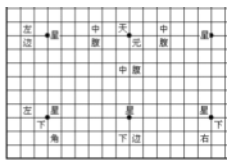
# ADL—DQN

2019年4月16日　　上午 11:06

value based
Deep-Q-Networks
評估每個state的好壞，決定選哪個動作

- value function: 評估state好還是壞，以這個state為基準未來可以獲得的期望值
  - 可以預測future reward的數值
  - Q-function去評估total reward的期望值
  - Q function去學total reward的期望值
  - S做了action在特定policy下，Q在這個state做了這個action，會得到個個時間點的reward，return會經過discount gamma

    Q-value function gives expected total reward
    - from state $s$ and action $a$
    - under policy $\pi$
    - with discount factor $\gamma$

  - $$Q^{\pi}(s,a) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a]$$

    Value functions decompose into a Bellman equation

    $$Q^{\pi}(s,a) = \mathbb{E}_{s',a'}[r + \gamma Q^{\pi}(s',a') \mid s,a]$$

  - 如果知道最後一個state是誰就可以慢慢推回來，從終點把數值填上來，往前推
  - 在s跟a之下可以讓output最大，從得到最大的value進行互動，最後得到的value便是optimal value
  - optimal value function可以讓每個時間點都做最好的action，每個state最多最好可以得到的value，每一個state最多可以得到多少value，選擇可以得到最高value的a來做，optimal policy
  - Q是future reward的期望值，有Q*就會follow最好的動作sequence來做，得到的reward就是rt+1、rt+2做這個動作最好的數值，每個時間點都可以選擇最高的reward來做

An optimal value function is the maximum achievable value

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

The optimal value function allows us act optimally

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

- The optimal value informally maximizes over all decisions

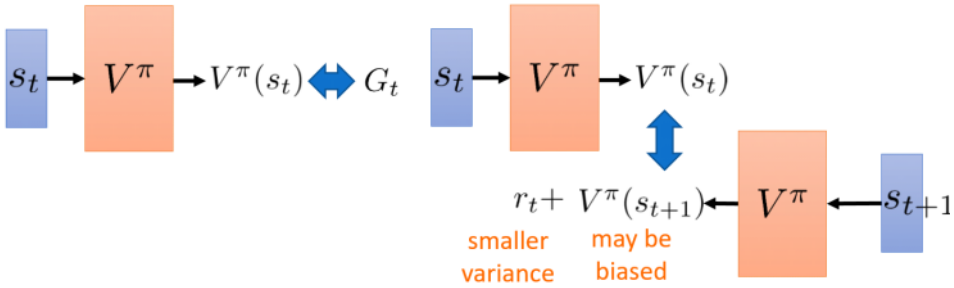
$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

Optimal values decompose into a Bellman equation

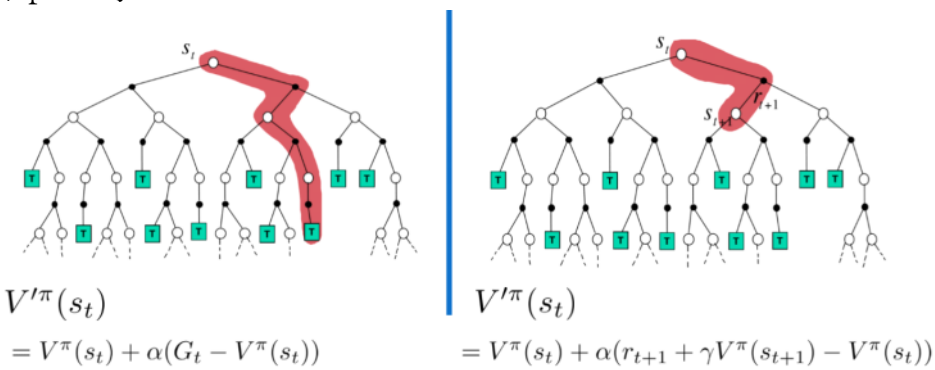
$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a') \mid s,a]$$

