

人工智能与机器学习

第十五讲 强化学习

倪东 叶琦

工业控制研究所

杭州·浙江大学·2022

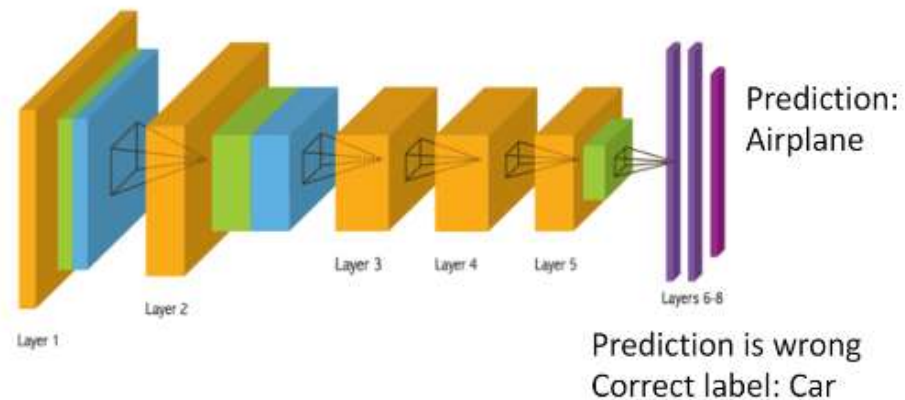
第十五讲 强化学习

- I. 简介
- II. 马尔可夫决策过程
- III. 策略迭代与价值迭代
- IV. 预测与控制
- V. 探索与利用
- VI. 深度强化学习



简介

- 监督学习
- 标记好的数据，独立同分布
- 智能体被告知数据的标签



airplane



automobile



bird



cat



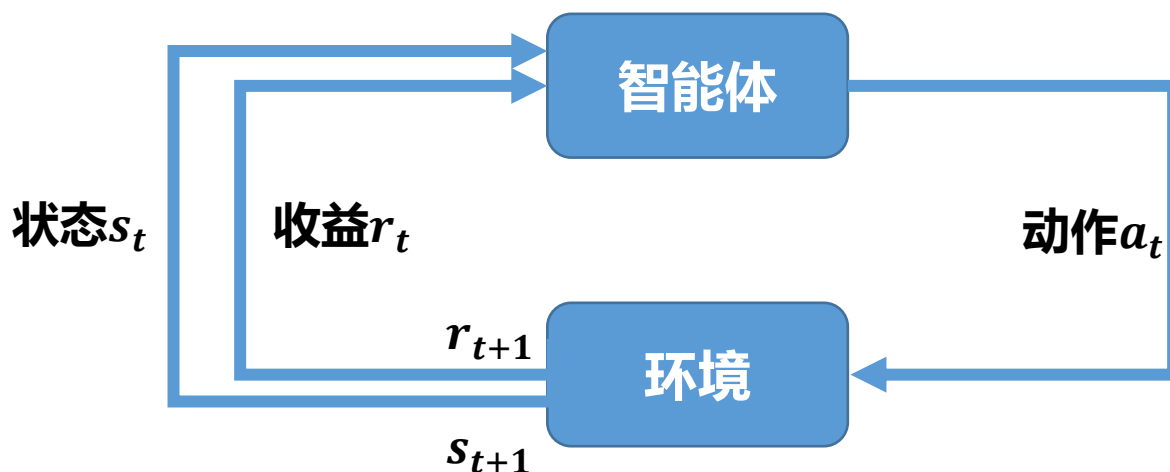
deer



dog



- 强化学习
- 人类是通过与环境交互来学习的，没有老师的指引，只有运动感知使其和外部环境直接联结，获得大量信息并从中获取因果关系、特定动作的后果
- 强化学习就是学习“做什么才能使收益信号最大化”
- 数据并非独立同分布，而是相互关联的序列数据
- 并不存在正确动作的标签

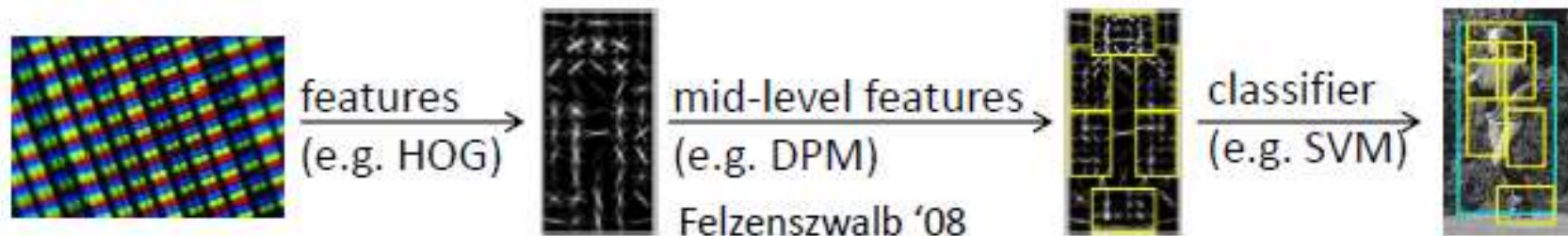


- **监督学习vs强化学习**
- **输入不再是独立同分布数据，而是序列数据**
- **监督学习从外部监督者提供的带标注数据中学习，强化学习中智能体不会被告知应该采取什么动作，而是去寻找在未来能够获得最多收益的动作**
- **通过试错探索，在探索和开发中寻求一种平衡**
- **没有监督，只有奖励信号**

