r08944035 呂翊愷

ADL Homework #3

Q1: Models

- Describe your Policy Gradient & DQN Model

Policy Gradient

- Model架構：
  Input: state 1 * 8

  Fully connected後經過ReLU，再Fully connected到4維後進行softmax，得到4種action的機率分佈。

  Output: action probability 1 * 4

- Make action：
  當Policy Gradient在遊戲中的每一個state的時候，將state輸入model中，預測出action的機率，再利用`torch.distributions.Categorical`得到對action抽樣的相對機率，再用`sample()`對action抽樣，得到下一個action，如此反覆直到遊戲結束。

- Loss：
  使用公式$R = \sum_{t=1}^{T} \gamma^{t-1} r_t$，算出整場遊戲中每一步的discount reward，再和log過後的action機率相乘之後相加，就得到loss。

- Optimize：
  使用Adam optimizer，learning rate設為3e-3。

DQN

- Model架構：
  online network和target network皆相同

  Input: state 1 * 4 * 84 * 84

  經過三層的Convolution和ReLU後，再經過兩層的Fully connected和ReLU後，得到9種action的機率。

  Output: action probability 1 * 9

- Make action：
  使用Epsilon greedy決定要不要使用network預測出來的action。首先先隨機產生一個0~1之間的數，如果該數大於epsilon就根據network預測出來的action probability，將機率最

大的action當作下一個action；如果該數小於epsilon就隨機選一個action當作下一個action。

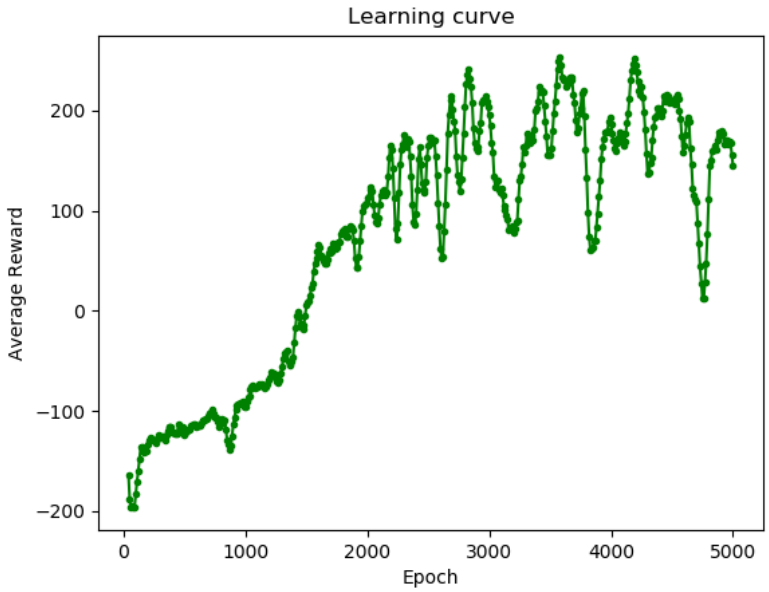
- Replay Buffer：
  使用一個自定義的class將當前的10000筆的state, action, next state, reward存起來，在訓練時會隨機取幾筆資料做訓練。

- Loss：
  根據公式 $L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^{-}) - Q(s, a, w)\right)^2\right]$
  將Replay Buffer中儲存的state, action, next state, reward拿出來，將Replay Buffer中的state傳入online network，得到9種action的機率，再取Replay Buffer中的action所對應的機率算出Q值。
  將Replay Buffer中的next state傳入target network，得到9種action的機率，再取最大的機率和gamma相乘後和Replay Buffer中的reward相加，得到expected Q值，接著即可將Q值和expected Q值用MSELoss算出loss。

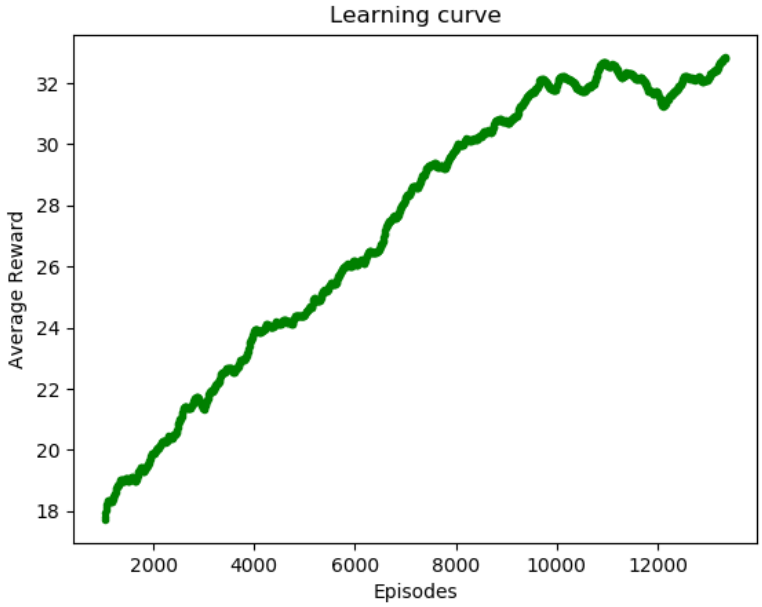
- Optimize：
  使用RMSprop，learning rate設為1e-4。

