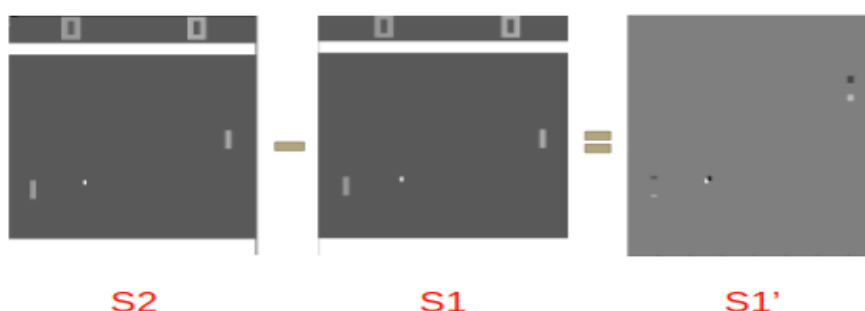


4-1

Describe your Policy Gradient model (1%)

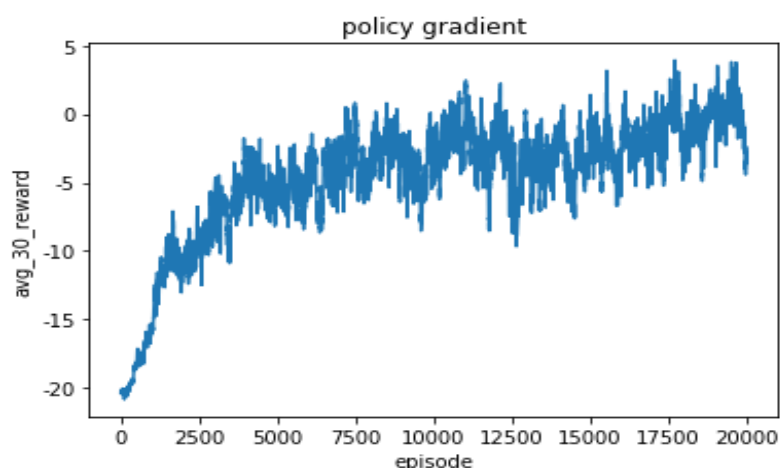
前處理：將圖片做 slice 把邊邊和分數板給去除然後轉成黑白兩色（黑色代表物體，板子和球）。留下 80x80 pixel 的圖片。將預處理的圖片做差分的動作（這一刻和減去上一刻）取出 Residual State。Residual State 直接餵進 Network。

Residual State



使用的 model 是兩層的 fully connected layer, 每一層兩百個 neuron, 最後通過 sigmoid 輸出機率。這裡只有留下上跟下兩種動作, 所以最後一層預測的是往上（不往下）的機率。

2. Plot the learning curve to show the performance of your Policy Gradient on Pong (1%)



可以看到在 reward curve 上面, 大約在 5000 個 episode 後就趨於緩慢的震盪向上, 之後上升速度很慢, 因此我們就慢慢的調小 learning rate, 大約到三萬 (圖

中只有顯示到兩萬，因為兩萬後面的 episode 沒有紀錄到 reward)就能超過 baseline。

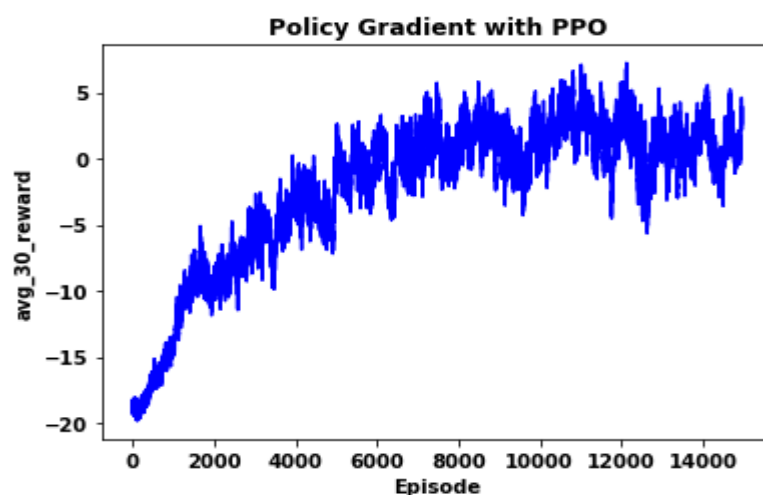
3. Implement 1 improvement method on page 8

a. Describe your tips for improvement (1%)

