Basic Performance (6%)

- Describe your Policy Gradient & DQN model (1% + 1%)
    - Policy Gradient
      # 參數
      learning_rate = 0.0005
      reward_decay = 0.99

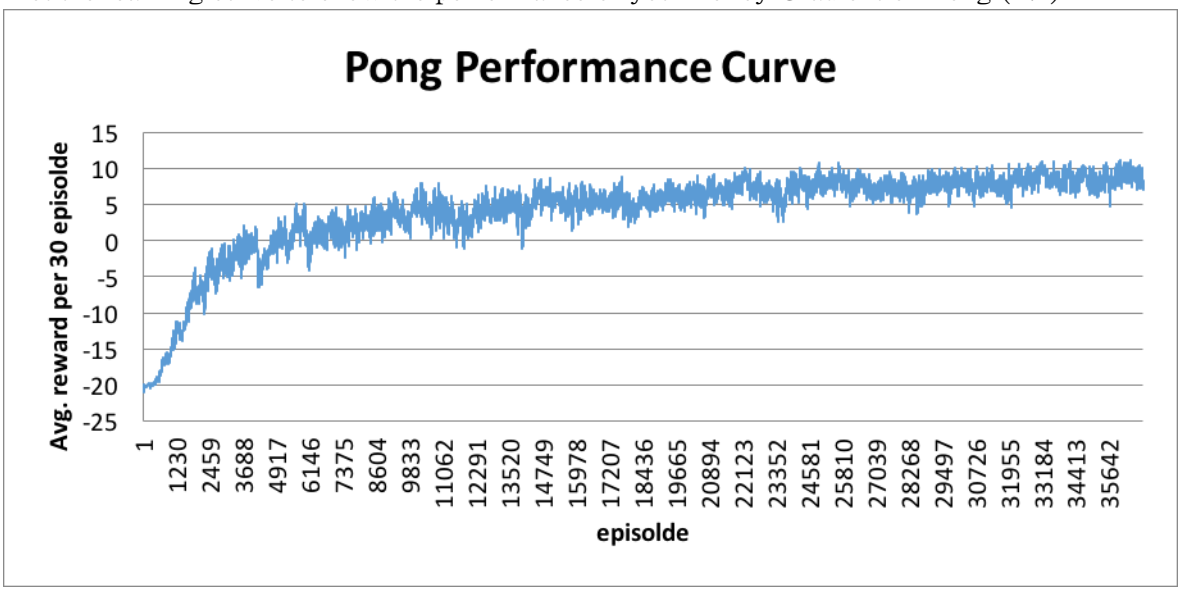
      # 前處理部份先將圖片轉為 5120 維的 vector
      prepro(crop[35:195,16:-16], down_sample=2) // output = 5120

      # 模型為一層 Dense
      Input(shape=5120)
      Dense(200, activation=relu)
      Dense(2, activation=softmax)
      Adam(lr=0.0005)

    - DQN
      # 參數
      learn_start = 10000 // 10000 step 後開始訓練
      learn_freq = 4 // 每 4 個 step 更新一次模型
      max_step = 5e6 // 最大 step
      min_explore_rate = 0.05 // explore_rate 在前 1M 個 step 由 1 遞減到 0.05
      gamma = 0.99 // reward decay
      batch_size = 32
      memory_size = 10000
      optimizer = Adam (lr=0.0001)

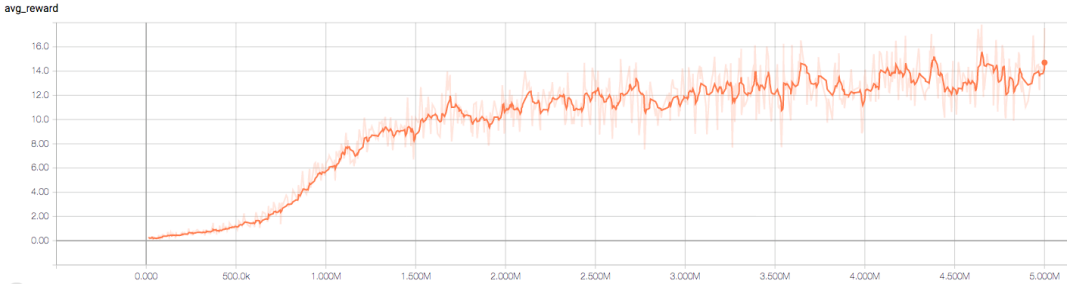
      # 模型如下
      Input(shape=(84,84,4))
      Conv2D(kernel=8, stride=4, filters=32, activation=relu)
      Conv2D(kernel=4, stride=2, filters=64, activation=relu)
      Conv2D(kernel=3, stride=1, filters=64, activation=relu)
      Flatten()
      Dense(512, activation=lrelu(alpha=0.01))
      Dense(4)
      Adam(lr=0.0001)

