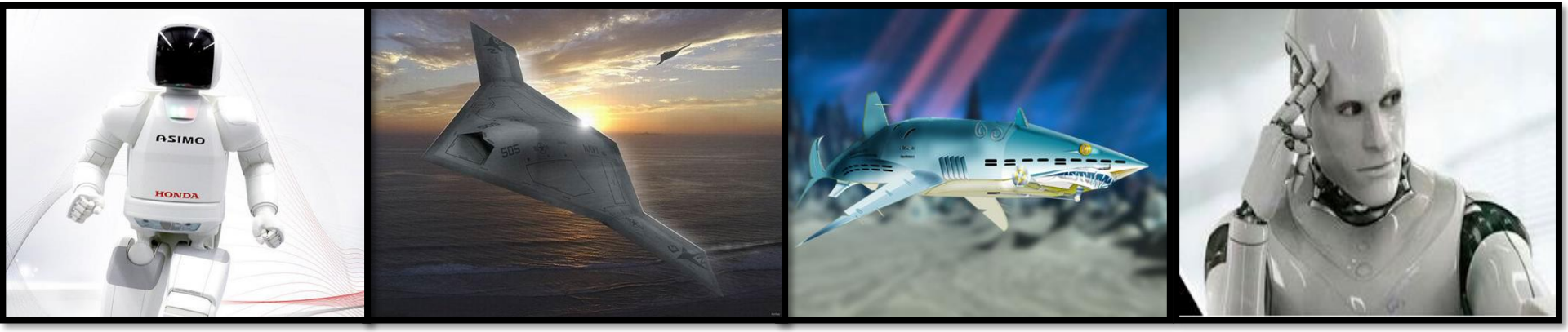
机器人智能控制

易建强 蒲志强 袁如意

中国科学院自动化研究所

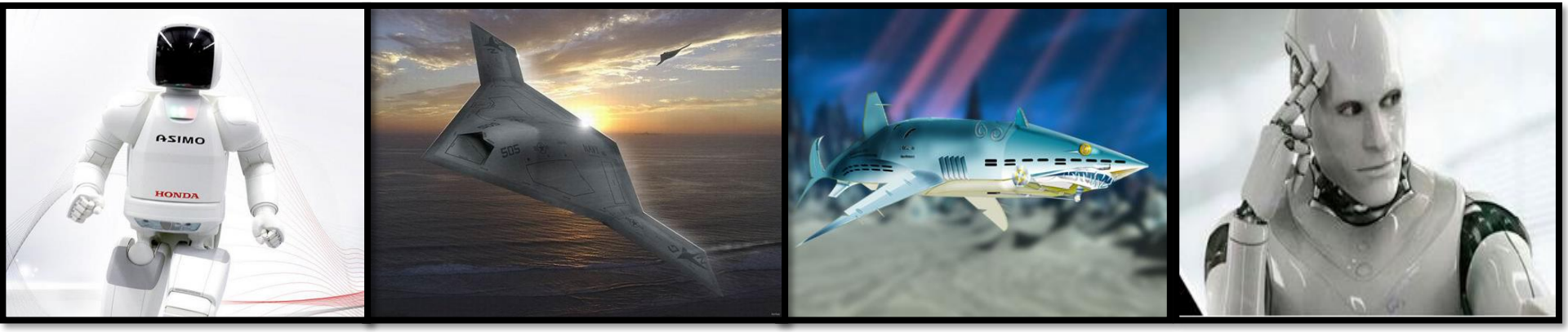
2020年秋季



强化学习

袁如意 高级工程师

ruyi.yuan@ia.ac.cn



本讲的主要内容

一、强化学习基本概念

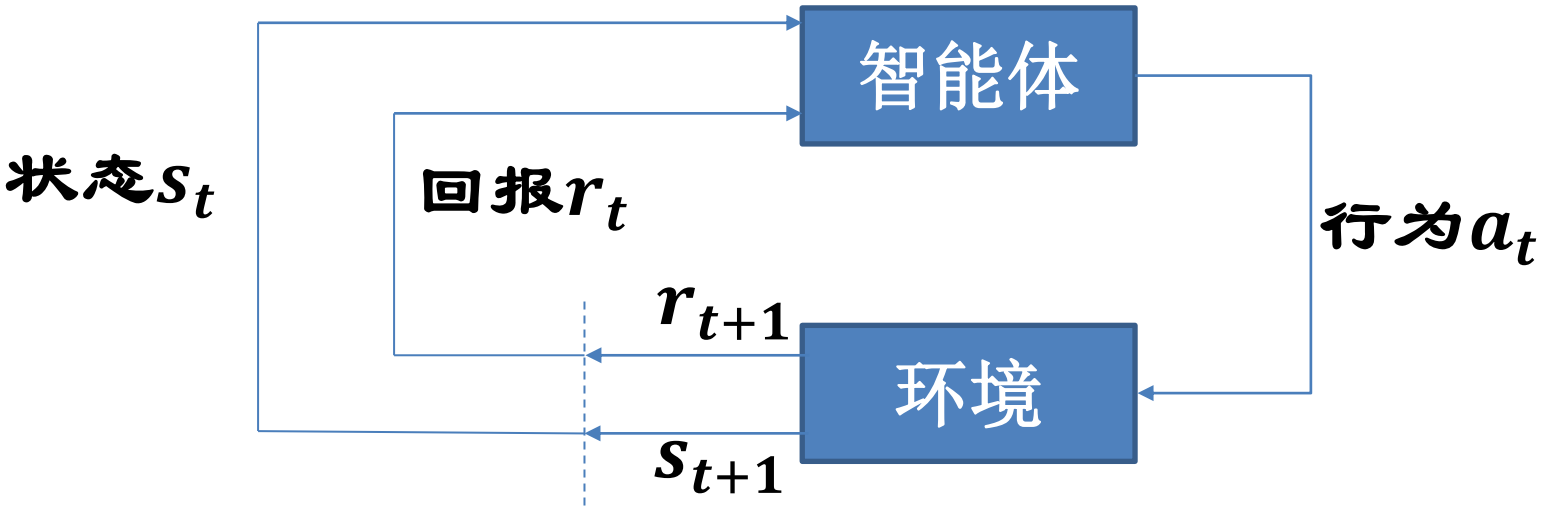
二、基于值函数的强化学习方法

三、基于策略搜索的强化学习方法

四、应用

强化学习(Reinforcement Learning)

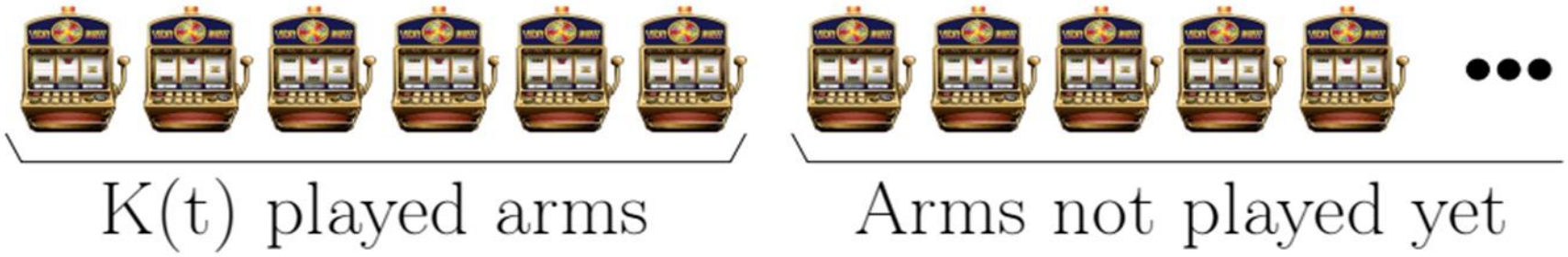
- 增强学习、再励学习
- 学习状态和行为之间的映射，使得系统行为从环境中获得的累积奖励值最大
- 主动试探，尝试与失败（trial-and-error），评价性反馈
- 某个动作导致环境正的回报，智能体产生这个动作的趋势会加强，反之趋势会减弱



强化学习

- 智能体为适应环境具有如下特征，可称为强化学习：
  - 智能体不是静止的、被动的等待，而是主动对环境做出试探
  - 环境对试探动作的反馈信息是评价性的（好或坏）
  - 智能体在行动—评价的环境中获取知识，改进行动方案以适应环境，达到预期目的。
- 弱的学习方式
  - 需要通过与环境不断交互试错来学习
  - 反馈信号为回报，不是教师信号
  - 回报信号可能是稀疏和合理延迟的
  - 不要求或要求较少的先验知识

N臂赌博机问题 (The N-armed bandit)



如何通过学习能够选择拉动可获得最大期望回报的臂？

$$Q_t(a) = \frac{r_1 + r_2 + \cdots r_k}{k}$$

r_i: 时刻i选择臂a的回报

t次尝试，k次选择臂a

开发与探险(Exploration-exploitation) tradeoff:

- ✓**Exploit: act optimally according to our current beliefs**
- ✓**Explore: learn more about the environment**

N臂赌博机问题 (The N-armed bandit)

10臂赌博机，假定实际期望回报为服从N(0,1)的正态分布

best choice

$Q^*(a) = [-0.4 \quad 1.3 \quad 0.04 \quad 0.53 \quad -0.15 \quad -1.01 \quad 0.2 \quad 1.48 \quad 0.36 \quad -0.5]$

初始选择（猜测）

$Q_{est}(0) = [0.05 \quad 0.86 \quad -0.96 \quad 0.73 \quad 1.98 \quad -1.19 \quad -0.66 \quad 0.82 \quad 1.97 \quad -0.13]$

贪婪策略，最大化开发(Exploration)

第1次更新

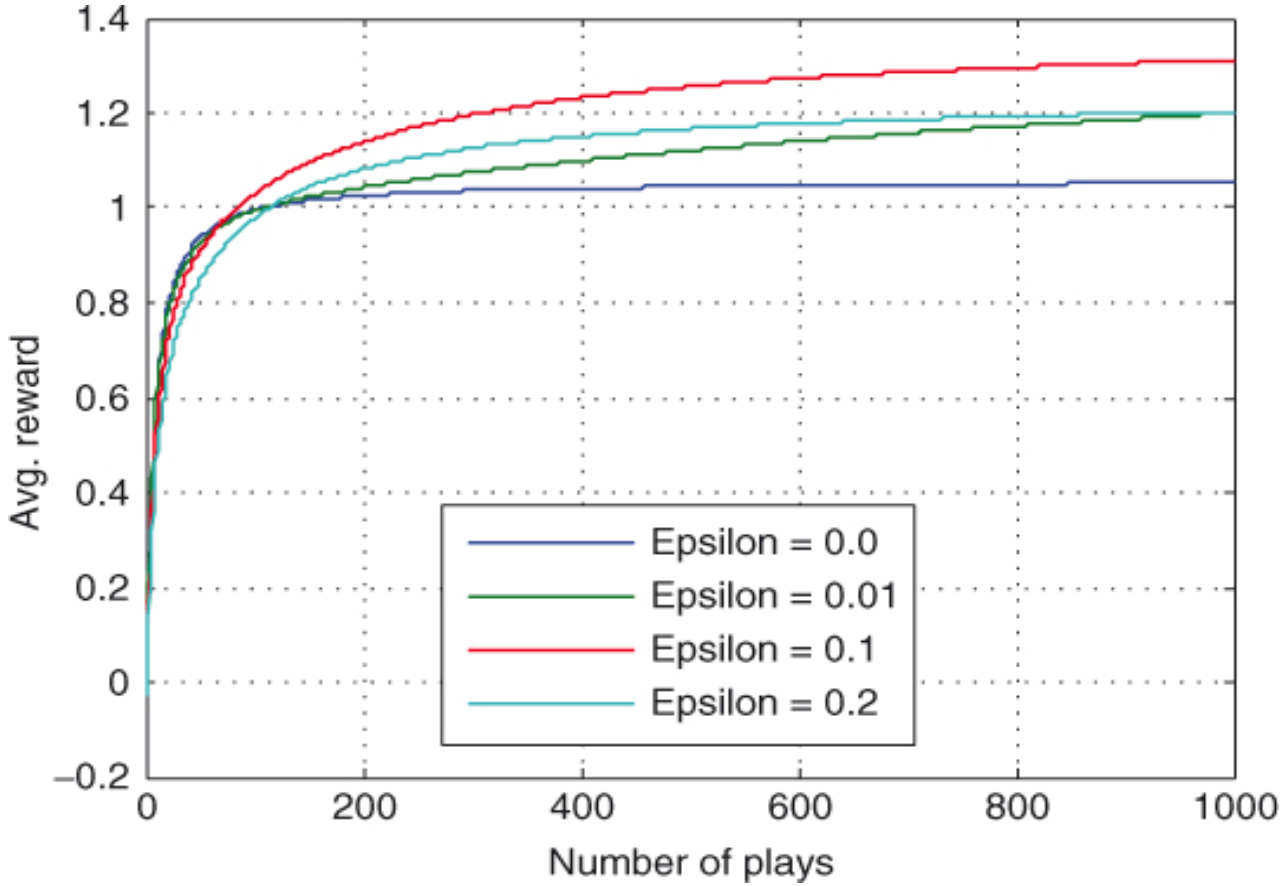
$Q_{est}(1) = [0.05 \quad 0.86 \quad -0.96 \quad 0.73 \quad -0.76 \quad -1.19 \quad -0.66 \quad 0.82 \quad 1.97 \quad -0.13]$

第2次更新 $N(-0.15,1) \quad Q_1(5) = -0.76$ $\quad Q_2(9) = 1.50$

$Q_{est}(2) = [0.05 \quad 0.86 \quad -0.96 \quad 0.73 \quad -0.76 \quad -1.19 \quad -0.66 \quad 0.82 \quad 1.5 \quad -0.13]$

ε - 贪婪方法(ε-greedy method),增强探索(exploitation)性

N臂赌博机问题 (The N-armed bandit)



不同ϵ值下的多臂赌徒机

强化学习基本要素

- 状态s是对环境的描述，可以是离散的或连续的，其状态空间为S；
- 动作a是对智能体行为的描述，可以是离散的或连续的，其动作空间为A；
- 策略$\pi(a|s)$是智能体根据环境状态s来决定下一步的动作a的函数；智能体完成状态到每种可能行为的选择概率之间的映射。$\pi(a|s)$为状态s时选择动作为a的概率

✓ 确定性策略（deterministic policy）

从状态空间到动作空间的映射函数π: $S \to A$

✓ 随机性策略（stochastic policy）

$$\pi(a|s) \triangleq p(a|s) \qquad \sum_{a \in A} \pi(a|s) = 1$$

引入一定随机性能更好地探索环境，能使策略具有多样性

强化学习基本要素

- 状态转移概率$p(s'|s,a)$是在智能体根据当前状态s做出一个动作a之后，环境在下一个时刻转变为状态s'的概率

- 即时奖励$r(s,a,s')$是一个标量函数，即智能体根据当前状态s做出动作a之后，环境会反馈给智能体一个奖励，这个奖励也经常和下一个时刻的状态s'有关。

强化学习的基本模型–MDP

考虑智能体与环境交互的离散时间序列

$$s_0, a_0, s_1, r_1, a_1, \cdots, s_{t-1}, r_{t-1}, a_{t-1}, s_t, r_t, \cdots,$$

$r_t = r(s_{t-1}, a_{t-1}, s_t)$为$t$时刻的即时奖励

- 马尔科夫过程（Markov process）：系统下一状态s_{t+1}仅与当前状态s_t相关，而与以前状态无关

$$p(s_{t+1}|s_1, \cdots, s_t) = p(s_{t+1}|s_t)$$

$p(s_{t+1}|s_t)$为状态转移概率

$$\sum_{s_{t+1} \in S} p(s_{t+1}|s_t) = 1$$

强化学习的基本模型–MDP

■ 马尔科夫决策过程（Markov decision process，MDP）

在马尔科夫过程中引入动作变量：动作a，即下一个时刻的状态s_{t+1}和当前时刻的状态s_t以及动作a_t相关

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

给定策略$\pi(a|s)$,马尔可夫决策过程的一个轨迹(trajectory)

$$\tau = s_0, a_0, s_1, a_1, \cdots, s_{T-1}, a_{T-1}, s_T, r_T$$

概率为

$$p(\tau) = p(s_0, a_0, s_1, a_1, \cdots)$$

$$= p(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

强化学习的基本模型–MDP

马尔科夫决策模型 (S,A,P,R)

S：环境状态空间

A：动作空间

P：状态转移矩阵

R：回报值或回报函数

■ 延迟或稀疏的回报：

执行动作之后反馈给学习器的只有一个立即回报，甚至没有有效的反馈，根据立即回报并不能直接判断动作的优劣。大多数时候智能体需要执行一系列的动作之后才能得到一个成功的反馈

强化学习

■ 时间信用分配（temporal credit assignment）

- 怎样确定最终回报的生成归功于动作序列的哪一个动作，或者怎样确定这一动作序列对最终回报的贡献率？[1961 Minsky]

■ 探索与利用平衡

- 影响强化学习效率
- 在获取知识与高回报之间折中
- 实质：设计合适的探索策略

强化学习目标函数

■ 总回报（return）

- 回合（episode）或试验（trial）：假设环境中有一个或多个特殊的终止状态（terminal state），当到达终止状态时，一个智能体和环境的交互过程就结束

- **有限阶段总回报**：给定策略$\pi(a|s)$，智能体和环境一次交互过程的轨迹τ所收到的累积奖励为总回报

$$G(\tau) = \sum_{t=0}^{T-1} r_{t+1} = \sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})$$

强化学习目标函数

- **无限折扣总回报**：如果环境中没有终止状态(持续性的任务)，即$T = \infty$，其总回报也可能是无穷大。
  - 引入折扣率来降低远期回报的权重，折扣回报（discounted return）定义为

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}, \quad \gamma \in [0,1)$$

  - γ接近于0，智能体更在意短期回报
  - γ接近于1，智能体更在意长期回报
  - 具有有用的理论性质
  - 可以产生稳定的最优策略，即给定状态，最优策略是相同的

$$r_i \le R \Rightarrow G(\tau) \le \frac{R}{1-\gamma}$$

强化学习目标函数

■ 期望回报

因为策略和状态转移都有一定的随机性，每次试验得到的轨迹是一个随机序列，其收获的总回报也不一样。一个策略$\pi(a|s)$的**期望回报**（expected return）为

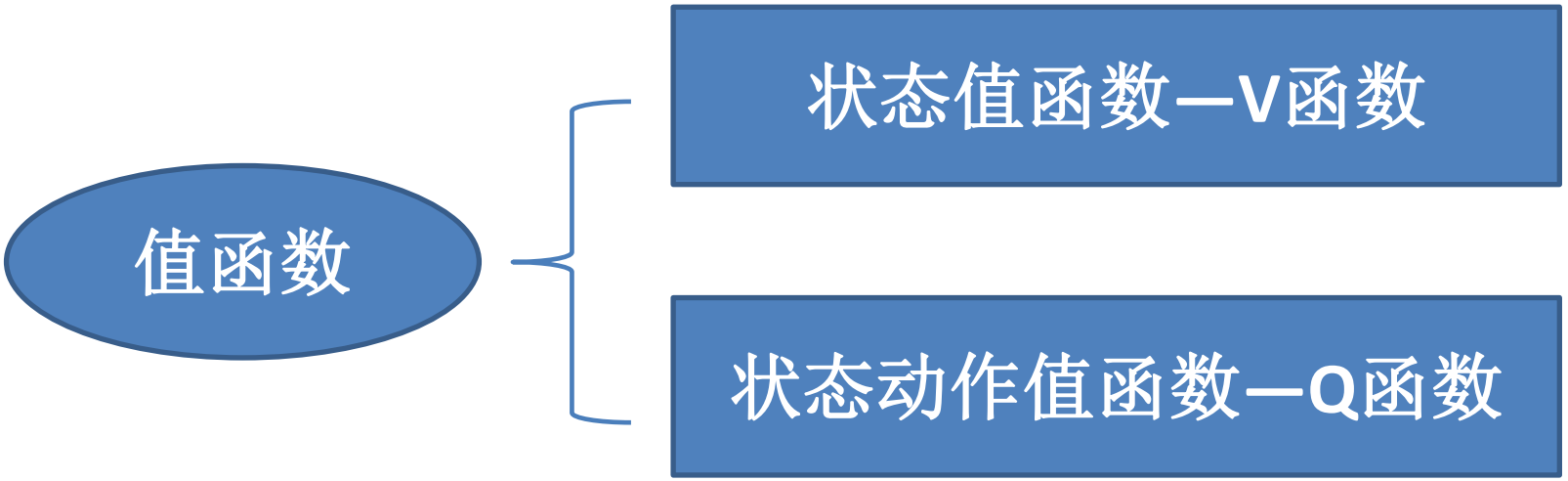
$$E_{\tau\sim p(\tau)}[G(\tau)] = E_{\tau\sim p(\tau)}\left[\sum_{t=0}^{T-1}\gamma^t r_{t+1}\right]$$