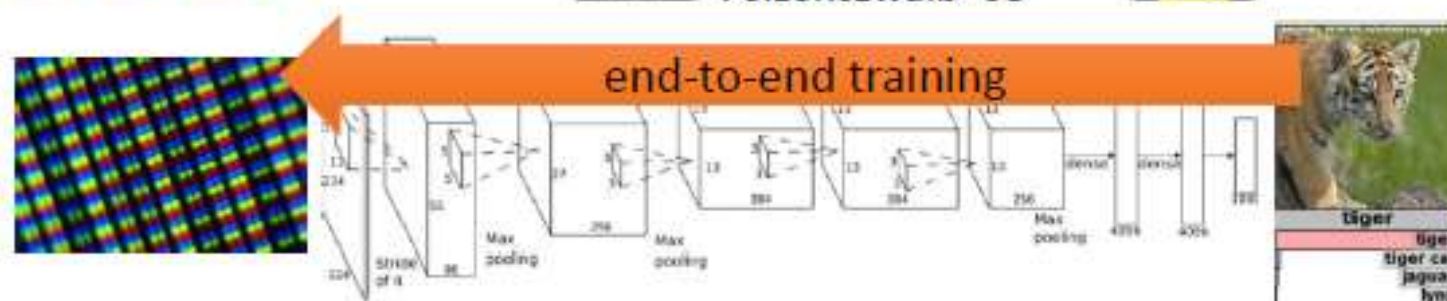


- 传统强化学习vs深度强化学习

传统视觉



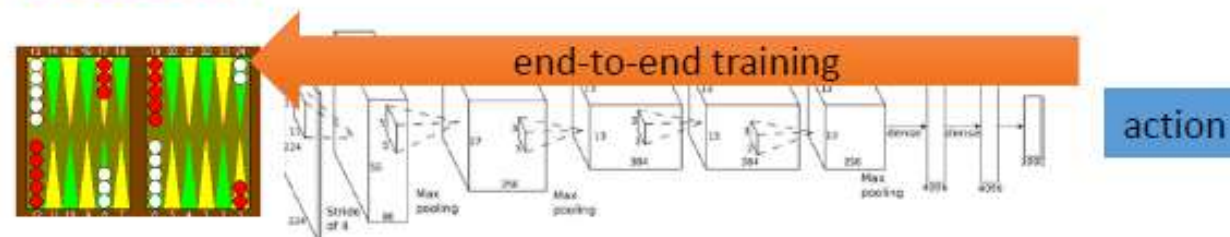
深度学习



传统强化学习

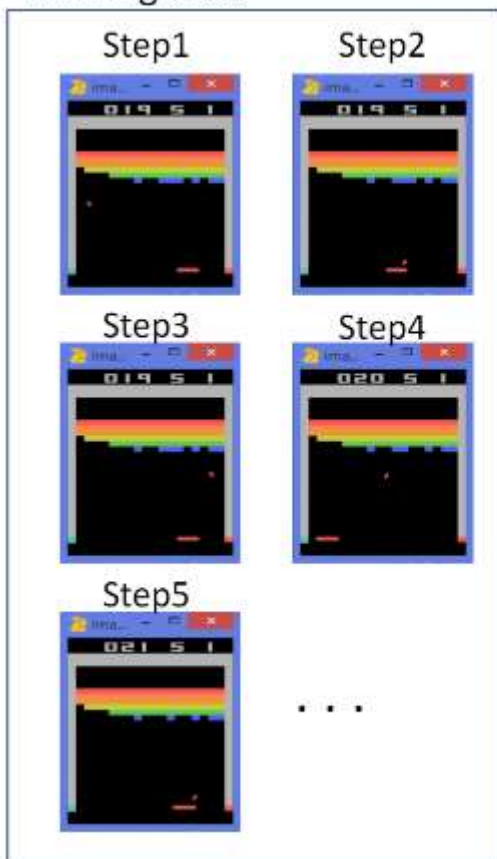


深度强化学习

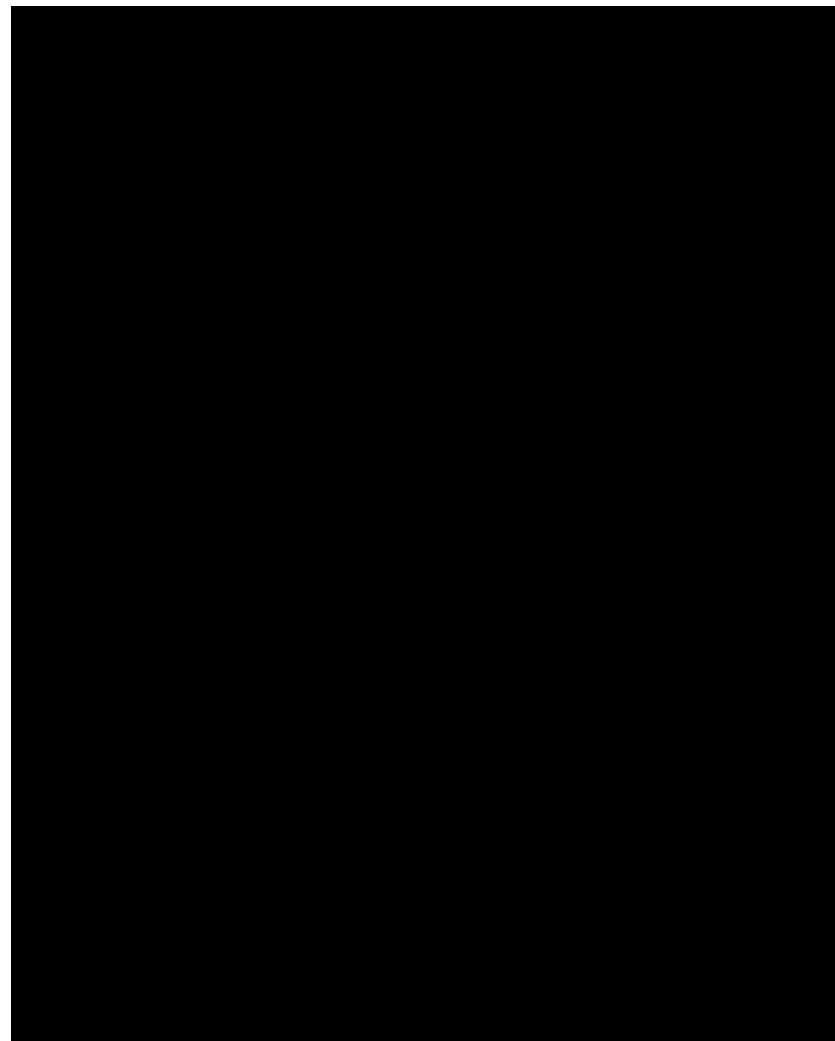


- Atari

Training data



Human playing?



<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

- 从深蓝到AlphaGo



<https://www.youtube.com/watch?v=ZlcZymAzifM>



<https://www.youtube.com/watch?v=8dMFJpEGNLQ>

第十五讲 强化学习

I. 简介

II. 马尔可夫决策过程

III. 策略迭代与价值迭代

IV. 预测与控制

V. 探索与利用

VI. 深度强化学习



- Markov Decision Process是一种通过交互式学习来实现目标的理论框架
- 智能体 *agent*: 进行学习及实施决策的机器
- 环境 *env*: 智能体之外所有与其互相作用的事物
- 每个离散时刻 $t = 0, 1, 2, 3 \dots$, 智能体与环境发生交互, 智能体观察到环境状态的某种特征表达 $s_t \in \mathcal{S}$, 并且在此基础上选择一个动作 $a_t \in \mathcal{A}$, 下个时刻智能体接收到一个数值化的收益 $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$, 并进入一个新的状态 s_{t+1} , 从而MDP和智能体共同给出了一个序列或轨迹



安德雷·安德耶
维齐·马尔可夫
1856年6月14
日 - 1922年7
月20日), 俄
国数学家

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3 \dots$$

- 已知历史状态序列

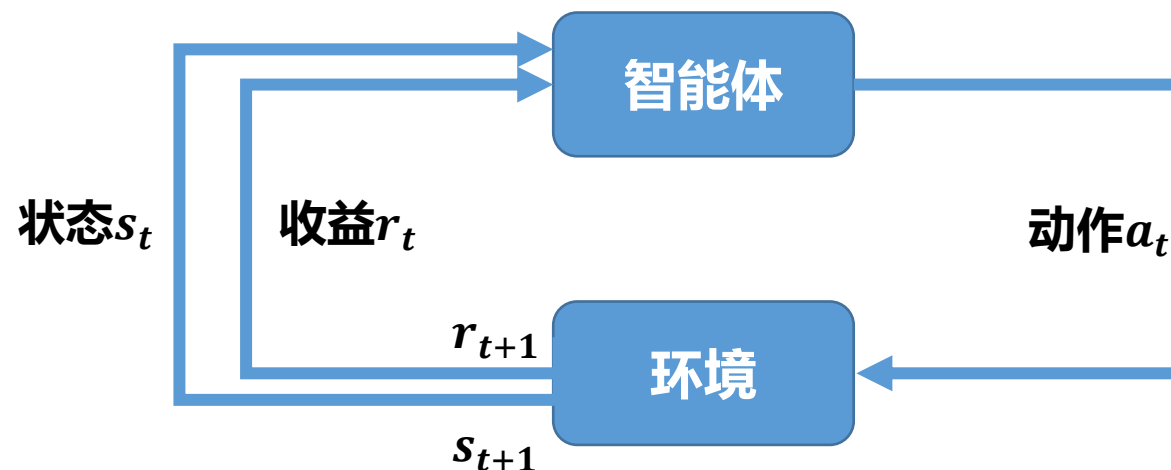
$$h_t = s_0, s_1, s_2, s_3 \dots s_t$$

- 状态 s_t 被认为具有马尔可夫性当且仅当

$$p(s_{t+1} | s_t) = p(s_{t+1} | h_t)$$

$$p(s_{t+1} | s_t, a_t) = p(s_{t+1} | h_t, a_t)$$

- 即 s_t, r_t 每个可能的值出现的概率只取决于前一个状态 s_{t-1} 和前一个动作 a_{t-1} , 并且与更早之前的状态和动作完全无关, 这样状态就被认为具有马尔可夫性

