模式识别
Pattern Recognition

李泽榉，复旦大学 生物医学工程与技术创新学院

目录

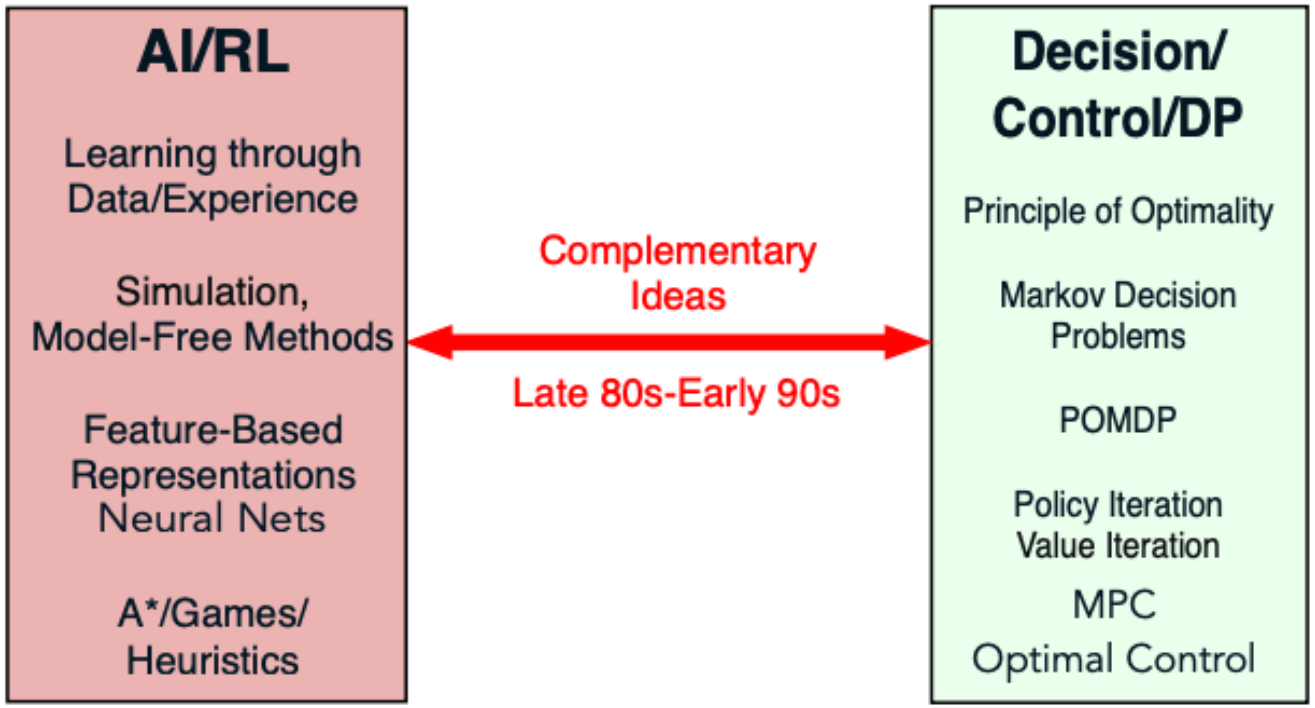
什么是强化学习

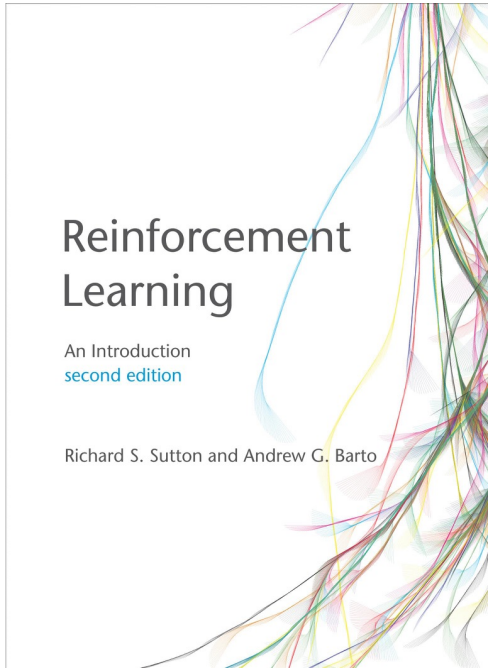
- 通过经验/数据学习在不确定性中做出良好决策
- 智能的核心组成部分
- 自20世纪50年代理Richard Bellman的理论与思想奠基以来蓬勃发展
- A number of impressive successes in the last decade

Richard Bellman
1920-1984

现代计算强化学习

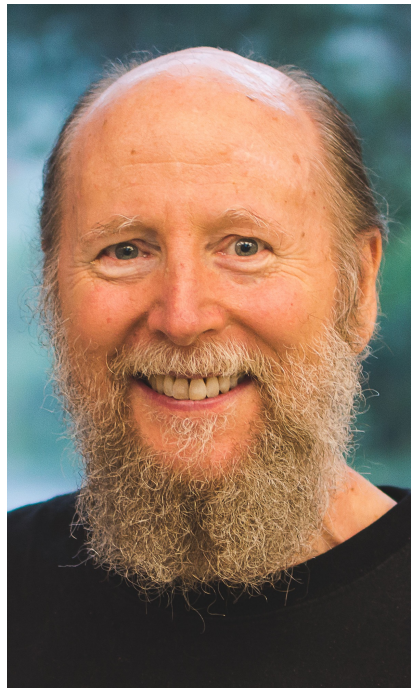
- 交叉学科
  - Reinforcement Learning: **AI**
  - Approximate Dynamic Programming: **Optimization**
  - Predictive and Adaptive Control: **Control Systems**





1998

Temporal-Difference
Policy Gradient



Richard S. Sutton
aka 强化学习之父

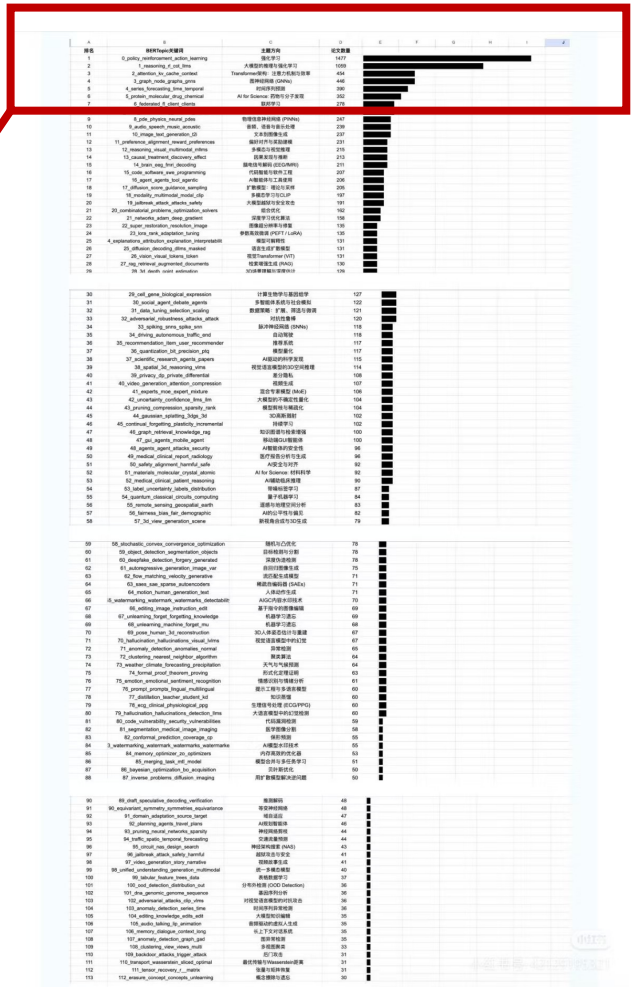
DeepMind

现代计算强化学习

- 当下最热门方向，没有之一

- **The Second RL Revolution** — RL has shifted from games to aligning LLMs (DPO, GRPO, RLV-R), becoming the backbone for model reasoning and control.

ICLR 2026 Keywords
~20k paper



排名	BERTopic关键词	主题方向	论文数量
1	0_policy_reinforcement_action_learning	强化学习	1477
2	1_reasoning_rl_cot_llms	大模型的推理与强化学习	1059
3	2_attention_kv_cache_context	Transformer架构：注意力机制与效率	454
4	3_graph_node_graphs_gnns	图神经网络 (GNNs)	446

https://github.com/zhihengli-casia/AI-Paper-Trends

30年前的想法

- From preface of Neuro-Dynamic Programming, Bertsekas and Tsitsiklis, 1996，第一次接触RL时的感受

> Our first impression was that the new methods were ambitious, overly optimistic, and lacked firm foundation. Yet there were claims of impressive successes and indications of a solid core to the modern developments in reinforcement learning, suggesting that the correct approach to their understanding was through dynamic programming.

> Three years later, after a lot of study, analysis, and experimentation, we believe that our initial impressions were largely correct. This is indeed an ambitious, often ad hoc, methodology, but for reasons that we now understand much better, it does have the potential of success with important and challenging problems.

- 是否与RL in LLM感觉类似?

今天的计划

Atari, AlphaGo, AlphaFold

ChatGPT

Deep Q learning

PPO, TRPO, GRPO

4 Value-based RL

5 Policy-based RL

人工智能领域的概念

3 Value and Policy Iteration

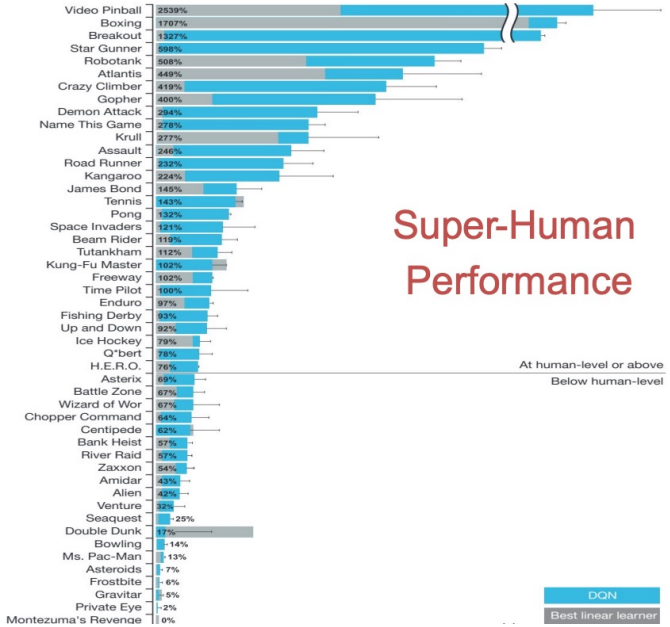
2 Bellman Equation

最优化领域的概念

强化学习的应用

Value-Based RL

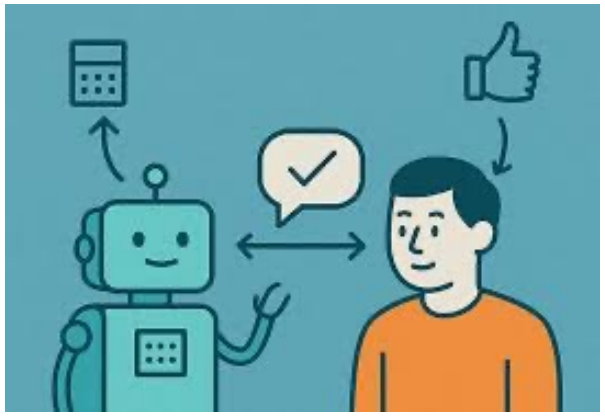
DeepMind



Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).

Policy-Based RL

OpenAI



强化学习的应用

"No simple yet reasonable evaluation function will ever be found for Go."

-- 2002, Martin Müller
(winner of 2009 Go program competition)

2016:

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

强化学习的应用

- OpenAI o1

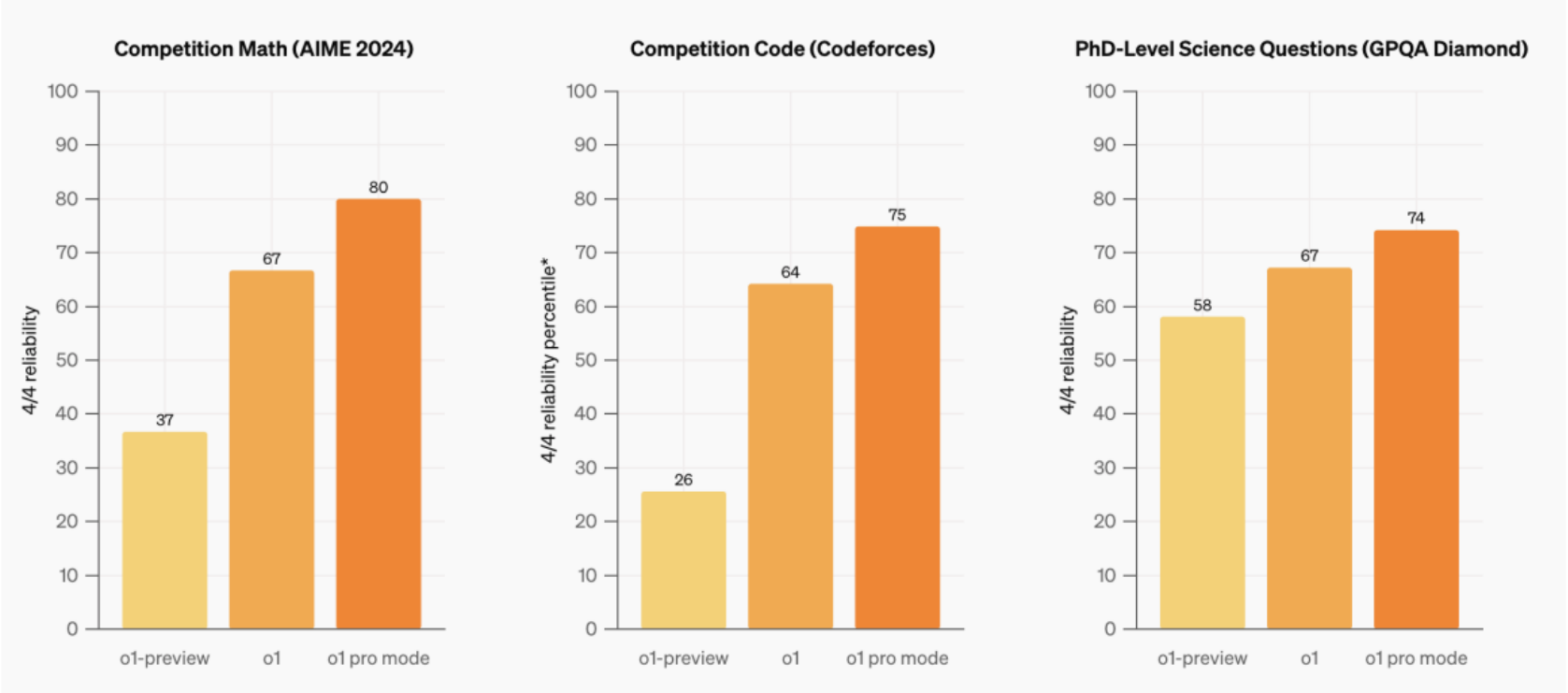
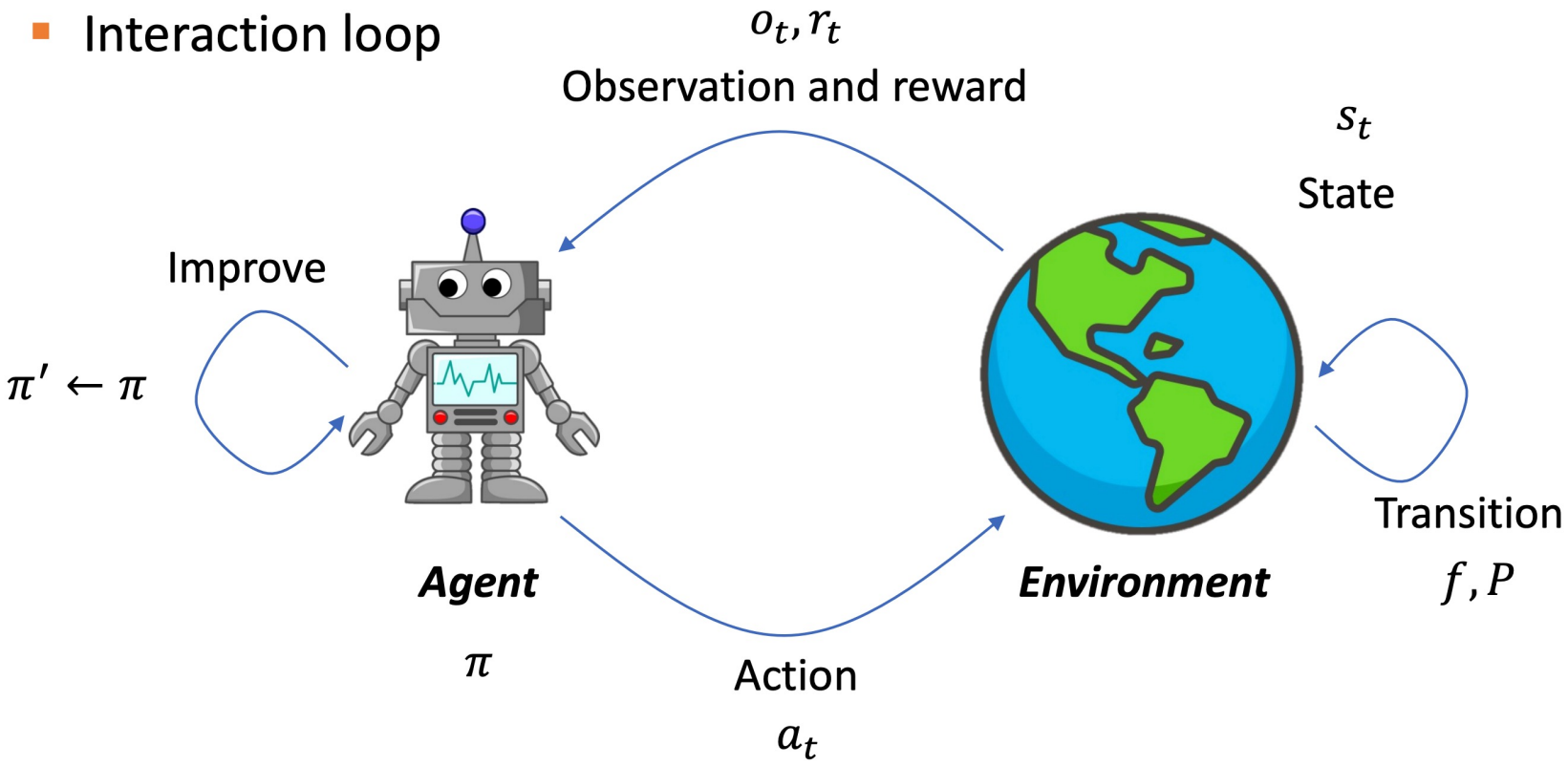


