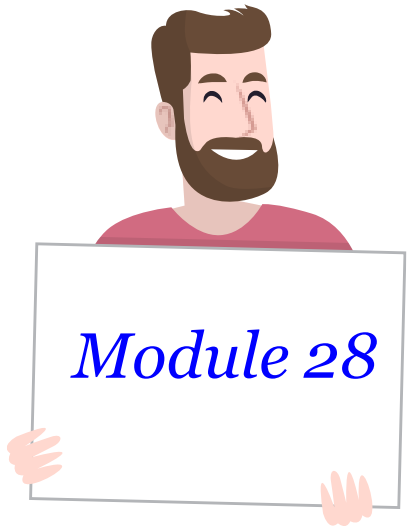
常見強化學習演算法



designed by freepik

Estimated time:
45 min.

學習目標

- 28-1:馬可夫決策過程
- 28-2:Q-Learning
- 28-3:DQN

28-1:馬可夫決策過程

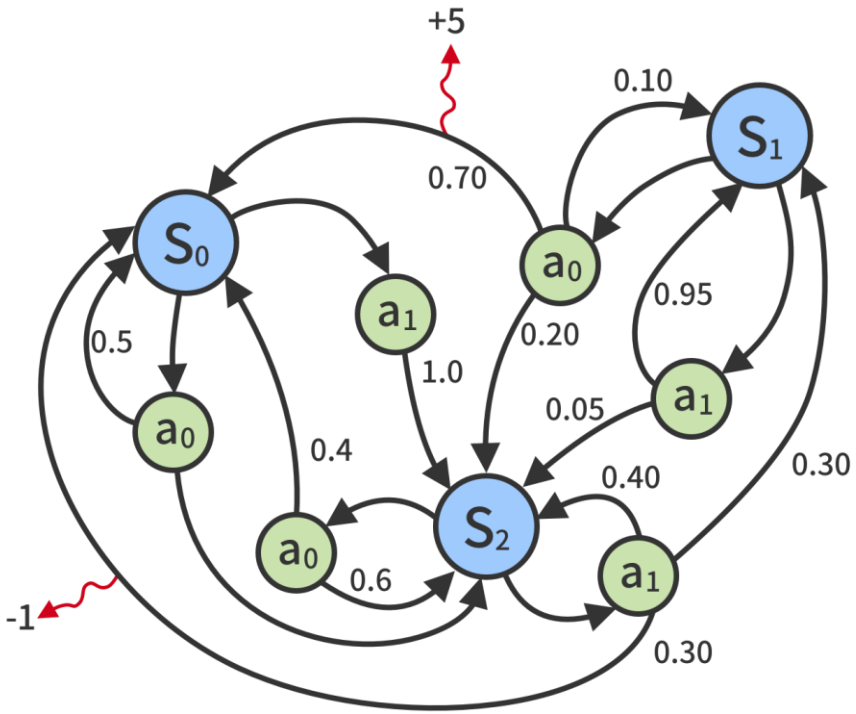
- 馬可夫決策過程介紹
- 馬可夫決策的目標
- 貝爾曼方程式
- 馬可夫決策過程與強化學習

馬可夫決策過程介紹

- 馬可夫決策過程一種離散隨機控制過程
  - 常被用來做決策管理
  - 幾乎所有強化學習問題都可以被視為馬可夫決策過程

馬可夫決策過程介紹

- 一個馬可夫決策過程包含五個元素(S, A, P_a, R_a, γ)
  - S是狀態集、 A是動作集、 P_a是狀態s下做a這個動作的轉移機率、 R_a是狀態s下做a這個動作所得到的賞酬、γ是折現因子(通常介於0~1之間)



馬可夫決策過程介紹

- 為了簡化問題，馬可夫決策過程常常會假設未來只跟現在有關，因為現在已經充分反映過去歷史
  - 這個假設叫做”馬可夫特性”

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, S_2, S_3, \ldots\ldots, S_t)$$

