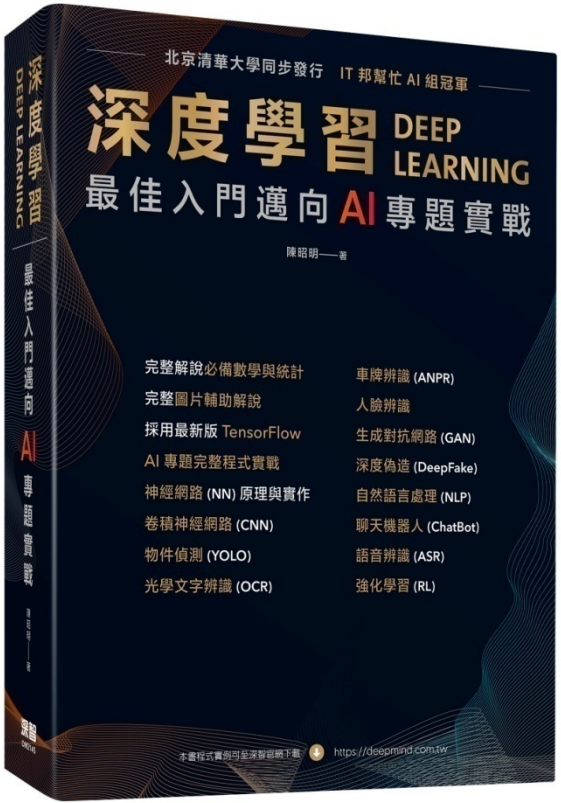
深度學習
最佳入門邁向AI專題實戰 (4)

陳昭明

2021-08-18


北京清華大學同步發行　IT 邦幫忙 AI 組冠軍
深度學習 DEEP LEARNING
最佳入門邁向AI專題實戰
陳昭明——著
完整解說必備數學與統計
完整圖片輔助解說
採用最新版 TensorFlow
AI 專題完整程式實戰
神經網路 (NN) 原理與實作
卷積神經網路 (CNN)
物件偵測 (YOLO)
光學文字辨識 (OCR)
車牌辨識 (ANPR)
人臉辨識
生成對抗網路 (GAN)
深度偽造 (DeepFake)
自然語言處理 (NLP)
聊天機器人 (ChatBot)
語音辨識 (ASR)
強化學習 (RL)
本書程式實例可至深智官網下載
https://deepmind.com.tw
深智
DEEP LEARNING
最佳入門邁向AI專題實戰
陳昭明——著

大綱

- 第一篇 深度學習導論
- 第二篇 TensorFlow基礎篇
- 第三篇 進階的影像應用
- 第四篇 自然語言處理
- 第五篇 強化學習

大綱

- 第一篇 深度學習導論
- 第二篇 TensorFlow基礎篇
- 第三篇 進階的影像應用
- 第四篇 自然語言處理
- 第五篇 強化學習

機器學習
(Machine Learning)

監督式學習
(Supervised Learning)

非監督式學習
(Unsupervised Learning)

強化學習
(Reinforcement learning)

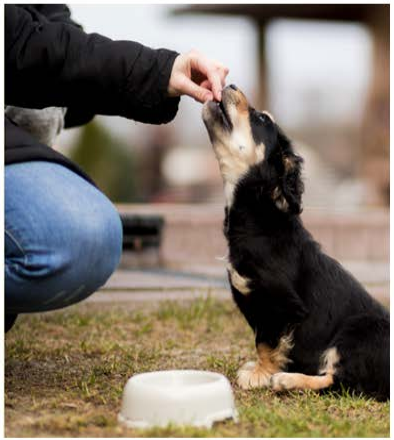
- 迴歸(Regression)
- 分類(Classification)

- 集群(Clustering)
- 降維(Dimensionality Reduction)

強化學習概念

強化學習

- 透過不斷的嘗試與錯誤(Trial and Error)，自我學習一段時間後，電腦就可以找到最佳的行動策略。
- 不教狗如何接飛盤，而是由飼主不斷地拋出飛盤讓狗練習，如果牠成功接到飛盤，就給予食物獎勵，反之就不給獎勵。



應用領域

- 下棋、電玩遊戲策略(game playing)。
- 製造/醫療/服務機器人的控制策略(Robotic motor control)。
- 廣告投放策略 (Ad-placement optimization)。
- 金融投資交易策略 (Stock market trading strategies)
- 運輸路線的規劃 (Transportation Routing)。
- 庫存管理策略 (Inventory Management)、生產排程 (Production scheduling)。
- 戰爭??

馬可夫決策過程
(Markov Decision Processes，MDP)

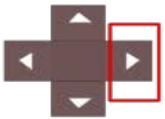
- 代理人行動後，環境會依據行動更新狀態，並給予獎勵。
- 代理人觀察所處的狀態及之前的行動，以最大化報酬的前提下，決定下次的行動。

遊戲

State
S_t

Agent

Reward
R_t



Action
A_t

演進

馬可夫過程
MP

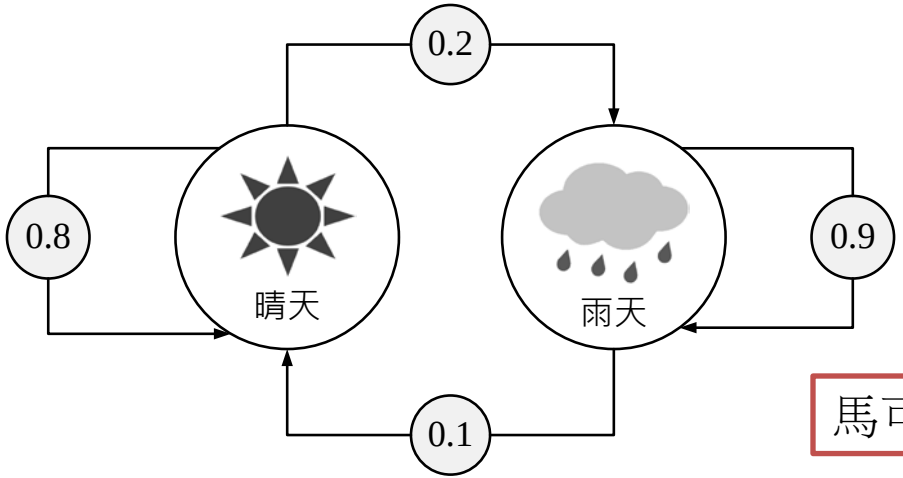
馬可夫獎勵過程
MRP

馬可夫決策過程
MDP

馬可夫過程 (Markov Process)

	sunny	rainy
sunny	0.8	0.2
rainy	0.1	0.9

狀態轉移機率
(State Transition Matrix)



馬可夫過程也稱馬可夫鏈(Markov Chain)

Quiz

- 請問下表機率為何?

