Hw3

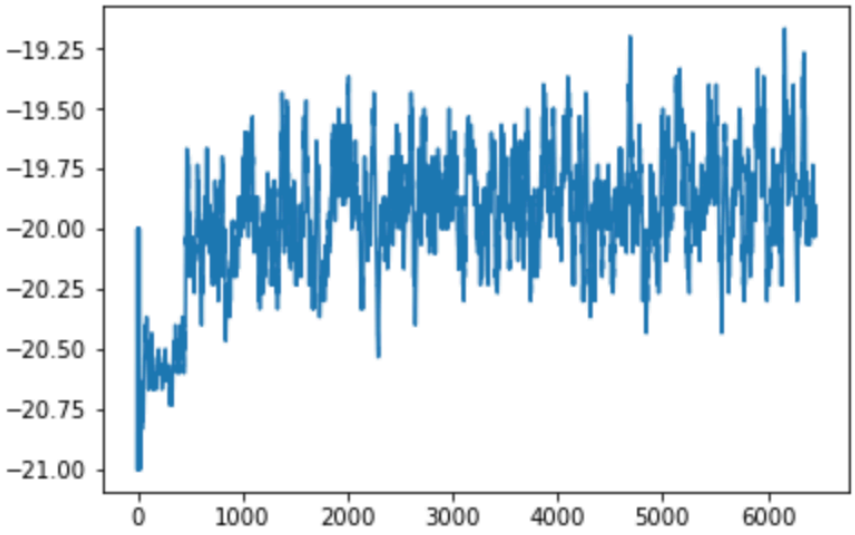
Policy Gradient

Model Description

以 keras 實作 Monte Carlo Policy Gradient，Observation 的前處理與 network 架構使用助教提供的範例，network 的 optimizer 是使用 rmsprop，參數也與提供的相同，並使用 category cross-entropy 做 loss function，在這邊不多做說明，而 network 的 input 為當前 observation 減去前一個 observation。 Training 時每個 episode 會儲存包含 model 的 input、model predict 的 probability distribution、選擇的 action 與獲得的 rewards，當 episode 結束時會將這些資訊拿來 training，首先 rewards 會隨著 steps 的增加而遞減，這邊乘以 gamma=0.99，並將 rewards 做 normalize，並乘以每個 action 的 probability 作為 training 的 label。

Learning Curve

下圖為單純 policy gradient 於 pong-v0 在 training 的 total rewards 表現（y 軸），x 軸為 episode 數，圖上 y 軸的值為每個前 30 個 episode 的 rewards 平均，可見成長速度較為緩慢，而且每次 training 的時間也較長（batch size 通常大於 1000），所以最後不是以這方法過 baseline，而是實作 a3c，導入 actor-critic 的方法做 training，在 A3C 的部分再詳細做說明。



