MLDS HW4 Report

4-1 Policy Gradient (Pong)

1. Describe your model

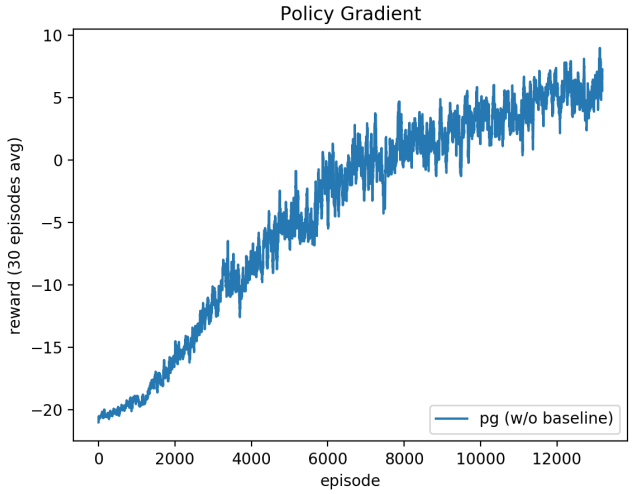
我們這次使用的 model 較小，其架構如下：

細節如下：

Observation size	(80, 80, 1)	Gamma	0.99
Optimizer	Adam	Weight Initialize	Xavier_normal
Learning rate	0.001	Total episodes	13,000

其中初始化使用 Xavier normal，80 * 80 為經過 preprocess 後的大小，訓練時採用 residual state。訓練過程使用的 optimizer 為 Adam，learning rate 為 1e-3，每完成一個 episode 就更新一次 model，訓練時我們讓 model 只會做出向上或向下兩個動作，由於只有兩種故輸出經過 sigmoid，而 loss 使用 binary cross entropy，其中我們的 reward 有經過 discount，gamma 為 0.99。

2. Learning curve [x: episode] - [y: 30 episodes average reward]



圖中 x 軸為 episode，y 軸為 30 個 episode 內的 reward 平均，reward 採用 21 分制為一局，採計最後的分數差。

3. Implement one improvement

a. Describe your tip

我們實作的 improvement 為 variance reduction，方法為增加 baseline。參考這份投影片：http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_4_policy_gradient.pdf。

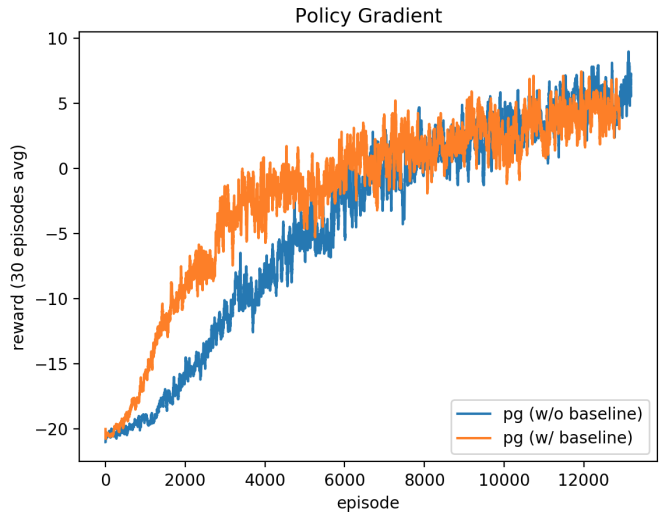
Variance 可以寫成：

$$\mathrm{Var} = E_{\tau\sim\pi_\theta(\tau)}[(\nabla_\theta \log \pi_\theta(\tau)(r(\tau)-b))^2] - E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau)-b)]^2$$

若要得到 baseline b 使 variance 最小，可以對 b 微分，得到：

$-2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$，也就是：$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$

b. Learning curve



圖中 x 軸和 y 軸意義同第二小題，比較有無 baseline 作為 variance reduction 所帶來的差別。

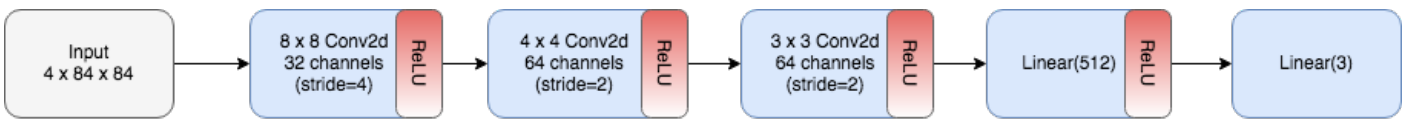
c. Compare to policy gradient

注意到在訓練過程的開始，有使用 baseline 做 variance reduction 在相同的 episode 數量下 reward 上升的速度較快，但到了大約 6000 個 episode 後，兩者的 reward 上升幅度就變得差不多，在 12000 個 episode 後都訓練到大約 5 的 reward。可以推測 baseline 能夠幫助訓練前期的收斂速度，但在訓練一段時間後則影響較小。

