsilon\text{-}greedy_\pi(s)$
　　执行动作a，观察奖励R和下一个状态s'
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)\right]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

思路：将q函数参数化（parametrize），用一个非线性回归模型来拟合q函数，例如（深度）神经网络

- 能够用有限的参数刻画无限的状态
- 由于回归函数的连续性，没有探索过的状态也可通过周围的状态来估计

用神经网络拟合（行动）价值函数：Deep Q-learning

用深度神经网络拟合q函数

初始化q_π函数的参数θ
循环
　　初始化s为初始状态
　　循环
　　　　采样$a \sim \epsilon\text{-}greedy_\pi(s;\theta)$
　　　　执行动作a，观察奖励R和下一个状态s'
　　　　损失函数$\mathrm{L}(\theta) = \frac{1}{2}\left[R + \gamma \max_{a'} q_\pi(s', a';\theta) - q_\pi(s, a;\theta)\right]^2$
　　　　根据梯度$\partial L(\theta)/\partial\theta$更新参数$\theta$
　　　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

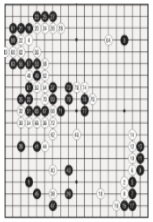
- 损失函数刻画了q的估计值$R + \gamma \max_{a'} q_\pi(s', a';\theta)$与当前值的平方误差
- 利用梯度下降法优化参数θ
- 如果用深度神经网络来拟合q函数，则算法称为**深度Q学习或者深度强化学习**