狀態	機率
今天是晴天，『後天』是晴天的機率	
今天是雨天，『後天』是晴天的機率	

Markov Reward Process

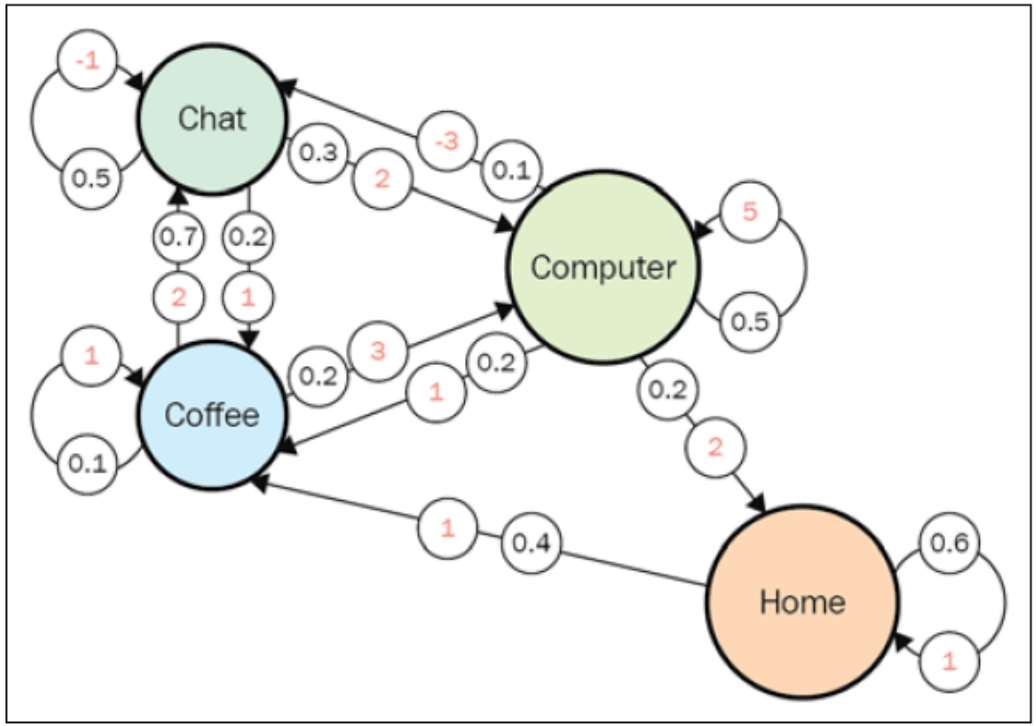
Markov Process

+

Reward

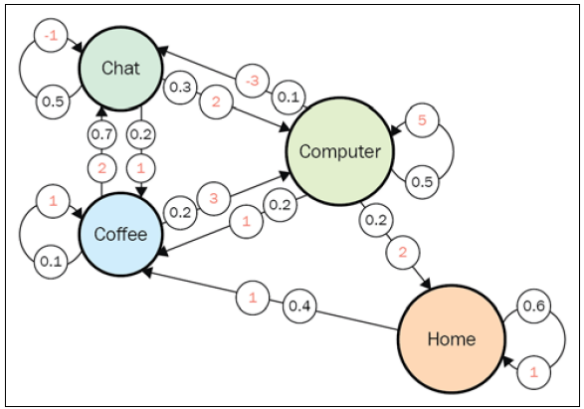
=

Markov Reward Process



Markov Reward Process (MRP)

MP + 獎勵(Reward)= MRP



計算狀態期望值

$$V(chat) = -1 * 0.5 + 2 * 0.3 + 1 * 0.2 = 0.3$$

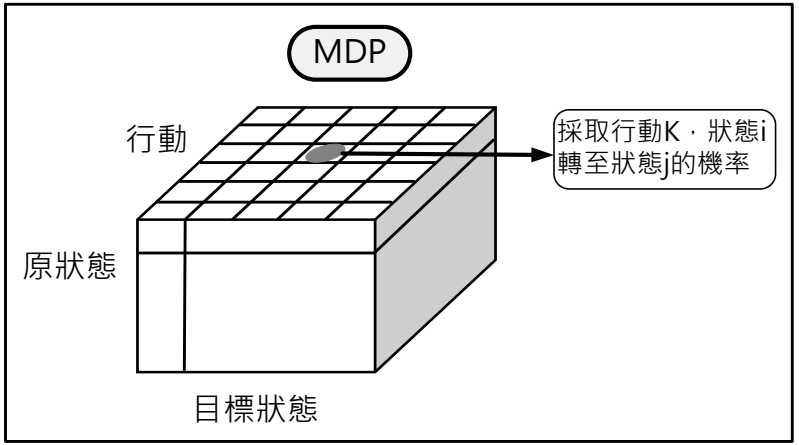
$$V(coffee) = 2 * 0.7 + 1 * 0.1 + 3 * 0.2 = 2.1$$

$$V(home) = 1 * 0.6 + 1 * 0.4 = 1.0$$

$$V(computer) = 5 * 0.5 + (-3) * 0.1 + 1 * 0.2 + 2 * 0.2 = 2.8$$

馬可夫決策過程
Markov Decision Process

MRP+行動轉移矩陣=MDP



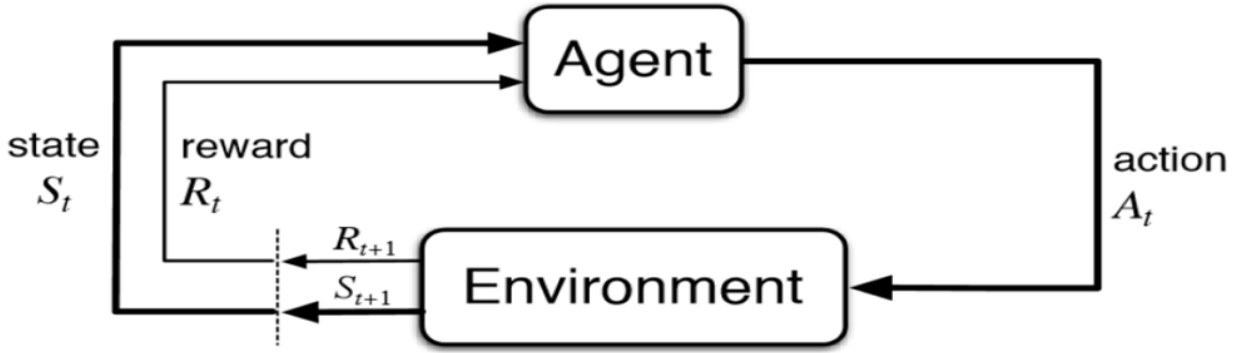
行動轉移矩陣(Action Transition matrix)

強化學習的目標

- 強化學習的目標就是在MDP的機制下，要找出最佳的行動策略，而目的是希望獲得最大的報酬。

Bellman 方程式

強化學習機制與數學符號



- Time Step: t
- State：S_t
- Action：a_t
- Reward: R_{t+1}
- Policy：π

行動軌跡 (trajectory)

- RL 的過程就是就如下面的軌跡(trajectory)：
  State Action Reward
  $\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, \cdots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}\}$
- 最佳行動就是取得最大長期獎勵(Return)：

  choose each A_t to maximize $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

  – γ：折扣率，介於[0,1)，避免要考慮未來無限期數的狀態(state)。

重要定義

- 狀態轉移機率(State Transition Matrix)
- 報酬(Return)
  - 折扣報酬(Discount Return)
- 狀態值函數(State Value Function)
- 行動值函數(Action Value Function)
- 倒推圖(Backup Diagram)

狀態轉移機率(State Transition Matrix)

- 行動軌跡(trajectory)如下：
- $\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, \cdots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}\cdots\}$

- 狀態轉移機率：到達狀態S_{t+1}的機率
  $p(S_{t+1} | S_t, A_t, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, S_{t-3}, A_{t-3}\cdots)$
- 依據『馬可夫性質』的假設，S_{t+1} 只與前一個狀態(S_t)有關，上式簡化為：
- $p(S_{t+1} | S_t, A_t)$

報酬(Return)、折扣報酬(Discount Return)

- 報酬(Return) ：就以走迷宮當例子，到達終點時，所累積的獎勵總和稱為報酬。
- 從t時間點走到終點(T)的累積獎勵：

  $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T = R_{t+k}$

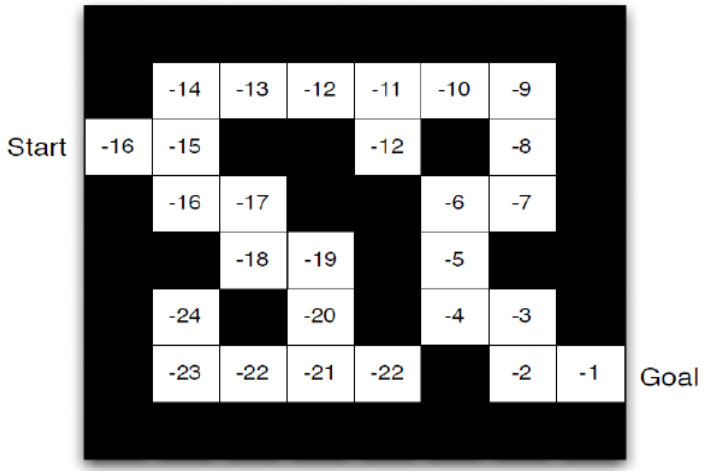
- 折扣報酬(Discount Return)：模型目標是追求報酬最大化，若迷宮很大的話要考慮的獎勵(R_i)個數也會很多，所以為了簡化模型，將每個時間的獎勵乘以一個小於1的折扣因子(γ)，讓越久遠的獎勵越不重要，避免要考慮太多的狀態，類似複利的概念。