4-2 Deep Q Learning (Breakout)

1. Describe your model

我們這次使用的 model，與 policy gradient 時的 model 有點相類似。大致上架構如下：

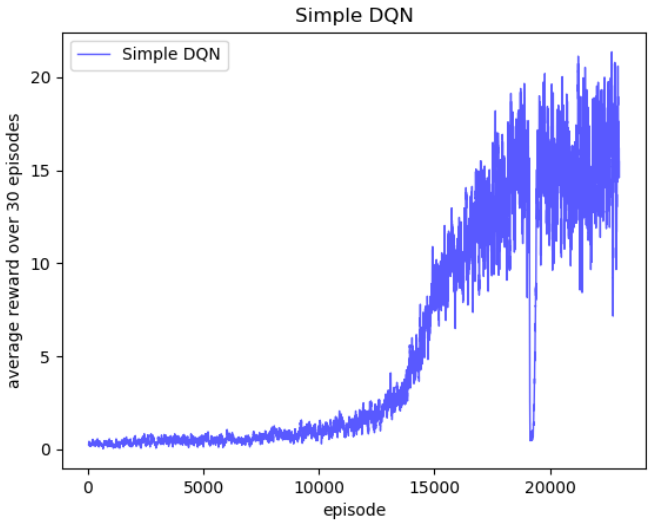


而主要的參數如下表：

Observation size	(84, 84, 4)	Target Q update freq.	1000

Batch size	32	Q update freq.	4
Optimizer	RMSprop	Epsilon Max	1.0
Learning rate	0.00015	Epsilon Min	0.025
Gamma	0.99	Exploration steps	1,000,000
Weight Initialize	Xavier_normal	Total episodes	25,000

2. Learning curve [x: episode] - [y: 30 episodes average reward]



其中 learning curve 的橫軸為 episode，縱軸則是 **clip 後**的 reward。發現在 12000 多 episode 之後 reward 開始慢慢上升，同時這時也是 step 超過 100 萬，我們此時將 epsilon 設為 0，action 不再是 random sample 出來的。前面的 exploration 的部分結束之後，model 從這階段之後開始學習如何基於 explored 過的環境來與目前的環境互動。而大致上在 17000 episodes 之後的 testing reward (沒有 clip) 可以大約超過 baseline 的 40 分。而 model 在自己的學習過程中，reward 是相對不穩定的，例如我們可以看到在 17000 episode 之後超過 baseline，在 18000 episode 左右出現 reward 突然的下降的狀況，但在數個 episode 有回復到原本的趨勢。

3. Implement one improvement

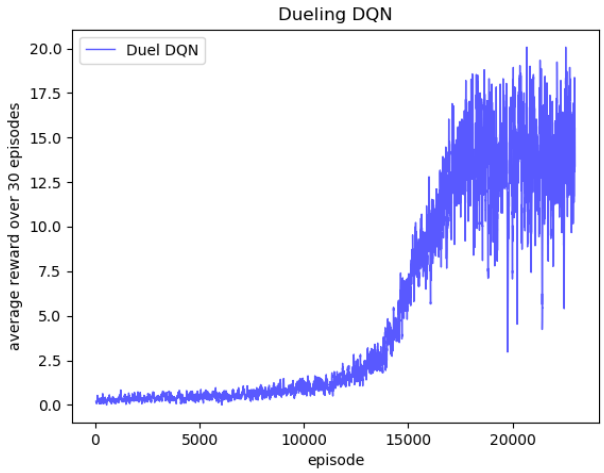
a. Describe your tip

我們這次使用 Dueling Network 的方法。其方法最主要目的就是加快原本 DQN 的收斂速度。在傳統的 DQN 上，在文獻上發現 model 較容易 overestimate Q 的值。因此在近年來，後人提出不同方法來改善這樣的情況。

Dueling Network 主要是將原本的 Q function 拆成兩部分，分別為 Value 和 Advantage，用式子簡單表達為：Q(s, a) = V(s) + A(s, a)。Advantage 的部分通常對下一個