Figure: "Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process." Text/Image from: https://openai.com/index/introducing-chatgpt-pro/

强化学习的建模方式

- Interaction loop



Goal: maximize reward over time (returns, cumulative reward)

强化学习的建模方式

- Supervised learning - $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - Regression - $y^{(i)} \in \mathbb{R}$
  - Classification - $y^{(i)} \in \{1, \ldots, C\}$
- Unsupervised learning - $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$
  - Clustering
  - Dimensionality reduction
- Reinforcement learning - $\mathcal{D} = \{(\boldsymbol{s}^{(n)}, \boldsymbol{a}^{(n)}, r^{(n)})\}_{n=1}^{N}$

强化学习的建模方式

- **输入：** $(\mathcal{S}, \mathcal{A}, P, R)$
- 状态空间，$\mathcal{S} = \{s_i\}_{i=1}^{N}$
- 动作空间，$\mathcal{A} = \{a_k\}_{k=1}^{M}$
- 转移函数，$P: \mathcal{S} \times \mathcal{A} \to S$
- 奖励函数，$R: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

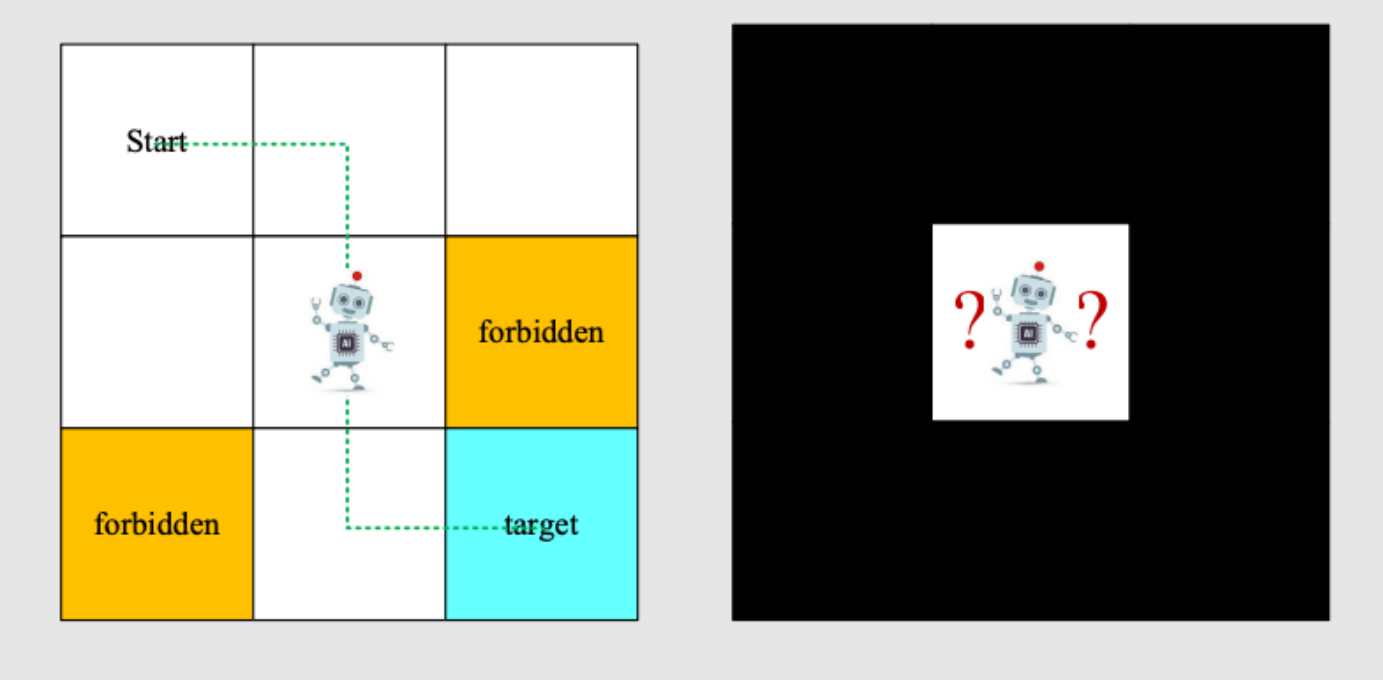
- **输出：** 策略，$\pi: \mathcal{S} \to \mathcal{A}$

强化学习的例子：自动驾驶

https://neptune.ai/blog/reinforcement-learning-applications

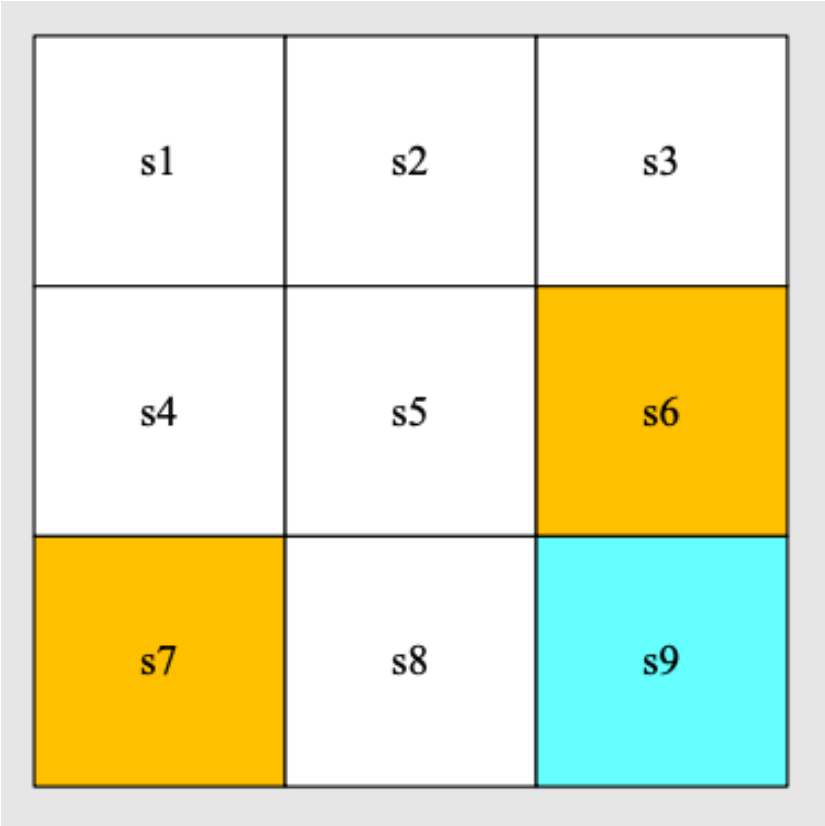
Gride-world Example

- 从任意位置找到能抵达targe的路线



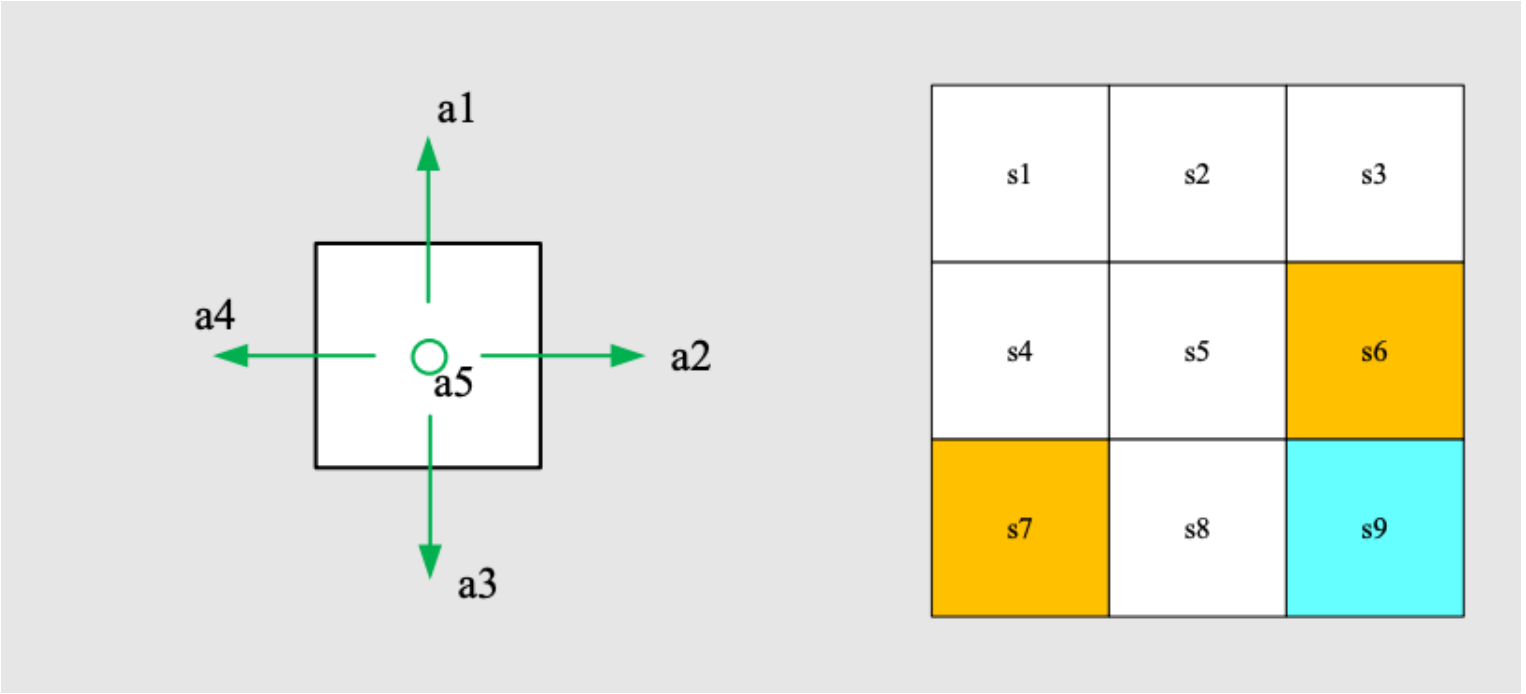
状态空间，$S = \{s_i\}_{i=1}^{N}$

- 智能体在环境中的位置
- 共有9个可能的状态：$s_1, s_2, \ldots, s_9$



动作空间，$\mathcal{A} = \{a_k\}_{k=1}^{M}$

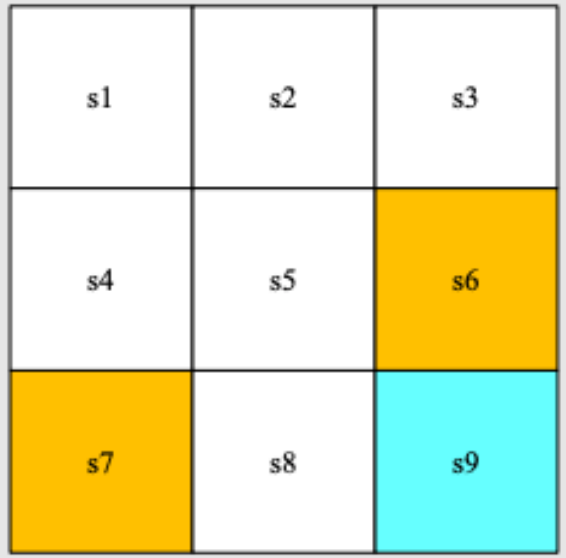
- 每一个状态下的动作
- 比如，a_1：向上走



转移函数，$P: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$

- 在一个状态，做一个动作之后的状态变化

	a_1 (upward)	a_2 (rightward)	a_3 (downward)	a_4 (leftward)	a_5 (still)
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_2	s_6	s_8	s_4	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

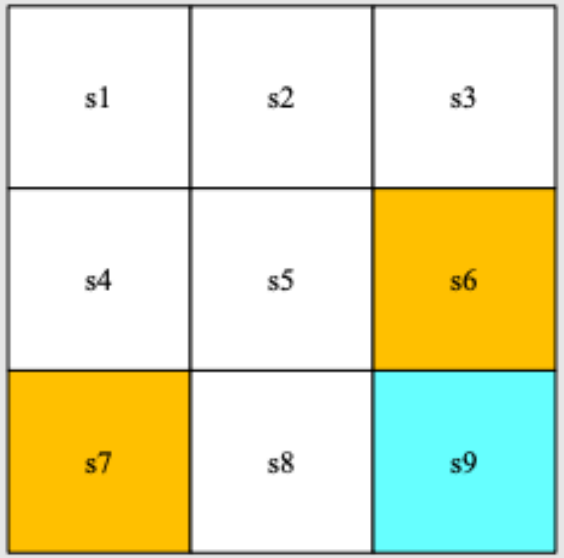


- 可以是Stochastic的，即$p(s'|s,a)$

奖励函数，$R: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

- 在一个状态，做一个动作之后获得的收益

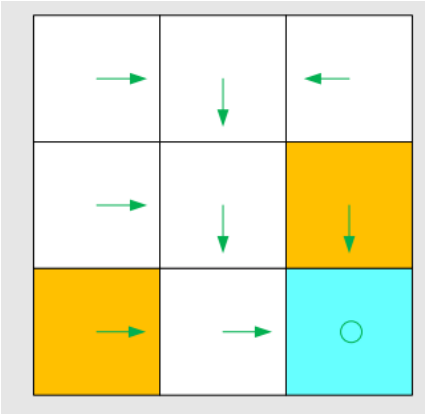
	a_1 (upward)	a_2 (rightward)	a_3 (downward)	a_4 (leftward)	a_5 (still)
s_1	r_{bound}	0	0	r_{bound}	0
s_2	r_{bound}	0	0	0	0
s_3	r_{bound}	r_{bound}	r_{forbid}	0	0
s_4	0	0	r_{forbid}	r_{bound}	0
s_5	0	r_{forbid}	0	0	0
s_6	0	r_{bound}	r_{target}	0	r_{forbid}
s_7	0	0	r_{bound}	r_{bound}	r_{forbid}
s_8	0	r_{target}	r_{bound}	r_{forbid}	0
s_9	r_{forbid}	r_{bound}	r_{bound}	0	r_{target}



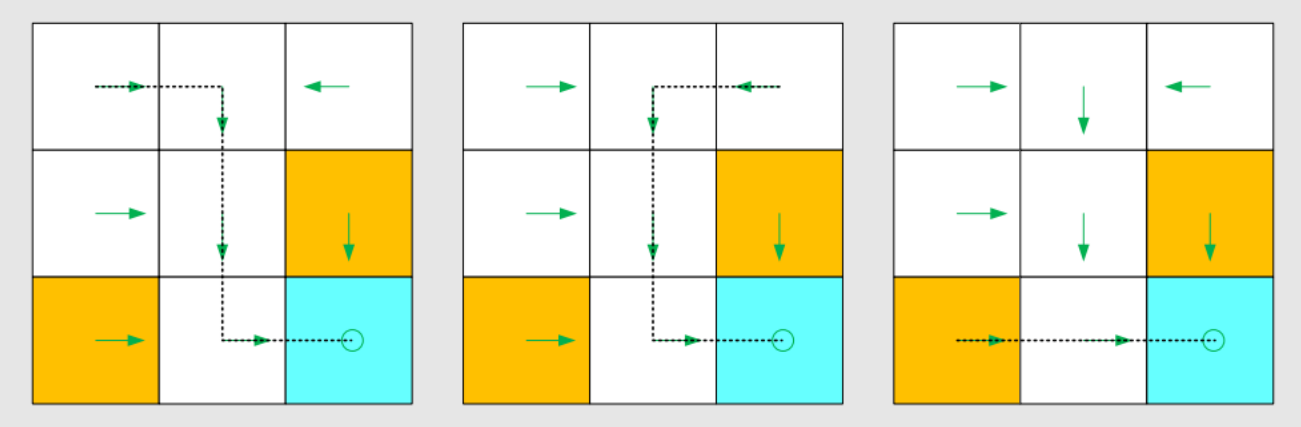
- 可以是Stochastic的，即$p(r|s,a)$

策略，$\pi: \mathcal{S} \rightarrow \mathcal{A}$

- 智能体在空间中的行为规则



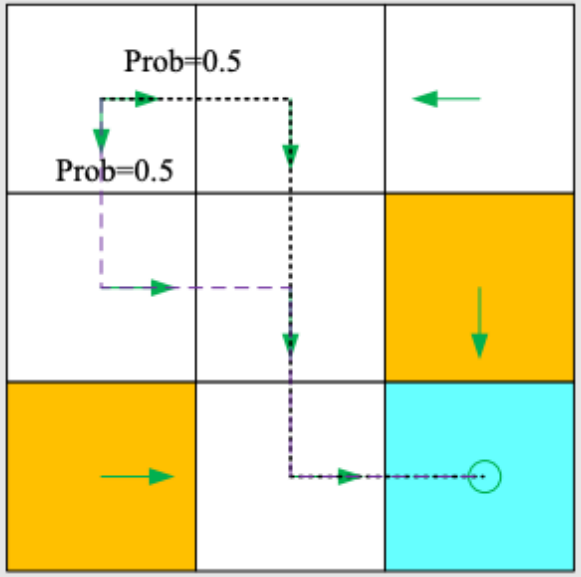
- 有了策略之后，就可以在任意位置找到抵达target的最优路径



策略，$\pi: \mathcal{S} \rightarrow \mathcal{A}$

- 本质，或者代码实现上也是一个表格

	a_1 (upward)	a_2 (rightward)	a_3 (downward)	a_4 (leftward)	a_5 (still)
s_1	0	0.5	0.5	0	0
s_2	0	0	1	0	0
s_3	0	0	0	1	0
s_4	0	1	0	0	0
s_5	0	0	1	0	0
s_6	0	0	1	0	0
s_7	0	1	0	0	0
s_8	0	1	0	0	0
s_9	0	0	0	0	1



