R05922131 祝子軒
ADLxMLDS hw3 report

Game Playing

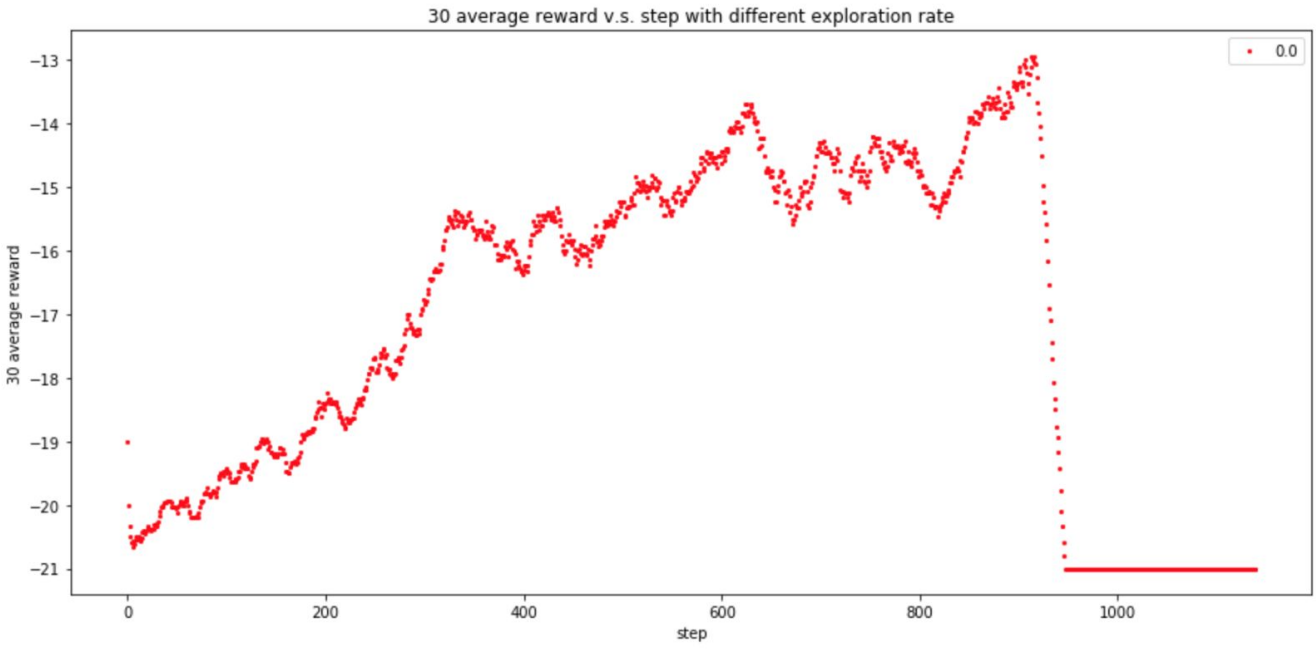
- Basic Performance (6%)
  - Describe your Policy Gradient & DQN model (1% + 1%)
    - Policy Gradient
      - input : （狀態，動作，獎勵）
      - model : 狀態通過兩層CNN＋兩層Dense
        - Conv2D(16, (8,8), strides=(4,4), activation='relu')
        - Conv2D(32, (4,4), strides=(2,2), activation='relu')
        - Dense(128, activation='tanh')
        - Dense(6, activation='softmax')
      - output : 各種動作的機率
      - 透過玩完一輪遊戲，紀錄每一個步驟的狀態，動作以及獎勵，算完折扣獎勵後（gamma=0.99），透過以下公式更新