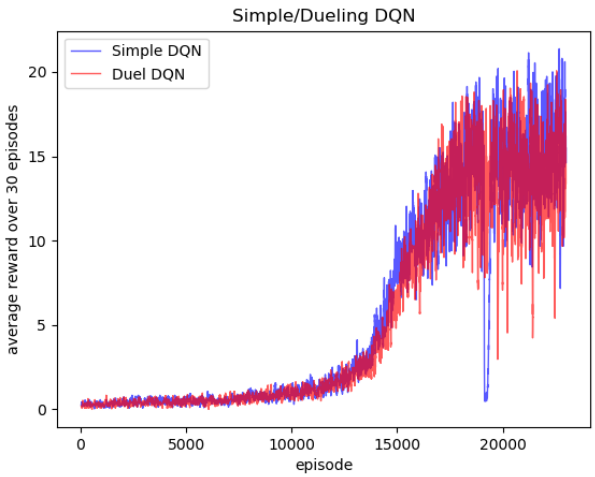
state 比較沒有太大的影響，但提供了 model 多一個 state 的估計，加強其穩定性，並且透夠這樣改變的架構，可以提升 model 學習的速度。

b. Learning curve



其中 learning curve 的橫軸為 episode，縱軸同樣也是 **clip 後**的 reward。發現在 12000 多 episode 之後 reward 開始慢慢上升，原因與前面 Simple DQN 的理由相同。而大致上在 17000 episodes 之後的 testing reward (沒有 clip) 可以能超過 40 (baseline)，甚至可以突破 60。

c. Compare to DQN



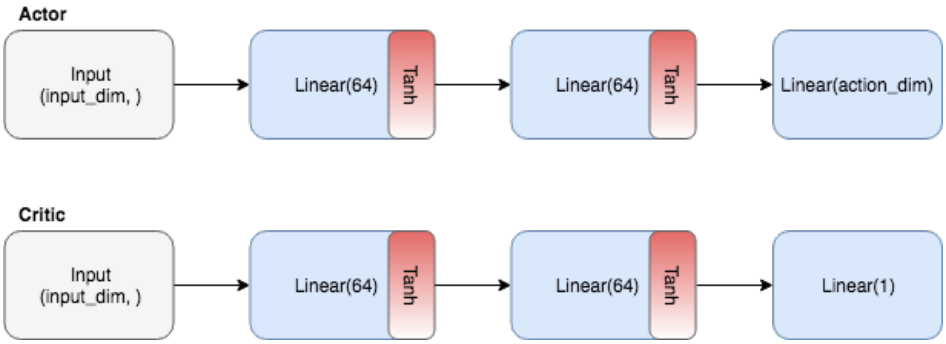
我們將以上兩個 learning curve 綜合起來作比較，看兩者的差異。但在以上的圖我們發現兩者的 learning curve 在 **clip 的 reward** 沒有特別的差異，但可能在 **沒有 clip reward** 會有較顯著的差異，因為 clip reward 將 stack 的 4 個 frame 所可能得到 1 ~ 4 的 reward 都變成 1，因此有可能像上述所說，在 episode 17000 左右，兩者所得到的 testing reward 有顯著差異。

以下再條列幾點原因造成這樣的現象。第一是可能原本的 simple DQN 沒有出現明顯的 overestimate，從上圖有看到 Simple DQN 的 reward 有稍微較高一些，但沒有超過多少，因

此我們的 improved model 的效果並沒有那麼顯著。再者，可能是 Advantage 可能 constraint 的不夠強，model 可能用自己另種方式鑽漏洞，因此學習的過程沒有特別收斂的比較快。

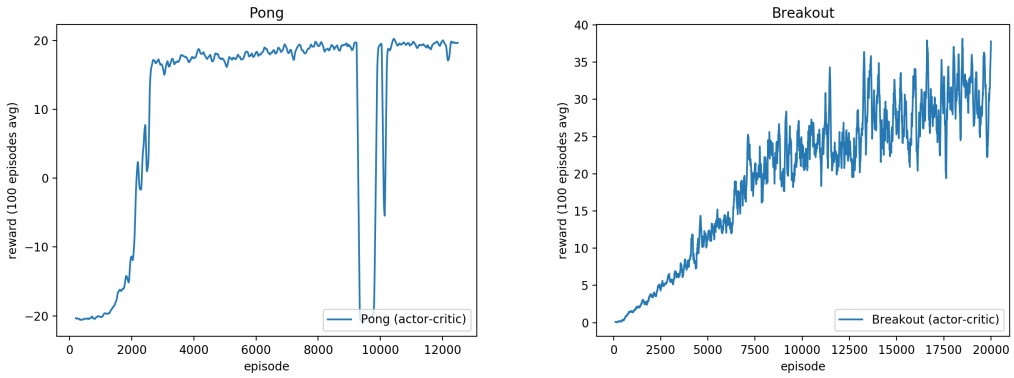
4-3 Actor-Critic (Pong + Breakout)