马尔可夫链模型$MP(S,P)$

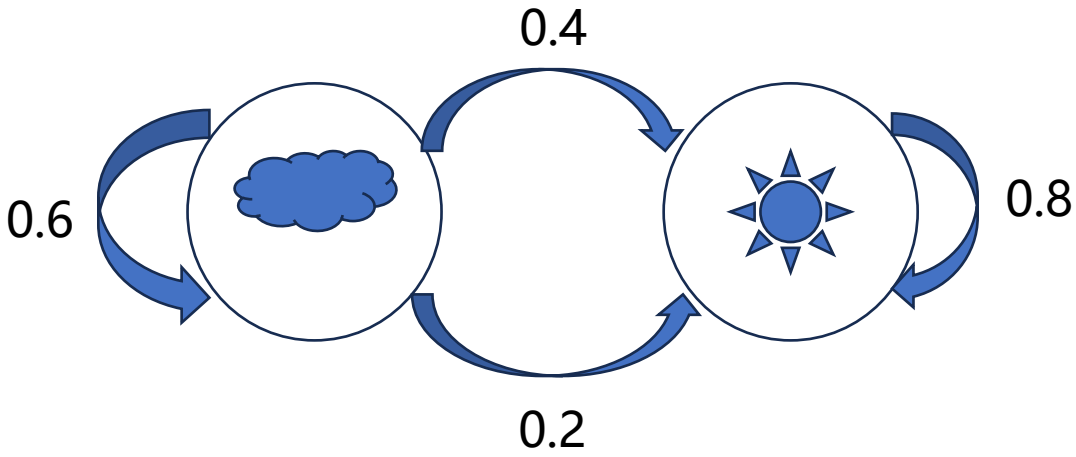
天气变化的数学语言表示：

P（明雨 | 今雨）= 0.6

P（明不雨 | 今雨）= 0.4

P（明雨 | 今不雨）= 0.2

P（明不雨 | 今不雨）= 0.8



并且第二天是否下雨只与今天的天气有关

P（明雨 | 今雨）= P（明雨 | 今雨、昨不雨）= P（明雨 | 今雨、昨不雨 、前天雨) =0.6

该性质是**马尔可夫性质（Markov Property）**，即**过程的无后效性**：在给定当前状态的条件下，未来状态与过去状态是独立的。过程的未来行为只依赖于当前状态，而与之前的历史状态无关。

$$Pr(S_{t+1}=x_{t+1}|S_0=x_0,S_1=x_1,\cdots,S_t=x_t)=Pr(S_{t+1}=x_{t+1}|S_t=x_t)$$

$t+1$时刻状态仅与t时刻状态相关

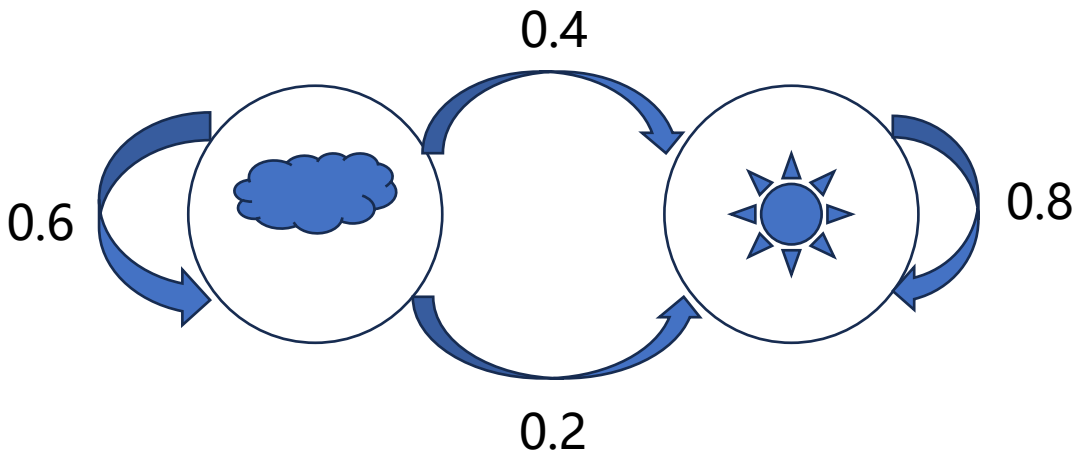
马尔可夫链模型$MP(S,P)$

马尔可夫链（Markov Chain）：

满足马尔可夫性质的离散随机过程，也被称为**离散马尔可夫过程。** 在离散的时间序列t=0,1,2,3，...中由系统状态与转移概率两者共同组成

- 系统状态：S={1,2,...，N}表示为整个状态空间，为所有可能的系统状态的合集，N为总状态个数；s(t)为系统在t时刻处在的系统状态，显然s(t)∈S
- 转移概率P_{ij}表示从状态i跳转为状态j的概率，即：

$$P_{ij} = \Pr\{ s(t+1)=j \mid \{s(t)=i \}$$
$$= \Pr\{ s(t+1)=j \mid \{s(t)=i , s(t-1)=k ,..., s(1)=a \}$$



在天气例子中，系统状态为S={晴天，雨天}，N=2，t代表天数，各个状态的转变概率则如图所示。

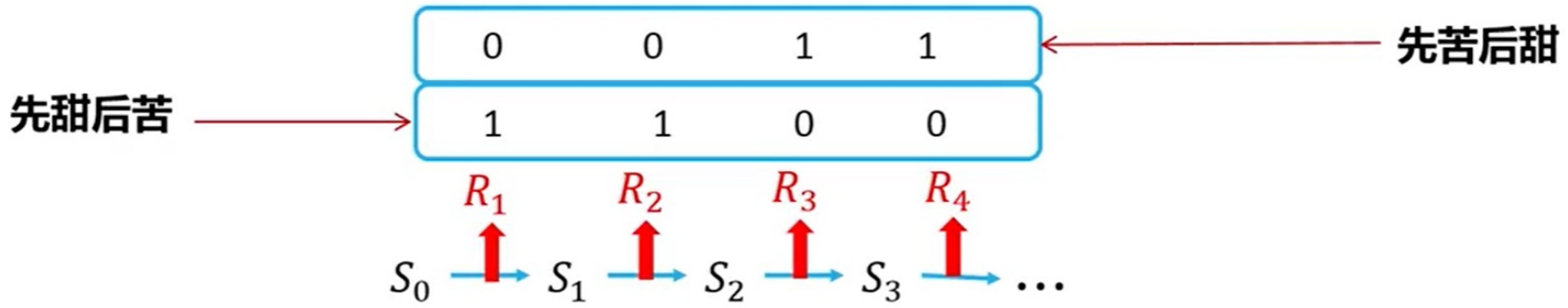
马尔可夫奖励过程$MRP(S,P,R,\gamma)$

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**$\boldsymbol{R}:\boldsymbol{S}\times\boldsymbol{S}\mapsto\mathbb{R}$，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$(0,0,1,1)$，

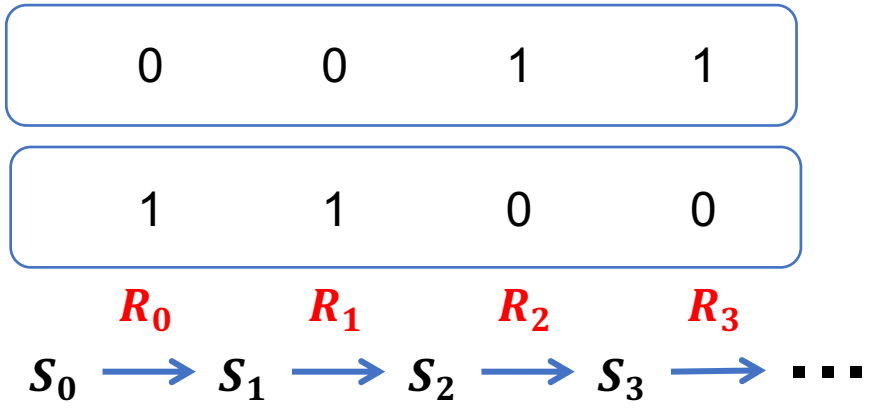
哪个序列决策更好?



马尔可夫奖励过程$MRP(S,P,R,\gamma)$

使用**奖励机制**对**序列决策**进行评价

奖励序列$(1,1,0,0)$和$(0,0,1,1)$，哪个序列决策更好？



折扣系数小于1时，越遥远的未来对累加回报的贡献越少

计算**累计奖励**：引入**折扣引子** $\gamma = 0.99$

$(1,1,0,0)$：$G_0 = 1 + 0.99\times1 + 0.99^2\times0 + 0.99^3\times0 = 1.99$

$(0,0,1,1)$：$G_0 = 0 + 0.99\times0 + 0.99^2\times1 + 0.99^3\times1 = 1.9504$

结论：$(1,1,0,0)$回报值高，是更好的奖励序列。 **关注眼前**

近的奖励，对该时刻的影响更大，RL奖励机制更注重短期收益。

强化学习基础概念

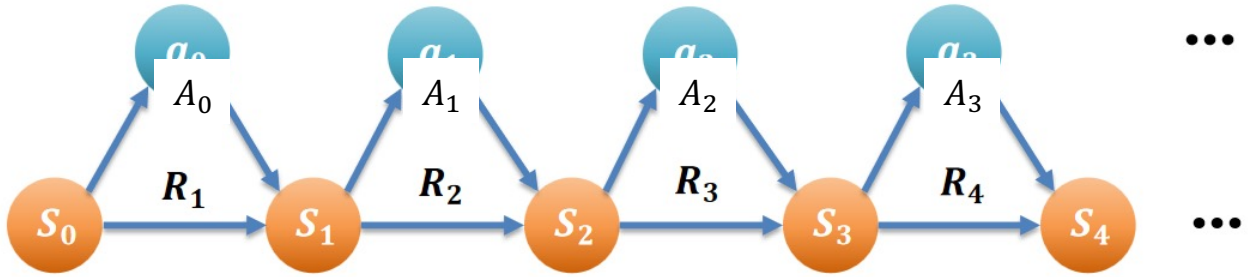
- 马尔可夫链模型$MP(\mathcal{S},P)$
- +奖励和折扣因子->马尔可夫奖励过程$MRP(\mathcal{S},P,R,\gamma)$
- +动作->马尔可夫决策过程$MDP(\mathcal{S},\mathcal{A},P,R,\gamma)$

- 理论上，所有强化学习都可以表述为马尔可夫决策过程

- 状态空间，$\mathcal{S}$
- 动作空间，$\mathcal{A}$
- 转移函数，P
- 奖励函数，R
- 折扣因子，γ

目录

马尔可夫决策过程：如何与环境交互



马尔可夫决策过程刻画了智能体与环境的交互特点，这一交互可用上图简洁概括。如图所示，智能体从初始状态S_0开始，根据策略执行动作A_0，得到奖励值R_1，同时状态转移到S_1，以此类推。于是，可得到一个状态序列$(S_0, S_1, \cdots,)$，该序列称为**轨迹（trajectory）**，轨迹长度可以是无限的，也可以有终止状态S_T。状态序列中包含终止状态的问题叫**分段（episodic）**问题，不包含终止状态的问题叫**持续（continuing）**问题。在分段问题中，一个从初始状态到终止状态的完整轨迹称为一个**片段（episode）**。

本课程讨论的主要是**分段问题**，有时也将动作和奖励记入状态序列中，用$(S_0, A_0, R_1, S_1, A_2, R_2, S_2, \cdots S_T)$来描述一个片段。

马尔可夫决策过程中的策略学习

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**

对环境中各种因素的说明	
已知的：S, A, R, γ	不一定已知的：Pr
观察到的：$(S_0, a_0, R_1, S_1, a_1, R_2, \cdots, S_T)$	

- **策略函数π：策略函数刻画了智能体选择动作的机制。**策略函数$\pi: S \times A \mapsto [0, 1]$，$\pi(s, a)$表示智能体在状态$s$下采取动作$a$的概率。策略函数可以是概率性的，也可以是确定的。**确定的策略函数**指在给定状态s情况下，只有一个动作a使得概率$\pi(s, a)$取值为1。对于确定的策略函数，为了简化符号，记$a = \pi(s)$。

一个好的策略函数应该能够使得智能体在采取了一系列行动后可得到最佳奖励，即最大化每一时刻的回报值$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$。从$G_t$的定义可知，$G_t$要根据一次包含了终止状态的轨迹序列计算所得。

马尔可夫决策过程中的策略学习

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π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

状态价值与动作-状态价值

- **状态-价值函数（State-Value Function）** $V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$，也可简称**状态价值、V值**
- **动作-价值函数 (Action-Value Function)** $q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$，也简称**动作价值、q值**

$$
\begin{aligned}
&V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\
&= \mathbb{E}_{a\sim\pi(s,\cdot)}\left[\mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]\right] \\
&= \sum_{a\in A} \underbrace{\pi(s,a)}_{\text{采取动作}a\text{的概率}} \times \underbrace{q_\pi(s,a)}_{\text{采取动作}a\text{后带来的回报期望}} \\
&= \sum\nolimits_{a\in A} \pi(s,a) q_\pi(s,a)
\end{aligned}
$$

状态价值与时间没有关系，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后续所得回报有关。**（马尔可夫性质）**

状态价值与动作-状态价值

- **状态价值函数（State - Value Function）** $V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$，也可简称**状态价值 V值**
- **动作-价值函数(Action-Value Function)** $q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$，也简称**动作价值 q值**

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

$$= \sum_{s' \in \boldsymbol{S}} \underbrace{P(s'|s,a)}_{\text{在状态}s\text{采取行动}a\text{进入状态}s'\text{的概率}} \times [\underbrace{R(s,a,s')}_{\text{在}s\text{采取}a\text{进入}s'\text{得到的即时奖励}} + \gamma \times \underbrace{V_\pi(s')}_{\text{在}s'\text{获得的未来回报}}]$$

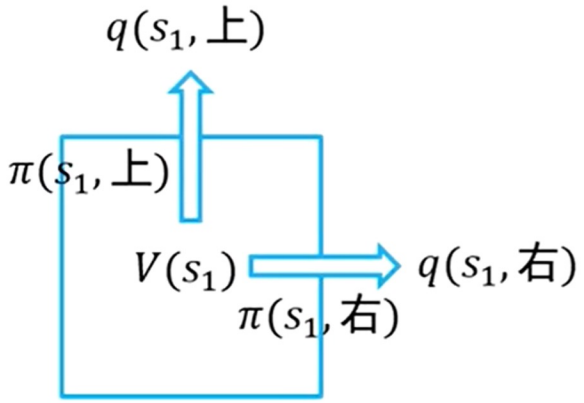
$$= \sum_{s' \in \boldsymbol{S}} P(s'|s,a)\,[R(s,a,s') + \gamma V_\pi(s')]$$

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。

价值函数与动作-价值函数的关系：以状态s1的计算为例

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

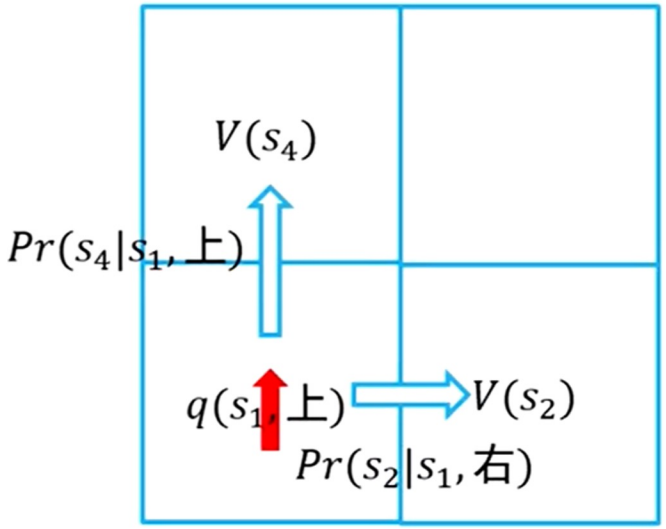
$$V_\pi(s_1) = \pi(s_1, 上) q_\pi(s_1, 上) + \pi(s_1, 右) q_\pi(s_1, 右)$$



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

贝尔曼方程

初始的reward

状态价值函数的贝尔曼方程 $V(s) = r(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$

所有跳转至状态S’ 的方法的概率与价值的乘积（方法期望价值）的和

- 状态价值函数的贝尔曼方程描述了状态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**即时奖励**的期望加上**后续状态的（折扣）价值函数**的期望。
- 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**。一旦找到了最优的价值函数或动作-价值函数，自然而然也就找到了对应的最优策略。

贝尔曼方程，又叫动态规划方程，是以Richard Bellman命名的，表示动态规划问题中相邻状态关系的方程。某些决策问题可以按照时间或空间分成多个阶段，每个阶段做出决策从而使整个过程取得效果最优的多阶段决策问题，可以用动态规划方法求解。某一阶段最优决策的问题，通过贝尔曼方程转化为下一阶段最优决策的子问题，从而初始状态的最优决策可以由终状态的最优决策(一般易解)问题逐步迭代求解。

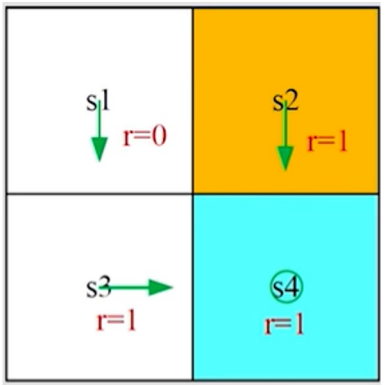
贝尔曼方程

价值函数的贝尔曼方程

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

If there are four states, $v_\pi = r_\pi + \gamma P_\pi v_\pi$ can be written out as

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



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

注意：矩阵形式的贝尔曼方程这里是$V_\pi(s)$,因为矩阵形式对所有状态（s_1, s_2, s_3, s_4）一起计算

贝尔曼方程

- 贝尔曼方程：在知晓奖励函数和状态转移矩阵的情况下即可计算价值函数的解析解。
  - For finite state MRP, we can express $V(s)$ using a matrix equation

$$\begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ \vdots \\ R(s_N) \end{pmatrix} + \gamma \begin{pmatrix} P(s_1|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \cdots & P(s_N|s_N) \end{pmatrix} \begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix}$$

$$\begin{aligned} V &= R + \gamma PV \\ V - \gamma PV &= R \\ (I - \gamma P)V &= R \\ V &= (I - \gamma P)^{-1} R \end{aligned}$$

  - Solving directly requires taking a matrix inverse $\sim O(N^3)$
  - Note that $(I - \gamma P)$ is invertible

贝尔曼方程

- 更一般的求解方式是动态规划

- Dynamic programming
- Initialize $V_0(s) = 0$ for all s
- For $k = 1$ until convergence
  - For all s in S

$$V_k(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s) V_{k-1}(s')$$

- Computational complexity: $O(|S|^2)$ for each iteration ($|S| = N$)