馬可夫決策的目標

- 馬可夫決策的目標
  - 尋找一個policy使得未來累積賞酬最大

$$G_t = R_{t+1} + R_{t+2} + \ldots$$

$$G_t = \sum_{k=0}^{T} R_{t+k+1}$$

馬可夫決策的目標

- 我們一樣可以使用折現因子來讓未來累積賞酬可以收斂
  - 這個跟之前章節提到的理由一樣

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} where \gamma \in [0, 1)$$

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots$$

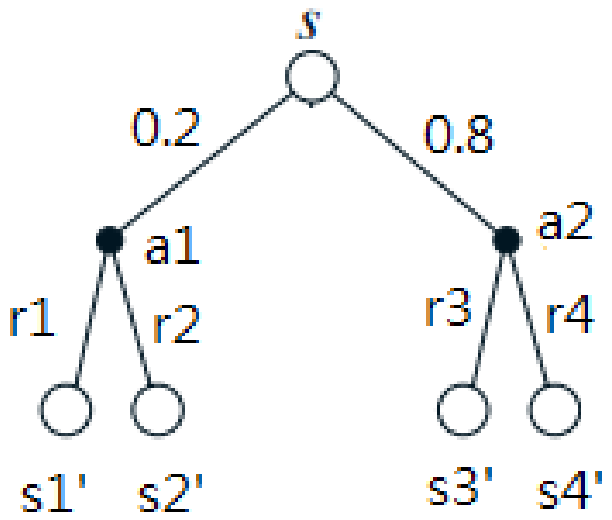
貝爾曼方程式

- 貝爾曼方程式是解決馬可夫決策的數學式子
  - 這個數學式子推導出兩個價值函數之間的關係
  - 可以用這樣一個關係式子反推出馬可夫決策任何一個想求的數值

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_\pi(s')\right], \qquad (3.10)
\end{aligned}
$$

貝爾曼方程式

- 下面是一個貝爾曼方程式的範例



$$v(s) = 0.2 * 0.5 * (r1 + v(s1')) + 0.2*0.5*(r2 + v(s2'))$$
$$+0.8*0.5*(r3 + v(s3')) + 0.8*0.5*(r4 + v(s4'))$$

馬可夫決策過程與強化學習

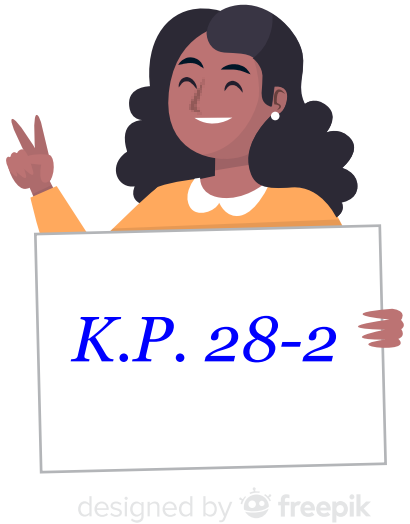
- 通常在強化學習問題裡，會缺少馬可夫決策過程的P_a以及R_a
  - 變成我們必須要用其他方法去計算這兩個數值，才能將問題解決出來
  - 也是為什麼我們無法常用貝爾曼方式式直接去解決強化學習問題

$$(S, A, P_a, R_a, \gamma)$$

？？

28-2: Q-Learning

- Q-Learning介紹
- Q-Learning演算法
- Q-Learning範例

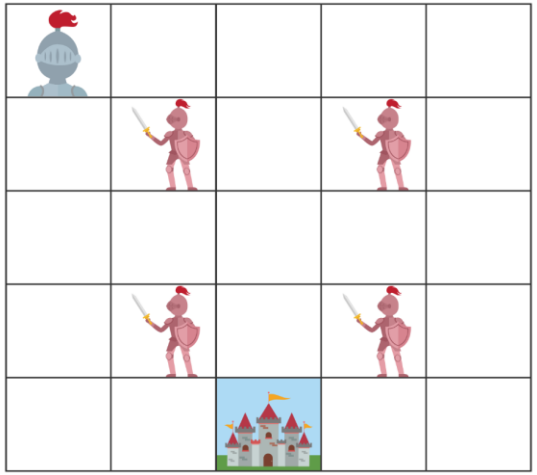


Q-Learning介紹

- Q-Learning是強化學習的一種演算法
  - 它是value-based的方法
- Q值代表此函數的賞酬
  - Q是在狀態s下做某a動作的quality

