HW4 Report

林哲賢、毛弘仁、王克安

HW4-1 Policy Gradient

Describe your policy gradient model (1%)

我們的 policy gradient model **只取 3 個 actions** 當 output：「往上」、「往下」、「不動」。也有試過只有「往上」和「往下」的 model，雖然訓練速度比較快，但是最後 converge 在 reward 比較小的地方。

Preprocessing 部分，我們將 **input image 降維成 80x80，轉為黑 (0) 白 (1)**，再壓成 6400 維的 vector。

這個 dim=6400 的 vector 會過**兩層 dim=256 的 feedforward layers**，最後再 feedforward 成 dim=3 的 output distribution。**Training** 的時候會從這個 distribution 做 **sampling**（機率不是最大的 action 也可能被執行），**testing** 的時候則**只選擇機率最大的 action**（我們發現這樣的 reward 比較大，同時也在固定 atari env 的 random seed 的狀態下，確定會過 baseline）。

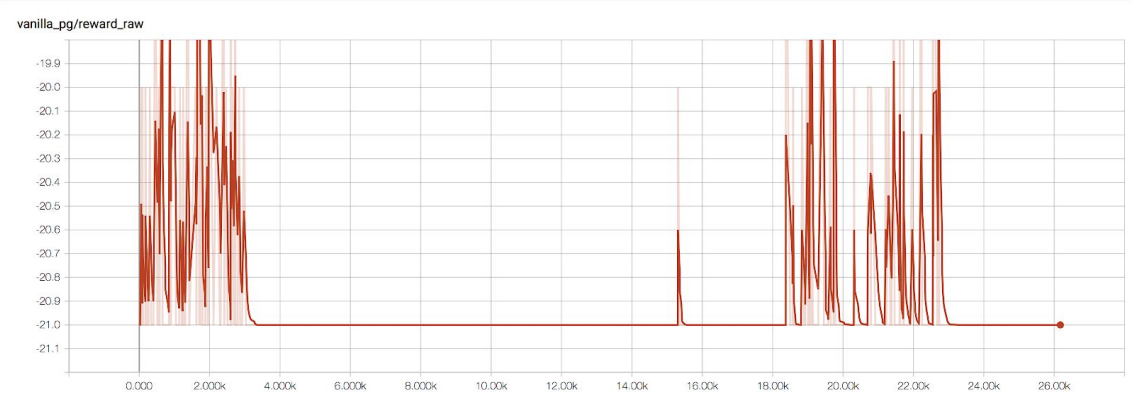
我們用了 **RMSprop** 當作 optimizer，並且將 learning rate 設成 **1e-4**。一整個 **episode** 結束（一方獲得 21 分）時，會 update parameters 一次。

在 training 過程，我們發現**如果 network 疊太深**，很容易在一開始只**重複同樣的 action**，沒辦法 sample 到新的經驗，推測**可能跟 overfit 單一 episode 有關係**（因為要拿到更大 reward 的話，有時候需要執行和原本 policy 差別比較大的 action sequence，例如「連續往上走 10 步」。

也試著**改用 CNN 過**，但有點意外的是，這樣**會 train 不起來**。我們推測是因為 **Pong 沒有什麼 higher-level features**（如人臉、物品等等）需要去學習，因此 feedforward 就可以成功「偵測」paddle 和 ball 的位置在哪裡（Andrej Karpathy 的 blog post 裡面也發現 feedforward network 確實可以對球去循跡），不需要另外訓練 convolution layer 去辨識。

另外還試著把影像縮小成 40x40，雖然人眼看起來和 80x80 沒有差很多，但一樣也讓 training 爛掉，所以將 **input 降維太多，也有可能讓 network 無發接收到夠多的資訊**。

Plot the learning curve to show the performance of your policy gradient on Pong (1%)



X 軸為 episode（21 分制的「回合」）數量，Y 軸為該 episode 的 reward。
由圖可發現，vanilla policy gradient 的表現很差，沒辦法 train 出好表現。

Implement one improvement method on page 8

Describe your tips for improvement (1%)

