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)

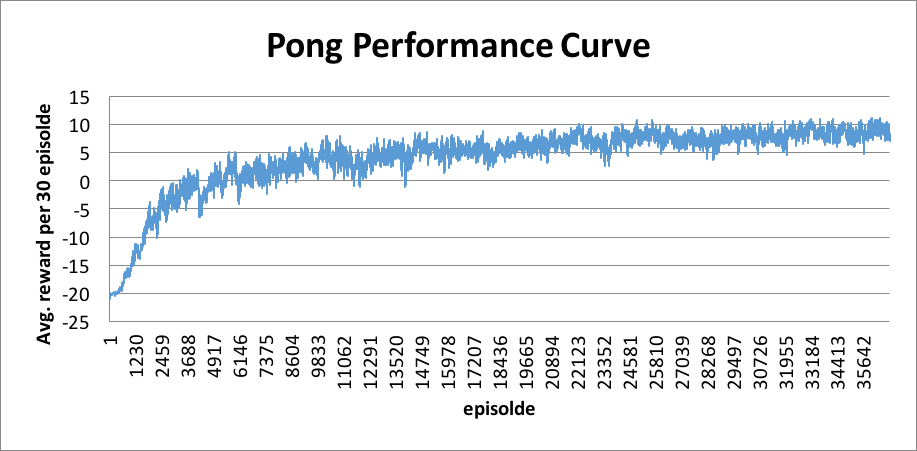
Flatten()

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%)

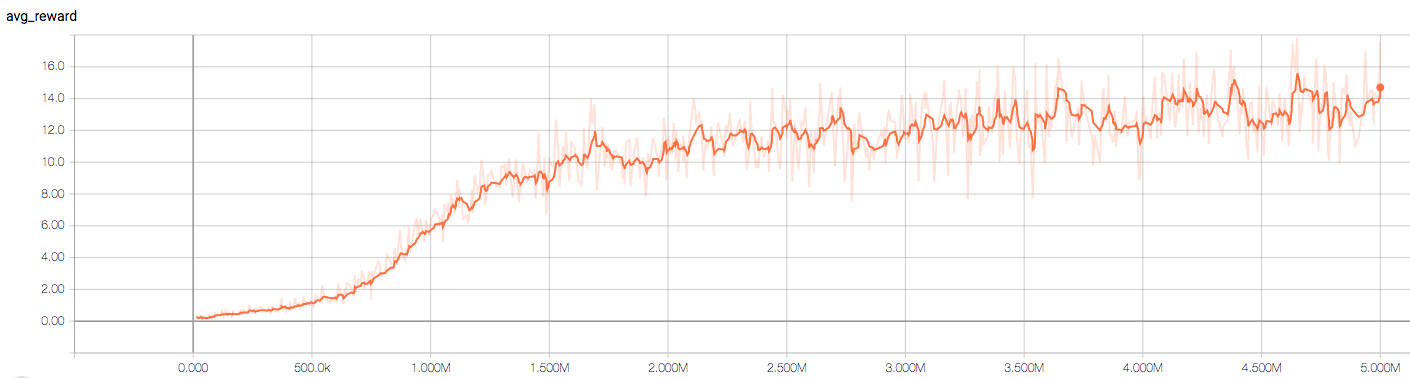


大約訓練500局之後能看到reward有明顯提升

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)