$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k)$$

這裡使用的是 proximal policy optimization (PPO)，利用上課所提的式子，不希望 θ 和 θ' 的差距過大，加上一些額外的限制(KL divergence 差距不能太大)限制參數更新的速度，減緩原本 policy gradient 參數更新不穩定的狀況。

b. Learning curve (1%)



c. Compare to the vanilla policy gradient (1%)

加上了限制參數更新的條件，在過程中發現抖動還是很厲害，但是整體來說似乎有往正確方向更新的趨勢(爬升的趨勢似乎比較好)。

4-2

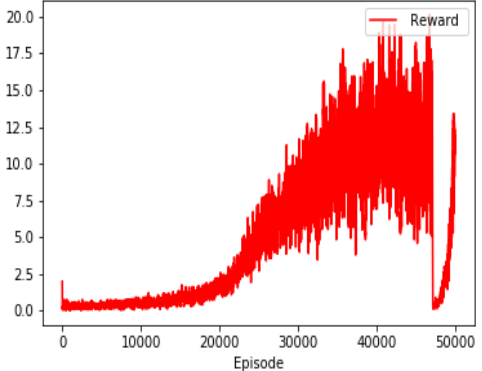
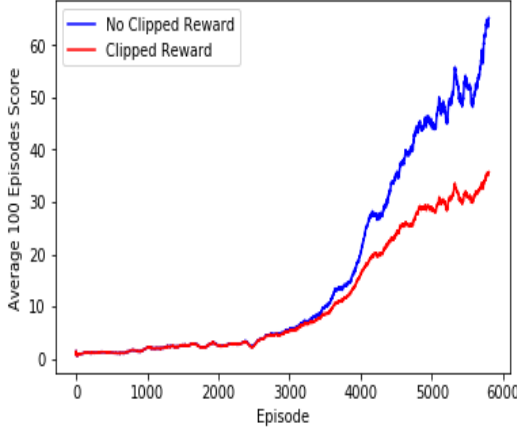
1. Describe your DQN model (1%)

我們採用的 Model 跟 Paper 上的架構一樣，唯一不疼就只是把最後一層 Fully Connected Layer 改成 size = 512。處理方式是把連續的四個 time frame 做色彩處理之後 stack 在一起當作 input 餵進 Online 的 Q-network，Q 的輸出分別代表每

個 action 所對應的 Q 值，我們根據 epsilon，來選擇我們的策略，並在每固定 iteration step 之後更新我們的 target Q net。Epsilon 會隨 Iteration 遞減，並在一定 iteration 之後降為 0 (我們也有做一個沒有降到 0 的版本，效果上感覺穩定一些，但由於另一個是用 tf 寫的，所以也很難真的比較差異。)

另外不確定什麼原因，Tf 的版本跑比較慢，但是學習比較穩定，Torch 就不太穩定，但比較快(最終上傳版本)。推測原因可能是兩個使用的 loss 不太一樣，我們 torch 版本使用 MSE Loss 而 tf 版本使用 Huber Loss。

2. Learning curve (1%)

	
<p>This is from our second implementation using Pytorch (Clipped reward averaging through 30 recent episodes)</p>	<p>This is from our first implementation (using TF) (Both Clipped and not_clipped averaging through recent 100 episodes)</p>