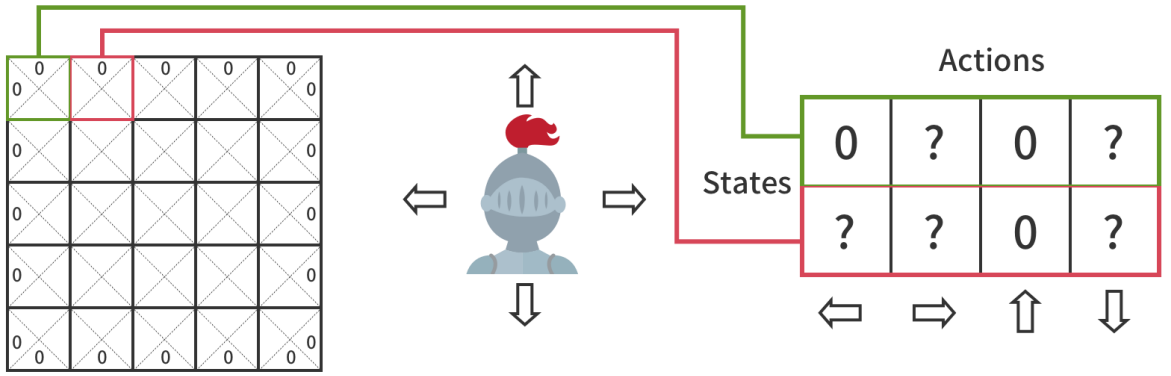
Q-Learning介紹

- 假設我們有一個遊戲是要讓騎士回城堡，這個遊戲規則如下
  - 騎士每走一步會損失1點(確保騎士用最快的速度回城堡)
  - 如果遇見敵人會損失100點
  - 如果順利成功進入城堡，會得正100點



Q-Learning介紹

- 在每個狀態下，我們動作的選擇有四個(左、右、上、下)
- 我們建立一個Q-table，這個Q-table會記錄每個狀態下未來最大期望累積賞酬
- 可以把Q-table當作是一個查表，之後可以輔助我們做決策



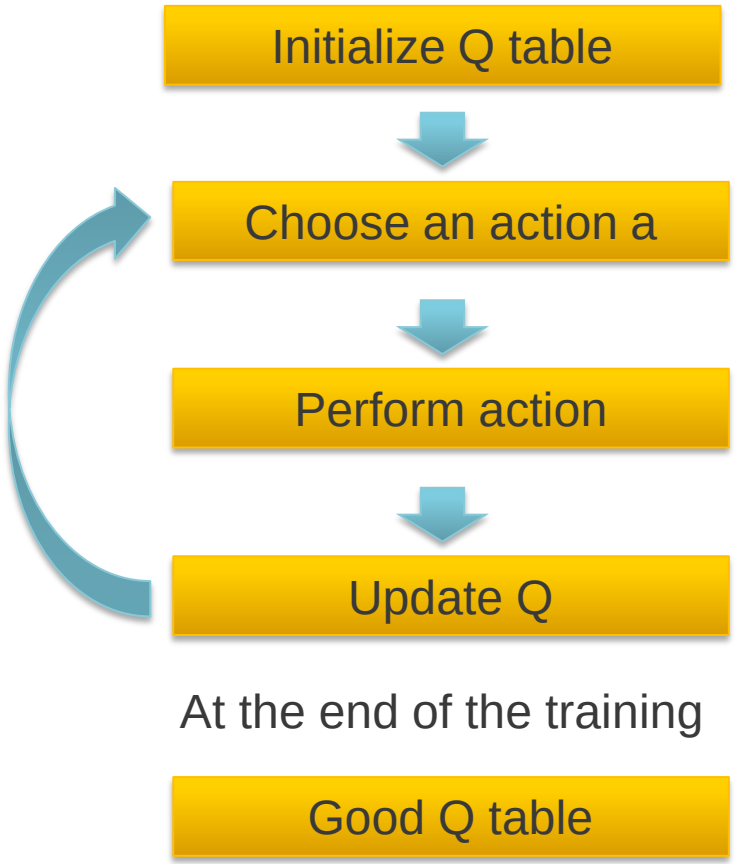
Q-Learning介紹

- **Q function的輸入有兩個，狀態跟動作**
  - **其輸出的值代表未來期望累積賞酬**

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | s_t, a_t]$$

Q-Learning演算法

- **Q-Learning演算法整體流程如下**



Q-Learning演算法

- 步驟1：一開始Q-Learning會先將Q table初始化
  - 所有狀態以及對應的動作都設定成0

States	Actions			
	0	0	0	0
	0	0	0	0
	0	0	0	0
	■	■	■	
	0	0	0	0

Q-Learning演算法

- 步驟2：重複步驟3~步驟5
  - 直到我們達到終止的條件(使用者可以設定)
- 步驟3：在當下狀態為s的情況下，根據Q-table去做一個動作a
  - 剛開始智能體還很笨，所以需要有Exploitation以及Exploration的機制去輔助做動作a