Bellman Equation

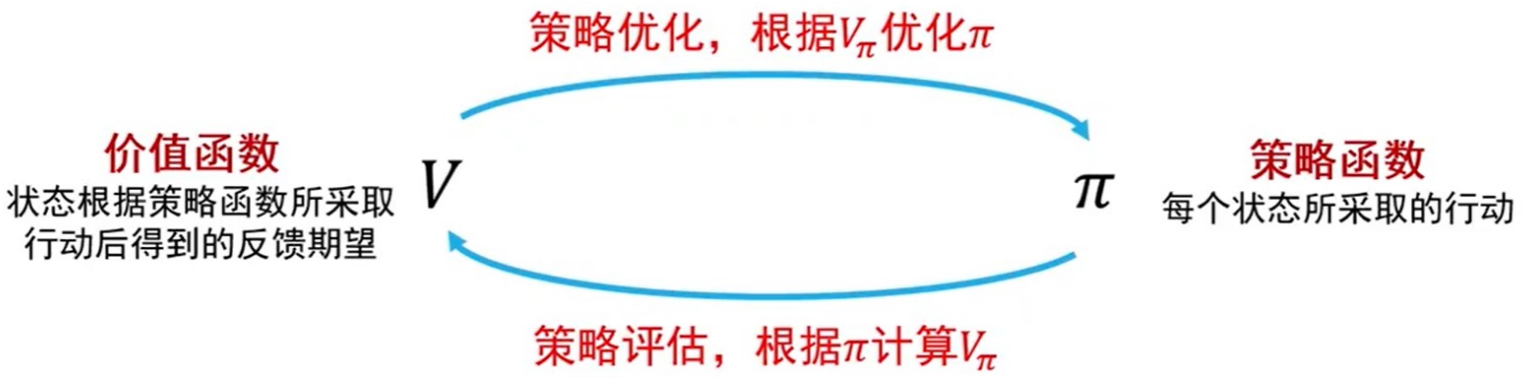
- 状态价值 $V_\pi(s)$是策略 π 在 s 的价值期望
- 动作-状态价值$q_\pi(s,a)$是策略 π 在 s 做 a 动作的价值期望
- 强化学习的目标是任意状态的$V_\pi(s)$都是最大
- 贝尔曼方程描述了状态价值函数的递推关系，是大多数RL优化的基础

目录

基于价值的强化学习

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$\pi^*$**，在策略$\pi^*$作用下使得任意状态$s \in S$对应的价值$V_{\pi^*}(s)$最大。

为了求解最优策略π^*,下图中展示了一种思路：从一个任意的策略开始，**首先计算该策略下状态价值V（或动作-价值q），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）**。



策略优化定理 与 贝尔曼最优公式

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**。

可以证明，给定任意给定状态$s \in S$，如果两个策略π和π'满足如下条件：

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

那么对于该任意给定状态$s \in S$，有

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

即策略π'不比策略π差。这个结论称为**策略优化定理（Policy Improvement Theorem）**。

*注意：$q_\pi\big(s,\pi'(s)\big)$的含义并不是在当前状态s下按照策略π'行动的回报期望，而是在当前状态s下按照策略π'去选择动作$\pi'(s)$，但是只改变这一个动作（后续状态的所有动作仍然按照原有策略π来选择）所得到的回报期望。

贝尔曼最优公式

$$v = f(v) = \max_{\pi}(r_\pi + \gamma P_\pi v)$$

收缩映射定理：

Theorem (Contraction Mapping Theorem)

For any equation that has the form of $x = f(x)$, if f is a contraction mapping, then

- **Existence**: *there exists a fixed point x^* satisfying $f(x^*) = x^*$.* v是一个不动点
- **Uniqueness**: *The fixed point x^* is unique.* v是唯一的
- **Algorithm**: *Consider a sequence $\{x_k\}$ where $x_{k+1} = f(x_k)$, then $x_k \to x^*$ as $k \to \infty$. Moreover, the convergence rate is exponentially fast.*

该最优化问题需要用迭代方法求解

贝尔曼最优公式与贪心最优策略

Theorem (Greedy Optimal Policy)

For any $s \in \mathcal{S}$, the deterministic greedy policy

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

is an optimal policy solving the BOE. Here,

BOE: Bellman Optimality Equation (贝尔曼最优方程)

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

找到使得q最大的动作a，该动作为最优动作

where $q^*(s,a) := \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v^*(s')$.

引入q，在状态s下采用控制动作a，所能达到的最大期望总奖励

计算q值，可以通过找q值最大的动作来将其定义为该状态下的最优动作

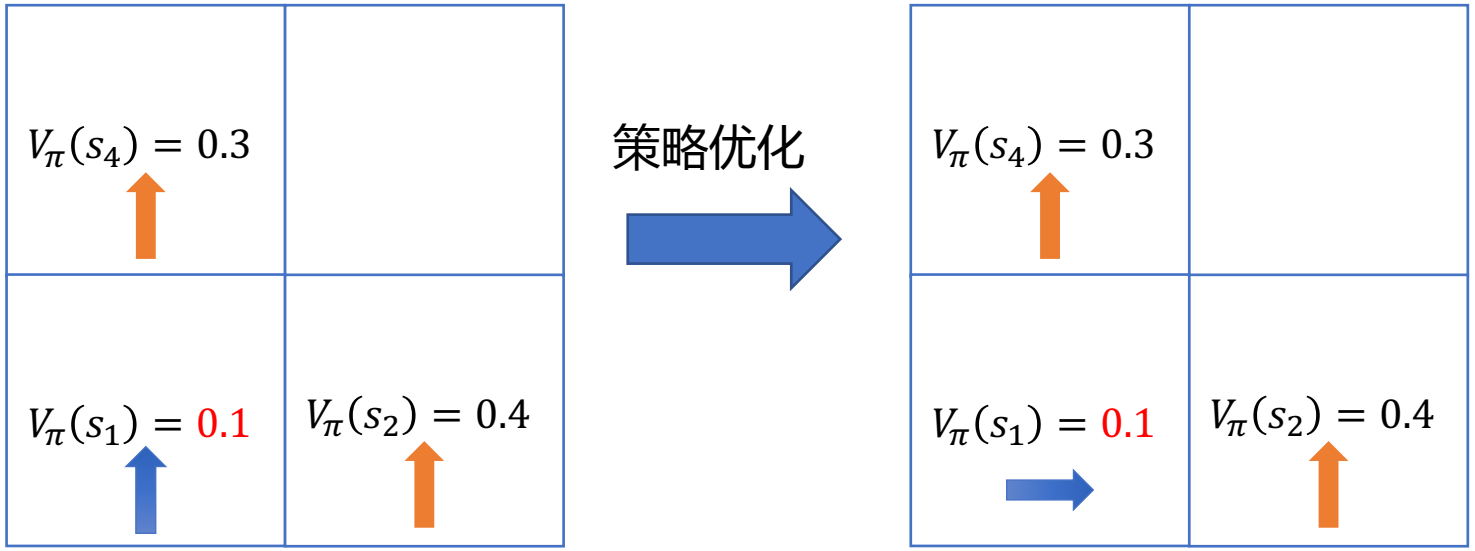
策略优化例子

问题：智能体目前在状态s_1，那么下一步是采取“向上移动一个方格”还是“向右移动一个方格”？

计算状态s_1选择这两个不同动作后分别所得动作-价值函数取值：

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

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



策略评估

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。

常见策略评估方法：

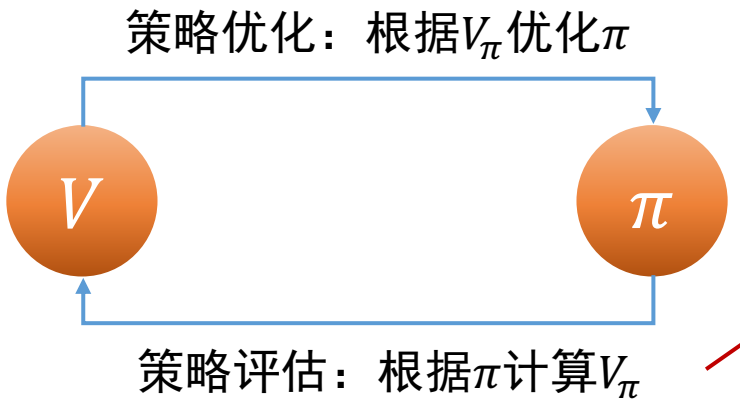
① 动态规划 (Dynamic Programming，DP)

② 蒙特卡洛 (Monte Carlo，MC)

③ 时序差分 (Temporal Difference，TD)

怎么知道你当前的策略好不好？

怎么知道你当前这一步玩的好不好？



实际中V_π是未知的，我们需要根据策略评估方法去评估我们选择的策略π，从而得到V_π的估计

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

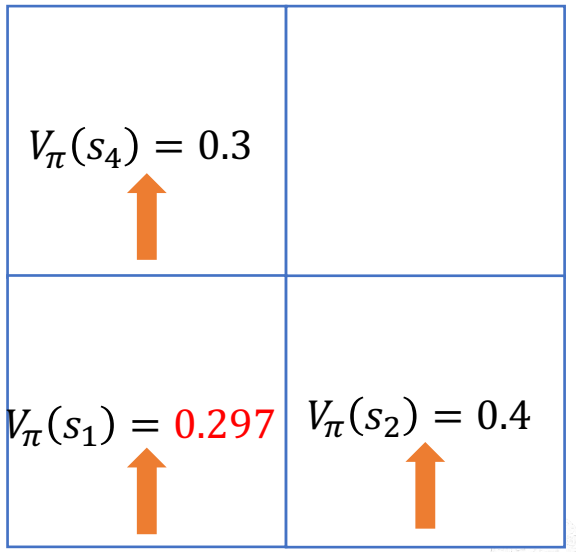
$q_\pi(s,a)$ ⬆ 等同于

直到V_π收敛

更新$V_\pi(s_1)$的值：

$$q_\pi(s_1,\text{上}) = 1\times(0+0.99\times0.3)+0\times(0+0.99\times0.4)$$
$$+\ldots = 0.297$$

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



动态规划缺点： 1、智能体需要事先知道**状态转移概率（需要额外对环境建模）**

2、无法处理状态集合大小无限的情况

策略评估：蒙特卡洛采样

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

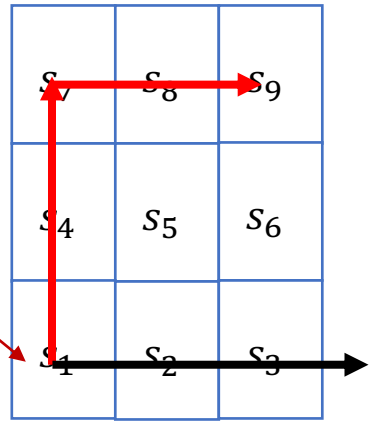
$$0 + \gamma \times 0 + \cdots + \gamma^3 \times 1 = 0.970$$

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

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，这里的例子是不确定策略（有两个路径）



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。即**大数定理**指出：对于独立同分布（IID）的样本数据，当样本足够大的时候，样本平均值向期望值收敛。

策略评估：时序差分

随机初始化V_π函数

repeat

　　$s \leftarrow$ 初始状态

　　repeat

　　$a \sim \pi(s,\cdot)$

　　执行动作a，观察奖励R和下一个状态s'

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

　　$s \leftarrow s'$

　　until s是终止状态

until V_π收敛

输入：策略π
输出：价值函数V_π

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

旧的价值函数值

学习得到的新价值函数值

策略评估：时序差分

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

旧的价值函数值　　**学习得到的新价值函数值**

- 由于通过采样进行计算，所得结果可能不准确，因此时序差分法并没有将这个估计值照单全收，而是以α作为权重来接受新的估计值，即把价值函数更新为$(1-\alpha)V_\pi(s) + \alpha[R + \gamma V_\pi(s')]$，对这个式子稍加整理就能得到：

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R + \gamma V_\pi(s') - V_\pi(s)]$$

$R + \gamma V_\pi(s')$为时序差分目标
$R + \gamma V_\pi(s') - V_\pi(s)$为时序差分偏差。

TD方法可以在每一步之后立即更新值函数，而不需要等待整个回合结束

- 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，同时也**缩短了反馈周期**。

策略评估：时序差分 举例

随机初始化V_π函数

repeat

　　$s \leftarrow$ **初始状态**

　　repeat

　　$a \sim \pi(s,\cdot)$

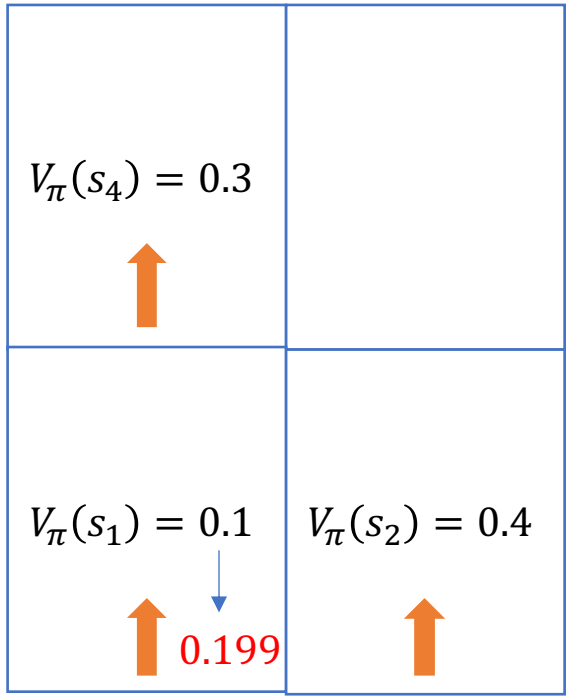
　　执行动作a，观察奖励R和下一个状态s'

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

　　$s \leftarrow s'$

　　until s**是终止状态**

until 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$s' = s_4$

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

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。

策略评估 总结

随机初始化策略π

repeat

$q_\pi(s,a)$ ← Policy-Evaluation(π)　// 策略评估，可使用**DP、MC、TD**等算法

for each $s \in S$ do

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

end

until π收敛

利用模型（动态特性 P 和回报函数 R）来从 V_π 中推导出

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

基于此我们得到RL学习算法： Policy-Iteration

存在的问题：

- Policy-Evaluation需要迭代求解，影响执行效率
- 随着迭代次数增多， q_π变化会越来越小

从动态规划策略评估引出价值迭代算法

在**动态规划**策略评估方法中，（**动态规划是基于马尔科夫决策过程和转态转移概率的**）
每次迭代中只对一个状态进行策略评估和策略优化，
可得到价值迭代算法：

输入：**马尔可夫决策过程** $MDP = (S, A, P, R, \gamma)$
输出：策略π

随机初始化π

repeat

$q_\pi(s,a) \leftarrow$ Policy-Evaluation(π)

for each $s \in S$ do

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

end

until π收敛

策略迭代算法 Policy-Iteration

随机初始化V_π

repeat

for each $s \in S$ do

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

end

until V_π收敛

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

价值迭代算法 Value-Iteration

价值迭代（Value Iteration）算法

随机初始化V_π

repeat

　　for each $s \in S$ do

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

　　end

until V_π收敛

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

第4行实际上包含两步，即：

$$\pi(s) \leftarrow \text{argmax}_a q_\pi(s,a)$$

$$V_\pi(s) \leftarrow q_\pi(s,\pi(s))$$

其中，$q_\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')]$。这两步分别对应了策略优化和策略评估两个阶段。也就是说，交替进行一次策略优化和一次策略评估。

价值迭代（Value Iteration）算法

以机器人寻路问题为例，假设价值函数的初值如中间图片所示，则完成值迭代算法中一次外层循环后，价值函数的值如最右侧图片所示。

以状态s_8为例：

$q_\pi(s_8, 右)$

$$= \sum_{s'} P(s'|s_8, 右)[R(s_8, 右, s') + \boldsymbol{\gamma V_\pi(s')}]$$

$$= 1\times(1 + 0.99\times 0) + 0\times\cdots = 1$$

$q_\pi(s_8, 上)$

$$= \sum_{s'} P(s'|s_8, 上)[R(s_8, 上, s') + \boldsymbol{\gamma V_\pi(s')}]$$

$$= 1\times(-1 + 0.99\times 0) + 0\times\cdots = -1$$

$$\boldsymbol{V(s_8) = max_a q_\pi(s_8, a) = max\{1, -1\}}$$

因此状态s_8的价值函数更新为1，同理可更新其他状态的价值函数。

Target

s_7	$\boldsymbol{s_8}$	s_9
s_4	s_5	s_6
s_1	s_2	s_3

0	0	0
0	0	0
0	0	0

0	1	0
0	0	1
0	0	0

价值迭代（Value Iteration）算法

第2次外层循环后

0.99	1	0
0	0.99	1
0	0	0.99

第3次外层循环后

0.99	1	0
0.98	0.99	1
0	0.98	0.99

第5次外层循环后

0.99	1	0
0.98	0.99	1
0.97	0.98	0.99

轨迹:（ s_1, s_4, s_7, s_8, s_9 ）

q_π相同情况下，
优先取动作"上"

→ s_7	→ s_8	s_9
↑ s_4	↑ s_5	↑ s_6
↑ s_1	↑ s_2	↑ s_3

Policy和Value的迭代

- 贝尔曼最优公式说明，我们**一定能**通过迭代方式求出最优状态值V
- 最优策略π^*会根据动作值q来选择$\pi'(s) = \text{argmax}_a q_\pi(s,a)$
- 最优策略有两种求解方法
  - Policy迭代先策略评估再优化
  - Value迭代每一步都有策略的优化和评估
- 两种优化方法都依赖于转移函数 $P\ (s'|s,a)$，需要额外用model-based方法建模环境，并不实用，现有的流行方法都是model-free的
- 时序差分（TD）能避免策略评估中的转移函数 $P\ (s'|s,a)$

目录

Q学习算法

输入：**马尔可夫决策过程** $MDP = (S, A, P, R, \gamma)$
输出：策略π

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$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R + \gamma V_\pi(s') - V_\pi(s)]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到V_π收敛

将策略迭代中的“策略评估”步骤替换为基于同策略（on-policy）的时序差分学习

等价于优化Bellman Equation

$$V(s) = r(s) + \gamma \sum_{s' \in S} P(s'|s) V(s')$$

Q学习算法

输入：**马尔可夫决策过程** $MDP = (S, A, P, R, \gamma)$
输出：策略π

初始化q_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$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R + \gamma V_\pi(s') - V_\pi(s)]$~~　$a' \sim \pi(s',\cdot)$
　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha[R + \gamma\, q_\pi(s',a') - q_\pi(s,a)]$
　　$s \leftarrow s'$
　　直到s是终止状态
直到q_π收敛

等价于下式：

$$q_\pi(s,a) \leftarrow (1-\alpha)q_\pi(s,a) + \alpha[R + \gamma\, q_\pi(s',a')]$$

直接记录和更新动作-价值函数q_π，也叫SARSA (State-Action-Reward-State-Action)

