# ADL—Policy baced

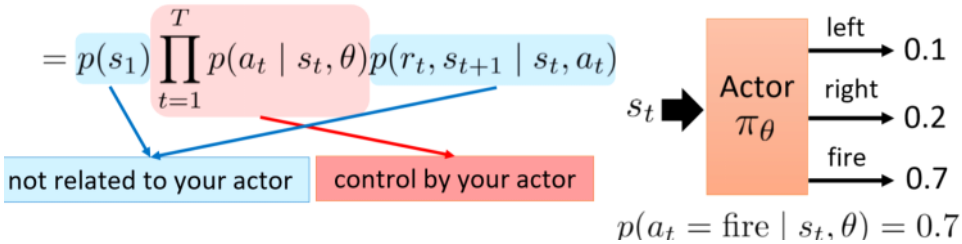
2019年4月23日　　上午 10:58

去學怎麼做動作，學習actor
中間焦急的是actor-critic
=
Policy: agent behavior

- deterministic policy
- stochastic policy

Policy Network

- 一組參數theta，表示policy怎麼做動作
- 給定state跟現在的參數決定怎麼做action
- 目標要去maximize total reward
- value-based是用GD去minimize mse
- 但是policy是為了maximize reward，gradient ascend
- on-policy: 邊玩邊學，玩跟學同一個人
  - off-policy: 做動作跟學的人是不同人，正在學習的agent是看之前蒐集的data
- trajectory tao: 一個完整的episode (互動的其中一種過程)
  - reward是rt的加總會乘上descount
  - 看到s1做a1的機率*看到s1做a1看到s2 … 有一些跟theta有關有一些是env給的
  - $$P(\tau \mid \theta) = p(s_1)p(a_1 \mid s_1, \theta)p(r_1, s_2 \mid s_1, a_1)p(a_2 \mid s_2, \theta)p(r_2, s_3 \mid s_2, a_2)\cdots$$
    $$= p(s_1)\prod_{t=1}^{T} p(a_t \mid s_t, \theta)p(r_t, s_{t+1} \mid s_t, a_t)$$

    
    
  - 估計reward的期望值去model
    - R(theata)比較低會得到比較爛的actor
    - 給定theata會sample出tao的機率，乘上reward，就是theta會得到的期望值
  - 直接拿pi theta去玩遊戲可以得到N個tao，從theta裡面sample N 次tao出來，當N足夠大希望可以趨近
    - EX: 玩了兩萬場的平均，就當成theta估計出來的期望值
    - 把所有可能的trajectory sum起來
    - 利用gradient ascend去maximize reward
  - 每一次在update的時候一句GA結果去update新的，optimal policy就是把所有的theta帶入得到最高的reward 期望值者
  - reward可以是黑箱，不用可以微分，不用跟actor相關，因為是環境給的

- Policy gradient
  - 從theta sample出tao的機率
  - 要去修改policy去得到更高的R
  - R的gradient看到state去做了a，如果R的數值>0，就代表會得到比較好的reward
    - 對每個state做各種action的機率值
    - 做某個action可以讓之後的reward上升，讓機率變大
    - 如果這個state做某個action 得到的reward<0表示要下降做這個action的機率
    - R是accumilate reward，要看到整個trajectory (total reward)
  - 要把整場遊戲玩完才可以有total reward
  - 玩很多場收集data，要去update policy，更新結果: 決定policy怎麼改變，看到各state做個action的機率要變高還是要變低
    - 得到新的theta，要把所有tao丟掉，再重新玩一連串遊戲，再去update
  - 要學一個模型去做分類，給定x要去做action的機率
    - 影像分類問題
    - 找到對應的training data，target function要去maximize cross-entropy
    - 給定s，看到at的機率，把所有data加總起來，要去maximize式子
    - 看到state決定output label的機率
    - 決定每一個state做每一個action，要weighted by total reward

