國立清華大學
National Tsing Hua University

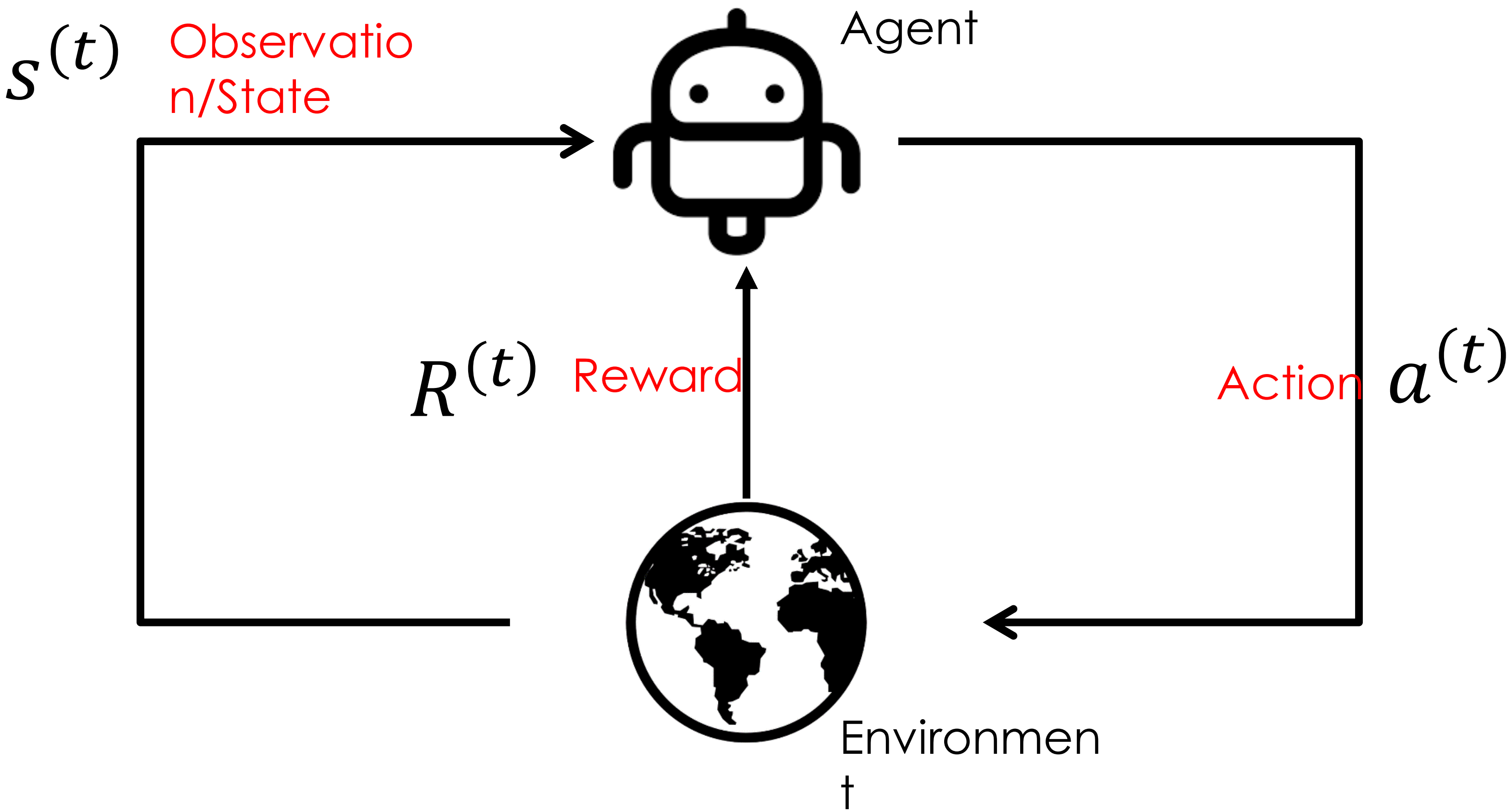
INNOVATION ENGINEERING LAB

智慧化企業整合
Intelligent Integration of Enterprise

Reinforcement Learning

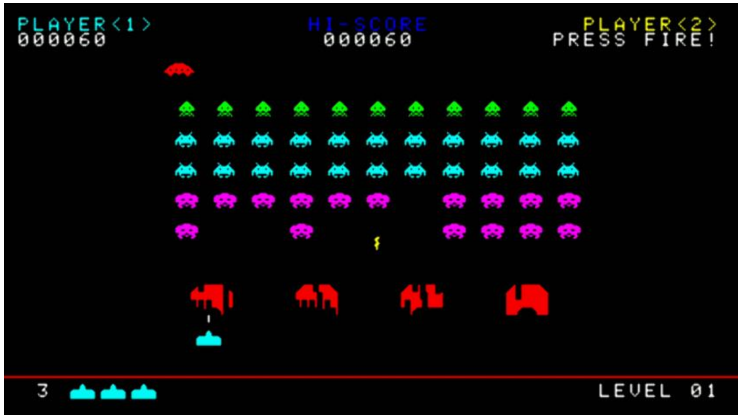
助教:黃日泓

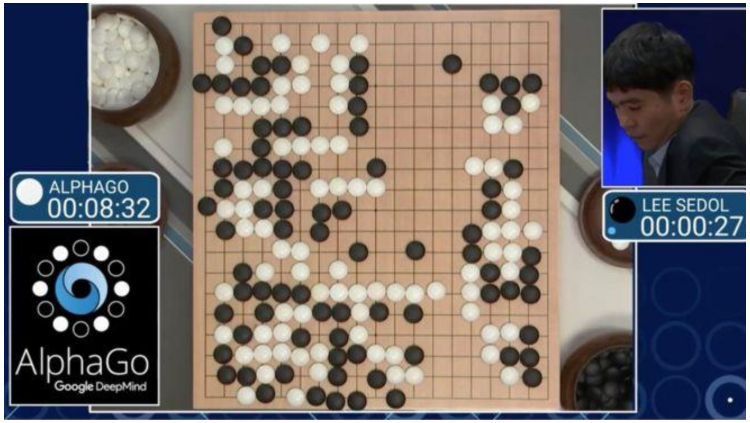
Reinforcement Learning



Reinforcement Learning

- An agent sees state, takes an action and receives a reward
- The state may change due to the action
- The state may also change without action
- Ex: space invader, AlphaGo…

