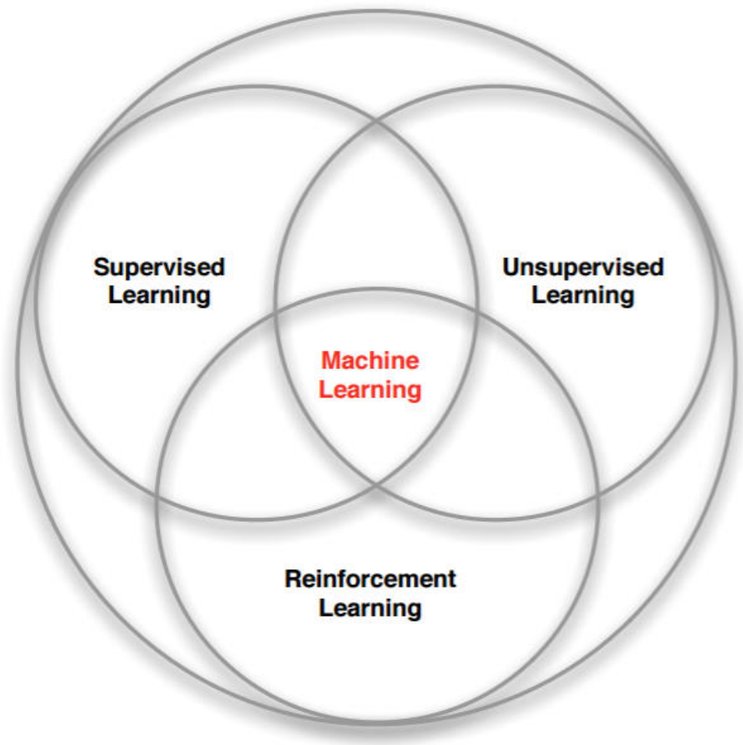
一、模型基础

1. 强化学习与机器学习的关系

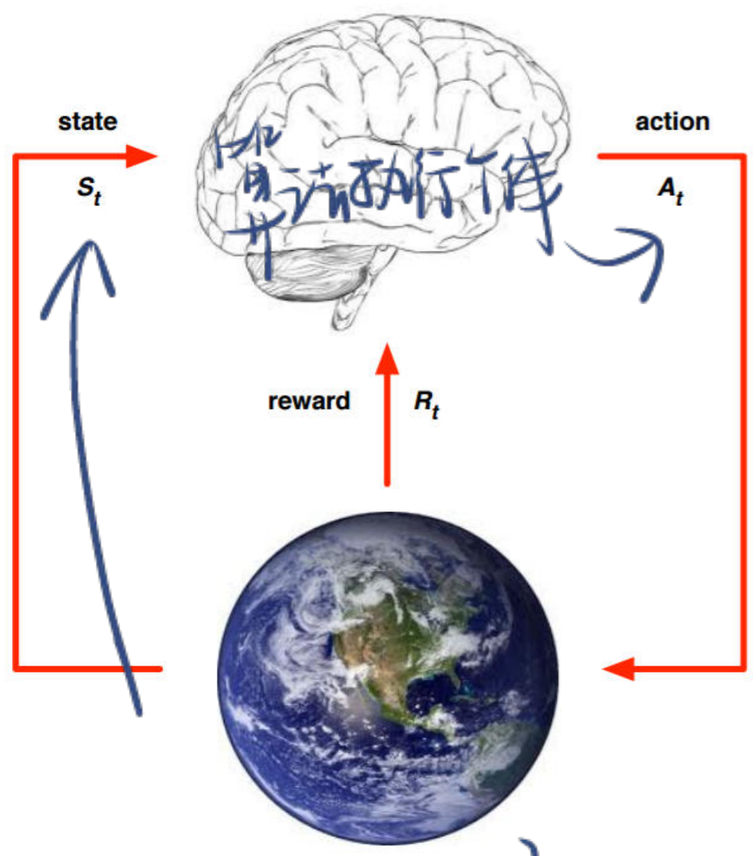
机器学习 { 监督学习
非监督学习
强化学习



监督学习：有标签值（须先存在的）
强化学习：奖励值（延后给出）

非监督学习：无标签值、无奖励值
强化学习：奖励值

2. 强化学习建模



环境 { 初始状态：时刻状态 S_0
算法选择，动作 A_t
环境反馈，< 新的状态 S_{t+1}
延时奖励 R_{t+1}

值

π（采取行动的概率）主观只是选策略的方式的一种

策略π：< 最常见的选择方式：条件概率分布（状态s采取动作a的概率）

条件概率分布 $\pi(a|s) = P(A_t = a \mid S_t = s)$

$v_\pi(s)$（状态s下根据策略π，采取行动的价值）

行动的价值：< 期望函数 $v_\pi(s) = E_\pi(G_t \mid S_t = s)$

$= E_\pi(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t)$

当前状态S_t，执行动作A_t会获得延时奖励R_{t+1}

但不可先看延时奖励R_{t+1}，因为当前延时奖励高，不代表后续奖励也高

因此价值要综合考虑当前延时奖励和后续奖励

奖励衰减因子：$\gamma \in [0,1]$

- $=0$ 贪婪法，价值只由当前延时奖励R_{t+1}决定
- $=1$ 一视同仁
- $\in(0,1)$ 当前延时奖励R_{t+1}权重更大

状态转化模型：< $P^a_{ss'}$（状态s采取动作a转到下一状态s'的概率）

概率状态机/模型 $P^a_{ss'} = E(S_{t+1} = s' \mid S_t = s, A_t = a)$

策略：ε（不选择使当前转移价值最大的动作的概率）前面的，算法选哪个动作

该概率有探索目的，即使某动作可获得更大价值，但不选的概率

二. 马尔科夫决策过程MDP

1. 强化学习引入MDP的原因：

(1) 状态转化模型 $P^a_{ss'}$，转化到下一状态的概率

真实的环境转化：
- 仅与上一个状态有关
- 还与上上个、上上上个…状态有关

⇒ 环境转化模型非常复杂，难以建模

假设状态转化的马尔科夫性：仅与上一个状态有关

$$P^a_{ss'} = E(S_{t+1}=s' \mid S_t=s, A_t=a)$$

(2) 决策策略 $\pi(a|s) = P(A_t=a \mid S_t=s)$

(3) 价值函数 $v_\pi(s) = E(G_t \mid S_t=s)$

$$= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t=s)$$

2. MDP的价值函数贝尔曼方程

$$\begin{cases} \text{状态价值 } v_\pi(s) = E(G_t \mid S_t = s) \\ \text{动作价值 } q_\pi(s,a) = E(G_t \mid S_t = s, A_t = a) \end{cases}$$

$$= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s, A_t = a)$$

贝尔曼方程：

$$v_\pi(s) = E(G_t \mid S_t = s)$$

$$= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s)$$

$$= E(R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s)$$

$$= E(R_{t+1} + \gamma G_{t+1} \mid S_t = s)$$

$$= E(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s)$$

⇒ 状态价值 = 状态奖励 + 后继状态价值

$$q_\pi(s,a) = E(G_t \mid S_t = s, A_t = a)$$

$$= E(R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a)$$

$$= E(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a)$$

$S_t, A_t \to S_{t+1}$

$\pi(a|s) = P(A_t = a \mid S_t = s)$

$v_\pi(s) = E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s)$ $\quad \gamma = \begin{cases} 0 \\ 1 \\ \in (0,1) \end{cases}$

$P^a_{ss'} = P(S_{t+1} = s' \mid S_t = s, A_t = a)$

&

$q_\pi(s,a) = E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s, A_t = a)$

$= E(G_t \mid S_t = s, A_t = a)$

$= E(R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a)$

$= E(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a)$

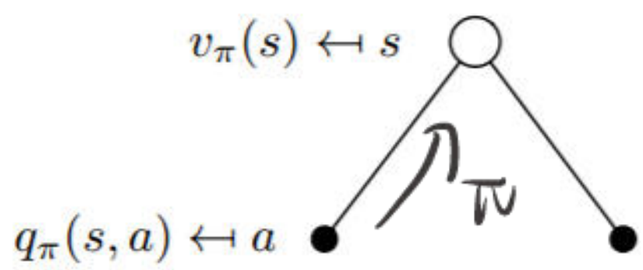
3、状态价值函数与动作价值函数的递推关系

状态价值函数 = Σ 某状态的动作价值 * 某状态执行该动作的概率

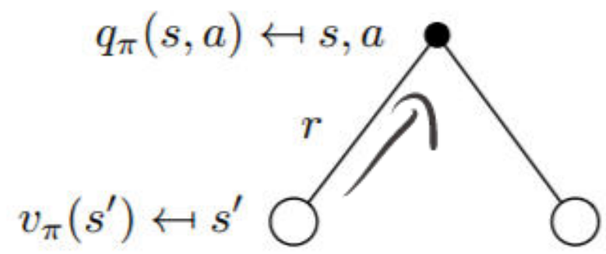
$$v_\pi(s) = \sum_{a \in A} \pi(a|s) * q_\pi(s,a)$$

动作价值函数 = 即时奖励 + γΣ 可能出现的下一状态的概率 * 其状态价值

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * v_\pi(s')$$



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

递推关系：

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) * \underbrace{\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * v_\pi(s')\right)}_{q_\pi(s,a)}$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * \underbrace{\left(\sum_{a' \in A} \pi(a'|s') * q_\pi(s',a')\right)}_{v_\pi(s')}$$

4. 最优价值函数

(1) 强化学习的最终目的 → 最优策略 π_*

(2) 如何比较策略优劣 < 最大状态价值函数 $v_*(s) = \max_{\pi} v_\pi(s)$

最大动作价值函数 $q_*(s,a) = \max_{\pi} q_\pi(s,a)$

当执行动作a可得到最大的价值函数值，则执行该动作a的概率为1

(此处是关于最大动作价值) 最优策略 $\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} q_*(s,a) \\ 0 & \text{else} \end{cases}$

状态价值函数 = Σ 某状态的动作价值 * 某状态执行该动作的概率

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) * q_\pi(s,a)$$

动作价值函数 = 即时奖励 + γ Σ 可能出现的下一状态的概率 * 其状态价值

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * v_\pi(s')$$

$$v_*(s) = \sum_{a \in A} \pi(a|s) * q_\pi(s,a)$$

$\pi(a|s) = 1$ or 0

$$= \max_{a \in A} q_*(s,a) + xxx =$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * v_\pi(s')$$

$$= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * v_*(s')$$

贝尔曼期望方程 = $v_\pi(s) = \sum_{a \in A} \pi(a|s) * \underbrace{\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * v_\pi(s')\right)}_{q_\pi(s,a)}$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a * \underbrace{\left(\sum_{a' \in A} \pi(a'|s') * q_\pi(s',a')\right)}_{v_\pi(s')}$$

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

三. 动态规划DP求解

1. 动态规划 & 强化学习 的联系

(1) 动态规划关键点：<
- 问题最优解由若干小问题的最优解构成
- 找到子问题状态间的递推关系

强化学习满足

(2) 强化学习所要解决问题

预测：<
- 给定要素：状态集S，动作集A，状态转化模型$P^a_{ss'}$，即时奖励R，衰减因子γ，给定策略π
- 求解该策略的状态价值函数$V_\pi(s)$

控制：<
- 给定要素：状态集S，动作集A，状态转化模型$P^a_{ss'}$，即时奖励R，衰减因子γ，
- 求解最优状态价值函数$V^*(s)$，最优策略$\pi^*(a|s)$

结合起来：
$$\boxed{V_\pi(s) = \sum_{a\in A} \pi(a|s) * \underbrace{\left(R^a_s + \gamma \sum_{s'\in S} P^a_{ss'} * V_\pi(s')\right)}_{q_\pi(s,a)}}$$

利用贝尔曼方程，可定义子问题递推

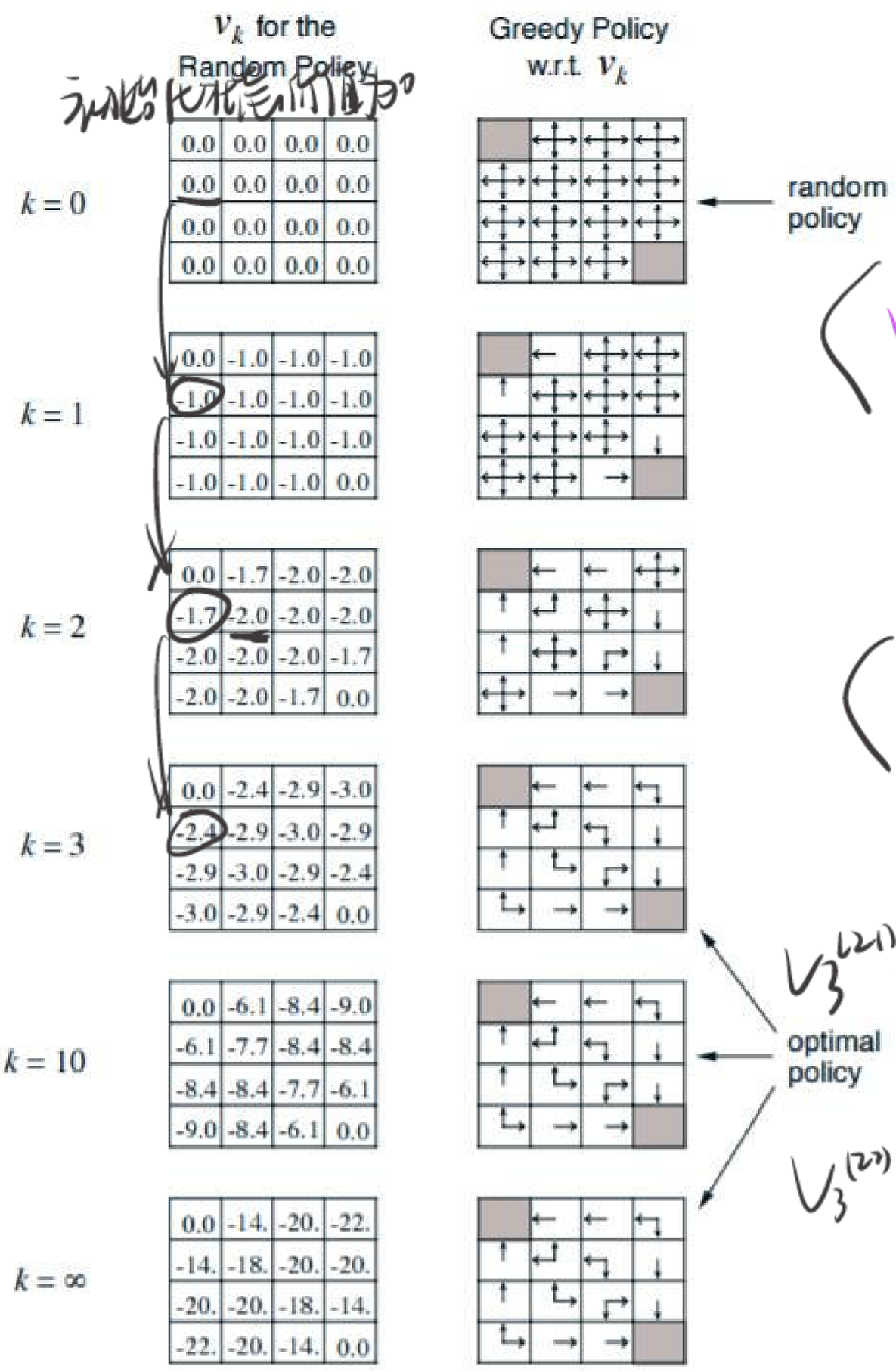
2. 策略评估求解预测问题

(1) 策略评估：从任一状态价值函数开始 $\xrightarrow{\text{基于贝尔曼}}$ 更新函数直到收敛 (更新 V_*)