■ **强化学习目标**：学习策略使得$\pi_\theta(a|s)$最大化期望回报

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{T-1}\gamma^t r_{t+1}\right]$$

值函数

- 用来衡量某一状态或动作-状态的优劣（对回报函数的一种预测），即对智能体来说是否值得选择某一状态或在某一状态下执行某一动作



一个状态s可能会产生一个较低的立即回报r，但从长远来看可能会带来更高的V值，或Q值。选择能带来最大值函数的动作，而不是选择能带来最大立即回报的动作

状态值函数—V函数

期望回报可以分解

$$E_{\tau\sim p(\tau)}[G(\tau)] = E_{s\sim p(s_0)}\left[E_{\tau\sim p(\tau)}\left[\sum_{t=0}^{T-1}\gamma^t r_{t+1}\,|\tau_{s_0}=s\right]\right] = E_{s\sim p(s_0)}[V^{\pi}(s)]$$

- 状态值函数$V^{\pi}(s)$（state value function）：从状态s开始，执行策略π得到的期望总回报

$$V^{\pi}(s) = E_{\tau\sim p(\tau)}\left[\sum_{t=0}^{T-1}\gamma^t r_{t+1}\,|\tau_{s_0}=s\right]$$

状态值函数：累积回报在状态s处的期望值

状态动作值函数—Q函数

■ **状态-动作值函数**（state-action value function）

初始状态为s并进行动作a，然后执行策略π得到的期望总回报，称为状态-动作值函数，也称为Q函数(Q-function)

$$Q^{\pi}(s,a) = E_{s' \sim p(s'|s,a)}[r(s,a,s') + \gamma V^{\pi}(s')]$$

■ **V函数—Q函数关系** $V^{\pi}(s) = E_{a \sim \pi(a|s)}[Q^{\pi}(s,a)]$

V是Q关于动作a的期望

值函数Bellman方程

■ V值函数Bellman方程

$\tau_{0:T}$表示轨迹$s_0, a_0, s_1, \cdots, s_T$； $\tau_{1:T}$表示轨迹$s_1, a_1, \cdots, s_T$

则$\tau_{0:T} = s_0, a_0, \tau_{1:T}$

$$V^{\pi}(s) = E_{\tau_{0:T}\sim p(\tau)}\left[r_1 + \gamma\sum_{t=1}^{T-1}\gamma^{t-1}r_{t+1}\,|\tau_{s_0} = s\right]$$

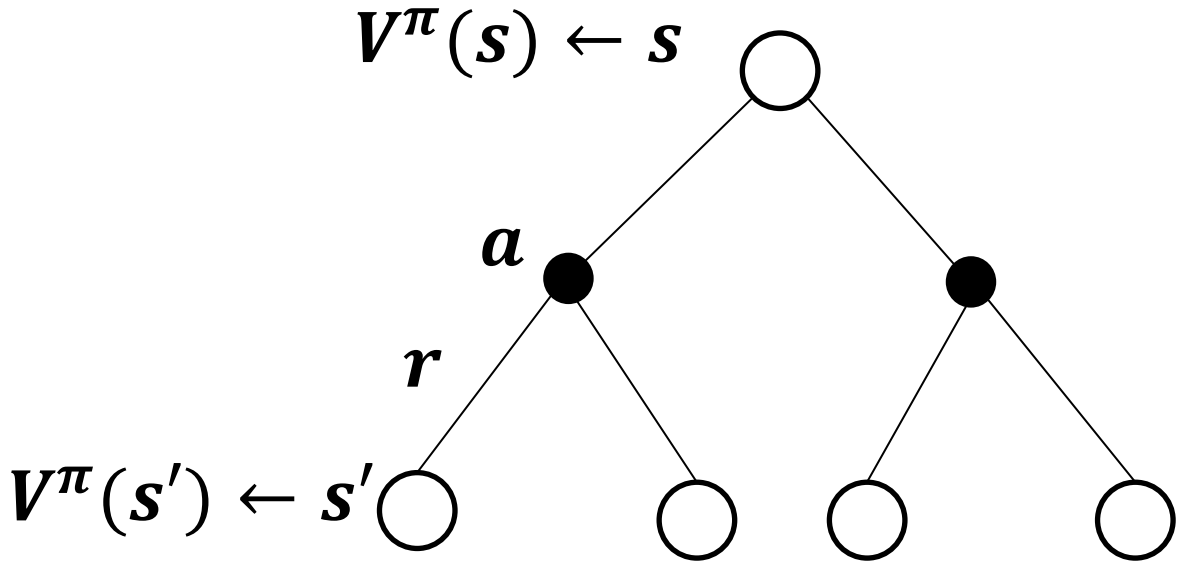
$$= E_{a\sim\pi(a|s)}E_{s\prime\sim p(s'|s,a)}E_{\tau_{1:T}\sim p(\tau)}\left[r(s,a,s') + \gamma\sum_{t=1}^{T-1}\gamma^{t-1}r_{t+1}|\tau_{s_1} = s'\right]$$

$$= E_{a\sim\pi(a|s)}E_{s\prime\sim p(s'|s,a)}\left[r(s,a,s') + \gamma E_{\tau_{1:T}\sim p(\tau)}\sum_{t=1}^{T-1}\gamma^{t-1}r_{t+1}|\tau_{s_1} = s'\right]$$

$$= E_{a\sim\pi(a|s)}E_{s\prime\sim p(s'|s,a)}[r(s,a,s') + \gamma V^{\pi}(s')]$$

$$= \sum_{\boldsymbol{a\in A}}\pi(a|s)\sum_{\boldsymbol{s\prime\in S}}\boldsymbol{P_{ss\prime}^{a}}(\boldsymbol{r_{ss\prime}^{a}} + \boldsymbol{\gamma}V^{\pi}(s'))$$

值函数Bellman方程



$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P^{a}_{ss'}(r^{a}_{ss'} + \gamma V^{\pi}(s'))$$

值函数Bellman方程

模型已知情况下，Bellman 方程可以表达成矩阵形式

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} R(1) \\ \vdots \\ R(n) \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

$$V = R + \gamma PV$$

$$\Longrightarrow (I - \gamma P)V = R$$

线性方程 AX=B

直接求解$\boldsymbol{V = (I - \gamma P)^{-1} R}$

复杂度$\boldsymbol{O(n^3)}$

值函数Bellman方程

■ Q值函数Bellman方程

$$V^{\pi}(s) = E_{a\sim\pi(a|s)}[Q^{\pi}(s,a)]$$

$$+$$

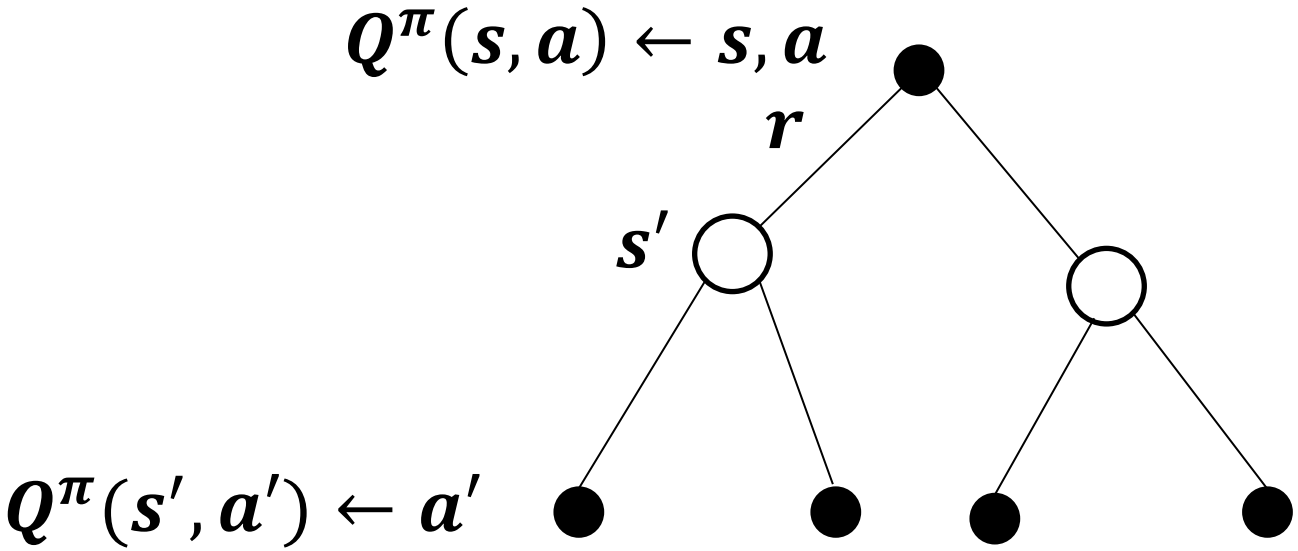
$$V^{\pi}(s) = E_{a\sim\pi(a|s)}E_{s\prime\sim p(s'|s,a)}[r(s,a,s') + \gamma V^{\pi}(s')]$$

$$\Downarrow$$

$$Q^{\pi}(s,a) = E_{s\prime\sim p(s'|s,a)}\left[r(s,a,s') + \gamma E_{a'\sim\pi(a'|s\prime)}[Q^{\pi}(s',a')]\right]$$

$$\mathbf{Q^{\pi}(s,a) = \sum_{s\prime\in S} P^{a}_{ss\prime}\left(r^{a}_{ss\prime} + \sum_{a\prime\in A}\pi(a'|s')Q^{\pi}(s',a')\right)}$$

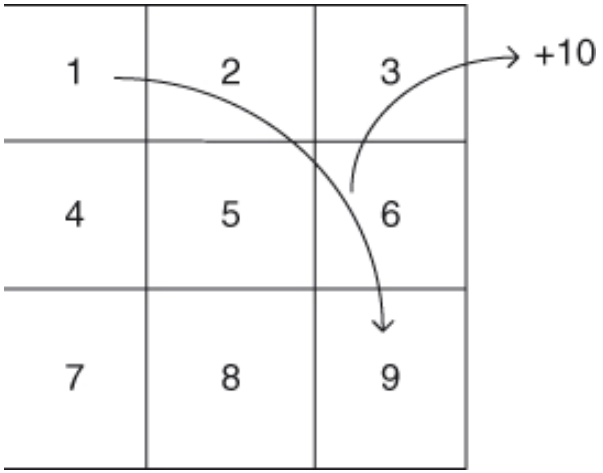
值函数Bellman方程



$$Q^{\pi}(s,a) = \sum_{s' \in S} P_{ss'}^{a} \left(r_{ss'}^{a} + \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s',a') \right)$$

机器人网格问题示例

行动集合$a \in [\mathbf{up}, \mathbf{down}, \mathbf{right}, \mathbf{left}]$



机器人网格问题

$$\gamma = 0.9, \pi(s, a) = 0.25$$

$$r_{19}^{\text{up}} = r_{19}^{\text{down}} = r_{19}^{\text{right}} = r_{19}^{\text{left}} = 10$$

$$r_{22}^{\text{up}} = r_{33}^{\text{up}} = -1$$

$$r_{33}^{\text{right}} = r_{66}^{\text{right}} = r_{99}^{\text{right}} = -1$$

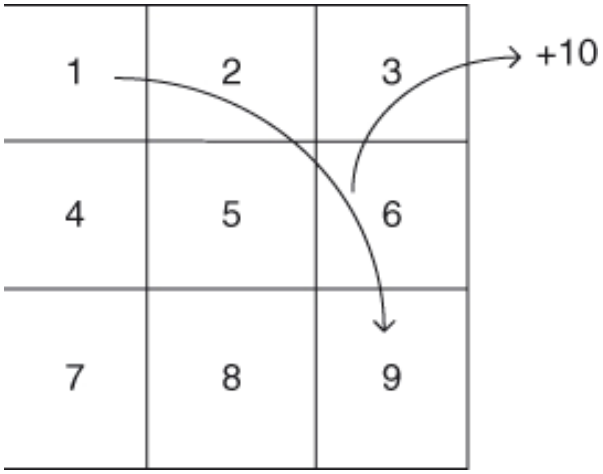
$$r_{44}^{\text{left}} = r_{77}^{\text{left}} = -1$$

$$r_{77}^{\text{down}} = r_{88}^{\text{down}} = r_{99}^{\text{down}} = -1$$

其余回报为0

H. M. Schwartz-Multi-Agent Machine Learning_ A Reinforcement Approach-Wiley

- 转移概率



$$P_{19}^{\text{up}} = P_{19}^{\text{down}} = P_{19}^{\text{right}} = P_{19}^{\text{left}} = 1$$

$$P_{22}^{\text{up}} = P_{25}^{\text{down}} = P_{23}^{\text{right}} = P_{21}^{\text{left}} = 1$$

$$P_{33}^{\text{up}} = P_{36}^{\text{down}} = P_{33}^{\text{right}} = P_{32}^{\text{left}} = 1$$

$$P_{41}^{\text{up}} = P_{47}^{\text{down}} = P_{45}^{\text{right}} = P_{44}^{\text{left}} = 1$$

$$P_{52}^{\text{up}} = P_{58}^{\text{down}} = P_{56}^{\text{right}} = P_{54}^{\text{left}} = 1$$

$$P_{63}^{\text{up}} = P_{69}^{\text{down}} = P_{66}^{\text{right}} = P_{65}^{\text{left}} = 1$$

$$P_{74}^{\text{up}} = P_{77}^{\text{down}} = P_{78}^{\text{right}} = P_{77}^{\text{left}} = 1$$

$$P_{85}^{\text{up}} = P_{88}^{\text{down}} = P_{89}^{\text{right}} = P_{87}^{\text{left}} = 1$$

$$P_{96}^{\text{up}} = P_{99}^{\text{down}} = P_{99}^{\text{right}} = P_{98}^{\text{left}} = 1$$

机器人网格问题示例

$$V^{\pi}(1) = \sum_{a=1}^{4}(0.25)\left(\sum_{9} \mathrm{P}_{19}^{a}(r_{19}^{a} + \gamma V^{\pi}(9))\right)$$

$$\sum_{9} \mathrm{P}_{19}^{a}\left(r_{19}^{a} + \gamma V^{\pi}(9)\right) = r_{19}^{a} + 0.9V^{\pi}(9)$$

$$\Downarrow$$

$$V^{\pi}(1) = 10 + 0.9V^{\pi}(9)$$

机器人网格问题示例

$$V^{\pi}(2)=\sum_{a=1}^{4}(0.25)\left(\sum_{s'}\mathrm{P}_{2s'}^{a}\left(r_{2s'}^{a}+\gamma V^{\pi}(s')\right)\right)$$

$$=(0.25)\left(\sum_{s'}\mathrm{P}_{2s'}^{\mathrm{up}}\left(r_{2s'}^{\mathrm{up}}+\gamma V^{\pi}(s')\right)\right)+(0.25)\left(\sum_{s'}\mathrm{P}_{2s'}^{\mathrm{down}}\left(r_{2s'}^{\mathrm{down}}+\gamma V^{\pi}(s')\right)\right)$$

$$+(0.25)\left(\sum_{s'}\mathrm{P}_{2s'}^{\mathrm{left}}\left(r_{2s'}^{\mathrm{left}}+\gamma V^{\pi}(s')\right)\right)+(0.25)\left(\sum_{s'}\mathrm{P}_{2s'}^{\mathrm{right}}\left(r_{2s'}^{\mathrm{right}}+\gamma V^{\pi}(s')\right)\right)$$

$$=0.25(-1+0.9V^{\pi}(2))+0.25(0+0.9V^{\pi}(5))+0.25(0+0.9V^{\pi}(1))$$
$$+0.25(0+0.9V^{\pi}(3))$$

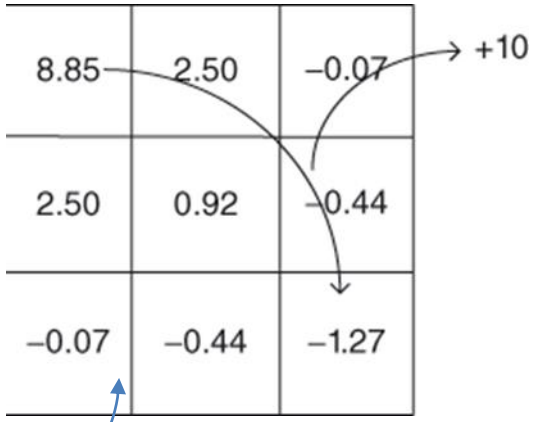
$$=0.225V^{\pi}(1)+0.225V^{\pi}(3)+0.225V^{\pi}(3)+0.225V^{\pi}(5)-0.25$$

$$\mathbf{A}\boldsymbol{V}^{\boldsymbol{\pi}}=\boldsymbol{B}$$

机器人网格问题示例

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.9 \\ -0.225 & 0.775 & -0.225 & 0 & -0.225 & 0 & 0 & 0 & 0 \\ 0 & -0.225 & 0.55 & 0 & 0 & -0.225 & 0 & 0 & 0 \\ -0.225 & 0 & 0 & 0.775 & -0.225 & 0 & -0.225 & 0 & 0 \\ 0 & -0.225 & 0 & -0.225 & 1 & -0.225 & 0 & -0.225 & 0 \\ 0 & 0 & -0.225 & 0 & -0.225 & 0.775 & 0 & 0 & -0.225 \\ 0 & 0 & 0 & -0.225 & 0 & 0 & 0.55 & -0.225 & 0 \\ 0 & 0 & 0 & 0 & -0.225 & 0 & -0.225 & 0.775 & -0.225 \\ 0 & 0 & 0 & 0 & 0 & -0.225 & 0 & -0.225 & 0.55 \end{bmatrix}$$

$$B^T = [10 \quad -0.25 \quad -0.5 \quad -0.25 \quad 0 \quad -0.25 \quad -0.5 \quad -0.25 \quad -0.5]$$



$$\Downarrow$$

$$\boldsymbol{V^{\pi} = [8.85 \quad 2.5 \quad -0.07 \quad 2.5 \quad 0.92 \quad -0.44 \quad -0.07 \quad -0.44 \quad -1.27]^{T}}$$

最优值函数

最优值函数：评价任意策略下最好的动作或状态—动作对

- 最优状态值函数$V^*(s)$： 在所有策略中值最大的值函数

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- 最优状态-行动值函数$Q^*(s,a)$：在所有策略中最大的状态-行动值函数

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

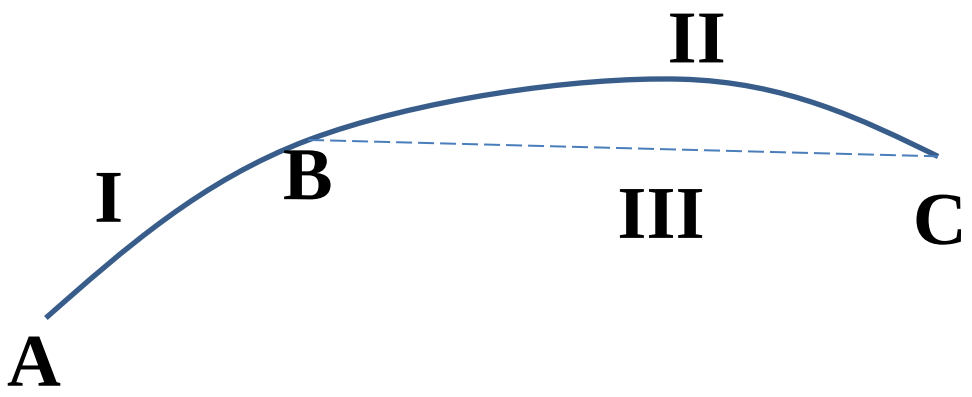
Bellman最优性原理

■ 最优性原理：

多阶段决策过程的最优决策序列具有这样的性质：不论初始状态和初始决策如何，对于前面决策所造成的某一状态而言，其后各阶段的决策序列必须构成最优策略。

含义：

最优策略的任何一部分子策略也必须使最优的



值函数最优Bellman方程

■ V值函数最优Bellman方程

$$V^*(s) = \max_{a \in A} \sum_{s'} P_{ss'}^{a} \left(r(s, a, s') + \gamma \sum_{s' \in S} V^*(s') \right)$$

