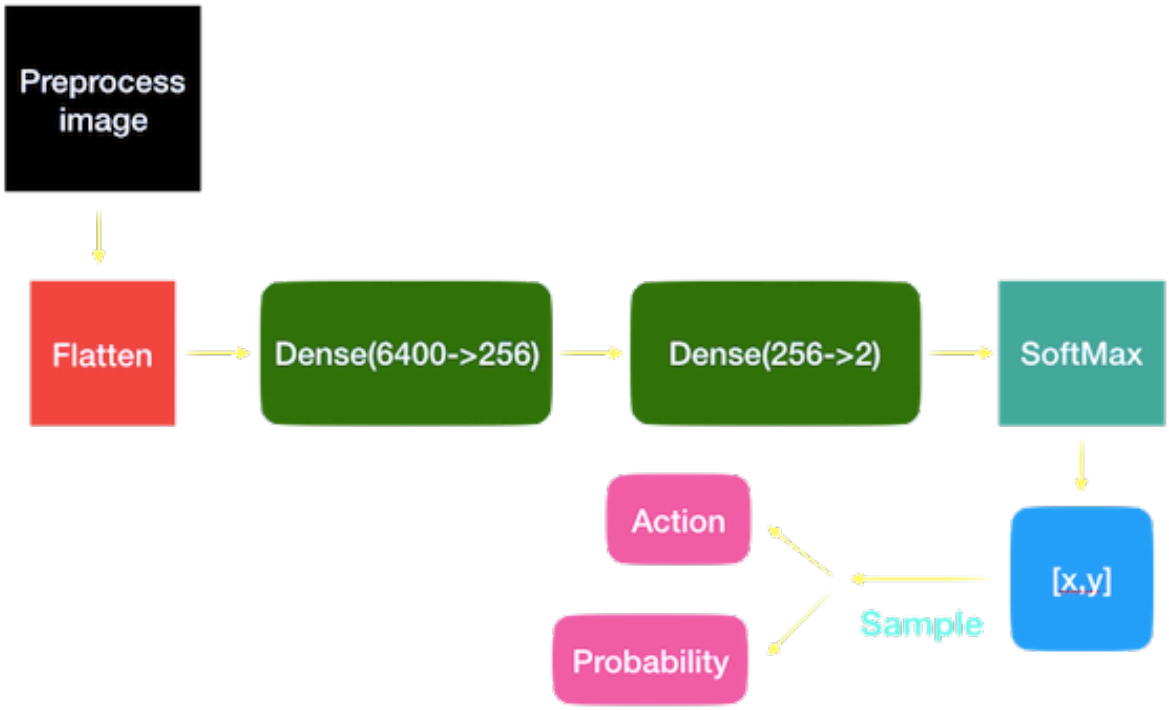
MLDS HW4 Reinforcement Learning

學號：b05902031, 系級：資工二, 姓名：謝議霆, 學號：b05902008, 系級：資工二, 姓名：王行健

4-1

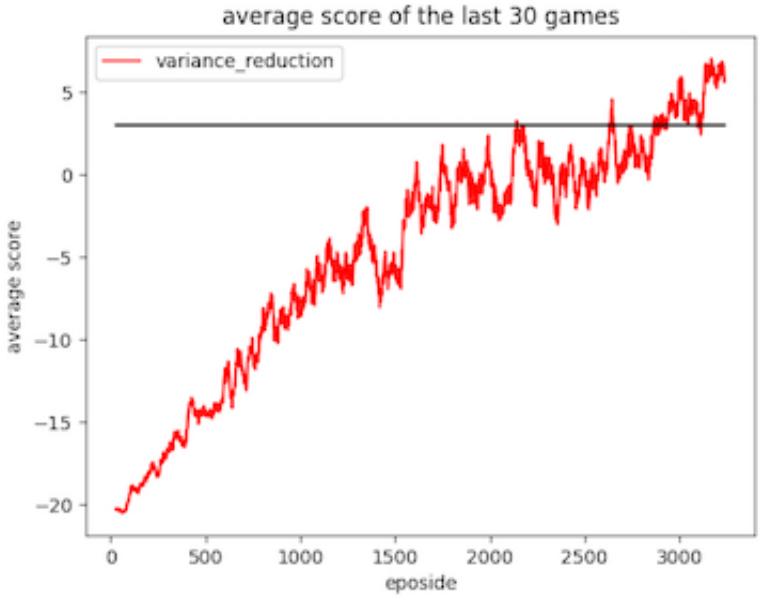
Policy Gradient model

- Network 架構



- 首先先用固定的model玩一個episode，得到每一個state(預處理過的圖片)中選取到動作的機率p和回饋r.
- 對r做0.99的discount後，再經過標準化
- 最後計算$gradient = log(p) \times r$
- 更新model後重複這幾個動作