贝尔曼方程 $V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \cdot \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s')\right)$ < 策略 π 已给定 / 加上迭代轮数下标

(2) 实例：Grid World

Note
- 终止状态：左上、右下，价值固定为 0，$V^{(1)} = V^{(14)} = 0$
- 只能上下左右走，若在边界格子外，直接回到该边界格子
- 每次移动下一格都固定，因此状态转化矩阵 $P=1$，$R=-1$，$\gamma=1$，$\pi = \frac{1}{4}$，$V(s)=0$



初始化价值为 0

$R=1$ $\gamma=1$ $P=1$

$V_1^{(21)} = \frac{1}{4} \times (-1 + 1 \times 1 \times 0) \times 4 = -1$

$V_{k+1}(s)$ $V_k(s')$

$V_1^{(22)} = \frac{1}{4} \times (-1 + 1 \times 1 \times 0) \times 4 = -1$

$V_2^{(21)} = \frac{1}{4} \times (-1 + 1 \times 1 \times 0) + \frac{1}{4} \times (-1 + 1 \times 1 \times -1) \times 3 = -1.5$

$V_2^{(22)} = \frac{1}{4} \times (-1 + 1 \times 1 \times -1) \times 4 = -2$

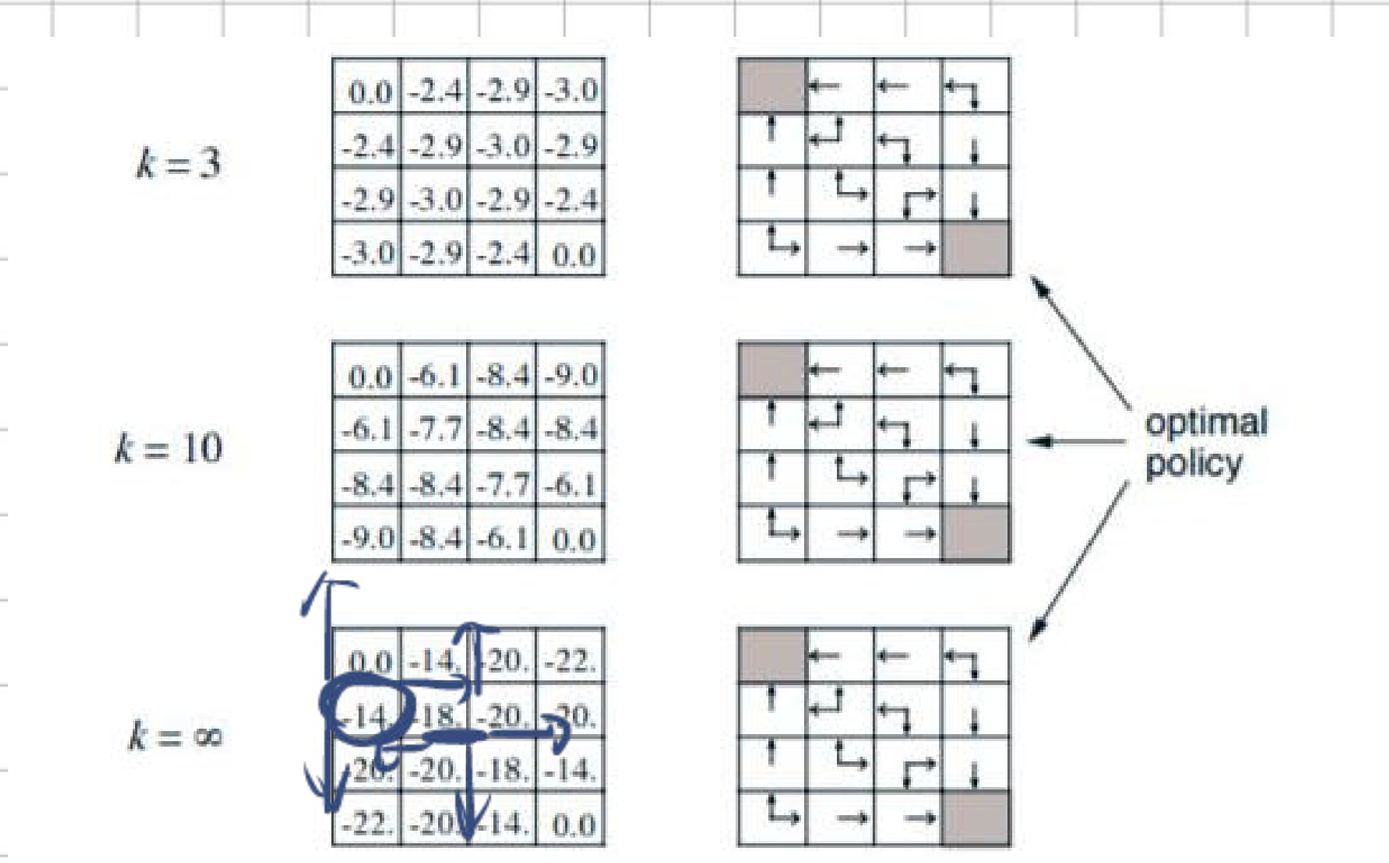
$V_3^{(21)} = \frac{1}{4} \times (-1 + 1 \times 1 \times 0) + \frac{1}{4} \times (-1 + 1 \times 1 \times -2) \times 2 + \frac{1}{4} \times (-1 + 1 \times 1 \times -1.5) = -2.4$

$V_3^{(22)} = \frac{1}{4} \times (-1 + 1 \times 1 \times -1.7) \times 2 + \frac{1}{4} \times (-1 + 1 \times 1 \times -2) \times 2 = -2.85$

3. 策略迭代求解强化问题

(1) 策略迭代：基于在给定策略下估计得到的状态价值，改进动作策略（更新π*）

(2) 贪婪法：某状态下选择一行为可达到后续所有可能状态中，状态价值最大的状态

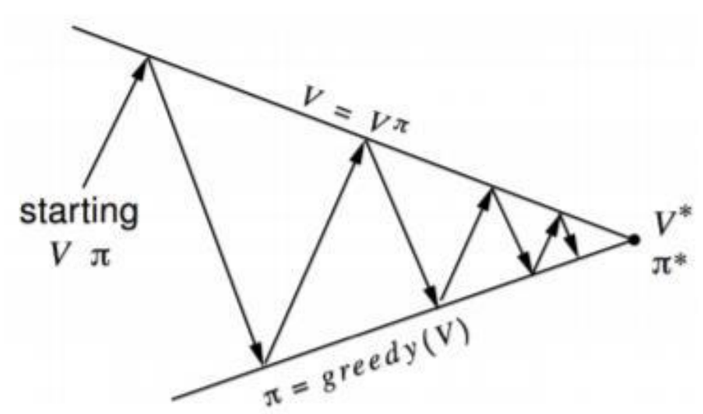


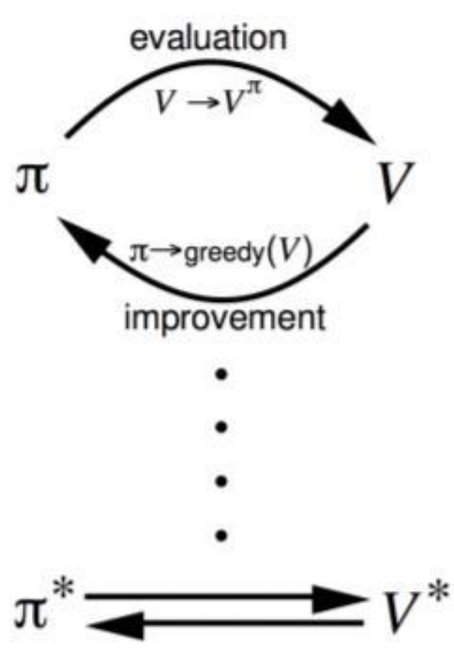
k=∞ 得到最佳状态价值后

贪婪法

第二行第一格到周围价值：0, -18, -20 → 向V=0移动 ↑

第二行第二格到周围价值：-14, -14, -20, -20 → 向V=-14移动 ↑





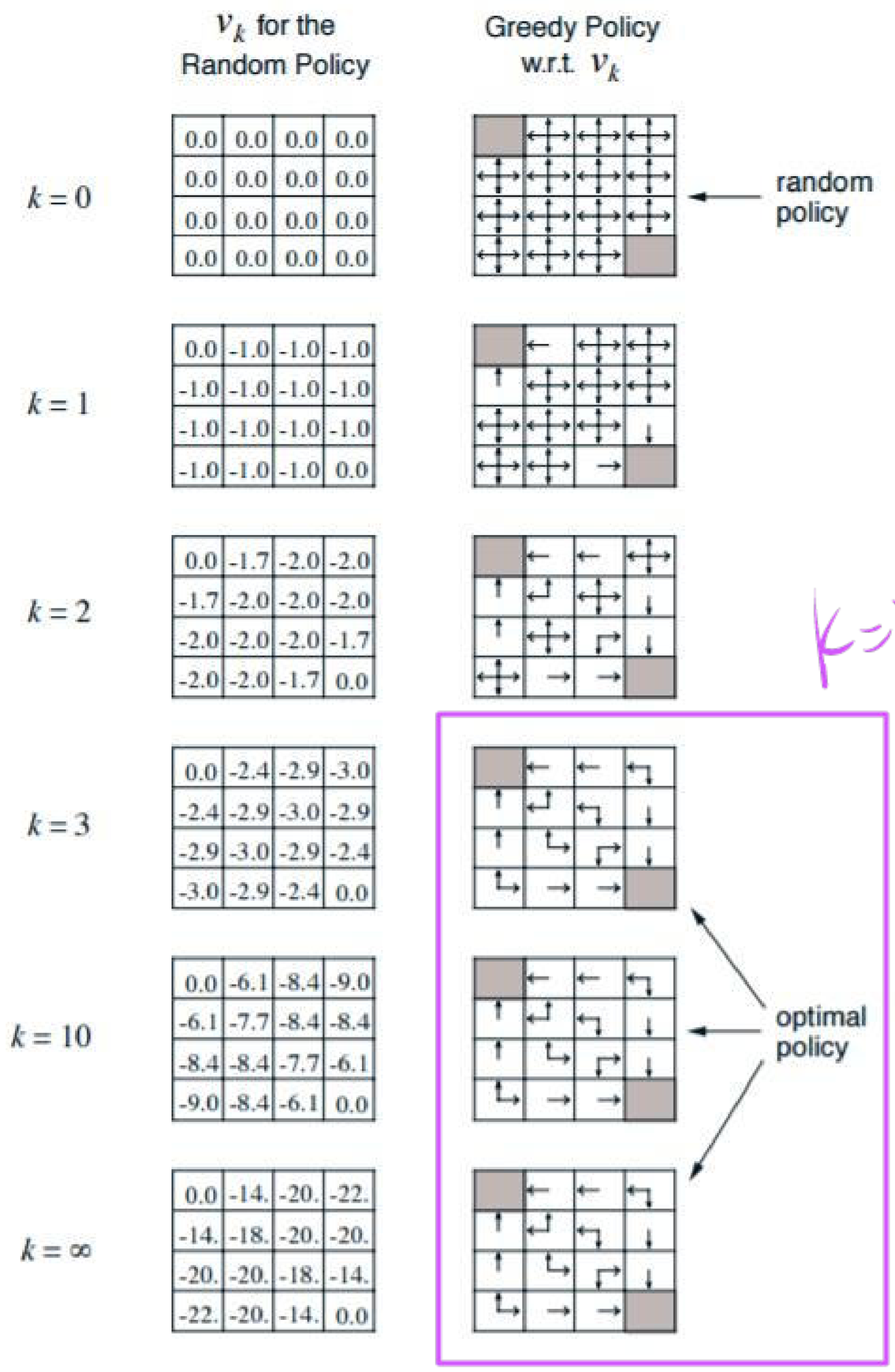
Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

使用当前策略 评估(迭代法) 最佳所有状态价值 V* (策略评估)

根据状态价值V* 用一定方法(如贪婪法)更新策略π* (策略迭代)

不动值迭代解决制约问题



k=3时，已经得到最大的动作策略，不用一直迭代到收敛价值收敛

↓

随机策略价值的变化，但策略不变

k=2时，不等到最大化价值

第二行第一格周围价值：0，-2，-2 → 向v=0移动 ↑

第二行第二格周围价值：-14，-14，-20，-20 → 向v=-14移动 ←↑

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \cdot \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \cdot v_k(s')\right)$$

$\pi(a|s) = 1$

$$v_{k+1}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \cdot v_k(s')\right)$$

