

HW3 Report

2017 ADLxMLDS

B03901163 電機三 鄭 煦


Problem description

利用 reinforcement learning 來訓練電腦玩 pong 和 atari 兩個 openAi 的遊戲

Policy gradient network

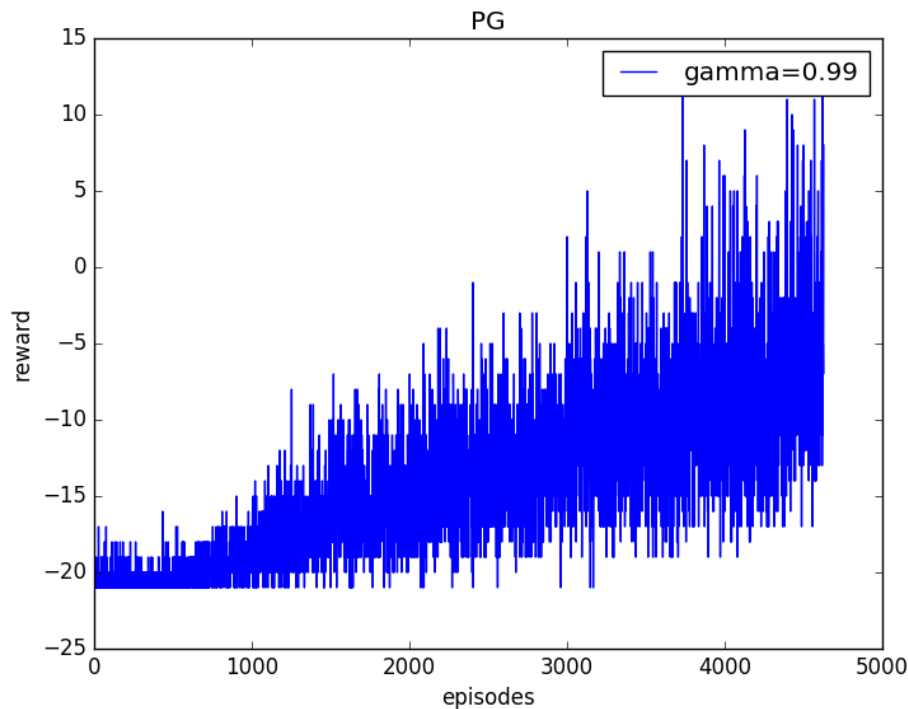
Policy network 基本上就是輸入一個狀態，然後直接輸出 action，而更新 policy network 的方式，是藉由 gradient descent 的方式，而 loss function 的設計上會考量過去的紀錄在一次乘上一個 discount 的係數，下圖是 policy gradient 的核心演算法。S 代表 state, 而 a 代表 action。Network 的架構是兩層的 dense layer，input 為 6400 維的 vector，而 output 出一個 3 維的 vector 代表 action。hidden layer 的大小是 200。

REINFORCE algorithm:

- 
1. sample $\{\tau^i\}$ from $\pi_{\theta}(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
 2. $\nabla_{\theta} J(\theta) \approx \sum_i \left(\sum_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
 3. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

訓練結果:

因為這是後來重跑的圖，所以沒有時間跑完原本的 episode，實際能過 baseline 的 model 大概要跑到 9000~10000 個 episodes，不過可以看出 reward 是有緩慢地在上升，大概跑到 10000 左右的時候，reward 就會卡在 3~4 中間。



DQN network

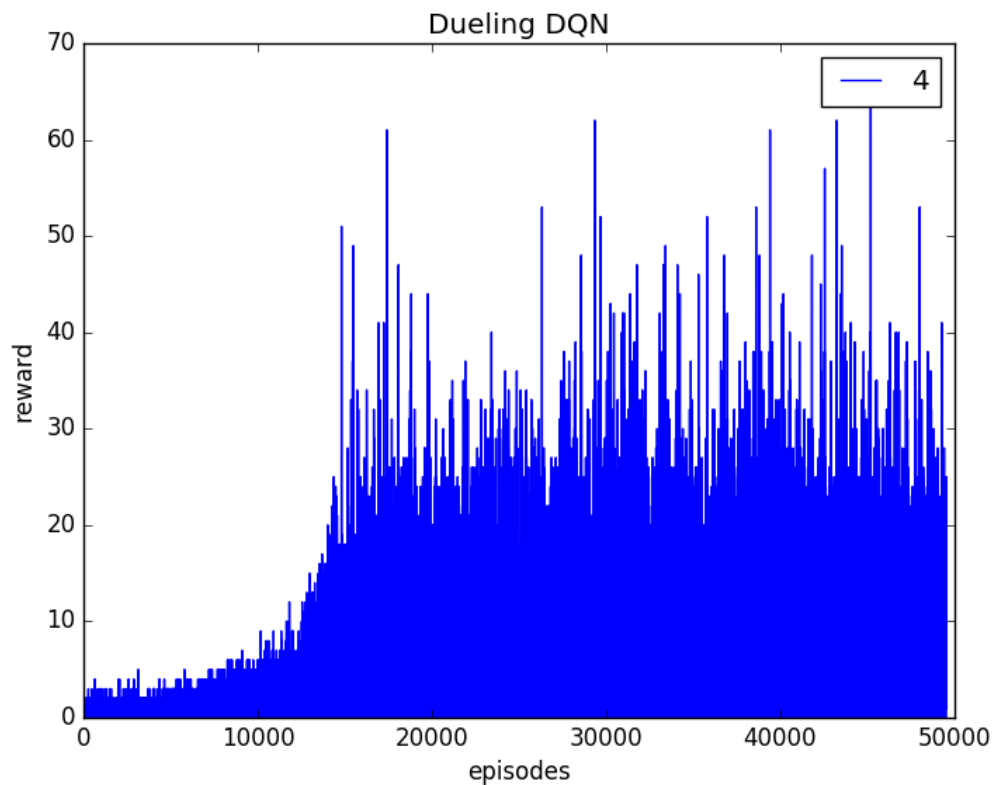
在這次的作業裡，我直接 implement dueling DQN，一般的 DQN 是 target network 和一個 online network，相同的結構，可是有不同的更新頻率，下圖是一般的 DQN 演算法。
“classic” deep Q-learning algorithm:

1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ' : copy ϕ every N steps

在 dueling network 裡面，會將後續的輸出分成這個 state 的值，加上每個動作在這個 state 上的 advantage，這樣的好處是可以估計出某個 action 所帶來的額外價值，如下圖所示。
然後這次所有的 hyper parameter 都跟助教提供的相同。

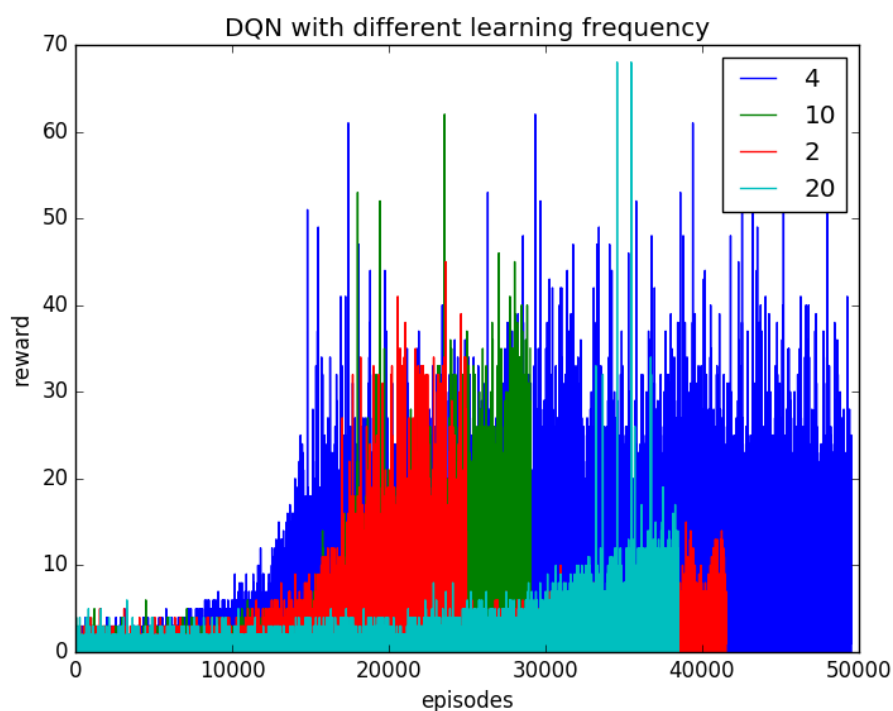
$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha),$$

訓練結果:



Hyper parameter experiment

這次的 parameter 選擇我選了 online network update frequency，network 全部都是原本的 dueling Q network，從圖中可以發現太大的 update frequency 或太小都不好，太大甚至會有可能會導致 network 幾乎 train 不起來。太小可能會導致 network 反反覆覆，train 起來需要更多的時間。選擇這個參數的原因是因為我覺得這個參數對 online network 的影響應該不小，進而影響到 target network。



Bonus

這次的作業裡面我額外 implement 了 double DQN，double DQN 主要是為了解決原本 Q learning 的時候會出現 q value overestimate 的問題，所以把原始的 q learning 的項改成下面的項，如此一來在 training 的過程中會比較穩定，不過 performance 好像不會有顯著進步的樣子

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a; \theta_t), \theta_t^-).$$

Training result:

我直接用 duel q 和 double q 比其實是不太公平的，畢竟兩個方法所想要解決的問題不同，而且我沒有時間讓 double q learning 跑很多 steps，所以 result 的比較可能參考就好。