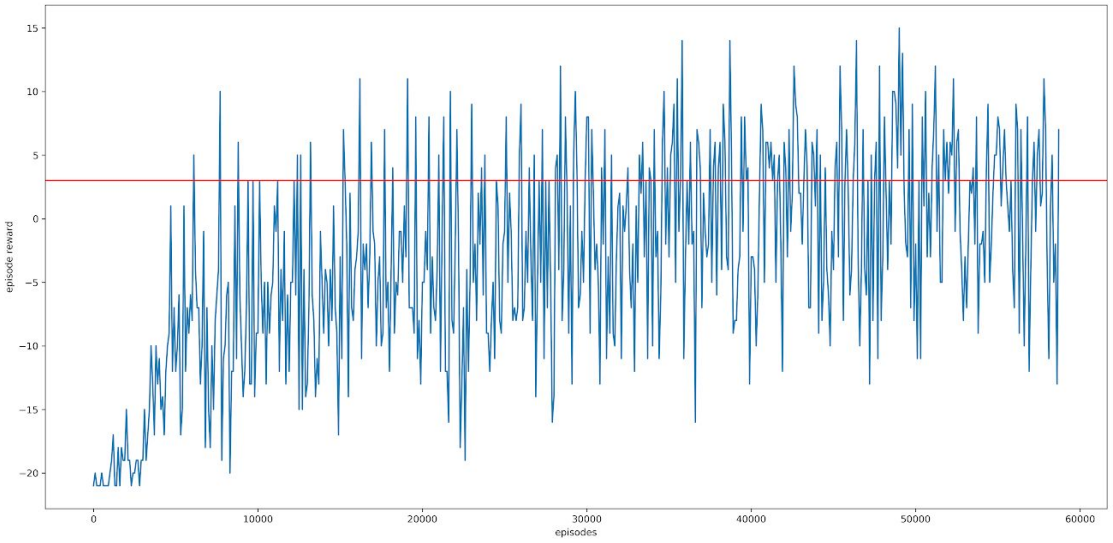
我們選擇使用 **variance reduction**，並且 implement 以下方法：

1. Reward discounting：我們的 gamma 設為 0.99，鼓勵 model 趕快獲得 reward

2. Consider causality：計算一個 action 對應到的 reward 時，只看那個 action 後面累積的 reward，不看前面發生的 reward

3. Add a baseline：參考了 Andrej Karpathy 的方法，將每個 reward 減去 mean 再除 standard deviation，讓大約一半的 action 拿到正 reward、一半的 action 拿到負 reward，發現效果比 running average 還好。

Learning curve (1%)



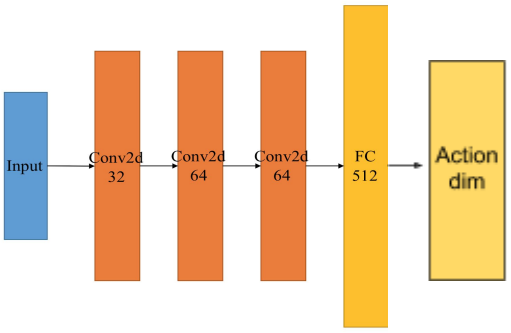
一個 episode 為 21 分制的「回合」。圖中紅線是 reward=+3，為本次的 baseline。
此圖中，model 的 action 是用 sample 的，不是取 max，因此表現會比我們最後拿來過 baseline 的 model 還差。

Compare with vanilla policy gradient (1%)

很明顯的，有加 variance reduction，會造成很大的進步！這次的作業，我們還參考了 Berkeley 的 DeepRL 課程內容，裡面一直提到 **policy gradient 會面臨到 high variance problem**，並且介紹了我們這次 implement 出來的方法來解決。看完課程也發現，**actor-critic 也算 variance reduction 的一種方式**，不過代價是會 introduce bias。因此如何去拿捏 **bias-variance tradeoff**，也是直得思考的問題！

HW4-2 Deep Q-Learning

Describe your DQN model (1%)