每次更新值时贪婪地选择最大策略对应的动作价值，加快收敛

5. 异步动态规划

同步：每轮迭代计算并保存所有状态价值

下一轮使用这些价值计算新一轮的状态价值

异步：每轮迭代并不更新所有状态价值

原位动态规划：不额外保存上一轮的状态价值 减少保存的状态价值数量，节省内存

即时计算即时更新 收敛速度较慢

优先级扫描：对每个状态优先级分级 加快收敛速度

优先级越高，状态价值越先被更新 要维护一个优先级队列

贝尔曼误差 = |新状态价值 − 前状态价值| 用于评估优先级

实时：使用个体与环境交互产生的实时经历来更新状态价值

经常访问的状态更新频率高

收敛速度可能受限

蒙特卡洛规划：全宽度的回溯机制，来更新状态价值

每次回溯到该状态的所有可能后续状态，并用贝尔曼方程更新

适用于状态数较少的强化学习问题

规模很大时，即使回溯贝尔曼方程也很耗时

四. 蒙特卡罗ML求解

1. 不基于模型的强化学习

- 动态规划：状态转化模型P始终已知 → 基于模型的强化学习
  - MDP 马尔科夫决策过程
- 强化学习：无法事先得到P → 不基于模型的强化学习

预测：
- 给定5要素：状态集S，动作集A，~~状态转化模型$P_{ss'}^{a}$~~，即时奖励R，衰减因子γ，给定策略π
- 求解该策略的状态价值函数 $V_\pi(s)$

控制：
- 给定5要素：状态集S，动作集A，~~状态转化模型$P_{ss'}^{a}$~~，即时奖励R，衰减因子γ，探索率ε
- 求解最优状态价值函数 $V_*(s)$，最优策略 $\pi_*(a|s)$

$q_*(s,a)$

2. 蒙特卡罗求解特点

- 通过若干经历完整的状态序列来估计状态真实价值
- 到达终点：
  - 下棋有输赢
  - 驾车到终点为止
- 不依赖状态转化模型P

从经验的完整序列学习，经验越多，学习效果越好

3. 蒙特卡罗ML求解预测问题

给定策略π，有历史上的一个状态。

$S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_t, A_t, R_{t+1}, S_{T-1}, A_{T-1}, R_T, S_T, A_T, R_{T+1}$

马尔可夫MDP — $V_\pi(s) = E(G_t \mid S_t = s)$

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

蒙特卡罗ML — $V_\pi(s) = \text{average}(G_t)$ s.t. $S_t = s$

$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T$ T个

$R_t = \gamma^0 R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{x-1} R_T$ $T = t+x$

$x = T-t$

$x-1 = T-t-1$

优化点：

1. 同一个状态可能在完整的状态序列中重复出现

首次访问：仅将第一次出现的收获值纳入收获平均值

每次访问：每次出现的该状态的收获值都纳入收获平均值

计算量更大

平均的数量更多

2. 累进更新平均值：$V_\pi(s) = \text{average}(G_t)$ s.t. $S_t = s$

保存所有该状态的收获值计算最后平均 浪费很多时间

迭代计算收获均值，保存上一轮迭代得到的收获均值与次数

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right)$$

$$= \frac{1}{k}(x_k + (k-1)\mu_{k-1})$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

不用保存所有，只需现值更新

算法内存大小固定

$\rightarrow N(S_t) = N(S_t) + 1$ 更新迭代次数

$\rightarrow V(S_t) = V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$ 更新平均收益值

非静态数据分布式迭代，无法精确计算当前次数 $N(S_t)$（并且） 用系数 α 代替

$\rightarrow V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$

$\rightarrow Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$

4. 蒙特卡罗ML求解控制问题

控制问题：蒙特卡罗ML ≈ DP价值迭代　回合更新：一个序列一个状态

策略评估：贝尔曼方程评估每一轮状态价值 V_π

策略迭代：贪婪法迭代该轮策略 π_*

不同
- 策略评估：所有后续状态 → 下个状态
- 优化目标：状态价值 V_* → 动作价值 q_*
- 更新策略：贪婪法 → ε-贪婪法 ← 设置较小的ε值，随迭代过程逐渐减小 ← 前期：鼓励探索；后期：贪婪为主

共m种动作
- 1种 → e.t. 价值最大
- m-1种 → else

- 1-ε 贪婪地选择当前最大动作价值
- ε 随机从另外m-1动作中选择

$1-\frac{(m-1)}{m}\varepsilon$

$$\pi(a|s)=\begin{cases}\varepsilon/m+1-\varepsilon & \text{if } a^*=\arg\max_{a\in A} Q(s,a)\\ \varepsilon/m & \text{else}\end{cases}$$

$\varepsilon=0.2,\ m=5$

(1)
$$\pi(a|s)=\begin{cases}\varepsilon\cdot\frac{1}{m}+1-\varepsilon=0.04+1-0.2=0.84 & \text{反 1-xx}\\ \varepsilon\cdot\frac{1}{m}=0.2\times0.2=0.04\ (0.04\times4=0.16) & \text{xx}\end{cases}$$

(2)
$$\pi(a|s)=\begin{cases}(1-\varepsilon)\cdot\frac{1}{m} & \text{vv}\\ q\cdot\frac{1}{m}=0.04 & \text{xx}\end{cases}$$

$$\Delta=\varepsilon\cdot\frac{1}{m}+1-\varepsilon-\left(\frac{1}{m}-\frac{\varepsilon}{m}\right)$$
$$=\frac{\varepsilon}{m}+1-\varepsilon-\frac{1}{m}+\frac{\varepsilon}{m}$$
$$=\frac{m-1}{m}-\left(\varepsilon-\frac{\varepsilon}{m}-\frac{\varepsilon}{m}\right)$$
$$=\underbrace{\frac{m-1}{m}(1-\varepsilon)}_{\times\checkmark}+\underbrace{\frac{\varepsilon}{m}}_{\checkmark\times}$$

5. 蒙特卡罗算法流程

Note:
- 在线版本 on-policy (离线版本后续讲述)
- every-visit 每个重复状态价值的平均值

Input: $S, A, R, \gamma, \pi, \varepsilon$

Output: q_*, π_*

1. 初始化 $Q(s,a)=0$ $N(s,a)=0$ $k=0$

   随机初始化 π

2. $k=k+1$，基于策略 π，进行第 k 次蒙特卡罗采样了状态序列：

   $S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_t, A_t, R_{t+1}, \dots, S_{T-1}, A_{T-1}, R_T, S_T, A_T, R_{T+1}$

3. 对于每个状态行为 (S_t, A_t)，计算收益 G_t，更新计数 $N(s,a)$ 及行为价值函数 $Q(s,a)$

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

   $$N(S_t, A_t) = N(S_t, A_t) + 1$$

   $$Q(S_t, A_t) = Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\left(G_t - Q(S_t, A_t)\right)$$

   $$= Q(S_t, A_t) + \alpha\left(G_t - Q(S_t, A_t)\right)$$

   优：不事先知道状态转化模型 P

   不回溯所有的后续状态

   可用于海量数据复杂模型

4. 基于新计算的动作价值，更新当前 ε-贪婪策略 $\pi(S_t)$

   令 $\varepsilon = \frac{1}{k}$

   $$\pi(S_t, A_t) = \begin{cases} \varepsilon/m + 1 - \varepsilon & \text{if } a = \arg\max_{a \in A} Q(s,a) \\ \varepsilon/m & \text{else} \end{cases}$$

   缺：每次采样要完整的状态序列

   若没有完整的状态序列，则很难求解

5. 当 $Q(s,a)$ 收敛，则对应的所有 $Q(s,a)$ 为最优动作价值函数 q_*

   对应的策略 $\pi(a|s)$ 为最优策略 π_*

否则继续回 2

五. 时序差分TD求解

1. 时序差分TD简介

时序差分 ~ 蒙特卡罗 < 不基于模型的强化学习 但没有完整的序列

(二) 蒙特卡罗ML求解

1. 不基于模型的强化学习

动作规划：状态转化模型P 奖励函数R → 基于模型的强化学习

MDP 可以解决策略问题

纯粹强化学习无法事先得到P → 不基于模型的强化学习

预测：给定5要素：状态集S，动作集A，~~状态转化模型$P^a_{ss'}$~~，即时奖励R，衰减因子γ，给定策略π

求解该策略的状态价值函数 $V_\pi(s)$

控制：给定5要素：状态集S，动作集A，~~状态转化模型$P^a_{ss'}$~~，即时奖励R，衰减因子γ，探索率ε

求解最优状态价值函数 ~~$V_*(s)$~~ $q_*(s,a)$，最优策略 $\pi_*(a|s)$

TD中主要应用 $V_*(s,a)$

同（不基于模型）

蒙特卡罗ML - $V_\pi(s) = \text{average}(G_t)$ s.t. $S_t = s$

$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ T个

$R_t = \gamma^0 R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{x-1} R_T$　$T = t+x$，$x = T-t$，$x-1 = T-t-1$

不同（不完整序列）

贝尔曼方程：

$$
\begin{aligned}
V_\pi(s) &= E(G_t | S_t = s) \\
&= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s) \\
&= E(R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s) \\
&= E(R_{t+1} + \gamma G_{t+1} | S_t = s) \\
&= E(R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s)
\end{aligned}
$$

$R_{t+1} + \gamma V(S_{t+1})$ 近似代替 G_t

⇒ TD用估值 $G_t = R_{t+1} + \gamma V(S_{t+1})$

TD误差 $\delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

（只需所经历的状态对应的奖励）

引导

2 时序差分TD求解预测问题

预测问题：时序差分TD ≈ 蒙特卡罗MC

不同 $\begin{cases} G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \rightarrow \approx R_{t+1} + \gamma V(S_{t+1}) \\ V_\pi(s) = \text{average}(G_t) \text{ s.t. } S_t = s \end{cases}$

$= V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t)) \rightarrow V(S_t) + \alpha(G_t - V(S_t))$

无完整序列，$N(S_t)$不存在，用α代替　　$Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$

实例：比较蒙特卡罗和时序差分的不同

假设某强化学习问题有状态A、B，模型P未知，衰减因子γ=1　不涉及策略和行动　只涉及状态转化和奖励

8个完整的状态序列：

① A,0, B,0　② B,1　③ B,1　④ B,1　⑤ B,1　⑥ B,1　⑦ B,1　⑧ B,0

蒙特卡罗：状态序列中所有状态收益取平均 $(\text{average}(G(A/B)))$ $\rightarrow G(A/B) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

状态A：仅第一个序列中包含状态A，其价值仅可通过第一个序列计算

$V(A) = \text{average}(G(A)) = G(A) = \underset{R_{t+1}}{R_A} + \gamma \underset{R_{t+2}}{R_B} = 0 + 1 \cdot 0 = 0$

状态B：要对所有8个序列中的收获值取平均

$V(B) = \text{average}(G(B)) = \frac{1}{8}\sum G(B) = \frac{1}{8}\sum \underset{R_{t+1}}{R_B} \underset{R_{t+2}=0}{} = \frac{1}{8}(0+1+1+1+1+1+1+0) = \frac{6}{8}$

时序差分：状态序列中后续状态收益估计 $(\text{average}(G(A/B)))$ $\rightarrow G(A/B) = R_{t+1} + \gamma V(S_{t+1})$

A后续为B，B无后续

状态B：每个序列中，状态B无后续状态

$V(B) = \text{average}(G(B)) = \frac{1}{8}\sum G(B) = \frac{1}{8}\sum \underset{R_{t+1}}{R_B} \underset{V(S_{t+1})=0}{} = \frac{1}{8}(0+1+1+1+1+1+1+0) = \frac{6}{8}$

状态A：仅第一个序列中包含状态A，其价值仅可通过第一个序列计算

$V(A) = \text{average}(G(A)) = G_A = \underset{R_{t+1}}{R_A} + \gamma \underset{V(S_{t+1})=V(B)}{V(B)} = 0 + 1 \cdot \frac{6}{8} = \frac{6}{8}$

时序差分	蒙特卡罗
得知结果前 没有结果时 可学习 持续环境中 ⇒ 更快速反馈 更新价值	得到最终结果才学习
TD目标值 ≈ 即时奖励 + 下一状态预估价值 G_t R_{t+1} $\gamma V(S_{t+1})$ → (有偏估计) ⇒ 代替完整轨迹到结束的实际收益	即时奖励 + 实际价值 R_{t+1} $\gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T$ → (无偏估计) ⇒ 无偏估计更精确
方差更低 初始时高敏感 ⇒ 更高效	

3. n步时序差分

时序差分：向前一步近似 G_t

$$G_t \approx R_{t+1} + \gamma V(S_{t+1})$$

n步时序差分：向前2/n步近似 G_t

$$G_t^{(2)} \approx R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

$$G_t^{(n)} \approx R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^n V(S_{t+n})$$

当 $n \to +\infty$，状态序列→完整，n步时序差分→蒙特卡罗

如何选择n？

4. TD(λ)

n步时序差分：如何选择n？ < 不增加计算复杂度 / 综合考虑更多步数

⇓

引入参数λ：TD(λ)：

λ-收获：n=1~+∞ 所有步收获乘以权重后求和

$$G_t^{\lambda} = \sum_{n=1}^{\infty} G_t^{(n)} * (1-\lambda)\lambda^{n-1}$$

$$= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

$$V(S_t) = V(S_t) + \alpha(G_t^{\lambda} - V(S_t))$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t^{\lambda} - Q(S_t, A_t))$$