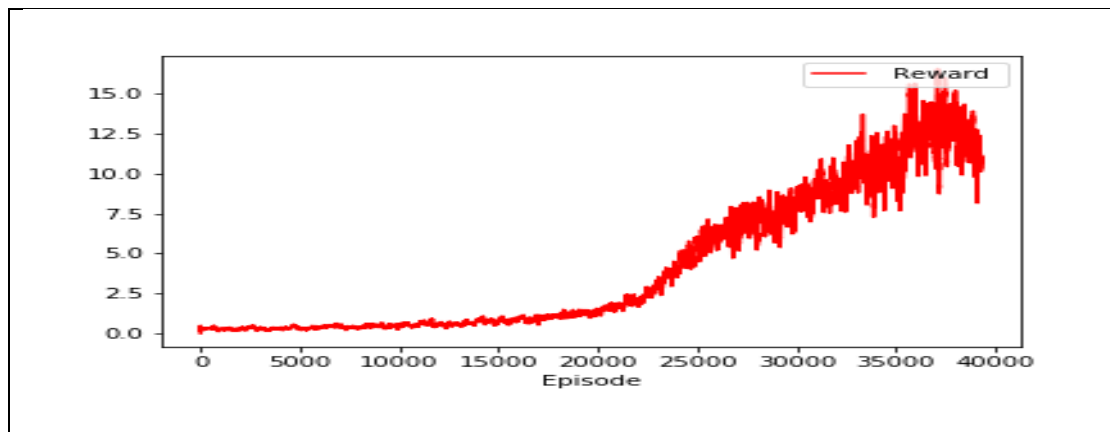
3. Describe your tips for improvement (1%)

Double DQN 是我們採取的改進方式，其實也就改一點點 code 而已，原本的 DQN 中我們是把正確的 Q 值假設成是 $r_t + \gamma * \text{Max}_a(Q_{\text{target}}(s', a))$ 來對 Online 的 Q 值進行更新，這樣常常會有過度樂觀估計 Q 值的問題發生，主要原因是透過這樣得迭代，我們只能保證兩個 action 所造成的相對 Q 的差正確，但真正絕對的 Q 卻可能更小。DDQN 為了緩解這樣的問題，把上面的式子改寫，加入另外一個 Q 做把關，因此我們的目標變為 $r_t + \gamma *$

$\text{Max}_a(Q_{\text{target}}(s', \text{argmax}_a(Q_{\text{online}}(s', a))))$ ，如此變換的結果就是保證得到的

$Q \leq$ 原本 DQN 中獲得的 Q 。其中等號只有在某個 a 同時是 Q_{target} 以及 Q_{online} 的 Maximizer 時才會成立，其餘時候都是 \leq 的情況，這樣理論上就能有效地減緩 DQN 中 Q 常常被 Over-Estimate 的狀況，但其實並沒有有一定保證這樣得到的最後結果一定比較好。