Learning curve



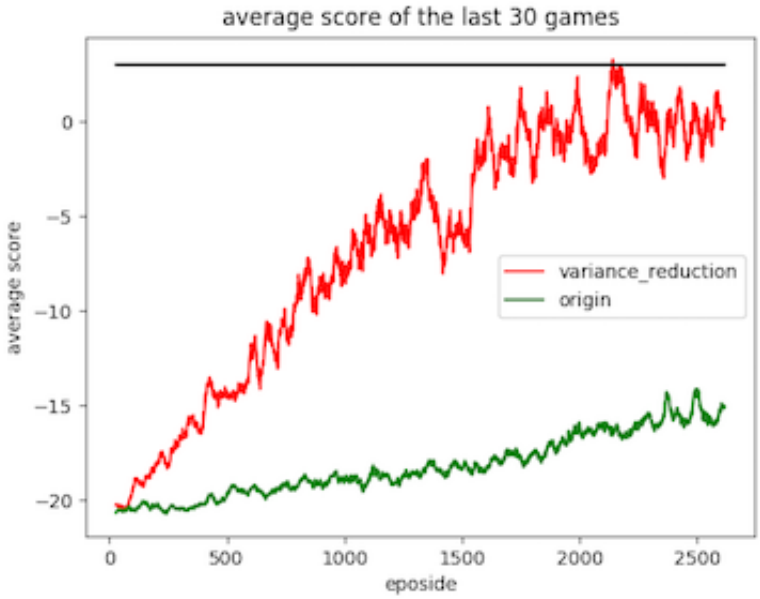
- 可以看出在3000個episode之後，就可以穩定的超過baseline

Improvement

Tips

- Variance reduction : 原始的policy gradient，分數只有1和-1，這樣的分佈對model來說，他的variance很大。
- Implementation : 先做0.99的reward discount，再減掉一個baseline，對於每個episode，我們的baseline是整個episode做完reward discount後的平均。