Q-Learning演算法

- **Exploitation以及Exploration機制**
  - **Exploitation表示使用現有的知識去做決策，適合當智能體越來越聰明的情況下所使用的**
  - **Exploration表示隨機作決策，適合當智能體什麼都不會的情況下所使用**
- **一個好的決策策略通常會混和Exploitation以及Exploration機制使用**
  - **有時候長遠最好的策略，在短期可能要做點犧牲**

Q-Learning演算法

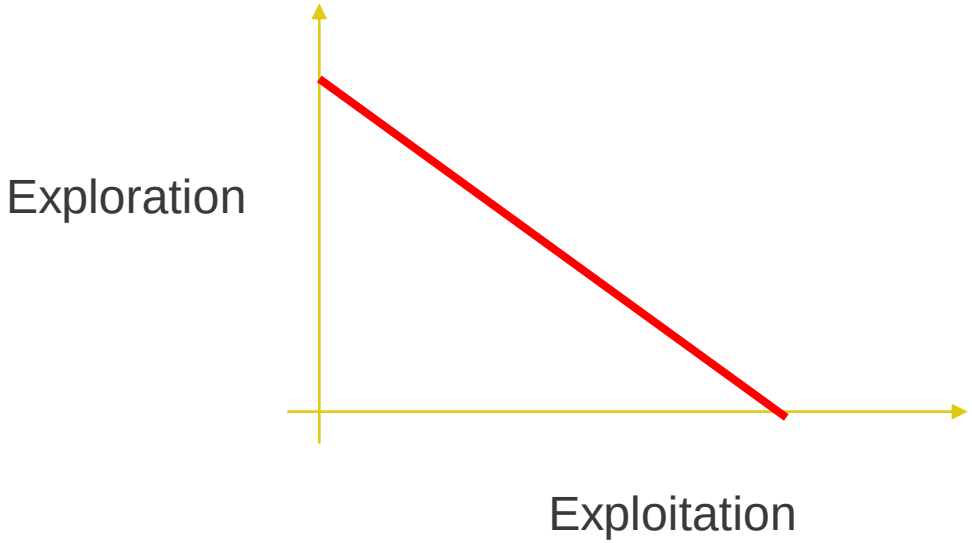
- Exploitation以及Exploration範例
- 假設我們要選擇中午要吃什麼
  - Exploitation的方式是找之前吃過最好吃的那家
  - Exploration的方式是去嘗試新的餐廳
- Exploration有時候能發現更好的餐廳，雖然也有風險找到比原本差的

Q-Learning演算法

- 回到步驟3，智能體一開始很笨，所以我們希望Exploration的比例高一下(隨機做決策)，但當智能體越來越聰明的時候，我們希望Exploitation的比例變高
  - 常見實作此概念的函數有$\epsilon - Greedy$函數，即有一部分的機率去隨機做動作，一部分的機率去使用現有的知識做動作

ϵ–Greedy

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$



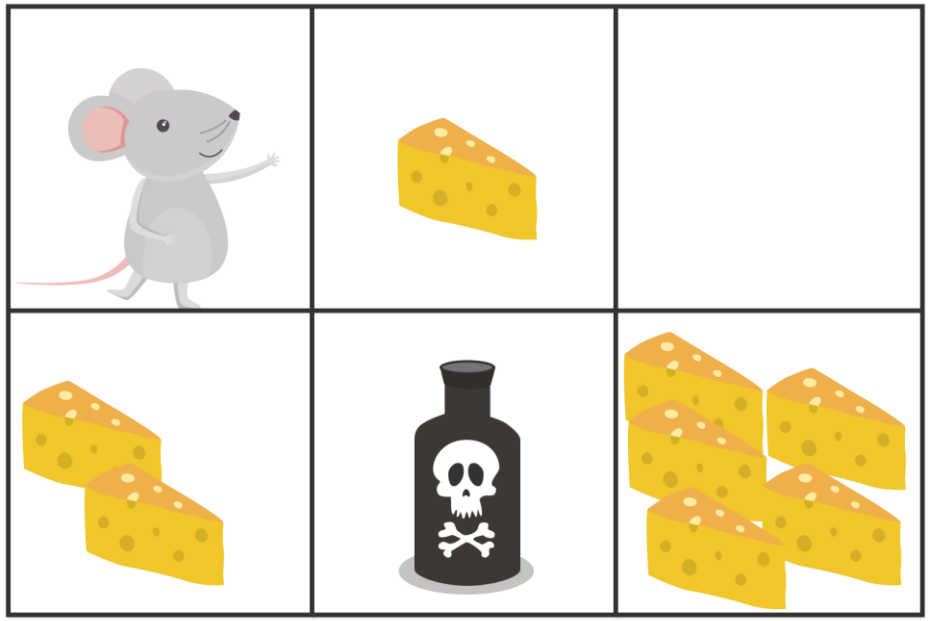
Q-Learning演算法

- **步驟4：執行動作a並獲得新的狀態s'以及賞酬r**
- **步驟5：更新Q(s,a)**
  - 使用以下式子更新

$$New\ Q(s,a) = Q(s,a) + \alpha\left[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)\right]$$