1. Describe your model



模型如上，訓練過程使用的 optimizer 為 RMSprop，learning rate 為 5e-4，其中 pong 約訓練了 13000 個 episode，而 breakout 約 20000 個。input_dim 取決於 environment，若為 pong 則 input_dim 為 1x80x80，breakout 為 4x84x84，與 4-1、4-2 相同。

2. Learning curve [x: episode] - [y: 100 episodes average reward] (pong + breakout)



兩張圖的 x 軸為 episode，y 軸為 100 個 episode 的 reward 平均，其中 breakout 的 reward 有經過 clipping。

3. Reproduce one improvement

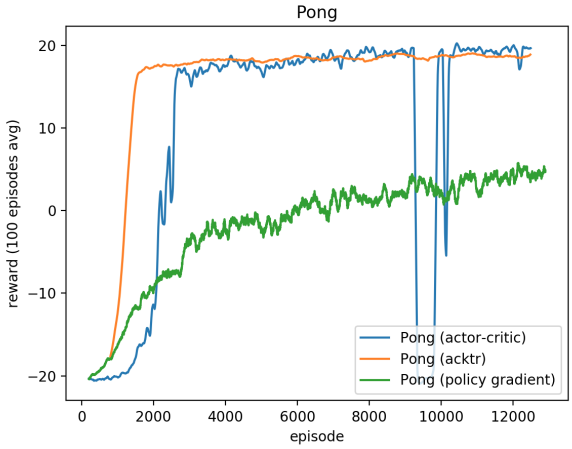
這次選擇的是 ACKTR 這個 improvement，我們使用這個實作 (https://github.com/ikostrikov/pytorch-a2c-ppo-acktr) 來試著重現 ACKTR 這個改進。

a. Describe the method

ACKTR 方法主要是將 KFAC 運用在原本的 A2C 等方法上，支援離散或連續控制，他結合了 actor-critic、trust region optimization 和 Kronecker factorization (KFAC) 等方法，相比其他方法他有更好的 sample 複雜度，所以在訓練過程上能更有效率。同時由於使用了當時剛提出的 KFAC 矩陣分解方法，故計算複雜度上也更低，使他成為第一個 scalable 的 trust region natural gradient 方法。

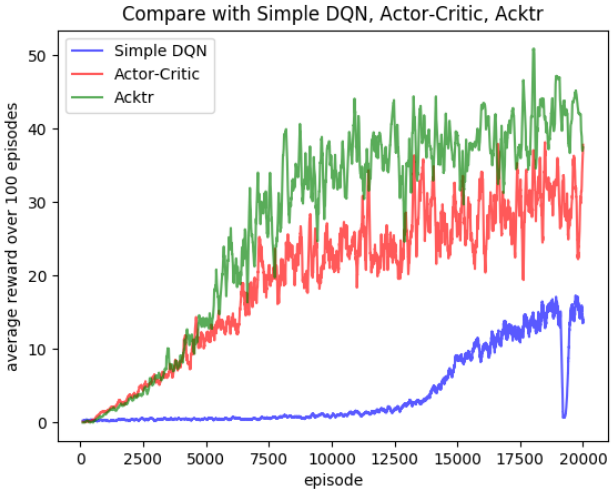
b. Learning curve and compare with 4-1, 4-2, 4-3

首先比較 4-1 和 4-3 的 Pong，其 learning curve 如下圖：



注意到 actor-critic 和改良 acktr 都能在更少個 episode 內收斂到更高的 reward，其中 actor-critic 和 acktr 都能收斂到接近 20，也就是非常接近滿分 21，而 policy gradient 則還沒收斂，但 reward 大約在 5 附近，故可以得出結論為 actor-critic 類的做法能在較短的時間內收斂到更好的結果。另外比較一般的 actor-critic 和其改良 acktr，注意到一般 actor-critic 在訓練過程中出現了一次非常劇烈的 reward 下降再上升，換言之他的訓練過程較不穩定，而 acktr 則可以發現整個訓練過程都非常平滑。

接著比較的是 4-2 和 4-3 的 Breakout。



可以觀察到 actor-critic 在訓練過程一開始 reward 就不斷上升，而不像 DQN 到了大約 12000 episode 附近才開始出現明顯的變化，故 actor-critic 在 breakout 這個環境下表現要比 DQN 好一些。另外比較 ACKTR 和 vanilla actor-critic，注意到兩者在訓練前期的上升幅度差不多，但在 5000 個 episode 附近開始 ACKTR 上升速度更快，最終分別達到 30 和 40，故 ACKTR 表現稍微好一些，不過在 breakout 的結果中不如 pong 中可以觀察到 AKCTR 的訓練過程較平滑。