learning curve



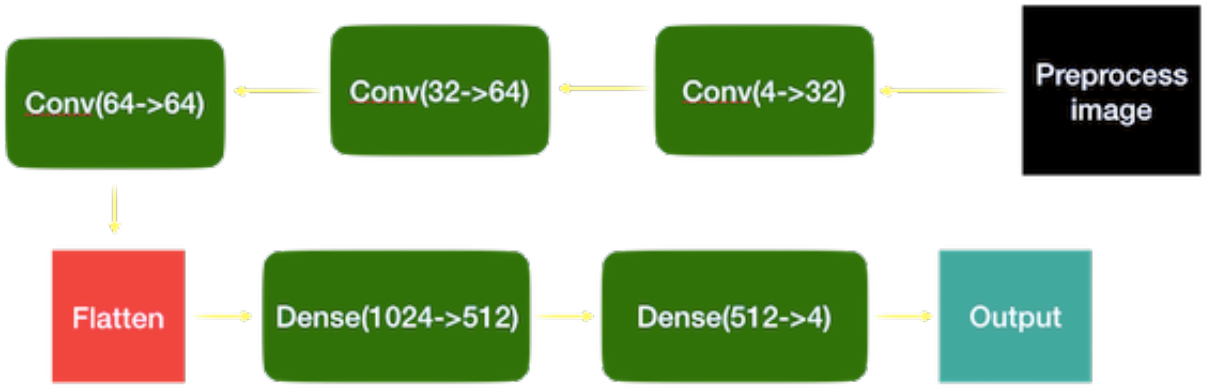
Compare

- 透過圖上可以看出在做了variance reduction後，在過了2000個episode之後就可以超過baseline了，但是原始的分數上升比較緩慢。

4-2

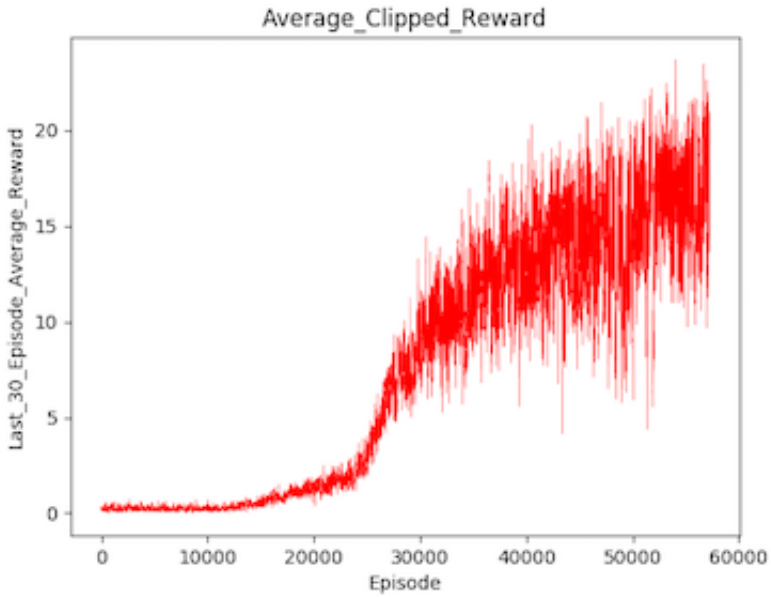
DQN model

- model架構：



- 先用原始的model玩一個episode，將我們要的state, reward和action存到reply buffer。
- 從reply buffer拿一個mini-batch，然後用原始的model去計算下一個state做什麼action之後能獲得最大的reward
- 預測現在這個state做什麼action能獲得最大的reward，然後計算gradient，更新一個新的model
- 等所有batch都拿完之後，再將新的model複製到舊的model

Learning curve



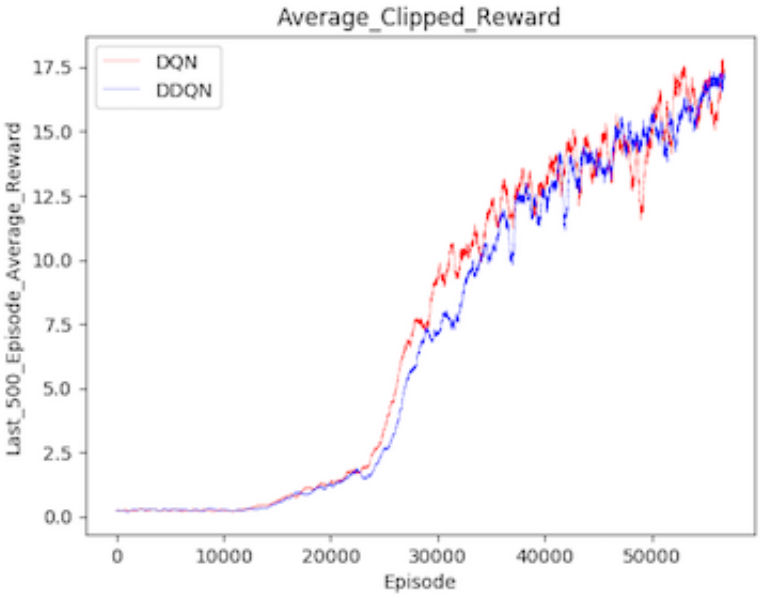
- 大概在30000個episode之後就可以超過baseline了

Improvement

Tips

- Double DQN : 原始DQN只使用舊的model來計算reward，DDQN在預測下一個state要使用哪個action才能獲得最大的reward的時候，用的是新的model，最後才用舊的model去計算reward。

learning curve



Compare

- 圖上可以看出兩個方法的表現差不多，只有一開始DQN的表現好一點點，之後就差不多了，可能是在做DQN的時候，model對於reward計算的overestimate並沒有非常的嚴重。

4-3

Actor-critic model

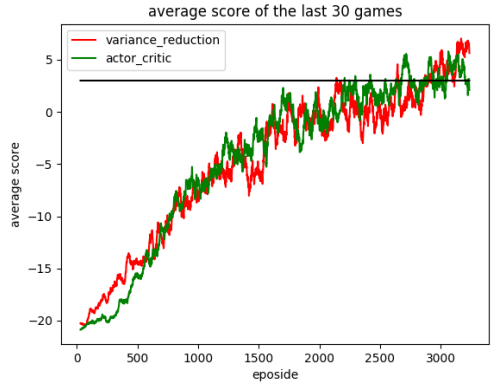
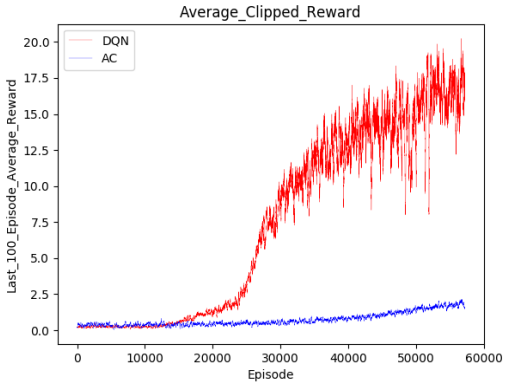
pong