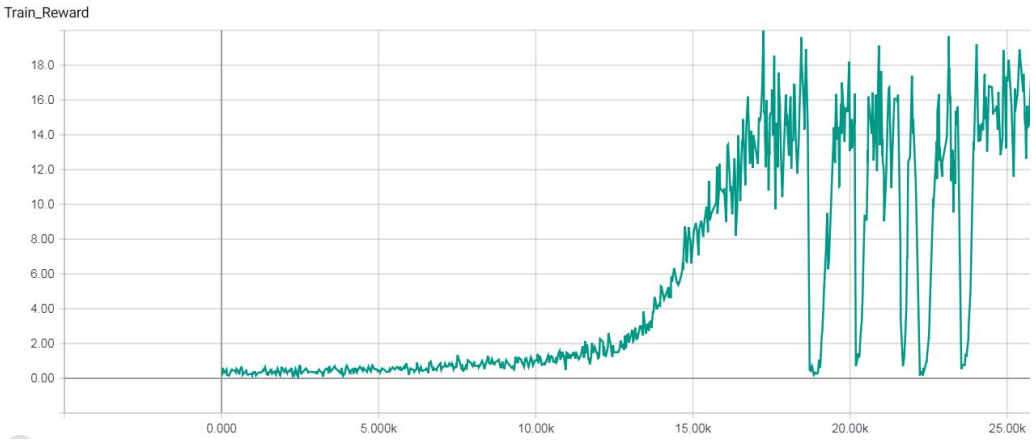


Input部分為Atari wrapper處理過的圖像，每一張為84*84，pixel value 除過 255的圖，而4個channel則是4個frame，再經過三層的convolution以及DNN之後，出來會是一個action dim的vector。在中間所有的**activation function，**我都是使用**relu**，比較特別的是，我把action print出來後發現是以下四種['NOOP', 'FIRE', 'RIGHT', 'LEFT']，而我覺得不要有NOOP的動作比較好，因此我把model 的**action dim設成是3**，分別代表'FIRE', 'RIGHT', 'LEFT'3個action。

演算法的部份我是照著ppt上面刻，而我的loss function是使用**mean square error loss**，**optimizer**是使用**RMSprop**，**lr = 0.00025**，以下是我的一些參數：

Exploration 是用 linear schedule 到1000000 會降至0.1
Replay buffer size : 50000
batch_size : 32
Gamma : 0.99
Perform Update Current Network Step : 4
Target_update_freq : 1000

Plot the learning curve to show the performance of your DQN on Breakout (1%)

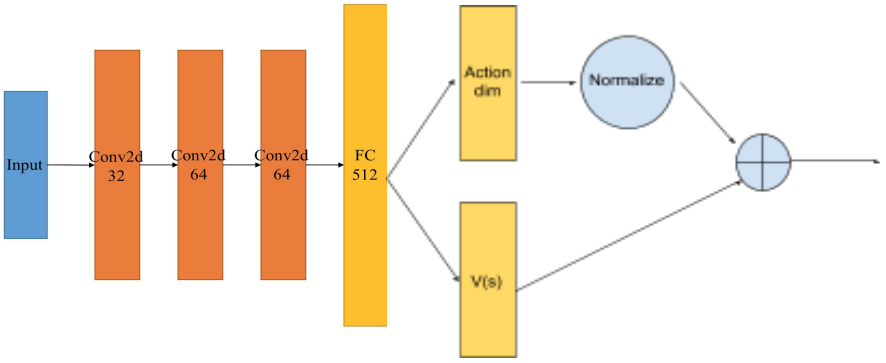


X-axis 為 episode 的數目，Y-axis 為 average reward in last 30 episodes

Implement one improvement method on page 6

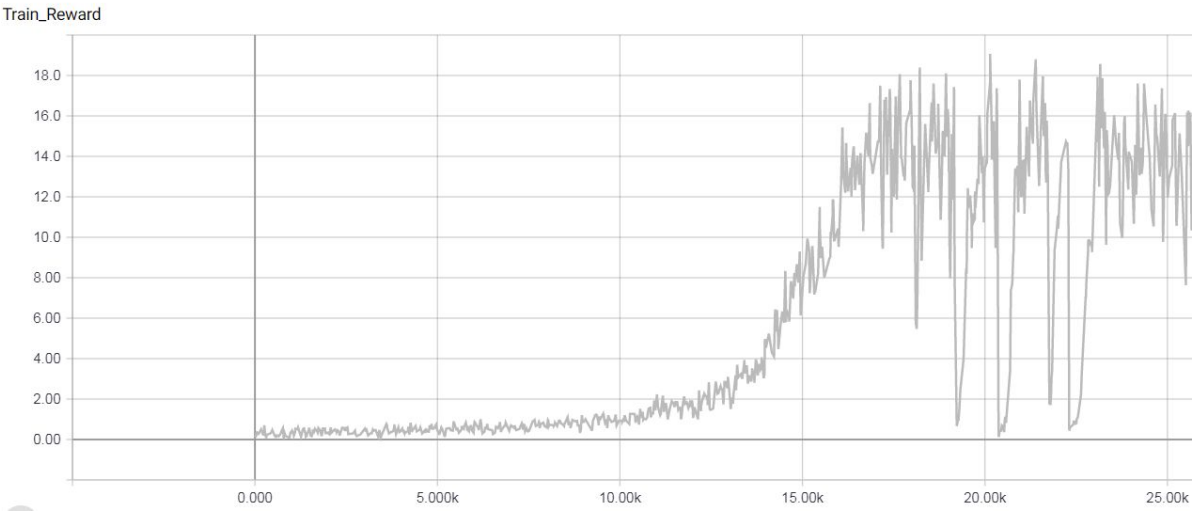
Describe your tips for improvement (1%)