■ Q值函数最优Bellman方程

$$Q^*(s, a) = \sum_{s' \in S} P_{ss'}^{a} \left(r(s, a, s') + \max_{a'} Q^*(s', a') \right)$$

最优值函数

	状态值函数	状态动作值函数
预测	V^{π}	Q^{π}
控制	V^{*}	Q^{*}

最优策略：

$$\pi^{*}(s) = \arg\max_{a \in A} \sum_{s'} P_{ss'}^{a} \left(r(s, a, s') + \gamma \sum_{s' \in S} V^{*}(s') \right)$$

$$\pi^{*}(s) = \arg\max_{a \in A} Q^{*}(s, a)$$

最优值函数

- 定义策略的偏序（partial ordering）如下

$$\pi \geq \pi', \text{if } V^{\pi}(s) \geq V^{\pi'}(s), \forall s$$

定理：对任意Markov决策过程：

- 存在一个最优策略π^*好于或至少不差于任意其他策略
  即$\pi^* \geq \pi, \forall \pi$
- 所有的最优策略有最优的状态值函数，$V^{\pi^*}(s) = V^*(s)$
- 所有的最优策略有最优的状态—动作值函数
  $Q^{\pi^*}(s,a) = Q^*(s,a)$

强化学习途径

■ 马尔科夫决策过程(MDP):

- 动态规划（DP），需要行为模型
- 强化学习（RL），无行为模型

■ 根据寻找最优策略的途径，DP和RL算法都可分为三种

- 值迭代（Value iteration）：寻找最优值函数，最优值函数包括每个状态或者每个状态动作对的最大回报。最优值函数用来计算最优策略
- 策略迭代（Policy iteration）：通过构建值函数（而不是最优值函数）来评估策略，然后利用这些值函数，寻找新的、改进的策略
- 策略搜索（Policy search）：使用优化技术，直接寻找最优策略

基于值函数的强化学习方法

- 动态规划
  - 策略迭代
  - 值迭代
- 蒙特卡罗方法
- 时序差分学习
- Q学习
- SARSA学习
- 值函数逼近强化学习

基于模型的动态规划方法

■ 动态规划:(1956,Bellman)

通过把复杂问题划分为子问题，并对子问题进行求解，最后把子问题的解结合起来解决原问题

- 动态：问题由一系列的状态组成，而且状态能一步步地改变
- 规划：即优化每一个子问题

因为MDP 的 Markov 特性，即某一时刻的子问题仅仅取决于上一时刻的子问题的 action，并且 Bellman 方程可以递归地切分子问题，所以可以采用动态规划来求解 Bellman 方程

动态规划算法

动态规划方法：

模型已知时，可以通过贝尔曼方程迭代来计算值函数

- 策略迭代
- 值迭代

■ 策略迭代(policy iteration):

1、策略评估（policy evaluation）：计算当前策略下，每个状态的值函数

策略评估可以通过贝尔曼方程进行迭代计算$V^{\pi}(s)$，如果状态有限时，也可以通过直接求解Bellman方程来得到$V^{\pi}(s)$

2、策略改进(policy improvement)：根据值函数来更新策略（可ε贪心算法找当前最优策略）

动态规划算法—策略迭代

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \cdots \rightarrow \pi^* \rightarrow V^{\pi^*} \rightarrow \pi^*$$

policy evaluation

policy improvement
"greedification"

$$V_{k+1}(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P^a_{ss'}(r^a_{ss'} + \gamma V_k(s'))$$

$$\pi'(s) = \arg\max_{a \in A} \sum_{s'} P^a_{ss'}(r^a_{ss'} + \gamma \sum_{s' \in S} V^{\pi}(s'))$$

Policy Iteration

动态规划算法—策略迭代

- 输入：MDP五元组：S, A, P, r, γ
- 1 初始化:$\forall s, \forall a, \pi(a|s) = \frac{1}{|A|}$
- 2 **repeat**
- //策略评估
- 3 **repeat**
- 4 根据贝尔曼方程，计算$V^{\pi}(s), \forall s$
- 5 **until** $\forall s, V^{\pi}(s)$收敛；
- //策略改进
- 6 根据公式，计算$Q(s, a)$；
- 7 $\forall s, \pi(s) = \arg\max_a Q(s, a)$；
- 8 **until** $\forall s, \pi(s)$收敛；
- 输出：策略π

$$V_{k+1}(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^{a}(r_{ss'}^{a} + \gamma V_k(s'))$$

$$Q^{\pi}(s, a) = E_{s' \sim p(s'|s,a)}[r(s, a, s') + \gamma V^{\pi}(s')]$$

动态规划算法—策略迭代

- 策略评估的收敛性：压缩映射原理

定义：设X是度量空间，其度量用ρ表示。映射$T: X \to X$,若存在$\gamma, 0 \leq \gamma \leq 1$,使得

$$\rho(Tx, Ty) \leq \gamma \rho(x, y), \forall x, y \in A$$

则称T是X上的一个**压缩映射**；

若存在$x_0 \in X$,使得$Tx_0 = x_0$,则称x_0是T的不动点

压缩映射定理：完备度量空间上的压缩映射具有唯一不动点。

动态规划算法—策略迭代

- 将值函数看作未知数，值函数求解其实是求解一组线性方程

采用高斯-赛德尔（Guass-Seidel）迭代

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^{a}(r_{ss'}^{a} + \gamma V_{k}(s'))$$

收敛条件：由压缩映射定理，未知数矩阵的谱半径小于1

动态规划算法—策略迭代

- 定义算子T

$$T(U) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^{a} (r_{ss'}^{a} + \gamma U(s')) = U(s)$$

$$T(V) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^{a} \left(r_{ss'}^{a} + \gamma V(s') \right) = V(s)$$

无穷范数： $\|v\|_\infty = \max_{s \in S} \|v(s)\|$

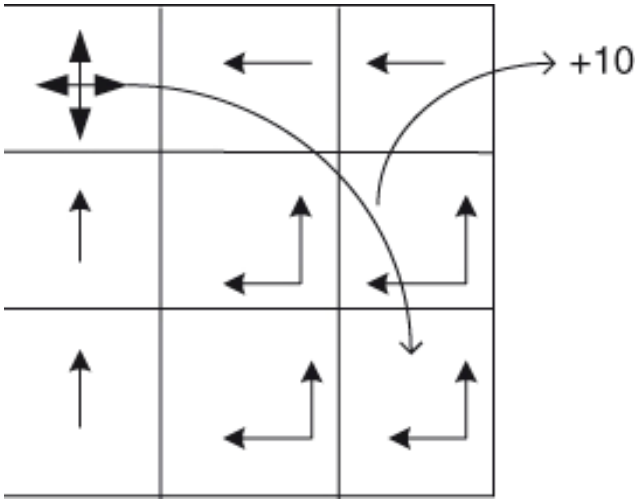
$$\|T(U) - T(V)\|_\infty = \max_{s,a} \left\| \gamma \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^{a} \gamma U(s') - \gamma \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^{a} \gamma V(s') \right\|$$

$$\leq \max_{s,a} \left\| \gamma \left(\sum_{s' \in S} P_{ss'}^{a} \gamma U(s') - \sum_{s' \in S} P_{ss'}^{a} \gamma V(s') \right) \right\|$$

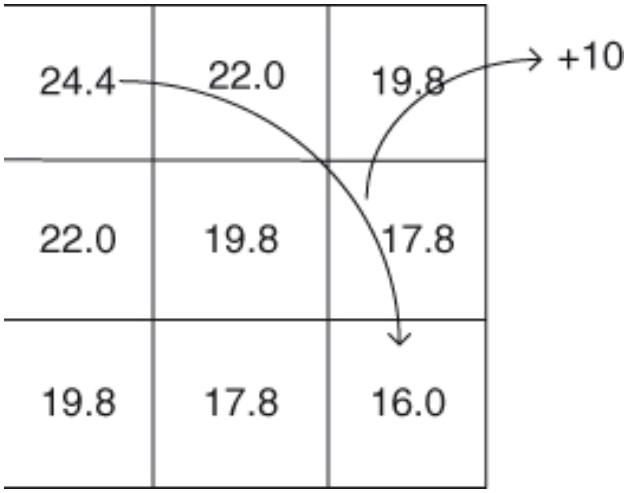
$$\leq \max_{s} \gamma \left\| \max_{s' \in S} \|U(s') - V(s')\| \right\| \leq \max_{s} \gamma \|U(s') - V(s')\|$$

$$\|T(U) - T(V)\|_\infty \leq \gamma \|U(s') - V(s')\|_\infty$$

机器人网格问题



所得最优策略



基于最优策略所得到的状态值

动态规划算法—值迭代

■ 值迭代（Value iteration）

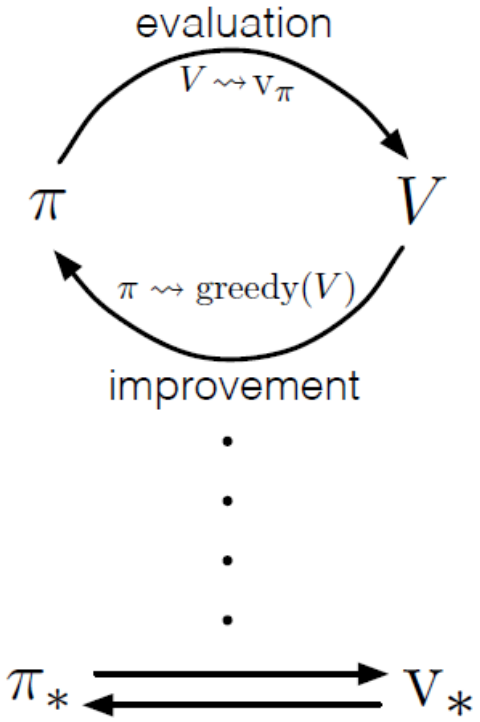
- 将策略评估和策略改进两个过程合并，直接计算最优策略
  - 值迭代算法直接使用贝尔曼最优方程来更新值函数，收敛时的值函数就是最优的值函数，其对应的策略也就是最优的策略
  - 在每一次迭代中都能找到让当前值函数最大的更新方式，并用这种方式来更新值函数，不断迭代，直到值函数不再发生变化

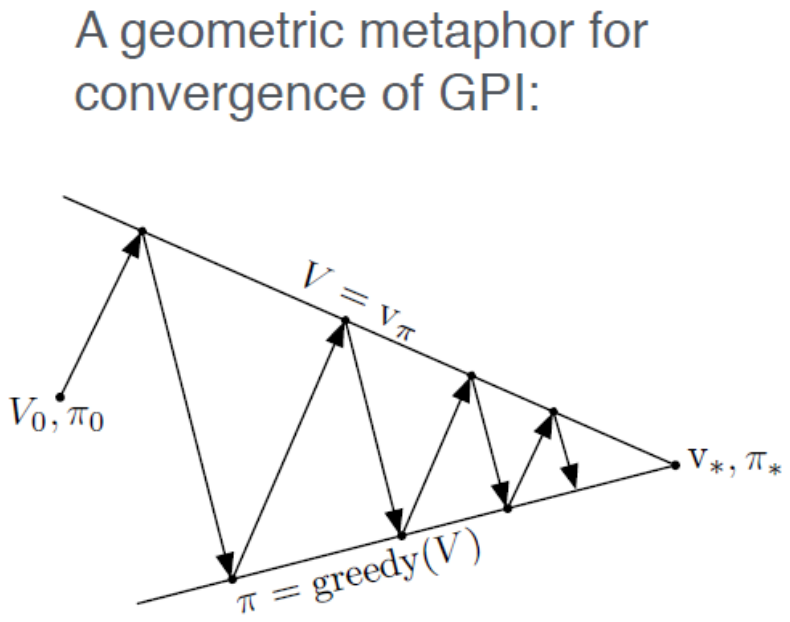
动态规划算法—值迭代

- 输入：MDP五元组：S, A, P, r, γ
- 1 初始化：$\forall s \in S, V(s) = 0$
- 2 **repeat**
- 3 $\quad \forall s, V(s) \leftarrow \max_a \sum_{s' \in S} \boldsymbol{P}_{ss'}^{\boldsymbol{a}} (\boldsymbol{r}_{ss'}^{\boldsymbol{a}} + \boldsymbol{\gamma} \boldsymbol{V}(\boldsymbol{s}'))$
- 4 **until** $\forall s, V(s)$收敛
- 5 根据公式计算$Q(s, a)$;
- 6 $\forall s, \pi(s) = \arg\max_a Q(s, a)$;
- 输出：策略π

动态规划算法—广义策略迭代

- **广义策略迭代**(Generalized Policy Iteration ,GPI):





广义策略迭代示意图

蒙特卡罗方法 Monte Carlo Methods

问题：无模型时如何求状态s值函数？

思路：多次试验，当访问到状态S后，观测回报，并取平均。

即利用经验和评价代替随机变量的期望

$$Q^{\pi}(s,a) = E_{\tau \sim p(\tau)}[G(\tau)|\tau_{s_0} = s, \tau_{a_0} = a]$$

无模型强化学习（model-free reinforcement learning）

—蒙特卡罗方法

能够得到正确的值函数取决于经验，如何获得充足的经验是无模型强化学习的核心所在

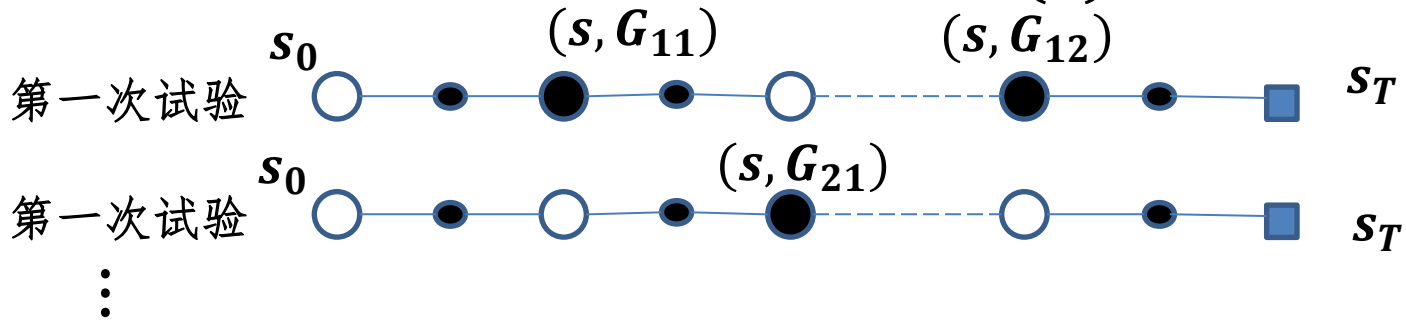
蒙特卡罗方法

- 首次访问的蒙特卡罗方法（First-visit MC:）：在计算s处的值函数时，只利用每次实验中第一次访问到状态s的回报值

$$V(s) = \frac{G_{11}(s) + G_{21}(s) + \cdots}{N(s)}$$

- 每次访问蒙特卡罗方法（Every-Visit MC）：利用所有访问到状态s时的回报值

$$V(s) = \frac{G_{11}(s) + G_{12}(s) + \cdots + G_{21}(s)}{N(s)}$$



蒙特卡罗方法

■ 蒙特卡罗策略评估：

假定共进行N次试验，得到N个轨迹$\tau^{(1)}, \tau^{(2)}, \cdots, \tau^{(N)}$

总回报分别为$G\left(\tau^{(1)}\right), G\left(\tau^{(2)}\right), \cdots, G\left(\tau^{(N)}\right)$

$$Q^{\pi}(s,a) \approx \hat{Q}^{\pi}(s,a) = \frac{1}{N}\sum_{n=1}^{N} G(\tau^{(n)})$$

理论上而言，需要无限次的策略评估才能保证绝对收敛

但很多实际例子经常在有限次迭代后就收敛

蒙特卡罗方法

■ 策略改进

对任一确定的策略π，其值函数为V^{π}，则对状态s，若采用策略π，其值函数为$V^{\pi}(s)$，若改变策略在s选择动作a，然后继续采用策略π，则有

$$Q^{\pi}(s,a)=\sum_{s'\in S}P_{ss'}^{a}\left(r_{ss'}^{a}+\sum_{a'\in A}\pi(a'|s')Q^{\pi}(s',a')\right)=\sum_{s'\in S}P_{ss'}^{a}\left(r_{ss'}^{a}+V^{\pi}(s')\right)$$

若$Q^{\pi}(s,a)>V^{\pi}(s)$，则选择动作a，然后继续采用策略π，会比直接选择π好。

一般地，如果π及π'是任意的两个确定的策略，对任意$s'\in S$，$Q^{\pi}(s,a)\geq V^{\pi'}(s)$，则称$\pi$相比于$\pi'$好或一样好

蒙特卡罗方法

■ 蒙特卡罗策略改进

- 利用经验平均估计策略值函数，对每个状态S,通过最大化动作值函数来进行策略改进，即

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

蒙特卡罗方法收敛性

$$\begin{aligned} \boldsymbol{Q^{\pi_k}\left(s,\pi_{k+1}(s)\right)} &= \boldsymbol{Q^{\pi_k}\left(s,\arg\max_a Q(s,a)\right)} \\ &= \max_{\boldsymbol{a}} \boldsymbol{Q^{\pi_k}(s,a)} \\ &\geq \boldsymbol{Q^{\pi_k}\left(s,\pi_k(s)\right)} \\ &\geq \boldsymbol{V^{\pi_k}} \end{aligned}$$

蒙特卡罗方法

1 初始化$s \in S, a \in A(s), Q(s,a) \leftarrow$ arbitrary, $\pi(s) \leftarrow$ arbitrary, Returns(s, a) $\leftarrow$ emptylist

Repeat

2 随机选择$s_0 \in s, a_0 \in A(s)$,从s_0, a_0开始以策略π生成一个实验(episode),对在这个实验中出现的状态和动作s, a:

3 G $\leftarrow$ s, a第一次出现后的回报

4 将G附加于回报Returns(s, a)上

5 Q(s, a) $\leftarrow$ average(Returns(s, a))

策略评估

6 对每一个s: $\pi(s) \leftarrow \arg\max_a Q(s,a)$ 策略改进

蒙特卡罗方法

■ 递增计算均值的方式

$$\hat{Q}_k^{\pi}(s,a) = \frac{1}{k}\sum_{n=1}^{k} G(\tau^{(n)}) = \frac{1}{k}\left(G(\tau^{(k)}) + \sum_{n=1}^{k-1} G(\tau^{(n)})\right)$$

$$= \frac{1}{k}\left(G(\tau^{(k)}) + (k-1)\hat{Q}_{k-1}^{\pi}(s,a)\right)$$

$$= \hat{Q}_{k-1}^{\pi}(s,a) + \frac{\mathbf{1}}{\boldsymbol{k}}(G(\tau^{(k)}) - \hat{Q}_{k-1}^{\pi}(s,a))$$

蒙特卡罗误差

■ 更新公式

$$\boldsymbol{Q(s,a) \leftarrow Q(s,a) + \alpha(G_t - Q(s,a))}$$

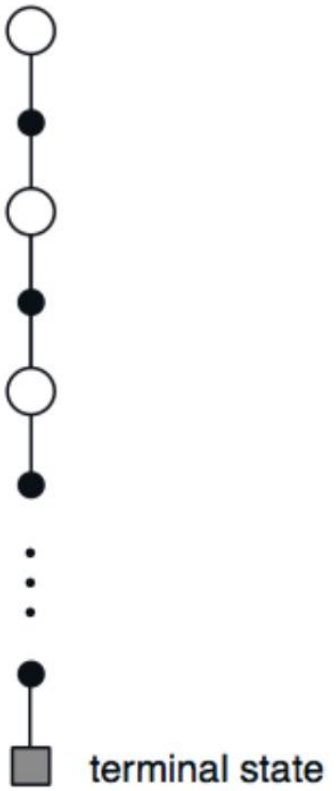
蒙特卡罗方法

- 采用确定性策略π,每次得到的试验轨迹一样，只能得到$Q^{\pi}(s,\pi(s))$,无法计算其他动作a'的Q函数

- 探索和利用平衡(exploration and exploitation)
  - ✓ $\epsilon-$贪心算法 $\epsilon-$greedy method

$$\pi^{\epsilon}(s)=\begin{cases}\pi(s) & \text{按概率}1-\epsilon\\ \text{随机选择}A\text{中的动作}, & \text{按概率}\epsilon\end{cases}$$