深度Q学习的应用实例: 围棋博弈

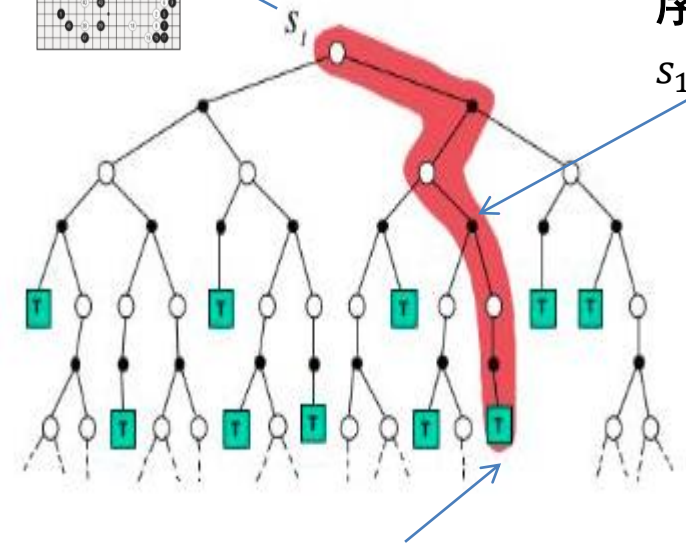
深度学习
有效编码黑白相间围棋的棋面

强化学习
助力做出长时间序贯决策

蒙特卡洛树搜索
支持从浩渺样本空间中采样



深度学习有效编码感知棋面



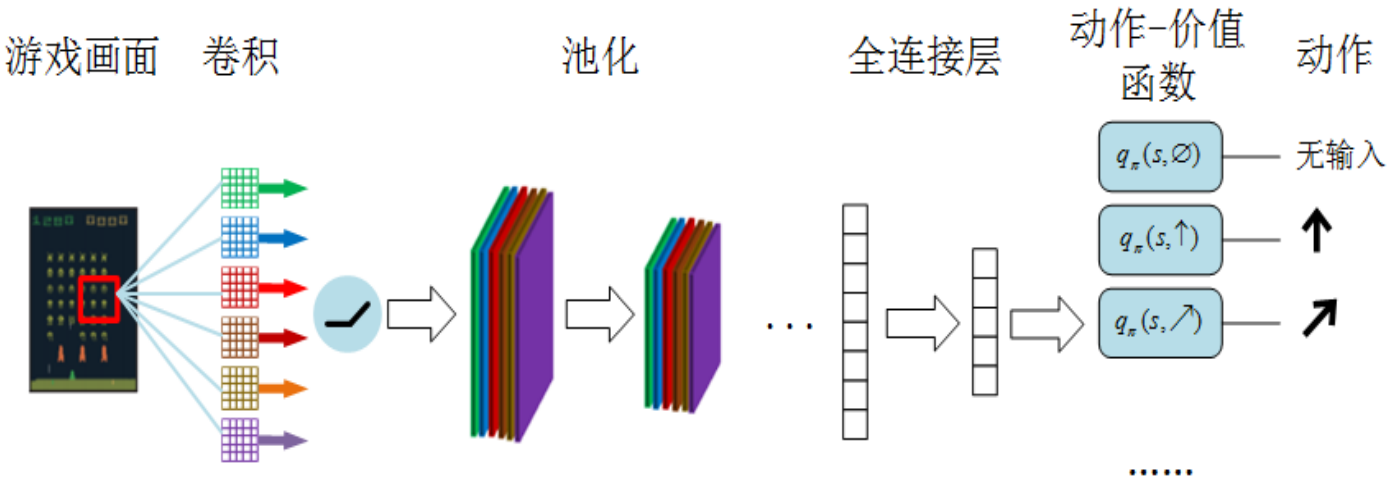
强化学习助力序贯决策行动

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots s_T \xrightarrow{a_T} Reward$$

蒙特卡洛搜索按照“探索与利用”原则采样序列来训练模型

$$s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} s_{t+2} \dots \xrightarrow{a_T} s_T$$