4.5. Learning Curve and Compare to original Deep Q Learning (2%)

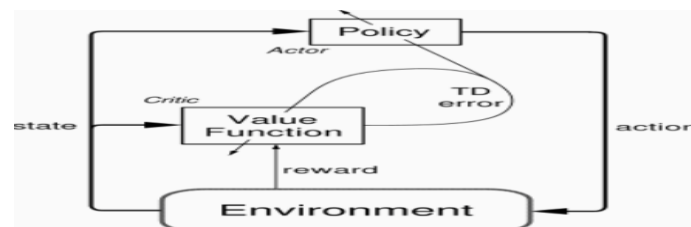


儘管沒有表現比較好，但可以發現訓練的過程中，結果是穩定許多的。不過缺點就是 train 起來比原本就很久久的 DQN 還要更久。

4-3

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

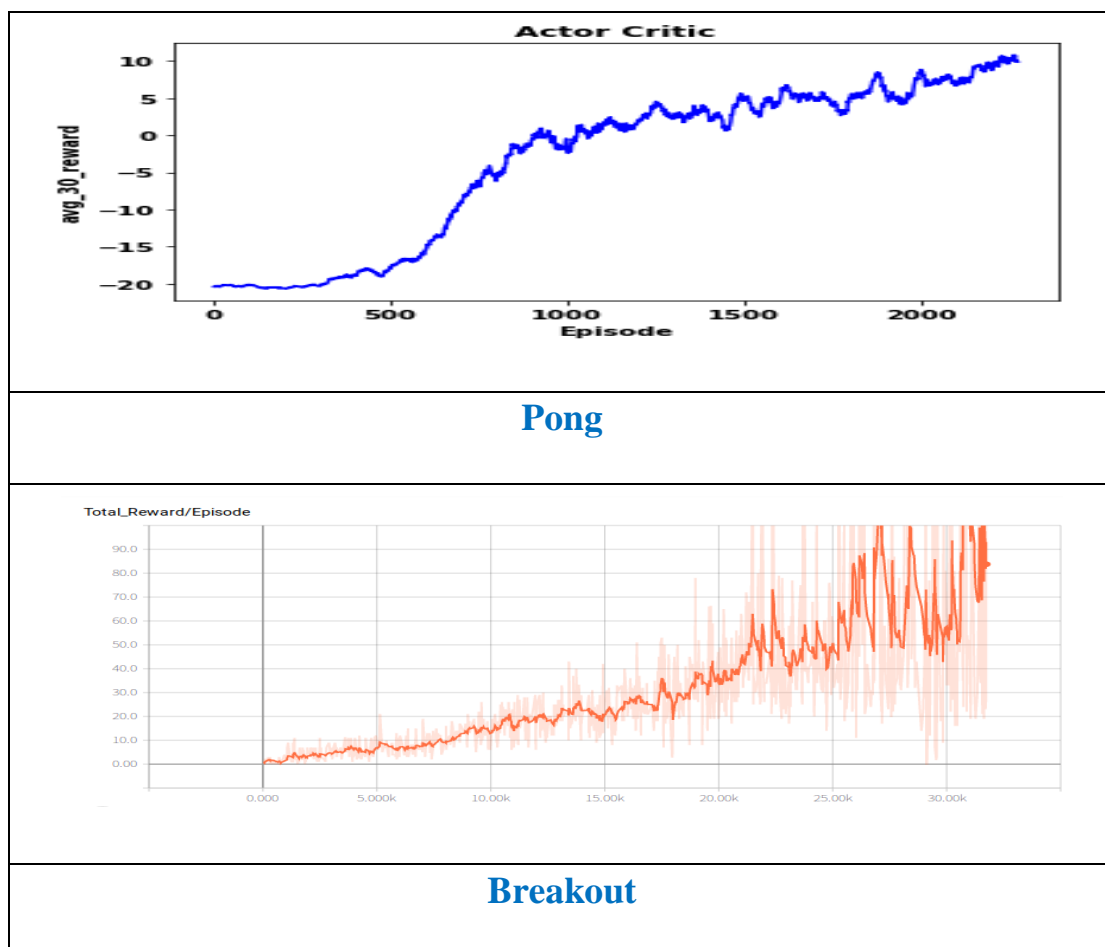
Pong: 圖片的前置處理和 policy gradient 相同(取 80x80, Residual State)，但是輸入時使用兩層 CNN(第一層 kernel size = 8*8, stride = 4, filter = 16；第二層 kernel size=4*4，stride=2, filter=32)，最後 flatten 後經過一層 NN(size = 256)。Critic 更新的時候使用 TD error，Actor 使用 Policy Gradient 方式做參數更新。



source: <https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/6-1-actor-critic/>

Breakout: Actor 和 Critic 的模型在前面的 Conv layer 都長一樣，只差在最後一層 Dense 層 Actor size 是 3 (動作數)而 Critic 是 1，代表 $Q(s,a)$ 。與單純的 DQN 比起來，這裡的 Conv 層少了一層，但結果並沒有比較差。

2. 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%)