  - ✓ 随机初始化起点（探索起点条件exploring starts）



蒙特卡罗方法

- **同策略（on policy）方法**：如果采样（行动）策略是$\pi^{\epsilon}(s)$，不断改进策略也是$\pi^{\epsilon}(s)$而不是目标策略$\pi(s)$。这种采样策略与目标策略相同（即都是$\pi^{\epsilon}(s)$）的学习方法叫做同策略方法

  直接了当，速度快，但不一定找到最优策略

- **异策略（off policy）方法**：如果采样策略是$\pi^{\epsilon}(s)$，而优化目标是策略$\pi(s)$，采样与改进分别使用不同策略的强化学习方法叫做异策略方法

  收敛慢，更为强大和通用，确保了数据的全面性，
  所有行为都能覆盖，可找到最优解

采样策略（行动策略）：产生样本用于评估的策略

目标策略：需要学习和改进的策略

蒙特卡罗方法

异策略的目标策略π和采样策略μ需满足

- 覆盖性条件：即采样策略μ产生的行为覆盖或包含目标策略π产生的行为，满足$\pi(a|s) > 0$的任何(s, a)均满足$\mu(a|s) > 0$

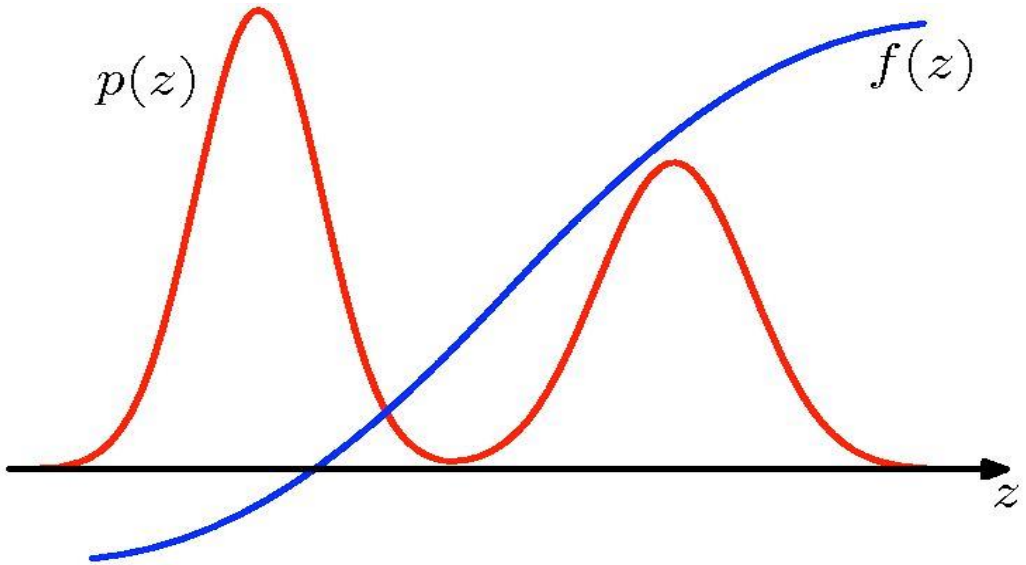
- 利用采样策略产生的数据评估目标策略需要利用重要性采样

蒙特卡罗方法

- 重要性采样来源于求期望

$$E[f] = \int f(z)p(z)dz \approx \frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}$$

$$\mathrm{var}[\hat{f}] = \frac{1}{N}E[(f - E(f))^2]$$

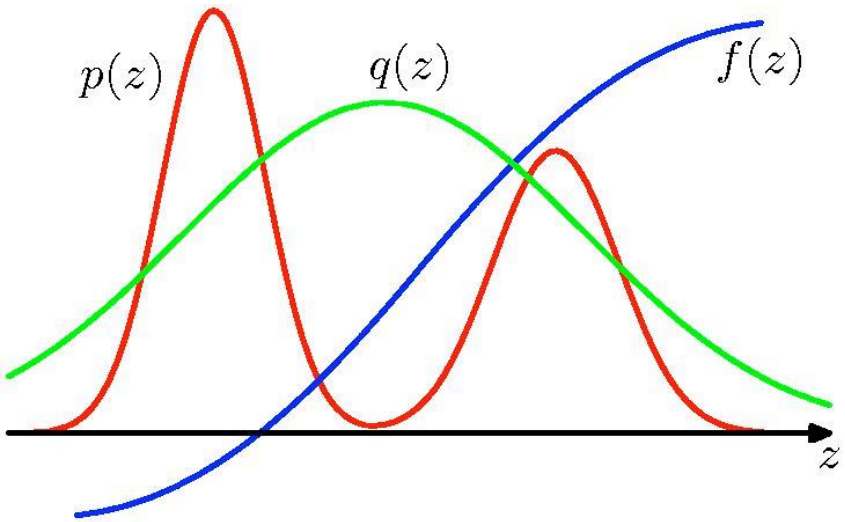


蒙特卡罗方法

- 可选用一个概率分布很简单，很容易产生样本的概率分布$q(z)$，比如正态分布

$$E[f] = \int f(z)p(z)dz = \int f(z)\frac{p(z)}{q(z)}q(z)dz$$

$$\approx \frac{1}{N}\sum_{n}\frac{p(z^n)}{q(z^n)}f(z^n), z^n \sim q(z)$$



蒙特卡罗方法

- 定义重要性权重

$$w^n = \frac{p(z^n)}{q(z^n)}, \qquad E[f] = \frac{1}{N}\sum_n w^n f(z^n)$$

- 基于重要性采样的积分估计为无偏估计，但积分估计的方差为无穷大（被积函数乘了一个重要性权重，改变了被积函数的形状及分布）
- 减小方差的方法：加权重要性采样

$$E[f] = \frac{1}{N}\sum_n \frac{w^n}{\sum_{m=1}^{N} w^m} f(z^n)$$

蒙特卡罗方法

- 异策略中，行动策略μ（产生样本）所生成的轨迹概率分布相当于重要性采样中的$q(z)$，用来评估和改进的策略π所对应的概率分布轨迹为$p(z)$

- 在目标策略π下，一次实验的概率

$$\Pr(a_t, s_{t+1}, \cdots, s_T) = \prod_{k=t}^{T-1} \pi(a_k|s_k)\, p(s_{k+1}|s_k, a_k)$$

- 在行动策略μ下，一次实验的概率

$$\Pr(a_t, s_{t+1}, \cdots, s_T) = \prod_{k=t}^{T-1} \mu(a_k|s_k)\, p(s_{k+1}|s_k, a_k)$$

- 重要性权重

$$\rho_t^T = \frac{\Pr(a_t, s_{t+1}, \cdots, s_T)}{\Pr(a_t, s_{t+1}, \cdots, s_T)} = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{\mu(a_k|s_k)}$$

蒙特卡罗方法

- 重要性采样的值函数估计

$$V(s) = \frac{\sum_{t \in T(s)} \rho_t^{T(t)} G_t}{|T(s)|}$$

t是状态s访问的时刻

$T(t)$是访问状态s相对应的实验终止状态所对应的时刻

$T(s)$是状态s发生的所有时刻的集合

时序差分学习算法

- 时序差分学习（temporal-difference learning，TD）
  1988年，Sutton等提出，解决时间信度分配

目标：策略π的试验轨迹学习值函数V^{π}

MC方法：$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$

向实际回报G_t方向更新$V(s_t)$

TD(0)：一步TD算法

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

$r_{t+1} + \gamma V(s_{t+1})$：TD目标

$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$：TD 误差

时序差分学习算法

- 1 初始化$V(s) = 0, \forall s \in S$
- 2 初始化s为任意值
- 3 repeat
- 4 对每个步长时刻，根据策略$\pi(s)$选择行为a
- 5 得到回报r和下一状态s'
- 6 $V^{\pi}(s) \leftarrow V^{\pi}(s) + \boldsymbol{\alpha}[r(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s)]$
- 7 $s = s'$
- 8 until 完成所需步长个数或s达到终止状态

时序差分学习算法

- DP vs. MC vs. TD

$$V^{\pi} = E_{\pi}\left[G_t | s_t = s\right]$$

$$= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]$$

$$= E_{\pi}\left[r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right]$$

$$= E_{\pi}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right]$$

MC：利用采样求平均回报来近似期望

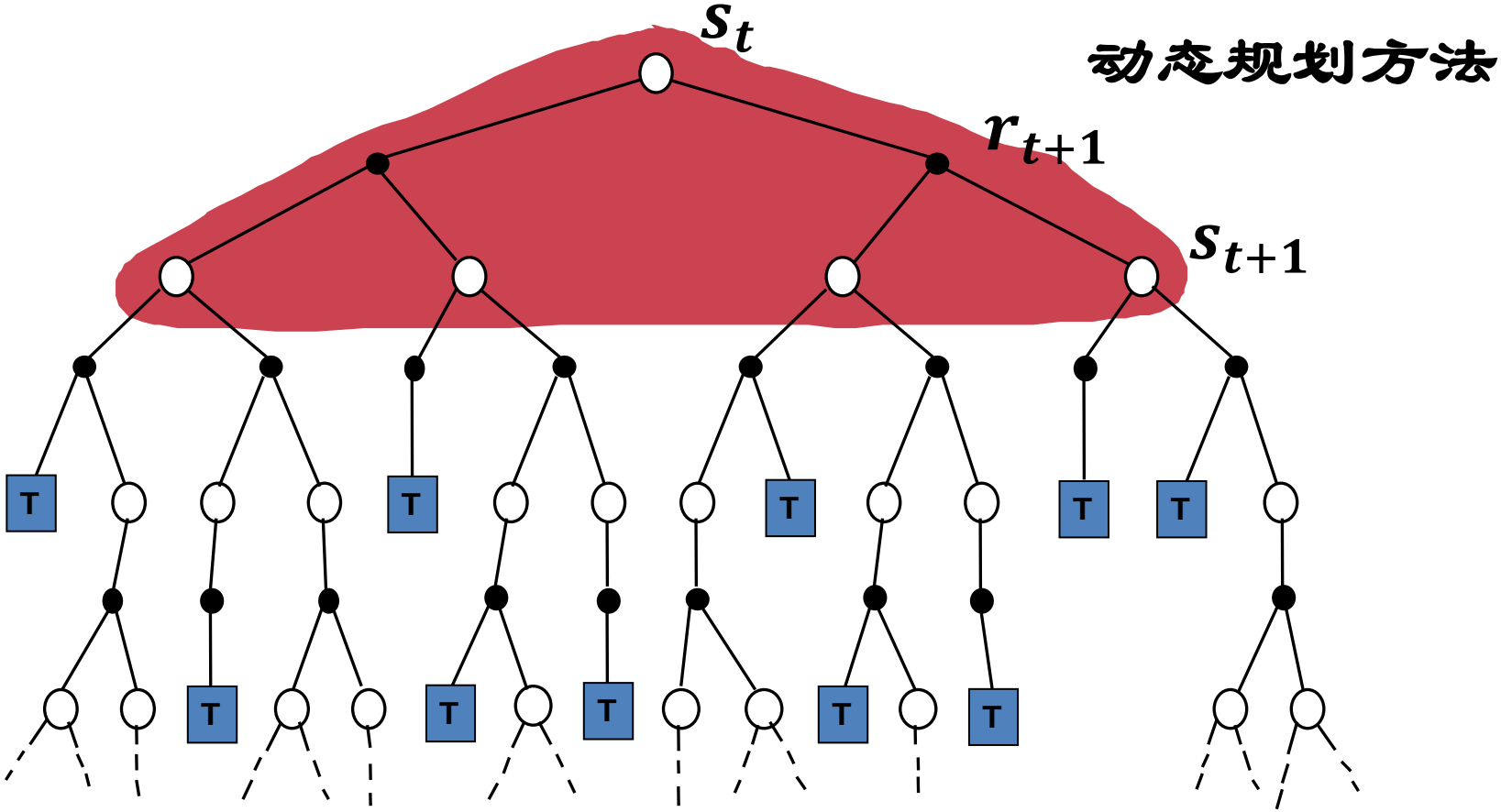
Bootstrapping

TD：两者结合，采样求期望，并利用真值$V^{\pi}(s_{t+1})$的当前估计值$V(s_{t+1})$

DP:期望值通过模型提供，但利用真值$V^{\pi}(s_{t+1})$的当前估计值$V(s_{t+1})$

DP vs. MC vs. TD

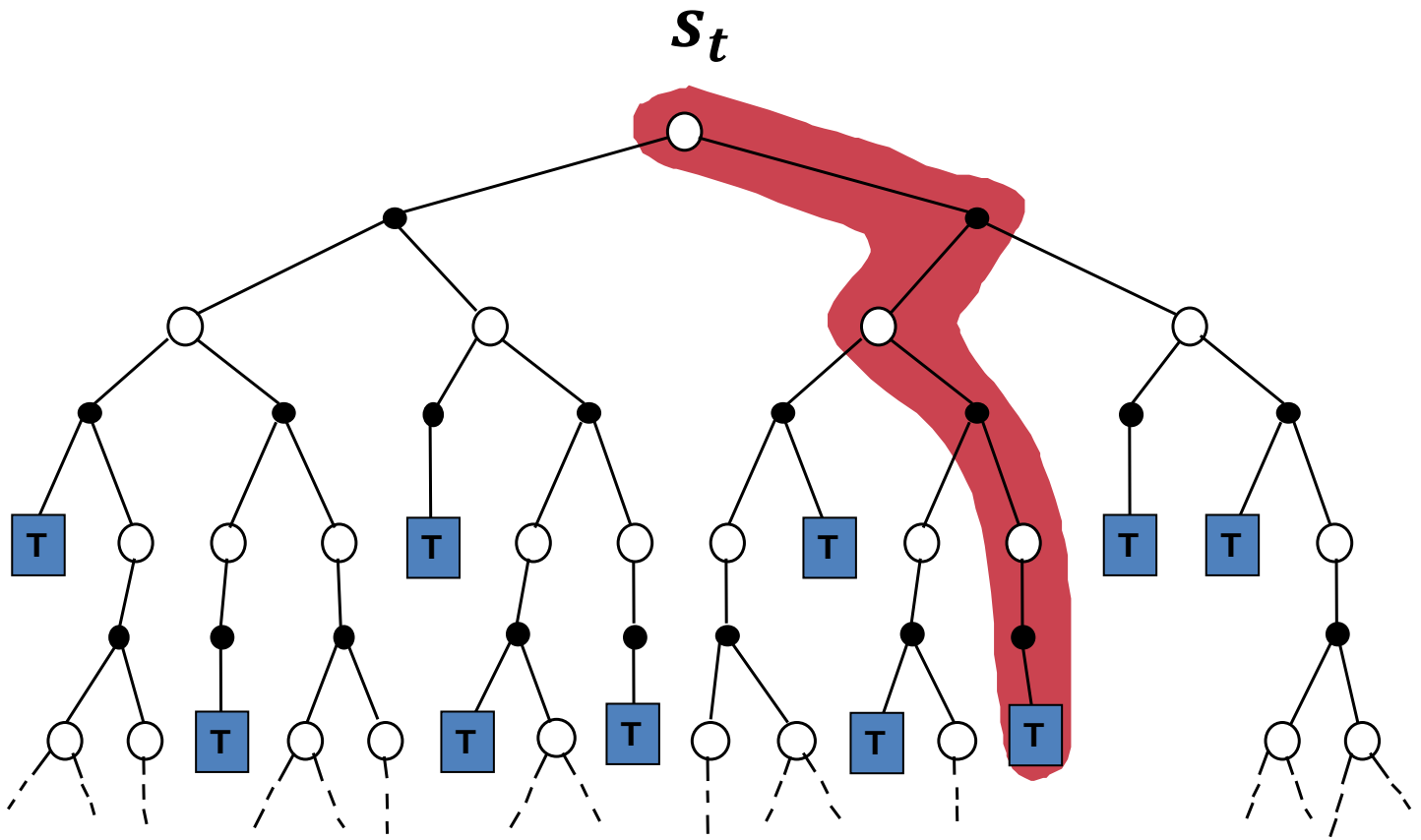
$$V(s_t) \leftarrow E_\pi[r_{t+1} + \gamma V(s_{t+1})] = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P^a_{ss'}(r^a_{ss'} + \gamma V^\pi(s'))$$



DP vs. MC vs. TD

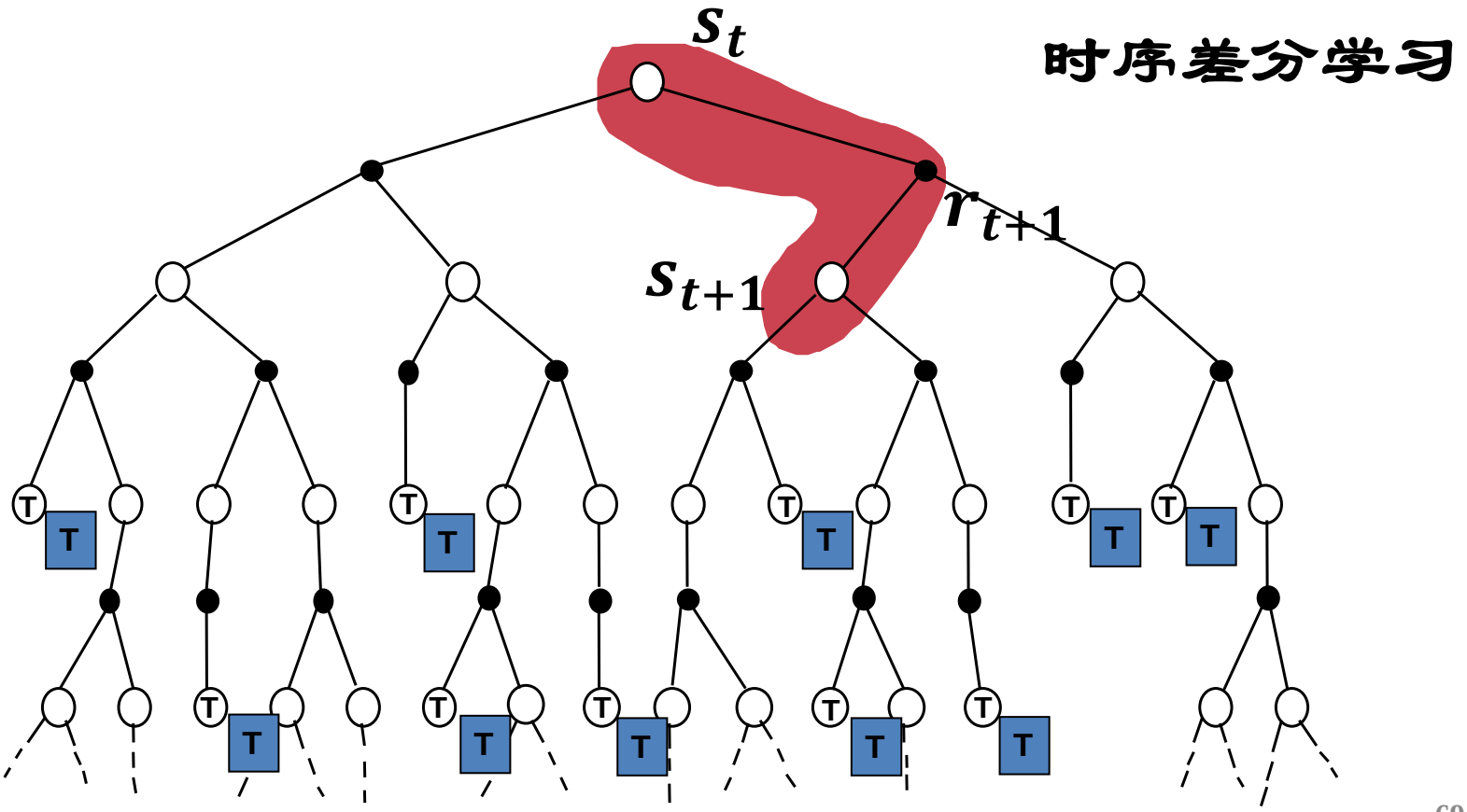
$$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$$

蒙特卡罗方法



DP vs. MC vs. TD

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



时序差分学习方法

■ 收敛性分析

Delta Rule

- 假定我们有一个理想状态，我们需要逐步调整当前状态，令它慢慢趋近这个理想，方法：

  当前状态：=当前状态+α(理想−当前状态)

- α 学习速率， Delta = 理想−当前状态

时序差分学习算法

■ 收敛性分析

TD(0)学习算法

$$V^{\pi}(s) = E_{a\sim\pi(a|s)}E_{s'\sim p(s'|s,a)}[r(s,a,s') + \gamma V^{\pi}(s')]$$
$$= E[r(s,a,s') + \gamma V^{\pi}(s')]$$

$$\boldsymbol{\delta} = r(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s)$$

取学习因子为α，TD(0)的迭代计算公式为

$$V^{\pi}(s) = V^{\pi}(s) + \boldsymbol{\alpha}[r(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s)]$$

Sutton: Learn a guess form a guess

时序差分学习算法

■ TD算法收敛性基础：随机逼近定理

If X_n are updated according to

$$X_{n+1} = X_n + \beta_n(\xi_n - X_n)$$

Where

$$0 \le \beta_n \le 1, \sum_{i=1}^{\infty} \beta_n = \infty, \sum_{i=1}^{n} \beta_n^2 < \infty,$$

and ξ_n are bounded random variables with mean Ξ, then

$$X_n \to \Xi,$$

as $n \to \infty$ with probability 1.