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



大約訓練 500 局之後能看到 reward 有明顯提升
大約訓練 5000 局之後能夠有一半的機會贏過電腦
大約在訓練了 30000 局之後能穩定超過 baseline(7 分)

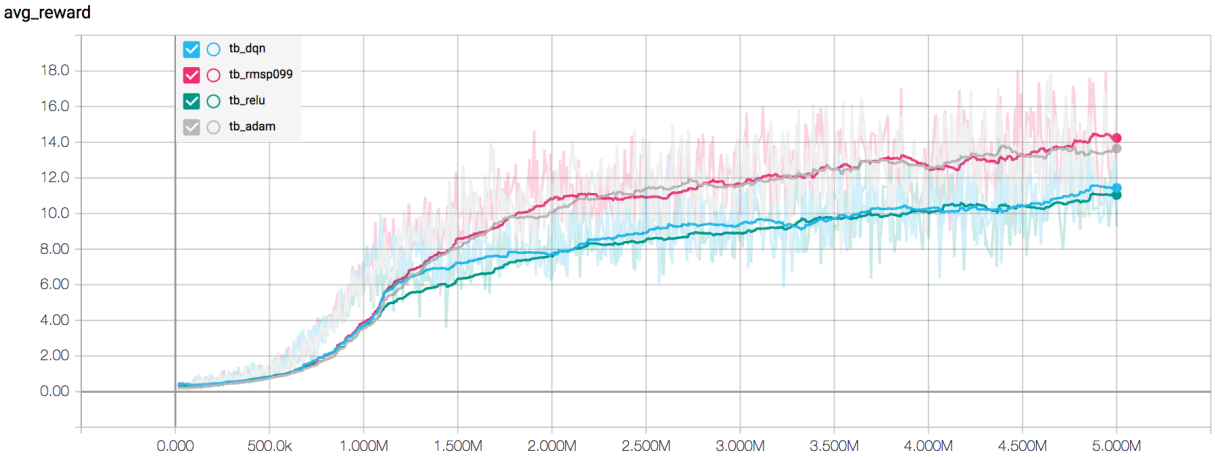
- Plot the learning curve to show the performance of your DQN on Breakout (2%)



X 座標為 step, Y 座標為 clip 過的平均 reward, 並且利用 tensorboard 做 smooth
大約訓練 700k 步之後能看到 reward 有明顯提升
大約訓練 1M 步之後(explore rate 降到 0.05)每場遊戲平均能有 6 分
大約在訓練了 2M 步之後每場遊戲平均能有 10 分，這時候的 model 去做 test 有機會能過 baseline(50 分)
訓練完 5M 步的 model 去做 test 可以拿到 70 分

Experimenting with DQN hyperparameters (4%)

- Plot all four learning curves in the same graph (2%)
  實驗以第一題 DQN model 架夠但 Optimizer 為 RMSProp(lr=0.0001, decay=0.9)為基準，分別做了以下三種調整:
  1. 將 RMSProp 的 decay 由 0.9 改為 0.99
  2. 將 RMSProp 改為 Adam，learning_rate 不變
  3. 將 Dense 層的 leaky relu 改為一般的 relu



X 座標為 step, Y 座標為 clip 過的平均 reward, 並且利用 tensorboard 做 smooth
tb_dqn 為實驗模型基準
tb_rmsp099 為實驗一
tb_adam 為實驗二
tb_relu 為實驗三