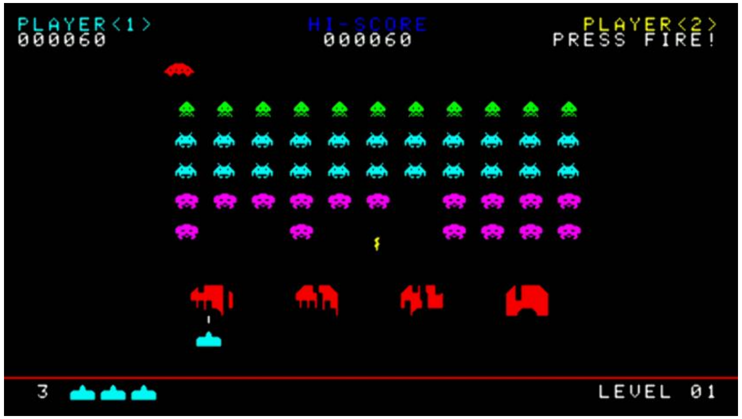


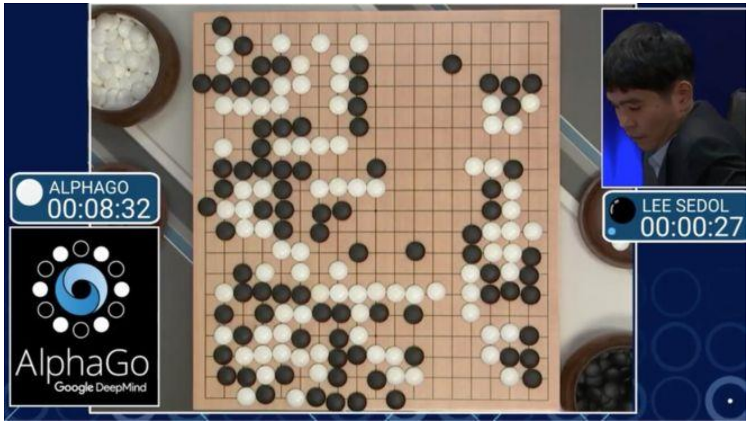


Reinforcement Learning

- Goal: to learn the best policy that maximize the total reward

$\pi^*(s^{(t)}) = a^{(t)}$ that maximizes $\Sigma_t R^{(t)}$





Markov process

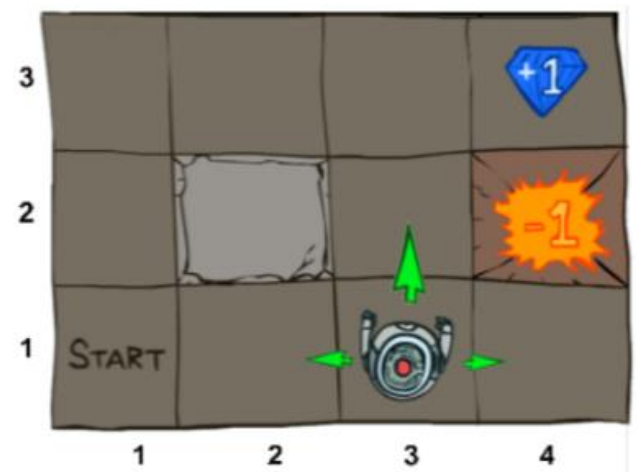
- A random process is a collection of time-indexed random variables
- A random process is called a Markov process if it satisfies the Markov property:

$$\mathrm{P}(\boldsymbol{s}^{(t+1)} | \boldsymbol{s}^{(t)}, \boldsymbol{s}^{(t-1)}, \cdots) = \mathrm{P}(\boldsymbol{s}^{(t+1)} | \boldsymbol{s}^{(t)})$$

- The future is independent of the past given the present
- The state captures all relevant information from history

Define Markov Decision Process (MDP)

- $\mathbb{S}$ the state space, $\mathbb{A}$ the action space
- Start state s^0
- $P(s'|s;a)$ the transition distribution, fixed over t
- $R(S,a,s')$ the deterministic reward function
- $\gamma \in [0,1]$ the discount factor



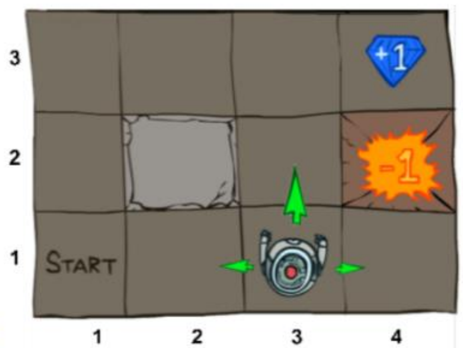
MDP

Given a policy $\pi(s) = a$, an MDP proceeds as follows:

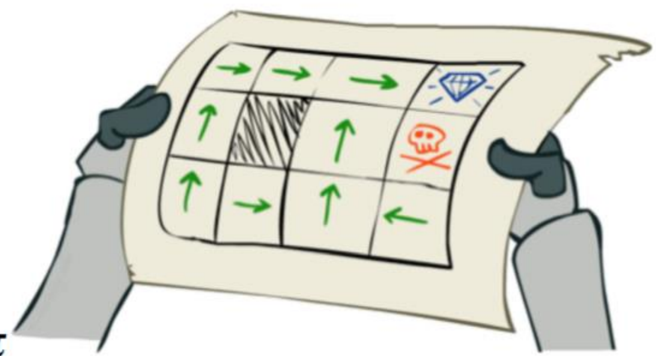
$$\boldsymbol{s}^{(0)} \xrightarrow{\boldsymbol{a}^{(0)}} \boldsymbol{s}^{(1)} \xrightarrow{\boldsymbol{a}^{(1)}} \cdots \xrightarrow{\boldsymbol{a}^{(H-1)}} \boldsymbol{s}^{(H)},$$

