ADLxMLDS HW3 Report

Name: 李宇哲 Student ID: r06942074 Dept.:電信所碩一

1. Basic Performance

Policy gradient:

我的 pseudo codes 如下：

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return θ
end function

理論上我的 model 會吃 input frame:

Input:

    RGB image: np.array

        RGB screen of game, shape: (210, 160, 3)

    Default return: np.array

        Grayscale image, shape: (80, 80, 1)

My policy gradient model:

Gamma = 0.99

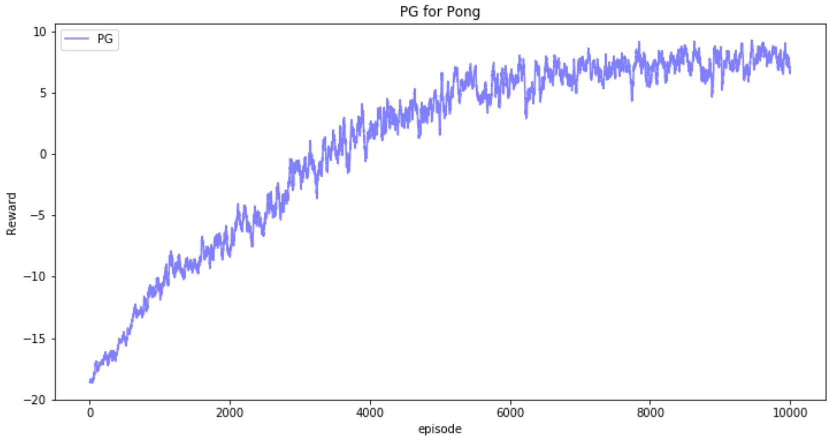
learning rate = 0.0001, optimizer = adam

My PG model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 25, 25, 16)        592
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 11, 11, 32)        12832
_________________________________________________________________
flatten_3 (Flatten)          (None, 3872)              0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 23238
=================================================================
Total params: 36,662
Trainable params: 36,662
Non-trainable params: 0
_________________________________________________________________
```

用了兩層 CNN 加上一層 dense 出去

Learning curve:



Testing result:

```
episode 26: 6.000000
episode 27: 7.000000
episode 28: 11.000000
episode 29: 10.000000
Run 30 episodes
Mean: 9.63333333333
```

My DQN model:

EXPLORATION_STEPS = 1000000, gamma = 0.99, epsilon = 1.0, epsilon min = 0.05, epsilon decay = 0.999, learning rate = 0.0001, optimizer=adam

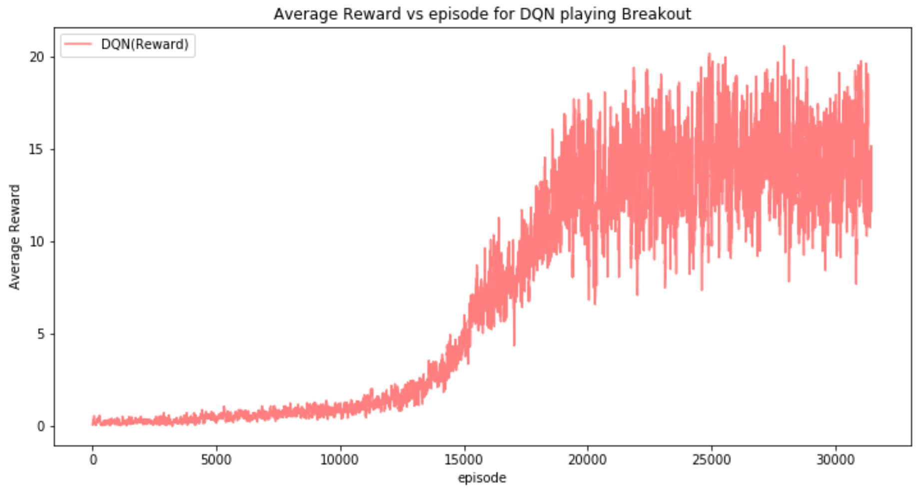
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 20, 20, 32)        8224
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 64)          32832
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1606144
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 1,686,180
Trainable params: 1,686,180
Non-trainable params: 0
_________________________________________________________________
```

Testing result:

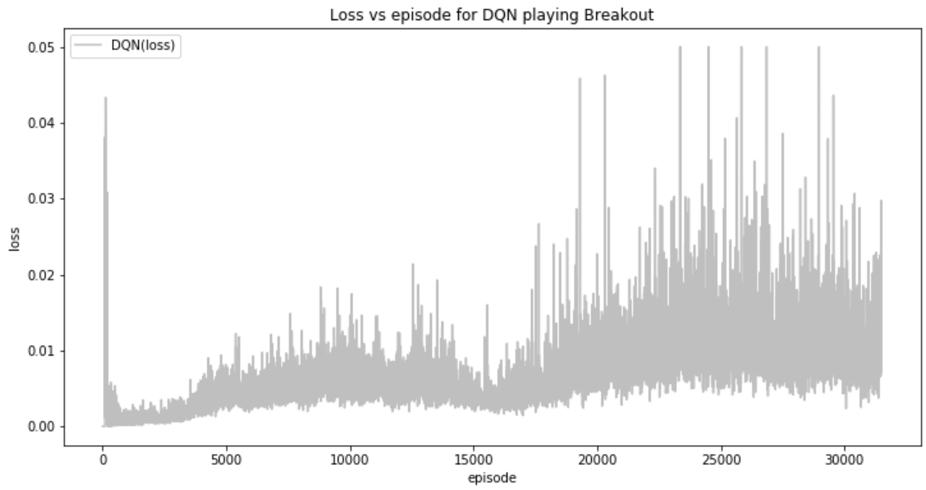
```
episode 96: 139.000000
episode 97: 174.000000
episode 98: 25.000000
episode 99: 4.000000
Run 100 episodes
Mean: 81.15
```

Learning Curve:

下圖是用了每 30 episode 的 moving average，可以看出 model 的 Ave. Reward 有隨著 episode 的增加越來越好。



下圖是 DQN 的 loss curve，可以發現 loss 在前的的時候是很大的狀態，但在後面 loss 似乎沒有變小的趨勢，所以可以看出 loss 的分佈並不能看出 model 的學習效果。

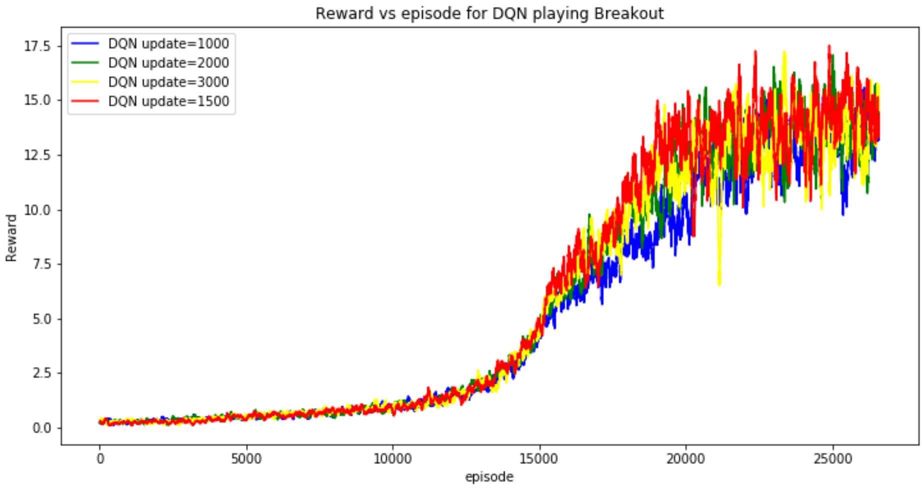


Experimenting with DQN hyperparameters

Chosen target update size: **1500**, 1000, 2000, 3000

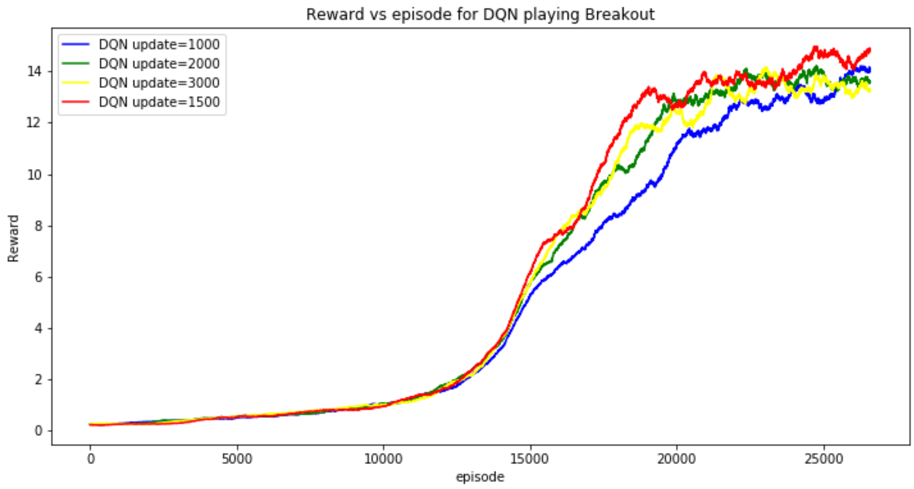
我取了 target update size 當作是我的 experimenting hyperparameters

