# ADLxMLDS 2017 Fall
# HW3 - Game Playing

B05901189 吳祥叡

December 16, 2017

## 1 Basic Performance

### 1.1 Model Description

1. Policy Gradient :
   限制只使用 action 1, 2, 3 (NoOp, Up, Down)
   input 使用兩個 frame 相減
   input_image: (80, 80, 1)
   Conv2d(16, size=8, stride=4)
   Conv2d(32, size=4, stride=2)
   Flatten([32*(10**2)])
   Dense(128, activation=relu)
   Dense(action_num, activation=sigmoid)
   optimizer = RMSProp(lr=1e-4, decay=0.99)

2. DQN :
   input_images: (84, 84, 1)
   input_actions (4)

   Conv2d(32, size=8, stride=4)
   Conv2d(64, size=4, stride=2)
   Conv2d(64, size=3, stride=1)
   Flatten([64*(11**2)])
   Concat(input_actions)
   Dense(512, activation=lrelu(alpha=0.01))
   Dense(1)
   optimizer = RMSProp(lr=1e-4, decay=0.99)
   Gamma = 0.99
   main_network update_freq = 4 steps
   target_network update_freq = 1000 steps
   explore_rate: 1 to 0.05 in first 1M steps

### 1.2 Learning Curves

PG 共訓練約 4000 episodes，取 30 episode 的移動平均作圖。
DQN 共訓練約 3M episodes，取 100 episode 的移動平均作圖。

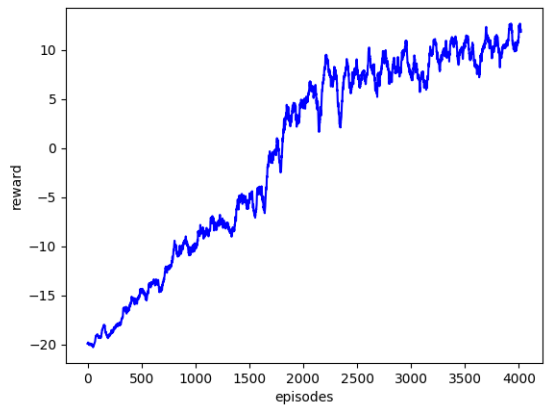


Figure 1: Learning Curve of PG

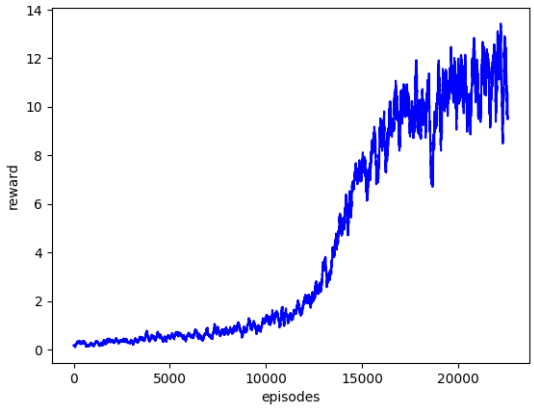


Figure 2: Learning Curve of DQN

## 2 Experimenting with DQN hyperparameters

因為一開始都很習慣的直接用 relu，直到和助教一樣用了很特殊的 leaky relu 作為最後一層 hidden layer 的 activation function 才有明顯的進步。所以決定試驗另五種 activation function。分別實驗了 relu，parametric relu，lrelu($\alpha$=0.001)，lrelu($\alpha$=0.01)，以及最近蠻常聽到宣稱在比較深網路裡表現很好的 swish。由上圖看出 lrelu 在使用

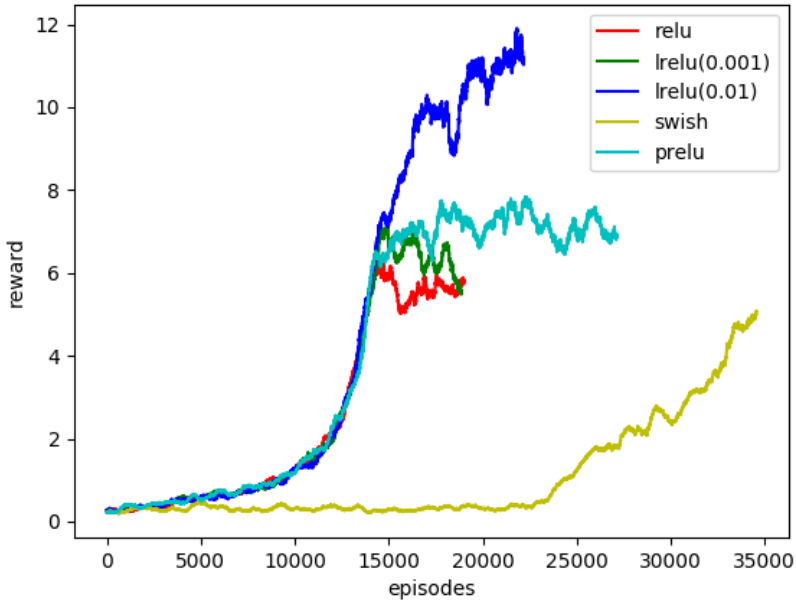


Figure 3: 比較不同 activation function

比較大的 $\alpha$ 遠勝其他 activation，而 swish 也沒有傳言中那麼好用，儘管函數長得很像 relu，但在這個 case 甚至完全壞掉。推測 swish 可能比較適合用來取代 convolution layer 中的 activation。