Kusher, H & Clark, D, Stochastic approximation methods for constrained and unconstrained systems. Berlin, Germany: Springer-Verlag.

时序差分学习TD(λ)

TD(0)在迭代过程中仅利用了当前时刻的时序差值（一步截断回报），为单步时间差值算法

提高效率手段：在时间差分学习中利用Markov链在多个时刻的时序差分信息

对$0 \leq \lambda \leq 1$,TD学习算法通过采用n步截断回报和λ回报，实现对多个时刻时间差分信息的利用，为多步时序差分学习算法

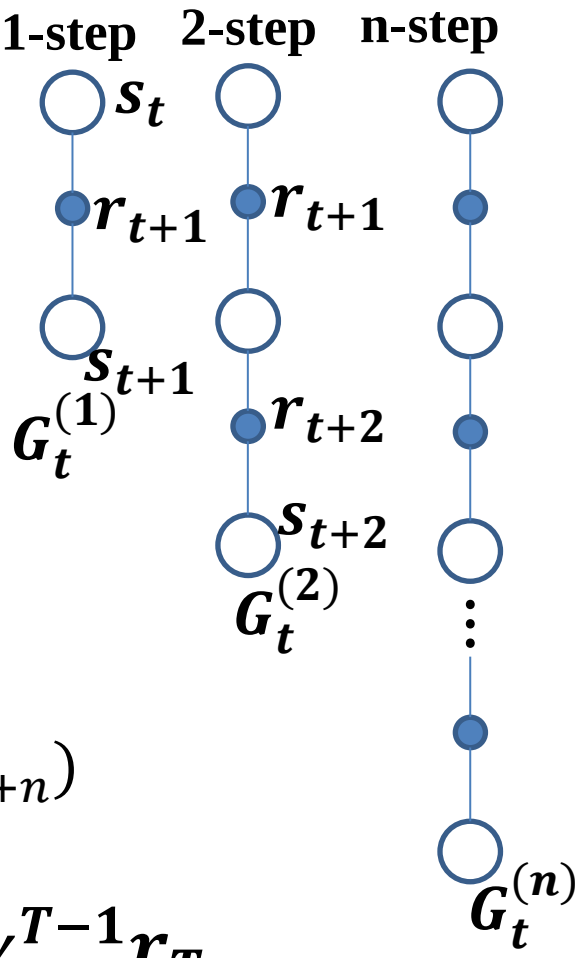
时序差分学习TD(λ)

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \boldsymbol{\alpha}[r(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s)]$$

另记为

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \boldsymbol{\alpha}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)]$$

$$\boldsymbol{G_t^{(1)}} = r_{t+1} + \gamma V^{\pi}(s_{t+1})$$



利用第2步值函数估计当前值函数

$$\boldsymbol{G_t^{(2)}} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V^{\pi}(s_{t+2})$$

利用第n步值函数估计当前值函数

$$\boldsymbol{G_t^{(n)}} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^{\boldsymbol{n}} V^{\pi}(s_{t+n})$$

$$\boldsymbol{G_t^{(\infty)}} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^{\boldsymbol{T-1}} \boldsymbol{r_T}$$

时序差分学习TD(λ)

n-step 时序差分

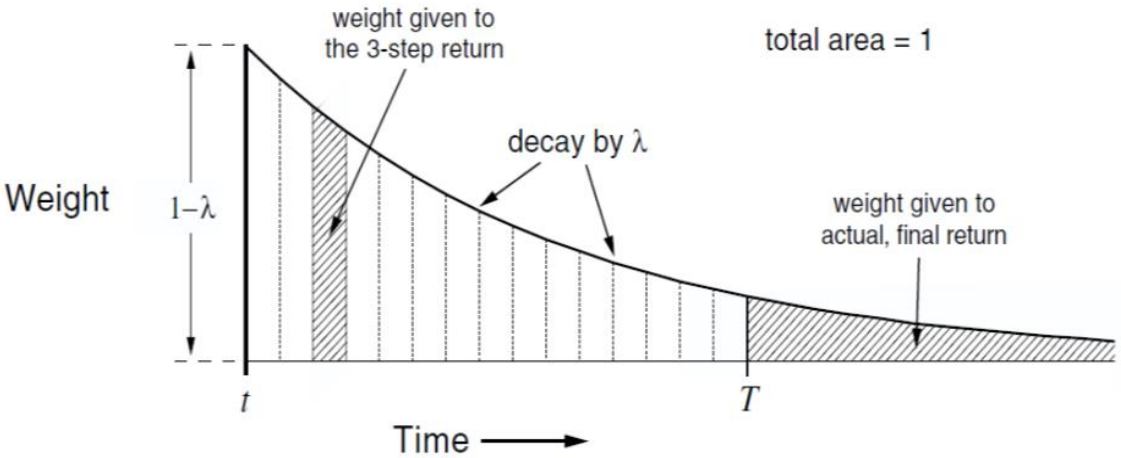
$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha[G_t^{(n)} - V^{\pi}(s_t)]$$

TD(λ)方法：加权n-step时序差分

$$G_t^{\lambda} = (1-\lambda)G_t^{(1)} + (1-\lambda)\lambda G_t^{(1)} + \cdots + (1-\lambda)\lambda^{n-1}G_t^{(n)}$$

$$\approx [(1-\lambda) + (1-\lambda)\lambda + \cdots + (1-\lambda)\lambda^{n-1}]V(s_t)$$

$$= V(s_t)$$

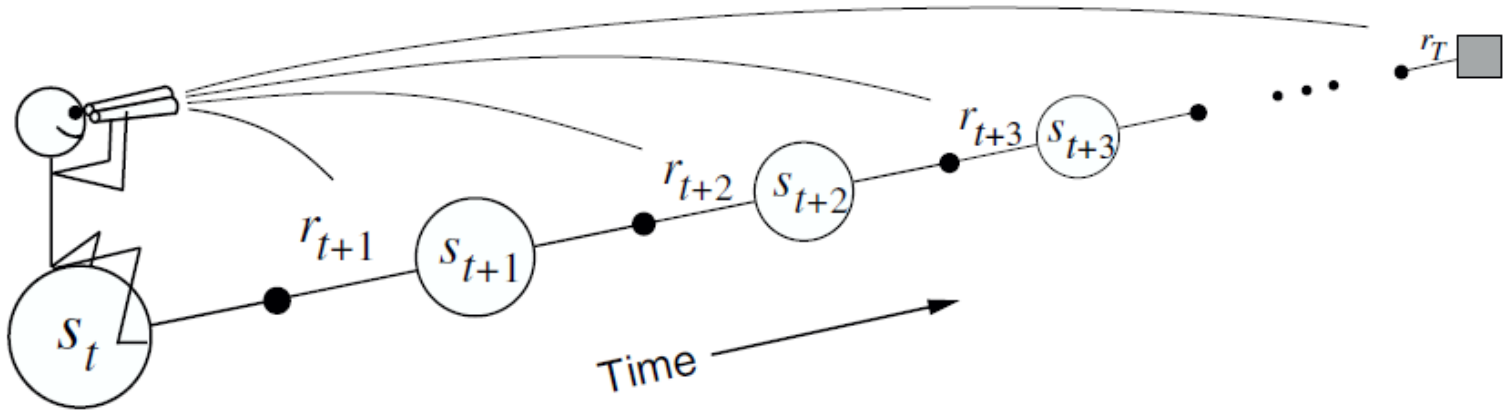


$$\Delta V(s_t) = \alpha[G_t^{\lambda} - V(s_t)]$$

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha\Delta V(s_t)$$

时序差分学习TD(λ)

- TD(λ)前向视角的解释：假设一个人坐在状态流上拿着望远镜看向前方，前方是那些将来的状态。当估计当前状态的值函数时，从TD(λ)的定义中可以看到，它需要用到将来时刻的值函数。



$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right) \qquad G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

需要等到试验结束才能计算

适合度轨迹/资格迹（Eligibility Traces）

- 跟踪上次访问特定状态的轨迹，然后将当前回报分配给最近访问的状态，长时间没有被访问的状态则没有资格获得当前回报
- 更新资格轨如下：（后向观点）

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) \; if \; s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 \;\; if \; s = s_t \end{cases}$$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

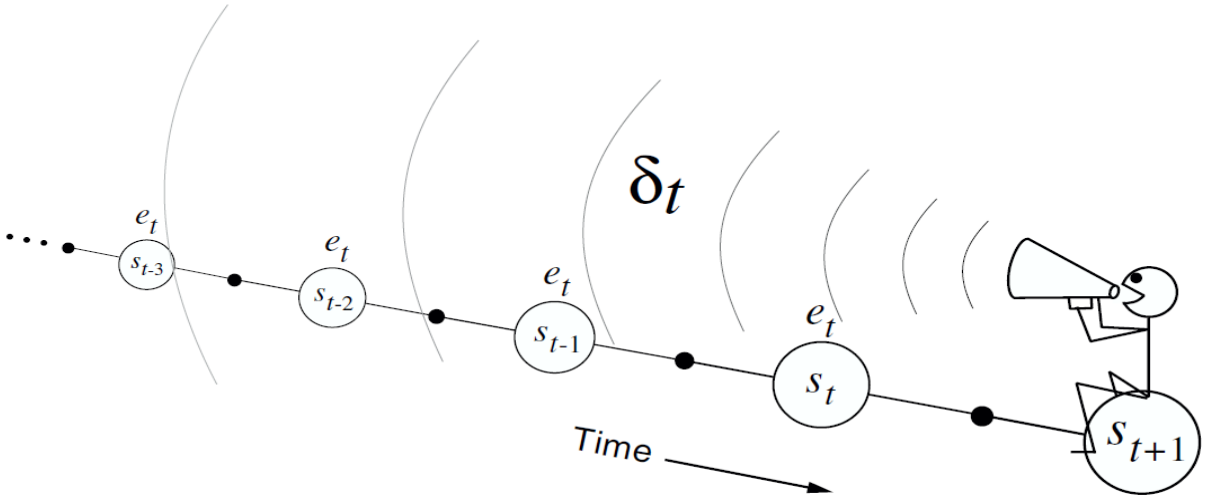
$$\Delta V(s) = \alpha \delta_t e_t(s) \; \forall s$$

可以证明前向观点和后向观点的等价性

Richard S. Sutton Reinforcement Learning-An Introduction, MIT press, 1998.

时序差分学习TD(λ)

- TD(λ)的后向视角解释：已经经历过的状态处的值函数需要利用当前时刻的TD偏差进行更新。过往的每个状态值函数更新的大小应该跟距离当前状态的步数有关



适合度轨迹/资格迹方法

时序差分学习TD(λ)

- 1 初始化$V(s)$为任意值（随机数）
- 2 初始化s为任意值，$e(s)=0$
- 3 repeat
- 4 根据策略$\pi(s)$选择行为a
- 5 执行行为a，观测回报并转到下一状态s'
- 6 计算TD误差，$\delta = r + \gamma V(s') - V(s)$
- 7 计算$e(s) = e(s) + 1$
- 8 repeat
- 9 $V(s) = V(s) + \alpha\delta e(s)$
- 10 $e(s) = \gamma\lambda e(s)$
- 11 until 所有状态更新，并设$s = s'$
- 12 until 完成特定步数或者s到达终止状态

时序差分学习算法TD(λ)收敛性

基础：随机逼近定理

Theorem: Converges w.p. 1 under certain boundaries conditions. Decrease $\alpha_i(t)$ s.t.

$$\sum_t \alpha_i(t) = \infty$$

$$\sum_t \alpha_i^2(t) < \infty$$

In practice, often a fixed α is used for all i and t.

Michael Jordan et. al. On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. Neural Computation,1994

时序差分学习特点

- 结合了蒙特卡罗和动态规划两种方法
- 不需要环境模型，只需要经验数据
- 增量式的学习方式
- 在试验中学习，不必等到试验结束，更小内存消耗，更少计算量
- $V(s_{t+1})$采用估计值，因此为有偏估计（蒙特卡洛方法为无偏估计）
- 只用了一步随机状态和动作，因此，TD目标的随机性比蒙特卡罗方法中的G_t要小，因此具有小的方差
- 通过预测每个动作的长期结果来给先前的动作赋予奖励或惩罚，即依赖于后续状态的值函数来更新先前状态的值函数

Q学习

- Q学习（Q-Learning）[Watkin 1989,1992]是一种异策略的时序差分学习算法（无模型强化学习算法）
  - 采用状态—动作值函数作为估计函数(表格)，每次学习迭代时都需要考察每一个行为
  - Q函数的估计方法为

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

  相当于让$Q(s,a)$直接去估计最优状态值函数$Q^*(s,a)$
  - 根据ϵ贪心策略确定动作

Q学习算法

- 输入：状态空间S,动作空间A,折扣率γ, 学习率α
- 1 随机初始化$Q(s,a)$
- 2 $\forall s, \forall a, \pi(a|s) = \frac{1}{|A|}$
- 3 **repeat**
- 4 初始化起始状态s;
- 5 **repeat**
- 6 在状态s, 选择动作$a = \pi^{\epsilon}(s)$
- 7 执行动作a,得到即时奖励r和新状态s'
- 8 $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
- 9 $s \leftarrow s'$
- 12 **until** s为终止状态或完成特定步长数
- 13 **until** $\forall s, a, Q(s,a)$收敛
- 输出：策略$\pi(s) = \arg\max_{a \in A} Q(s,a)$

行动策略

评估策略为贪婪策略

Q学习-收敛性

- 当满足如下两个条件时，Q学习可以在时间趋于无穷时得到最优控制策略

1)

$$\sum_{t=0}^{\infty} \alpha_t^2 < \infty, \sum_{t=0}^{\infty} \alpha_t = \infty$$

2) 所有的状态动作都能够被无限次地遍历

SARSA学习算法

- Rummery 和Niranjan提出，1994
- Q学习采用值函数的最大值进行迭代，而SARSA采用实际的Q值进行迭代，它严格的根据执行某个策略所获得的经验来更新值函数

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

- Q学习算法中选择策略和值函数的迭代相互独立（异策略）
- SARSA以严格的TD学习形式实现行为值函数的迭代，即选择策略和值函数的迭代一致（同策略的时序差分学习）

SARSA学习算法

- 时序差分学习方法

$$\hat{Q}_k^{\pi}(s,a) = \hat{Q}_{k-1}^{\pi}(s,a) + \frac{\mathbf{1}}{\boldsymbol{k}}\left(G\left(\tau^{(k)}\right) - \hat{Q}_{k-1}^{\pi}(s,a)\right)$$

$$\hat{Q}_k^{\pi}(s,a) = \hat{Q}_{k-1}^{\pi}(s,a) + \boldsymbol{\alpha}\left(G\left(\tau^{(k)}\right) - \hat{Q}_{k-1}^{\pi}(s,a)\right)$$

借助于动态规划来计算$G\left(\tau^{(k)}\right)$,不需要得到完整轨迹

从s,a开始，采样下一个状态和动作(s',a'),得到奖励$r(s,a,s')$,利用贝尔曼方程近似估计$G\left(\tau^{(k)}\right)$

SARSA学习算法

$$G\left(\tau_{0:T}^{(k)}\right) = r(s,a,s') + \gamma G\left(\tau_{1:T}^{(k)} \middle| \tau_{s_1} = s', \tau_{a_1} = a'\right)$$
$$\cong r(s,a,s') + \gamma \hat{Q}_{k-1}^{\pi}(s',a')$$

$$\hat{Q}_k^{\pi}(s,a) = \hat{Q}_{k-1}^{\pi}(s,a) + \boldsymbol{\alpha}(r(s,a,s') + \gamma \hat{Q}_{k-1}^{\pi}(s',a') - \hat{Q}_{k-1}^{\pi}(s,a))$$
$$= (\mathbf{1} - \boldsymbol{\alpha})\hat{Q}_{k-1}^{\pi}(s,a) + \boldsymbol{\alpha}(r(s,a,s') + \gamma \hat{Q}_{k-1}^{\pi}(s',a'))$$

SARSA算法(state action reward state action)

SARSA学习算法

- 输入：状态空间S,动作空间A,折扣率γ, 学习率α
- 1 随机初始化$Q(s,a)$
- 2 $\forall s, \forall a, \pi(a|s) = \frac{1}{|A|}$
- 3 **repeat**
- 4 　初始化起始状态s;
- 5 　选择动作$a = \pi^{\epsilon}(s)$
- 6 　**repeat**
- 7 　　执行动作a,得到即时奖励r和新状态s'
- 8 　　在状态s',选择动作$a' = \pi^{\epsilon}(s')$
- 9 　　$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$
- 10 　　更新策略$\pi(s) = \arg\max_{a \in A} Q(s,a)$
- 11 　　$s \leftarrow s', a \leftarrow a'$
- 12 　**until** s为终止状态或完成特定步长数
- 13 **until** $\forall s, a, Q(s,a)$收敛
- 输出：策略$\pi(s)$

行动策略

评估策略

基于值函数逼近的强化学习方法

■ 表格型强化学习方法

- 状态空间和动作是离散的，值函数是一个表格，索引是状态—行为对，状态和动作空间不能太大

- 状态空间为连续空间的时候，值函数无法用一张表格来表示，需要利用函数逼近方法来表示值函数

基于值函数逼近的强化学习方法

- **蒙特卡罗方法**

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_t - Q(s,a))$$

- **TD方法**

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

- **TD(λ)方法**

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_t^{\lambda} - Q(s,a))$$

- **Q学习方法**

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

基于值函数逼近的强化学习方法

- 函数逼近过程是一个监督学习过程，数据和标签对为(s_t, U_t)

训练目标：

$$\min_{\theta}\left(U_t - \hat{Q}(s,a,\theta)\right)^2$$

$$\min_{\theta}\left(U_t - \hat{V}(s,\theta)\right)^2$$

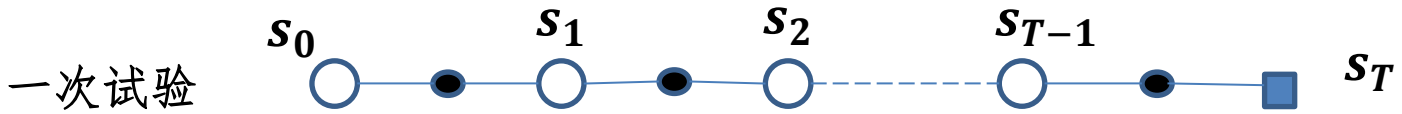
随机梯度下降

$$\theta_{t+1} = \theta_t + \alpha(U_t - \hat{V}(s_t,\theta_t))\nabla_{\theta}\hat{V}(s_t,\theta_t)$$

基于值函数逼近的强化学习方法

■ 基于蒙特卡罗方法的函数逼近

给定要评估的策略π，产生一次实验



收集数据$\langle s_1, G_1\rangle, \langle s_2, G_2\rangle, \cdots, \langle s_T, G_T\rangle$