Adding Baseline
- 要model的policy是一個機率分布(of actions)
- 每一個action都有做到的話，有些機率要提高有些機率要提高更多，因為有些多有些少所以可以做出差異
- 但其實在做update不會所有可能性都看過
- 因為是做sampling，把a都看過，因為其他拉高而降低，當reward都是正的問題，沒被sample到的機率沒被拉高
- 所以要讓reward有正有負，有些人上升有些人減少，減掉b，減去平均值就會有正有負

Actor-Critic
- 有兩個東西都是用參數來學的
- value based負責去估計Q值
- 額外參數去學policy怎麼做，有參數可以去學怎麼maximize value，比較適合continuous scenario
- A2C，有value function，在learn policy的時候，改用value function估出來的值當成total reward，不用從trajectory看reward，有一個critic，用估計出來的值取代total reward
- 用advantage function去帶total reward

    - 把Q值拆解，V+A，A就是advantage function
- 當作了at之後真的可以拿到的reward 減去 估計的，如果>0估計的太低了，<0實際上太高了要下降機率
- 希望可以藉由估計的reward跟實際的reward調整差異
- 原本估計比較低，實際上比較高
- A2C
    - 會把actor跟critic一部份的參數會share，讓彼此互相影響
    - 讓ouput機率分布當成regularization，希望entropy不要太高，希望不要太sharp，其他動作也可以有一些機率

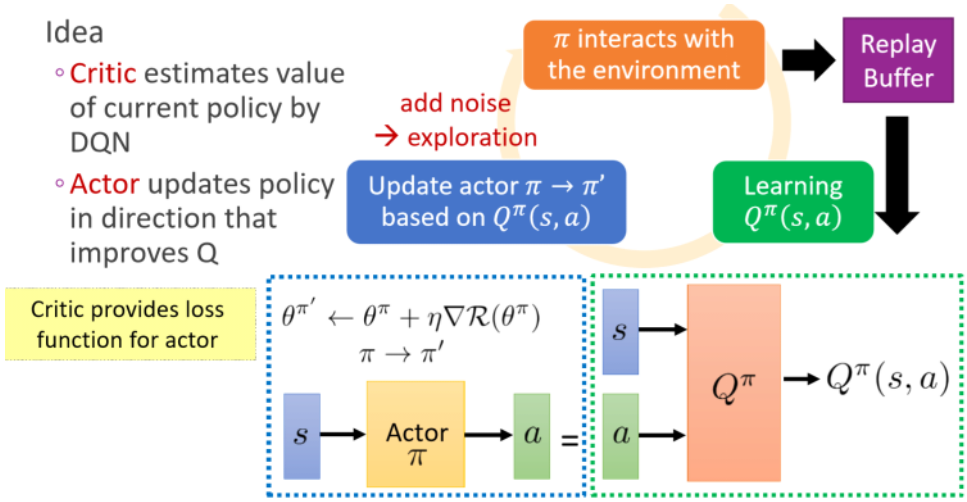
Asynchronous Advantage Actor-Critic (A3C)

- 很多小worker收集資訊去update global network
- 一開始小worker會去copy global參數
- 各個worker各自update，去update global，沒有規定要怎樣update，可能已經update過了
- 算玩gradient就去update

Pathwise Derivative Policy Gradient

- 原本的AC，告訴一個action好或是不好
- 不是直接去major function 好或不好，直接去做較高分的action就好
- 調整policy希望可以讓數值變大
- 把policy network跟value network合在一起，調整pi可以讓最後的Q變大，所以用GA
- 調整actor的時候要fix Q，希望update過程中讓Q上升

DDPG

- critic適用DQN 來估計現在policy value，告訴Q要怎麼去學
- actor要去藉由調整policy去improve Q 值
- 讓policy network
- 調整actor，要fix Q network (會用replay buffer更新)
- 類似A3C，但是適用critic給的資訊去GA
- 

- 跟還記互動收集experience放進去buffer，sample experience出來，用TD mse，去用差值可以符合期待當成target去學習Q function
- 去update actor，fix Q network，maximize
- optimize的function要去跟舊的組合不是取代，依照比例去update一點