# 3 Bonus

## 3.1 Improvements on DQN

實做了 Double DQN 和 Prioritized Replay Memory。以下簡略敘述做法和比較方法:

### 3.1.1 Double DQN

為了解決 Q 值的 overestimate 問題，Double DQN 將原始的 main_network update target $r_{i+1} + maxQ'(s_i, a)$ 改為: $r_{i+1} + Q(s_i, argmax(Q'(s_i, a))$
比較時都使用同樣的 network 和同樣的 hyperparameter。實做 double dqn 發現在後段

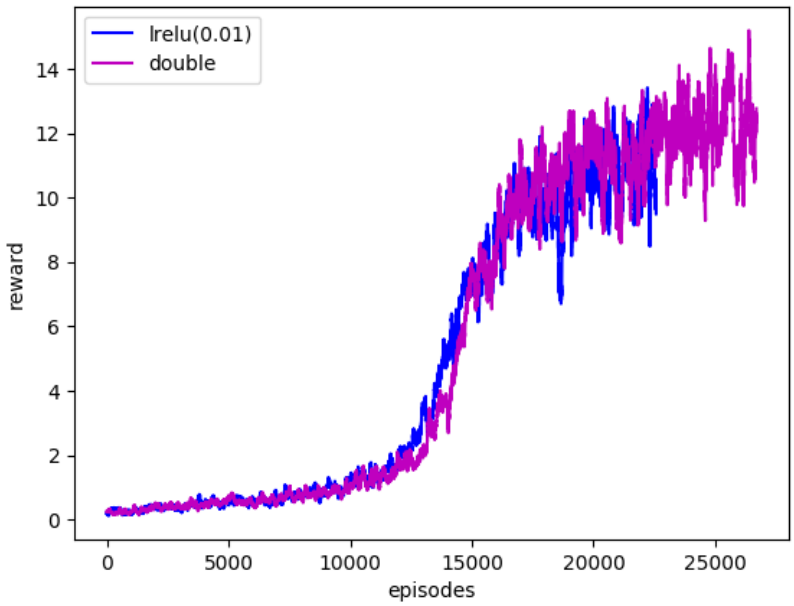


Figure 4: 比較原始 dqn 及 double dqn

learning curve 有比較原始 dqn 上方確實收斂比較好。實驗時發現同樣 12 分的 clipped reward 在實際用 test.py 跑出來的分數其實可以差很多，前段在約 20000 episode 的時候實際大約只有 30 分，而到後段可以到 60 分。

### 3.1.2 Proportional Prioritized Replay Memory

利用 Sum Tree 資料結構實做，達到 O(log(N)) 的 insert 以及 sample 複雜度。並且使用 Importance-Sampling Weight 平衡被 priority 影響的取樣分佈。
其中 priority 計算方式為: $TD = abs(Q_{pred} - Q_{target}) = abs(Q_{pred} - \gamma(r_i + maxQ_{i+1}))$
$p_i = min(1, TD_i)^{\alpha}$
IS Weight 計算方式為: $(p_i/min(p)))^{-\beta}$
實驗時使用兩組參數，分別為
黑色: $\alpha$=0.6，$\beta$=0.4
紅色: $\alpha$=0.7，$\beta$=0.5
黃色: $\alpha$=0.3，$\beta$=0.3。

對照藍色曲線為 random sample 的原始方法，前兩種參數是參考原始 paper 裡面經由 grid search 所得到較好了參數。但原始 paper 為了要有通用性是比較了全部 atari 遊戲的結果。此處實驗顯示使用比較多的 priority 因素反而會使收斂速度和整體表現下降。因此我多實驗了黃色的一參數，看起來比 paper 上的參數好一些，但沒有超越原始的方法。

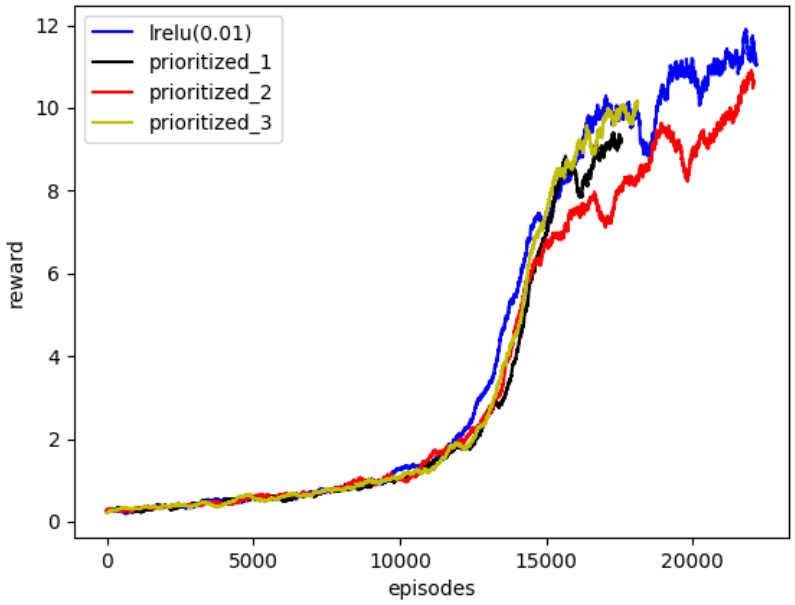


Figure 5: 比較原始有無 prioritized 的 memory