ADLxMLDS2017 - HW3

電信碩一 宋易霖 r06942076

1 Problem Define

實作policy gradient以及deep Q-learning兩個演算法，用來玩pong及breakout這兩個atari遊戲。

2 Describe your Policy Gradient and DQN model

我的兩個model都是參考助教的，不同的是我的兩個model都加上batch normalization，希望model可以收斂快一些 (結構如Figure 1)。
PG的model結構如 Table 1所示，實作的演算法就是最基本的reinforce policy gradient，目標函數爲：

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

其中 $R(\tau^{\pi})$ 爲 discounted的累積reward，就是把pong遊戲中的一次得分的reward爲單位，每得一分後累積的reward就歸零。model的目標就是maximize此目標函數，也代表整場遊戲的累積reward愈大愈好。
DQN的結構如 Table 2所示, 目標是學到最好的Q函數 (描述遊戲中每個state和action的好壞)。model要最小化以下目標：

$$\mathbb{E}[(r + \gamma max_{a'} Q(s', a', w^{-}) - Q(s, a, w))^2]$$

其中s', $a't$爲s, a的的下一個state以及下一個action，w^{-}爲target model的參數，w爲online model的參數。會有一個replay memory紀錄之前玩過遊戲的state, action及reward。這個期望值就是從memory裏面sample一些數據來計算的。
Learning curve 的結果在 Figure 2，DQN除了取前三十場平均外，在畫圖時又取了前後各50個點平均讓線不要那麼崎嶇。pg大約在4000場左右過baseline，breakout因爲有clip reward則很難用learning curve看出什麼時候過。而breakout學習變動很大，有時候在時間t過了baseline，時間t+1 反而沒過。

```
PG (
  (feature): Sequential (
    (0): Conv2d(1, 16, kernel_size=(8, 8), stride=(4, 4))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU (inplace)
    (3): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU (inplace)
  )
  (output): Sequential (
    (0): Linear (2048 -> 128)
    (1): ReLU (inplace)
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True)
    (3): Linear (128 -> 6)
    (4): Softmax ()
  )
)
```

(a) Policy gradient model (text)

```
DQN (
  (feature): Sequential (
    (0): Conv2d(4, 32, kernel_size=(8, 8), stride=(4, 4))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU (inplace)
    (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU (inplace)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU (inplace)
  )
  (output): Sequential (
    (0): Linear (3136 -> 512)
    (1): ReLU (inplace)
    (2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True)
    (3): Linear (512 -> 4)
  )
)
```

(b) DQN model (text)

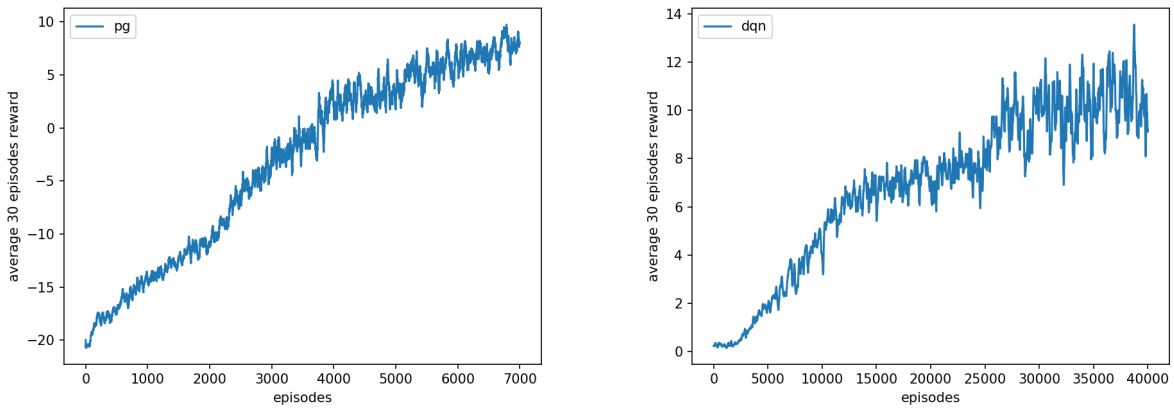
Figure 1: Model structures

total episode	best test reward
7000	9.03
optimizer, learning rate	**reward decay**
RMSprop, 10^{-3}	0.99

Table 1: PG model settings (best test reward是用得到最高的training reward的model算的)

total episode	best test reward	replay memory
40000	68.32	10000
optimizer, learning rate	**reward decay**	**episodes before learning**
RMSprop, 2.5×10^{-4}	0.99	1000 episodes
batch size	**update freq. of target model**	**update freq. of online model**
64	50 episodes	4 actions

Table 2: DQN model settings (best test reward是用得到最高的training reward的model算的)