在這裡我implement的是**Duel DQN**，而這個tip和上面DQN唯一的差別就只在於model長得不一樣而已，其他的參數設定都一模一樣，因此不再贅述(詳見前一部分)，以下是我duel DQN的model：



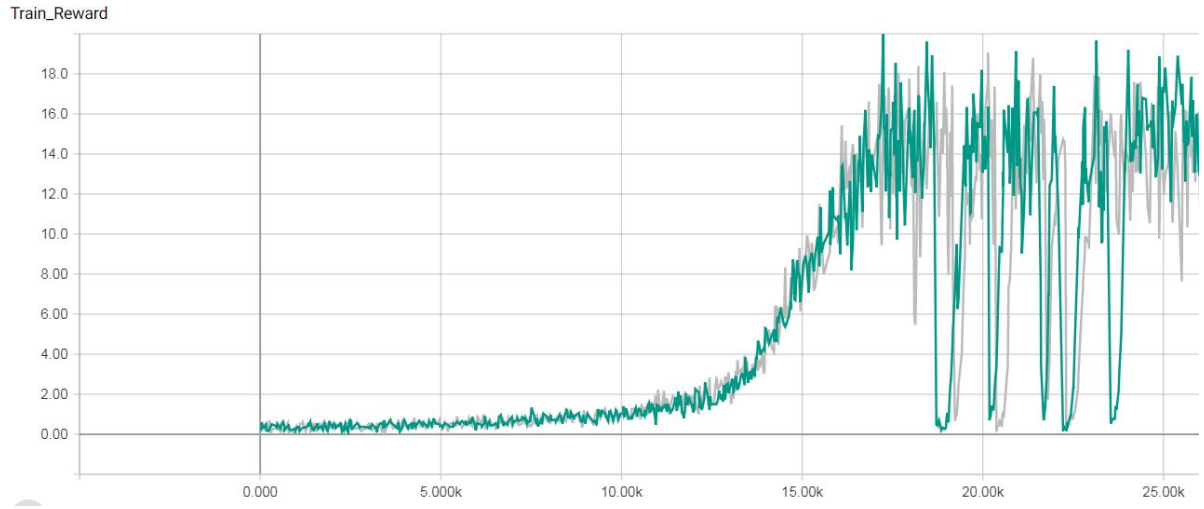
可以看到和DQN不一樣的地方是，FC512出來以後並非接一個DNN然後output，而是分成了兩條路，上面那條(**Advantage**)和原本的一樣，只是在output的地方要做normalize，也就是要把sum弄成0。下面那一條路(**Value**)則是output出一個scalar，最後兩個值相加當作output Q(s, a)，這樣做的好處是，在某個state明明沒有sample到某個action，但是卻可以因為改v而update到。要做到這一點，就要讓上面那條路比較難update，所以我們會normalize後才加在一起。

Learning curve (1%)



X-axis 為 episode 的數目，Y-axis 為 average reward in last 30 episodes

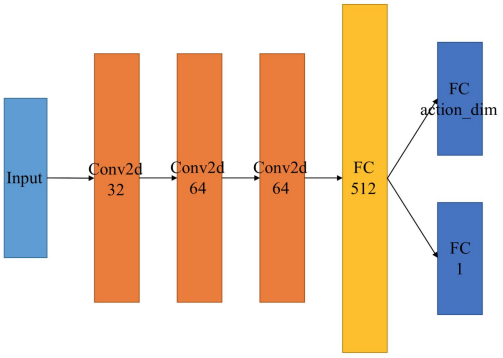
Compare with original DQN (1%)



上面的圖是我把DQN的learning curve和duel DQN的learning curve疊在一起的結果，可以看得出來雖然灰色的平均在18Kepisode以前有高過綠色的線一點點，但是效果並不顯著，可能的原因是在這個breakout這個task state常常重複(連續撞到牆然後沿著原本的軌跡彈回來)，而random action可能都會sample到，因此v(s)效果相對來說比較微弱一點。

4-3 Actor Critic

Describe your actor-critic model on Pong and Breakout (2%)