折扣報酬 公式

- $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T = \gamma^{k-1} R_{t+k}$
  - 又$G_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots + \gamma^{T-2} R_T$
  - 故$G_t = R_{t+1} + \gamma G_{t+1}$

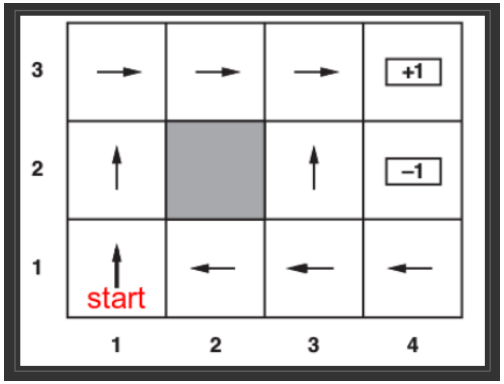
範例1. 迷宮每一個位置的報酬計算

- 目標是以最短路徑到達終點，故設定每走一步獎勵為-1，即可算出每一個位置的報酬，計算方法是由終點倒推回起點，結果如下圖中的數字所示。



範例2

- 起點為(1, 1)，終點為(4, 3)或(4, 2)，走到(4, 3)獎勵為1，走到(4, 2)獎勵為 -1，每走一步獎勵均為 -0.04。



假設有三種走法如下，請計算(1, 1)狀態期望值。

① (1,1) → (1,2) → (1,3) → (1,2) → (1,3) → (2,3) → (3,3) → (4,3)
-0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 +1

② (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,2) → (3,3) → (4,3)
-0.04 -0.04 -0.04 -0.04 -0.04 -0.04 -0.04 +1

③ (1,1) → (2,1) → (3,1) → (3,2) → (4,2)
-0.04 -0.04 -0.04 -0.04 -1

解答

- 三條走法的報酬計算如下：
  - (1) 1 - 0.04 x 7 = 0.72
  - (2) 1 - 0.04 x 7 = 0.72
  - (3) -1 - 0.04 x 4 = -1.16

- (1, 1)狀態期望值 =
  (0.72 + 0.72 + (-1.16)) / 3 = 0.28 / 3 ≒ 0.09

狀態值函數定義

- 以$V_{\pi}(s)$表示，其中π為特定策略。
- $V_{\pi}(s) = E(G|S=s)$

Bellman方程式

- Bellman方程式：特定策略下採取各種行動的機率。

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right]$$

- 其中：
  - s'：下一狀態。
  - π(a|s)：採取特定策略時，在狀態s採取行動a的機率。
  - $P^a_{ss'}$：為行動轉移機率，即在狀態s採取行動a，會達到狀態s'的機率。
  - 後面[]的公式係依據$G_t = R_{t+1} + \gamma G_{t+1}$ 轉換而來。

說明

- Bellman方程式讓我們可以從下一狀態的獎勵/狀態值函數推算出目前狀態的值函數。

- 在目前狀態下怎麼會知道下一個狀態的值函數？
  - 強化學習是以嘗試錯誤(Trial and Error)的方式進行訓練，我們可以從之前的訓練結果推算目前回合的值函數。
  - 譬如，要算第50回合的值函數，可以從1~49回合推算每一狀態的期望值。

行動值函數(Action Value Function)

- 行動值函數：採取某一行動的值函數，類似狀態值函數，公式如下。

$$q_\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(a'|s')\, q_\pi(s',a')\right]$$

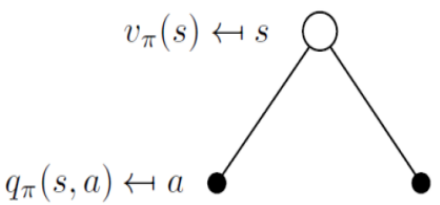
- VS.

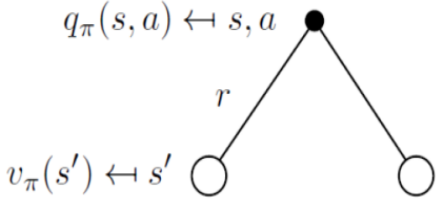
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right]$$

同樣可以從下一狀態的獎勵/行動值函數推算目前行動值函數。

倒推圖(Backup Diagram)

- 幫助理解演算法邏輯。





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

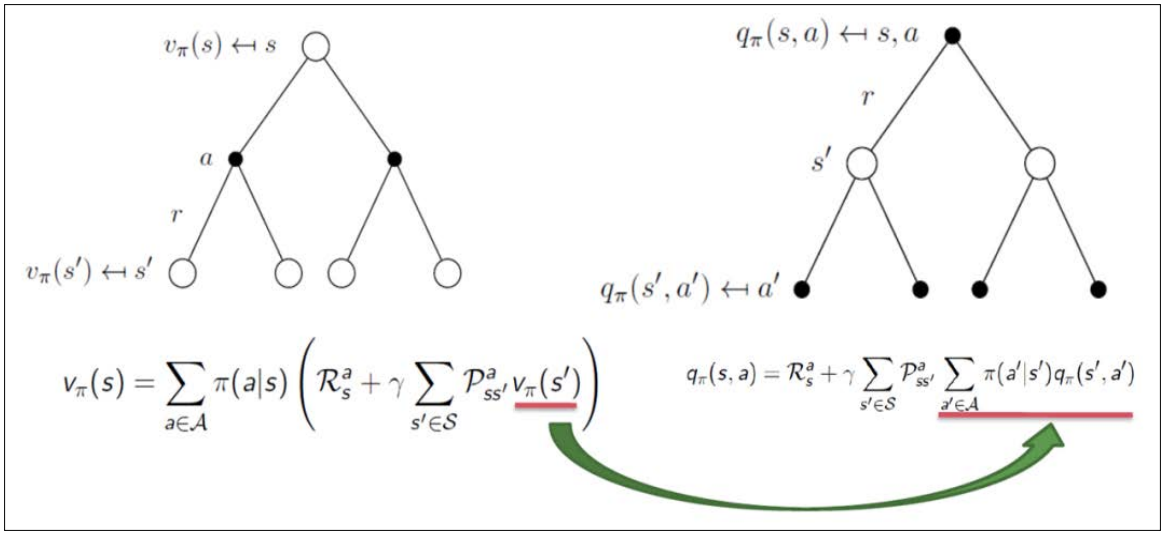
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

v：所有行動得到的獎勵和

q：單一行動得到的獎勵與狀態值總和

倒推圖 2

- 再往下展開。

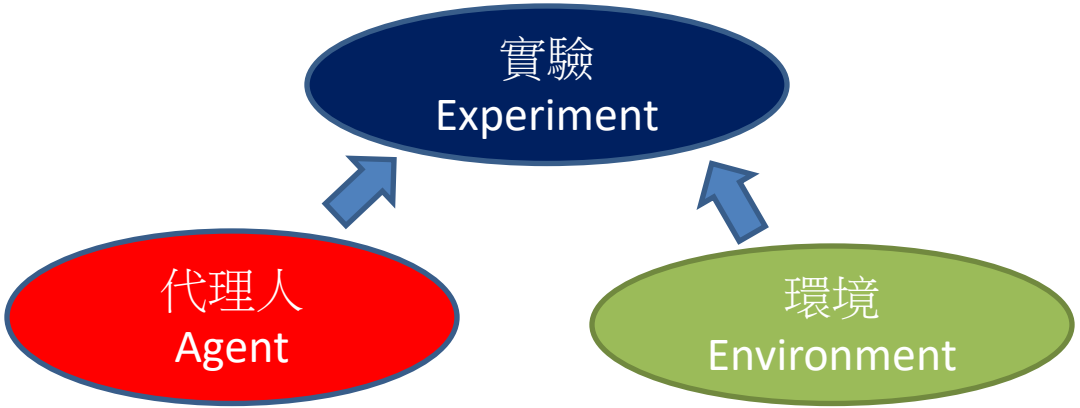

$v_\pi(s) \leftarrow s$
a
r
$v_\pi(s') \leftarrow s'$
$q_\pi(s,a) \leftarrow s,a$
r
s'
$q_\pi(s',a') \leftarrow a'$
$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\right)$
$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}}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