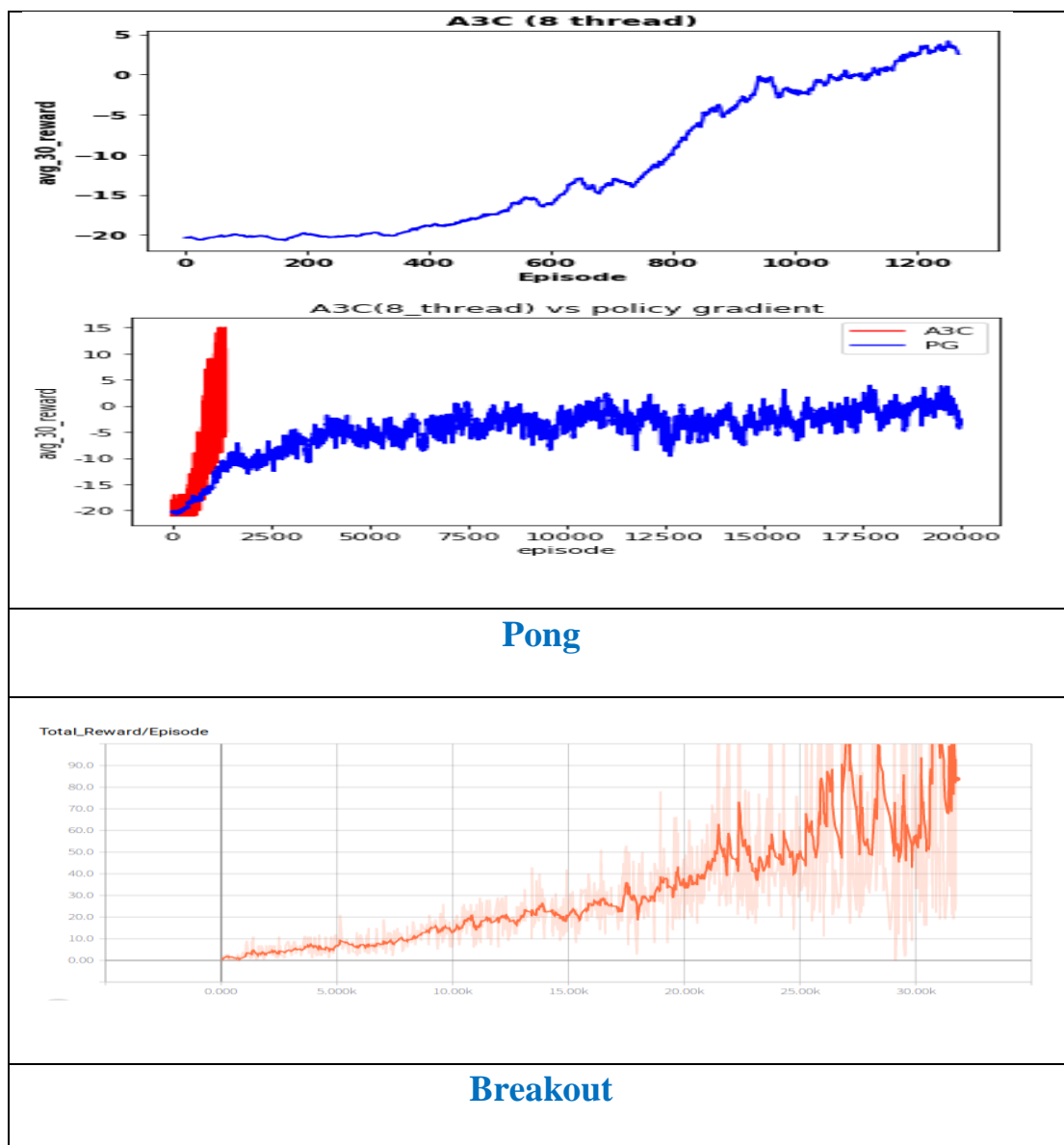
3. Reproduce 1 improvement method of actor-critic

a. Describe the method (1%)

Pong: A3C, 前置處理(包括圖片的處理、網路架構)和 actor critic 一樣，但在訓練 agent 的時候開了 8 個 thread 同時更新參數，可以達到不錯的效果。

Breakout: A3C，除了 thread 數以外其他都維持不變，同時開 8 個 thread，讓處理的效率有效提升，而結果接近單純 Actor-Critics.。

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



分工表

王垣尹	許哲瑋	李中原
4-2 tf	4-1	4-2 pytorch
4-3-2(breakout)	4-3-1(breakout)	4-2 finetune