更新：$t = 1, \ldots, T$

$$\Delta\theta = \alpha\left(G_t - \hat{V}\left(s_t, \theta_t\right)\right)\nabla_\theta \hat{V}\left(s_t, \theta_t\right)$$

基于值函数逼近的强化学习方法

■ 基于半梯度的TD(0)值函数逼近

TD(0)目标函数$U_t = R_{t+1} + \gamma \hat{V}(s_{t+1}, \theta)$

算法：

初始化函数参数θ（如$\theta = 0$）

Repeat

初始化状态S

Repeat

选择动作a $\sim \pi(\cdot|s)$

采用动作a, 观测回报R, s'

$$\theta_{t+1} = \theta_t + \alpha\left(R + \gamma \hat{V}(s', \theta_t) - \hat{V}(s, \theta_t)\right)\nabla \hat{V}(s, \theta_t)$$

$s \leftarrow s'$

直到s'是终止状态

$$\min_{\theta}\left(U_t - \hat{V}(s, \theta)\right)^2$$

基于值函数逼近的强化学习方法

- 值函数逼近：深度Q网络（deep Q-networks, DQN), Mnih et al.[2015] 提出

在连续的状态和动作空间中计算函数$Q^{\pi}(s,a)$可以用一个函数$Q_{\phi}(s,a)$来近似计算

$$Q_{\phi}(\boldsymbol{s},\boldsymbol{a}) \approx Q^{\pi}(s,a)$$

$\boldsymbol{s},\boldsymbol{a}$分别是状态$s$和动作$a$的向量

$Q_{\phi}(\boldsymbol{s},\boldsymbol{a})$是一个参数为$\phi$的函数，可以用神经网络实现，输出一个实数，称为Q网络

学习参数ϕ, 逼近值函数$Q^{\pi}(s,a)$

基于值函数逼近的强化学习方法

如果动作为有限离散的m个，可让Q网络输出一个m维向量

$$Q_\phi(\boldsymbol{s}) = \begin{bmatrix} Q_\phi(s,a_1) \\ \vdots \\ Q_\phi(s,a_m) \end{bmatrix} \approx \begin{bmatrix} Q_\phi^\pi(s,a_1) \\ \vdots \\ Q_\phi^\pi(s,a_m) \end{bmatrix}$$

蒙特卡罗方法 $Q_\phi(\boldsymbol{s},\boldsymbol{a})$逼近平均回报$\boldsymbol{G(t)}$

TD方法 $Q_\phi(\boldsymbol{s},\boldsymbol{a})$逼近$\boldsymbol{E[r+\gamma Q(s'a')]}$

TD(λ)方法 $Q_\phi(\boldsymbol{s},\boldsymbol{a})$逼近$\boldsymbol{G_t^\lambda}$

Q学习方法 $Q_\phi(\boldsymbol{s},\boldsymbol{a})$逼近$\boldsymbol{r+\gamma \max_{a'} Q(s',a')}$

基于值函数逼近的强化学习方法

■ 训练目标函数(Q学习方法)

$$L(s,a,a';\phi)=\left(\boldsymbol{r}+\boldsymbol{\gamma}\max_{\boldsymbol{a'}}\hat{Q}_{\phi}(\boldsymbol{s'},\boldsymbol{a'})-\hat{Q}_{\phi}(\boldsymbol{s},\boldsymbol{a})\right)^2$$

增量式学习方法：随机梯度下降

假定数据和标签对为(s_t,u_t)，则参数随机梯度更新为

$$\phi_{t+1}=\phi_t+\alpha[u_t-\hat{Q}_{\phi_t}(s_t)]\nabla_{\phi}\hat{Q}_{\phi_t}(s_t)]$$

目标函数问题：

- 目标不稳定，参数学习的目标依赖于参数本身
- 样本之间有很强的相关性

基于值函数逼近的强化学习方法

深度Q网络采取两个措施：

- 目标网络冻结（freezing target networks），即在一个时间段内固定目标中的参数，来稳定学习目标；
- 经验回放（experience replay），构建一个经验池来去除数据相关性。经验池是由智能体最近的经历组成的数据集。经验回放可以形象地理解为在回忆中学习。

DQN采用深度卷积神经网络逼近函数值，并设置了目标网络来单独处理时间差分算法中的TD偏差

基于值函数逼近的强化学习方法

- DQN训练：随机从经验池中抽取样本来来代替当前的样本用来进行训练，可以就打破和相邻训练样本的相似性，避免模型陷入局部最优。

- 优先回放

- 智能体的经验即经历过的数据对于智能体的学习并非具有同样重要的作用。智能体在某些状态的学习效率比其他状态学习效率高
- 优先回放打破对历史数据的均匀采样，赋予学习效率高的状态以更大的权重

基于值函数逼近的强化学习方法

- 如何选择权重？理想标准是智能体的学习效率越高，权重越大。一个符合该标准的选择是时间差分（TD）学习的偏差信号δ。TD偏差越大，说明该状态的值函数与TD的目标差距越大，智能体的更新量越大，该状态的学习效率越高。假定样本i处的TD偏差为δ_i,则该处样本的采样概率为

$$\mathrm{P(i)} = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

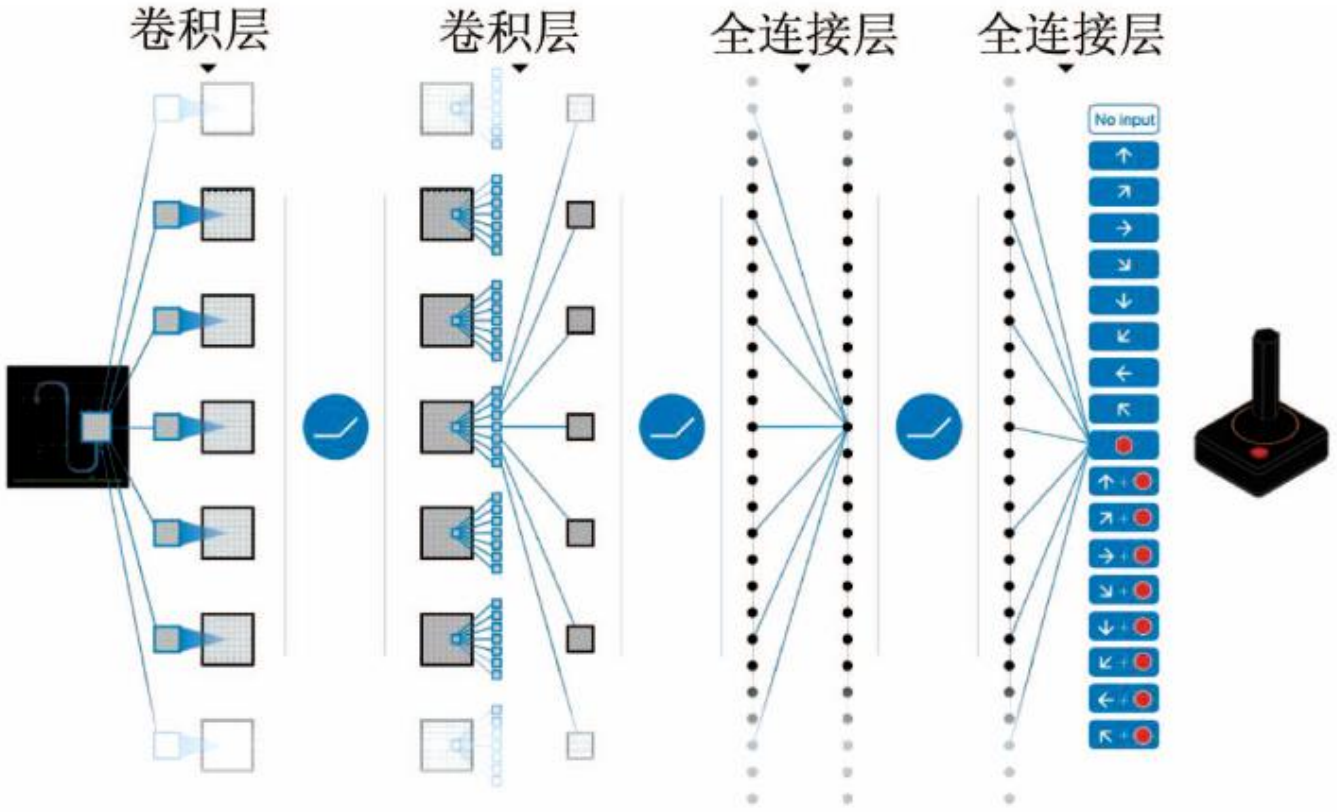
其中p_i^α由TD偏差决定，一般可选的形式有$p_i = |\delta_i| + \epsilon$或$p_i = \frac{1}{rank(i)}$, 其中$rank(i)$根据$|\delta_i|$排序得到

基于值函数逼近的强化学习方法

- 当采用优先回放概率分布采样时，动作值函数估计是一个有偏估计，因为采样分布与动作值函数的分布式两个完全不同的分布，为此，可采用重要性采样系数$W_i = \left(\frac{1}{N \cdot P(i)}\right)^{\beta}$来矫正这个偏差。

- 同时为克服Q学习容易过估计的固有缺点，采用将动作选择和动作评估分别用不同的值函数来实现（Double DQN）

基于值函数逼近的强化学习方法



DQN行为值函数逼近网络

基于值函数逼近的强化学习方法

■ 优先回放Double DQN算法伪代码

- 1：输入 minibatch大小k，步长ε，回放周期K，存储总数据大小N，常数α, β，总时间T
- 2：初始化回放记忆库H $= \emptyset, \Delta = 0, p_1 = 0$
- 3：观测初始状态S_0，选择动作$A_0 \sim \pi_\theta(S_0)$
- 4：for t=1 to T do
- 5：　利用动作A作用于环境，观测S_t, R_t, γ_t
- 6：　将数据$(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$存储到记忆库H中，令其优先级为$p_t = \max_{i<t} p_i$
- 7：　if t $\equiv$ 0 mod K then
- 8：　　for j=1 to k do
- 9：　　采样j~P(j) $= \frac{p_j^\alpha}{\sum_k p_k^\alpha}$

基于值函数逼近的强化学习方法

- 10: 计算样本点重要性权重 $w_j = \frac{(N \cdot P(j))^{-\beta}}{\max_i w_i}$
- 11: 计算TD误差 $\delta_j = R_j + \gamma_j Q_{target}\left(S_j, \arg\max_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$
- 12: 更新样本权重 $p_j \leftarrow |\delta_j|$
- 13: 累积权重的改变量 $\Delta \leftarrow \Delta + w_j \delta_j \nabla_\theta Q(S_{j-1}, A_{j-1})$
- 14: end for
- 15: 更新权重 $\theta \leftarrow \theta + \varepsilon \cdot \Delta$，重置 $\Delta = 0$
- 16: 偶尔地复制新权重到目标网络中 $\theta_{target} \leftarrow \theta$
- 17: end if
- 18: 选择动作 $A_t \sim \pi_\theta(S_t)$
- 19: end for

基于策略搜索的强化学习方法

- 值函数法中，行为值的微小变化会引起策略的很大变化，采用函数逼近器的值函数方法在很多问题中不能保证收敛

■ 策略搜索（policy search）：

- 在策略空间直接搜索来得到最佳策略
- 本质是一个优化问题，可以分为基于梯度的优化和无梯度优化
- 和基于值函数的方法相比，策略搜索可以不需要值函数，直接优化策略。参数化的策略能够处理连续状态和动作，可以直接学出随机性策略

基于策略搜索的强化学习方法

■ 策略搜索途径：

将策略参数化，即$\pi_\theta(s)$：利用参数的线性函数或非线性函数（如神经网络）表示策略，寻找最优参数，使累积回报的期望最大

- 与对值函数进行参数化相比，策略参数化更简单，有更好的收敛性
- 利用值函数求解最优策略时，策略改善需求解$\arg\max_a Q_\phi(s,a)$，当需要解决的问题动作空间很大或者动作为连续集时，该式无法有效求解
- 直接策略搜索方法通常采用随机策略，因为随机策略可以将探索直接集成到所学习的策略之中

基于策略搜索的强化学习方法

■ 常见策略表示

- 线性策略：构造特征向量$X(s)$

$$\mu_\theta(s) = \theta^T X(s)$$

- 径向基策略

$$\pi_\theta(s) = \omega^T \phi(s)$$

$$\phi_i(s) = \exp\left(-\frac{1}{2}(s-\mu_i)^T \mathrm{D}_i(s-\mu_i)\right), \theta = \{\omega, \mu_i, d_i\}$$

- 随机策略：确定性策略+随机部分

$$\pi_\theta = \mu_\theta + \epsilon$$

基于策略搜索的强化学习方法

■ 策略梯度推导

- 用τ表示一组状态行动序列$s_0, a_0, \cdots, s_H, a_H$
- $G(\tau) = \sum_{t=0}^{H} r(s_t, a_t)$表示轨迹$\tau$的回报，$P(\tau;\theta)$表示轨迹$\tau$出现的概率
- 强化学习目标函数

$$J(\theta) = E\left(\sum_{t=0}^{H} r(s_t, a_t); \pi_\theta\right) = \sum_{\tau} P(\tau;\theta)G(\tau)$$

- 学习目标为找到最优参数θ,使得

$$\max_\theta J(\theta) = \max_\theta \sum_{\tau} P(\tau;\theta)G(\tau)$$

基于策略搜索的强化学习方法

- 最速下降法（策略梯度法）

关键：计算$\nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P(\tau;\theta) G(\tau) = \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) G(\tau)$$

$$= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} G(\tau) = \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta)\, G(\tau)$$

求策略梯度⇔求$\nabla_\theta \log P(\tau;\theta)\, \boldsymbol{G}(\tau)$期望

m条轨迹的经验平均逼近策略梯度

$$\nabla_\theta J(\theta) \approx \hat{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau_i;\theta)\, \boldsymbol{G}(\tau_i)$$

基于策略搜索的强化学习方法

$\nabla_\theta \log P(\tau;\theta)$ 的计算

$$\tau = s_0, a_0, \cdots, s_H, a_H$$

动力学，无参数θ

轨迹的似然率可以写为

$$P(\tau;\theta) = \prod_{t=0}^{H} P(s_{t+1}|s_t, a_t)\, \pi_\theta(a_t|s_t)$$

$$\nabla_\theta \log P(\tau;\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} P(s_{t+1}|s_t, a_t)\, \pi_\theta(a_t|s_t)\right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}|s_t, a_t) + \sum_{t=0}^{H} \log \pi_\theta(a_t|s_t)\right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log \pi_\theta(a_t|s_t)\right] = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

基于策略搜索的强化学习方法

$$\nabla_\theta J(\theta) = E_\tau\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right] \quad G(\tau) = \sum_{t=0}^{H-1} \gamma^t r_{t+1}$$

$$t' < t, E_\tau[r(s_{t'}, a_{t'}) \nabla_\theta \log \pi_\theta(a_t|s_t)] = 0$$

$$\nabla_\theta J(\theta) = E_\tau\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t G(\tau_{t:H})\right]$$

$$G(\tau_{t:H}) = \sum_{t'=t}^{H-1} \gamma^{t'-t} r_{t'+1}$$

https://spinningup.openai.com/en/latest/spinningup/extra_pg_proof1.html

基于策略搜索的强化学习方法

$$\nabla_\theta J(\theta) \approx \hat{\boldsymbol{g}} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau_i;\theta)\, \boldsymbol{G}(\tau_i)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\, \boldsymbol{G}(\tau_i)\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\, \gamma^t G\left(\tau_{t:H}^{(i)}\right)\right)$$

- 结合随机梯度上升算法，可以每次采集一条轨迹，计算每个时刻的梯度并更新参数，称为REINFORCE算法[Williams, 1992]

基于策略搜索的强化学习方法

■ REINFORCE算法

- 输入：状态空间S，动作空间A，可微分的策略函数$\pi_\theta(a|s)$，折扣率γ,学习率α
- 1 随机初始化参数θ
- 2 repeat
- 3 根据策略$\pi_\theta(a|s)$生成一条轨迹$\tau = s_0, a_0, \cdots, s_H, a_H$
- 4 for t=0 to H do
- 5 计算$G(\tau_{t:H})$
- 6 $\theta \leftarrow \theta + \alpha\gamma^t G(\tau_{t:H})\nabla_\theta \log \pi_\theta(a_t|s_t)$
- 7 end
- 8 until θ 收敛
- 输出：策略π_θ

基于策略搜索的强化学习方法

REINFORCE算法的一个主要缺点是不同路径之间的方差很大，导致训练不稳定，这是在高维空间中使用蒙特卡罗方法的的通病。一种减少方差的通用方法是引入一个控制变量。

引入与a_t无关的$b(s_t)$减小方差

引理

$$\nabla_\theta J(\theta) \approx \hat{\boldsymbol{g}} = \frac{1}{\boldsymbol{m}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \nabla_\theta \log P(\tau_i;\theta)\, \boldsymbol{G}\left(\boldsymbol{\tau}_{\boldsymbol{t:H}}^{(\boldsymbol{i})}\right)$$

$$= \frac{1}{\boldsymbol{m}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \nabla_\theta \log P(\tau_i;\theta) \left(\boldsymbol{G}\left(\boldsymbol{\tau}_{\boldsymbol{t:H}}^{(\boldsymbol{i})}\right) - \boldsymbol{b}(\boldsymbol{s_t})\right)$$

基于策略搜索的强化学习方法

证明

$$E[\nabla_\theta \log P(\tau_i;\theta)\, b(s_t)\,] = \sum_\tau P(\tau_i;\theta)\nabla_\theta \log P(\tau_i;\theta) b(s_t)$$

$$= \sum_\tau P(\tau_i;\theta)\frac{\nabla_\theta P(\tau_i;\theta) b(s_t)}{P(\tau_i;\theta)} = \sum_\tau \nabla_\theta P(\tau_i;\theta) b(s_t)$$

$$= \nabla_\theta \left(\sum_\tau P(\tau_i;\theta) b(s_t) \right) = \nabla_\theta b(s_t) = 0$$

https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

基于策略搜索的强化学习方法

■ 基准线的选择

- 使策略梯度方差最小的常数基线b

$$X = \nabla_\theta \log P(\tau_i;\theta)\,(G(\tau_i) - b)$$

方差为$Var(X) = E(X - \bar{X})^2 = EX^2 - E\bar{X}^2$

$$\frac{\partial Var(X)}{\partial b} = E\left(X\frac{\partial X}{\partial b}\right) = 0$$

$$b = \frac{\sum_{i=1}^{m}\left[\left(\sum_{t=0}^{H}\nabla_\theta log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)^2 G(\tau)\right]}{\sum_{i=1}^{m}\left[\left(\sum_{t=0}^{H}\nabla_\theta log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)^2\right]}$$

基于策略搜索的强化学习方法

■ 基准线的选择

$b(s_t)$与$G(\tau_{t:H})$越相关，方差越小，一个很自然的选择为令$b(s_t)$为值函数$V(s_t)$

值函数未知，可用函数$V_\phi(s_t)$来逼近

$$\phi^* = \arg\min_{\phi}\left(V(s_t) - V_\phi(s_t)\right)^2$$

带基准线的REINFORCE算法

- 输入：状态空间S,动作空间A,可微分的策略函数$\pi_\theta(a|s)$,可微分的状态值函数$V_\phi(s)$，学习率α,β
- 1 随机初始化参数θ,ϕ
- 2 repeat
- 3 从策略$\pi_\theta(a|s)$生成一条轨迹$\tau = s_0, a_0, \cdots, s_{T-1}, a_{T-1}, s_T$
- 4 for t=0 to T do
- 5 计算$G(\tau_{t:T})$
- 6 $\delta \leftarrow G(\tau_{t:T}) - V_\phi(s_t)$
- 7 $\phi \leftarrow \phi + \beta\delta \frac{\partial V_\phi(s_t)}{\partial \phi}$
- 8 $\theta \leftarrow \theta + \alpha\gamma^t \delta \nabla_\theta log\pi_\theta(a_t|s_t)$
- 9 end
- 10 until θ收敛
- 输出： 策略π_θ