        $$\theta' \leftarrow \theta + \eta \nabla \mathcal{R}(\theta)$$

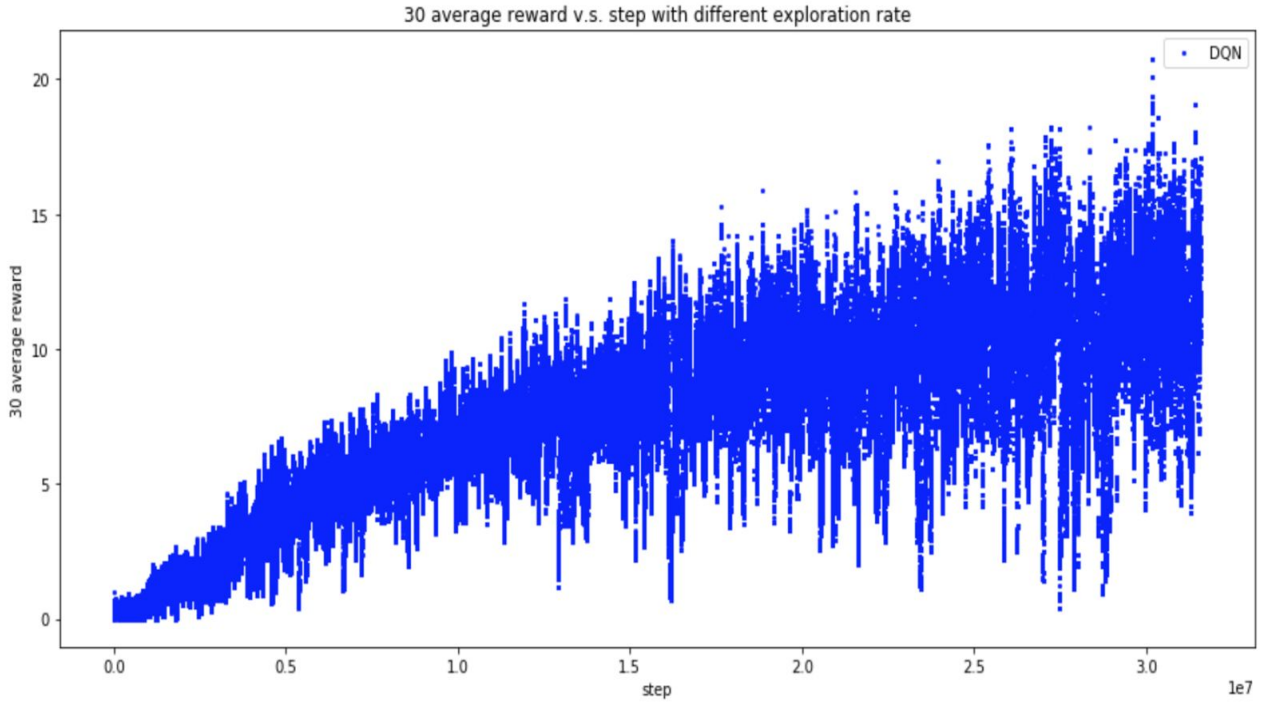
        $$\nabla \mathcal{R}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta)$$

        note:上面$R(r^n)$使用折扣後的r_t代替
    - DQN
      - input : （狀態，動作，獎勵，下一個狀態）
      - model : 狀態通過三層CNN＋兩層Dense
        - Conv2D(32, (8,8), strides=(4,4), activation='relu')
        - Conv2D(64, (4,4), strides=(2,2), activation='relu')
        - Conv2D(64, (3,3), strides=(1,1), activation='relu')
        - Dense(512, activation='relu')
        - Dense(4, activation='linear')
      - output : 各種狀態的Q值
      - 先將一連串的遊戲過程（狀態，動作，獎勵，下一個狀態）放在記憶庫，隨機抽取出來訓練Q，經過32*50 step 後作更新另一個架構一樣但參數不同的Q'，重複訓練。