为何确定为 $(1-\lambda)\lambda^{n-1}$？

步数n↑ 权重weight呈几何级数↓

意味着性能不稳定，权重太小

当在T时刻到达终止状态时，未分配的权重全给予终止状态的实际收获值

使得完整状态序列中所有n步的收获权重相加为1

$W = (1-\lambda)\lambda^{n-1}$

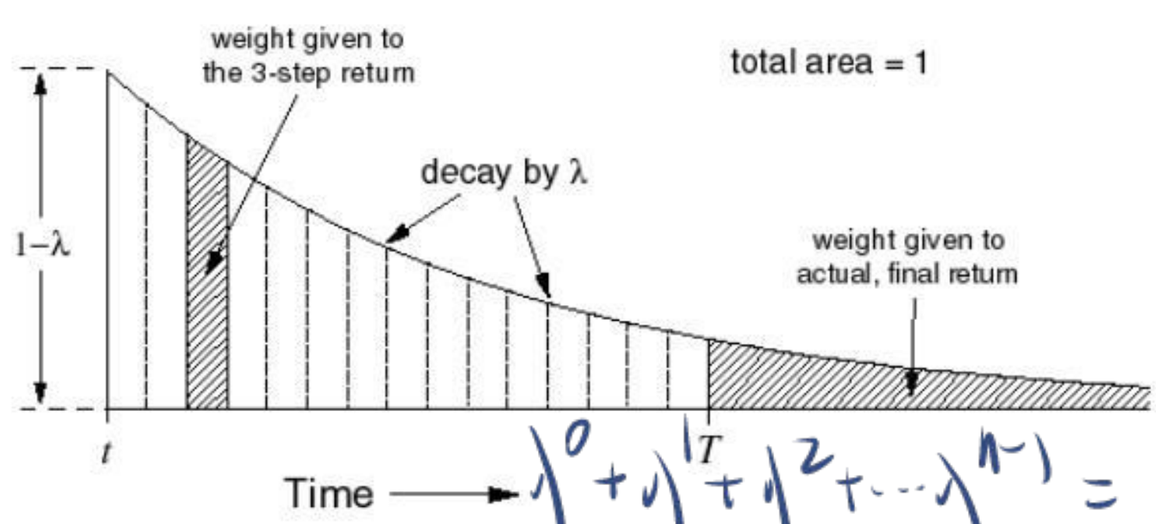
$n=1 \quad w_1 = (1-\lambda)\lambda^0$

$n=2 \quad w_2 = (1-\lambda)\lambda^1$

$n=3 \quad w_3 = (1-\lambda)\lambda^2$

⋮

$n=n \quad w_n = (1-\lambda)\lambda^{n-1}$



$$\lambda^0 + \lambda^1 + \lambda^2 + \cdots \lambda^{n-1} = \frac{1-\lambda^{n-1}\cdot\lambda}{1-\lambda} = \frac{1-\lambda^n}{1-\lambda}$$

$$\int w\,d\lambda = \sum_{n=1}^{\infty} w_n = (1-\lambda)\cdot\frac{1-\lambda^n}{1-\lambda} = 1-\lambda^n \approx 1$$

前向TD(λ)：V(s)由G_t得到，G_t由后续状态价值得到（后 $\xrightarrow{得}$ 前）

更新一个状态价值，要知道所有后续状态的价值

必须要经历完整的状态序列，获得包括此状态的每个状态的即时奖励

$\lambda=0$，$n=1$时 初始权重 $(1-\lambda)\cdot\lambda^0=1$

→只有一步 → 普通的时序差分

$\lambda=1$，$n=\infty$时 每一步权重 $(1-\lambda)\cdot\lambda^{n-1}=0$

→每一步权重都很小，要走完到所有后续状态，→蒙特卡罗

反向TD(λ)：分析状态对后续状态的影响（前 $\xrightarrow{影响}$ 后）

老鼠接受了3次响铃，1次亮灯后，遭到雷击

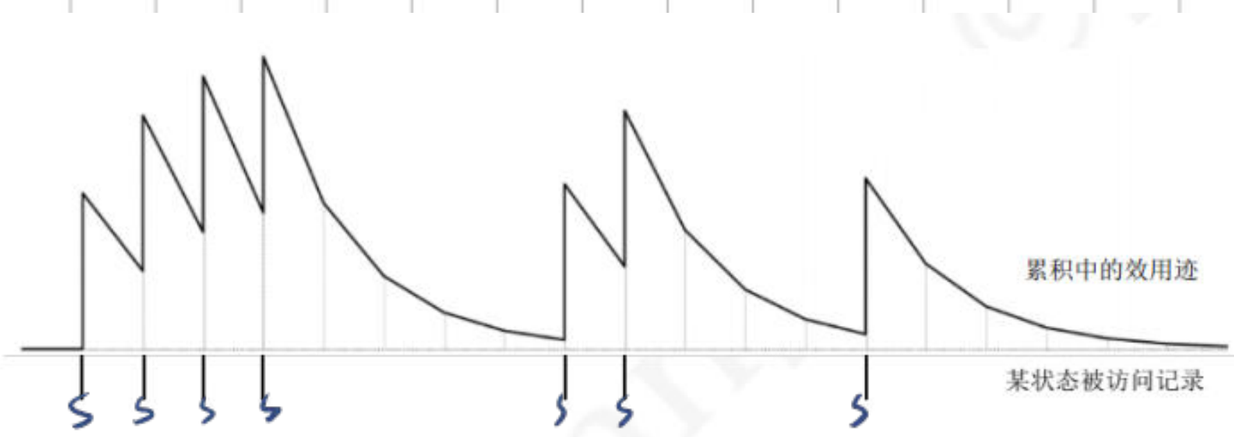
分析遭电原因：
- 3次响铃：频率启发式
- 1次亮灯，就近启发式

每个状态：效用E（该状态对后续状态的影响）

所有状态：效用值 $E_t(s)$

$E_0(s)=0$

$$E_t(s)=\gamma\lambda E_{t-1}(s)+\underbrace{1(s_t=s)}_{s_t=s\text{时取1否则取0}}=\begin{cases}0 & t<k\\ (\gamma\lambda)^{t-k} & t\geq k\end{cases}\quad \text{s.t. } \lambda,\gamma\in[0,1],\ s \text{ is visited once at time } k$$



当某一状态连续出现时,E值会在一定衰减的基础上有一个单位数值的提高,此时认为该状态将对后续状态的影响较大

如果该状态很长时间没有经历,那么该状态的E值将逐渐趋向于0,表明该状态对于较远的后续状态价值的影响越来越少

若状态s出现，对应的效用迹的值就加1；若未发生，则按照某个衰减因子进行衰减

TD(λ)更新价值函数：

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \quad (\delta_t = G_t - V(S_t))$$

$$V(S_t) = V(S_t) + \alpha \delta E_t(s)$$

$$\begin{cases} \lambda = 0 & E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s) = 1 \\ \lambda = 1 & \\ \lambda \in (0,1) & \end{cases}$$

仅后向一个状态的误差 $V(S_t) = V(S_t) + \alpha\delta$ 普通TD

考虑到后向所有状态的影响 MC

不完全MC

$$G_t^\lambda - V(S_t) = -V(S_t) + (1-\lambda)\lambda^0 (R_{t+1} + \gamma V(S_{t+1}))$$
$$+ (1-\lambda)\lambda^1 (R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}))$$
$$+ (1-\lambda)\lambda^2 (R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3}))$$
$$+ \cdots$$
$$= -V(S_t) + (\gamma\lambda)^0 (R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1}))$$
$$+ (\gamma\lambda)^1 (R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2}))$$
$$+ (\gamma\lambda)^2 (R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3}))$$
$$+ \cdots$$

$$= (1-\lambda)(R_{t+1} + \gamma V(S_{t+1}))$$
$$= R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1}) - \lambda R_{t+1}$$
$$= (\gamma\lambda)^0 (R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1}))$$

$$= \lambda R_{t+1} + \lambda\gamma R_{t+2} + \lambda\gamma^2 V(S_{t+2}) - \lambda^2 R_{t+1} - \lambda^2\gamma R_{t+2} - \lambda^2\gamma^2 V(S_{t+2})$$
$$= (\gamma\lambda)^1 (R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2}) - \lambda R_{t+2})$$
$$= (\gamma\lambda)^1 (R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2}))$$

$$= -V(S_t) + (\gamma\lambda)^0 (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$
$$+ (\gamma\lambda)^1 (R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1}))$$
$$+ (\gamma\lambda)^2 (R_{t+3} + \gamma V(S_{t+3}) - V(S_{t+2}))$$
$$+ \cdots$$

$$\gamma\lambda V(S_{t+1}) = V(S_t)$$
$$\gamma\lambda V(S_{t+2}) = V(S_{t+1})$$
$$\gamma\lambda V(S_{t+3}) = V(S_{t+2})$$

$$= -V(S_t) + \underbrace{\delta_t + (\gamma\lambda)^1 \delta_{t+1} + (\gamma\lambda)^2 \delta_{t+2} + \cdots}_{G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots}$$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

5、时序差分TD求解控制问题

(普通)

DP < π all / V non. 贪

MC V non ε-贪

< TD V non ε-贪

控制问题：普通时序差分二家族长多

- 同：ε-贪婪近似值迭代
- 不同：收敛的计算方式不同（估计Qπ.贝曼）

在线 on-policy：仅一个策略（ε-贪婪）

SARSA

离线 off-policy：一般有两个策略（ε-贪婪 → 生成新的动作；贪婪 → 更新价值函数）

Q-learning

六、时序差分在线控制 SARSA

1. SARSA算法引入

控制 < 给定5要素：S, A, R, γ, ε
求解：q_*, π_*

预测：不基于模型，无需P / 一个后续状态

控制之基于字字 { 价值迭代 $v_*/q_* \xrightarrow{策略} \pi_*$
在线控制：仅一个策略用于更新 v_*/q_*，更新新动作

ε-贪婪法：设置较小ε值，前期探索，后期贪婪
$\pi(a|s) = \begin{cases} \varepsilon/m + 1-\varepsilon & 当 a^* = \arg\max_{a \in A} Q(s,a) \\ \varepsilon/m & else \end{cases}$

DP < SAPRγπ → v_π / SAPRγ → v_*, π_* < All 贪 < 策略 价值

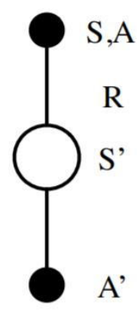
ML < SARγπ → v_π / SARγε → q_*, π_* < 1个经验轨迹 / ε-贪婪价值

TD < SARγπ → v_π / SARγε → q_*, π_* <
- 普通：1个后续轨迹
- n步：n个后续轨迹（MC）
- TD(λ)：λ∈[0,1]

在线：SARSA
SAPSAγ(λ)
离线：Q-Learning

2. SARSA算法概述

状态S：State
动作A：Action
奖励R：Reward



基于ε-贪婪法在当前状态S选择动作A
系统转到新的状态S'，同时返回即时奖励R
基于状态S'选择新动作A'，此时并不执行动作A'，仅用于更新价值函数（仍使用ε-贪婪来选择）
更新价值函数 $Q(S,A)=Q(S,A)+\alpha(q_t-Q(S,A))$

$$=Q(S,A)+\alpha(R+\gamma Q(S',A')-Q(S,A))$$

α：迭代步长
可调整学习率，保证收敛与稳定性

γ：衰减因子

3. SARSA算法流程

Input：迭代轮数T，S，A，α，γ，ε
Output：所有状态价值V，动作价值Q

(1) 随机初始化V，Q（终止状态Q=0）

(2) for t from 1 to T：

a) 初始化当前状态S为第一个状态

在状态S $\xrightarrow{\text{ε-贪婪法}}$ 选择动作A

b) 执行动作A → 新状态S'，奖励R

c) 在新状态S' $\xrightarrow{\text{ε-贪婪法}}$ 选择动作A'

d) 更新价值函数 $Q(S,A)=Q(S,A)+\alpha(R+\gamma Q(S',A')-Q(S,A))$

e) S=S'，A=A'

f) 若S'为终止状态，当前轮迭代完成，否则转回b

4. Windy GridWorld

5. SARSA(λ)

TD(λ): 前向、反向等价

SARSA(λ): 反向可有效地找出对当前数据有贡献的状态-动作对

→ SARSA(λ)默认基于反向进行价值函数迭代

$G_t \approx R_{t+1} + \gamma V(S_{t+1})$

$V(S_t) + \alpha(G_t - V(S_t))$

$Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$

$G_t^{(1)} \approx R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$

$G_t^{(n)} \approx R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^n V(S_{t+n})$

$G_t^{\lambda} = \sum_{n=1}^{\infty} G_t^{(n)} * (1-\lambda)\lambda^{n-1}$

$= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$

$V(S_t) = V(S_t) + \alpha(G_t^{\lambda} - V(S_t))$

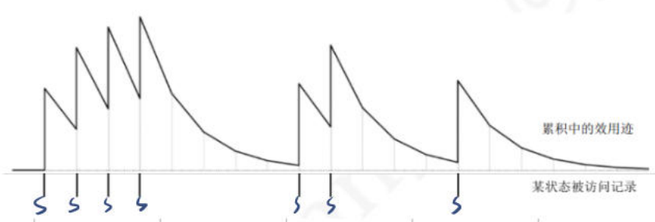
$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t^{\lambda} - Q(S_t, A_t))$

每个状态: 效用 E (该状态对引起状态的影响)

所有状态: 效用值 E

$E_0(s) = 0$

$E_t(s) = \gamma\lambda E_{t-1}(s) + \underbrace{1(S_t = s)}_{S_t=s\text{时取1否则取0}} = \begin{cases} 0 & t<k \\ (\gamma\lambda)^{t-k} & t \geq k \end{cases}$ s.t. $\lambda, \gamma \in [0,1]$, s is visited once at time k



当某一状态连续出现时,E值会在一定衰减的基础上有一个单位数值的提高,此时认为该状态将对后续状态的影响较大

如果该状态很长时间没有经历,那么该状态的E值将逐渐趋向于0,表明该状态对于较远的后续状态价值的影响越来越少

若状态s出现，对应的效用迹的值就加1；若未发生，则按照某个衰减因子进行衰减

TD(λ)更新价值函数: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ $(\delta_t = G_t - V(S_t))$

$V(S_t) = V(S_t) + \alpha \delta E_t(s)$

$\Rightarrow \delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$

$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \delta_t E_t(s, A)$

Input: 迭代参数 T, S, A, α, γ, ε, 多步参数 λ∈[0,1]

Output: 所有状态价值V, 动作价值Q

(1) 随机初始化 V, Q (终止状态 Q=0)