Q学习算法

输入：**马尔可夫决策过程** $MDP = (S, A, P, R, \gamma)$

输出：策略π

直接找最佳动作！！

初始化q_π函数

循环

　　初始化s为初始状态

　　循环

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

　　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

等价于下式：

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

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，

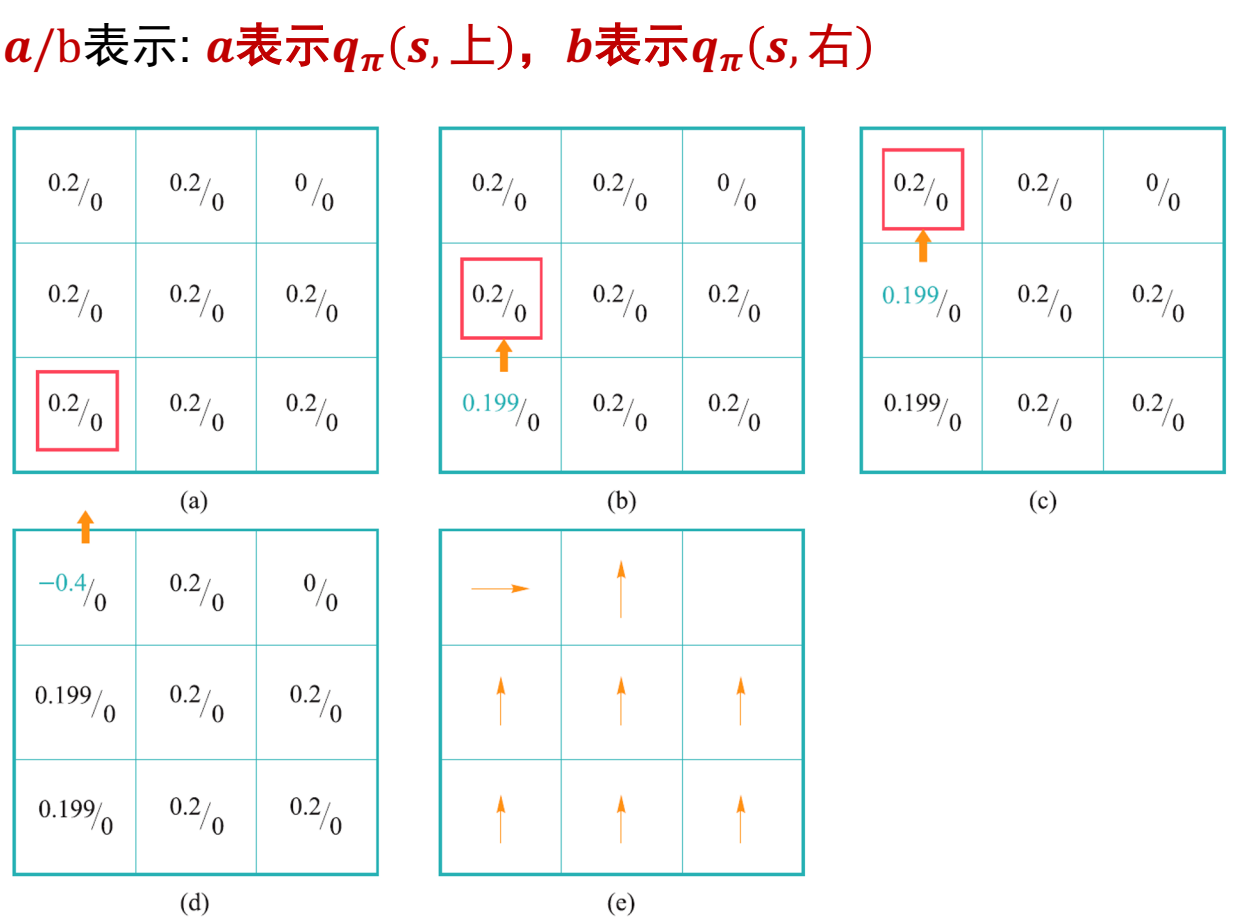
如果算法仍然记录价值函数V_π，在不知道状态转移概率的情况下将无法求出q_π。

等价于优化Bellman Optimality Equation $\max_\pi (r_\pi + \gamma P_\pi v)$

Q学习算法

- 图(a)表示算法的初始状态，其中a/b表示对应状态的动作-价值函数的取值，**斜线上方的a表示$q_\pi(s,$上$)$，斜线右侧的b表示$q_\pi(s,$右$)$**
- 图(a)中显示除终止状态外的所有的向上的动作-价值函数值为0.2，所有向右的动作-价值函数值为0。

当然，令这些动作-价值函数的初始值全部为0也是可以的，图(a)中所设置的初始值只是为了让这个例子更容易说明策略学习



Q学习算法

图(a)中的红色方框代表了智能体当前所处的位置。从图(a)为出发，在算法第5行，智能体算出应该往上走，执行这个动作，得到奖励$R=0$，并进入下一状态$s'=s_4$（图b），因此可如下更新对应的动作-价值函数：

$$q_\pi(s_1,\text{上}) \leftarrow q_\pi(s_1,\text{上}) + \alpha[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)]$$

$$= 0.2 + 0.5\times[0 + 0.99\times max\{0, 0.2\} - 0.2]$$

同理算法再执行一次内层循环得到图(c)，接着得到图(d)。这个过程中每一步都通过后续状态的动作-价值函数更新当前状态的动作-价值函数，图(b)-(c)都没有对策略产生影响。

当智能体在图(d)中到达了损坏状态s_d，一个负的奖励（即奖励为-1）使$q_\pi(s_7,\text{上})$变为负值，此时策略$\pi(s_7)$从“向上移动一个方格”变成了“向右移动一个方格”，至此智能体完成了一个片段（一次外层循环）的更新。

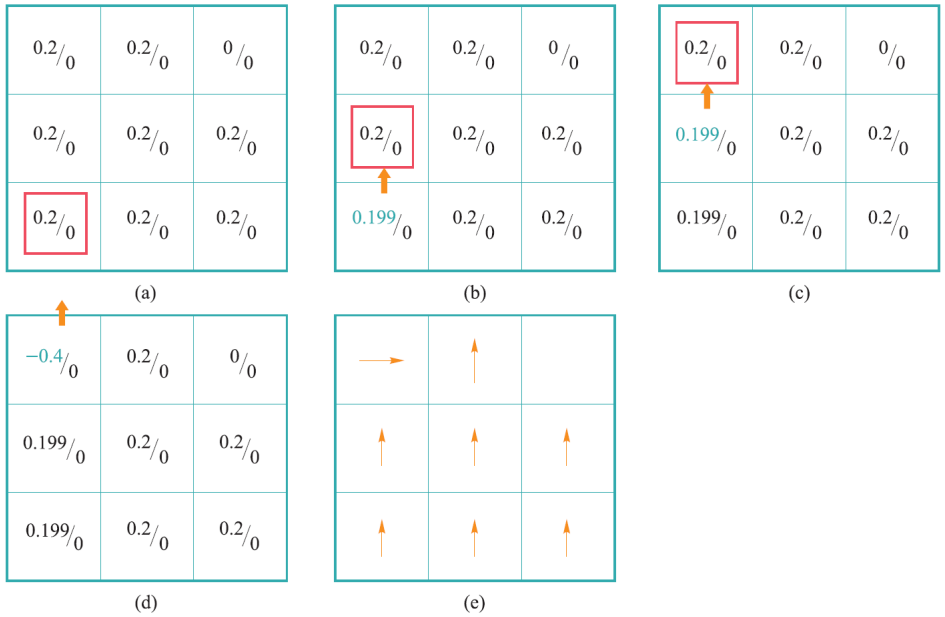
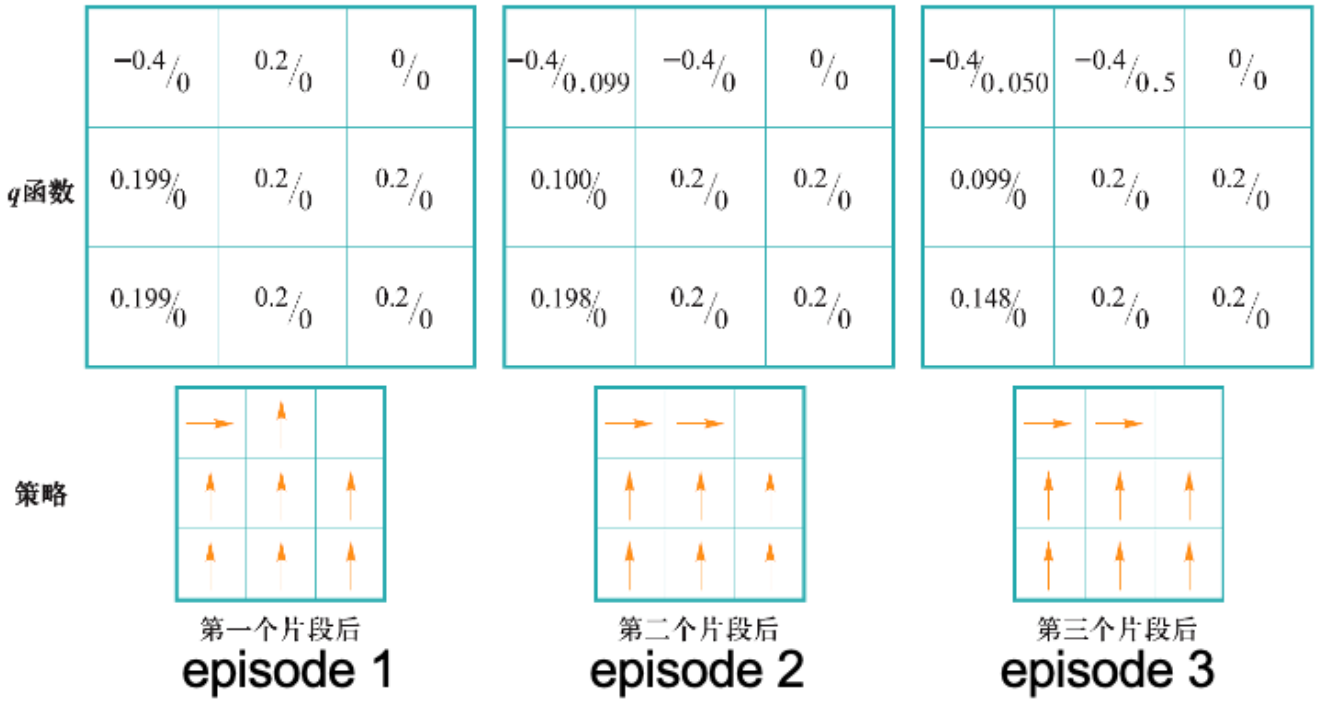


图 6.10　Q 学习的一个片段的执行过程

Q学习算法

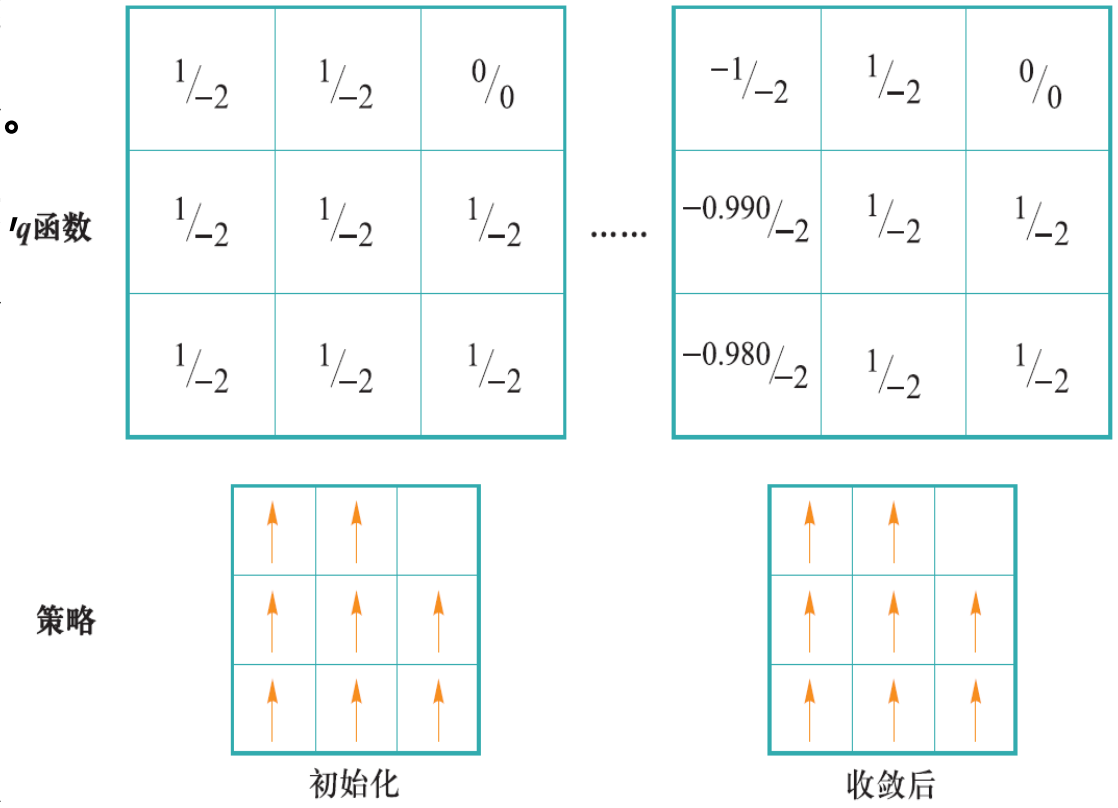


右图展示了Q学习算法执行三个episode的过程，从第二个片段结束时起，算法已经学习得到了沿着轨迹$(s_1, s_4, s_7, s_8, s_9)$到达目标状态$s_9$的策略。实际上，虽然此时动作-价值函数尚未收敛，但此后策略已经不会再发生变化了。

探索与利用

如果将动作-价值函数初始化为$1/-2$，使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，Q学习执行过程如右图所示，可见即使在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)导致机器人被损坏。

出现此问题的原因是$q_\pi(\cdot, 右)$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。



动作－价值函数初始化为 1/−2 时 Q 学习算法的执行过程

探索与利用

探索（exploration）和利用（exploitation）之间存在对立关系

使用估计奖励值最大的赌博机（**exploitation**）

探索未知奖励值的另外赌博机（**exploration**）

贪心算法

ϵ-贪心算法

探索与利用

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

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。

探索与利用

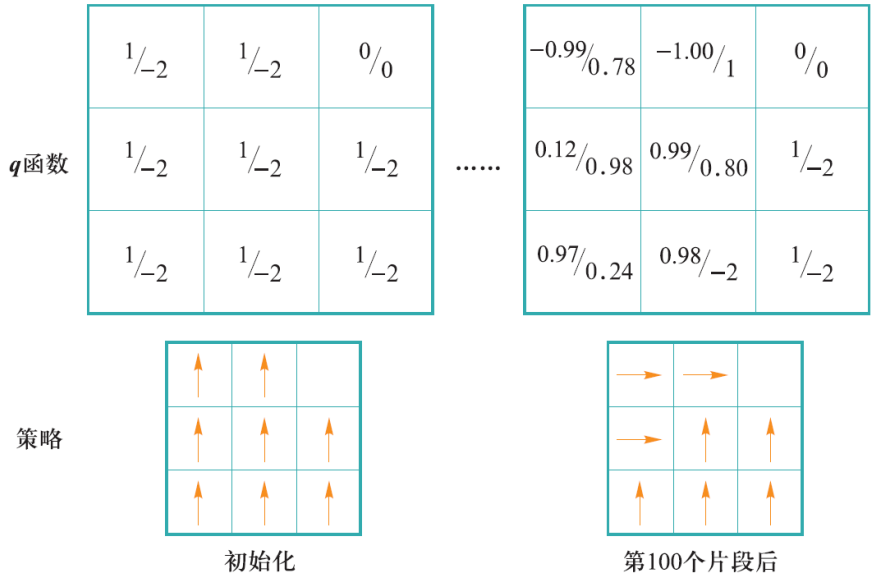
使用 ϵ 贪心算法进行探索的 Q 学习

函数： EpsGreedy
输入： 状态 s，动作－价值函数 q_π，参数 ϵ
输出： 动作 a

```
n ~ uniform(0, 1)
if n < ε then
    a ← 从 A 中随机选择
else
    a ← arg max_a q_π(s, a')
end
```

函数： QLearning
输入： 马尔可夫决策过程 MDP $=(S, A, P, R, \gamma)$
输出： 策略 π

```
随机初始化 q_π
repeat
    s ← 初始状态
    repeat
        a ← EesGreedy(s, q_π, ε)
        执行动作 a，观察奖励 R 和下一个状态 s'
        q_π(s, a) ← q_π(s, a) + α[R + γ max_a q_π(s', a') − q_π(s, a)]
        s ← s'
    until s 是终止状态
until q_π 收敛
π(s) := arg max_a q(s, a)
```



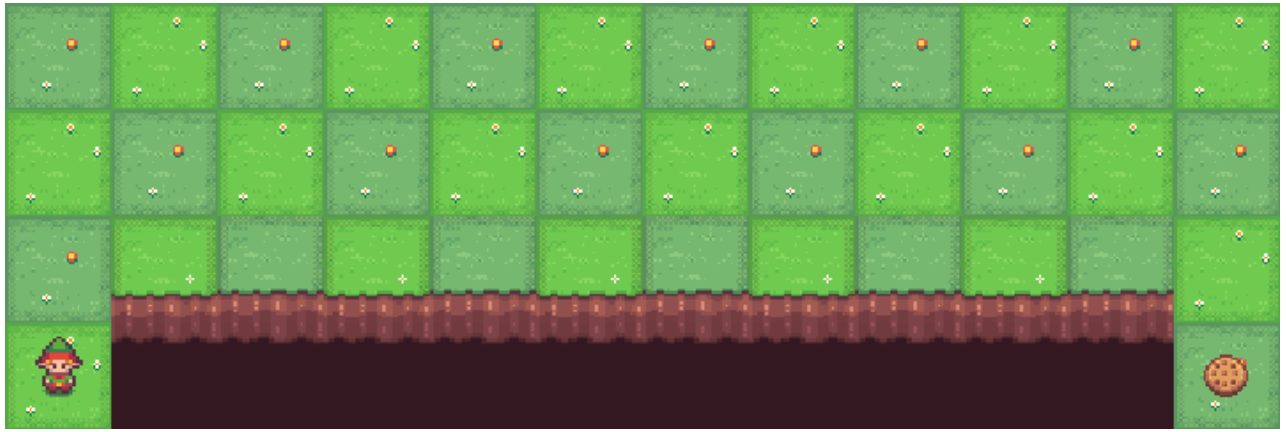
使用 ϵ 贪心策略进行探索的 Q 学习的执行过程

从上图可知，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。

Q学习算法实例 CliffWalking悬崖行走

Action

- 0: 上
- 1: 右
- 2: 下
- 3: 左



State

- 4x12 矩阵
- [3, 0] 最左下角起点
- [3, 11] 右下角终点
- [3, 1..10] 最下一行从1-10格为悬崖陷阱

Reward

- 每一步都会有-1的奖励，跌落悬崖有-100的奖励

Action Space	Discrete(4)
State Space	Discrete(48)