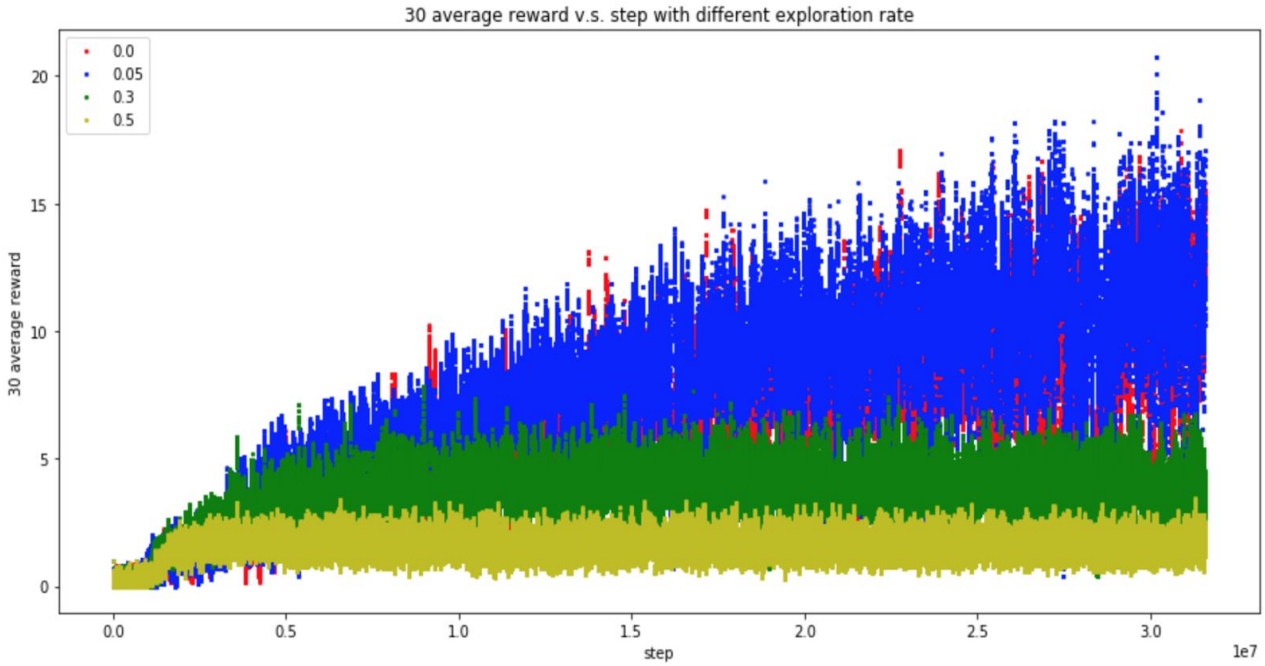
- Plot the learning curve to show the performance of your Policy Gradient on Pong (2%)



- Plot the learning curve to show the performance of your DQN on Breakout (2%)