with the accumulative reward

$$R(\boldsymbol{s}^{(0)},\boldsymbol{a}^{(0)},\boldsymbol{s}^{(1)}) + \gamma R(\boldsymbol{s}^{(1)},\boldsymbol{a}^{(1)},\boldsymbol{s}^{(2)}) + \cdots + \gamma^{H-1} R(\boldsymbol{s}^{(H-1)},\boldsymbol{a}^{(H-1)},\boldsymbol{s}^{(H)})$$



MDP



π

MDP

Given a policy $\pi(s) = \boldsymbol{a}$, an MDP proceeds as follows:

$$\boldsymbol{s}^{(0)} \xrightarrow{\boldsymbol{a}^{(0)}} \boldsymbol{s}^{(1)} \xrightarrow{\boldsymbol{a}^{(1)}} \cdots \xrightarrow{\boldsymbol{a}^{(H-1)}} \boldsymbol{s}^{(H)},$$

with the accumulative reward

$$R(\boldsymbol{s}^{(0)}, \boldsymbol{a}^{(0)}, \boldsymbol{s}^{(1)}) + \gamma R(\boldsymbol{s}^{(1)}, \boldsymbol{a}^{(1)}, \boldsymbol{s}^{(2)}) + \cdots + \gamma^{H-1} R(\boldsymbol{s}^{(H-1)}, \boldsymbol{a}^{(H-1)}, \boldsymbol{s}^{(H)})$$

- To acquire rewards ASAP
- Different accumulative rewards in different trials

Goal

- Given a policy π, the expected accumulative rewards collected by taking actions can be expressed as:

$$V_\pi = \mathrm{E}_{\mathbf{s}^{(0)},\cdots,\mathbf{s}^{(H)}}\left(\sum_{t=0}^{H}\gamma^t R(\mathbf{s}^{(t)},\pi(\mathbf{s}^{(t)}),\mathbf{s}^{(t+1)});\pi\right)$$

- Our goal is to find the optimal policy

$$\pi^* = \arg\max_{\pi} V_\pi$$

Value Iteration

Optimal Value Function

$$\pi^* = \arg\max_{\pi} \mathrm{E}_{\mathbf{s}^{(0)},\cdots,\mathbf{s}^{(H)}} \left(\sum_{t=0}^{H} \gamma^t R(\mathbf{s}^{(t)}, \pi(s^{(t)}), \mathbf{s}^{(t+1)}); \pi \right)$$

- Optimal value function:
  Maximum expected accumulative rewards when starting from state s and acting optimally for h steps

每一步都是最佳的policy

$$V^{*(h)}(s) = \max_{\pi} \mathrm{E}_{\mathbf{s}^{(1)},\cdots,\mathbf{s}^{(h)}} \left(\sum_{t=0}^{h} \gamma^t R(\mathbf{s}^{(t)}, \pi(\mathbf{s}^{(t)}), \mathbf{s}^{(t+1)}) | \mathbf{s}^{(0)} = s; \pi \right)$$

Optimal Value Function

$$V^{*(h)}(\mathbf{s}) = \max_{\pi} \mathrm{E}_{\mathbf{s}^{(1)},\cdots,\mathbf{s}^{(h)}} \left(\sum_{t=0}^{h} \gamma^t R(\mathbf{s}^{(t)}, \pi(\mathbf{s}^{(t)}), \mathbf{s}^{(t+1)}) | \mathbf{s}^{(0)} = \mathbf{s}; \pi \right)$$

- Having $V^{*(\mathrm{H}-1)}(s)$ for each s, we can solve π^* by

由前一步可以找出現在這一步的最佳policy

$$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}} \sum_{\mathbf{s}'} \mathrm{P}(\mathbf{s}'|\mathbf{s};\mathbf{a})[R(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma V^{*(H-1)}(\mathbf{s}')], \forall \mathbf{s}$$