代码结构总览


CliffWalking - SARSA/Q-learning
main.py
Hyper Parameters
Training Process
Test Process
agent.py
Members
Q-Table (pandas.DataFramc)
Parameters
Methods
select_action() (ε-grccdy)
learn() (updatc Q-tablc)
save_q_table() / load_q_table() (.csv)

on policy与off-policy

SARSA (on-policy)

学习策略和行为策略相同

- 行为策略：ε-greedy
- 学习策略：ε-greedy

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy)

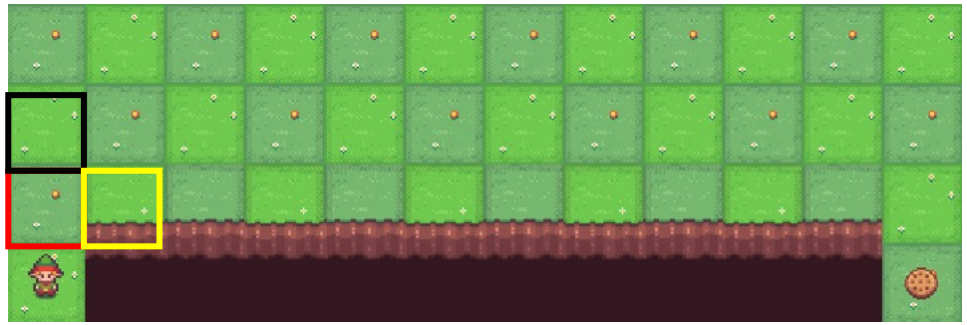
学习策略和行为策略不同

- 行为策略：ε-greedy
- 学习策略：最大化 Q 函数

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$;
  until S is terminal

on policy与off-policy

SARSA (on-policy)



Q-learning (off-policy)

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$;
  until S is terminal

Q学习方法的核心——q_table

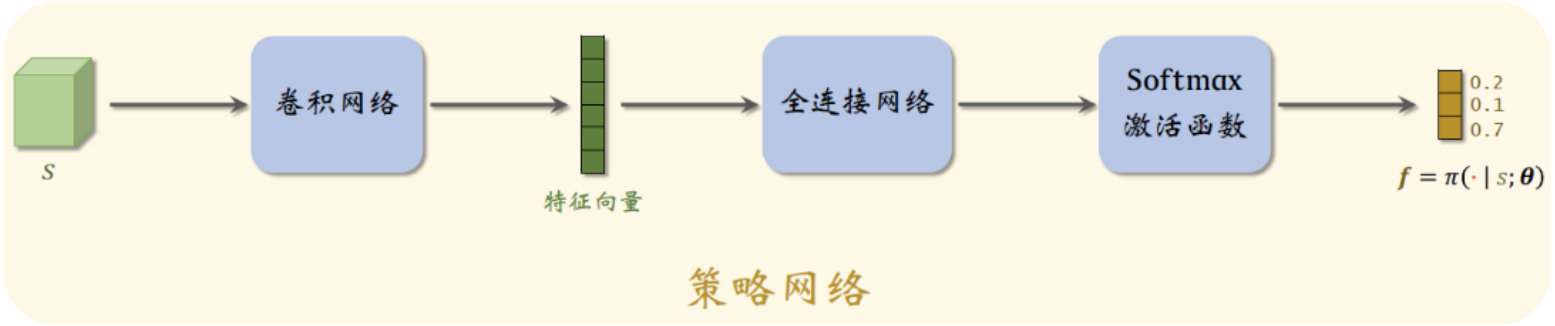
```python
class Agent:
  def __init__(self, ... ):
    self.q_tab_ = pd.DataFrame(columns=np.arange(self.act_dim_), dtype=np.float16)
     ...

  def save_q_table(self, file_path):
    self.q_tab_.to_csv(file_path)
    print('Q-table saved to "{}"'.format(file_path))

  def load_q_table(self, file_path):
    print('loading Q-table from "{}"'.format(file_path))
    self.q_tab_ = pd.read_csvfile_path, index_col=0, header=0, names=[None, 0, 1, 2, 3])
    print('Q-table loaded!')
```

	A	B	C	D	E
1		0	1	2	3
2	(36, {	-7.664356962	-15.61342041	-7.694123283	-7.693071842
3	36	-7.656120901	-17.01068437	-7.659941807	-7.658727695
4	24	-7.488716169	-7.350782471	-7.490020202	-7.496968378
5	25	-7.124733367	-6.981376435	-20.07079396	-7.124786756
6	13	-6.990925399	-6.990341748	-6.991266523	-6.994217257
7	14	-6.700412009	-6.698190073	-6.698361808	-6.701212021
8	15	-6.368743485	-6.366565564	-6.366847847	-6.368627362
9	26	-6.818439898	-6.600067902	-19.14250628	-6.822013243
10	2	-6.647001745	-6.642155266	-6.642302783	-6.642862761
11	3	-6.353653141	-6.351048334	-6.350616203	-6.353741718
12	16	-5.999468946	-5.997408597	-5.999115369	-6.003915488
13	28	-5.892364933	-5.759779413	-20.87596597	-5.90158946
14	29	-5.39499125	-5.288814012	-13.73746685	-5.423102958
15	17	-5.59019319	-5.585472918	-5.586260222	-5.59006599
16	18	-5.131872579	-5.128473004	-5.129937568	-5.128335551
17	19	-4.624321304	-4.619684917	-4.619560756	-4.627704493
18	7	-4.811033117	-4.809569618	-4.811106293	-4.809006624
19	8	-4.323064683	-4.323760683	-4.323466425	-4.331985324
20	9	-3.803057559	-3.796662609	-3.797573621	-3.79594759
21	20	-4.055526901	-4.050484874	-4.050587063	-4.051708733
22	10	-3.235079506	-3.230451421	-3.231130851	-3.231338969
23	22	-2.709890911	-2.69918533	-2.698955998	-2.707034012
24	21	-3.421837274	-3.412381131	-3.412421987	-3.416055405
25	32	-3.69280949	-3.523728955	-12.84052242	-3.717977642
26	31	-4.305954139	-4.232710947	-15.58826025	-4.278091061
27	12	-7.250527256	-7.250381854	-7.250166559	-7.253130957
28	0	-7.084272148	-7.08491206	-7.084840695	-7.085067153
29	1	-6.894724797	-6.892134361	-6.891356756	-6.894451004
30	4	-6.027327186	-6.022056629	-6.023548083	-6.027534686
31	27	-6.409808373	-6.190936333	-22.55240816	-6.423227557
32	5	-5.662919678	-5.656870222	-5.657204925	-5.657222938
33	6	-5.255794839	-5.252670062	-5.256256682	-5.252883592
34	30	-4.874706503	-4.761131626	-14.72340255	-4.851318103
35	34	-2.005557554	-1.90175027	-12.0064864	-2.065439659
36	33	-2.82488332	-2.716195212	-18.37441519	-2.806121284
37	11	-2.658119135	-2.64477383	-2.644856005	-2.654422566
38	23	-1.951956828	-1.928866753	-1.903510316	-1.940118114
39	35	-1.234805735	-1.12032612	-1	-1.191239915
40	47	0	0	0	0

基于策略的强化学习

- Q-learning 本质上是值函数近似算法。本质上还是一个value-based的方法
- 值函数近似算法都是**先学习动作-价值函数**，然后根据估计的动作价值函数选择动作。
- 如果没有动作价值函数的估计，策略也就不会存在。

但是，强化学习的目标，是学习最优策略。
那么有没有一种可能，我们可以跳过动作价值的评估环节，
直接从输入状态，到输出策略呢？



研究策略，就先将策略参数化

状态数量太多时

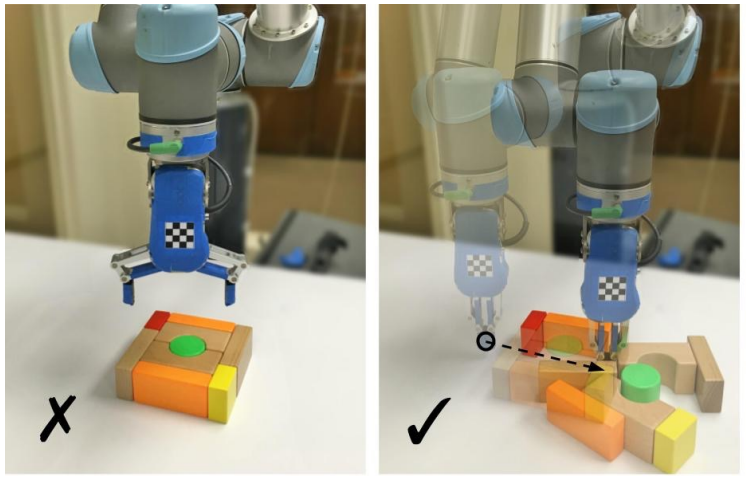
有些状态可能始终无法采样到，对这些状态的q函数进行估计很困难

状态数量无限时

不可能用一张表（数组）来记录q函数的值

若将强化学习算法应用在机械臂上，状态数量将是无限的：
1. 连续状态空间：
机械臂的状态通常由其关节角度、关节速度、末端执行器的位置和速度等参数组成。
这些参数通常是连续的实数值，而不是离散的有限集合。
例如，一个具有6个自由度的机械臂，其每个关节的角度和速度都是连续变量，这就导致状态空间是一个高维的连续空间。

2. 高维状态空间：
机械臂的每个关节都可以独立运动，导致状态空间的维度随着关节数量的增加而增加。
例如，一个6自由度的机械臂，如果每个关节的角度和速度都需要描述，那么状态空间的维度至少是12维（6个角度和6个速度）。



DQN(Deep Q-learning)

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，没探索过的状态也可通过周围的状态来估计

初始化q_π函数的参数$\boldsymbol{\theta}$
循环
　　初始化s为初始状态
　　循环
　　采样$a \sim \epsilon-greedy_\pi(s;\boldsymbol{\theta})$
　　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

DQN(Deep Q-learning)

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，没探索过的状态也可通过周围的状态来估计

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

Value-Based RL: DQN

- SARSA通过直接更新q值，从而不依赖于转移函数 $P\ (s'|s,a)$
- Q-Learning相比SARSA，摆脱当前策略直接优化目标，更高效，最终效果更好，但会做出一些更冒险的决策
- Q-Learning用一个 Q 函数计算出的目标值 (R + γ * max Q)，来逼近和更新它自己对当前状态-动作对 (s, a) 的估计值 Q(s, a)
- DQN用神经网络直接拟合策略，拓展了Q-learning，解决了现实问题中状态空间过大难建模的问题

目录

策略梯度法（Policy Gradient Methods）

通过直接**参数化**（即用神经网络来表示）策略函数的方法求解强化学习问题，由于算法需要求参数化的策略函数的梯度，因此这些方法称为**策略梯度法（Policy Gradient Methods）**

- 策略函数的参数化可以表示为$\pi_{\boldsymbol{\theta}}(s,a)$，其中$\boldsymbol{\theta}$为一组参数，函数取值表示在状态$s$下选择动作$a$的概率。和Q学习的$\epsilon$贪心策略相比，这种参数化的一个显著好处是：选择一个动作的概率是随着参数的改变而光滑变化的，实际上这种光滑性对算法收敛有更好的保证。
- 为了优化策略函数，首先需要**有一个对策略函数优劣进行衡量的标准**。不妨假设强化学习问题的初始状态为s_0，不难定义算法希望达到的**最大化目标为**

$$J(\boldsymbol{\theta}) \coloneqq V_{\pi_{\boldsymbol{\theta}}}(s_0)$$

- 这是一个关于参数$\boldsymbol{\theta}$的函数，其值为从初始状态出发的价值函数。如果能求出梯度$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$，就可以用梯度上升（Gradient Ascent）法求解这个问题了，即**求解价值函数的最大值**。

策略梯度法（Policy Gradient Methods）

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{s} \mu_{\pi_{\boldsymbol{\theta}}}(s) \sum_{a} q_{\pi_{\boldsymbol{\theta}}}(s,a)\pi_{\boldsymbol{\theta}}(s,a) \propto \sum_{s} \mu_{\pi_{\boldsymbol{\theta}}}(s) \sum_{a} q_{\pi_{\boldsymbol{\theta}}}(s,a)\boldsymbol{\nabla_{\theta}\pi_{\theta}(s,a)}$$

因为$\nabla\pi=\pi\cdot\nabla\ln\pi$

$$= \mathbb{E}_{S,\mathcal{A}}\left[\nabla_{\theta}\ln\pi_{\theta}(\mathrm{s},a) q_{\pi_{\theta}}(s,a)\right]$$

$\mu_{\pi_{\boldsymbol{\theta}}}(s)$称为策略$\pi_{\boldsymbol{\theta}}$的策略分布（这里假设折扣系数$\gamma = 1$）。

- 在连续问题中，$\mu_{\pi_{\boldsymbol{\theta}}}(s)$为算法在策略$\pi_{\boldsymbol{\theta}}$安排下从$s_0$出发经过无限多步后位于状态$s$的概率；
- 在离散问题中，$\mu_{\pi_{\boldsymbol{\theta}}}(s)$为归一化后的算法从$s_0$出发访问$s$次数的期望。当$\gamma \in (0,1)$时，则需要给每个状态的$\mu_{\pi_{\boldsymbol{\theta}}}(s)$值加上一个权重。在保证算法通用性的前提下，为了简化说明，下文在进行公式推导时始终假设$\gamma = 1$，但是在算法流程中会体现折扣系数γ的影响。

上式称为**策略梯度定理（Policy Gradient Theorem）**，它提供了策略梯度法的求解思路：

如果能够计算或估计策略函数的梯度，智能体就能直接对策略函数进行优化。

策略梯度法（Policy Gradient Methods）

- 利用得到的梯度，我们就可以基于梯度下降更新参数

- 第一步，梯度下降

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \boldsymbol{\theta_t} + \alpha \mathbb{E}_{S,\mathcal{A}} \left[\nabla_\theta \ln \pi_\theta(\mathrm{s}, a) q_{\pi_\theta}(s, a) \right]$$

- 第二步，随机找一个样本

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) = \boldsymbol{\theta_t} + \alpha \nabla_\theta \ln \pi_\theta(s_t, a_t) q_{\pi_\theta}(s_t, a_t)$$

- 第三步，因为q未知，想办法估计一下当前的动作价值q_{π_θ}

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) = \boldsymbol{\theta_t} + \alpha \nabla_\theta \ln \pi_\theta(s_t, a_t) q_{\mathrm{t}}(s_t, a_t)$$

策略梯度法（Policy Gradient Methods）

- 如果用蒙特卡洛方法来估计q值，这个算法就是REINFORCE

Pseudocode: Policy Gradient by Monte Carlo (REINFORCE)

Initialization: Initial parameter θ; $\gamma \in (0, 1)$; $\alpha > 0$.
Goal: Learn an optimal policy to maximize $J(\theta)$.

For each episode, do
  Generate an episode $\{s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T\}$ following $\pi(\theta)$.
  For $t = 0, 1, \ldots, T-1$:
    Value update: $q_t(s_t, a_t) = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    Policy update: $\theta \leftarrow \theta + \alpha \nabla_\theta \ln \pi(a_t|s_t, \theta) q_t(s_t, a_t)$