深度Q学习的应用实例: 雅达利游戏



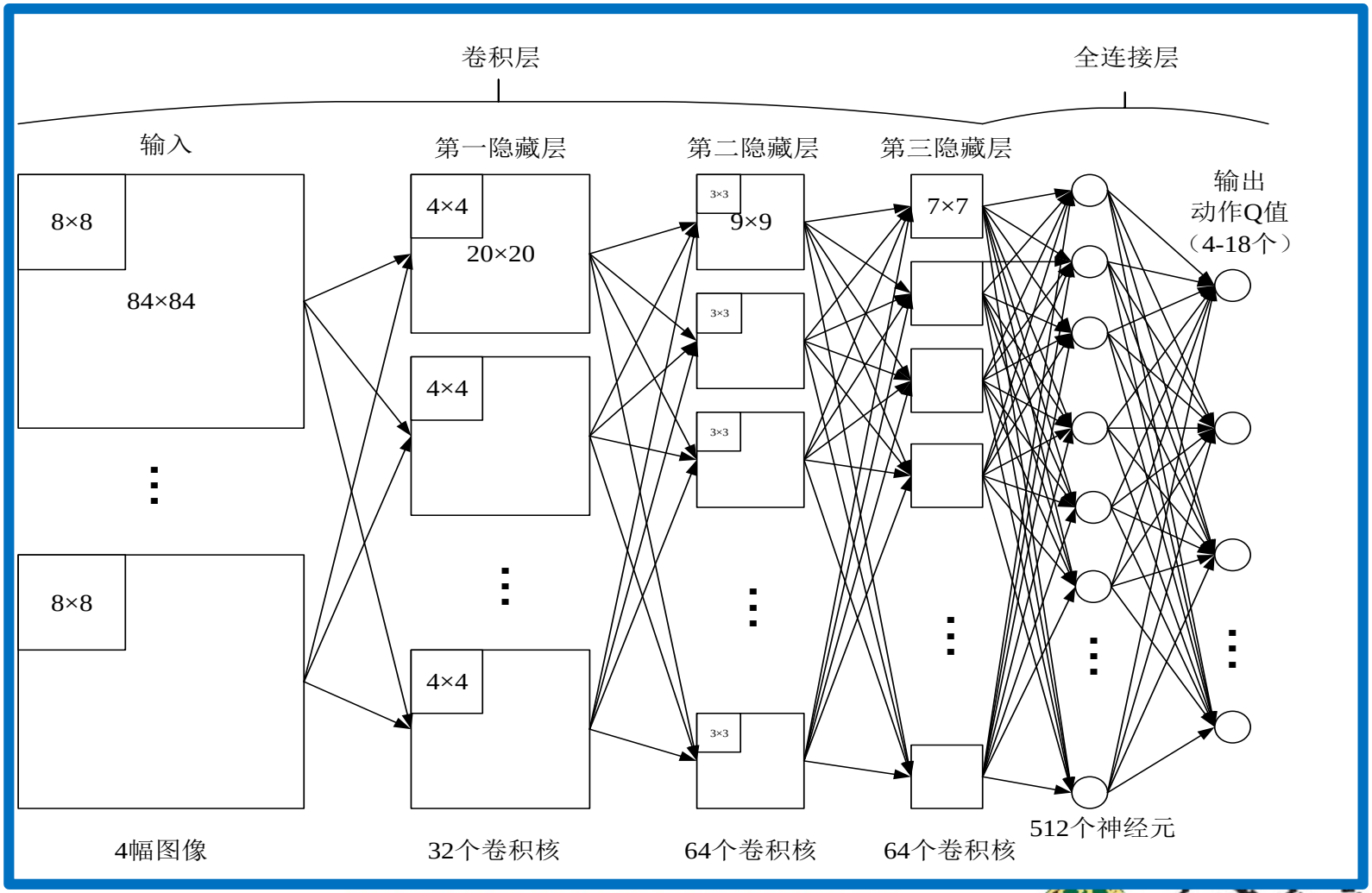
q函数的学习模型

Mnih, Volodymyr, et al, Human-level control through deep reinforcement learning, Nature 518.7540 (2015)

DQN算法

- **DQN算法常用于处理Atari游戏，但又不局限于此，它可以通过修改Q网络来处理不同的任务。例如：**
  - ✓ 如果输入为图像信息，则可以通过CNN构造Q网络；
  - ✓ 如果输入为序列数据，则可以通过RNN构建Q网络；
  - ✓ 如果要增加历史记忆能力，则可以通过结合CNN和长短期记忆模型（LSTM）来构建具有记忆能力的Q网络。

DQN算法


卷积层
全连接层
输入
第一隐藏层
第二隐藏层
第三隐藏层
8×8
84×84
8×8
4×4
20×20
4×4
4×4
3×3
9×9
3×3
3×3
3×3
7×7
输出
动作Q值
（4-18个）
4幅图像
32个卷积核
64个卷积核
64个卷积核
512个神经元

DQN训练算法

经验回放机制

- ✓ 在深度学习中，要求输入的样本数据满足独立同分布。而在强化学习任务中，样本间往往是关联的、非静态的，如果直接使用关联的数据进行模型训练，会导致模型难收敛、损失值持续波动等问题。
- ✓ **DQN算法引入经验回放机制：将每个时刻Agent与环境交互得到的经验迁移样本存储到经验池中，在执行数步之后，从经验池中随机取出批量（例如32个样本）大小的样本，作为离散数据输入神经网络，然后再采用小批量随机半梯度下降法（MBSGD）更新网络参数。**

DQN训练算法

经验回放机制

- ✓ 将状态s到s'之间产生的信号(s,a,r,s')或(s,a,r,s',T)称为经验迁移样本。其中，T为布尔值类型，表示新的状态s'是否为终止状态。经验回放机制采用随机采样的方式，既提高了数据的利用率，又去除了数据间的关联性、非静态分布等问题，使得网络模型更加稳定和高效。

DQN训练算法

DQN算法的优缺点：

- ✓ 优点在于算法通用性强，是一种端到端的处理方式，可为监督学习产生大量的样本。
- ✓ 其缺点在于：无法应用于连续动作控制，只能处理具有短时记忆的问题，无法处理需长时记忆的问题，且算法不一定收敛，需要仔细调参。