(a) Pong (pg)

(b) Breakout (DQN)

Figure 2: Learning curves, average reward of last 30 episodes to numbers of episodes

3 Experimenting with DQN hyperparameters

我選擇的參數是replay memory，因爲一開始在還沒使用助教公佈的參數時，我用10000的memory size時的training reward大概在3 ~ 4分，但是同樣的場數用500000 memory size卻可以train到7 ~ 8分，因此覺得memory size應該對model影響很大，所以選擇了他。

由Figure 3可以看出replay memory對於model的影響的確頗巨大的，基本上memory愈大的話reward幾乎都愈高，而且幾乎training從頭到尾都是領先的。因爲memory size調大的話，可以記住較多前面幾場exploration後留下的結果，因此在training的時候也較容易sample到那些state(因爲exploration主要在training前期，若是memory太小一下就被洗掉了)，所以model等於是可以比較好explore到reward高的地方，進而讓training更好。

但是也可以發現memory size愈大的在training中後期的變動是較大的，因爲在中後期大部份state可能都探索過了，在reward都趨於穩定的情況下，若是memory size較大的話就會記住那些用很久以前的Q function所進行的遊戲，但是那時候的Q肯定較差，導致那時候選擇的action可能有問題，因此若sample到那些點可能會讓Q function更新出錯。但是因爲Q function本來就不是完美的，因此舊的Q function選擇的action可能誤打誤撞選到對的，因此這樣的出錯會導致結果有好有壞，因此learning curve變動較大。

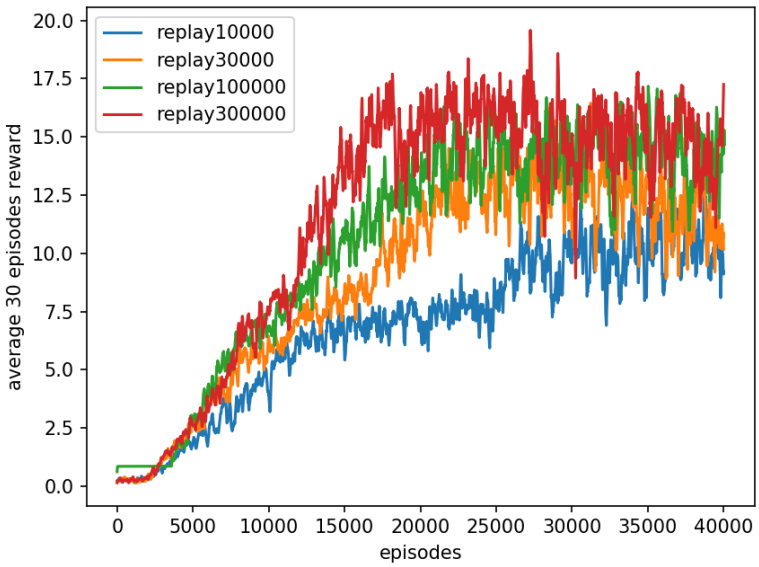


Figure 3: DQN learning curve (different size of replay memory)

4 Bonus

4.1 Improvements to DQN

我實作的兩個DQN的improvement是double DQN以及dueling DQN。

4.1.1 Double DQN

Double DQN是用online model來選擇action，用target model來評估下一個state及action的好壞，如此用同一個model選擇action及評估會造成高估某一個action的問題可以被減小。Double DQN的目標函數如下：

$$\mathbb{E}[(r + \gamma max_{a'} Q(s', argmax_{a'} Q(s', a', w), w^-) - Q(s, a, w))^2]$$

我利用double DQN玩breakout的結果如Figure 4，基本上其餘參數設定都和原本的DQN一模一樣。我發現double DQN玩這個遊戲的結果在40000個episodes內是比原本的DQN差的，但是double DQN其實仍然在成長，只是爲了和DQN比較就只能停在40000場遊戲了。有兩個可以觀察的地方，第一個是Double DQN成長速度較慢，第二個是Double DQN training變動比較小。第一個我認爲是因爲breakout這個遊戲主要只有左右移動兩個動作，因此用傳統的DQN學習greedy一點也不太容易出錯，而且這樣greedy的更新某個動作，若那個動作是好的就會學習較快。同時這也對應到第二個問題，太greedy的更新會導致若發現了另一個好的動作後，就會馬上更新成該動作，因此training的變化會比較劇烈。

4.1.2 Dueling DQN

Dueling DQN是將network的輸出$Q(s, a)$改成$V(s) + A(s, a)$，這樣一來network更可以分別學到每個state的以及搭配action後的分數如何。從Figure 5 可以發現dueling DQN表現確實比較好，因爲他可以明確的學到state的分數以及action的分數，因此model可能會更偏向移動到state分數比較高的地方以獲取比較好reward。但是dueling DQN training的變化又更大了，我想是因爲output有兩個network，要兩個network一起合作輸出就會造成變動是兩個network加起來的。