Deep Q-learning

Model Description

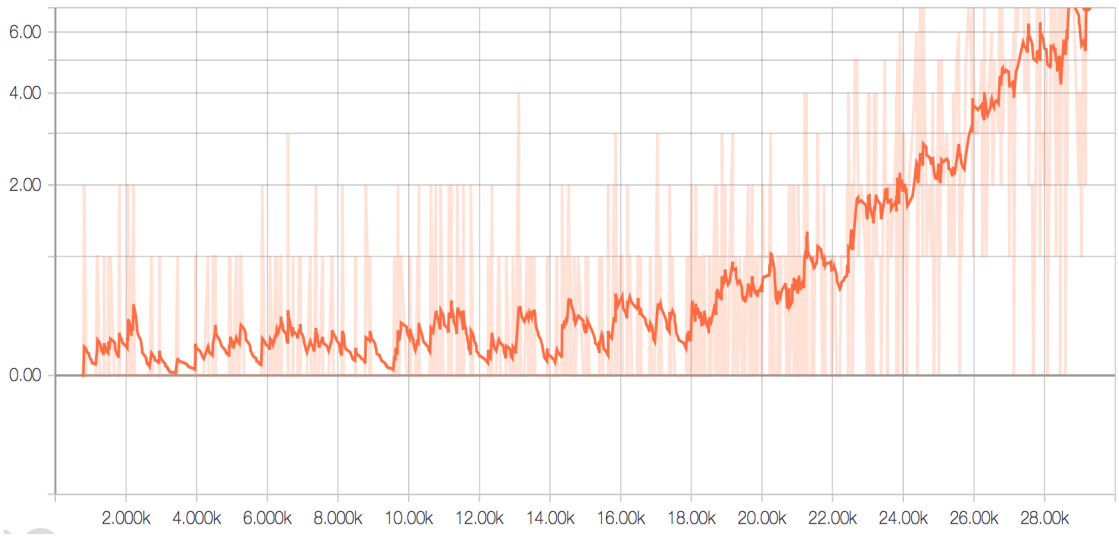
這邊 build 兩個神經網絡, target_network 用於預測 target_q_values 值，他不會及時更新 weights，而 q_network 用於預測 q_values，擁有最新的 weights，不過這兩個 network 結構是完全一樣的，下圖為 model 架構，使用 rmsprop optimizer，

```
model = Sequential()
model.add(Convolution2D(32, 8, 8, subsample=(4, 4), activation='relu', input_shape=(84,84,4)))
model.add(Convolution2D(64, 4, 4, subsample=(2, 2), activation='relu'))
model.add(Convolution2D(64, 3, 3, subsample=(1, 1), activation='relu'))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(self.num_actions))
```

target_network 是 q_network 的一個舊版本，擁有 q_network 很久之前的 weights，而且這組 weights 會被固定一段時間，然後再被 q_network 的新 weights 所替換，替換的頻率為每 4000 個 episodes 一次，而 q_network 是不斷在被提升的，所以是一個要被 training 的 network，每 4 個 steps 要 train 一次，但 steps 數要大於設定的 INITIAL_REPLAY_SIZE 為 10000，而每次 training 的 batch size 為 32，為 random 從 replay memory 取出，而 replay memory 為一 queue，以紀錄遊戲所有過程，包含前一個 observation、目前 observation、action、reward 與是否 terminal，而 memory size 為 80000。 決定 action 部分，當 random (0~1) 出來的數值小於 epilson 時都採用隨機的 action，而 epilson 一開始是設為 1，會隨著每個 step 遞減（-(1-0.05)/1000000）至 0.05。