Q-Learning範例

- 假設我們想要讓老鼠去吃起司
  - 吃到一個起司得+1分、吃到兩個起司得+2分
  - 吃到一堆起司得+10分並結束遊戲
  - 吃到毒藥得-10分並結束遊戲



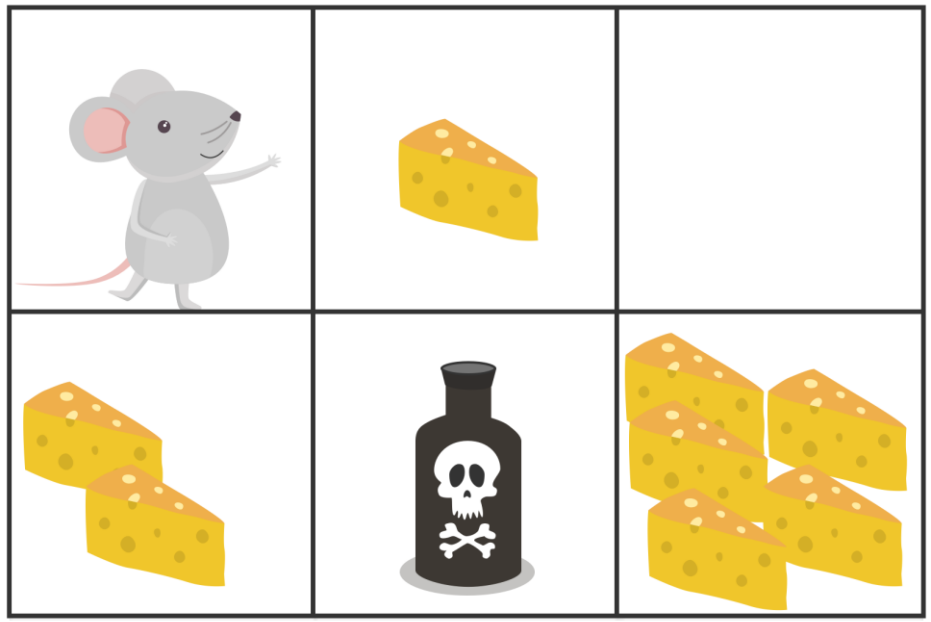
Q-Learning範例

- 一開始先隨機初始化Q-table
  - 將所有狀態以及對應的動作給予0值

State	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Q-Learning範例

- 因為一開始智能體(老鼠)還沒有很聰明，所以假設隨機選取動作(我們假設選到往右的動作)



State	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Q-Learning範例

- 我們把對應的Q值做更新
  - 更新公式如下

$$New\ Q(s,a) = Q(s,a) + \alpha\left[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)\right]$$