策略梯度——离散动作空间

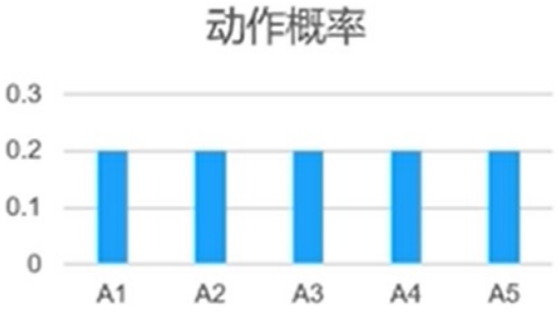
对于随机策略$\pi_\theta(a|s) = P(a|s;\theta)$

直觉上我们应该

- 降低带来较低价值/奖励的动作出现的概率
- 提高带来较高价值/奖励的动作出现的概率

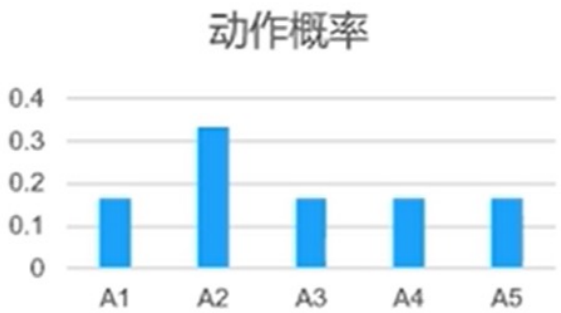
一个离散动作空间维度为5的例子

1. 初始化θ



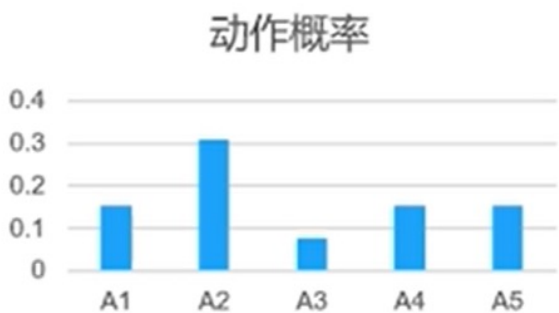
2. 采取动作A2
观察到正的奖励

3. 根据策略梯度更新θ

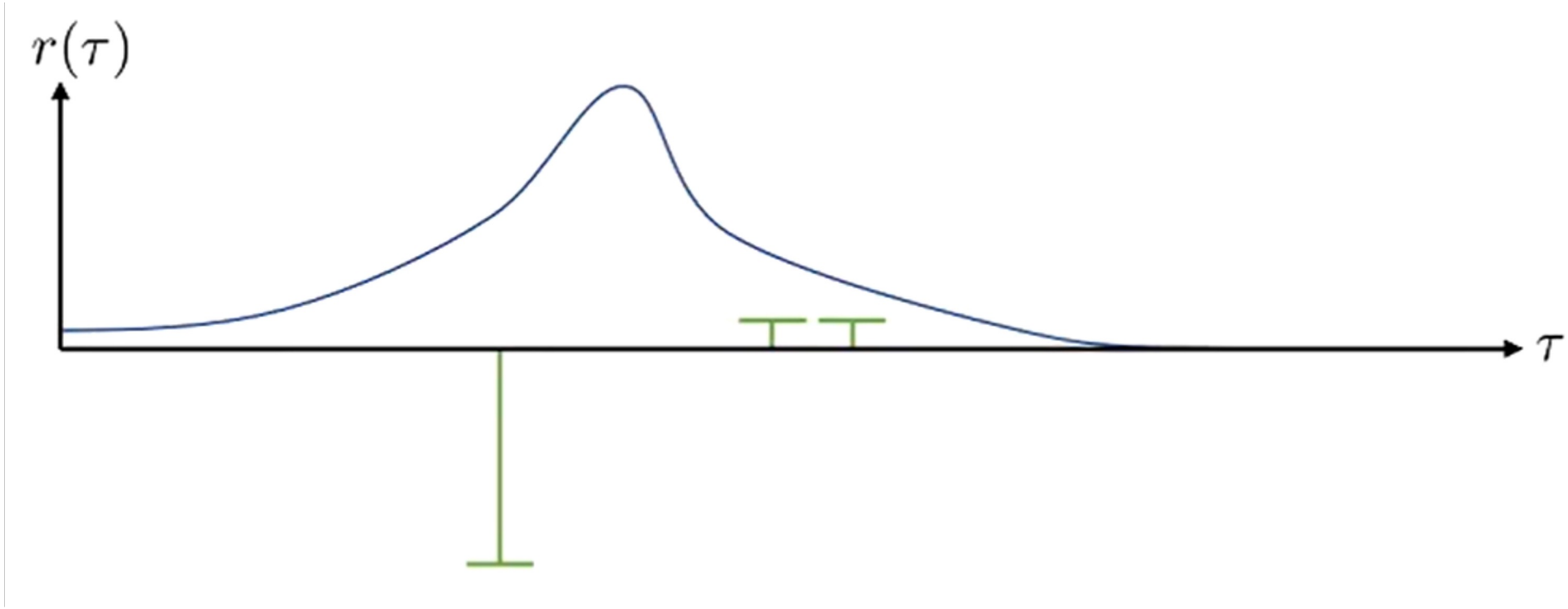


4. 采取动作A3
观察到负的奖励

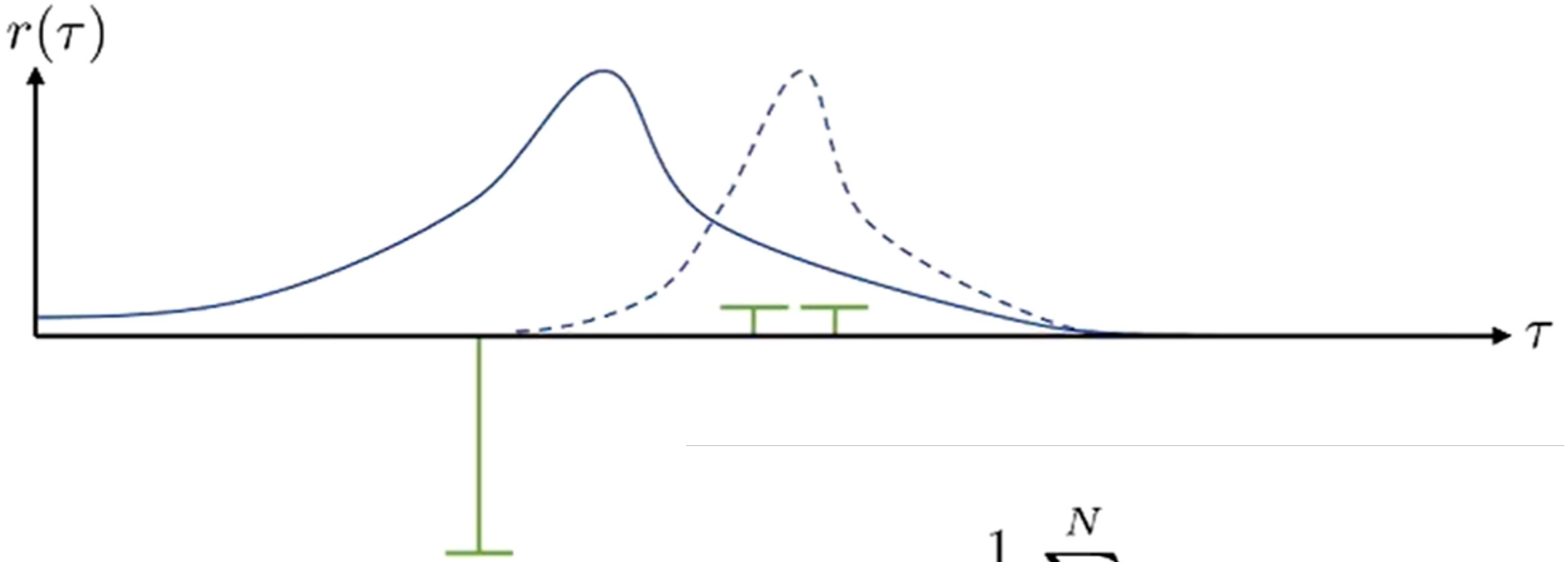
5. 根据策略梯度更新θ



策略梯度——连续状态空间



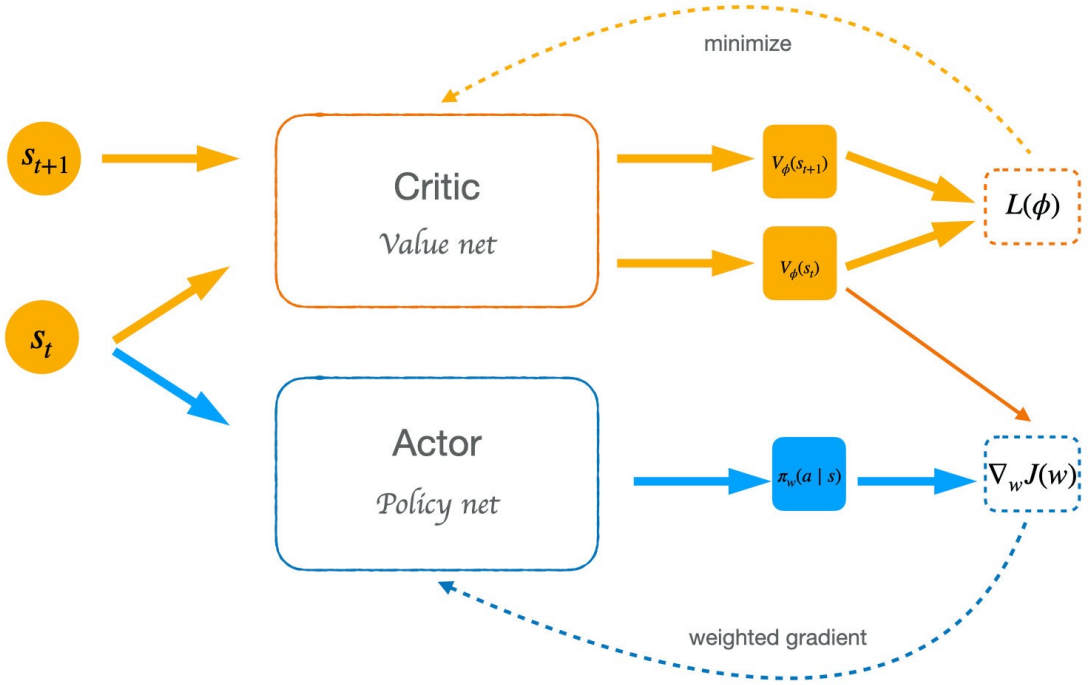
策略梯度——连续状态空间

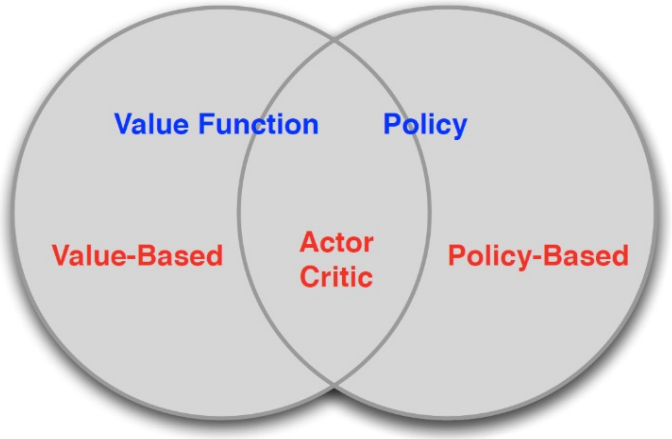


$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$

Actor-Critic算法

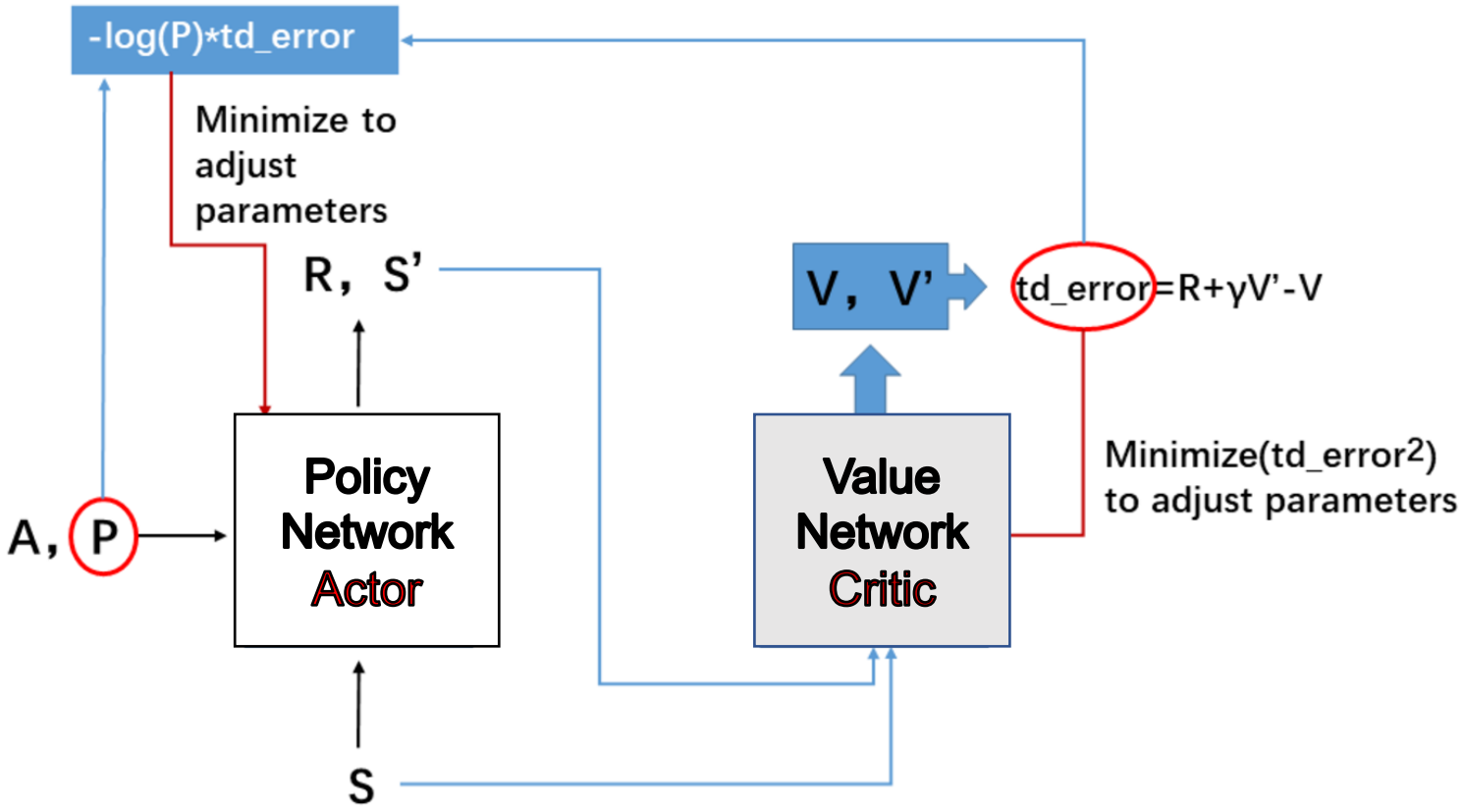
- 同时设计Policy和Value的显式计算





Actor-Critic算法

是否能够从时序差分的角度设计一个策略梯度求解方法呢?
现代强化学习中最重要的方法：**Actor-Critic算法**



- **演员（actor）**，负责动作的选择，代表策略——policy network
- 价值函数就像一个**评论家（critic）**，评价执行者选出的动作的好坏——value network

- Actor在状态S下，根据策略网络P做出行动A，得到神经网络R和S’ 的输出，将其输出反馈给Critic；
- 经过Critic神经网络计算得出误差td_error；
- td-error用来更新Critic自身的神经网络，使Critic更加准确地估计出动作-价值函数，同时也将传输给Actor神经网络，更新Actor的神经网络参数。

Proximal Policy Optimization算法

[Schulman et al., 2017] 将Actor-Critic框架与信任域优化技术相结合，提出：近端策略优化（Proximal Policy Optimization，**PPO**）算法。

- PPO算法通过限制策略更新的幅度，从而提高算法的稳定性。
- OpenAI当年在发布PPO算法时称之为 **Default** Reinforcement Learning算法，足见其易用性与效果之惊艳。

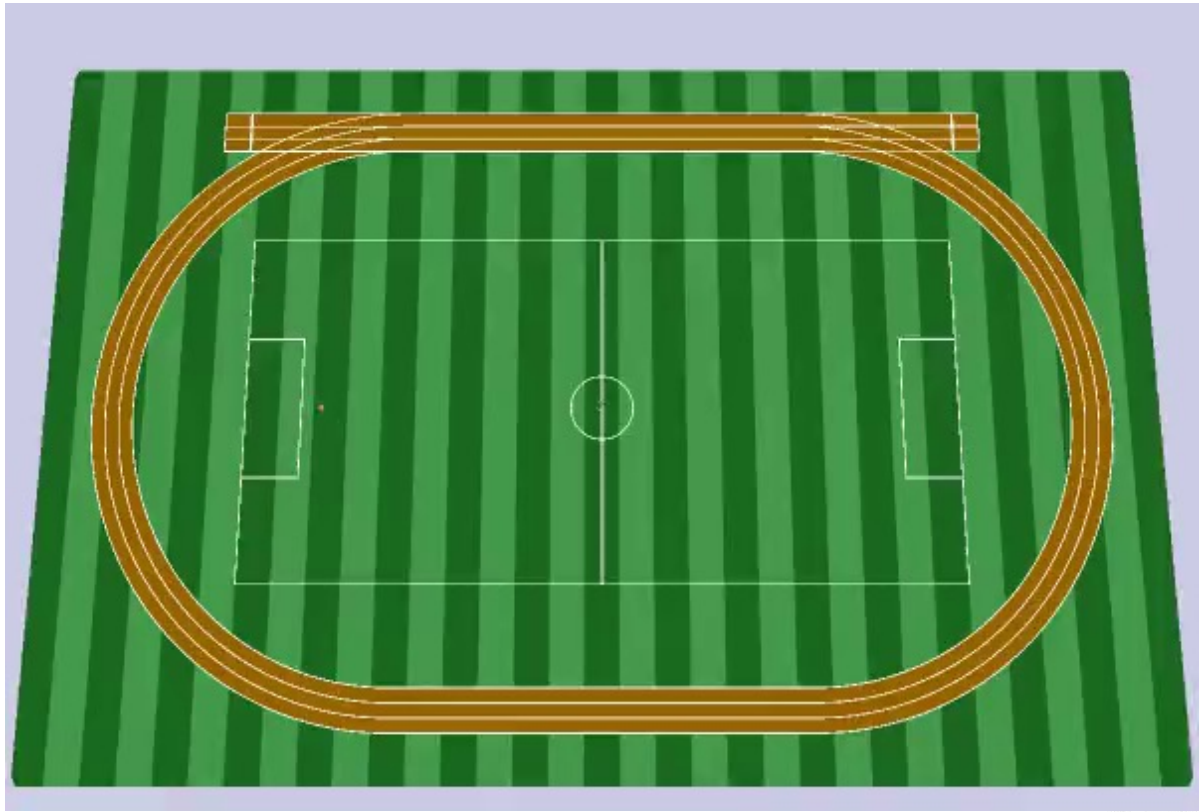
Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

2w次引用

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.



Proximal Policy Optimization算法

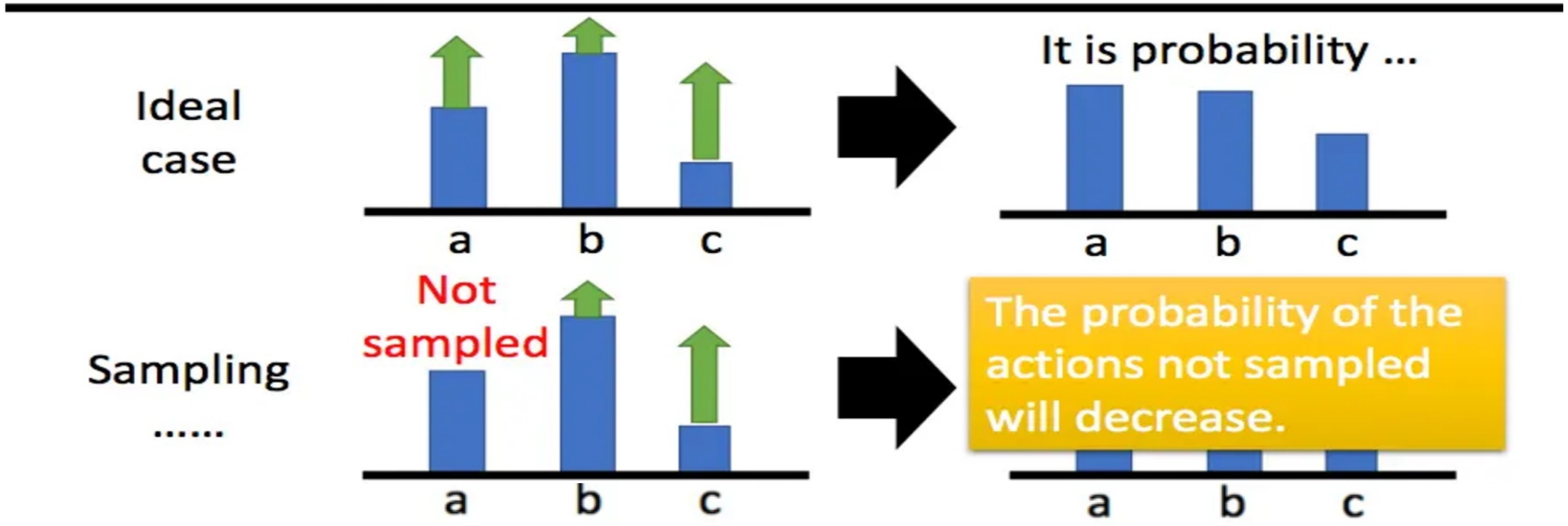
Tip1:增加一个基线(baseline)

策略梯度方法在更新策略时，基本思想就是增加reward大的动作出现的概率，减小reward小的策略出现的概率。那如果奖励一直为正怎么办？增加一个奖励的基线，让reward有正有负。一般增加的基线是所获得奖励的平均值：

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

It is possible that $R(\tau^n)$ is always positive.