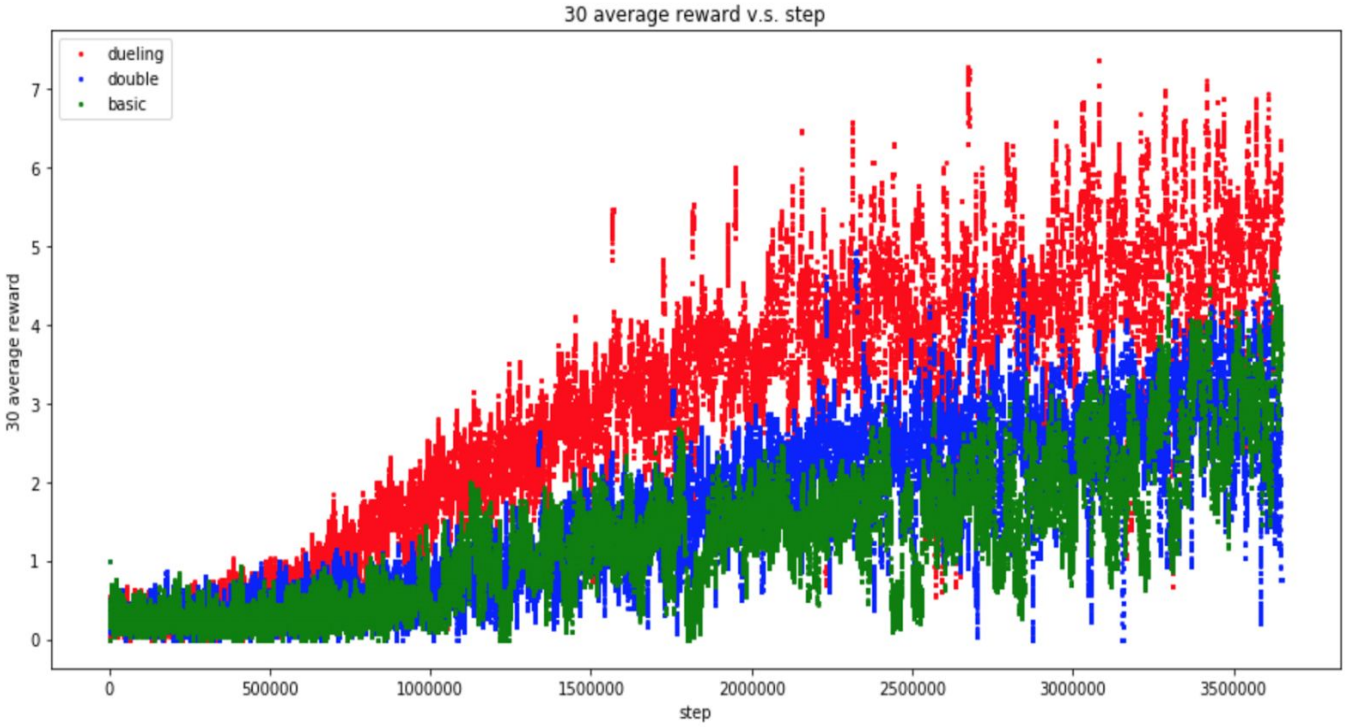
- Experimenting with DQN hyperparameters (4%)
  - 我選擇exploration schedule的參數當作調整的參數 : E_rate
    exploration rate 在100萬步左右會由100%逐漸降低到E_rate,
    取E_rate ∈{0.0, 0.05, 0.3, 0.5}
  - Plot all four learning curves in the same graph (2%)



  - Explain why you choose this hyperparameter and how it effect the results (2%)
    - 因為很好奇Exploration and Exploitation這個新學到的概念可能會造成的影響，因此選擇這個參數。
    - 由上圖的reward紀錄看來，前期大家表現差不多，因為幾乎都是隨機的，而中後期exploration的比例不宜太高，不然表現不會太好，（若E_rate=0.5，則兩步裡面有一步是隨機亂動，因此在玩breakout這個遊戲時會因為隨機亂動而遊戲結束），但也可看出若加上一點隨機性（0.05），表現並不會比較差甚至稍好一點，而卻能避免遊戲鬼打牆之類的遊戲過程。

Bonus

- Improvements to DQN (2%)
  - Double DQN
    - 因為會被更新的是Q(s,a,w)，若是取max Q(s,a,w') 有可能會造成高估的情況（因為在w' 和 w的參數不一樣下不一定會選同一種action），所以使用 argmax Q(s,a,w) 來決定動作，而不會讓Q(s,a,w')自己估算值又自己選動作，因此會更合理。
  - Dueling DQN
    - 將原本的DQN拆成兩個function來綜合評估Q值，V(s)只需要知道狀態是什麼，而不需要知道action就能夠輸出一個評估狀態好壞的值，因此對於環境的判斷更貼近一點，而A(s,a)能作狀態和動作的評估，等於是原本的一位評審差解成兩位更專業的評審來評斷Q值。