$$R(s,a) + \gamma \max Q'(s',a') - Q(s,a) = 1 + 0.9 * 0 \ - \ 0 = 1$$

$$New\ Q(s,a) = 0 + 0.1 * 1 = 0.1$$

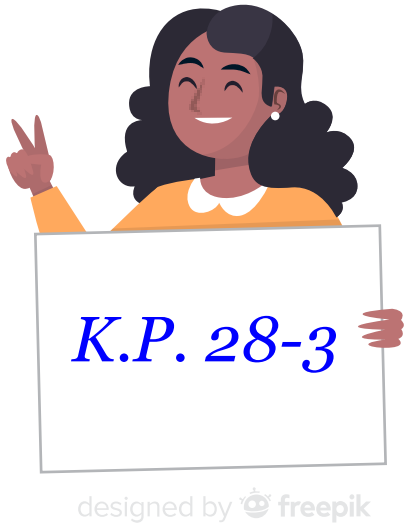
Q-Learning範例

- 這是更新玩Q值的結果，我們重複上面的步驟將Q值更新的更多
  - 更新到一定程度後，未來我們就可以根據此表來做決策，讓智能體快速找到一大堆起司的路徑

State	←	→	↑	↓
Start	0	0.1	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

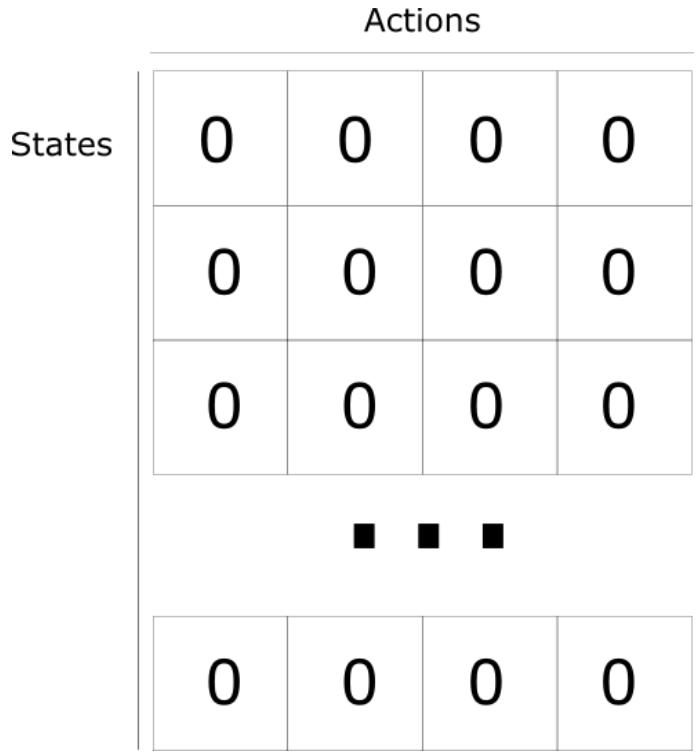
28-3: DQN

- Q-Learning不足的地方
- DQN介紹
- DQN演算法



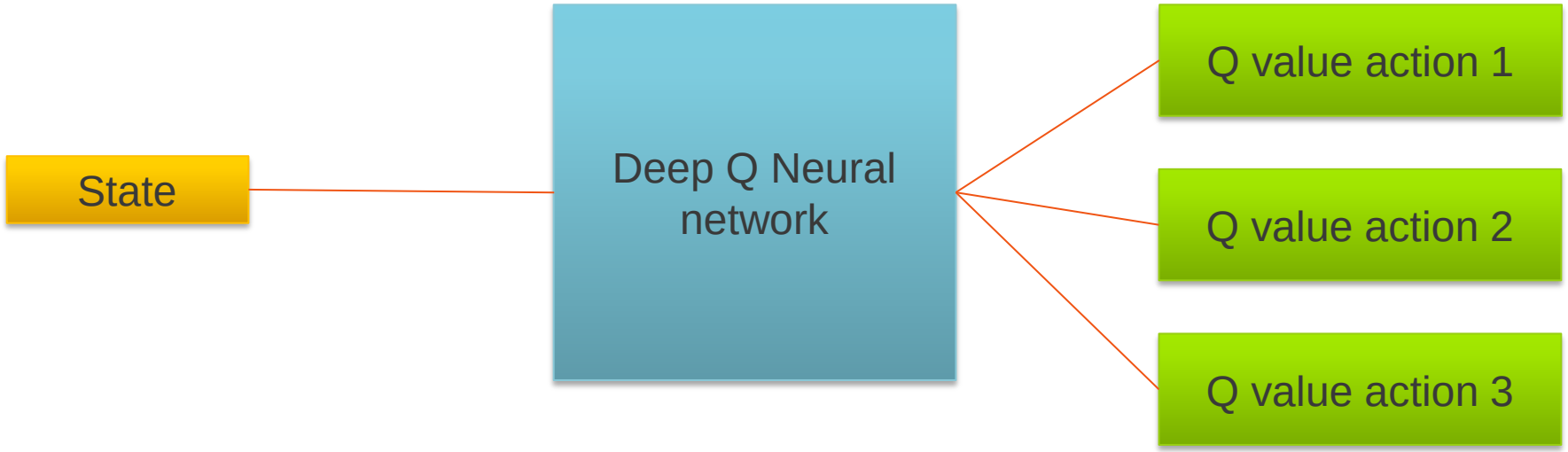
Q-Learning不足的地方

- **Q-Learning最大的問題在於它需要紀錄每一個狀態以及動作的Q-table**
  - **實際上常常Q-table會非常巨大，因此會無法使用Q-Learning演算法**