- value function lookup table，但是可能會有沒看過的就會找不出來，但如果是function即便當時沒看過那些可能state，也可以模擬，看到的只有其中幾個還是可以有output 數值去學大致上的變化，s跟a近來output value是多少，如果是原本的lookup table就會無法估計。function的話就可以看過一些pair就可以model出沒看過的state之數值，用network來學就是Q-Network
- Q-Network: 學習找出最好的w，訓練過程不斷update w去接近真實的Q*=>Q-learning
- 希望Q function可以一直接進target，學習希望數值可以盡量接近，從後面時間點往前推，把Q值填上去
- DL value function (critic)
  - actor有多好
  - state value function: 可以得到reward 期望值
  - 看到這個state之後再得到的加總reward要比較高
  - 可以藉由說這個state好不好，可以用function output決定要怎麼做動作
  - 學習critic，但是會影響最後的動作，critic會影響最後的policy
- Monte-Carlo (MC)
  - 估計V的一種方法，估計critic，讓這個人去看這個人在玩遊戲，一場遊戲得到一個episode，會得到很多個
  - 依照很多場的結果，去學每一個s可能得到的value，收集很多場遊戲跟reward了，所以當這個state看到的時候得到的數值去取平均
  - 看到state sa，餵進去function就會有個output，可以手動計算出來，去算平均，給定sa當成input要output的對象，有一個對應的reward要去趨近，去update model
  - 缺點: 每次都要玩完才知道，episode會很長，所以學習過程更慢，因為要玩完才可以得到一組資訊，沒辦法部分episode就去update
- Temporal-Difference (TD)
  - 希望不用全部episode學，critic一樣去學

        - 從估計出來的return當成target去update
        - 兩個時間點的差異，應該要是rt
        - st近來得到的target，跟st+1近來得到的target，兩者相差要是rt
        - 如果估的不好，就會越來越爛
    - 比較: 前者有真實的
        - MC variance 大，TD只考慮兩個時間點差異所以variance小，update容易
        - MC un-bias因為是真正得到的不是估計出來的數值，TD 有bias
        - TD 有Markov假設 預測t+1時間只要看t就夠了

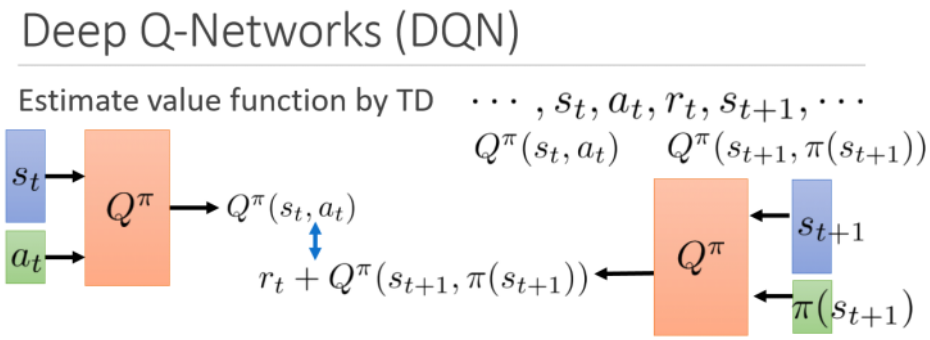




        - 找一條最好的branch (path)，MC真的跑了很多path完整遊戲得到的平均。TD在做value預測只看中間那段出來的reward就可以update了




        - 上圖的下面式子是更新的方法
- Critic = value function
    - input s跟a，ouput scalar (continuous action)
    - 希望可以得到不同動作所得到的ouput 期望值 (discrete action)
- Q-Learning
    - 給定Q(s,a)找出比較好的actor，pi會去跟環境互動，得到很多episode可以去learn Q，每個state的value，更新成比較好的value再去跟環境互動
    - 新的Q更新pi
    - 只有一個function更新value，pi就是從function來，不用額外參數決

定怎麼做動作，僅僅是依據Q而來

- learning target:
- 

- mse loss，希望藍色箭頭兩者越相近越好
- naïve DQN會有難以update 問題，估計值對象會變動所以很難知道要怎麼去做