- 跟原本不一樣的是，我們讓model多輸出一個值，代表這個state之後可以拿到的reward。一個state進來之後，先過一層dense後，拿輸出分別經過兩個dense，一個輸出action機率的分佈，一個輸出預測的reward，拿預測的reward跟真實的reward計算MSE loss之後，加上原始policy gradient的loss之後，再更新model。

breakout

- 我們的model在收到一個state後，會經過兩層conv，之後會有兩串dense，分別會輸出預測的reward和action的probability，拿預測的reward跟真實的reward計算MSE loss之後，加上原始policy gradient的log probability loss之後，再更新model。

Plot the curve & compare

Pong	Breakout
	

pong

- 從圖上可以看出，兩個方法的preformance其實是差不多的，不過從圖上無法看出，其實actor critic方法的每個episode的step是比原始的policy gradient少很多的，意思是actor critic能用較少的步數拿到分數，在訓練的時候也比較快。

breakout

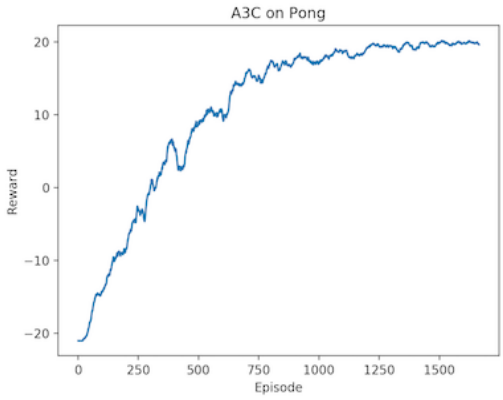
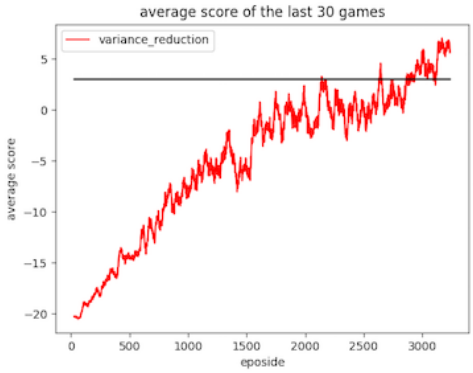
- 從圖上可以看出，原始的DQN筆actor critic好很多，推測可能的原因是因為在actor critic中，actor只輸出哪一個action的機率分佈而已，表現出哪一個動作最有可能得到正的reward，但是在breakout中（一個動作可能會拿到不只一個reward）卻不能最大化得到的reward，雖然critic做的事情也是在判斷得到的reward，但是就單純Value-base的DQN來說，透過找到能獲的最大reward的動作在圖上是好很多的。

Improvement method - A3C

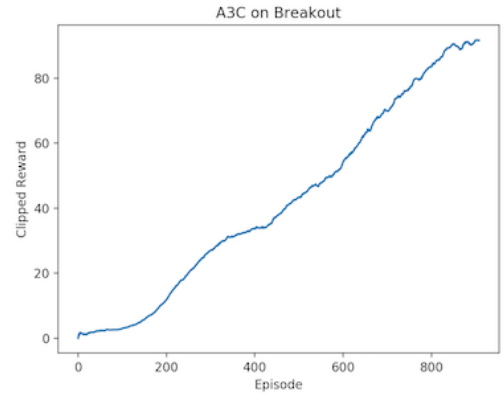
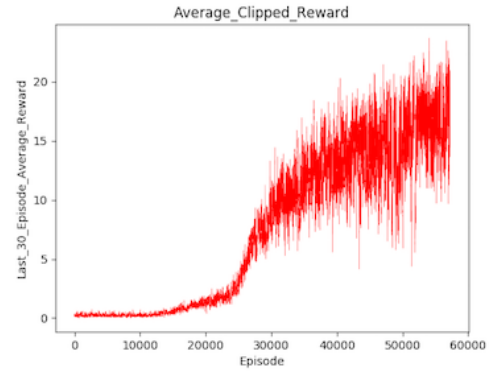
method

- 原本有share layer的actor critic是單線程的，在計算完gradient更新後才能繼續算下一個gradient，可是在A3C中，每一個thread會從model複製一份參數，再去拿一個mini-batch，計算完gradient後，直接更新主線程的model，就算在主線程的model已經不是原本複製的那一份，還是照樣更新，這樣在多個線程同時運行的時候，便不會有busy waiting的問題，在pong跟breakout的case中表現得也比較好。
- Refer : https://github.com/dgriff777/rl_a3c_pytorch

learning curve

pong-A3C(Thread number : 3)	pong-policy gradient
	

- compare：A3C大概在第300個episode的時候平均的成績就已經可以超過對手了，1000個episode後就可以穩定的超過對手20分。

breakout-A3C(Thread number : 3)	breakout-DQN
	

- Compare : 可以很明顯看出A3C只要幾百個episode就可以超越訓練了幾萬個episode的DQN。