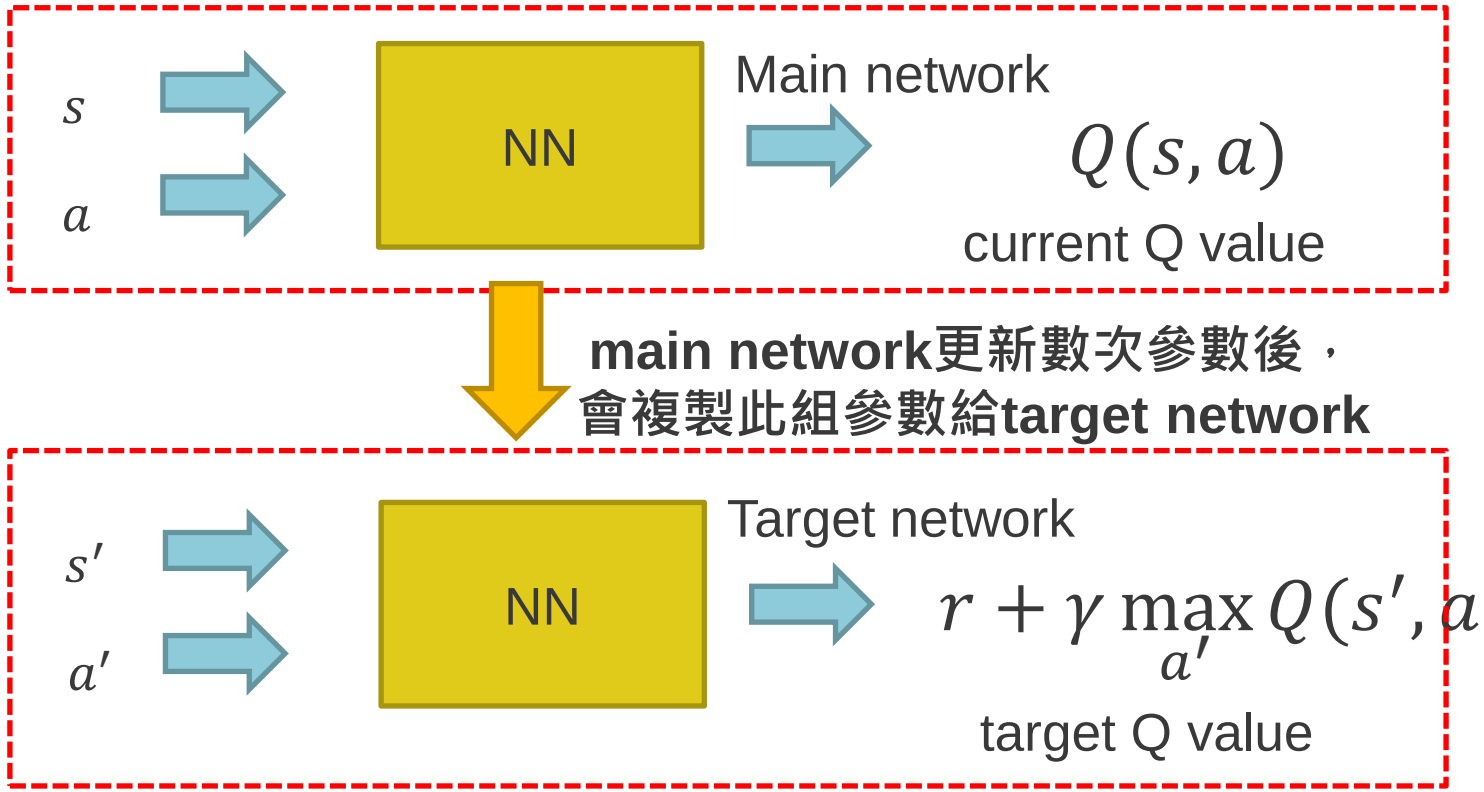
DQN介紹

- 有人就提出說是否我們有辦法用一個神經網路去預測Q-table裡面的數值
  - 這就是DQN最原始的想法
  - 也是強化學習與深度學習融合在一起的一個演算法



DQN演算法

- DQN的作法是產生兩個一樣架構的神經網路
  - 一個叫做main network、一個叫做target network
  - main network主要預測當下Q值，target network預測目標Q值