直接根據 accumulative reward最高的的policy行動

國立清華大學 National Tsing Hua University

INNOVATION ENGINEERING LAB

Dynamic Programming

$$V^{*(h)}(s) = \max_{\pi} \mathrm{E}_{\mathbf{s}^{(1)},\cdots,\mathbf{s}^{(H)}} \left(\sum_{t=0}^{h} \gamma^t R(\mathbf{s}^{(t)}, \pi(\mathbf{s}^{(t)}), \mathbf{s}^{(t+1)}) | \mathbf{s}^{(0)} = s; \pi \right)$$

- $h = H - 1$:

$$V^{*(H-1)}(\boldsymbol{s}) = \max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V^{*(H-2)}(\boldsymbol{s}')], \forall \boldsymbol{s}$$

- $h = H - 2$:

$$V^{*(H-2)}(\boldsymbol{s}) = \max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V^{*(H-3)}(\boldsymbol{s}')], \forall \boldsymbol{s}$$

- $h = 0$:

$$V^{*(0)}(\boldsymbol{s}) = \max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} \mathrm{P}(\boldsymbol{s}'|\boldsymbol{s};\boldsymbol{a})[R(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V^{*(-1)}(\boldsymbol{s}')], \forall \boldsymbol{s}$$

- $h = -1$:

$$V^{*(-1)}(\boldsymbol{s}) = 0, \forall \boldsymbol{s}$$

Policy Iteration

Policy Iteration

- Given a policy π, define value function:

$$V_{\pi}(\boldsymbol{s}) = \mathrm{E}_{\mathbf{s}^{(1)},\ldots}\left(\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}^{(t)}, \pi(\mathbf{s}^{(t)}), \mathbf{s}^{(t+1)}) | \mathbf{s}^{(0)} = \boldsymbol{s}; \pi\right)$$

- Start from state s and act based on policy π

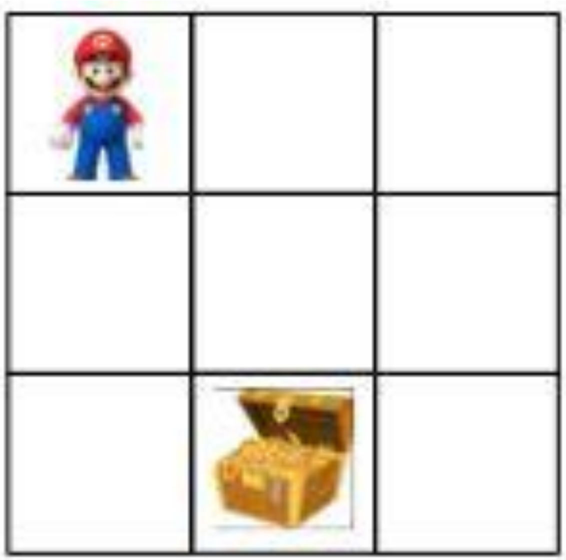
Policy Iteration

1. 隨機選擇一個策略當作初始值
2. 根據目前的策略計算Value function的值
3. 根據Value function的值計算目前狀態中最好的action
4. 修改策略

國立清華大學
National Tsing Hua University

INNOVATION ENGINEERING LAB

Policy Iteration



1. 假設有一個 3 x 3 的尋寶遊戲
2. 初始化: 不管在哪，一律往下走
3. 策略評估:
   - 如果寶藏在正下方，期望值較高
   - 如果寶藏不在正下方，期望值較低
4. 策略改善:
   - 如果寶藏在正下方，策略不變
   - 如果寶藏不在正下方，最好的策略是向右一步
5. 策略更新: 現在的策略是往下或往右走

這邊的policy的樣子就會類似一張告訴Agent「在什麼位置，要做什麼動作」的地圖

Policy Iteration

Algorithm: Policy Iteration

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do**
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$*'s converge*;
  foreach s **do**
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$*'s converge*;

Algorithm 5: Policy iteration.