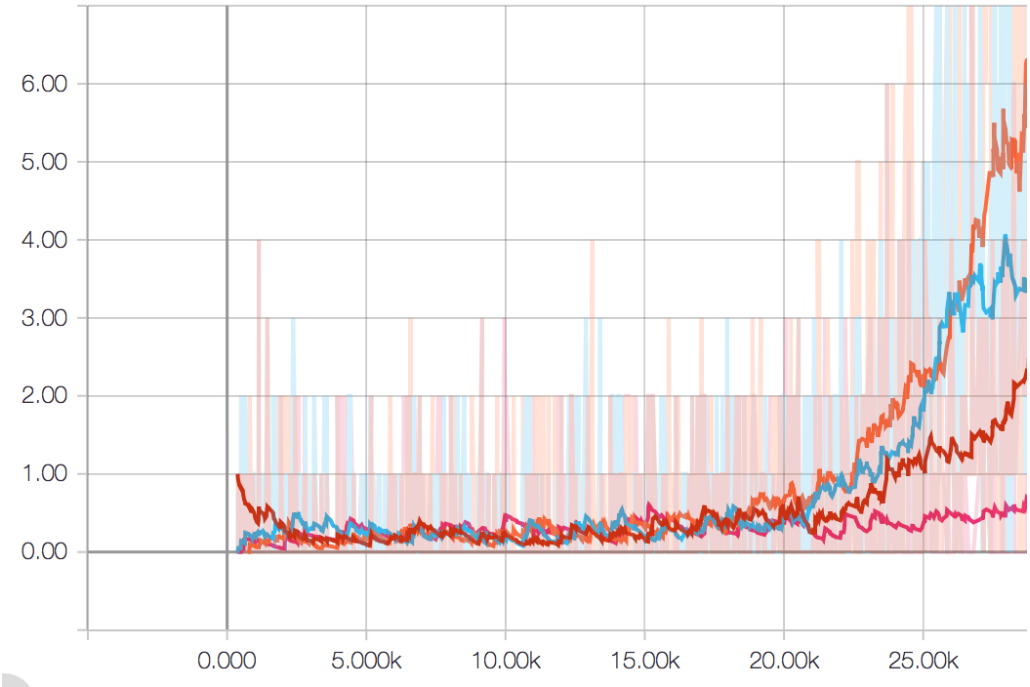
Learning Curve

下圖為 dqn 於 breakoutnoframeskip-v4 在 training 的 total rewards（每個 step reward 做 clip -1~1 後）表現，x 軸為 episode 數，y 軸的值已經過 Smoothing（0.9）。



Experiment with DQN hyperparaments

下圖為不同 target network update frequency（4000：橘色、2000：藍色、1000：紅色、8000：粉紅色）於 DQN training 時的 rewards 表現，x 軸為 episode 數，y 軸的值已經過 Smoothing（0.95）。

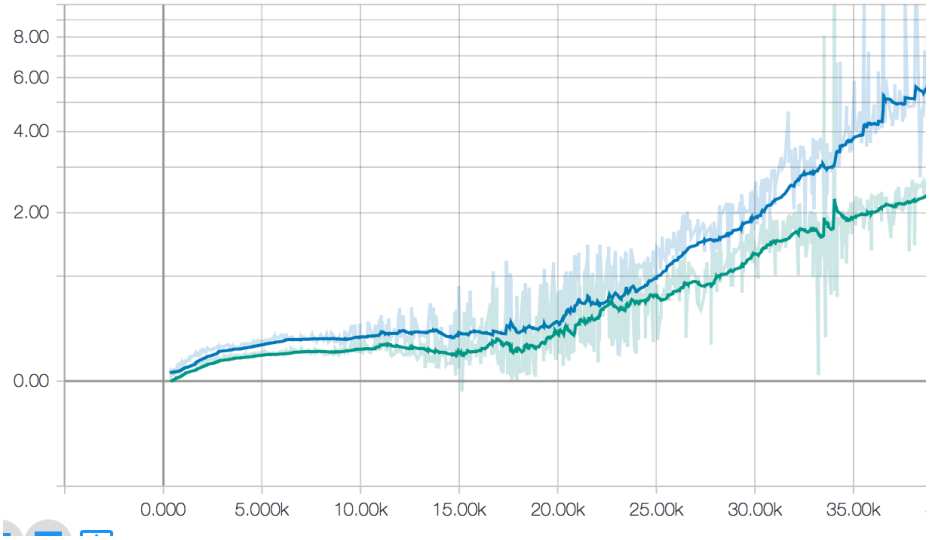


根據 target_network weight 的 update frequency 會影響 training 表現的穩定度與進步速度，當 update 太過頻繁會使 rewards 震盪太過激烈且 network 難以收斂導致進步緩慢，由圖可知，當 frequency 為 1000 時在 20000 個 episodes 後與 frequency 為 2000 和 4000 相差甚多，若太過久沒更新的話又會造成 target_network 的 weight 與現實太過脫節導致較難進步，所以選擇 update frequency 也是個影響表現的重要因素，現在上傳的 model 是選擇 4000 作為 frequency，在圖中是 rewards 進步速度最快的參數。