我一樣拿 average reward 去比較如下圖：



上圖可以發現，update 的次數為 1500 時的學習效果相對的比較好，因為可以發現 1500 的 network 表現略大於其他 update 的 setting。

為了再更清楚比較，我把 average reward 的計算變成每 1000 個的平均，如下圖：這裡就更明顯 1500 的次數是最完美的，畢竟 2000 跟 3000 次數多反而沒有比較好，但是次數少也不高。



Bonus:

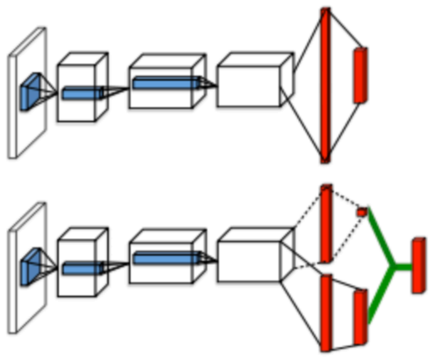
1. Improvements to DQN

DDQN:

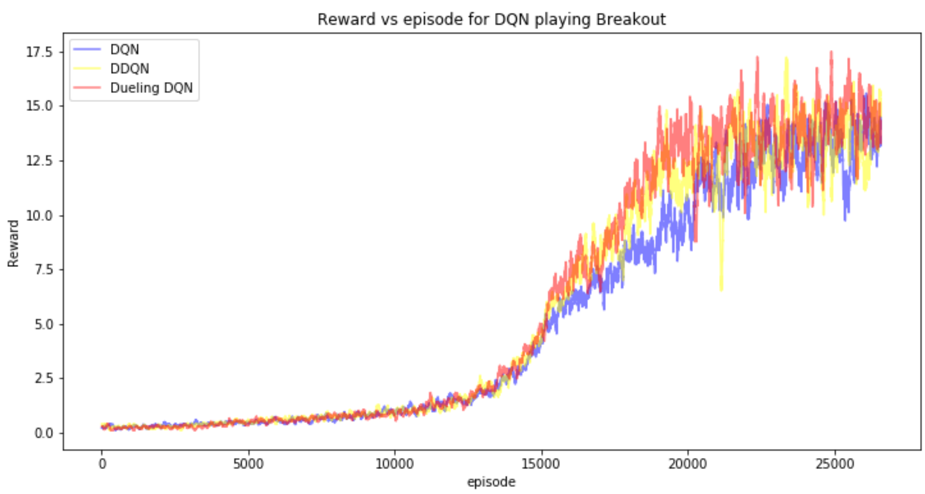
DDQN 就是利用 target network 去更新，架構跟 DQN 一樣

Dueling Network:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 20, 20, 32)        8224
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 9, 9, 64)          32832
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
flatten_4 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               1606144
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 2565
_________________________________________________________________
lambda_1 (Lambda)            (None, 4)                 0
=================================================================
Total params: 1,686,693
Trainable params: 1,686,693
Non-trainable params: 0
_________________________________________________________________
```



下圖為三者的比較，Dueling network 和 DDQN 的效果明顯都比 DQN 好，至少在後面的狀況都是 performance 比較好。所以 DDQN 會比較好的原因主要是因為 target network 的更新使 network 比較穩定，再者 Dueling network 也會比較好就是因為就是把 state vale 和 action value 都考慮在內。



A2C：

結合了 Policy Gradient (Actor) 和 Function Approximation (Critic) 的方法. Actor 基於概率選行為, Critic 基於 Actor 的行為評判行為的得分, Actor 根據 Critic 的評分修改選行為的概率.

Value network (left)& Policy network(right):

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 4, 84, 84)	0
conv2d_14 (Conv2D)	(None, 16, 20, 20)	4112
conv2d_15 (Conv2D)	(None, 32, 9, 9)	8224
flatten_6 (Flatten)	(None, 2592)	0
dense_8 (Dense)	(None, 256)	663808
value (Dense)	(None, 1)	257

Total params: 676,401
Trainable params: 676,401
Non-trainable params: 0

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 4, 84, 84)	0
conv2d_14 (Conv2D)	(None, 16, 20, 20)	4112
conv2d_15 (Conv2D)	(None, 32, 9, 9)	8224
flatten_6 (Flatten)	(None, 2592)	0
dense_8 (Dense)	(None, 256)	663808
policy (Dense)	(None, 4)	1028

Total params: 677,172
Trainable params: 677,172
Non-trainable params: 0

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

利用 A2C 建立 Value network (left) & Policy network(right)的架構可以發現整體的狀況在 Breakout 的遊戲的確比一般的 DQN 穩定。