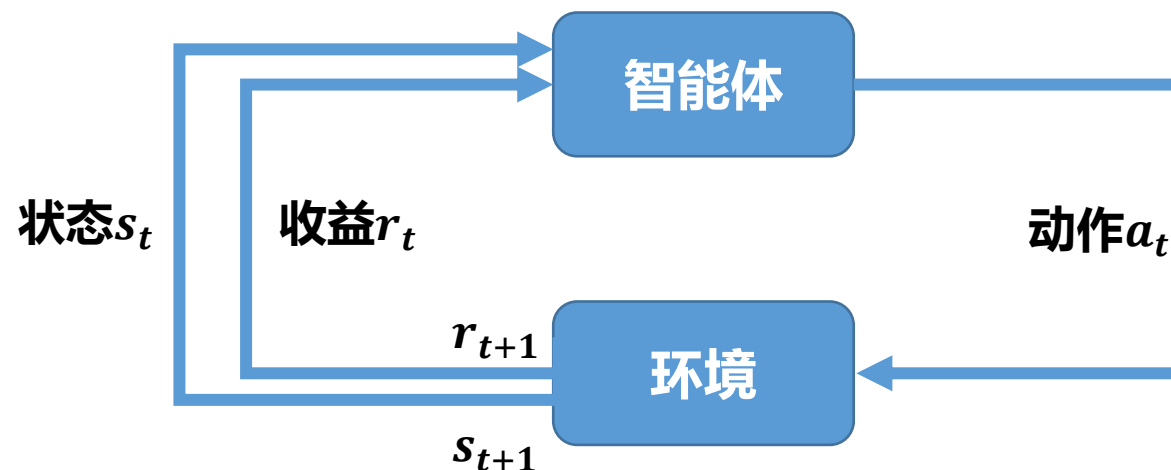


- 在 $t + 1$ 时刻, s_{t+1} 和 r_{t+1} 出现的概率为

$$p(s_{t+1}, r_{t+1} | s_t, a_t)$$

- 函数 p 定义了 MDP 的动态特性
- 动态函数 $p: \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ 为有 4 个参数的确定性函数

马尔可夫决策过程



- 函数 p 为每个 s_t 和 a_t 的选择指定了一个概率分布

$$\sum_{s_{t+1} \in \mathcal{S}} \sum_{r_{t+1} \in \mathcal{R}} p(s_{t+1}, r_{t+1} | s_t, a_t) = 1$$

- 函数 p 给出的概率完全刻画了环境的动态特性

- 从函数 p , 我们可以计算出关于环境的任何其他信息
- 状态转移概率 $p: \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

$$p(s_{t+1} | s_t, a_t) = \sum_{r_{t+1} \in \mathcal{R}} p(s_{t+1}, r_{t+1} | s_t, a_t)$$

- 状态-动作期望收益 $r: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

$$r(s_t, a_t) = \sum_{r_{t+1} \in \mathcal{R}} \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}, r_{t+1} | s_t, a_t)$$

- 目标和收益
- 在强化学习中个体的目的或目标被形式化为从环境传递到个体的特殊信号（称为奖励），在每个时间步骤，奖励是一个简单的数字 $r \in \mathbb{R}$
- 个体的目标是最大化其收到的总奖励。这意味着最大化不是立即奖励，而是长期累积奖励
- 例如国际象棋游戏个体应该仅仅因为实际获胜而获得奖励，而不是为了实现拿走对手的棋子或控制棋盘中心这样的子目标。如果实现这些类型的子目标得到奖励，那么个体可能会找到一种方法来实现它们而不实现真正的目标。例如，即使以失去游戏为代价，它也可能找到一种方法来获取对手的棋子。

- 目标和收益
- 在强化学习中，智能体寻求最大化预期收益，其中收益 g_t 被定义为奖励序列的某个特定函数。
- 在最简单的情况下，回报是奖励的总和：

$$g_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$$

- 当个体-环境交互自然地分解为子序列时，我们称之为episode，每个episode在称为终端状态的特殊状态结束，随后是重置到标准起始状态或从起始状态的标准分布的抽样。
- 回报公式 g_t 对于连续的任务是有问题的，因为最终时间步长将是 $T = \infty$ ，试图最大化的 g_t 本身是无限的，因此需要引入衰减因子 $\gamma \in [0, 1]$

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- 目标和收益
- 衰减因子 γ 决定了未来奖励的现值：未来收到的 $k + 1$ 个时间步骤的奖励价值仅为立即收到的 γ^k 倍
- 如果 $\gamma < 1$ ，只要奖励序列 $r_{t+1}, r_{t+2}, r_{t+3}, \dots, r_T$ 有界，则无限和 g_t 具有有限值
- 如果 $\gamma = 0$ ，个体是“短视”的，只关注最大化立即奖励，如果每个个体的行为恰好只影响即时奖励，而不影响未来的奖励，那么短视个体可以通过单独最大化每个即时奖励来最大化 g_t
- 当 γ 接近1时，回报目标更加强烈地考虑了未来的回报，个体变得更有远见



- 策略和价值函数
- 估计个体在给定状态下（或者在给定状态下执行给定动作）的好坏程度。 有多好根据未来的奖励来定义的
- 个体未来可能获得的回报取决于它将采取的行动，因此价值函数是根据特定的行为方式定义的，称为策略。
- 策略：从状态到选择每个可能动作的概率的映射

$$\pi(a_t|s_t)$$

- $\pi(a_t|s_t)$ 为当状态为 s_t 选择动作 a_t 的概率

- 策略和价值函数
- 策略 π 下状态 s 的价值函数表示为 $V_\pi(s)$,是从 s 开始并且遵循策略 π 的预期收益
- 对于 MDPs, $V_\pi(s)$ 定义为

$$V_\pi(s) = \mathbb{E}_\pi[g_t | s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s\right]$$

- 其中 \mathbb{E} 表示个体遵循策略 π 的随机变量的期望值,函数 $V_\pi(s)$ 是策略 π 的状态-价值函数
- 策略 π ,状态 s 下采取动作 a 的价值, 表示 $Q_\pi(s, a)$, 作为从状态 s 下采取动作 a 此后遵循策略 π 的预期收益

$$Q_\pi(s, a) = \mathbb{E}_\pi[g_t | s, a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s, a\right]$$

- 函数 $Q_\pi(s, a)$ 是策略 π 的动作值函数

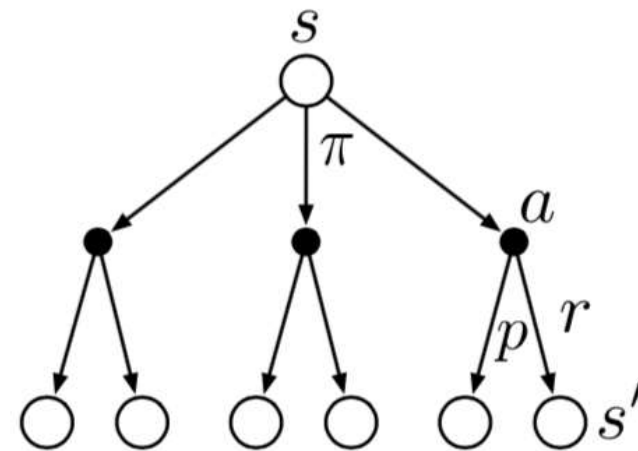
- 策略和价值函数
- $V_{\pi}(s)$ 和 $Q_{\pi}(s, a)$ 可以根据经验估计——如果个体遵循策略 π ，遇到的状态的次数接近无穷大，为每个 s 采取的每个 a 保留单独的平均值，那么这些平均值将收敛于 $Q_{\pi}(s, a)$ ，即蒙特卡洛方法
- 在强化学习和动态规划中，价值函数满足类似于我们为回报建立的递归关系

$$V_{\pi}(s_t) = \mathbb{E}_{\pi}[g_t|s_t] = \mathbb{E}_{\pi}[r_{t+1} + \gamma g_{t+1}|s_t]$$

$$= \sum_{a_t} \pi(a_t|s_t) \sum_{s_{t+1}} \sum_{r_{t+1}} p(s_{t+1}, r_{t+1}|s_t, a_t) [r_{t+1} + \gamma \mathbb{E}_{\pi}[g_{t+1}|s_{t+1}]]$$

$$= \sum_{a_t} \pi(a_t|s_t) \sum_{s_{t+1}} \sum_{r_{t+1}} p(s_{t+1}, r_{t+1}|s_t, a_t) [r_{t+1} + \gamma V_{\pi}(s_{t+1})]$$

- 策略和价值函数
- 贝尔曼方程表达了状态价值与下一个状态价值之间的关系。考虑从一个状态向可能的下一个状态的情况，如右图所示。每个空心圆表示状态，每个实心圆表示状态-动作对。
- 从状态 s 开始，顶部的根节点，个体可以采取基于其策略 π 的任何一组动作。对于每一个动作 a ，环境根据 $p(s_{t+1} | s_t, a_t)$ 可以响应后继状态中的一个 s_{t+1} 以及奖励 r 。