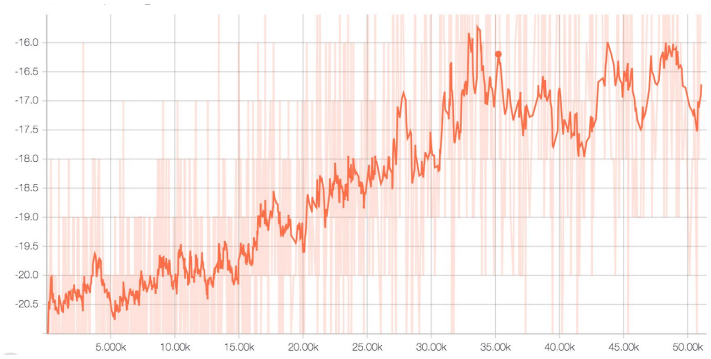


上圖是我們使用的模型。optimizer為RMSprop，learning rate為2e-4。pong跟breakout都是train了50000個step。

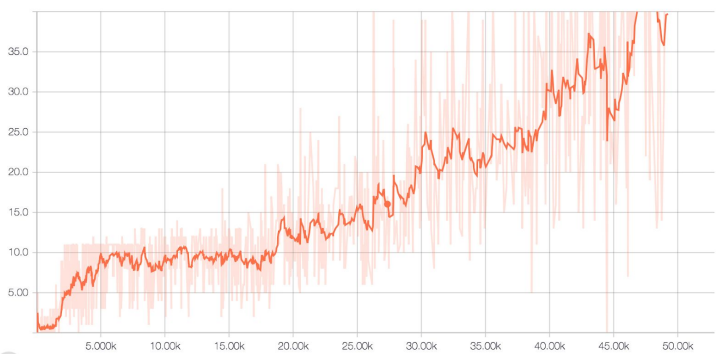
Plot the learning curve and compare with 4-1 and 4-2 to show the performance of your actor-critic model on Pong & Breakout (2%)

以下的圖，X-axis 皆為 number of time steps，Y-axis 皆為 average reward in last 100 episode。

Pong



Breakout



從**Pong**的學習曲線可以看到，在train了50000個step之後，雖然有在進步，但reward還是負的，這個情況**沒有比policy gradient還要好**。而在**Breakout**的部分，在同樣是25000個step的地方reward**幾乎是一樣的**。我們的實驗結果說明在4-3的模型架構之下，**actor-critic的表現並沒有比較突出**。

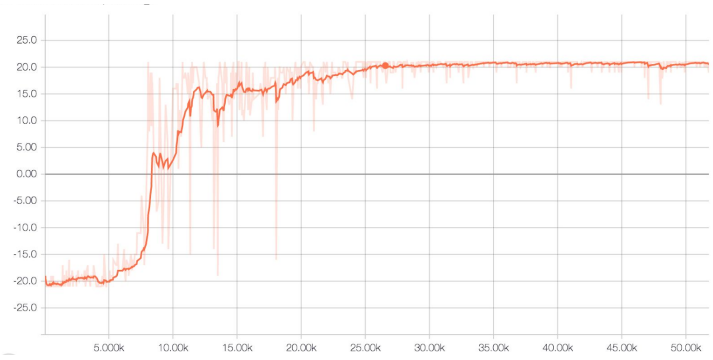
Reproduce 1 improvement method of actor-critic (Allow any resource)

Describe the method (1%)

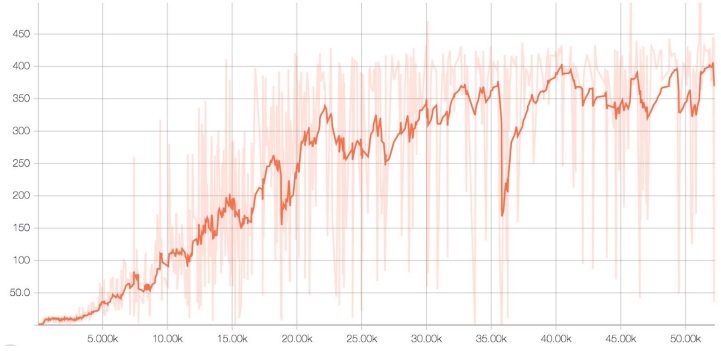
我們使用的是**A2C**，跟普通的actor-critic差別在A2C是學習**advantage function**，而正常的**actor critic是學習action value function**。

Plot the learning curve and compare with 4-1, 4-2, 4-3 to show the performance of your improvement (1%)

Pong



Breakout



可以明顯看到，在同樣的training step數量之下，**A2C的表現是比較好的**。在50000個step內都可以拿到相當高的分數，比同時間4-3的model有顯著進步，而跟4-1, 4-2相比，雖然有一些training step的結果不同，但還是有很大的進步。