貪婪(Greedy)行動策略

- 依照上述公式行動，不斷更新所有狀態值函數，之後在每一個狀態下，即以最大化狀態或行動值函數為準則，採取行動，以獲取最大報酬。

強化學習程式架構

強化學習程式架構



類別與方法

- 環境(Environment)：例如迷宮、遊戲或圍棋，它必需給予獎勵並負責狀態轉換，若是單人遊戲，環境還必須擔任玩家的對手，例如電腦圍棋。

- 代理人(Agent)：即玩家。

環境(Environment)類別與方法

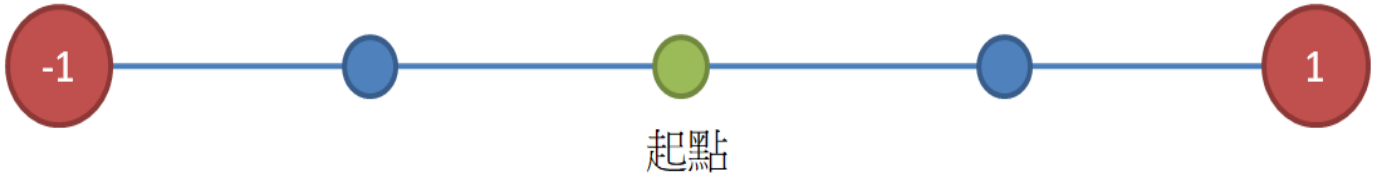
- 職責列舉如下：
  - 初始化(Init)：需定義狀態空間(State Space)、獎勵(Reward)辦法、行動空間(Action Space)、狀態轉換(State Transition definition)。
  - 重置(Reset)：每一回合(Episode)結束時，需重新開始，重置所有變數。
  - 步驟(Step)：代理人行動後，會驅動行動軌跡的下一步，環境會更新狀態，給予獎勵，並判斷回合是否結束及勝負。
  - Render（渲染）：更新畫面顯示。

代理人(Agent) 類別與方法

- 代理人(Agent)：即玩家，職責列舉如下。
  - 行動(Act)：代理人依據既定的策略以及所處的狀態，採取行動，例如上、下、左、右。
  - 通常我們要訂定特殊的策略，就繼承代理人類別(Agent class)，在衍生的類別中，覆寫行動函數(Act)，撰寫策略邏輯。

實作1. 簡單的遊戲

- 共有5個位置，玩家一開始站中間位置，每走一步扣分0.2，走到左端點得-1分，走到右端點得1分，走到左右端點該回合即結束。



- 程式：RL_15_01_simple_game.py。

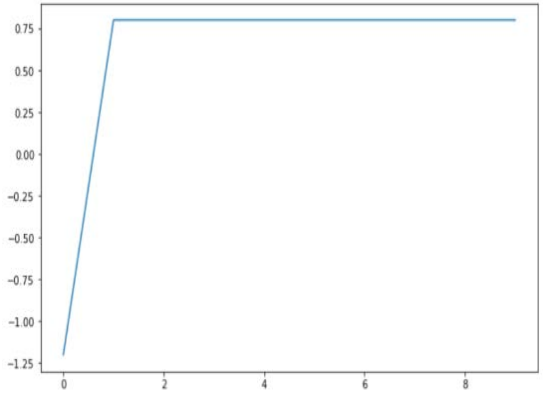
實作2. 重複實驗

- 呼叫RL_15_01_simple_game.py，執行10回合，觀察結果。
- 程式：RL_15_02_simple_game_test.py。

```
累計報酬:  -1.2000
累計報酬:  -1.6000
累計報酬:  0.8000
累計報酬:  -1.2000
累計報酬:  0.4000
累計報酬:  -1.2000
累計報酬:  -1.2000
累計報酬:  -1.2000
累計報酬:  -1.6000
累計報酬:  0.4000
```

實作3. 貪婪行動策略

- 以狀態值函數最大者為行動依據，執行10回合，觀察結果。



- 程式：RL_15_03_simple_game_with_state_value.ipynb。

結論

- 以狀態值函數最大者為行動依據，訓練模型，果然可以找到最佳解。
- 模型就是每個狀態的值函數，之後實際上線時即可載入模型執行。

Gym套件

Gym套件

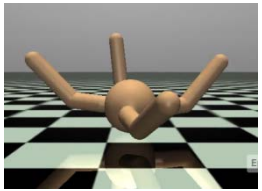
- OpenAI開發的學習套件，提供各式各樣的環境，供大家實驗，也能藉由動畫來展示訓練過程。
- Gym官網：https://gym.openai.com/
- 安裝：pip install gym
  - 安裝全部遊戲：pip install -e '.[all]'
  - 只能在Linux環境下執行
- 原始程式碼：https://github.com/openai/gym

Atari遊戲

- Windows作業環境下僅能加裝Atari遊戲。
- 安裝：pip install -e '.[atari] '
- Atari為1967年開發的遊戲機，擁有幾十種的遊戲，例如打磚塊(Breakout)、桌球(Pong)、…等

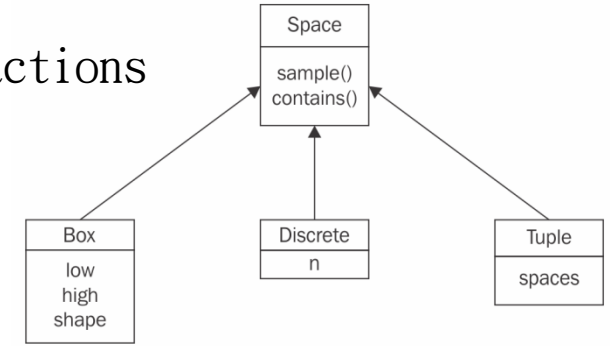
Gym提供的環境

- 經典遊戲(Classic control)和文字遊戲(Toy text)。
- 演算法類(Algorithmic)：像是多位數的加法、反轉順序等，這對電腦來說非常簡單，但使用強化學習方式求解的話，是一大挑戰。
- Atari：Atari遊戲機內的一些遊戲。
- 2D and 3D 機器人(Robot)：機器人模擬環境，有些是要付費的，可免費試用30天。可惜在Windows作業環境安裝會有問題，還好網路上有些文章有提到解決方案，需要的讀者可以google一下或參考『Install OpenAI Gym with Box2D and Mujoco in Windows 10』。
  - 多關節機器人(MuJoCo)。



Gym 遊戲設計

- 遊戲設計
  - Action space：discrete and continuous actions
  - Observation space
  - 上述兩者空間的類別如右圖。
    - Discrete：0~n-1
    - Box：
      Box(low=0, high=255,shape=(210, 160, 3), dtype=np.uint8)
    - Tuple：複雜的結構，例如車子
      steering wheel angle, brake pedal position, and accelerator, pedal position.