- 策略和价值函数
- 解决强化学习任务大概意味着要从长远的角度找到一个取得很大回报策略，利用价值函数对策略进行排序。如果策略 π 所有状态的预期返回值大于或等于策略 π' 的值，则该策略 π 被定义为优于或等于策略 π'
- 对所有 $s_t \in \mathcal{S}$ ，当且仅当 $V_\pi(s_t) \geq V_{\pi'}(s_t)$ ， $\pi \geq \pi'$ ，至少存在一个策略 π^* 不劣于其他所有策略，这就是最优策略，对应的最优状态价值函数为

$$V_*(s_t) = \max_{\pi} V_{\pi}(s_t)$$

- 对应动作值函数为

$$\begin{aligned} Q_*(s_t, a_t) &= \max_{\pi} Q_{\pi}(s_t, a_t) \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma V_*(s_{t+1}) | s_t, a_t] \end{aligned}$$



$$V_{\pi}(s) = \mathbb{E}_{\pi}[g_t | s] = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s]$$

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[g_t | s, a] = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s, a]$$

第十五讲 强化学习

- I. 简介
- II. 马尔可夫决策过程
- III. 策略迭代与价值迭代
- IV. 预测与控制
- V. 探索与利用
- VI. 深度强化学习



最优价值和最优策略

- 在给定的马尔科夫决策过程 $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ 下
- 如果有获得最优状态价值函数，或者最优状态-动作价值函数

$$V_*(s_t) = \max_{\pi} V_{\pi}(s_t)$$

$$Q_*(s_t, a_t) = \max_{\pi} Q_{\pi}(s_t, a_t)$$

- 设计如下的贪婪策略 (greedy)，就可以获得最优策略 (实现最优回报)

$$\pi_*(a_t|s_t) = \operatorname{argmax}_{a_t} Q_*(s_t, a_t) = \operatorname{argmax}_{a_t} [r_t + \gamma V_*(s_{t+1})]$$

- 然而**没有最优策略**，如何获得最优价值函数？



评估给定策略的价值函数

- 在求解最优价值函数之前，首先解决给定策略的价值函数计算问题
- 给定策略函数 π ，和马尔科夫决策过程，贝尔曼方程具有如下递推形式

$$\begin{aligned} V_{\pi}(s_t) &= \mathbb{E}_{\pi}[g_t | s_t] = \mathbb{E}_{\pi}[r_t + \gamma g_{t+1} | s_t] \\ &= \sum_{a_t} \pi(a_t | s_t) (r_t + \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \gamma \mathbb{E}_{\pi}[g_{t+1} | s_{t+1}]) \\ &= \sum_{a_t} \pi(a_t | s_t) (r_t + \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \gamma V_{\pi}(s_{t+1})) \end{aligned}$$

- 是否能解出价值函数 $V_{\pi}(s_t)$?



求解贝尔曼方程

- 用矩阵形式表示, 其中 $P_{s-s'}$ 的第 i 行 j 列的元素表示从状态 i 到 j 的概率 (与策略有关)

$$V_{\pi}(s_t) = \sum_{a_t} \pi(a_t|s_t)(r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_{\pi}(s_{t+1}))$$



$$V = R + \gamma P_{s-s'} V$$

- 可以看到贝尔曼方程是一个关于价值函数向量 V 的线性方程, 可以用最小二乘求解

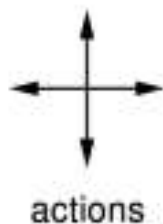
$$(I - \gamma P_{s-s'}) V = R$$

迭代策略评估

- 存在问题：当状态很大的时候，求解最小二乘非常耗时
- 迭代算法

$$V_{k+1} = R + \gamma P_{s-s'} V_k$$

- 简单证明：左右k趋向于无穷，上式即贝尔曼方程，取 V_∞ 作为当前策略的价值函数
- 该方法称为迭代策略评估 (Iterative policy evaluation)
- 案例：计算均匀随机策略的价值函数



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

$r = -1$
on all transitions

$$\pi(a_t|s_t) = 0.25$$

迭代策略评估

- 迭代过程:

v_k for the
Random Policy

$k = 0$

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

$k = 1$

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

$k = 2$

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

$k = 3$

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

$k = 10$

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

$k = \infty$

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

- 对于给定策略，可以通过策略评估得到其价值函数
- 然而，该策略（通常不是最优策略）的价值函数，是否**有助于获得最优策略**？
- 对给定策略的价值函数，采用贪婪策略

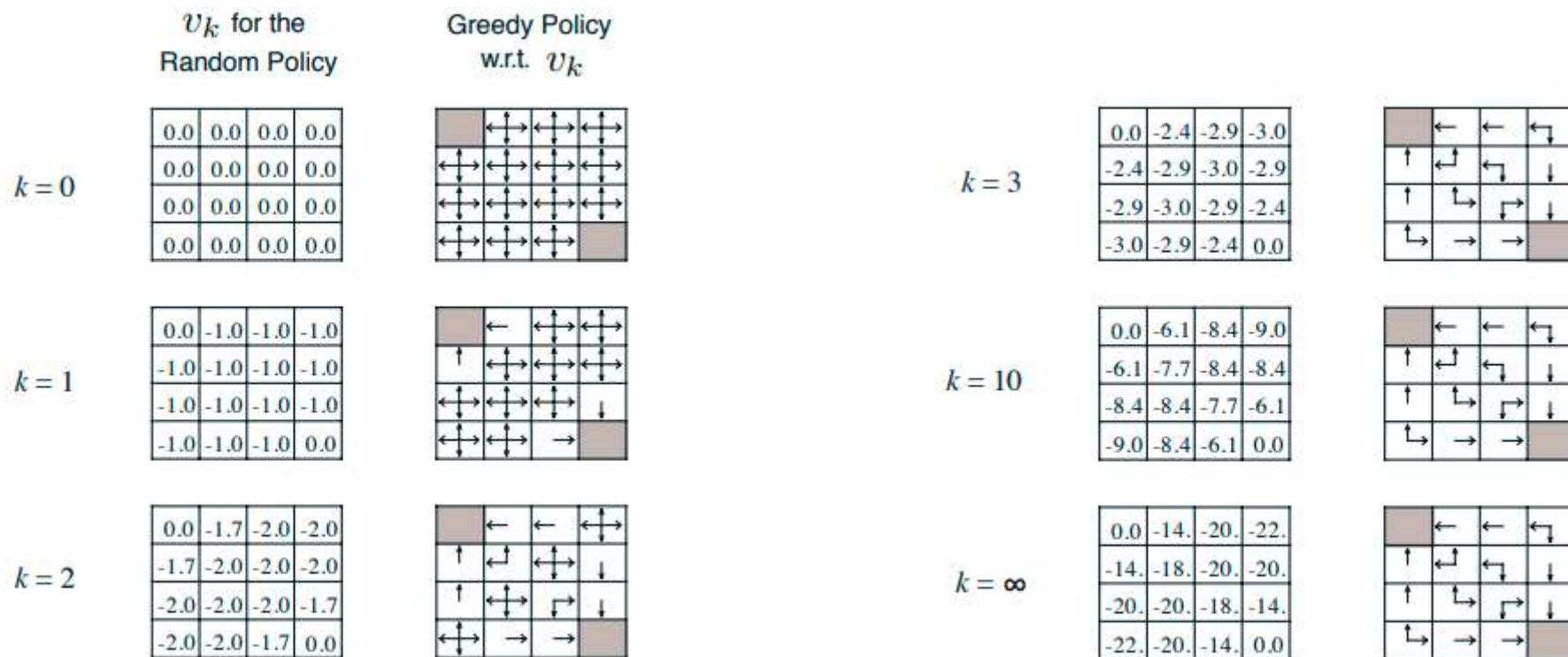
$$\pi'(a_t|s_t) = \operatorname{argmax}_{a_t} [r_t + \gamma V_{\pi}(s_{t+1})]$$