Actor-Critic算法

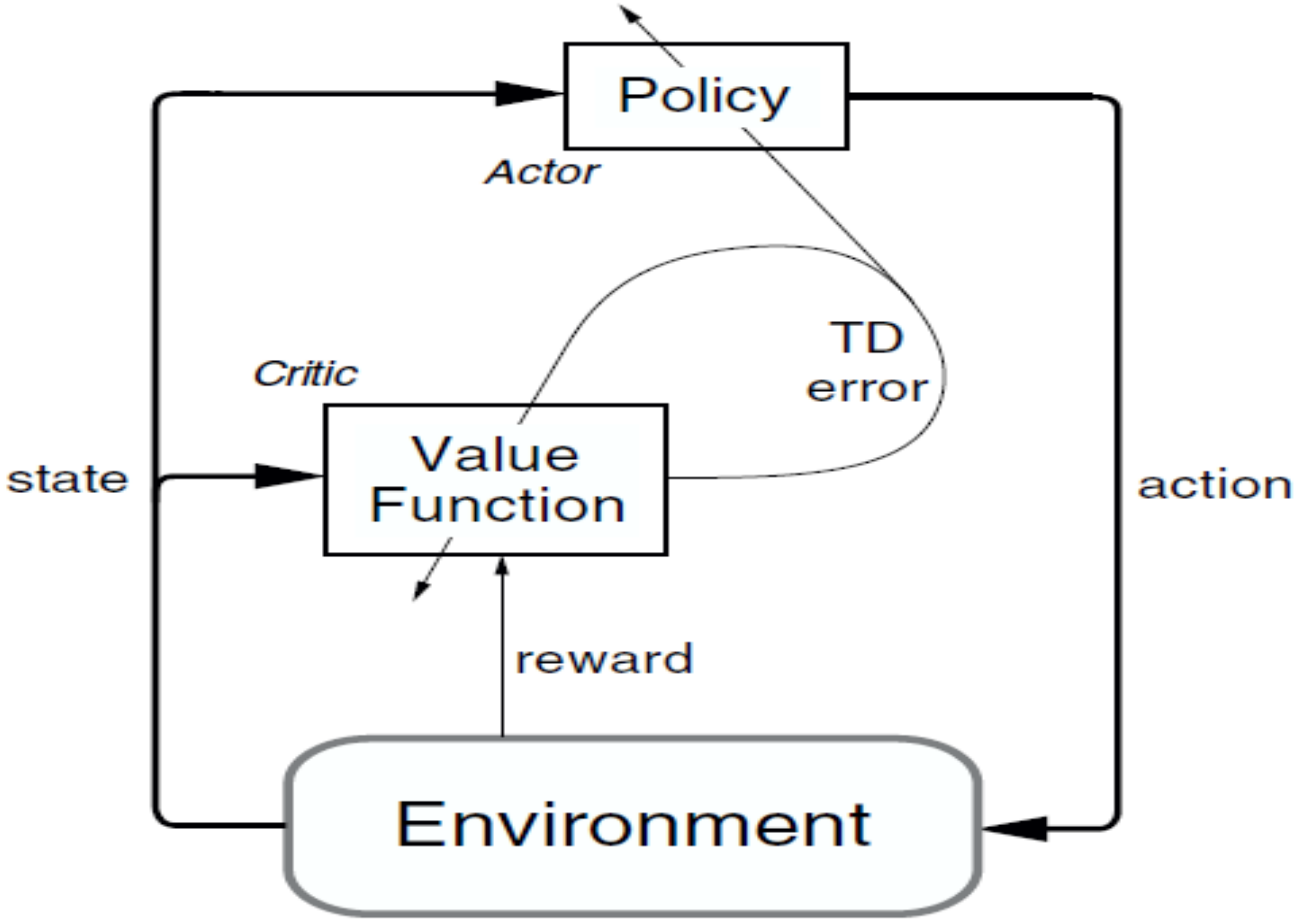
- 在REINFORCE算法中，每次需要根据一个策略采集一条完整的轨迹，并计算这条轨迹上的回报。方差比较大，学习效率也比较低。

- 借鉴时序差分学习的思想，使用动态规划方法来提高采样的效率，即从状态开始s的总回报可以通过当前动作的即时奖励$r(s,a,s')$和下一个状态s'的值函数来近似估计。

Actor-Critic算法

- Actor-Critic算法：一种结合策略梯度和时序差分学习的强化学习方法
- Actor是指策略函数$\pi_\theta(s,a)$，即学习一个策略来得到尽量高的回报
- Critic是指值函数$V_\phi(s)$，对当前策略的值函数进行估计，即评估actor 的好坏
- 借助于值函数，actor-critic 算法可以进行单步更新参数，不需要等到回合结合才进行更新
- $\pi_\theta(s,a)$与$V_\phi(s)$都是待学习的函数，需要在训练过程中同时学习

Actor-Critic算法



The actor–critic architecture

Actor-Critic算法

- 假定从t开始的回报为$G(\tau_{t:T})$

  近似估计$\hat{G}(\tau_{t:T}) = r_{t+1} + \gamma V_\phi(s_{t+1})$

学习过程：

更新参数ϕ使得值函数$V_\phi(s_t)$接近于估计的真实回报$\hat{G}(\tau_{t:T})$

$$\min_\phi \left(\hat{G}(\tau_{t:T}) - V_\phi(s_t)\right)^2$$

将值函数$V_\phi(s_t)$作为基函数来更新参数θ，减小策略梯度方程

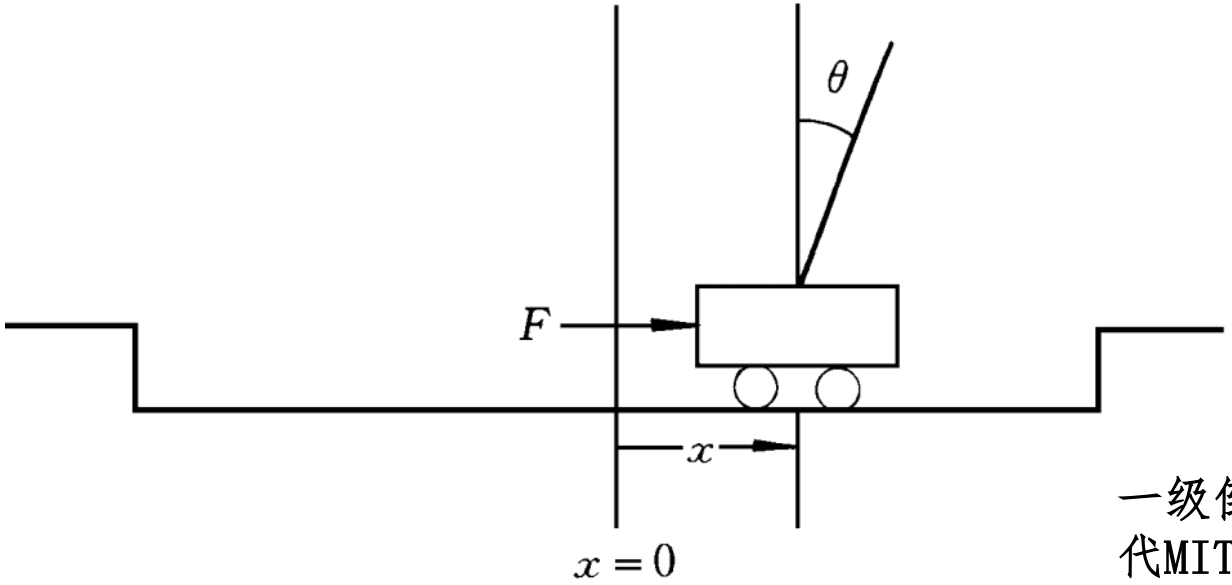
$$\theta \leftarrow \theta + \alpha\gamma^t \left(\hat{G}(\tau_{t:T}) - V_\phi(s_t)\right) \frac{\partial}{\partial\theta} log\pi_\theta(a_t|s_t)$$

Actor-Critic算法

- 输入：状态空间S,动作空间A,可微分的策略函数$\pi_\theta(a|s)$,可微分的状态值函数$V_\phi(s)$，γ，学习率α,β
- 1 随机初始化参数θ,ϕ;
- 2 repeat
- 3 初始化起始状态s；$\lambda=1$;
- 4 repeat
- 5 在状态s，选择动作$a=\pi_\theta(a|s)$;执行动作a，得到即时奖励r和新状态s'
- 6 $\delta \leftarrow r+\gamma V_\phi(s')-V_\phi(s)$
- 7 $\phi \leftarrow \phi+\beta\delta\frac{\partial V_\phi(s_t)}{\partial\phi}$
- 8 $\theta \leftarrow \theta+\alpha\lambda\delta\nabla_\theta log\pi_\theta(a_t|s_t),\lambda\leftarrow\gamma\lambda$
- 10 $s\leftarrow s'$
- 11 until s为终止状态
- 10 until θ收敛
- 输出：策略π_θ

算法	步骤
SARSA	（1）执行策略，生成样本：s, a, r, s', a' （2）估计回报：$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$ （3）更新策略：$\pi(s) = \arg\max_{a \in \lvert\mathcal{A}\rvert} Q(s,a)$
Q学习	（1）执行策略，生成样本：s, a, r, s' （2）估计回报：$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big)$ （3）更新策略：$\pi(s) = \arg\max_{a \in \lvert\mathcal{A}\rvert} Q(s,a)$
REINFORCE	（1）执行策略，生成样本：$\tau = s_0, a_0, s_1, a_1, \cdots$ （2）估计回报：$G(\tau) = \sum_{t=0}^{T-1} r_{t+1}$ （3）更新策略：$\theta \leftarrow \theta + \sum_{t=0}^{T-1} \Big(\frac{\partial}{\partial\theta} \log \pi_\theta(a_t \lvert s_t) \gamma^t G(\tau_{t:T})\Big)$
Actor-Critic	（1）执行策略，生成样本：s, a, s', r （2）估计回报：$G(s) = r + \gamma V_\phi(s')$ $\phi \leftarrow \phi + \beta\Big(G(s) - V_\phi(s)\Big) \frac{\partial}{\partial\phi} V_\phi(s)$ （3）更新策略：$\lambda \leftarrow \gamma\lambda$ $\theta \leftarrow \theta + \alpha\lambda\Big(G(s) - V_\phi(s)\Big) \frac{\partial}{\partial\theta} \log \pi_\theta(a \lvert s)$

强化学习倒立摆控制



一级倒立摆系统

一级倒立摆：20世纪50年代MIT控制论专家根据火箭发射助推器原理设计的试验设备
二级倒立摆：双足机器人控制问题

蒋国飞,吴沧浦.基于Q学习算法和BP神经网络的倒立摆控制. 自动化学报, Vol.24, No.5, 1998.

强化学习倒立摆控制

$$\ddot{\theta}_t = \frac{g\sin\theta_t + \cos\theta_t\left(\frac{-F_t - ml\dot{\theta}_t^2 + \mu_c sgn(\dot{x}_t)}{m_c + m}\right) - \frac{\mu_p\dot{\theta}_t}{ml}}{l\left(\frac{4}{3} - \frac{m\cos^2\theta_t}{m_c + m}\right)}$$

$$\ddot{x} = \frac{F_t - ml(\dot{\theta}_t^2\sin\theta_t - \ddot{\theta}_t\cos\theta_t) - \mu_c sgn(\dot{x}_t)}{m_c + m}$$

$g = 9.8\,m/s^2, m_c = 1.0kg, m = 0.1kg, l = 0.5m, \mu_c = 0.0005, \mu_p = 0.000002, F_t = \pm 10N$

轨道长度4.8m

μ_c和μ_p分别为小车和轨道及小车和倒立摆的摩擦系数

强化学习倒立摆控制

- 任务失败：倒立摆偏离垂直方向的角度超过$\pm 12^{\circ}$或者小车在± 2.4米处和轨道两端相撞

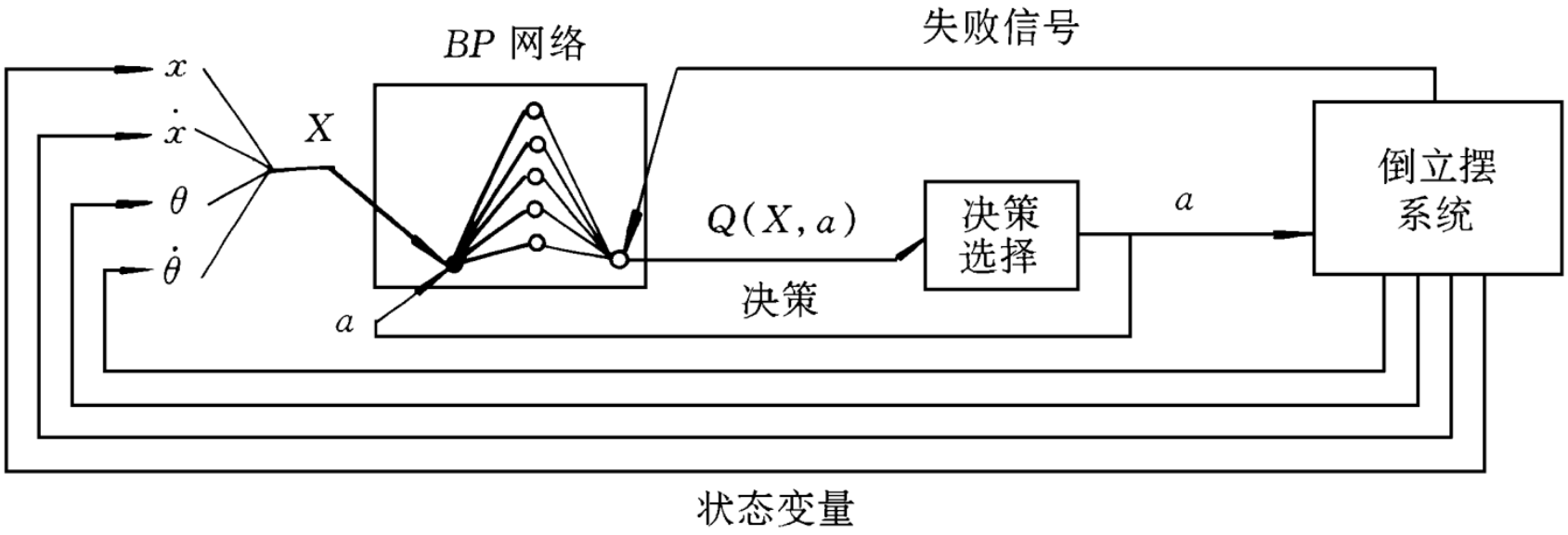
- 定义即时回报

$$r_t = f(x) = \begin{cases} -1, & \text{if } |\theta_t| > 12^{\circ} \text{ or } |x_t| > 2.4\text{m} \\ 0, & \text{other} \end{cases}$$

控制器是在执行了一系列决策后才得到这个延迟的失败信号，控制器必须解决奖励或惩罚随时间分配的问题，即确定在这过程中哪些决策应该对最后的失败负责

强化学习倒立摆控制

- Q学习算法在各时间步 Q值的更新迭代中将这失败信号进行反传并根据 Q值来确定相应决策的优劣



基于 Q学习和 BP网络的倒立摆控制系统的结构图

强化学习倒立摆控制

- 输入：$X = \left(x, \dot{x}, \theta, \dot{\theta}\right)^T, a$
- 输出：$Q(X, a, W)$
- 在每个时间步k
- 观测当前状态X_k
- 根据BP网络的实际输出$Q(X_k, a, W_k)$值并按照某种探索策略选择当前决策a_k
- 观测倒立摆后续状态X_{k+1},检测即时回报r_k
- 更新(X_k, a_k)的Q值，利用误差信号$e = Q(X_k, a_k) - Q(X_k, a_k, W_k)$更新BP权值$W_k$为$W_{k+1}$
- 转到时间步k+1

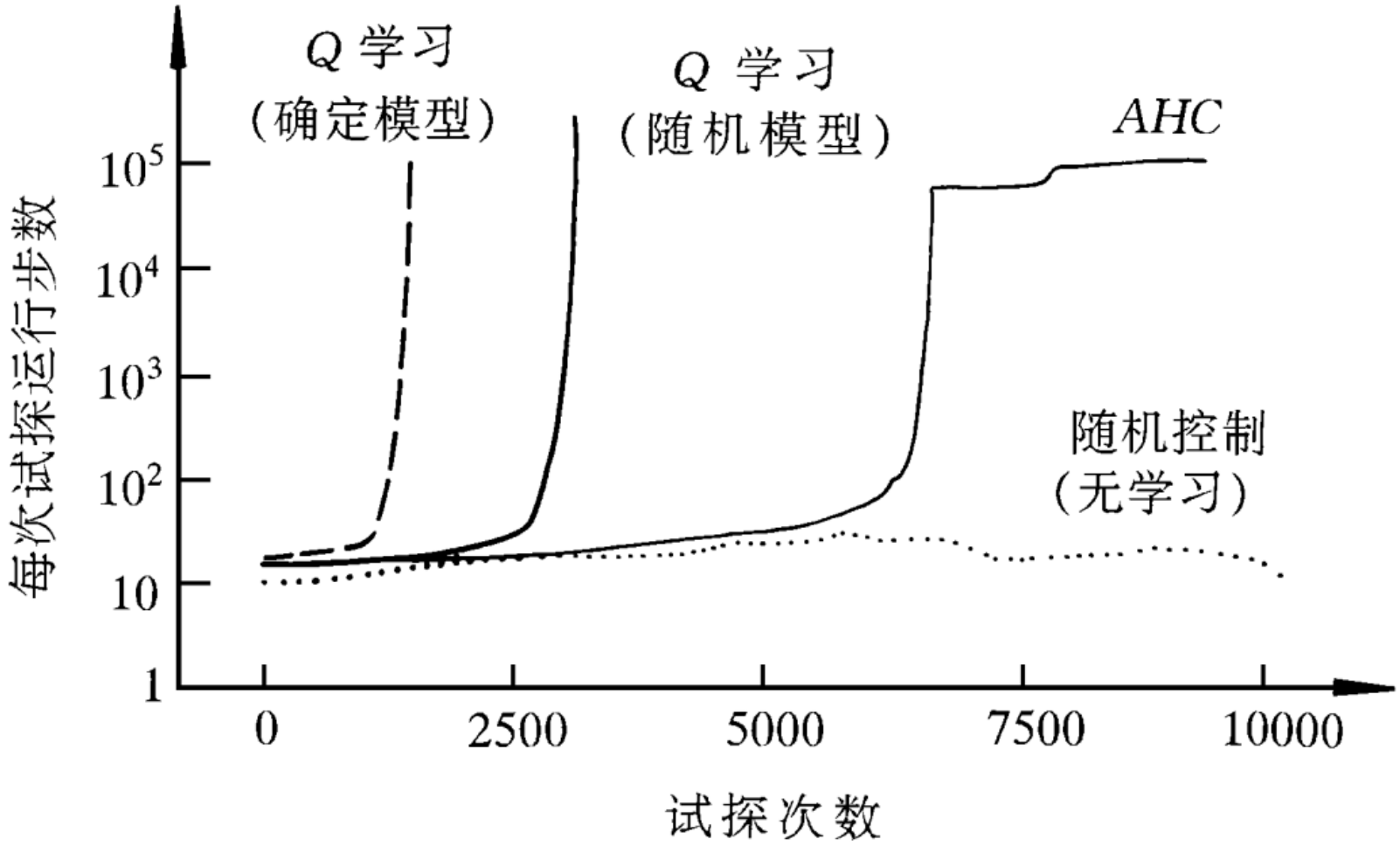
$$\boldsymbol{Q(X_k, a_k) = (1 - \alpha_k) Q(X_k, a_k, W_k) + \alpha_k (r_k + \gamma \max_b Q(X_{k+1}, b, W_k))}$$

