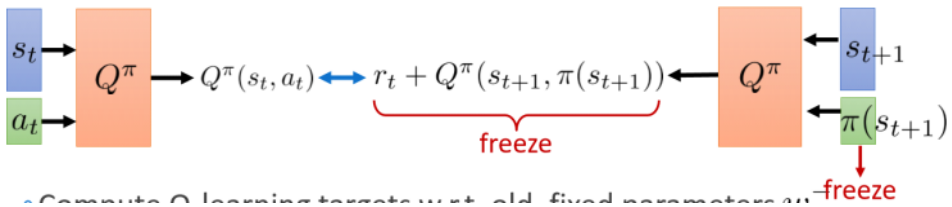
ADL—Advanced DQN

2019年4月23日　　上午 09:19

Q-learning

- Critic = Value Function
- 期望值越高表示之後越可能有好的reward
- MC
  - 學評估critic是去看很多場遊戲，一定要玩完，才有最後真實的reward，一個state得到的平均值reward
  - 可以把很多input state，output empirical mean
  - variance可能會很大，因為是一大長串去平均，不知道哪一個才是關鍵的
- TD
  - 學習只需要部份episode就可以學了
  - 要預測的值也是趨近的，回推前一個時間點預測的值
  - 希望兩個state之間的value差 = rt
  - 缺點: 希望學的V function跟真正的會有rt，所以可能會有bias，如果估不好的話可能會學壞，variance比較小因為只隔一點
- policy是做動作的actor，再透過TD或是MD去學習Q值，利用新的Q值得到新的policy
  - 回傳哪一個action可以比較高的reward
  - 所以把可以獲得高的動作做成最後要用的，所以policy會越來越強因為會估的越來越準
- 學習過程容易震盪不容易converge，是因為適用NN去做optimization
  - sample之間有correlation，下一個sample其實跟前一個sample有關係，但是NN假設每個sample是獨立的 (sequential data是同樣distribution來的)
  - learning target是st跟st+1的數值差了rt，希望Q這個function可以接近，但是St+1這個數值一直在變動，會跟著移動而移動
  - policy是根據Q來的，做這個action Q 值比較高
    - 只有一點點Q 值得改變就會震盪，極端值到極端值
  - 希望Q值的ouput可以接近reward，reward scale差很多的時候，learning會很困難
    - 0~100 or 0~3000
    - 所以BP的時候會很不stable
- 解法
  - experience replay: 存一個buffer，每次都把得到的experience放進去，要update再去隨機拿一個update，因為是從一個pool所以可以變成看作是獨立的
  - Fix住target，用兩個Q function，讓要學習的對象先freeze不動，避免一值震盪導致不穩定

    - 用不同東西去model
- Experience replay
    - 解決correlation，先讓agent去互動有很多experience，根據epsilon-geedy policy
    - 選一個state做哪個action 分數比較高，有一定機率會隨機挑一個action做去探索，大部分時間是依照最好的path，但其實還可能存在更好的
        - exploration: 同樣時間下面得到的不要太侷限
        - epsilon會漸漸decay，一開始比較高的探索機率 (0.05)
        - Boltzman sampling: 從一個機率分布選一個action，把Q值變成機率分布，根據機率分布sample一個Q值
    - 從pool隨便sample一個mini-batch，就沒有correlation了
        - 放進去很多incomplete transition
    - Replay Buffer是從不同policy得到的，有些比較就有些比較新，滿的話就drop古老的，每個iteration都sample一個batch去update Q function=>off-policy
        - 邊玩邊學=>on-policy
        - 看別人完=>off-policy
- Fixed target
    - 因為左邊是要學的，但是右邊的Q function會依值變動
    - 

    - 參數會變多，一組是要fix的(敵方不要動)，等我更新成那組參數，把新的w取代本來的w-
- Reward/Value scale
    - 把她clip在-1~1之間，gradient比較好控制
- 把多個frame合在一起才知道球往哪裡飛
    - Nature DQN (deepmind)

==========

Double DQN (DDQN):

- Q值通常都是被高估的
    - 因為如果一開始就被高估st+1，那St也會跟著被一直抬高
    - 只要有人被高估就會一直選那個動作
- 讓估計Q 值的人跟真的做動作的人分開
    - 提案和執行要分開
    - 原本的a'是從Q^來，現在改成現在的Q去學怎麼action，之前的Q去學怎麼evaluate
    - Nature DQN

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} \hat{Q}(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

    - Double DQN: remove upward bias caused by $\max_a Q(s, a, w)$

Nature DQN

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} \hat{Q}(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

- Double DQN: remove upward bias caused by $\max_a Q(s, a, w)$

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \hat{Q}(s', \arg\max_{a'} Q(s', a', w), w^-) - Q(s, a, w)\right)^2\right]$$

  - 如果Q^被高估，只要Q沒有被高估就可以了，Q值就不會被拿來使用

Dueling DQN:

- 估計Q值，拆成Value function跟Action function
- V只要估計現在這個state好不好
- A去估計這個state要做啥動作
- sample比較少的資料可以更新得比較快
- 為了不讓A直接去學Q值，會做一些限制

$Q(s,a)$ = $V(s)$ + $A(s,a)$

action \ state				
	3	~~3~~ 4	3	1
	1	~~-1~~ 0	6	1
	2	~~-2~~ -1	3	1
average of column	2	~~0~~ 1	4	1
sum of column = 0	1	3	-1	0
	-1	-1	2	0
	0	-2	-1	0

- 在network多加一層normalization去做限制

Prioritized Replay

- 希望每一個experience的被sample機率是不一樣的，根據是否足夠特別，如果現在這個state沒辦法預測這個結果，就要拿出來學
- 希望weight是根據某個數值來的，差異越大代表error越高，希望放進去個數值會有個差，error越高越可能sample (估計比較差者)

Multi-Step

- 希望可以平衡MC跟TD
- 改成多步，用incomplete episode去做update
- TD本來是只拿一小段，拿長段一點的就會比較不會有bias，variance不會像是MC那麼大

Distributional Q-function

- Q值有可能是從一個distribution來的，即便得到相同Q值可能實際上data分布會很不一樣，所以只有一個數值可能無法有代表性
- 所以希望可以改成估計distribution
- 每一個action會有五個bin，每個action五個數值，在不同range的機率，決定policy的時候去學這個action平均之下較高分數者
- 但是可以多估計variance，如果variance太高的可能不要選，不如選擇

entropy比較高的

Rainbow

- 把各種DQN合再一起，把他們估計的Q值medean當成Q值(mean會發生太大bias)
- double DQN比較沒用，因為是為了可以解決Q值會不會高估，因此就會選到相同action，因此希望可以不要高估
- 但是因為已經是meadian了所以不需要兩個network一個去做估計一個去做選擇，所以就算拿掉performace都不太會掉