- 该贪婪策略一定会比原策略获得更高的回报，因为任意非贪婪策略下获得的回报，也就是价值函数的取值，必然不可能大于最大值

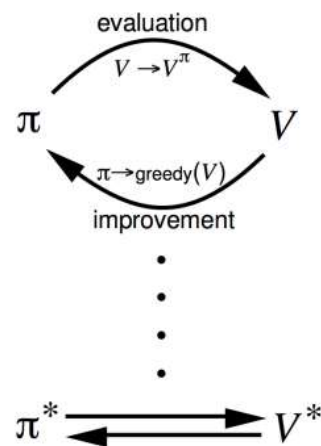
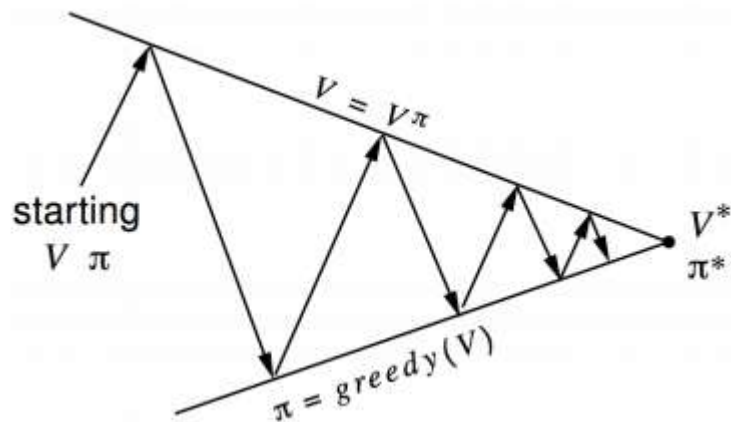
$$r_t + \gamma V_{\pi}(s_{t+1}) \leq \max_{a_t} [r_t + \gamma V_{\pi}(s_{t+1})]$$

- 如果小于，则策略改进会获得更大的回报；如果等于，则当前策略是最优策略

- 在均匀随机策略的价值函数上设计贪婪策略



- 从上述案例中可以看到：
 - 发现1**：一般策略的价值函数不同于最优策略的价值函数，但其对应的贪婪策略可以提高回报，甚至就是最优策略
 - 发现2**：在 k 不大的时候，也就是价值函数还未收敛的时候，其贪婪策略就已经可以获得更好的回报
- 基于发现1，设计策略迭代（Policy iteration），交替迭代：**给定价值函数，贪婪策略改进**




- 考虑到准确的价值函数并非需求。基于发现2，设计一般化的策略迭代（Generalized policy iteration）
- **策略迭代**：对当前策略，运行**k=1步**迭代策略评估（Iterative policy evaluation）更新价值函数
- **策略改进**：对当前价值函数，运行策略改进（Policy improvement），得到回报更好的策略
- **迭代停止条件**：策略改进步骤没有提升，比如**连续m步策略不变**

- 思路：既然贪婪策略是最优策略，是否可以直接通过策略评估计算策略的价值函数，就可以得到最优价值函数？
- 给定贪婪策略 π

$$V_{\pi}(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_{\pi}(s_{t+1}))$$

- 由于引入max函数，价值函数的迭代公式成为一个非线性方程
- 无法构造解析解，如何求解？
- 如何确定该式的解是最优价值函数？

- 注意：该式的形式是**贝尔曼最优方程**，贝尔曼最优方程的**解是最优价值函数**，对应最优策略，即最优价值函数上的贪婪策略

$$V_{\pi}(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_{\pi}(s_{t+1}))$$

$$V_*(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_*(s_{t+1}))$$

- 最优性原理**：一个过程的最优策略，对于以中间任意状态为始的过程，其后的策略构成最优策略

- 一个过程的最优策略，对于以中间任意状态为始的过程，其后的策略构成最优策略



- 如果知道 $V_*(s_{t+1})$ ，那么 $V_*(s_t)$ 可以代入贝尔曼最优方程获得

$$V_*(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_*(s_{t+1}))$$

- 可以看成是从最后的回报开始往前推，写成迭代形式

$$V_{k+1}(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_k(s_{t+1}))$$

- 该方法称为价值迭代 (Value iteration)

- 一个过程的最优策略，对于以中间任意状态为始的过程，其后的策略构成最优策略



- 如果知道 $V_*(s_{t+1})$ ，那么 $V_*(s_t)$ 可以代入贝尔曼最优方程获得

$$V_*(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_*(s_{t+1}))$$

- 可以看成是从最后的回报开始往前推，写成迭代形式

$$V_{k+1}(s_t) = \max_{a_t} (r_t + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \gamma V_k(s_{t+1}))$$

- 该方法称为价值迭代 (Value iteration)

- 最短路径价值迭代案例

g			

Problem

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

V_1

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

V_2

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

V_3

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

V_4

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

V_5

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

V_6

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

V_7

- 给定策略，可以用迭代策略评估获得该策略的价值函数，一般用于**预测**问题
- 基于价值函数，采用贪婪策略可以获得更高的回报，称为策略改进，一般用于**控制**问题
- 交替使用策略评估和策略改进，可以导出策略迭代算法PI，获得最优策略，实现**最优控制**
- 通过直接代入贪婪策略，可以导出贝尔曼最优方程VI，并导出价值迭代算法，获得最优价值函数，并导出**最优控制**
- 对比PI和VI
 - VI没有显式的建模策略，直接导出最优价值函数，其中间步骤的结果不能用于导出策略，必须等到价值函数收敛
 - PI显式建模策略，需要迭代最后收敛到最优价值函数，但中间步骤的策略也同样是可用的策略

第十五讲 强化学习

- I. 简介
- II. 马尔可夫决策过程
- III. 策略迭代与价值迭代
- IV. 预测与控制
- V. 探索与利用
- VI. 深度强化学习



- 在给定的马尔科夫决策过程 $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ 下
- 预测 (prediction)
 - 策略评估 (policy evaluation) :
 - 给定策略 π , 得到其状态值函数 V_π 或状态动作值函数 Q_π
- 控制 (control)
 - 策略提升 (policy improvement) :
 - 获得最优的状态值函数 V_π^* 或状态动作值函数 Q_π^* , 或者最优策略 π^*



- 存在问题：
- 动态规划方法：
 - 需要知道完备的环境知识（模型）
 - 绝大多数情况下，难以获取环境的真实状态转移模型
- 解决方案：
- 蒙特卡洛方法：
 - 通过与交互获取状态转移模型的采样数据
 - 利用平均的样本回报进行学习



- **蒙特卡洛方法：**
 - **经验：与环境交互的状态与反馈序列**
 - **无模型：不需要环境的状态转移模型**
 - **分幕式任务：完整的轨迹（替代动态规划中在线根据自举进行更新学习的方式）**
 - **应对于有限任务，任务有终止状态**



- 存在问题：
- 预测：策略评估 (policy evaluation)
 - 经验折扣回报：
 - $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$
 - 期望回报：
 - $V(s_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(s_{t+1})]$
 - 利用平均经验折扣回报代替模型求得的回报期望
 - $V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)} (G_t - V(s_t))$
 - 随着经验样本的增多，该平均值将收敛至期望

首次访问蒙特卡洛方法 (first-visit MC method)

- 预测对所有的回合中，第一次访问到状态 s 得到的回报求取均值
- 根据大数定理，当经验样本足够大时，该均值将逼近 $V_{\pi}(s)$ （或者 $Q_{\pi}(s, a)$ ）
- 具体方法：倒序遍历轨迹，找到每条轨迹中第一次遇到该状态时的位置，计算未来回报，更新均值

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

Input: a policy π to be evaluated

Initialize:

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

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

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

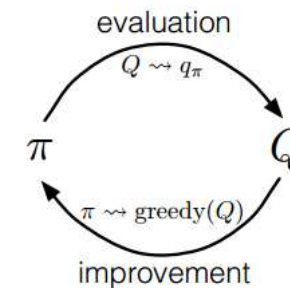
Append G to $Returns(S_t)$

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

- 策略提升 (policy improvement)
- 根据值函数，更新并提升策略至最优
- 当前值函数下的最优策略，贪婪地进行动作选择：

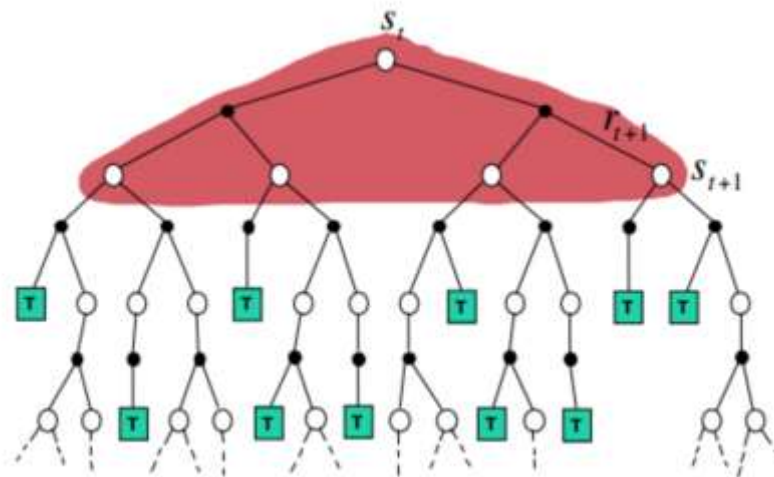
$$\pi(s) = \arg \max_a Q(s, a)$$

- 假设
 - 1、策略估计过程根据无限条轨迹收敛至回报的期望（误差阈值设计，值迭代）
 - 2、状态集合中所有状态都有可能被选中为轨迹的初始状态（on-policy, off-policy）



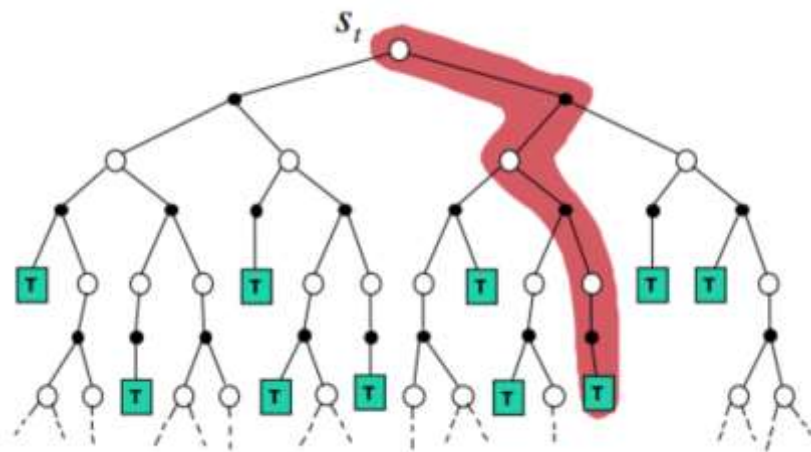
时序差分学习 (Temporal-Difference learning)

- 存在问题:
- 动态规划:
 - $V(s_t) \leftarrow \mathbb{E}_{\pi}[R_{t+1} + \gamma V(s_{t+1})]$
 - 模型已知, 利用自举的方法进行迭代更新 \Rightarrow 需要准确的环境模型



时序差分学习 (Temporal-Difference learning)

- 存在问题:
- 蒙特卡洛:
 - $V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$
 - 模型未知, 从轨迹经验中进行学习 \Rightarrow 需要完整轨迹, 难以处理长任务或连续型任务



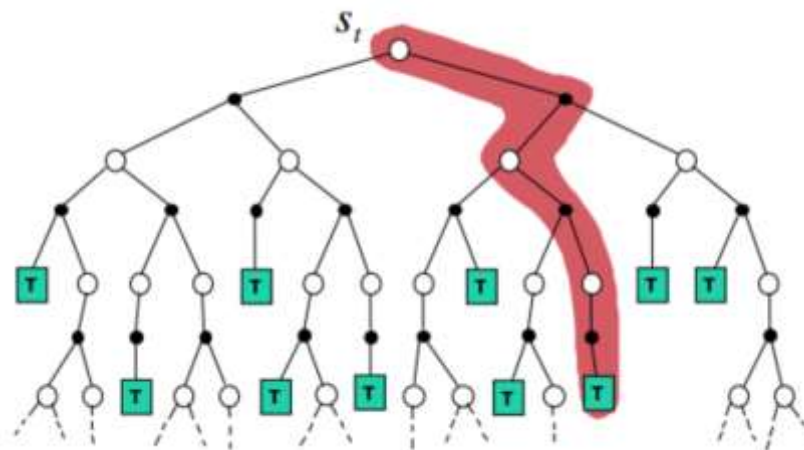
时序差分学习 (Temporal-Difference learning)

- 时序差分学习:
 - 模型未知
 - 根据交互经验进行学习
 - 在线进行实时更新，不依赖于完整轨迹
 - 根据学习得到的价值估计进行更新
- $V(s_t) \leftarrow V(s_t) + \alpha(\mathbf{G}_t - V(s_t))$
- $V(s_t) \leftarrow V(s_t) + \alpha(\mathbf{R}_{t+1} + V(s_{t+1}) - V(s_t))$



时序差分学习 (Temporal-Difference learning)

- 模型未知，在线进行实时更新，根据学习得到的价值估计进行更新
- $V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$
- TD target : $R_{t+1} + \gamma V(s_{t+1})$
- TD error : $\delta = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$



时序差分学习 (Temporal-Difference learning)

- 查表式时序差分学习

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

$A \leftarrow$ action given by π 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

时序差分学习 (Temporal-Difference learning)

- 蒙特卡洛 v.s. 时序差分 —— 优缺点对比
- 蒙特卡洛：
 - 高方差，无偏：依赖于完整的随机状态动作序列
 - 离线学习：需要完整轨迹
 - 只适用于分幕式场景
- 时序差分：
 - 低方差，有偏：仅依赖于当前状态动作
 - 在线学习：不需要完整轨迹
 - 适用于连续场景



第十五讲 强化学习

- I. 简介
- II. 马尔可夫决策过程
- III. 策略迭代与价值迭代
- IV. 预测与控制
- V. 探索与利用**
- VI. 深度强化学习



探索 vs. 利用

- **利用 (Exploitation) : 在已有认知下, 选择最优动作**
- **探索 (Exploration) : 选择新动作, 希望能获得更高的回报**
- **选择餐厅:**
 - **利用: 去最喜欢的餐厅**
 - **探索: 尝试新餐厅**
- **广告推送:**
 - **利用: 选择成功率最高的广告**
 - **探索: 尝试推送新广告**



On-policy vs. Off-policy

- On-policy学习方式：采数据的策略 = 待学习的策略
- Off-policy学习方式：采数据的策略 \neq 待学习的策略
- 待学习的策略 π ：目标策略，在训练过后成为最优策略
- 采数据的策略 μ ：行为策略，探索性更强，与环境交互生成数据



- 命名: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$
- 根据 s_{t+1} 选取 a_{t+1} 时采用的策略与当前时刻根据 s_t 选取 a_t 的策略一致
- 若 s_{t+1} 为终止态, 则 Q 为0
 - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
- 策略根据 ϵ -greedy原则进行探索

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s, a)$, for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

Initialize S

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

Loop for each step of episode:

Take action A , observe R, S'