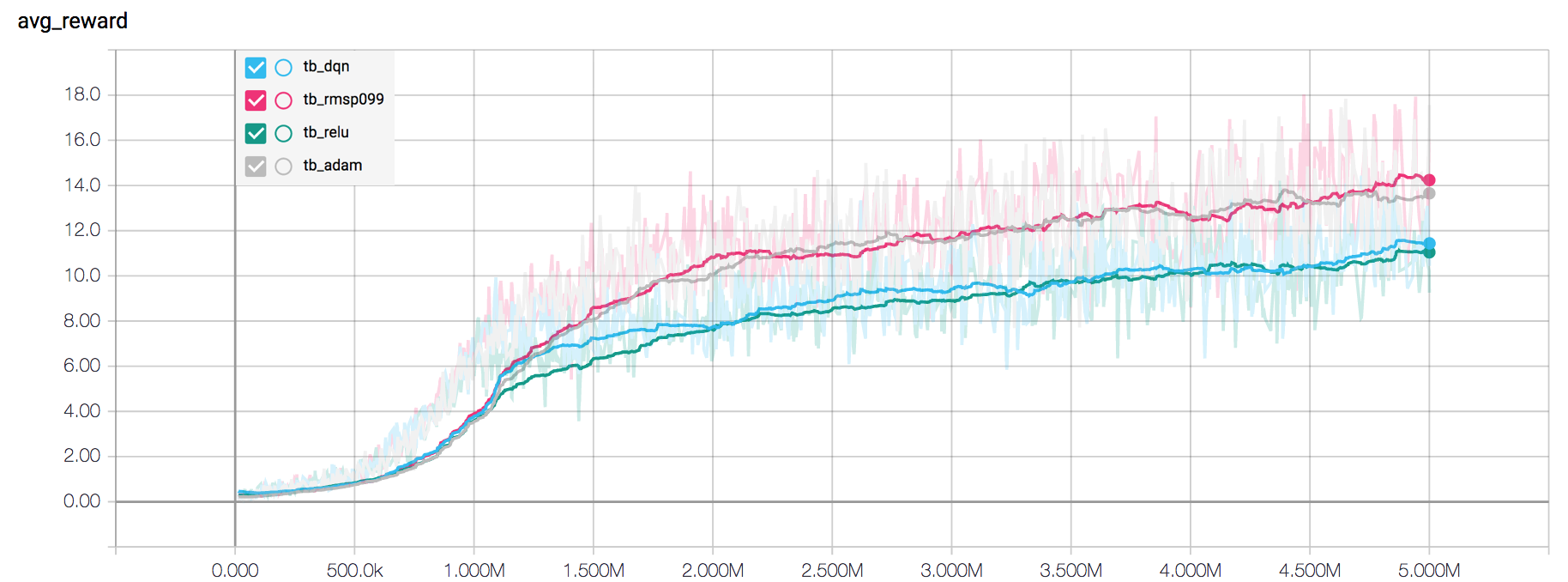
訓練完5M步的model去做test可以拿到70分

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。

1. 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。

1. 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%)

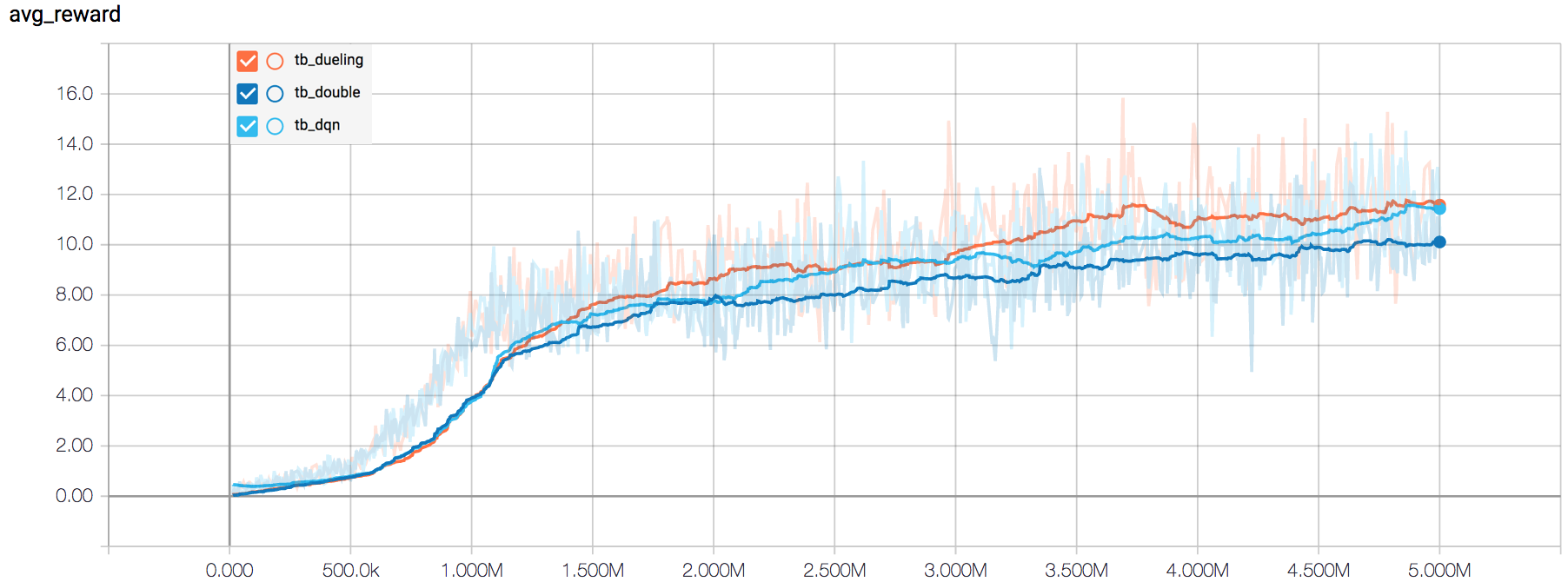
* Implement at least two improvements to DQN (p.9) and describe why they can improve the performance (1%)

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。