DQN演算法

- 以下為DQN的損失函數
  - 其中當下Q值是由main network所預測
  - 目標Q值是由target network所預測

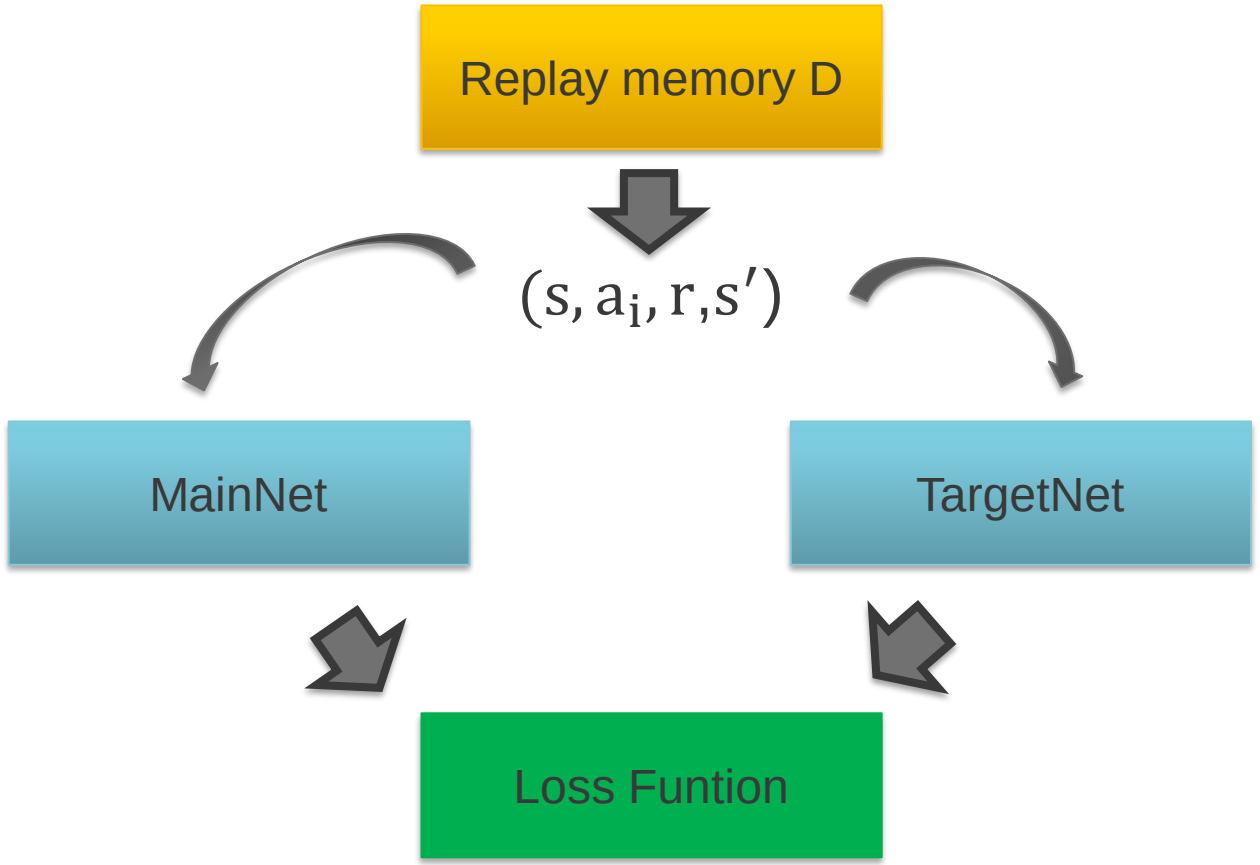
$$L(\theta) = E[(R + \gamma \max Q(s', a'; \theta) - Q(s, a; \theta))^2]$$

DQN演算法

- 為了要讓DQN網路它可以真的學到一個好的policy，它會使用一個叫做experience replay的技巧來讓網路重複去學習過往看過的知識
  - 它會將過往的資料儲存成 (s_t, a_t, r_t, s_{t+1}) 的格式在記憶體當中
  - 把這些資料當批次資料去做訓練

DQN演算法

- 以下是DQN演算法的流程圖



$$L(\theta) = E[(R + \gamma \max Q(s', a'; \theta) - Q(s, a; \theta))^2]$$

Demo 28-3

- 初始化Q-table
- Q-learning基礎範例
- Q-learning進階範例

線上Corelab

- 題目1：建立Q-table
  - 完成初始化Q-table並讓Q-learning玩此遊戲可以順利執行
- 題目2：更新Q-table
  - 完成更新Q-table式子並讓Q-learning玩此遊戲可以順利執行
- 題目3：完成choose_action函數
  - 完成epsilon-Greedy函數並讓Q-learning玩此遊戲可以順利執行

本章重點精華回顧

- 馬可夫決策過程
- Q-Learning
- DQN

Lab: Q-learning

- **Lab01:初始化Q-table**
- **Lab02:Q-learning基礎範例**
- **Lab03:Q-learning進階範例**

Estimated time:
20 minutes


THANK YOU