Improvement to DQN

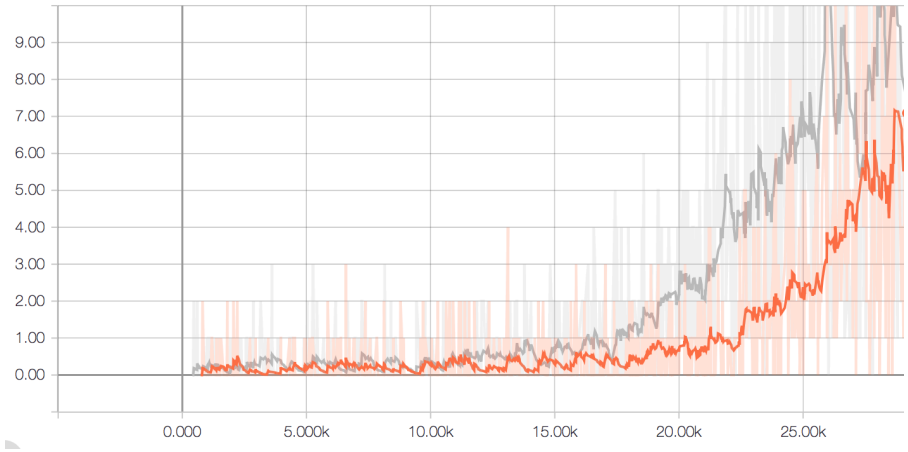
Double DQN

因為我們的 network 預測的 max q_values 本來就有誤差，每次也用最大誤差的 q_values 改進神經網絡，就是因為這個 max q_values 導致了 overestimate。所以 Double DQN 的想法就是引入另一個 network 來減少 overestimate 的影響。而 DQN 中本來就有兩個神經網絡，所以我們用 target_network 來估計 q_values 中 max q_values 的最大 action value，然後用這個被 target_network 估計出來的動作來選擇 q_values。 下圖為 training 時的 max q_value 趨勢圖，藍色的線為 DQN，而綠色的是 Double DQN，x 軸為 episode 數，y 軸的值已經過 Smoothing（0.9），可見 Double DQN 的 Max q_value 皆比 DQN 的還低並且更穩定。



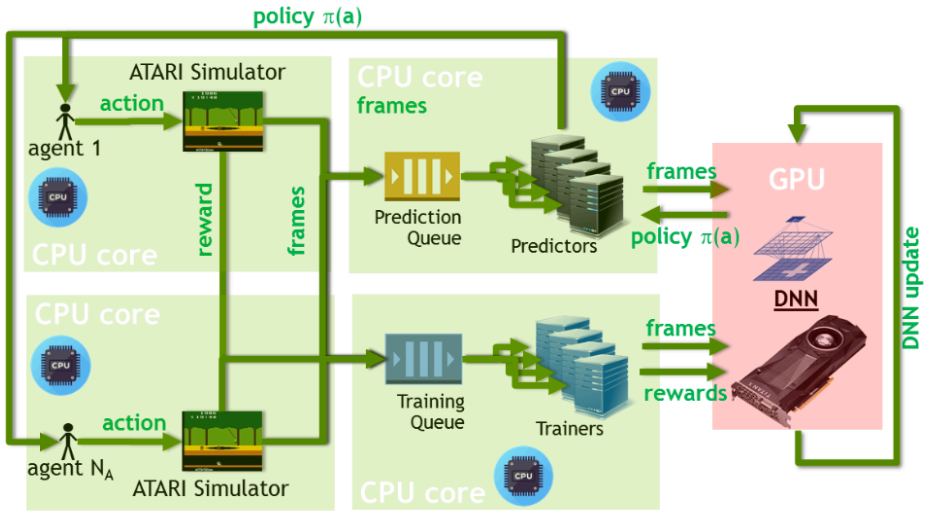
Dueling DQN

只要稍稍修改 DQN 中 network 結構，就能大幅提升學習效果，加速收斂，這種新方法叫做 Dueling DQN。 將每個動作的 Q 拆分成了 observation 的 value 加上每個動作的 advantage，原來 DQN 神經網絡直接輸出的是每種 action 的 q_value，而 Dueling DQN 每個 action 的 q_value，它分成了這個 observation 的值，加上每個 action 在這個 observation 上的 advantage，因為有時候在某種 observation，無論做什麼動作，對下一個 observation 都沒有多大影響。 下圖為 training 時的 total rewards（每個 step reward 做 clip -1~1 後）趨勢圖，灰色的線為 Dueling DQN，而橘色的是 DQN，x 軸為 episode 數，y 軸的值已經過 Smoothing（0.95），可見 Dueling DQN 的 reward 明顯的進步快速，model 較快收斂。