(2) for i from 1 to T:

a) 初始化当前状态S为所到达的第1个状态

在状态S $\xrightarrow{\varepsilon\text{-贪婪法}}$ 选择动作A, 初始化 E=0

b) 执行动作A → 新状态S', 奖励R

c) 在新状态S' $\xrightarrow{\varepsilon\text{-贪婪法}}$ 选择动作A'

d) 更新价值函数 $\boxed{Q(s,A) = Q(s,A) + \alpha(R + \gamma Q(s',A') - Q(s,A))}$

e) S=S', A=A'

f) 若S'为终止状态, 当前轮迭代完成, 否则转回b

更新: $E(s,A) = E(s,A) + 1$

$\delta = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$

对当前所到所有出现的状态S下的所有动作a

更新: $Q(s,a) = Q(s,a) + \alpha\delta E(s,a)$

$E(s,a) = \gamma\lambda E(s,a)$

七、时序差分与离线控制 Q-learning

1. Q Learning 引入

控制 <
- 给定5要素：$S, A, R, \gamma, \varepsilon$
- 求解：q_*, π_*

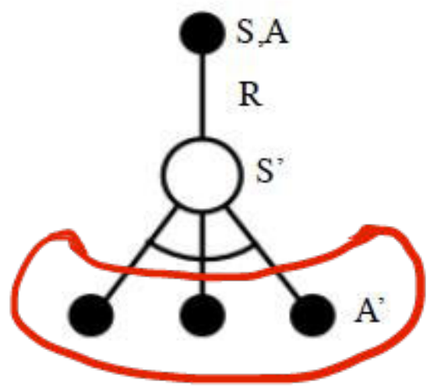
- 预测：不基于模型，无需P / 一个后继状态
- 控制：基于采样
  - 价值迭代 $v_*/q_* \xrightarrow{\text{贪婪化}} \pi_*$
  - ~~在线控制：仅一个策略用于更新 v_*/q_*，选择新动作~~
    - ε-贪婪法：设置较小ε值，前期探索，后期贪婪

$$\pi(a|s) = \begin{cases} \varepsilon/m + 1 - \varepsilon & \text{if } a_t = \arg\max_{a \in A} Q(s,a) \\ \varepsilon/m & \text{else} \end{cases}$$

  - 离线控制：两个策略用于更新 v_*/q_*（ε贪婪）
    - 选择新动作（ε-贪婪）

2. Q-Learning算法原理

- 状态S：State
- 动作A：Action
- 奖励R：Reward



choose $\max_{a \in A} Q(S',a)$

- 基于ε-贪婪法在当前状态S选择动作A
- 系统转到新的状态S'，同时返回即时奖励R
- 新状态S'选择新动作A'，此时并不执行动作A'，仅用于更新价值函数（仍使用ε-贪婪策略）
- 更新价值函数 $\cancel{Q(S,A)} = Q(S,A) + \alpha(q_t - Q(S,A))$

$$= \cancel{Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))}$$

学习速率

到新奖励，保证收敛 → 无收敛

衰减因子

新状态S'选择新动作A'，使用贪婪法

→ 选择使Q(S',a)最大的a作为A'来更新价值函数

更新价值函数 $Q(S,A) = Q(S,A) + \alpha(R + \gamma \max_{a \in A} Q(S',a) - Q(S,A))$

价值函数更新后，新的动作A'需基于状态S'，用ε-贪婪策略选择得到

SARSA → 新的动作A'直接作为下一阶段开始时的动作

3. Q-learning 算法流程

Input: 迭代次数 T, S, A, α, γ, ε

Output: 所有状态价值 V, 动作价值 Q

(1) 随机初始化 V, Q (终止状态 Q=0)

(2) for i from 1 to T:

a) 初始化当前状态 S, 为到达终止状态

在状态 S $\xrightarrow{\varepsilon\text{-贪婪法}}$ 选择动作 A

b) 执行动作 A → 新状态 S', 奖励 R

c) 在新状态 S' $\xrightarrow{\varepsilon\text{-贪婪法}}$ 选择动作 A' 贪婪法

$Q(S,A) = Q(S,A) + \alpha(R + \gamma \max_{a \in A} Q(S',a) - Q(S,A))$

d) ~~更新价值函数 $Q(S,A) = Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$~~

更新后再用 ε-贪婪策略选择 A'

e) S=S', ~~A=A'~~

f) 若 S' 为终止状态, 当前轮迭代完成, 否则转回 b 转回 c

4. Windy GridWorld

5. SARSA vs Q-learning

SARSA

ε-贪婪选择 A 并更新 Q

↳ 为了最优策略时强化探索

Q-learning

贪婪选择 A 并更新 Q

↳ 直接为了最优策略

⇒ 最优策略依赖于训练中产生的一系列数据

⇒ 学习到数据的分布 受到训练数据的影响大 甚至影响 Q 收敛

⇒ 为保证收敛, 需制定策略, 使 ε-贪婪中的 ε 逐渐减小

⇒ 收敛过程中鼓励探索, 学习过程比较平滑, 不会过于过于激进 (Q-learning 最优路径, Cliff Walk)

适合于在线生产

适合于模拟环境

八. 价值函数的近似表示 & Deep Q-Learning

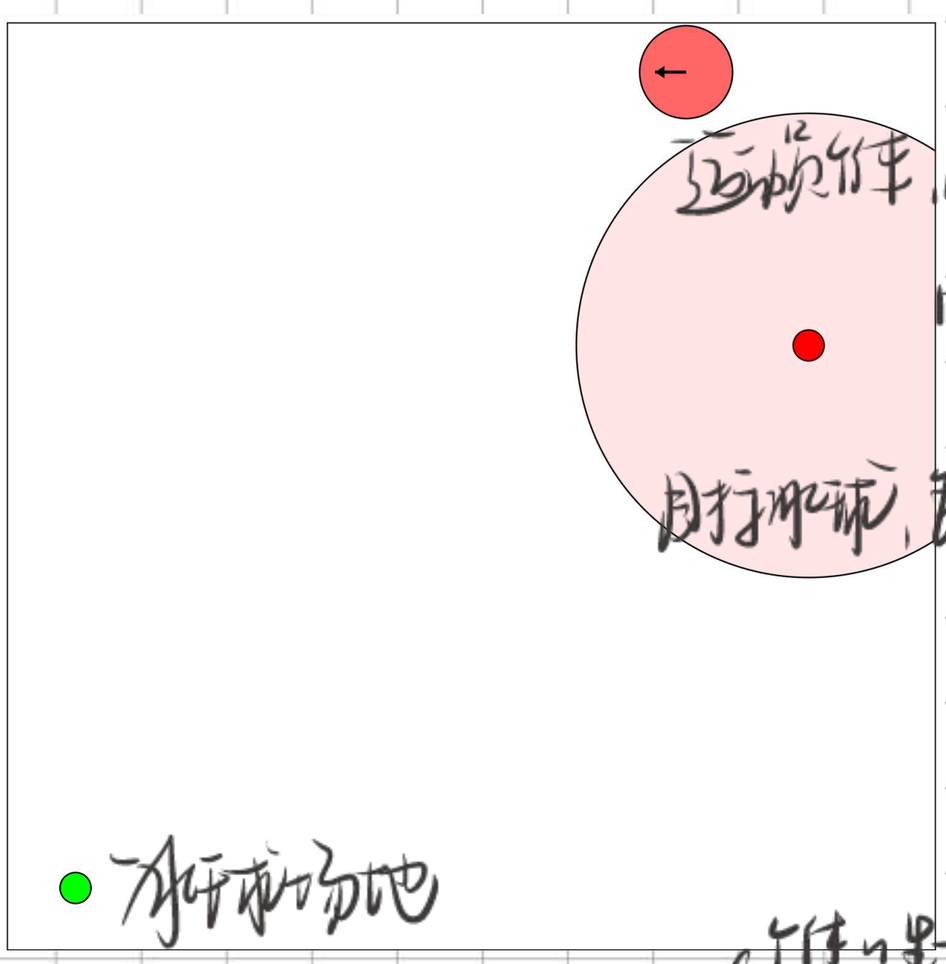
1. 为何需要价值函数的近似表示?

使用状态空间的有限个状态集合S
- 动态规划 DP
- 蒙特卡罗 MC
- 时序差分 TD

问题规模较小，比较容易求解

若状态连续 or 动作离散化后集合比较大

若使用传统方法 比如 Q Learning，根本无法在内存中保存这么大的Q表



环境在每个时间步内告诉
- 个体的坐标 (水平, 垂直)
- 速度的分量 (水平, 垂直)
- 周围的坐标 (水平, 垂直)

奖励值为个体与目标距离的倒数，距离越大奖励值越低

状态空间是连续值

↓

离散化：水平方 100×100 = 10000个格

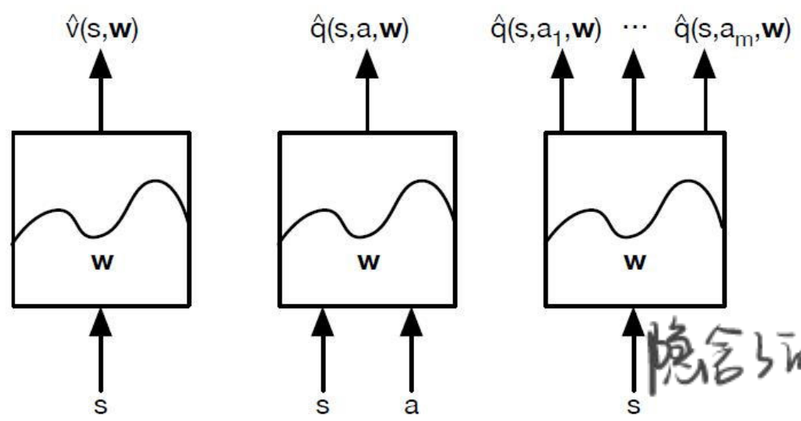
运动员、小球坐标 $10^4 * 10^4 = 10^8$ 种 若再加上个体速度分量，种类会更大

2. 价值函数的近似表示

由参数 w 指定

- 引入状态价值函数 $\hat{v}(s,w) \approx v_\pi(s)$
- 引入动作价值函数 $\hat{q}(s,a,w) \approx q_\pi(s,a)$
- 价值函数近似方法，如线性表示 $\hat{v}(s,w) = \phi(s)^T w$

决策树，最近邻，傅立叶变换，神经网络



隐含了动作是有限的离散动作

神经网络最为广泛应用，如：DNN、CNN、RNN

把价值函数的神经网络可看成一个黑盒子

3. Deep Q Learning 算法思路

	Deep Q Learning vs Q Learning	
基本思路	相同	
计算Q值	Q网络 Deep Q learning (DQN) { DNN, CNN, RNN }	通过状态值s、动作a
保存Q值	经验回放	Q表

每次的环境交互得到一条历史的图子更新时，先保存起来，用于更新后面的Q值

通过Q网络计算的Q值 和 经验回放得到的历史Q值 是有误差的

可通过梯度反向传播来更新神经网络的参数 w。

w 收敛后即可得到近似的Q值计算方法

Input: T, A, α, γ, ε, Q网络结构，批梯度下降的样本数m，状态特征维度n

Output: Q网络参数

1. 随机初始化Q网络的所有参数w

基于w初始化所有的状态s和动作a对应的价值Q

确定经验回放的集合D

2. for i from 1 to T

a) 初始化当前状态s为序列的第一个状态，获得特征向量φ(s)

b) Q网络中，φ(s)作为输入，得到所有动作的Q值输出

在状态s $\xrightarrow{\varepsilon\text{-贪婪法}}$ 在当前Q值输出中选择动作A

c) 执行当前动作A → 新状态s'的特征向量φ(s')，奖励R，是否状态is_end

d) 将{φ(s), A, R, φ(s'), is_end}五元组存入经验回放集合D

e) s=s'

f) 从经验回放集合D中采样m个样本{φ(sj), Aj, Rj, φ(sj'), is_endj}, j=1,2,...,m

计算当前目标Q值 $y_j = \begin{cases} R_j & \text{is_end}_j \text{ is true} \\ R_j + \gamma \max\limits_{a' \in A} Q(\varphi(s_j'), A_j', w) & \text{is_end}_j \text{ is false} \end{cases}$ Q现实

g) 均方差损失函数 $\frac{1}{m}\sum_{j=1}^{m}(y_j - \boxed{Q(\varphi(s_j), A_j, w)})^2$ Q估计

通过神经网络的梯度反向传播来更新Q网络的所有参数w

h) 若s'为终止状态，当前轮迭代完毕，否则转到步骤b)

解决DQN太不稳定或收敛慢的问题

不一定保证Q网络的收敛

7. Deep Q-learning进阶 Nature DQN

1. DQN (NIPS 2013) 的问题

可以像CartPole这样的简单游戏，但存在很多问题

目标Q值 y_j 计算方式：$y_j = \begin{cases} R_j & \text{is_end}_j \text{ is true} \\ R_j + \gamma \max\limits_{a' \in A} Q(\phi(S'_j), A'_j, w) & \text{is_end}_j \text{ is false} \end{cases}$

Q网络参数 w 计算 $Q(\phi(S'_j), A'_j, w)$

实际希望通过 y_j 来反向更新Q网络参数

两者循环依赖，迭代收敛两者相关性太强，不利于算法收敛

进阶版DQN：Nature DQN 尝试用两个Q网络来减少目标Q值计算和要更新Q网络参数之间的依赖关系

2. Nature DQN的建模

使用两个Q网络：

- 当前Q网络 Q，选择动作，更新模型参数 w — Q估计
- 目标Q网络 Q'：计算目标Q值 — Q现实

→ 其网络参数 w 无需迭代更新

所隔一段时间从当前Q网络 Q 复制过来，即延时更新

从而减少目标Q值与当前Q值的相关性

→ 结构必须一模一样才能复制

3. Nature DQN 算法流程

Input: T, A, ∂, γ, ε, Q网络结构, 批梯度下降的样本数m, targetQ更新频率C

Output: Q网络参数 < 当前Q网络结构Q, 目标Q网络结构Q', 参数更新频率C

1. 随机初始化Q网络的所有参数w, 目标Q网络结构Q'的参数w'=w

基于w初始化所有状态s和动作a对应的价值Q

清空经验回放的集合D

> 目标Q值计算是什么意思? 全用max Q计算是否可行?
> 随m梯度下降吗?
> Q值代表状态和动作的价值, 算出动作价值干什么呢?