實作

- RL_15_04_Gym.ipynb
- RL_15_05_CartPole.ipynb
- RL_15_06_Action_Wrapper_test.py
- RL_15_07_Record_test.py

動態規劃(Dynamic Programming)

- 將大問題切分成小問題，然後逐步解決每個小問題，由於每個小問題都很類似，因此整合起來就能解決大問題。
- 譬如『費波那契數列』（Fibonacci），其中 $F_n=F_{n-1}+F_{n-2}$，要計算整個數列的話，可以設計一個$F_n=F_{n-1}+F_{n-2}$函數(小問題)，以遞迴的方式完成整個數列的計算(大問題)。

實作：費波那契數列計算

- RL_15_08_Fibonacci_Calculation.ipynb

動態規劃 + 強化學習

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right]$$

- 上述的行動轉移機率(π)、狀態轉移機率(P)均為已知，亦即環境是明確的(deterministic)，我們就可以利用Bellman方程式計算出狀態值函數、行動值函數，以反覆的方式求解，這種解法稱為『動態規劃』(Dynamic Programming, DP)。

策略評估 vs. 策略改善

- 策略評估(Policy Evaluation)
  - 當玩家走完一回合後，就可以更新所有狀態的值函數，這就稱為策略評估，即將所有狀態重新評估一次，也稱為『預測』(prediction)。

- 策略改善(Policy Improvement)：依照策略評估的最新狀態，採取最佳策略，以改善模型，也稱為『控制』(control)。
  - 通常都會依據最大的狀態值函數行動，我們稱之為『貪婪』(Greedy)策略

策略循環(Policy Iteration)

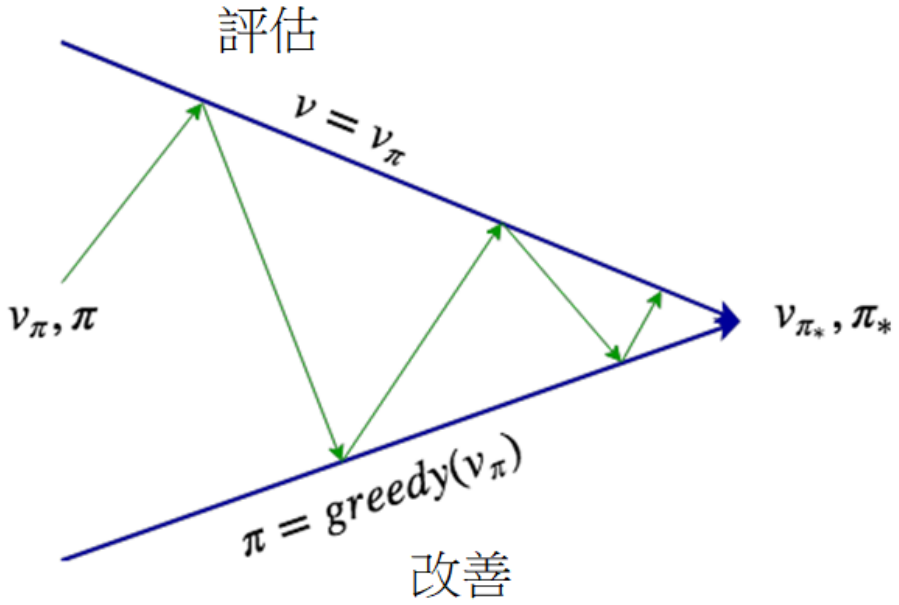


圖 a.

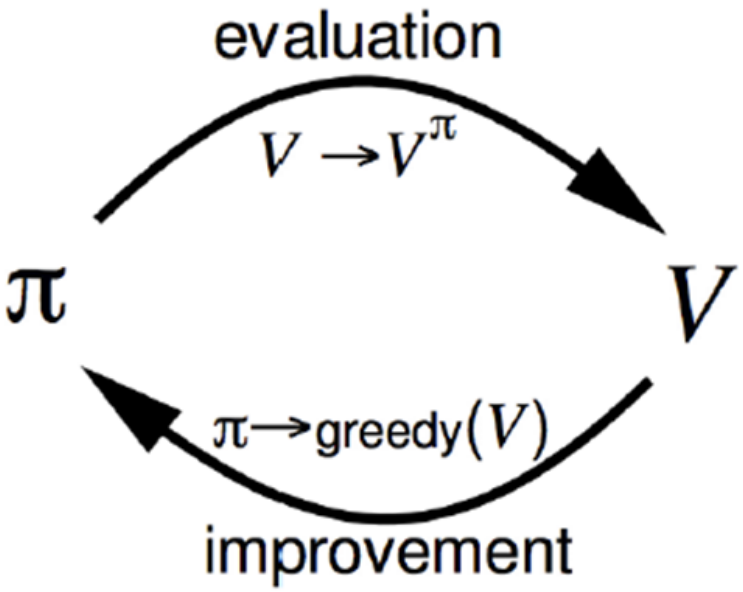


圖 b.

值循環(Value Iteration)

- 策略循環在每次策略改善前，必須先作一次策略評估，執行迴圈，更新所有狀態值函數，直至收斂，非常耗費時間。

- 考量到狀態值函數與行動值函數的更新十分類似，乾脆將其二者合併，以改善策略循環的缺點。

實作：動態規劃

- RL_15_09_Policy_Evaluation.ipynb
- RL_15_10_Policy_Iteration.ipynb
- RL_15_11_Value_Iteration.ipynb

動態規劃優缺點

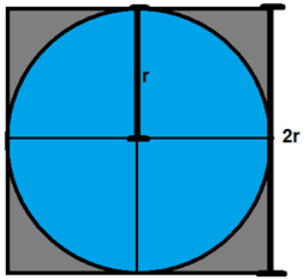
- 適合定義明確的問題，即策略轉移機率、狀態轉移機率均為已知的狀況。
- 適合中小型的模型，狀態空間不超過百萬個，比方說圍棋，狀態空間=3^{19x19}=$1.74x10^{172}$，狀態值函數更新就會執行太久。
  - 另外，可能會有大部份的路徑從未走過，導致樣本代表性不足，進而造成維數災難(Curse of Dimensionality)。

蒙地卡羅(Monte Carlo)

- 玩遊戲時，通常不會知道策略轉移機率、狀態轉移機率，這稱為無模型(Model Free)學習。
- 在這樣的情況下，動態規劃就無法派上用場。
- 蒙地卡羅(Monte Carlo, MC)演算法：透過模擬的方式估計轉移機率。

範例. 以蒙地卡羅演算法求圓周率(π)

- 程式：RL_15_11_Value_Iteration.ipynb。
- 在正方形的範圍內隨機產生亂數一千萬個點，計算落在圓形內的點數。
  - 落在圓形內的點數 / 一千萬 ≒ $\pi r^2 / 4r^2 = \pi/4$。
  - 化簡後π = 4 x（落在圓形內的點數）/ 一千萬。



延伸

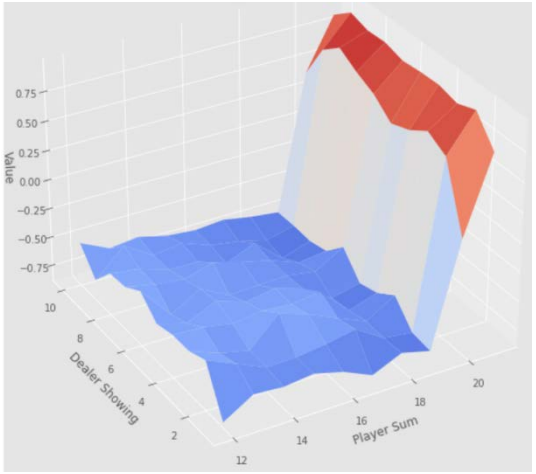
- 假如轉移機率未知，我們也可以利用蒙地卡羅演算法估計轉移機率，利用隨機策略去走迷宮，根據結果計算轉移機率。

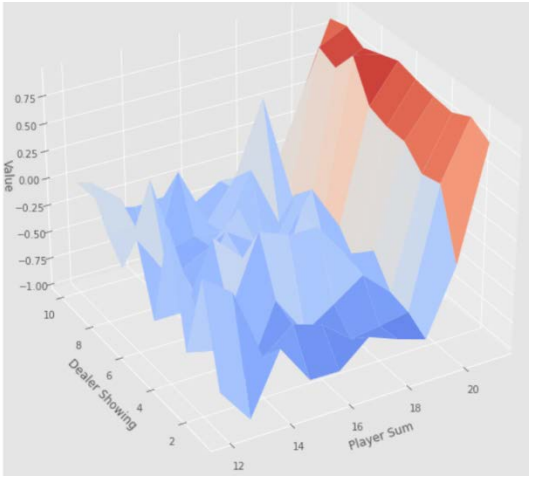
範例. 21點樸克牌 (Blackjack)