由上圖可看出，相較於原本basic DQN，double DQN 和 dueling DQN 能夠更快速的學習，dueling DQN的改善較為顯著，而double DQN 則有小幅的改善。對於breakout這款遊戲而言，兩種方法皆有改善。

- Implement other advanced RL method, describe what it is and why it is better (2%)
    - 我選擇實作 actor-critic，這是一個以value-base 和 policy-base結合的一種方法，以policy gradient為主體，把baseline換成由advantage function 計算出的值來取代原本的R，如下式：

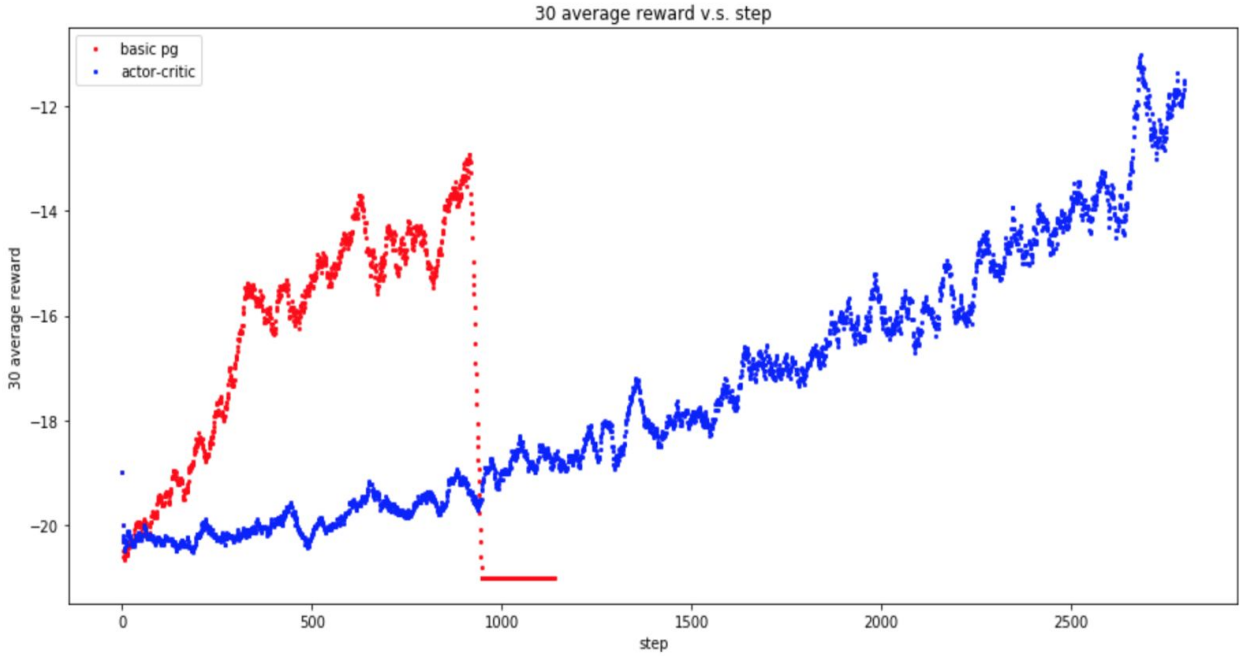
$$\nabla \mathcal{R}(\theta^{\pi}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla \log p(a_t^n \mid s_t^n, \theta^{\pi})$$

baseline is added

evaluated by critic

Advantage function: $r_t^n - \left(V^{\pi}(s_t^n) - V^{\pi}(s_{t+1}^n)\right)$

如此可以更彈性的透過critic function來給獎勵，而critic function主要是基於現在狀態和下一個狀態的預估差來給值，因此若真實的reward大於critic估計出來的reward時，此步的更新應該要被加大，而若真實的reward小於critic估計出來的reward時，此步的更新應該要被避免，因而達到較好的學習效果。
下圖是和basic policy gradient 比較的結果。



雖然actor-critic學習較慢，但比較不會有爆掉的問題。來不及跑下去...