2. for i from 1 to T

a) 初始化当前状态s为序列的第一个状态, 得到其特征向量φ(s)

b) Q网络中, φ(s)作为输入, 得到所有动作的Q值输出

在状态s $\xrightarrow{\varepsilon\text{-贪婪法}}$ 在当前Q值输出中选择动作A

c) 执行当前动作A → 新状态s'的特征向量φ(s'), 奖励R, 是否状态结束is_end

d) 存{φ(s), A, R, φ(s'), is_end} 进入经验回放集合D

e) s=s'

f) 从经验回放集合D中采样m个样本{φ(sj), Aj, Rj, φ(sj'), is_endj}, j=1,2,...,m

计算当前目标Q值 $y_j = \begin{cases} R_j & \text{(用目标Q网络Q' 计算的Q值)} \quad is_end_j \text{ is true} \\ R_j + \gamma \max_{a' \in A} Q(\varphi(s_j'), A_j', w) & is_end_j \text{ is false} \end{cases}$ Q现实

g) 均方差损失函数 $\frac{1}{m}\sum_{j=1}^{m}(y_j - Q(\varphi(s_j), A_j, w))^2$ Q估计 当前网络Q参数

通过神经网络的梯度反向传播来更新Q网络所有参数w

< 每C次, 则更新目标Q网络参数w'=w

h) 若s'为终止状态, 当前轮迭代完毕, 否则转到步骤b)

7. Double DQN (DDQN)

1. DQN的存在的问题

Q-learning - DQN - Nature DQN → 贪婪法 $\max_{a \in A}$

$$y_j = \begin{cases} R_j & \text{is_end}_j \text{ is true} \\ R_j + \gamma \max_{a' \in A} Q'(\phi(s_j'), a_j', w') & \text{is_end}_j \text{ is false} \end{cases}$$

使用max可以快速让Q值可能性优化朝向靠近

但容易造成不收敛、导致过度估计 (Over Estimation)

使最终得到的算法模型存在比较大的偏差 (bias)

(max Q' 最大化误差 → 把误差放进去估计的Q值偏大)

2. DDQN的算法建模

DDQN和Nature DQN一样，有两个Q网络的结构

通过解耦目标动作的选择和目标Q值的计算，消除过度估计

不再直接在目标Q网络里面找各个动作中最大Q值的

而是先在当前网络中去找出最大Q值对应的动作 (a)

$$a^{max}(s_j', w) = \arg\max_{a \in A} Q(\phi(s_j'), a, w)$$

当前网络Q

$$y_j = R_j + \gamma Q'(\phi(s_j'), a^{max}(s_j', w), w')$$

目标网络Q'

$$\Rightarrow y_j = R_j + \gamma Q'(\phi(s_j'), \arg\max_{a \in A} Q(\phi(s_j'), a, w), w')$$

3. DDQN算法流程

与Nature DQN的第一个不同步骤2.f中用目标网络值的计算

Input: T, A, ϕ, γ, ε, Q网络结构, 批梯度下降的样本数m, targetQ网络更新频率C

Output: Q网络参数

1. 随机初始化Q网络的所有参数w, 目标Q网络Q'的参数w'=w

基于w初始化所有的状态s和动作a对应的价值Q

清空经验回放的集合D

2. for i from 1 to T

a) 初始化当前状态s为序列的第一个状态, 得到其特征向量$\phi(s)$

b) Q网络中, $\phi(s)$作为输入, 得到所有动作的Q值输出

在状态s $\xrightarrow{\varepsilon\text{-贪婪法}}$ 在当前Q值输出中选择动作a

c) 执行当前动作A → 到新状态s'的特征向量$\phi(s')$, 奖励R, 是否结束is_end

d) 将$\{\phi(s), A, R, \phi(s'), is_end\}$这个四元组存入经验回放集合D

e) s=s'

f) 从经验回放集合D中采样m个样本$\{\phi(s_j), A, R, \phi(s'_j), is_end_j\}$, j=1,2,...,m

计算当前目标Q值 $y_j = \begin{cases} R_j & is_end_j \text{ is true} \\ R_j + \gamma \max_{a' \in A} Q'(\phi(s'_j), A'_j, w') & is_end_j \text{ is false} \end{cases}$

Q'的参数 $R_j + \gamma \max_{a' \in A} Q'(\phi(s'_j), \arg\max_{a' \in A}(Q(\phi(s'_j), a, w), w'), w')$

g) 均方差损失函数 $\frac{1}{m}\sum_{j=1}^{m}(y_j - Q(\phi(s_j), A, w))^2$ Q估计 (新的参数的当前网络Q)

通过神经网络的梯度反向传播来更新Q网络的所有参数w

(若i%C=1, 则更新目标Q网络参数w'=w

h) 若s'为终止状态, 当前轮迭代完毕, 否则转到步骤b)

十二. Prioritized Replay DQN

1. Prioritized Replay DQN 算法的内涵

DQN. Nature DQN. DDQN，经验回放等等，再次回顾的计算

- 采样的时候，经验回放池里的所有样本都有同样的采样到的概率
- 在经验回放池里的样本，由于时间，对目标值的作用也不一样
  - TD误差绝对值大，作用越大
  - TD误差绝对值小，作用越小
    - 目标Q网络的目标Q值 q_target
    - 当前Q网络的估计Q值 q_eval

TD误差绝对值 $|\delta(t)| = |q_target - q_eval|$

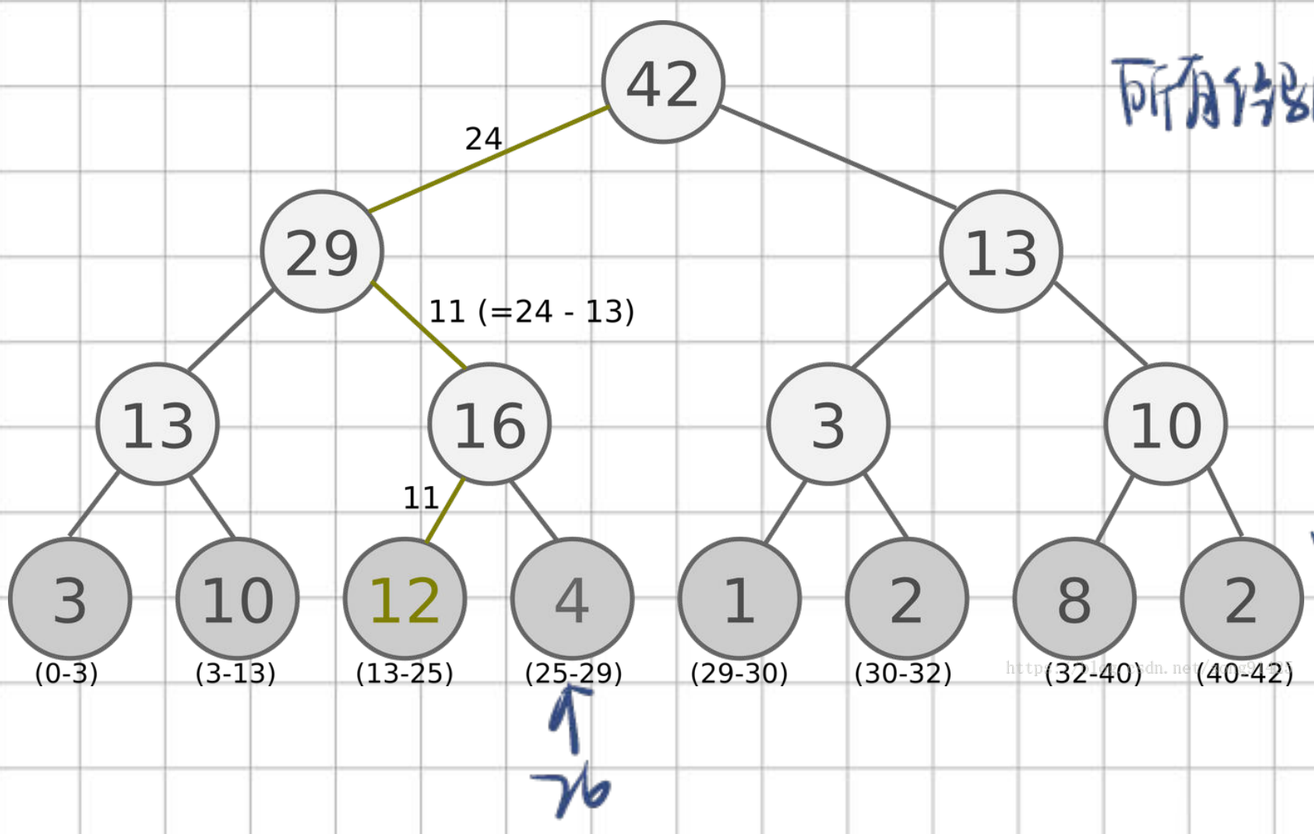
绝对值大的样本更容易被学习，再结合比较容易收敛

2. Prioritized Replay DQN 的算法实现

根据每个样本的TD误差绝对值 $|\delta(t)|$，给定该样本的优先级正比于 $|\delta(t)|$

$\Rightarrow$ 将优先级加在的经验回放池

实际使用中，通常使用SumTree二叉树的形式，存储带有优先级的经验回放池样本



所有经验回放样本只保存在最下面的叶子节点上

内部节点：
- 不保存样本数据
- 以子节点的优先级之和

叶子节点：
- 保存数据
- 优先级(数字)

(2) Q网络的损失函数：引入优先级权重 w_j

$$\frac{1}{m}\sum_{j=1}^{m}\left(y_j - Q(\phi(s_j), A(j), w)\right)^2$$

$$\frac{1}{m}\sum_{j=1}^{m} w_j \left(y_j - Q(\phi(s_j), A(j), w)\right)^2$$

第j个样本的优先级权重，由TD误差δ的归一化得到

(3) 对Q网络参数更新后，需重新计算TD误差，并修改更新到SumTree

3. Prioritized Replay DQN 算法流程

Input：T, B, α, γ, ε，批量梯度下降的样本数m，记忆阵列的长度n，

当前网络Q，目标网络Q'，目标网络Q'参数更新频率C

每次抽取数B，SumTree叶子节点数S

Output：Q网络参数

1. 随机初始化Q网络的所有参数w，目标Q网络的Q'的参数w'=w

基于w初始化所有的状态s和动作a对应的价值Q

给出的SumTree累加数组的叶子节点的优先级P(j)

2. for i from 1 to T

a) 初始化当前状态 S 为序列的第1个状态，获得特征向量 $\phi(S)$

b) 在网络中，$\phi(S)$ 作为输入，得到所有动作的Q值输出

在状态S $\xrightarrow{\varepsilon\text{-贪婪法}}$ 在当前Q值输出中选择动作A

c) 执行当前动作A → 新状态S'的特征向量 $\phi(S')$ 奖励R 是否状态结束end

d) 存 $\{\phi(S), A, R, \phi(S'), is_end\}$ 五元组存入SumTree

e) $S = S'$

f) 从SumTree中按照m个样本 $\{\phi(S_j), A, R, \phi(S'_j), is_end_j\}, j=1,2,\dots,m$

每个样本被采样的概率 $P(j) = \frac{p_j}{\sum_i p_i}$

计算出重要权重 $w_j = (N * P(j))^{-\beta} / \max_i(w_i)$

计算当前目标Q值 $y_j = \begin{cases} R_j & is_end_j \text{ is true} \\ R_j + \gamma Q'(\phi(S'_j), \underset{a'}{\arg\max}\, Q(\phi(S'_j), a, w), w') & is_end_j \text{ is false} \end{cases}$

g) 均方误差损失函数 $\frac{1}{m}\sum_{j=1}^{m} w_j (y_j - Q(\phi(S_j), A, w))^2$

通过神经网络的梯度反向传播来更新Q网络的所有参数w

h) 更新计算所有样本的TD误差 $\delta_j = y_j - Q(\phi(S_j), A_j, w)$

更新SumTree中所有节点的优先级 $p_j = |\delta_j|$

i) 若 $i \% C = 1$，则更新目标Q网络参数 $w' = w$

j) 若S'为终止状态，当前轮迭代完毕，否则转到步骤b)

十二、Dueling DQN

1、Dueling DQN 的优点总结

DDQN：优化目标Q值

$$\Rightarrow y_j = R_j + \gamma Q'(\varphi(s_{j+1}), \underbrace{\arg\max_{a'} Q(\varphi(s_{j+1}), a, w)}_{a'\text{选择}}, w')$$

Prioritized Replay DQN：优化经验回放池的采样优先级等待

$\Rightarrow$ 给采样的每个经验回放池SumTree

Q网络的损失函数：引入优先级权重 w_j

$$\frac{1}{m}\sum_{j=1}^{m} w_j (y_j - Q(\varphi(s_j), A, w))^2$$

重新计算TD误差，并将其更新到SumTree

Dueling DQN：将Q网络分为两部分，优化网络结构

第一部分：价值函数 $V(s, w, \alpha)$ 仅与状态有关，与所要选择的动作无关

第二部分：优势函数 $A(s, A, w, \beta)$ 同时与状态与动作有关

$$Q(s, A, w, \alpha, \beta) = V(s, w, \alpha) + A(s, A, w, \beta)$$

加上每种行动多获取的价值

如：沿路行驶，怎样不撞车？

网络参数
- w：公共部分
- α：价值函数独有
- β：优势函数独有

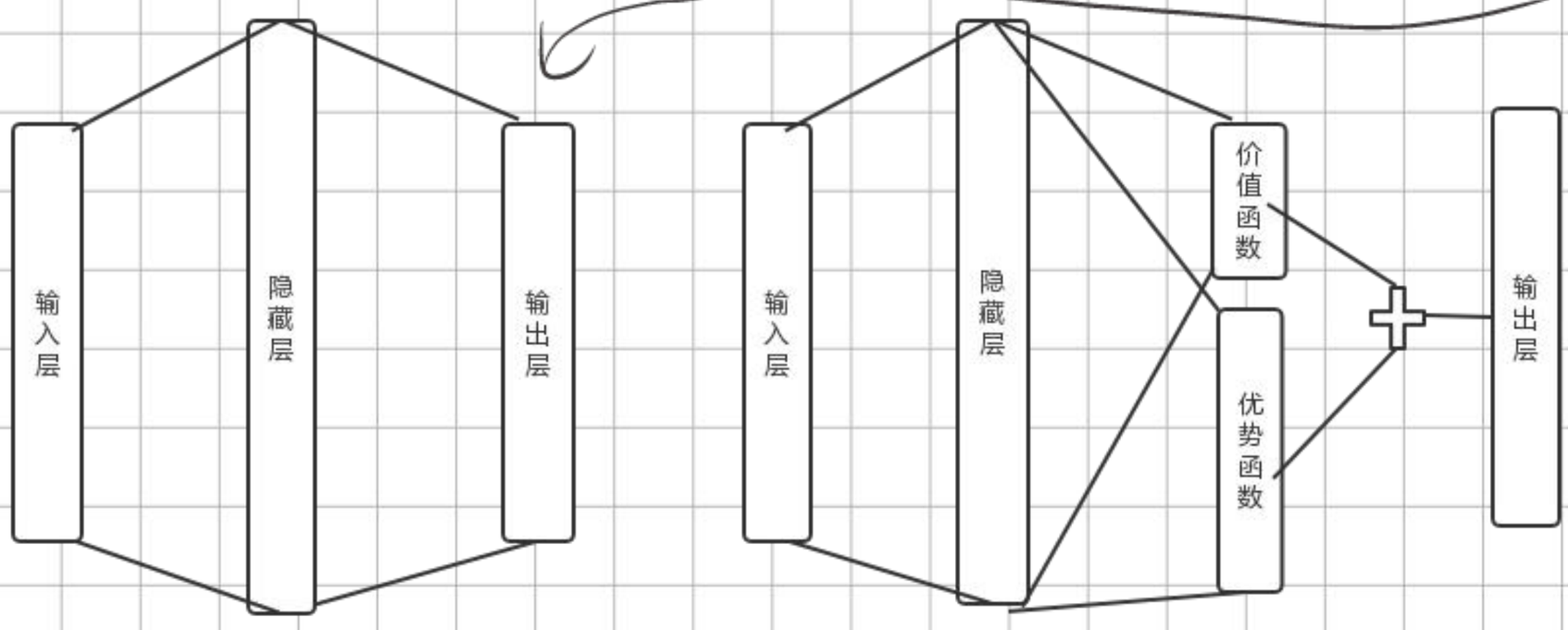
VALUE：驾驶任何行动对状态的变化没有影响

ADVANTAGE：可根据路况，每种行动从而改变状态 → 不会撞车

DQN
Nature DQN
Double DQN
Prioritized Replay DQN
Dueling DQN

2. Dueling DQN 的结构

以为简化算法描述，未使用CNN，采用最简单的三层神经网络 $\begin{cases}\text{输入层}\\ \text{隐藏层}\\ \text{输出层}\end{cases}$