- 遊戲規則：
  https://zh.wikipedia.org/wiki/二十一點。
- 實驗21點樸克牌(Blackjack)之策略評估。
  - 程式： RL_15_13_Blackjack_Policy_Evaluation.ipynb
  - 分別實驗10,000與500,000回合。

執行結果

- 比較10,000(左)與500,000（右)回合結果。
- 分成持有A(比較容易獲勝)及未持有A兩種狀況比較。



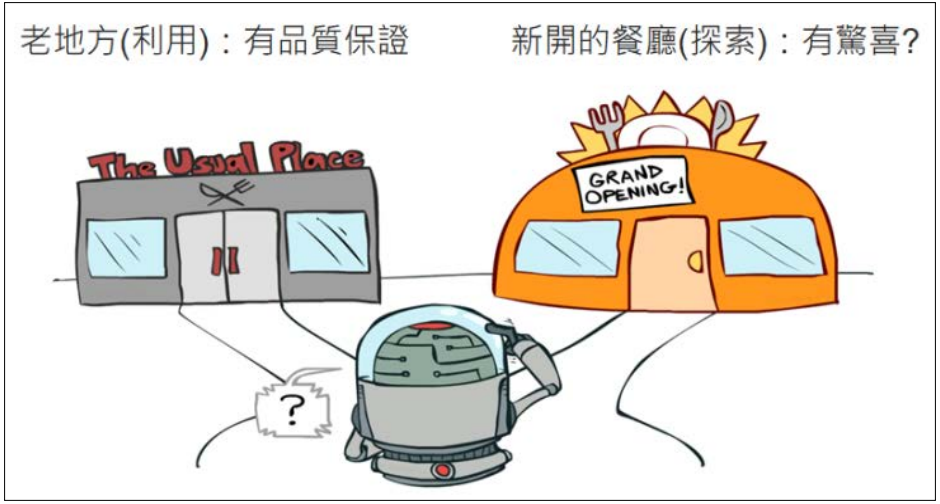


值循環

- 策略改善改採 ε -greedy 策略。
- 採用貪婪(greedy)策略有弱點：
  - 一旦發現最大值函數的路徑後，貪婪策略會一直走相同的路徑，這樣便失去了找到更好路徑的潛在機會。
  - 舉例來說，家庭聚餐時，都會選擇最好的美食餐廳，若為了不踩雷，每次都去之前最好吃的餐廳用餐，那新開的餐廳就永遠沒機會被發現了。

探索與利用 (Exploration and Exploitation)

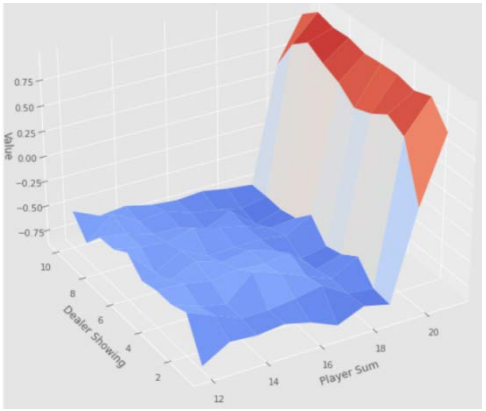
- ε-greedy：保留一個比例去探索，不走既有的老路。



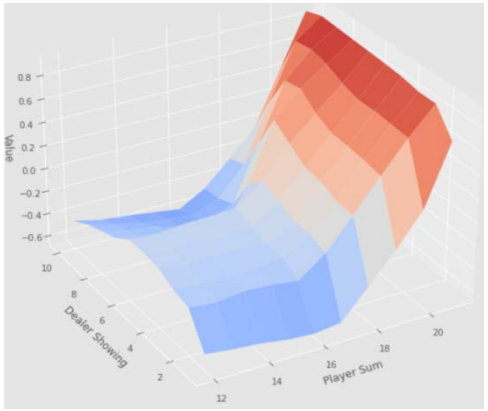
範例. 21點樸克牌(Blackjack)之值循環

- 程式：RL_15_14_Blackjack_Value_Iteration.ipynb
- 上個範例只有當玩家的分數接近20分的時候，值函數特別高。
- 在低分時也有不差的表現。

策略評估



值循環



On-policy vs. Off-policy

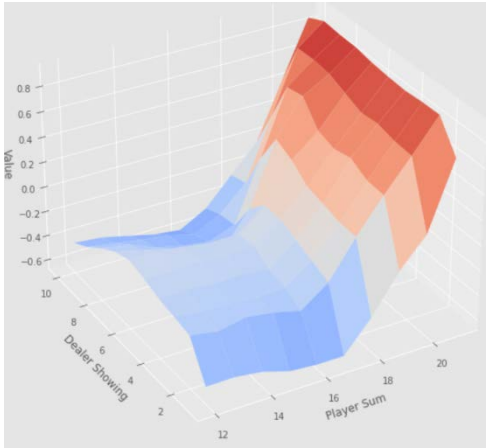
- On-policy：策略評估與策略改良上，均採用同一策略，例如 ε-greedy。

- Off-policy：策略評估與策略改良採用不同策略，策略評估時採用隨機策略，盡可能走過所有路徑，在策略改良時，改採貪婪策略，盡量求勝。

範例. 21點樸克牌(Blackjack) Off-policy值循環

- 程式： RL_15_15_Blackjack_Off_Policy.ipynb
- 執行結果：玩家分數在低分時勝率也明顯提升。

Off-policy 值循環



蒙地卡羅演算法缺點

- 每個回合必須走到終點，才能夠倒推每個狀態的值函數。
- 假使狀態空間很大的話，還是一樣要走到終點，才能開始下行動決策，速度實在太慢，例如圍棋。
- 根據統計，每下一盤棋平均約需150手，而且圍棋共有$3^{19x19} \fallingdotseq 1.74x10^{172}$個狀態，就算使用探索也很難測試到每個狀態，計算值函數。

時序差分(Temporal Difference)

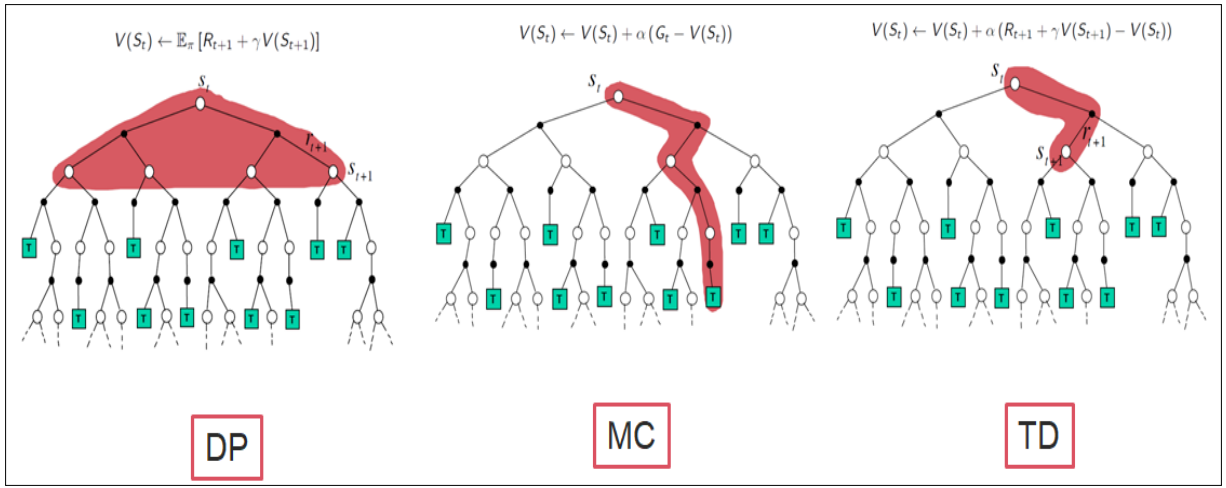
- 邊走邊計算值函數。

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

- 值函數每次加上下一狀態值函數與目前狀態值函數的差額，以目前的行動產生的結果代替Bellman公式的期望值，另外，再乘以學習率(Learning Rate)α。
  - 走一步更新一次的作法稱為TD(0)。
  - 走n步更新一次的作法稱則為TD(λ)。