- Explain why you choose this hyperparameter and how it effect the results (2%)
  1. 將 RMSProp 的 decay 由 0.9 改為 0.99 的原因為 Tensorflow 和 Pytorch 的預設 decay 值不同，想知道這個值是否對訓練過程有很大的影響。
     實驗結果為 decay=0.99 會有較好的 performance，推測若 decay 過快，會影響到學習的效率。
  2. 將 RMSProp 改為 Adam 的原因為想比較不同 Optimizer 對訓練的影響。
     實驗結果為 Adam 會有較好的 performance，與 RMSProp(decay=0.99)差不多，推測原因同上。
  3. leaky relu 一般是為了避免 dead neural 過多的問題，實驗修改為 relu 是否會影響 performance。
     實驗結果為 performance 沒有明顯的改變，推測本來就沒有太嚴重的 dead neural 問題。

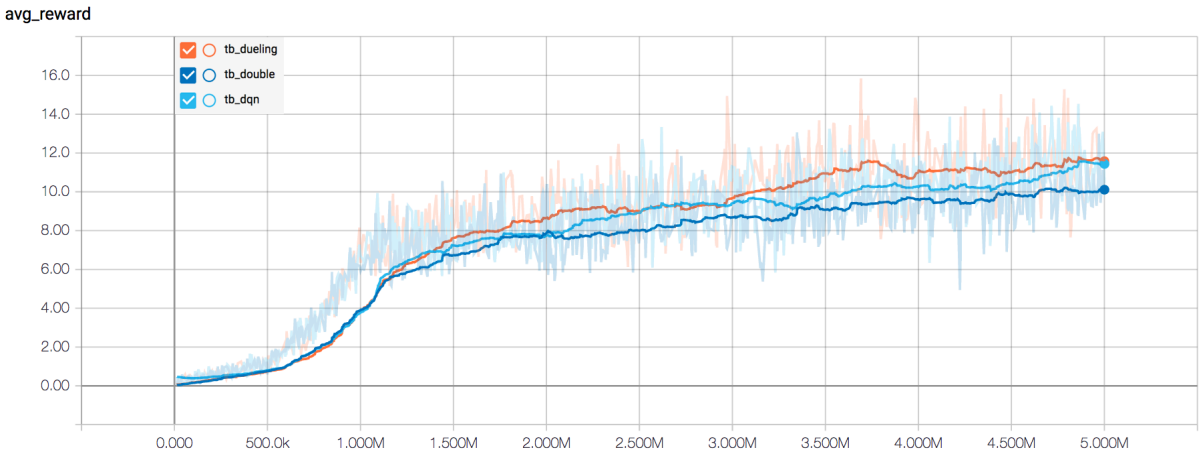
Improvements to DQN (2%)

- Implement at least two improvements to DQN (p.9) and describe why they can improve the performance (1%)
  這次實作了兩種 DQN 的 improvements，分別為 DoubleDQN 和 DuelingDQN。

  DoubleDQN 在計算 target_q 值的時候由原本的 Max(all_target_q)改為選擇 all_target_q[argmax(all_eval_q)]，這樣的作法能夠減少 overestimate 的問題。

  DuelingDQN 則是將網路分為 Value(狀態好壞)的部分和 Advantage(動作好壞)兩個部分，而 Q 值的計算改為 Value + Advantage，分開計算的原因是因為在某些 State 下不論做什麼 Action 對狀態轉變的影響不大，這個時候計算動作的價值並沒有太大的意義。

- Plot a graph to compare and analyze the results with and without the improvements (1%)



X 座標為 step, Y 座標為 clip 過的平均 reward, 並且利用 tensorboard 做 smooth
tb_dqn 為一般的 DQN(同上一題架構)
tb_double 為 DoubleDQN
tb_dueling 為 DuelingDQN

由訓練 5M 步的結果可以看到 dueling > dqn > double，但是差距並沒有十分的明顯，我認為實驗結果沒辦法區分這幾個模型在這個遊戲上的明顯優劣，而且 dqn > double 的結果有點奇怪，或許只是因為初始權重得不同而導致些微的差異，應該做更多的實驗來比較。