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

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

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

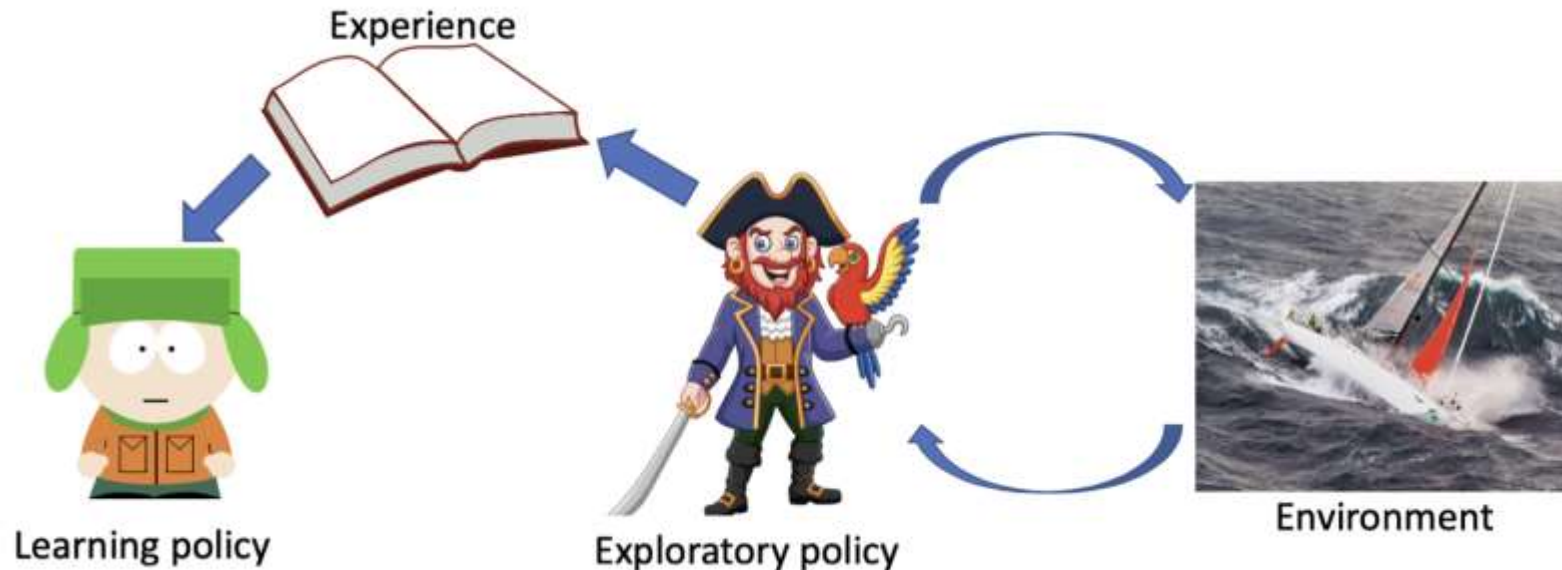
until S is terminal

探索与利用 (Exploration & Exploitation)

- 采用同样的策略进行交互采集数据，并利用该数据进行策略训练，易陷入局部最优
- On-policy: 原始策略与更新策略采用相同的策略 => SARSA
 - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
- Off-policy: 原始策略与更新策略采用不同的策略 => Q-learning
 - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t)]$



Off-policy学习方式



- 根据行为策略 μ 收集数据 $s_1, a_1, r_2, s_2, a_2, r_3, \dots$
- 根据数据更新策略 π
- 好处：重复使用所有旧策略生成的数据

Off-policy学习方式——Q Learning

- 回顾：贝尔曼迭代：

- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t))$

- 动作 a' 由策略 π 产生：

- $\pi(s_{t+1}) = \operatorname{argmax}_{a'} Q(s_{t+1}, a')$

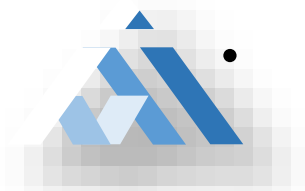
- 此时的TD target也称作Q-learning target:

- $r_{t+1} + \gamma Q(s_{t+1}, a') = r_{t+1} + \gamma Q(s_{t+1}, \operatorname{argmax}_{a'} Q(s_{t+1}, a')) = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')$

- 行为策略 μ :

- 选择1：随机策略

- 选择2： ε -greedy策略



Off-policy学习方式——Q Learning

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop 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



Sarsa vs. Q Learning

- Sarsa: on-policy的TD方法:

- 在状态 s_t 下, 根据 ϵ -greedy策略选择动作 a_t
- 使用动作 a_t 与环境交互, 获得 s_{t+1} 和 r_{t+1}
- 在状态 s_{t+1} 下, 根据 ϵ -greedy策略选择动作 a_{t+1}
- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
- a_t 与 a_{t+1} 来源于同一策略, 所以是on-policy

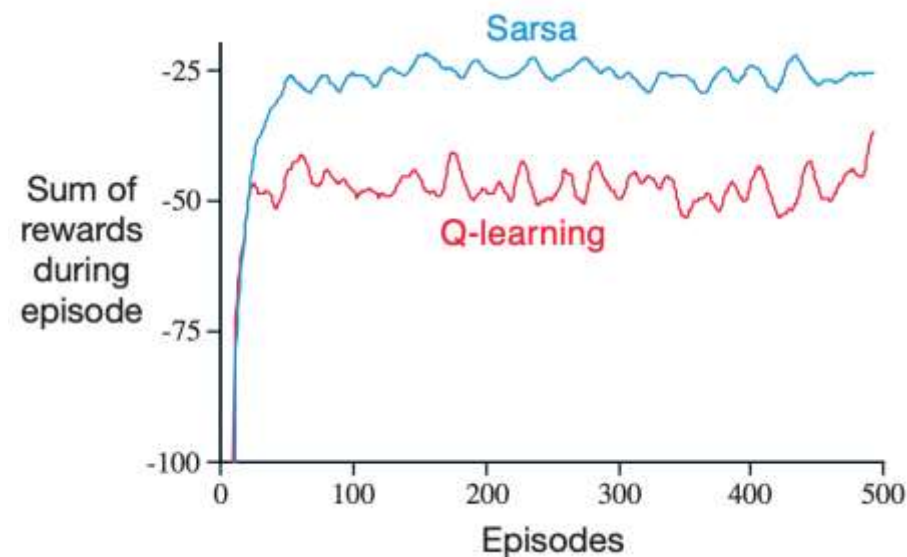
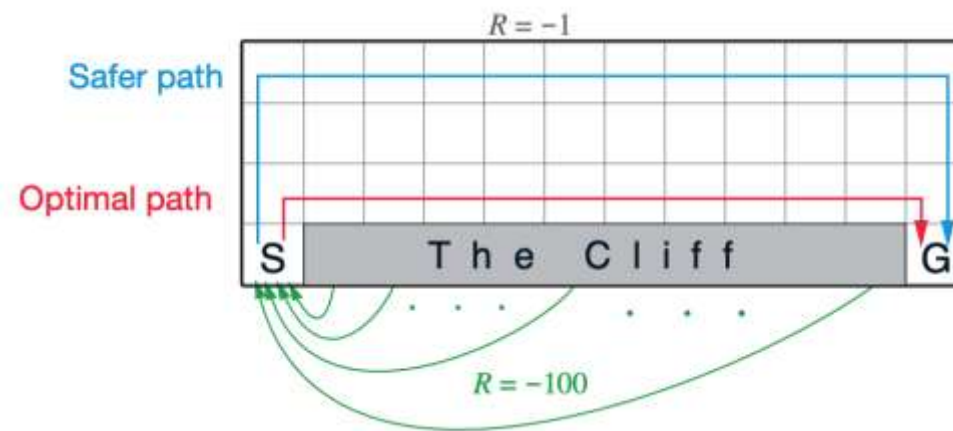
- Q Learning: off-policy的TD方法:

- 在状态 s_t 下, 根据 ϵ -greedy策略选择动作 a_t
- 使用动作 a_t 与环境交互, 获得 s_{t+1} 和 r_{t+1}
- 在状态 s_{t+1} 下, 根据greedy策略选择动作 a_{t+1}
- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
- a_t 与 a_{t+1} 来源于不同策略, 所以是off-policy

例子: Cliff Walk

- 问题设定:

- $\gamma = 1$
- episodic task
- 状态空间: 二维栅格
- 动作空间: 上下左右
- 单步奖励: -1
- 终止奖励: -100 (如果进入cliff)