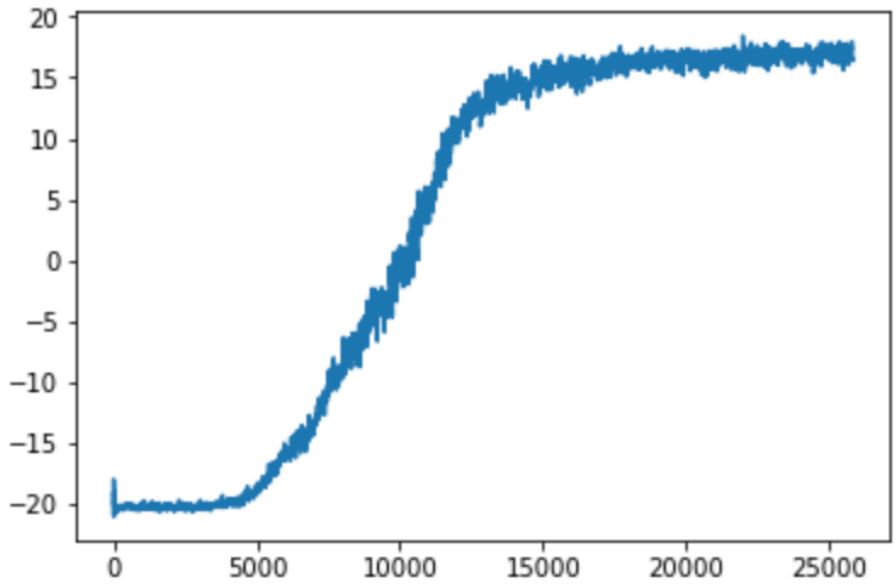


A3C

Google DeepMind 提出的一種解決 Actor-Critic 不收斂問題的算法，它會創建多個並行的環境，讓多個擁有副結構的 agent 同時在這些並行環境上更新主結構中的參數，並行中的 agent 們互相不干擾，而主結構的參數更新會受到副結構提交更新的不連續性干擾，所以更新的相關性被降低，使收斂性提高。 但這邊實作的 A3C 有點不一樣，參考 Nvidia 提出的 GA3C，其混合使用 CPU 與 GPU，也解決了原 A3C 訓練時需要給每個並行的 agent 複製一份子 network 來收集樣本計算累計 gradient 使並行的 agent 數量很多時相當佔 memory。 如下圖所示，GA3C 只需要一個 network，並可用 GPU 做 training。 Agent 部分和 A3C 的功能一樣會收集樣本，但不需要在每個 agent 都複製一份 model，在每一次要選擇 action 前將目前的 observation 加入 prediction queue，而進行遊戲 n 個 step 後再算 total rewards 並得到 n 組（s,a,R,s'）加入 training queue。 Predictor 則將 prediction queue 的樣本做 mini-batch 餵進 model 做 predict actions，再將 actions 還給各自的 Agent。 Trainer 將 training queue 中的樣本也做 mini-batch 餵進 model 做 training。



下圖為 GA3C 於 pong-v0 上 training 的 total rewards 趨勢，x 軸為 episode 數，圖上 y 軸的值為每個前 30 個 episode 的 rewards 平均，相較於一般的 policy gradient 其 rewards 的進步有明顯的差異，而 training 的時間也大幅降低。


20
15
10
5
0
−5
−10
−15
−20
0
5000
10000
15000
20000
25000