$$Q(s,A,w,\alpha,\beta)=V(s,w,\alpha)+A(s,A,w,\beta)$$

无法保证能输出 $V(s,w,\alpha)$ 与 $A(s,A,w,\beta)$ 各自作用

以为保证可辨识性，实际使用组合方式如下：

$$Q(s,A,w,\alpha,\beta)=V(s,w,\alpha)+\underbrace{\left(A(s,A,w,\beta)-\frac{1}{|A|}\sum_{a'\in A}A(s,a',w,\beta)\right)}_{\text{平均化处理}}$$

$\begin{cases}\text{DQN、Nature DQN}\\ \text{DDQN、Prioritized DQN、Dueling DQN}\begin{cases}\text{无偏、}\\ \text{可混合使用}\\ \text{基于价值 Value-Based}\end{cases}\end{cases}$

但绝大多数DQN只能处理离散的动作集合

NAF DQN可解决连续问题，但很复杂 ⇒ 基于策略 Policy-Based

十三 策略梯度 Policy Gradient

DQN系列强化学习：对价值函数近似表示，基于价值学习

1. Value Based 强化学习小结

(1) 对连续动作的处理能力不足，只能处理离散动作

最有名的 DQN 家族变强的几乎都不作用

如：水球世界，若小圆在某方向上动，大小和方向是可以实现连续的 → 普通DQN得无能为力处理

因为此时策略是一个有具体值和方向的力，可以将其在水平和垂直方向上分解为两个连续的向量

→ 不必离散表述

(2) 对受限状态下的问题处理能力不足

使用特征值表征状态的某一特征时，

可能因为使用测量制或建模的限

导致真实环境下不同的两个状态，部分建模后有相同的特征值

→ 导致 value Based 方法无法分辨得到最优解

(3) 无法解决随机策略问题

Value Based 强化学习对应的最优策略，通常是确定性策略

从所有行为价值中选择最大价值的行为

2. Policy Based 强化学习方法引入

Value Based：对价值函数近似表示

引入动作价值函数 $\hat{q}$，由参数 w 描述并接受状态 s 与动作 a 作为输入：

$$\hat{q}(s,a,w) \approx q_\pi(s,a)$$

Policy Based：对策略近似表示

策略 π 可以被描述为一个包含参数 θ 的函数：

$$\pi_\theta(s,a) = P(a|s;\theta) \approx \pi(a|s)$$

将策略表示成参数函数后，就可用参数优化方法寻找最优策略，最常用的方法就是梯度上升法

3. 策略梯度优化目标

最简单的优化目标：初始状态的期望

$$J_1(\theta) = V_{\pi_\theta}(s_1) = E_{\pi_\theta}(v_1)$$

个别有的问题没有明确的初始状态，可将优化目标定义为平均价值：

$$J_{avV}(\theta) = \sum_s d_{\pi_\theta}(s) V_{\pi_\theta}(s)$$

$d_{\pi_\theta}(s)$ 是策略 π_θ 生成的马尔可夫链关于状态的静态分布

定义为每一时间片的平均奖励：

$$J_{avR}(\theta) = \sum_s d_{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a$$

无论用 J_1、J_{avV}、J_{avR} 等优化目标，最终对 θ 求导的一般形式为：

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, Q_\pi(s,a)\right]$$

分值函数模型　　回报

4. 策略函数的设计

最常用的策略函数：softmax 主要应用于离散空间

使用状态特征和行为的特征 $\phi(s,a)$ 与参数 θ 的线性组合来计算一个行为发生的概率

$$\pi_\theta(s,a) = \frac{e^{\phi(s,a)^T\theta}}{\sum_b e^{\phi(s,b)^T\theta}}$$

求导后得分值函数：$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - E_{\pi_\theta}[\phi(s,\cdot)]$

应用于连续行为空间的策略：高斯策略

该策略对于一个行为从高斯分布 $N(\phi(s)^T\theta, \sigma^2)$ 中产生

求导后得分值函数：$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a-\phi(s)^T\theta)\phi(s)}{\sigma^2}$

5. 蒙特卡洛策略梯度 reinforce 算法

仅对策略函数迭代更新，不学价值函数

不使用价值函数 $v(s)$，直接从策略梯度公式中的 $Q_{\pi_\theta}(s,a)$

Input：N个MC序列，训练步长 α

Output：参数 θ

1. for 每个MC序列 $\{s_1, a_1, r_2, \ldots, s_T, a_T, r_T\}$

   a. 用MC计算序列每个时间位置 t 的状态价值 v_t

   b. 对序列每个时间位置 t，使用梯度上升法更新 θ

   行动函数　中间度

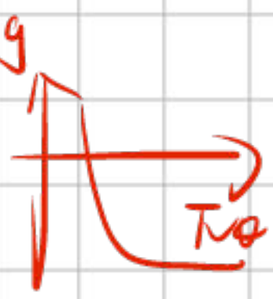
   $$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot v_t$$

   使此次选中的行为更有可能在下一次被发生（增大概率）

   但该行为是否"正确"？→ reward ＜ ×增加中间度 ✓增加中间度

2. 返回策略函数的参数 θ

状态 s 对所选动作 a 的计算



TD. Actor-Critic

基于模型：
- 基于策略：所有能行动的状态
- 基于价值：当前能行动的状态的V更新 → 动态规划DP

基于价值，贪婪算法 K个T状态序列

不基于模型：

回合更新 <

基于策略：< Policy Gradient $\theta=\theta+\alpha\cdot\nabla\cdot R$

贪婪算法 Policy Gradient $\theta=\theta+\alpha\cdot\nabla\cdot V$ 更新θ

回合更新的动作

步更新：时序差分TD 基于价值
- 在线 ε-贪婪
  - Sarsa 更新：更新 $Q=Q+\alpha(R+\gamma Q'-Q)$
  - Sarsa(λ) 更新 $Q=Q+\alpha\delta E$
- 离线 ε-贪婪 + 贪婪
  - Q Learning $Q=Q+\alpha(R+\gamma\max Q'-Q)$

动作价值转换机制算法

Actor-Critic

ABC = 多线程

DDPG = 4网络

更新θ.w

更新w

DQN: Q表 → Q网络

Nature DQN: 两个Q网络

DDQN: 当前网络选择 $\arg\max A'$

Prioritized Replay DQN: 优先级等待

Dueling DQN: 两个Q网络结构形成V+A

1. Actor-Critic 算法简介

Actor：演员 —策略函数→ 生成动作并和环境交互

Critic：评价者 —价值函数→ 评估Actor表现，指导Actor下一阶段动作

Policy Gradient：Actor

蒙特卡罗策略梯度 reinforce：计算价值函数 $U(s)$ 来代替Critic

Actor-Critic：类似DQN用网络拟合价值函数来替代 → 作为效率更高的Critic

做两个近似：

- 策略函数：$\pi_\theta(s,a) = P(a|s,\theta) \approx \pi(a|s)$
- 价值函数：
  - $\hat{v}(s,w) \approx v_\pi(s)$
  - $\hat{q}(s,a,w) \approx q_\pi(s,a)$

更新策略参数：$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s_t,a_t) v_t$

Actor

利用 v_t 迭代更新策略函数参数 θ

Critic：

类似DQN，用Q网络计算状态最优价值 v_t

蒙特卡罗和时序差分结合的方法

利用均方误差和梯度下降更新Q网络参数 w

2. Actor-Critic算法可选形式

(1) Critic评估点有很多情况：

a) 基于状态价值：前面所提到的

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) \cdot V(s,w)$$

b) 基于动作价值：在DQN中一般使用动作价值函数Q

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) \cdot Q(s,a,w)$$

c) 基于TD误差：(2)中TD误差

$$\delta(t) = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$\delta(t) = R_{t+1} + \gamma Q(s_{t+1}, A_{t+1}) - Q(s_t, A_t)$$

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) \cdot \delta(t)$$

d) 基于优势函数：

$$A(s,A,w,\beta) = Q(s,A,w,\alpha,\beta) - V(s,w,\alpha)$$

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) \cdot A(s,A,w,\beta)$$

e) 基于TD(λ)误差：一般定义为Forward TD(λ)误差

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) \cdot \delta(t) E(t)$$

(2) 对于Critic学习的模型参数w.

一般使用的误差最小化的数学优化迭代更新
类似于前DQN的Q的中所讲的迭代方法

若使用最简单的线性Q函数：$Q(s,a,w) = \phi(s,a)^T \cdot w$

则Critic学习模型参数w：$\delta = R_{t+1} + \gamma Q(s_{t+1}, A_{t+1}) - Q(s_t, A_t)$

$$w = w + \beta \delta \phi(s,a)$$

实际应用中，一般不使用线性Q函数，而是用神经网络来拟合状态和Q值的关系

3. Actor-Critic算法流程

Input: T. A. σ. β. γ. ε 状态特征维度 n

Critic网络. Actor网络

Output: Critic网络参数 w. Actor网络参数 θ

1. 随机初始化所有状态S. 动作a对应的价值Q

2. for i from 1 to T:

a) 初始化当前状态S为序列第一个状态. 得到特征向量 $\phi(S)$

回 b) Actor网络中. $\phi(S)$ 作为输入. 得到动作A

└ 执行当前动作A → 新状态S'. 反馈R

回 c) Critic网络中. $\phi(S)$. $\phi(S')$ 作为输入. 得到两个输出 $V(S)$. $V(S')$

└ d) 计算TD误差 $\delta = R + \gamma V(S') - V(S)$

e) 更新Critic网络参数w: 使用均方误差损失函数

$$\sum (R + \gamma V(S') - V(S, w))^2$$

f) 更新Actor网络参数 θ:

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(S, A) \delta$$

要实现强化学习算法. 有两个神经网络. 都需要梯度更新. 且互相依赖

十五. A3C

1. A3C 的引入

Actor-Critic：不能收敛

DQN：方便收敛，使用经验回放

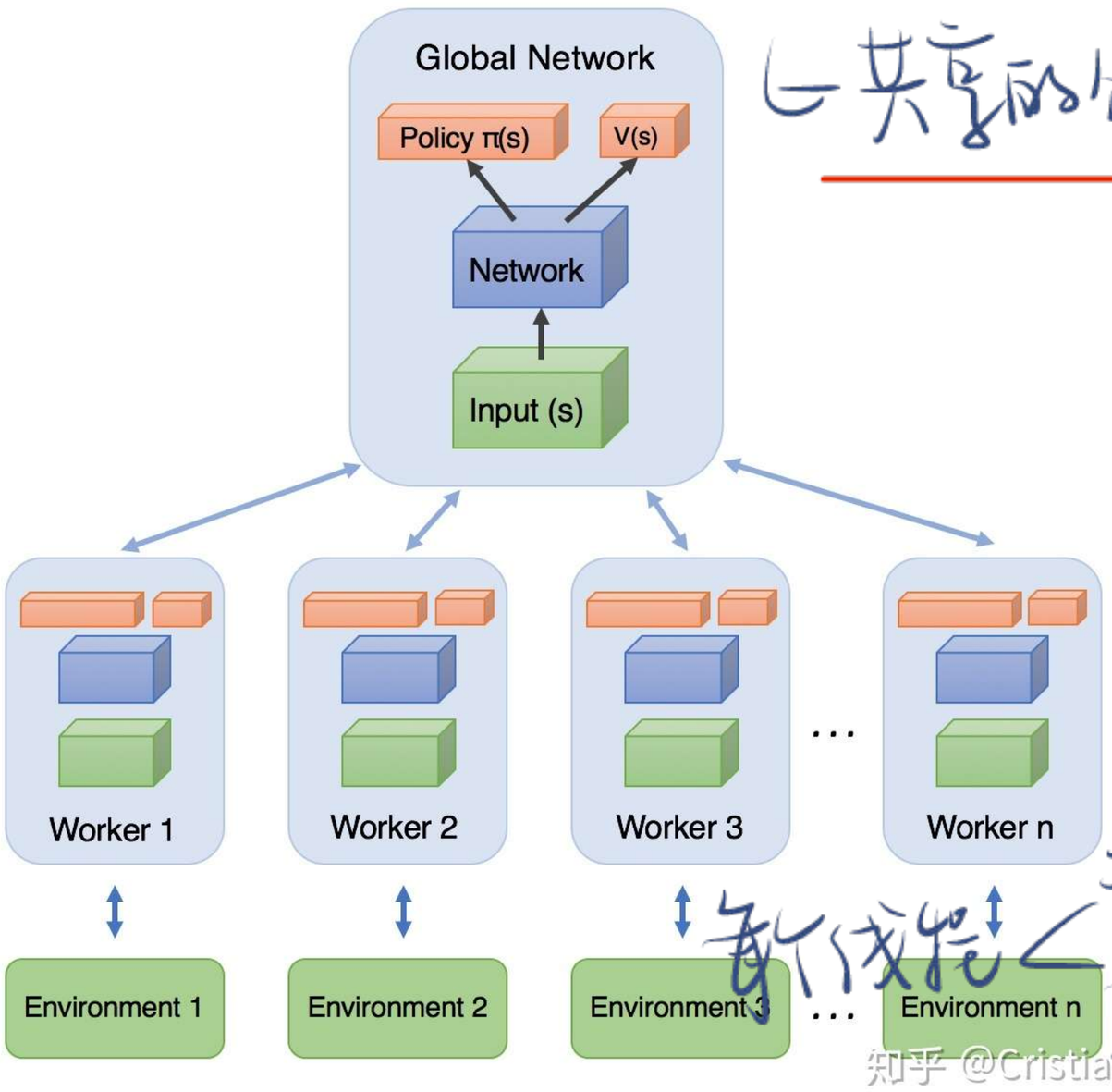
- 回放池经验数据时间关联性太强，用于训练效果可能不佳
  例如：当打下棋，总和同一个人下，到一定程度就很容易赢