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla log p_\theta(a_t^n | s_t^n) \qquad b \approx E[R(\tau)]$$



Proximal Policy Optimization算法

Tip2:重要性采样（importance sampling）

PPO想要做到一次采样多次训练的效果，让采样到的数据可以重复使用；

这就涉及到采样的问题：无法直接从分布p(x)（当前策略下的行为分布）采样，但可以从另一个分布q(x)（旧策略下的行为分布）采样，这样的数据怎么用呢？用重要性权重来修正采样偏差，若p(x)和q(x)偏差大，则这个样本在估值中贡献大，反之则说明这个样本重要性低。

Importance Sampling

$$E_{x\sim p}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x^i)$$

x^i is sampled from $p(x)$

We only have x^i sampled from $q(x)$

$$= \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim q}[f(x)\frac{p(x)}{q(x)}]$$

当前策略/旧策略

Importance weight

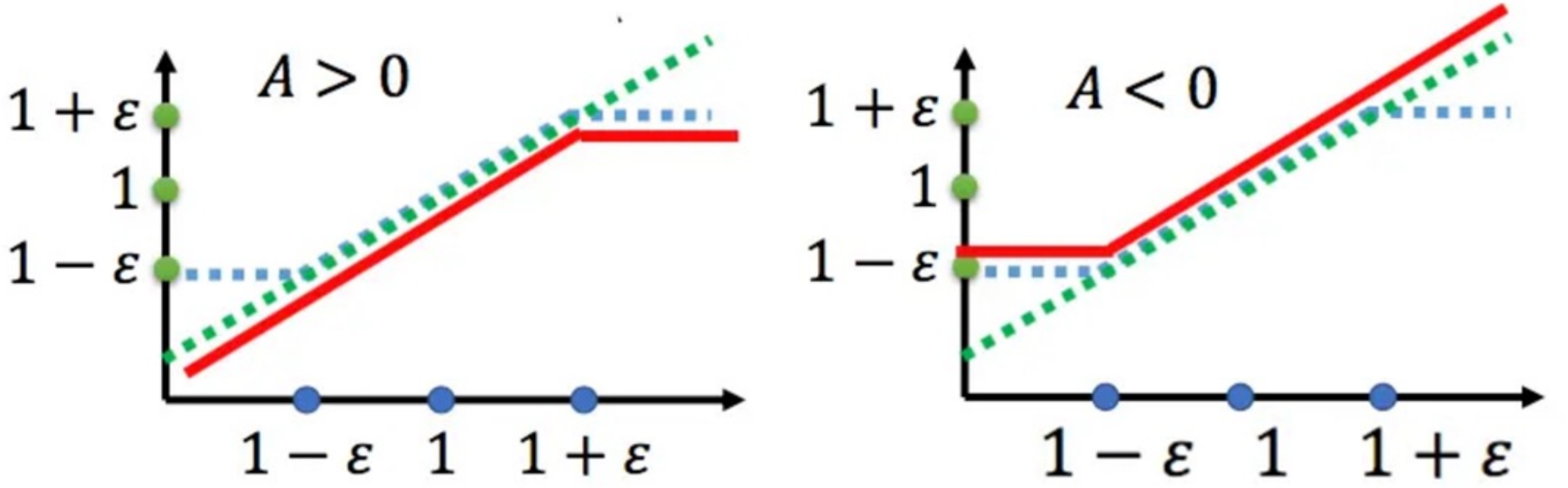
Proximal Policy Optimization算法

Tip3: clip裁剪防止更新梯度过大

我们希望θ和θ'不能差太远，这并不是说参数的值不能差太多，而是说，输入同样的state，网络得到的动作的概率分布不能差太远。

下图中绿色的线代表min中的第一项，即不做任何处理；蓝色的线为第二项，如果两个分布差距太大，则进行一定程度的裁剪。最后对这两项再取min，防止了θ更新太快。

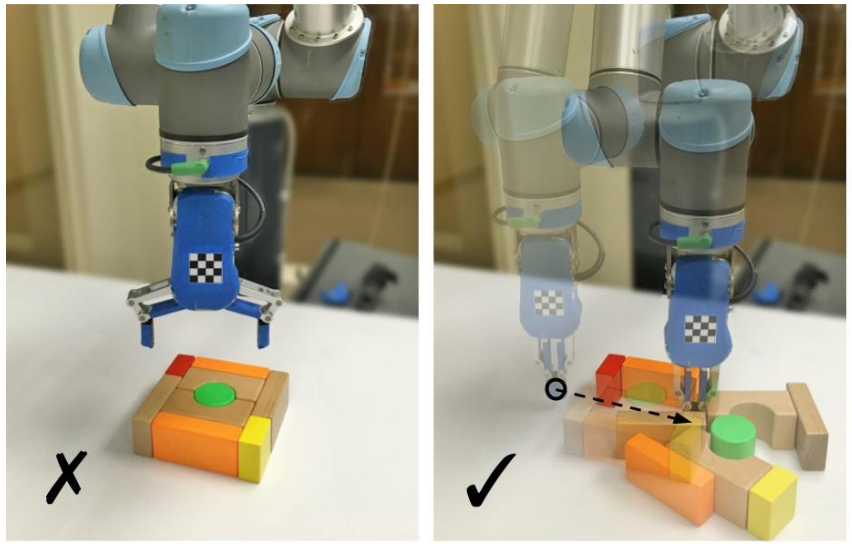
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

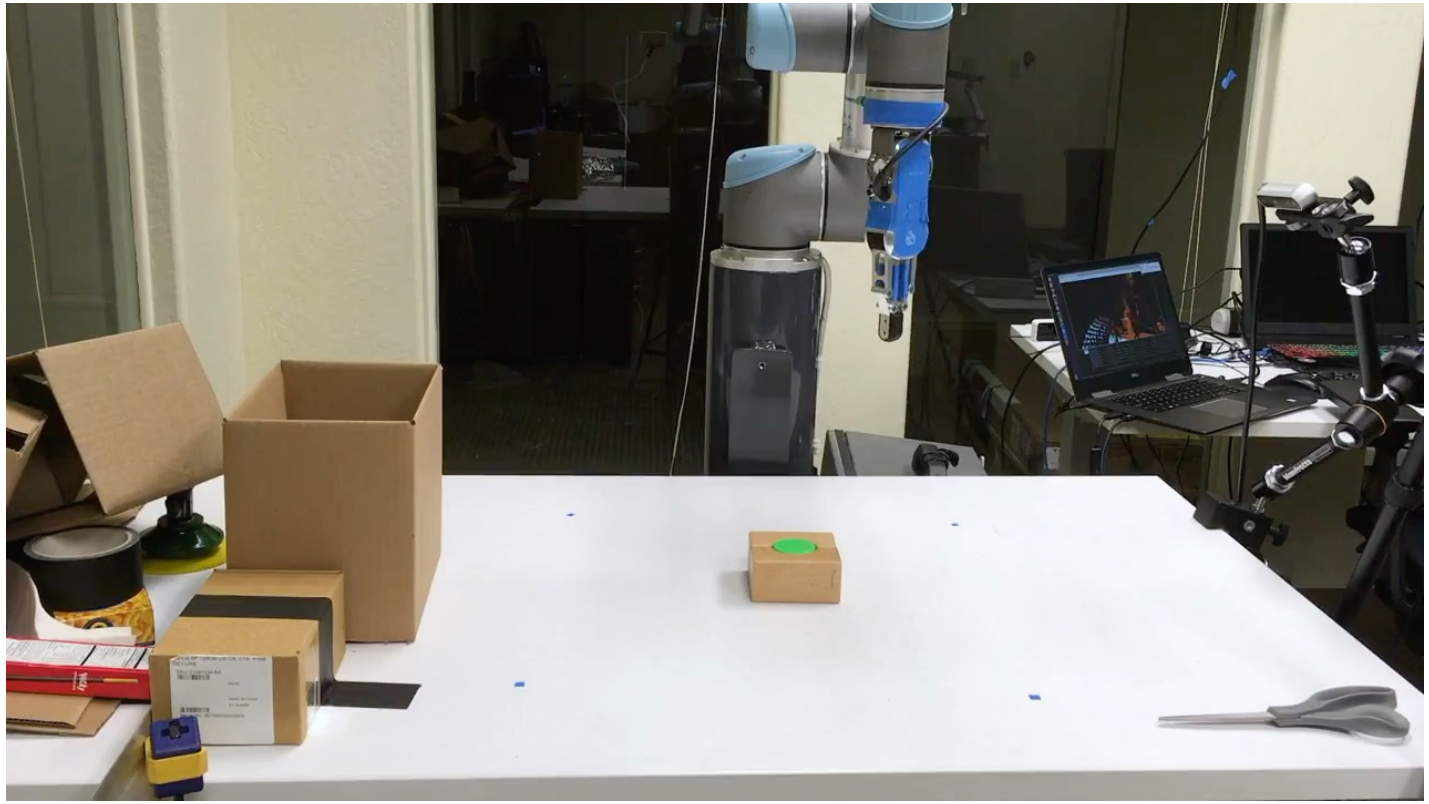


Policy-Based RL: PPO

- 策略梯度法的目标是直接优化策略参数 θ，以最大化期望回报 $J(\theta)$
- 其关键突破在于使用对数似然技巧 $\nabla_\theta \pi_\theta = \pi_\theta \cdot \nabla_\theta \ln \pi_\theta$，将对期望的梯度（一个难以通过采样解决的积分问题）转化为一个期望内的梯度（一个可以通过采样轻松估计的期望问题）
- Actor-Critic是Value-based和Policy-based的结合
- PPO通过工程化技巧成为了广为应用RL技术

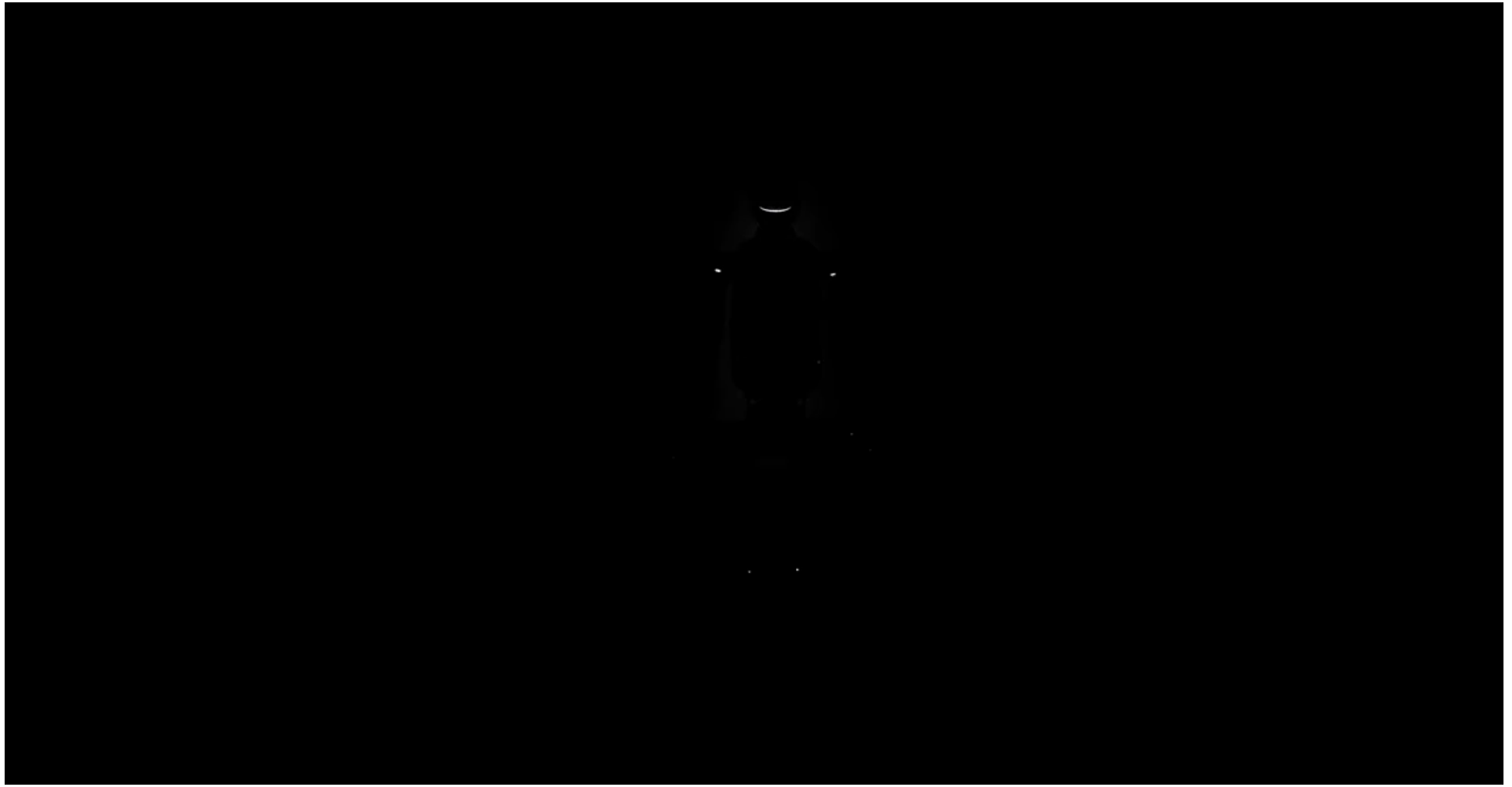
What’s Next：具身智能





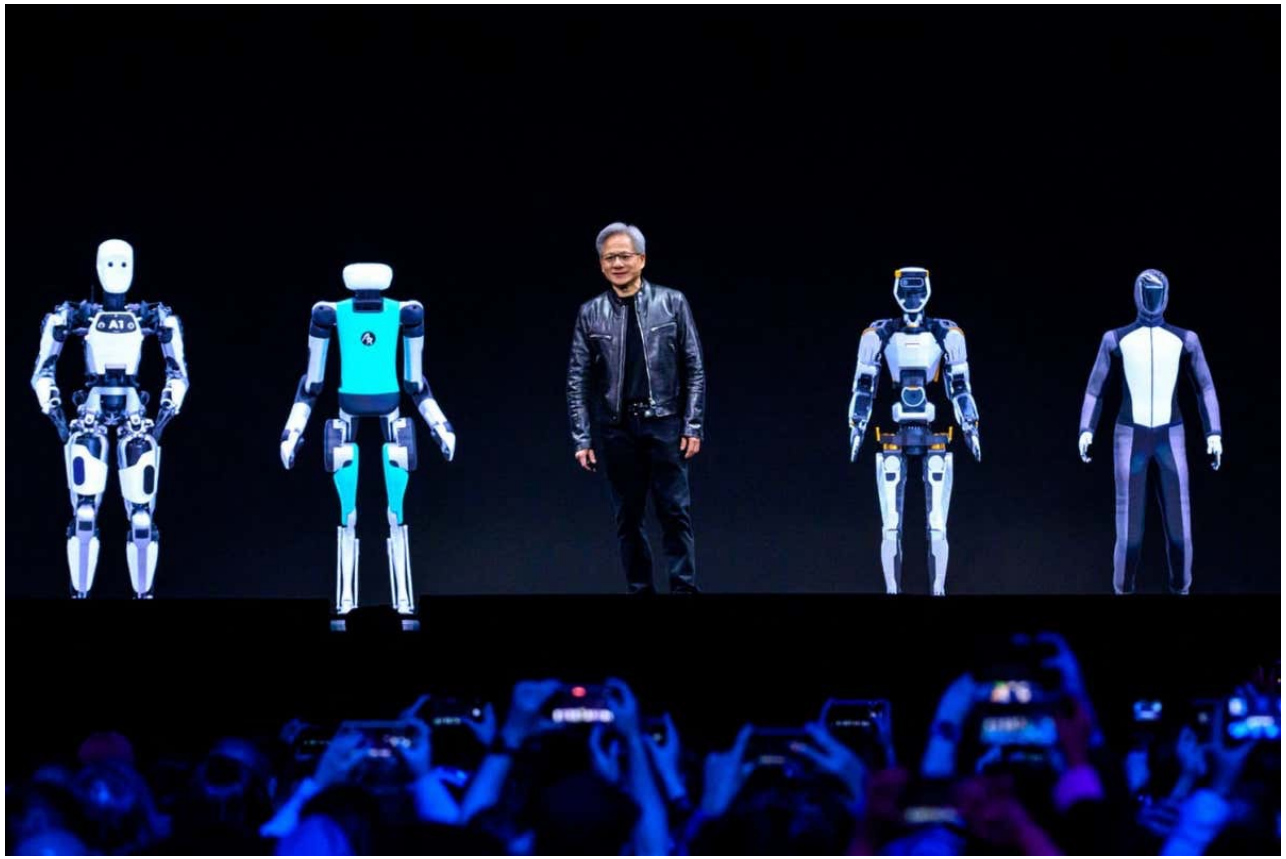
https://vpg.cs.princeton.edu/

具身智能



https://www.figure.ai/

具身智能



英伟达发布人形机器人通用基础模型Project GR00T和新型计算机 Jetson Thor

18 March 2024

具身智能

Tesla's Optimus robots were serving drinks, handing out gift bags, and dancing at the company's 'We, Robot' event

9 October 2024

https://www.tesla.com/we-robot

https://www.youtube.com/watch?v=Q9Ze7OSfZzE

小结

- RL采用与传统机器学习不同的问题定义方式
- 其理论普遍建立在马尔可夫决策过程与贝尔曼方程之上
- 主要分为两大流派：
  - **Value-Based**（SARSA->Q-Learning->DQN）
  - **Policy-Based**（Actor-Critic->PPO）
- 这些代表性方法大多是对早期理论的工程化实现与改进。
- 现代RL的解决方案倾向于融合两大流派，形成混合方法。