倒推圖

- 動態規劃：逐步搜尋所有的下一個可能狀態，計算值函數期望值。
- 蒙地卡羅：試走多個回合，再以回推的方式計算值函數期望值。
- 時序差分：每走一步更新一次值函數。

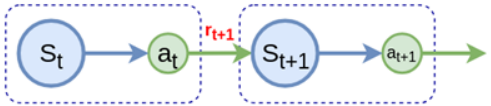


時序差分種類

- 時序差分有兩類演算法：
  - SARSA演算法：On Policy的時序差分。
  - Q-learning演算法：Off Policy的時序差分。

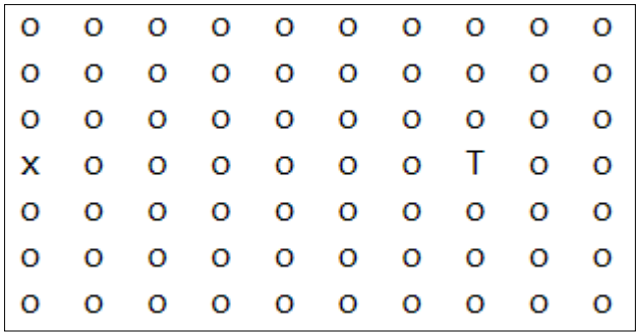
SARSA演算法

- 行動軌跡中5個元素的縮寫s_t，a_t，r_{t+1}，s_{t+1}，a_{t+1}。
- 如下圖，意謂著每走一步更新一次。



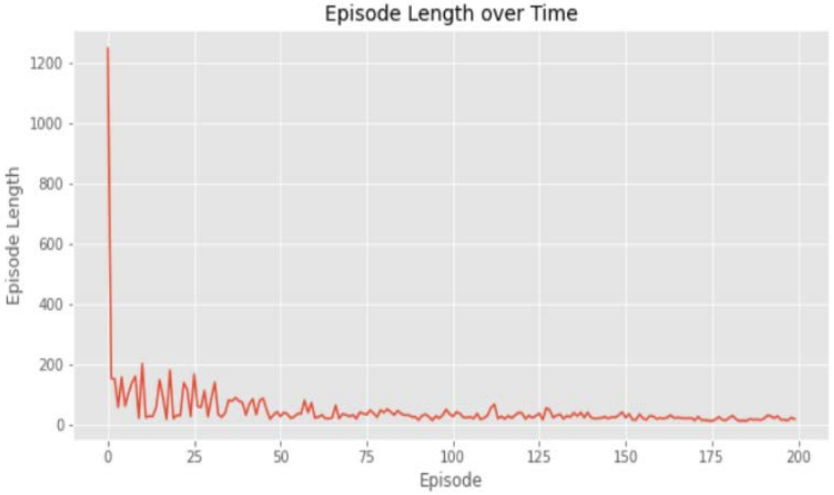
Windy Gridworld 遊戲規則

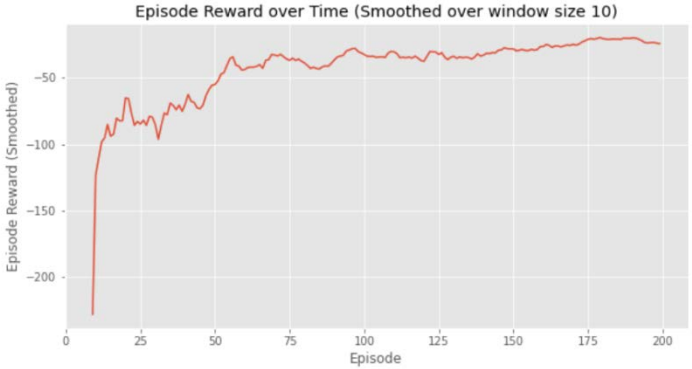
- 第4, 5, 6, 9行的風力1級，第7, 8行的風力2級，會把玩家往上吹1格和2格。



範例. 實驗Windy Gridworld之SARSA策略

- 程式：RL_15_16_SARSA.ipynb。
- 約第50回合後就逐漸收斂了。
- 每回合獲得的報酬越來越高。





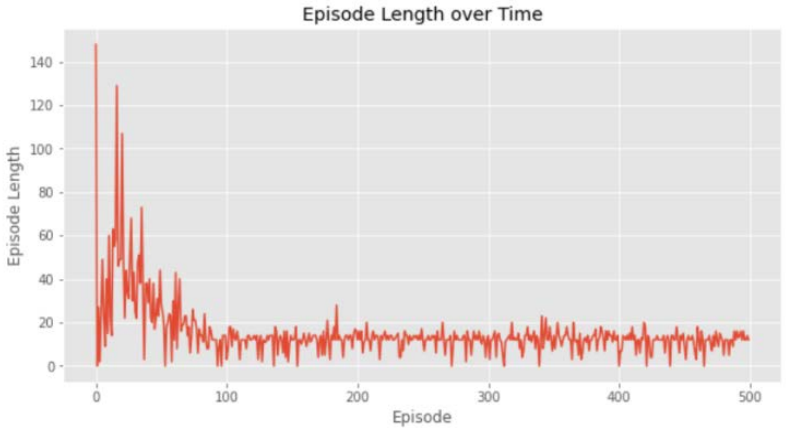
Cliff Walking 遊戲規則

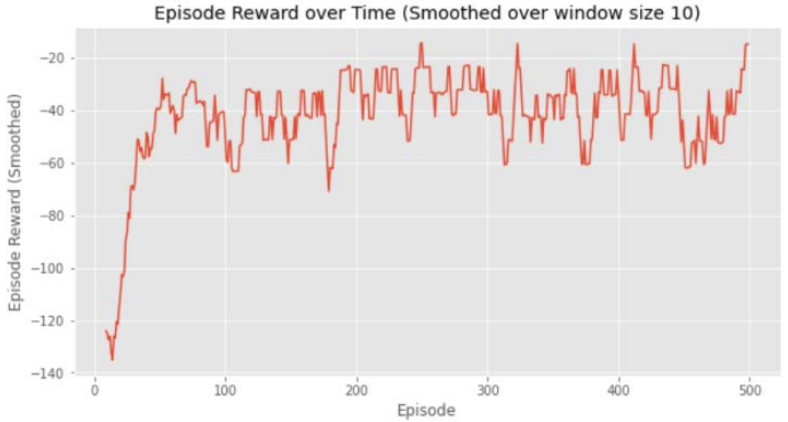
- 最下面一排除了起點(x)與終點(T)之外，其他都是陷阱(C)，踩到陷阱即Game Over。

o	o	o	o	o	o	o	o	o	o	o	o
o	o	o	o	o	o	o	o	o	o	o	o
o	o	o	o	o	o	o	o	o	o	o	o
x	C	C	C	C	C	C	C	C	C	C	T

範例. 實驗Cliff Walking之Q-learning策略

- 評估時使用 ε-greedy 策略，改良時選擇greedy 策略。
- 程式：RL_15_17_Q_learning.ipynb
- 執行到大概第100回合就逐漸收斂。
- 每回合獲得的報酬越來越高，不過尚未收斂。