A3C：利用多线程的方法

- 同时开多个线程分别和环境交互
- 每个线程把梯度累加成果元信息，整理后传到公共处
- 它们从公共处拿回成果，指导自己和环境后面的交互

2. A3C的算法优化

(1) 异步训练框架 ★

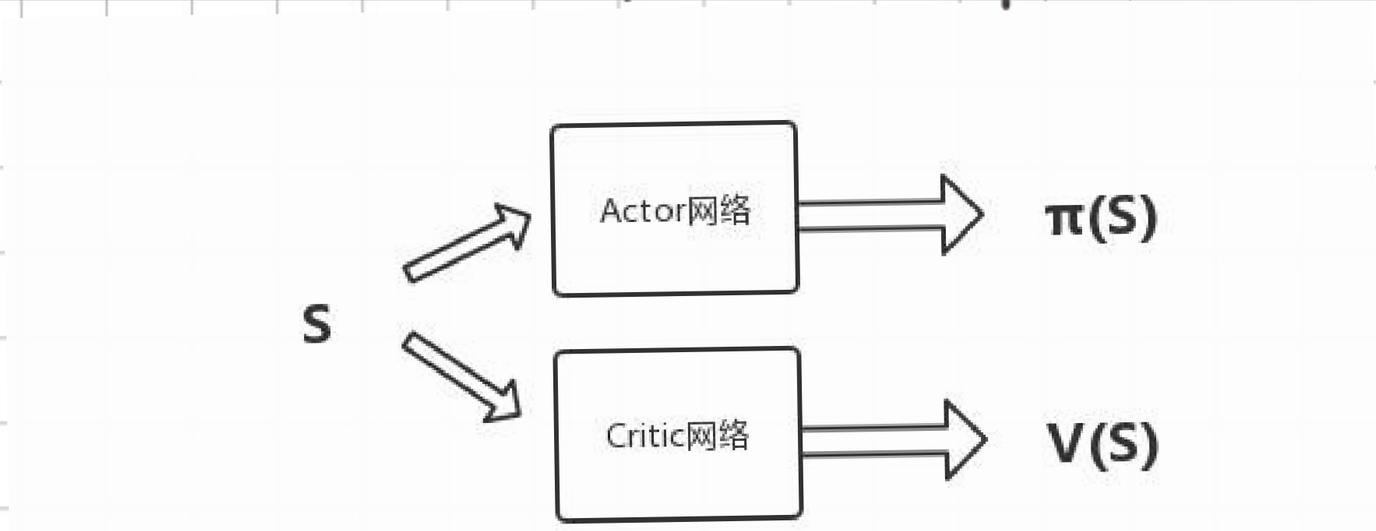


知乎 @Cristiano

- 共享的公共部分：公共的神经网络模型
  - 包括：
    - Actor网络 → Policy π(s)
    - Critic网络 → V(s)
- n个worker线程
  - 每个线程：
    - 和公共神经网络一样的网络结构
    - 独立和环境交互，控制公共策略
    - 和环境交互到一定数目后
    - 计算在每个线程中神经网络的损失函数梯度
    - → 更新公共神经网络的模型参数

(2) 网络结构优化

- Actor-Critic使用的两个不同的网络 Actor、Critic
- A3C把两个网络合并到一起，分别可共享底层参数



输入状态S，可输出状态价值V，动作策略π

0.3.3 优势函数估计优化

其中一种：可使用优势函数估计

$$A(s,a,\theta) = Q(s,a) - V(s)$$

$$Q(s,a) = R + \gamma V(s')$$

$$A(s,a,\theta) = R + \gamma V(s') - V(s)$$

由critic网络给出

A2C中，等待更新一步，使用n步等待，以加速收敛

$$A(s,a,\theta) = R_t + \gamma R_{t+1} + \cdots + \gamma^{n-1} R_{t+n-1} + \gamma^n V(s') - V(s)$$

Actor和critic共享卷积层，A2C与Actor-critic基本相同

但为防止梯度爆炸，在标准策略梯度上

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) A(s,\theta) + \beta \nabla_\theta H(\pi_\theta(s,\theta))$$

$r + \gamma V(s')$

$R_{t+n-1} + \gamma V(s')$

$R_{t+n-2} + \gamma R_{t+n-1} + \gamma^2 V(s')$

$R_{t+n-3} + \gamma R_{t+n-2} + \gamma^2 R_{t+n-1} + \gamma^3 V(s')$

3. A3C算法流程

A3C是异步多线程，此处给出单一线程的算法流程

Input: T, A, γ, β, γ, 状态特征维度n

全局最大迭代次数 T_{max}

线程内单次迭代时间序列最大长度 T_{local}

价值神经网络，对应参数 θ, w

本线程神经网络，对应参数 θ', w'

Output: 价值神经网络参数 θ, w

1. 更新时间序列 $t=1$
2. 重置Actor和Critic的梯度更新量：$d\theta \leftarrow 0$，$dw \leftarrow 0$
3. 从价值神经网络同步参数到本线程神经网络：$\theta' = \theta$，$w' = w$
4. $t_{start} = t$，初始化状态 s_t
5. 基于策略 $\pi(a_t|s_t;\theta)$ 选择出动作 a_t Actor
6. 执行动作 a_t，得到新状态 s_{t+1}，奖励 r_t
7. $t \leftarrow t+1$，$T \leftarrow t+1$
8. 如果 s_t 是终止状态，或 $t - t_{start} == t_{local}$，则进入步骤9，否则回到步骤5
9. 计算最后一个时间序列位置 s_t 的 $Q(s_t, t)$ Critic

迭代存储完整序列各状态 s_{t+1}, r_{t+1}

$$Q(s,t) = \begin{cases} 0 & \text{terminal state} \\ V(s_t, w') & \text{none terminal state, bootstrapping} \end{cases}$$

10. for $t \in (t-1, t-2, \ldots, t_{start})$:

1) 计算每个时刻的 $Q(s_t, a_t) = r_t + \gamma Q(s_{t+1}, a_{t+1})$ ② critic

2) 累计 Actor 的梯度更新 ③.1

$$d\theta \leftarrow d\theta + \underbrace{\nabla_{\theta'} \log \pi_{\theta'}(s_t, a_t)}_{\text{logprob}} \underbrace{(Q(s_t, a_t) - V(s_t, w))}_{\text{td}} + \underbrace{\alpha \nabla_{\theta'} H(\pi(s_t, \theta'))}_{\text{entropy-beta}}$$

normal-dist-entropy

3) 累计 critic 的梯度更新 ③.2

$$dw \leftarrow dw + \frac{\partial (Q(s_t, a_t) - V(s_t, w))^2}{\partial w'}$$

td²

11. 更新全局神经网络参数 ④

$$\theta = \theta - \alpha d\theta, \quad w = w - \beta dw$$

12. 若 $T > T_{max}$，则算法结束，输出全局神经网络参数 θ, w，否则重复步骤 2.

十六. DDPG 深度确定性策略梯度

解决Actor-Critic的问题：

A3C：多线程

DDPG：使用类似DQN的方法稳定网络&收敛

1. 从随机策略到确定性策略

DDPG：D(Deep) + D(Deterministic) + PG(Policy Gradient)

DPG：

对某一些动作集合，可能是连续数值 or 很高维的离散值 → 动作的空间维度很大

若用随机策略：像DQN一样研究所有可能的动作的概率，并计算各个可能的动作的值

→ 需要很大的样本量才可以

在相同的策略，在同一个状态处，

→ 每个动作是基于概率分布的不确定的

若用确定策略：

在相同的策略，即使在同一个状态，

→ 每个动作的概率不同，但最大概率只有一个

如果只取最大概率的动作，去掉这个概率分布，就简单很多 ⇒ $\pi_\theta(s) = a$

2. 从DPG到DDPG

基于θ值的随机性策略的梯度的梯度计算公式：

$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_\pi(s,a)]$$

状态的选择空间　　价值函数

需在整个动作的空间求导数

基于θ值的确定性策略梯度计算公式：

$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^\pi}[\nabla_\theta \pi_\theta(s)\, \nabla_a Q_\pi(s,a)|_{a=\pi_\theta(s)}]$$

与随机策略梯度相比，少了对动作的积分，多了相对θ的偏导的动作的导数

DPG → DDPG == DQN → DDQN

分类网络

双网络：当前网络、目标网络

由于本身就有网络Actor、Critic，

加上双网络后 → 4个网络 < Actor < 当前 目标；Critic < 当前 目标

5. DDPG原理

DDQN：

当前网络：

对当前状态s $\xrightarrow{\varepsilon\text{-贪婪法}}$ 选择动作A (q_eval)

执行动作A → 获得新状态s'，奖励R

存储本次入经验回放池

对经验回放池中选择的下一状态s' $\xrightarrow{\varepsilon\text{-贪婪法}}$ 选择动作A' (q_eval)

$y_j = R_j + \gamma \max_{a' \in A} Q'(\phi(s'_j), \arg\max_{a' \in A}(Q(\phi(s'_j), a, w), w')$

基于经验回放池计算当前q值 (q_eval)

当前网络更新参数 w

目标网络：

基于经验回放池计算目标q值 (q_target)

定期将当前网络复制到目标网络 $w' = w$

供当前网络计算损失值、训练参数

DDPG

Actor当前网络：策略网络参数 θ 迭代更新

根据当前状态 s → 选择当前动作 A

执行动作 A → 新状态 s'，奖励 R

Actor目标网络：根据经验回放池中采样的下一状态 s' → 选择下一动作 A'

定期网络参数 $\theta' = \theta$

Critic当前网络：价值网络参数 w 迭代更新

计算当前网络 Q 值 $Q(s, A, w)$ (q_eval)

Critic目标网络：计算目标网络 Q' 值 $Q'(s', A', w')$ (q_target)

定期网络参数 $w' = w$

参数复制：

DQN：硬更新 直接复制 $w'=w$

DDPG：软更新 每次参数只更新一点点

$$w' \leftarrow \tau w + (1-\tau) w'$$

$$\theta' \leftarrow \tau \theta + (1-\tau) \theta'$$

τ 是更新系数，一般取比较小，如：0.1 or 0.01

引入噪音：

为了探索可以增加一些随机性，增加它的复杂

对选择出的动作 A 增加一定的噪音 N：$A = \pi_\theta(s) + N$

损失函数：

对于 actor 当前网络，其损失函数和 DQN 类似，都是均方差：

$$J(w) = \frac{1}{m}\sum_{j=1}^{m}(y_j - Q(\varphi(s_j), A_j, w))^2$$

对于 Actor 当前网络，其损失函数和 DQN 不同，是时策略梯度

$$\nabla_J(\theta) = \frac{1}{m}\sum_{j=1}^{m}\left[\nabla_a Q(s_i, a_i, w)\big|_{s=s_i,\, a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)\big|_{s=s_i}\right]$$

对于同一个状态，输出两个不同的动作 a_1, a_2，从评论当前网络得到两个互换的 Q 值 Q_1, Q_2

若 $Q_1 > Q_2$，采取动作 a_1 可得到更多奖励，策略梯度：增加 a_1 的概率，降低 a_2 的概率

得到的互换 Q 值越大，损失越小；互换 Q 值越小，损失越大

$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{m} Q(s_i, a_i, w)$$ ✓ 取反

4. DDPG算法流程：

Input：T、γ，批量梯度下降的样本数m

最大迭代次数T

网络网络参数更新步长C

随机噪音函数N

Actor：当前网络 & 参数θ
目标网络 & 参数θ'

Critic：当前网络 & 参数w
目标网络 & 参数w'

Output：最优Actor当前网络参数θ
最优Critic当前网络参数w

1. 随机初始化θ，w，w'=w，θ'=θ
清空经验回放的集合D

2. for i from 1 to T：

a) 初始化S为当前状态序列的第一个状态，拿到其特征向量φ(S)

b) 在Actor当前网络 → 基于动作 $A=\pi_\theta(\phi(S))+N$

c) 执行动作A → 新状态S'，奖励R，是否is_end

d) 将{φ(S)，A，R，φ(S')，is_end}五元组存入经验回放集合D

e) S=S'

7) 从经验回放集合D中选择m个样本：

$\{\phi(s_j), a_j, R_j, \phi(s_j'), is_end_j\}, j=1,2,\cdots,m$

计算目标值 y_j 　　选择下一动作 a_j'

$$y_j=\begin{cases} R_j & is_end_j \text{ is true} \\ R_j+\gamma Q'(\phi(s_j'), \pi_{\theta'}(\phi(s_j')), w') & is_end_j \text{ is false} \end{cases}$$

8) 使用均方差损失函数：

计算当前价值

$$\frac{1}{m}\sum_{j=1}^{m}(y_j-Q(\phi(s_j), a_j, w))^2$$

通过神经网络梯度反向传播，更新Critic当前网络参数w

9) 使用策略梯度损失函数：

$$J(\theta)=-\frac{1}{m}\sum_{j=1}^{m}Q(s_i, a_i, \theta)$$

通过神经网络梯度反向传播，更新Actor当前网络参数θ

10) 如果T%C=1，更新 Critic目标网络参数 w'，Actor目标网络参数 θ'

$$w'\leftarrow\tau w+(1-\tau)w'$$

$$\theta'\leftarrow\tau\theta+(1-\tau)\theta'$$

11) 如果s'是终止状态，当前轮迭代完毕，否则转到步骤6)