$$\boldsymbol{\alpha_k = 0.2, \gamma = 0.95}$$

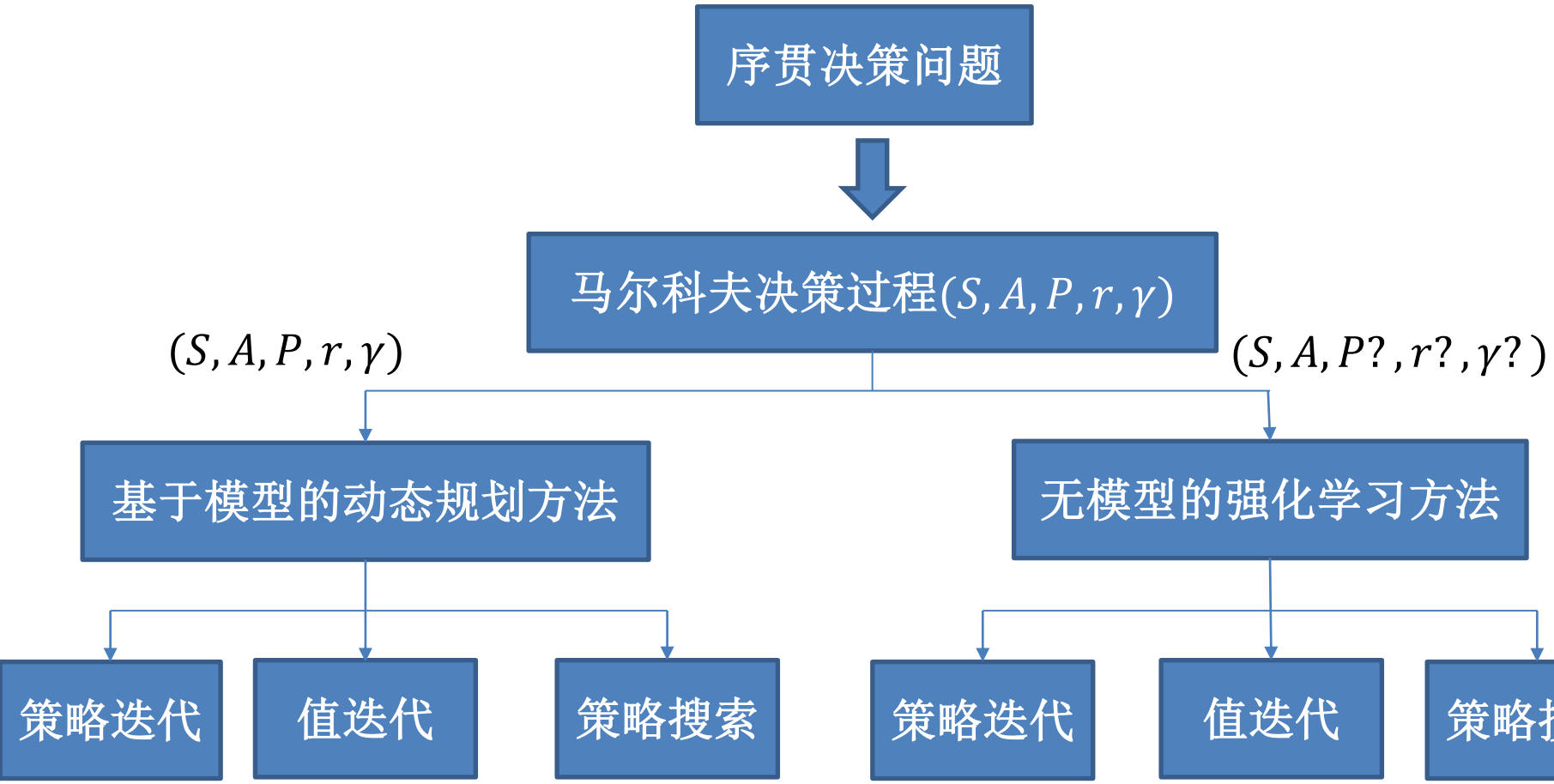
强化学习倒立摆控制

- BP网络输入标准化，分布在[-1.0,1.0]之间
- 确定模型试验和随机模型（加入高斯噪声N(0,0.1)）试验各10次
- 每次试验当倒立摆的试探次数(失败次数)超过10000次或一次试探的平衡步数超过500000步时,中止倒立摆的学习并重新开始另一次试验
- 如果倒立摆在一次试探中能保持500000步不到,就认为本次试验已经能成功控制倒立摆平衡了
- 在每次平衡失败后，将初始状态复位为一定范围内的随机值

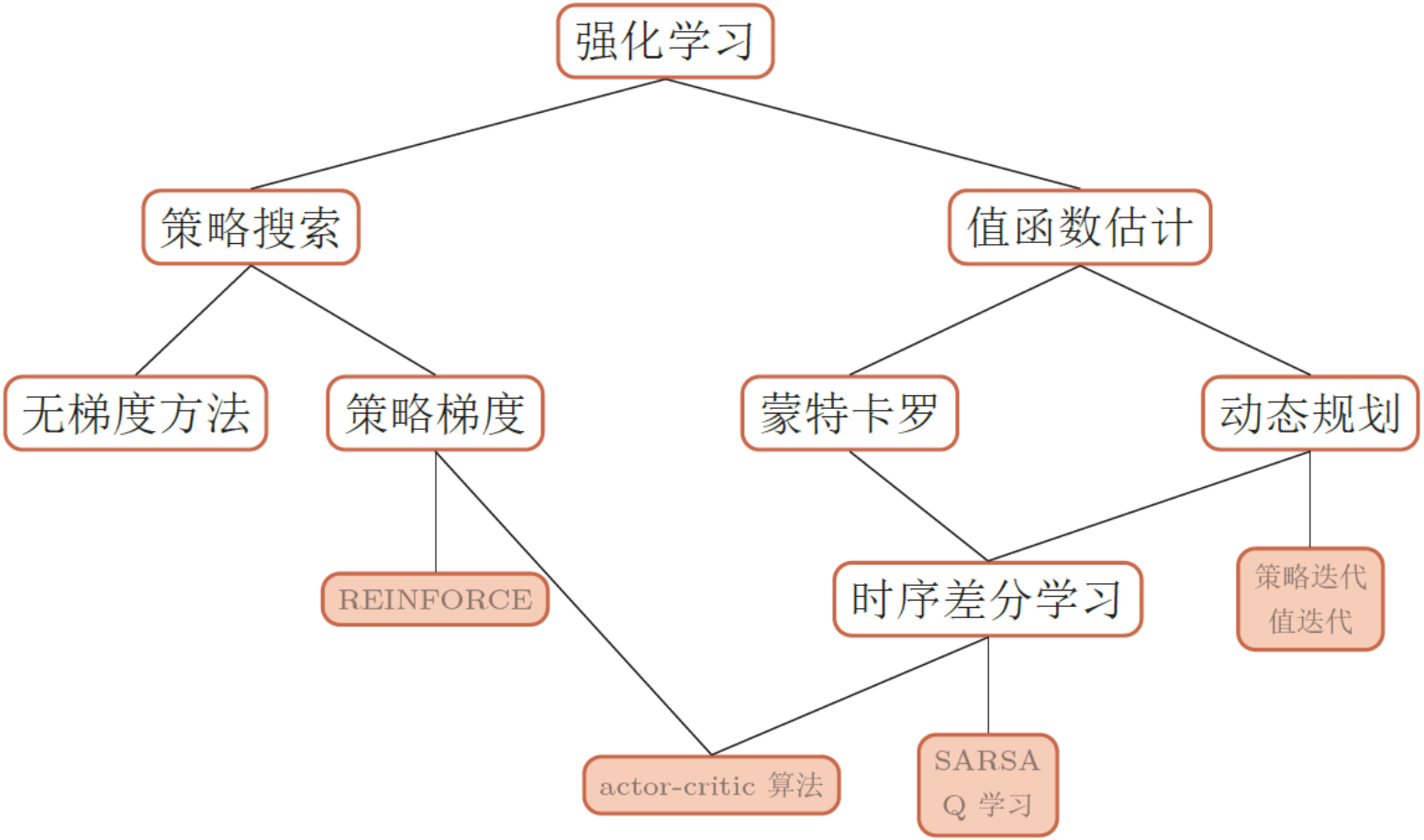
强化学习倒立摆控制


Q 学习
（确定模型）
Q 学习
（随机模型）
AHC
随机控制
（无学习）
每次试探运行步数
10^5
10^4
10^3
10^2
10
1
0
2500
5000
7500
10000
试探次数

强化学习分类


序贯决策问题
马尔科夫决策过程(S, A, P, r, γ)
(S, A, P, r, γ)
$(S, A, P?, r?, \gamma?)$
基于模型的动态规划方法
无模型的强化学习方法
策略迭代
值迭代
策略搜索
策略迭代
值迭代
策略搜索

不同强化学习算法之间的关系



强化学习发展历史

■ 早期主线：

1、动物学习心理学，通过试误（Trial and Error）达到学习目的

- 1911 Edward Thorndike,“效应定律（Law of Effect）”强调行为的结果有优劣之分并成为行为选择的依据，同时指出能够导致正的回报的行为选择概率将增加，而导致负回报的行为选择概率将降低
- 1954 Minsky 等最早进行“试误”学习计算研究，提出了SNARCs(Stochastic Neural-Analog Reinforcement Calculators)的强化学习计算模型

强化学习发展历史

- 1961 年Minsky 的论文《Steps Toward Artificial Intelligence》讨论了几个与强化学习相关的主题，其中包括信任分配问题（Credit Assignment Problem），这问个题是强化学习必须涉及到的，也是至今研究最多的难点
- 1965 年Waltz，1966 年Mendel 等较早的在工程文献中引用了“强化”和“强化学习

强化学习发展历史

2、最优控制的研究：值函数和动态规划，不涉及学习

动态规划是由Bellman 等在20 世纪50 年代晚期提出，同时他于1957 年提出了著名的马尔可夫决策过程（Markov Decision Processes，MDP），即最优控制问题的离散统计模式。策略迭代和值迭代是DP 的两个主要方法。

强化学习发展历史

3、时序差分方法的研究

时序差分是指对同一个事件或变量在连续两个时刻观测的差值，这一概念来自于学习心理学中有关“次要强化器”（Secondary Reinforcer）的研究

- 1959 年Samuel 第一次提出和执行了一个含有时序差分方法的学习算法用于下棋程序
- 1972 年，Klopf 将时间序分方法作为试误学习的一个重要部分，用兴奋的输入作为奖励同时用抑制的输入作为惩罚

强化学习发展历史

3、时序差分方法的研究

- 1981年，研究者提出了将时序差分方法用到试误学习的“动作—评价”结构（Actor-Critic Architecture）
- 1984 Sutton年, 在他的博士论文中提出了AHC（Adaptive Heuristic Critic）算法，比较系统的介绍了AHC 思想
- 1988 年 Sutton出了TD 方法,解决了强化学习中根据时间序列进行预测的问题,并且在一些简化条件下证明了TD 方法的收敛
- 1989年Chris Watkins提出的Q-Learning，将TD和动态规划与其很好地结合起来，并证明了Q-learning的收敛

强化学习发展历史

■ 现代发展：

现代强化学习将三条线索很好的结合起来，得到飞速进展

1994 Rummery等提出了SARSA学习算法

1996 Bertsekas等提出了解决随机过程优化控制的神经动态规划方法

1999 Thrun提出了部分可观测马尔科夫决策过程中的蒙特卡罗方法

2004 Ng A. Y 等提出了基于学徒学习的逆强化学习方法

2006 Kocsis等提出了置信上限树算法

2009 Lewis等提出了反馈控制自适应动态规划算法

2010 Deisenroth 等提出了基于模型的强化学习强化学习方法：PILCO

2014 Silver等提出确定性策略梯度算法

2016 Shulman等提出基于置信域策略优化的强化学习方法

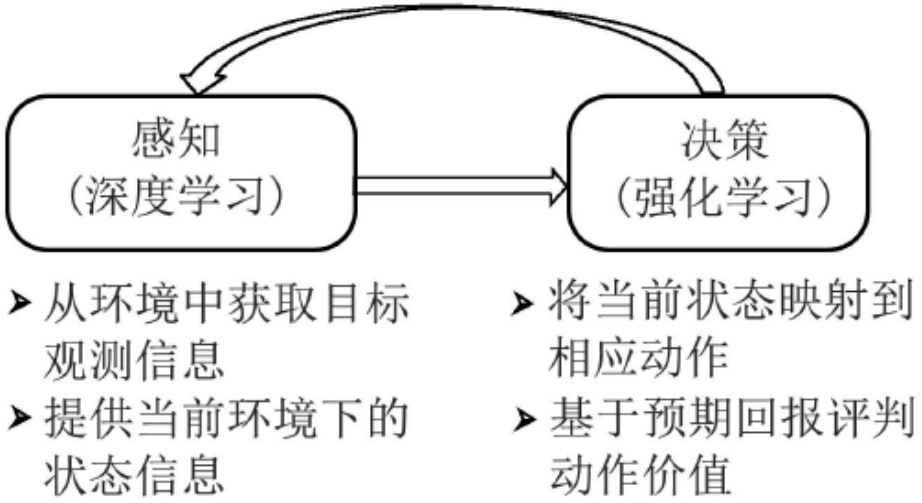
2017 DeepMind 与OpenAI 分别提出具有鲁棒性的无模型强化学习方法：近端优化算法（PPO）

……

强化学习发展历史

■ DeepMind 深度强化学习

具有感知能力的深度学习和具有决策能力的强化学习紧密结合在一起，构成深度强化学习算法，算法的卓越性能远超出人们的想象，极大地震撼了学术界和社会各界



赵冬斌等，深度强化学习综述：兼论计算机围棋的发展. 控制理论与应用. **Vol. 33 No. 6, 2016**

参考书目

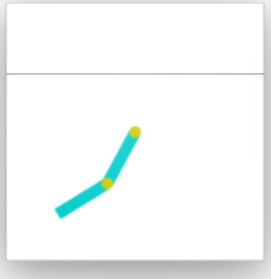
- Richard S. Sutton, Andrew G. Barto. Reinforcement Learning An Introduction. MIT Press, 1998, first edition, 2018 second edition
- 郭宪、方勇纯.深入浅出强化学习. 电子工业出版社, 2018
- 王雪松、朱美强、程玉虎. 强化学习原理及其应用. 科学出版社, 2014
- Howard M. Schwartz. Multi-Agent Machine Learning: A Reinforcement Approach. Wiley 2014
- 邱锡鹏, 神经网络与深度学习

强化学习工具和仿真环境： gym

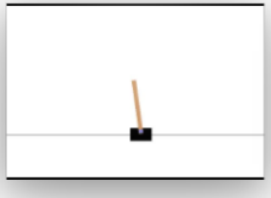
https://gym.openai.com/

Classic control

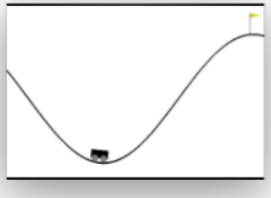
Control theory problems from the classic RL literature.



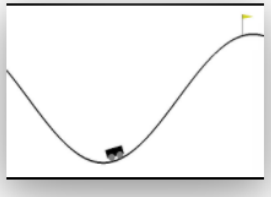
Acrobot-v1
Swing up a two-link robot.



CartPole-v1
Balance a pole on a cart.



MountainCar-v0
Drive up a big hill.



MountainCarContinuous-v0
Drive up a big hill with continuous control.

Pendulum-v0
Swing up a pendulum.

Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

View documentation ›

View on GitHub ›

强化学习算法

https://github.com/openai/baselines

Baselines

OpenAI Baselines is a set of high-quality implementations of reinforcement learning algorithms.

These algorithms will make it easier for the research community to replicate, refine, and identify new ideas, and will create good baselines to build research on top of. Our DQN implementation and its variants are roughly on par with the scores in published papers. We expect they will be used as a base around which new ideas can be added, and as a tool for comparing a new approach against existing ones.

- A2C
- ACER
- ACKTR
- DDPG
- DQN
- GAIL
- HER
- PPO1 (obsolete version, left here temporarily)
- PPO2
- TRPO

强化学习工具和仿真环境： OpenSpiel

https://github.com/deepmind/open_spiel

OpenSpiel: A Framework for Reinforcement Learning in Games

OpenSpiel is a collection of environments and algorithms for research in general reinforcement learning and search/planning in games

Algorithms	Category
Information Set Monte Carlo Tree Search (IS-MCTS)	Search
Minimax (and Alpha-Beta) Search	Search
Monte Carlo Tree Search	Search
Lemke-Howson (via `nashpy`)	Opt.
Sequence-form linear programming	Opt.
Counterfactual Regret Minimization (CFR)	Tabular
CFR against a best responder (CFR-BR)	Tabular
Exploitability / Best response	Tabular
External sampling Monte Carlo CFR	Tabular
Outcome sampling Monte Carlo CFR	Tabular
Q-learning	Tabular
Policy Iteration	Tabular
Value Iteration	Tabular
Advantage Actor-Critic (A2C)	RL
Deep Q-networks (DQN)	RL
Ephemeral Value Adjustments (EVA)	RL
Deep CFR	MARL
Exploitability Descent (ED)	MARL
(Extensive-form) Fictitious Play (XFP)	MARL
Neural Fictitious Self-Play (NFSP)	MARL
Neural Replicator Dynamics (NeuRD)	MARL
Regret Policy Gradients (RPG, RMPG)	MARL

强化学习工具和仿真环境：OpenSpiel

https://github.com/deepmind/open_spiel

Neural Replicator Dynamics (NeuRD)	MARL
Regret Policy Gradients (RPG, RMPG)	MARL
Policy-Space Response Oracles (PSRO)	MARL
Q-based ("all-actions") Policy Gradient (QPG)	MARL
Regression CFR (RCFR)	MARL
Rectified Nash Response (PSRO_rn)	MARL
α-Rank	Eval. / Viz.
Replicator / Evolutionary Dynamics	Eval. / Viz.

双陆棋、突围棋、定约桥牌、Coin Game、屏风式四子棋、协作推箱子、国际象棋、第一价格密封拍卖、围棋、Goofspiel（一种多玩家纸牌游戏）
三宝棋、六贯棋、Kuhn扑克、Leduc扑克、大话骰、Markov Soccer、
配对硬币（3人游戏）、矩阵游戏、Oshi-Zumo、西非播棋、转盘五子棋、
Phantom三连棋、Pig游戏、三连棋、Tiny Bridge、Y（一种棋类游戏）、
Catch（仅支持Python）、Cliff-Walking在悬崖边走的醉汉（仅支持Python）。

常和博弈、零和博弈、协调博弈、一般博弈

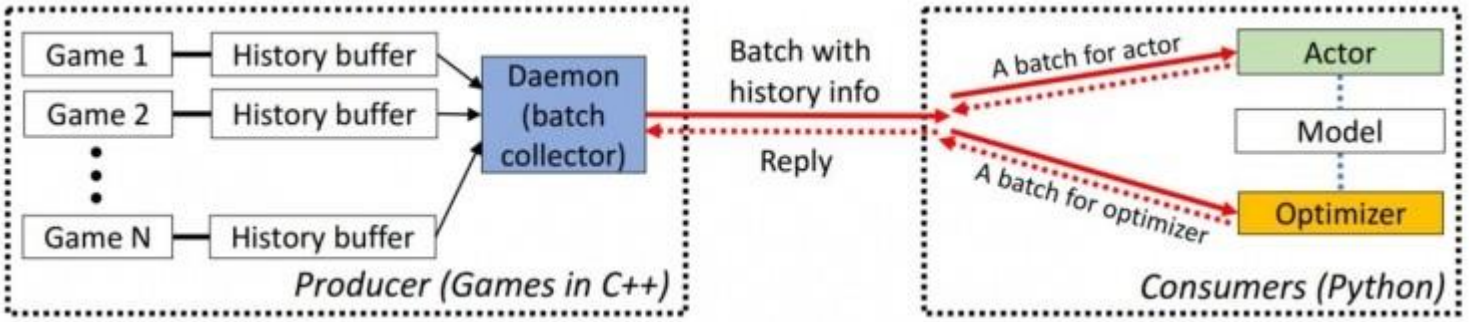
强化学习环境 pysc2（StarCraft II Learning Environment）

https://github.com/deepmind/pysc2

训练星际争霸游戏的AI对战的强化学习平台

强化学习环境： **ELF(** Extensive, Lightweight, Flexible)
Facebook
https://github.com/facebookresearch/ELF
C++/Python 混合框架

- ELF包含三种游戏环境（微型实时策略、夺旗和塔防）的高度可定制的实时策略引擎
- 为策略游戏提供了多样化的属性、高效率的模拟和高度可定制的环境设置。在这个平台上不仅可以更改游戏的参数，也可以构建新的游戏
- 对并行发生的事件的模拟进行了优化



ELF 执行架构

思考

- 强化学习算法思想及与别的类型算法之间的区别？
- 确定性策略和随机策略比较？
- 探索-利用问题？
- 同策略与异策略方法的优缺点？
- 时间差分、蒙特卡洛方法、动态规划方法的区别与联系？
- 时序差分的信用分配机制？
- 引入值函数逼近的原因？优缺点？
- 直接策略搜索与值函数方法的比较？