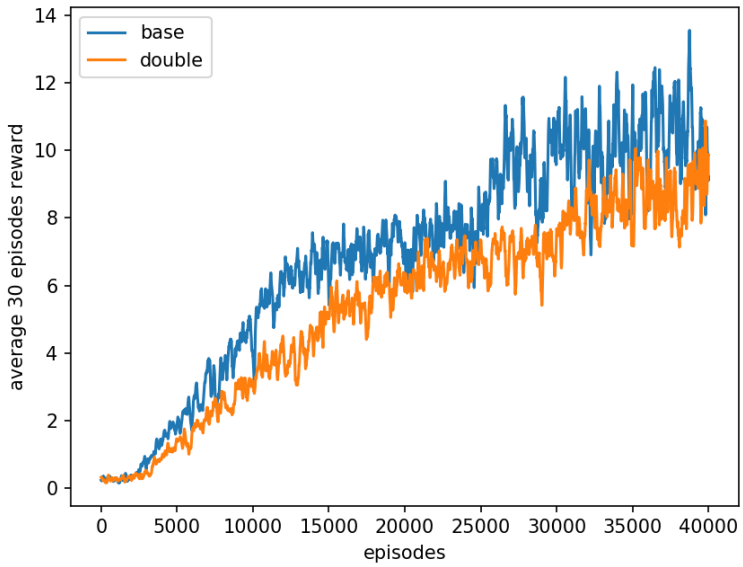


Figure 4: DQN learning curve (double DQN vs DQN)

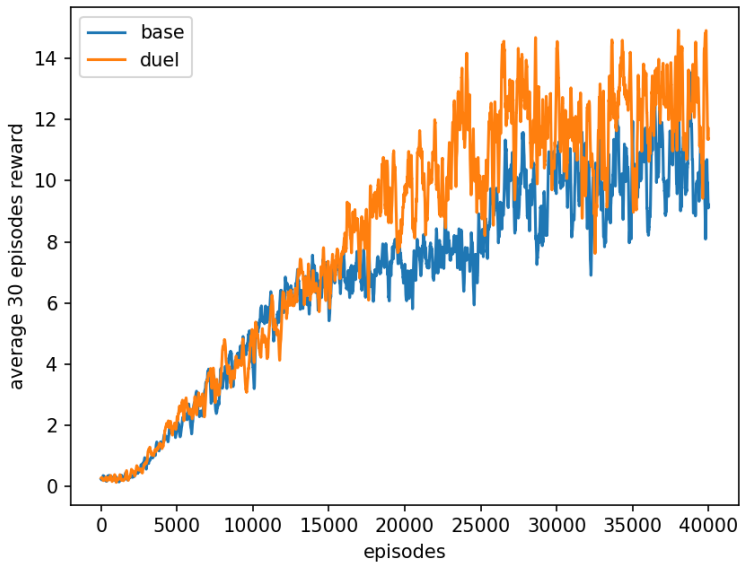


Figure 5: DQN learning curve (dueling DQN vs DQN)

4.2 Implement other advanced RL method, describe what it is and why it is better

我實作的是Asynchronous Advantage Actor-Critic(A3C)演算法，他合併了policy gradient和Q learning兩個演算法的好處，訓練一個model可以output每個action被選擇的機率，以及可以output每個state的分數(該state未來的reward)。核心的概念是有一個actor學習如何選擇action來得到最高的reward，搭配一個critic判斷actor的表現如何，若是actor現在得到的reward大於critic給的，那麼就鼓勵actor做這個動作，reward較低則相反。而critic也會學習如何打分數更準(output更準的state分數)。因此兩個model的目標函數就可以如下表示：

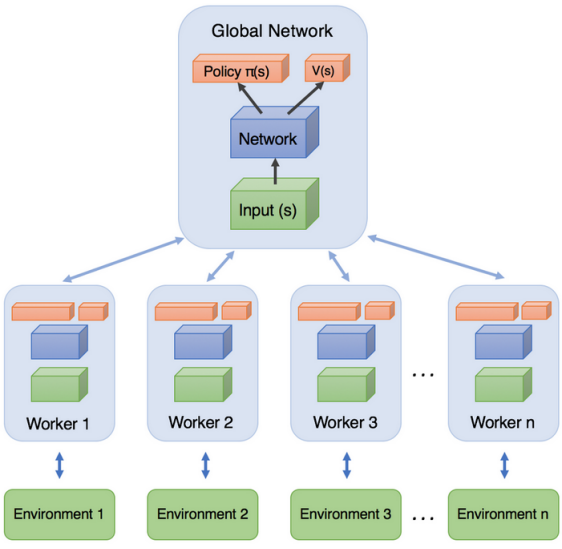
Actor要maximize以下目標:

$$\mathcal{R}(\theta^a) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(R_n^t - V(s_t^n;\theta^c))log(p(a_t^n|s_t^n,\theta^a))$$