第十五讲 强化学习

- I. 简介
- II. 马尔可夫决策过程
- III. 策略迭代与价值迭代
- IV. 预测与控制
- V. 探索与利用
- VI. 深度强化学习

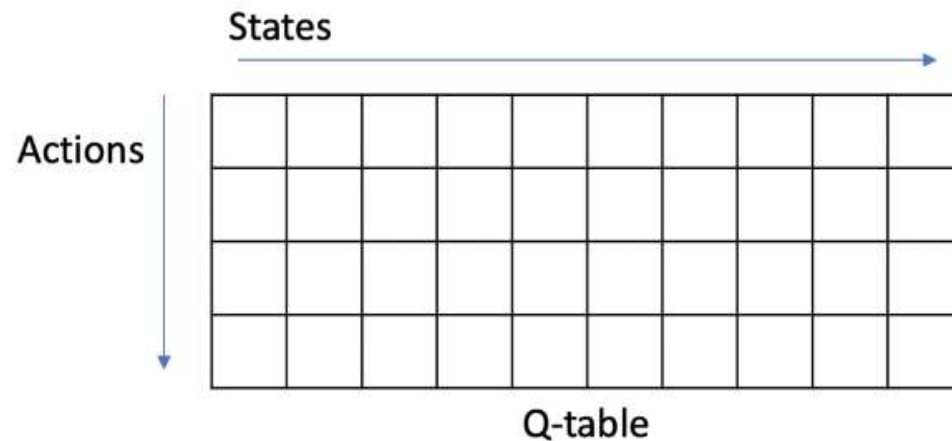


表格型方法的缺点

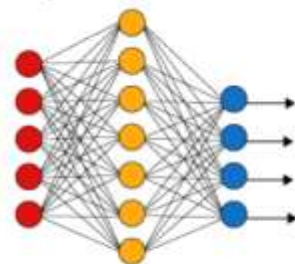
- DP、PI、VI、Sarsa、Q learning统称为表格型方法
- 理论上只能处理状态空间与动态空间均为离散的问题
- 实际上只能处理离散且规模较小的问题
- 规模较小的强化学习任务：
 - Cliff walk: 状态数目为 4×16
 - Mountain car: 状态数目为1600
 - Tic-Tac-Toe: 状态数目为1000
- 规模较大的强化学习任务：
 - Backgammon: 状态数目为 10^{20}
 - chess: 状态数目为 10^{47}
 - Game of Go: 状态数目为 10^{170}
 - 宇宙中原子的个数: 10^{80}

表格型方法的缺点

- 表格型方法中，我们使用表格存储V与Q值
- 当任务规模变大，带来两大挑战：
 - 需要巨大的表格存储
 - V与Q值的更新很慢
- 如何避免单独学习每个状态，或显式存储所有状态？
 - 引入函数估计器（模型）
 - 通过模型的参数或结构“存储”所有状态
 - 从已学习的状态推演到未学习的状态
- 常用函数估计器
 - 线性模型，决策树，最近邻
 - 神经网络

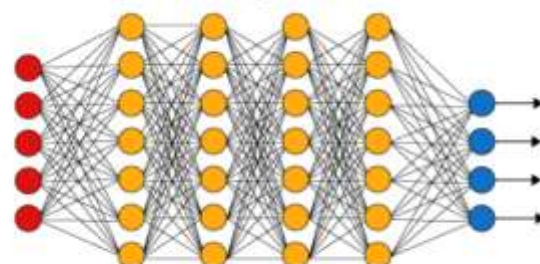


Simple Neural Network



● Input Layer

Deep Learning Neural Network



● Hidden Layer

● Output Layer

Deep Q-Networks (DQN)

- DeepMind发表在Nature的论文：Human-level control through deep reinforcement learning, 2015
- DQN在Atari游戏中达到了人类专家的水平
- Q表 \rightarrow 神经网络（输入状态 s_t 与 a_t ，输出 $Q(s_t, a_t)$ ）



四种Atari游戏：
Breakout, Pong,
Montezuma's Revenge,
Private Eye



Deep Q-Networks (DQN)

- 在强化学习中，引入神经网络需要考虑的问题：
 - 神经网络结构的设计
 - Loss函数的设计
- Loss函数由神经网络的输出 (prediction) 与真值 (label) 组成
 - 在DQN中，网络输入状态 s_t 与 a_t ，输出 $Q(s_t, a_t)$
 - 网络参数记作 θ
 - Label为TD target，记作 y_θ (label与网络参数相关!)
 - $L(\theta) = (Q_\theta(s, a) - y_\theta)^2$
 - 采用梯度下降系列的方法优化loss函数，即 θ





Deep Q-Networks (DQN)

- 引入深度学习的训练方式带来了两大新挑战
- sample correlation
 - 序列决策获取的样本存在时域的关联
 - 一般的深度学习假设样本独立同分布 (Independent and identically distributed, i.i.d.)
- non-stationary targets
 - 使用TD target作为深度学习的标签
 - 一般的深度学习使用的标签是stationary的 (标签与网络参数无关)
- 上述两大问题均会导致强化学习训练的不稳定



- **DQN对于两大挑战的解决方案为：**
 - **sample correlation:** **Experience replay**
 - **non-stationary targets:** **Fixed Q targets**



Deep Q-Networks (DQN)

- Experience replay (回放内存)

- 序列决策获取的样本存在时域的关联
- 为减少样本间的correlation, 将状态转移 $(s_t, a_t, s_{t+1}, r_{t+1})$ 存入回放内存 D 中

- 带有回放内存的训练算法

- 从回放内存中随机采样一些状态转移: $(s, a, s', r) \sim D$
- 计算TD target: $r + \gamma \max_{a'} Q_{\theta}(s', a')$
- 优化Loss函数, 采用梯度下降等方法更新网络参数

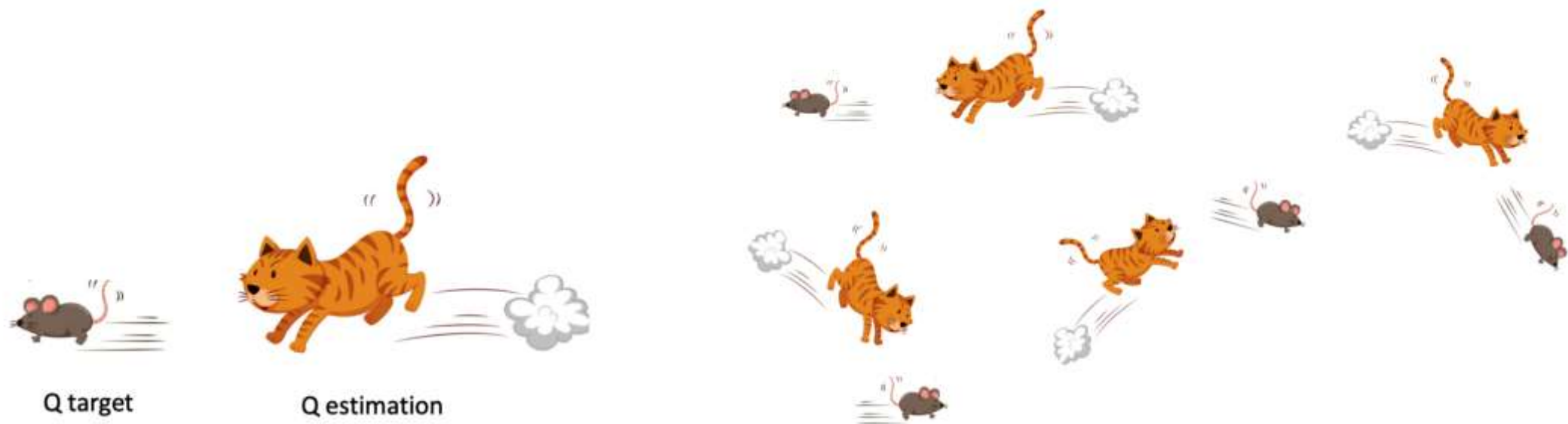
- $$\Delta\theta = \alpha(r + \gamma \max_{a'} Q_{\theta}(s', a') - Q_{\theta}(s, a)) \frac{\partial}{\partial \theta} Q(s, a)$$

s_1, a_1, r_1, s_2
s_2, a_2, r_2, s_3
s_3, a_3, r_3, s_4
...
s_t, a_t, r_t, s_{t+1}



Deep Q-Networks (DQN)

- Fixed Q targets
- 在上述训练过程中, label从网络参数 θ 计算得到, 因此是non-stationary的
- 直观理解: 猫捉老鼠
 - 猫和老鼠都在跑, 猫必须接近老鼠
 - 解决方案: 让老鼠比猫跑得慢





Deep Q-Networks (DQN)

- Fixed Q targets: 不从当前网络参数计算TD target
- 使用一组不同的网络参数 θ^- 计算TD target
- 带有Experience replay与Fixed Q targets的训练算法
 - 从回放内存中随机采样一些状态转移: $(s, a, s', r) \sim D$
 - 计算TD target: $r + \gamma \max_{a'} Q_{\theta^-}(s', a')$
 - 优化Loss函数, 采用梯度下降等方法更新网络参数
 - $\Delta\theta = \alpha(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_{\theta}(s, a)) \frac{\partial}{\partial \theta} Q(s, a)$



- 实验结果：在不同设定下的得分

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