• Plot the learning curves of rewards

Policy Gradient

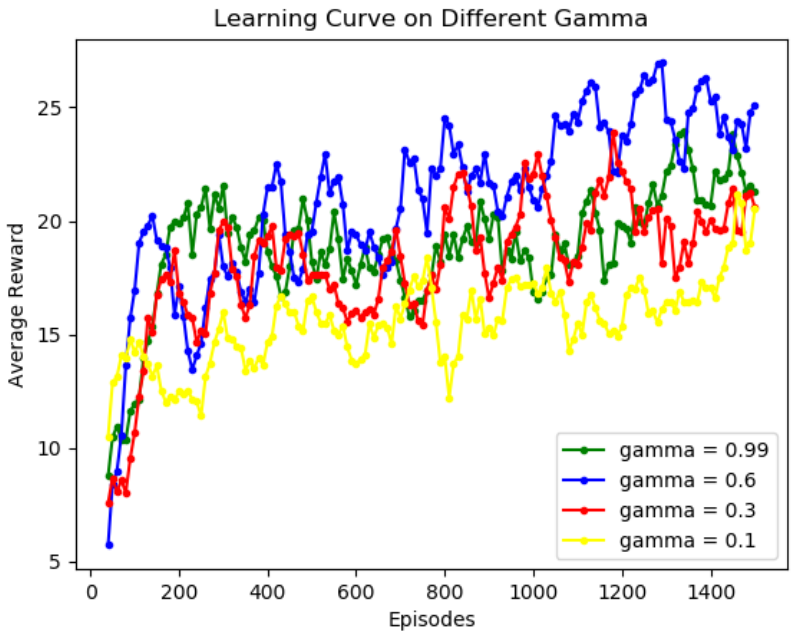


DQN



Q2: Hyperparameters of DQN

我將gamma分別以0.1, 0.3, 0.6, 0.99做測試，想了解discount reward的計算方式對結果的影響有多大。



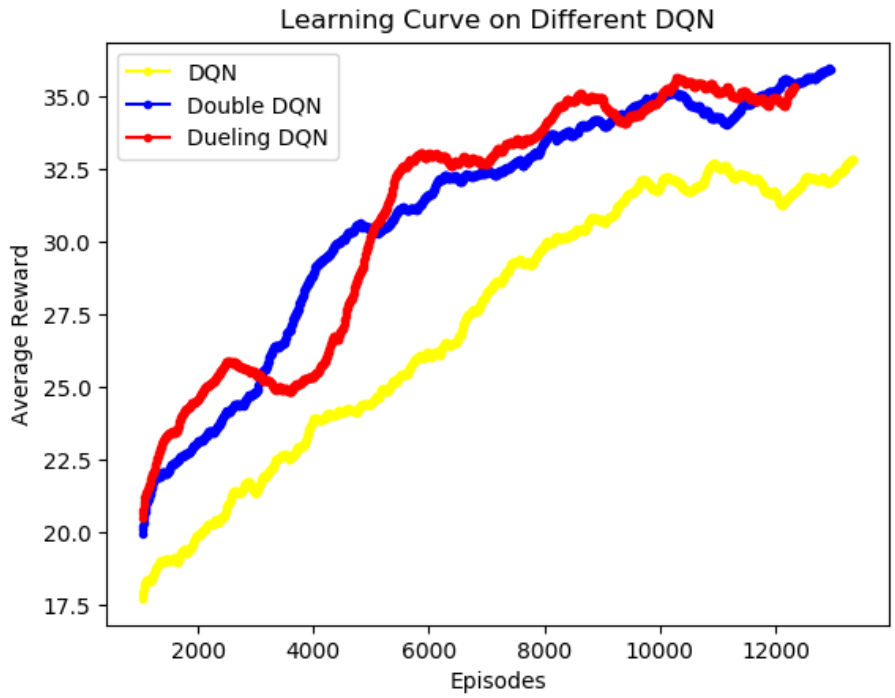
測試結果其實蠻讓我意外的，原本我以為gamma越高會越好，但從上圖可以發現，gamma = 0.6的時候表現最好，且比其他gamma值好蠻多的。我認為gamma值的意義是在於，gamma值越高，代表agent越會受未來影響；gamma值越低，代表agent越受到當前的狀態影響。至於為什麼gamma = 0.6時會最好，我認為可能跟遊戲本身的性質有關，也許每種遊戲需要注重當前還是未來的程度有所不同，因此適合的gamma值都不同，不一定是越高越好。

Q3: Improvements of Policy Gradient / DQN

我實作了Double DQN和Dueling DQN，想看是否能影響結果。

Double DQN的實作只要將DQN計算expected Q value的地方修改成 $r + \gamma \hat{Q}(s', \arg\max_{a'} Q(s', a', w), w^-)$就好，如此一來就變成用online_net預測action，然後用target network計算expected Q value。

Dueling DQN只要修改DQN的network架構就好，將原本DQN經過三層convolution和ReLU後的layer接上兩個不同的fully connected：一個linear到9維，用來預測每個動作的advantage；一個linear到1維，用來預測state的value，最後的Q值就是把advantage加上value得出來的。



從上圖可以發現，Double DQN和Dueling DQN都比DQN好上不少。Double DQN表現會比DQN好的原因是，原本的DQN在計算expected Q value只有用target network去計算最大的Q值，而若target network預測出來的Q值是被高估的，target network有可能會朝著錯誤的方向去學習。但Double DQN使用兩個不同的network來預測action和Q value，所以即使action或Q value有被高估的情況，只要另外一個network沒有使用到被高估的值就不會有高估的問題。

Dueling DQN表現的也比DQN好，原因是network除了計算state的advantage值，還加上了一個value值去估計這個state的重要性。此外，由於network預測出來的advantage值需要先經過normalize之後才能和value相加，所以network在學習的過程中，會傾向改進value值，也就是傾向去學習state的重要性，知道當前應該注重在哪一個state，進而再去對這個state裡面的action去做預測，因此效果會較原本的DQN好。