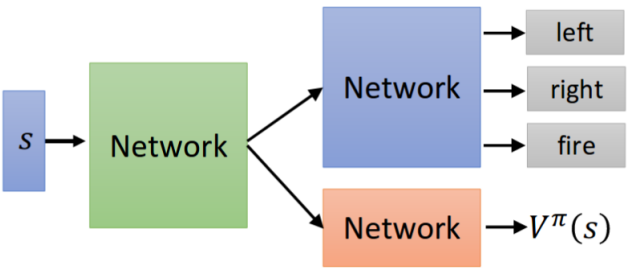
Critic要minimize以下目標:

$$L(\theta^c) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(R_n^t - V(s_t^n;\theta^c))^2$$

傳統的policy gradient只能玩完整場才知道策略的好壞，並且更新參數，這樣的作法缺點是variance太大。這個演算法相較於policy gradient的好處是可以訓練一個critic對每個state打分數，這樣就可以用Temporal-Difference(TD)的方式更新參數，但是這個state分數是這個state以後期望的reward，所以又有Monte-Carlo的性質，因此可以結合兩個方法的好處。而比DQN好的地方就是有actor可以自己決定該怎麼移動，而不是只是靠critic給的分數高低來移動。
而Asynchronous的意思是利用很多個agent在不同的environment setting下玩遊戲，因此學出來的model會遇過更多情況，表現也會更好。
最後是actor和critic的model可以共用參數，因爲他們都是看影片學習怎麼移動(打分數)，因此convolution layer的部份應該可以共用。(可參考Figure 6)



(a) A3C model (https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2)



(b) Actor and Critic share part of the model (MLDS 12/7上課投影片)

Figure 6: Learning curves, average reward of last 30 episodes to numbers of episodes

我A3C model的設定跟pg幾乎一樣，只是拿掉了batch normalization(加了結果差不多)，並且用了16個actor來訓練，設定如Table 3所示。 Figure 6 是我把a3c玩在pong上面的結果，可以發現的確比傳統的pg還要好，玩到250多場遊戲就可以接近10分(即使算所有actor總共的episodes還是比pg好不少)。只不過learning curve 看起來比pg崎嶇，我想應該是我learning rate調太大和因爲有16個不同的actor在訓練導致variance較大的緣故。

total episode	num of actors
270	16
optimizer, learning rate	**reward decay**
ADAM, 10^{-3}	0.99

Table 3: A3C model settings

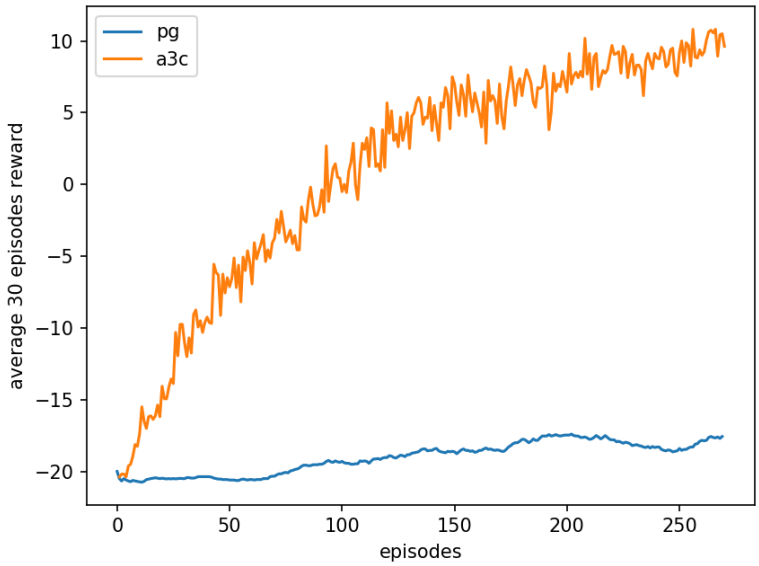


Figure 7: a3c and pg learning curve (playing Pong)

5 Reference

https://www.zhihu.com/question/56692640

https://www.csie.ntu.edu.tw/~yvchen/f106-adl/doc/171204+171207_DeepRL2.pdf

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-pa

https://arxiv.org/pdf/1602.01783.pdf

https://gist.github.com/sangmin082/461079044a6686b7566319b4b7afad4a#new_comment_field

https://github.com/transedward/pytorch-dqn

http://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html