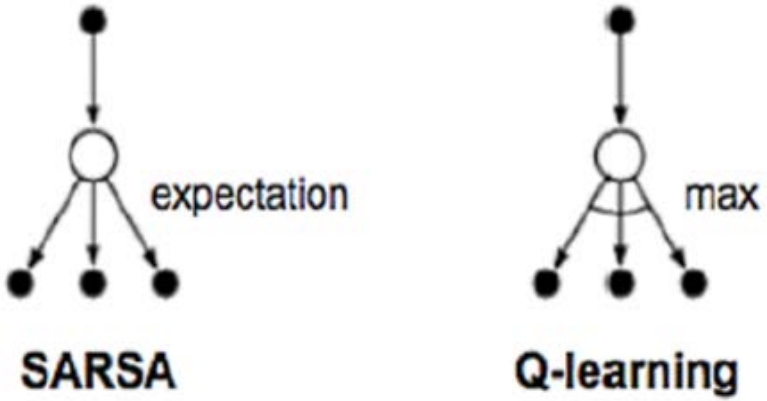




SARSA與Q-learning比較

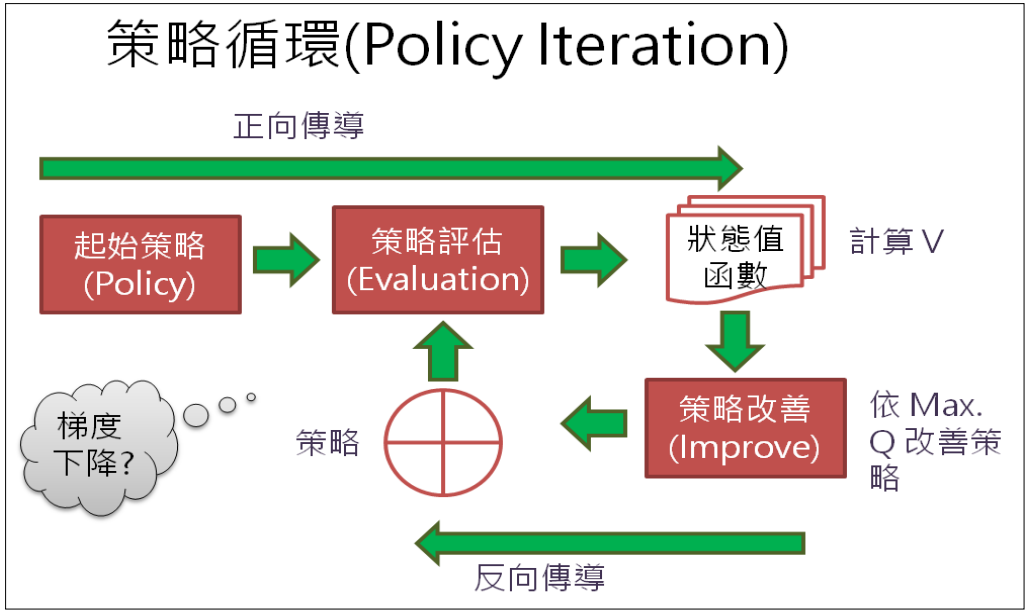
Sarsa : $q(s_t, a_t) = q(s_t, a_t) + \alpha[r_{t+1} + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t)]$

Q-learning : $q(s_t, a_t) = q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a q(s_{t+1}, a) - q(s_t, a_t)]$



強化學習流程

邏輯與神經網路優化求解，其實有那麼一點相似。



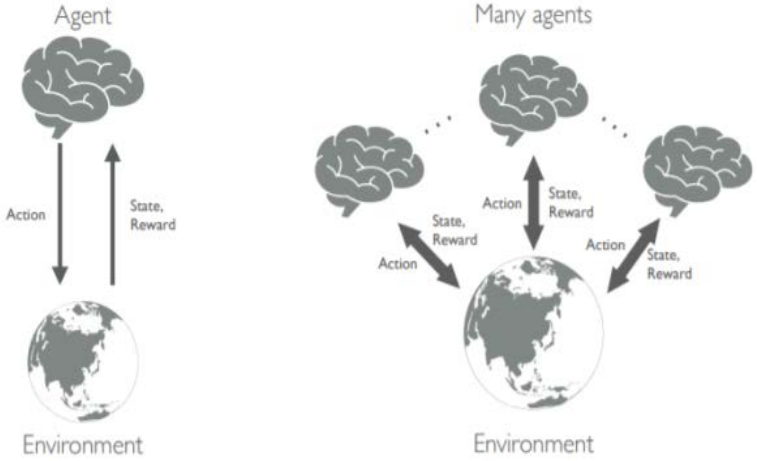
其他演算法 -- 連續型變數

- 前述的演算法都是建立一個陣列(List)，來記錄所有的對應值，之後就從陣列中選擇最佳的行動，所以這類演算法被稱為『表格型』(Tabular)強化學習，但只適合離散型的狀態，像木棒台車這種的連續型變數，就不適用。

- 變通的作法有兩種：
  - 將連續型變數進行分組，轉換成離散型變數。
  - 使用機率分配或神經網路模型取代表格，以策略評估的訓練資料，來估計模型的參數(權重)，選擇行動時，就依據模型推斷出最佳預測值，而Deep Q-learning(DQN)即是利用神經網路的Q-learning演算法。

多玩家強化學習 (Multi-agent Reinforcement Learning)

- 多位玩家同時參與一款遊戲，這時就會產生協同合作或互相對抗的情境，玩家除了考慮獎勵與狀態外，也需觀察其他玩家的狀態。
- 許多撲克牌遊戲都屬於多玩家遊戲。



各種演算法

Algorithm	Description	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Either	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State–action–reward–state with eligibility traces	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Off-policy	Continuous	Continuous	Advantage

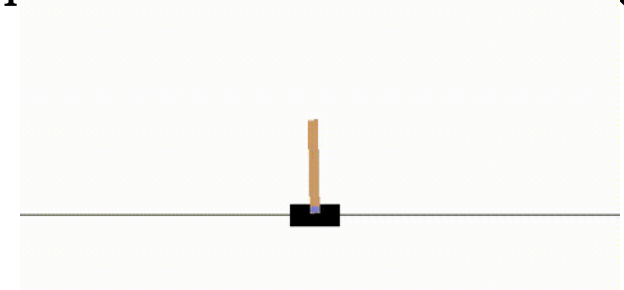
https://en.wikipedia.org/wiki/Reinforcement_learning

實戰練習(1)

- 井字遊戲(TicTacToe)
  - TicTacToe_1\ticTacToe.py

實戰練習(2)

- 木棒台車(CartPole)
- 遊戲規則：
  - 可控制台車往左 or 往右
  - 平衡桿一開始是直立(upright)，要保持它在行駛中仍然保持平衡
  - 遊戲結束要件，二擇一
    - 平衡桿偏差15℃ ➔ 敗
    - 離中心點 2.4 單位即勝
  - 行動後平衡桿保持直立 ➔ reward += 1



原始程式碼

- https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py

```
Observation:
    Type: Box(4)
    Num     Observation               Min         Max
    0       Cart Position             -4.8          4.8
    1       Cart Velocity             -Inf          Inf
    2       Pole Angle                 -24 deg      24 deg
    3       Pole Velocity At Tip      -Inf          Inf

Actions:
    Type: Discrete(2)
    Num     Action
    0       Push cart to the left
    1       Push cart to the right

    Note: The amount the velocity that is reduced or increased is not fixed; it depends on the angle the pole is pointing. This is beca

Reward:
    Reward is 1 for every step taken, including the termination step

Starting State:
    All observations are assigned a uniform random value in [-0.05..0.05]

Episode Termination:
    Pole Angle is more than 12 degrees
    Cart Position is more than 2.4 (center of the cart reaches the edge of the display)
    Episode length is greater than 200
    Solved Requirements
    Considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.
```

行動者與評論者(Actor Critic)演算法

- 木棒台車(CartPole)
  - RL_15_18_Actor_Critic.py
  - 使用『行動者與評論者』(Actor Critic)演算法。
    - Actor Critic類似GAN，主要分為兩個神經網路，Actor(行動者)在評論者(Critic)的指導下，優化行動決策，而評論者則負責評估行動決策的好壞，並主導值函數模型的參數更新，詳細的說明可參閱『Keras 官網說明』。

動畫

- 官網也提供兩段錄製的動畫，分別為訓練初期與後期的比較，可以看出後期的木棒台車行駛得相當穩定。
  - 訓練初期：https://i.imgur.com/5gCs5kH.gif。
  - 訓練後期：https://i.imgur.com/5ziiZUD.gif。

參考書籍

Reinforcement Learning: An Introduction