Randomly pick

Policy Iteration

Algorithm: Policy Iteration

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
  For each state s, initialize $V_\pi(s) \leftarrow 0$;
  repeat
    foreach s **do**
      $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
    end
  until $V_\pi(s)$ *'s converge*;
  foreach s **do**
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
  end
until $\pi(s)$ *'s converge*;

Algorithm 5: Policy iteration.

Evaluate $V_\pi(s)$

Policy Iteration

Algorithm: Policy Iteration

Input: MDP $(\mathbb{S}, \mathbb{A}, \mathrm{P}, R, \gamma, H \to \infty)$
Output: $\pi(s)$'s for all s's

For each state s, initialize $\pi(s)$ randomly;
repeat
    For each state s, initialize $V_\pi(s) \leftarrow 0$;
    repeat
        foreach s **do**
            $V_\pi(s) \leftarrow \sum_{s'} \mathrm{P}(s'|s;\pi(s))[R(s,\pi(s),s') + \gamma V_\pi(s')]$;
        end
    until $V_\pi(s)$ *'s converge*;
    foreach s **do**
        $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma V_\pi(s')]$;
    end
until $\pi(s)$ *'s converge*;

Algorithm 5: Policy iteration.

Improve $\pi(s)$

Q-Learning

Q Function

$$Q^*(s,a) = \max_{\pi} \mathrm{E}_{\mathbf{s}^{(1)},\ldots}\left(R(s,a,\mathbf{s}^{(1)}) + \sum_{t=1}^{\infty} \gamma^t R(\mathbf{s}^{(t)}, \pi(\mathbf{s}^{(t)}), \mathbf{s}^{(t+1)}); s,a,\pi \right)$$

$$Q^*(s,a) = \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')], \forall s$$

Policy iteration:
會有一張告訴Agent「在什麼位置，要做什麼動作」的地圖

Q learning:
會有一個由state跟action組成的表格以儲存Q值

Q-table

Q-Table	a1	a2
s1	q(s1,a1)	q(s1,a2)
s2	q(s2,a1)	q(s2,a2)
s3	q(s3,a1)	q(s3,a2)

Q table

兩小時後要meeting，我們得到的初始Q-table可能是這個樣子

	a1: 看paper	a2: 玩楓之谷
s1	5	-3
s2	4	-2

1. 假設我們在s1，決定看paper (因為Q高)
2. 到達s2，不採取動作，而是看Q(s2,a1)與Q(s2,a2)哪個大
3. Q(s2,a1)比較大，將它乘γ，再加上s1到s2拿到的reward(5)
   →得到現實中Q(s1,a1)的值，並和Q-table中原本的值(5)比較
4. 更新表格

Temporal Difference

- We want to estimate:

$$Q^*(s,a) = \sum_{s'} \mathrm{P}(s'|s;a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')], \forall s$$

Temporal difference (TD) estimation of $Q^*(s,a)$ for exploitation policy π:

1. $\hat{Q}^*(s,a) \leftarrow$ randon value, $\forall s,a$
2. Repeat until converge ***for each action*** $a^{(t)}$:

$$\hat{Q}^*(s^{(t)},a^{(t)}) \leftarrow \hat{Q}^*(s^{(t)},a^{(t)}) + \eta\left[(R(s^{(t)},a^{(t)},s^{(t+1)}) + \gamma \max_a \hat{Q}^*(s^{(t+1)},a)) - \hat{Q}^*(s^{(t)},a^{(t)})\right]$$

現實的情況下可能拿到的reward
(注意: 這裡只是想像未來)

Q-table中存的reward值

References

- 清大資工吳尚鴻老師的講義
- 清大資工吳尚鴻老師的影片
- 台大電機李宏毅老師的影片
- Policy iteration介紹
- Q-learning介紹

Homework

- Please see the attached files (run.ipynb/RL_brain.py/maze_env.py) carefully and write comments to illustrate how these code work in run.ipynb/RL_brain.py/maze_env.py(optional).
- (Optional)You are also encouraged to modify the code under different situation (different maze size, episodes, learning rate, gamma or episilon) and illustrate your observations.
- Turn in your work with the package (*.zip) that contains the three files above.
- Folder name: hw8_Your Chinese Name