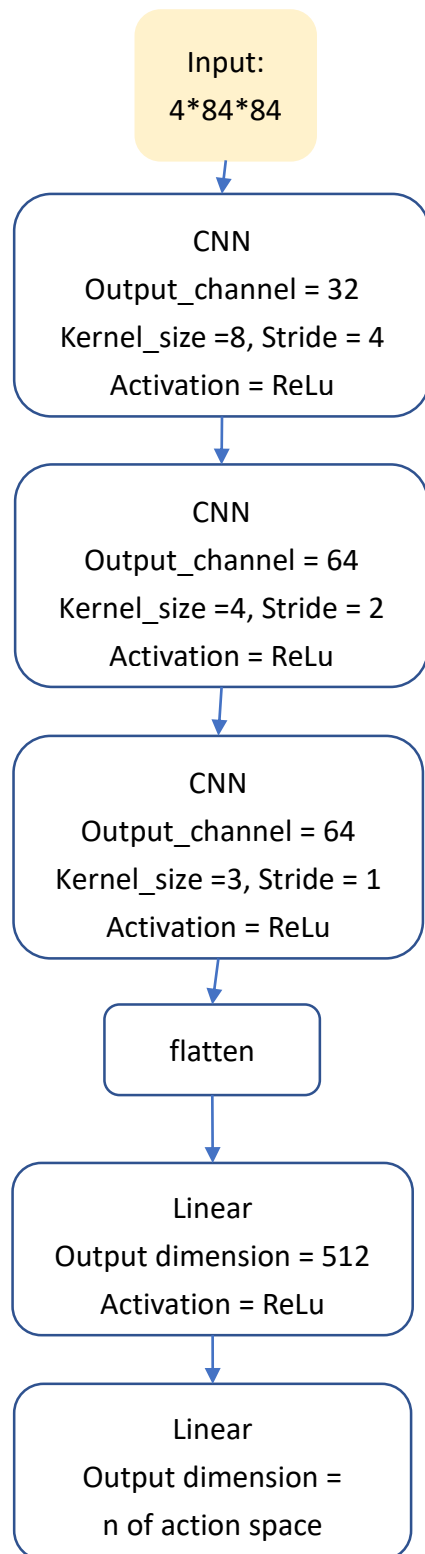


(1) DQN model



左圖為用來計算(state, action)的 value 的模型，input 為 environment 輸出的圖片(state), output 為在這個 state 的條件下各個 action 的 value 值。

共訓練 3,000,000 steps

Optimizer = RMSProp

Learning rate = 0.0001

GAMMA = 0.99

relay memory 大小 = 10000

target model update frequency =

1 次 / 1000 steps

online model update frequency =

1次 / 4 steps

Learning start: 10000 steps 之後

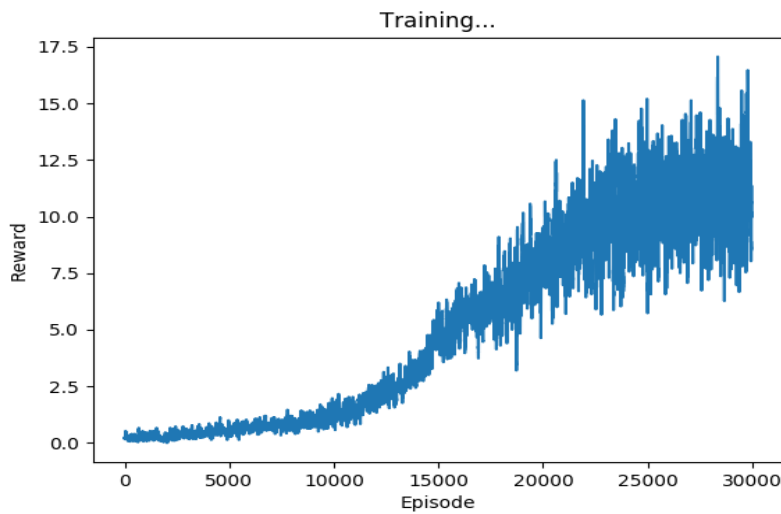
Random action 的機率:

在 1,000,000 steps 內從 0.99 遞減到

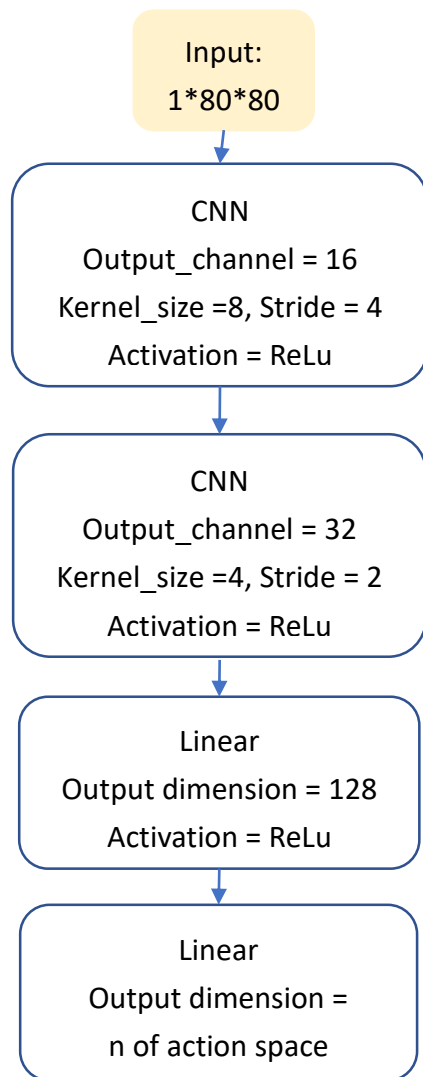
0.05, 以已執行過的 steps 數 /

1000000 的比例線性遞減

訓練表現:



(2) Policy Gradient



左圖為 Policy network 的架構

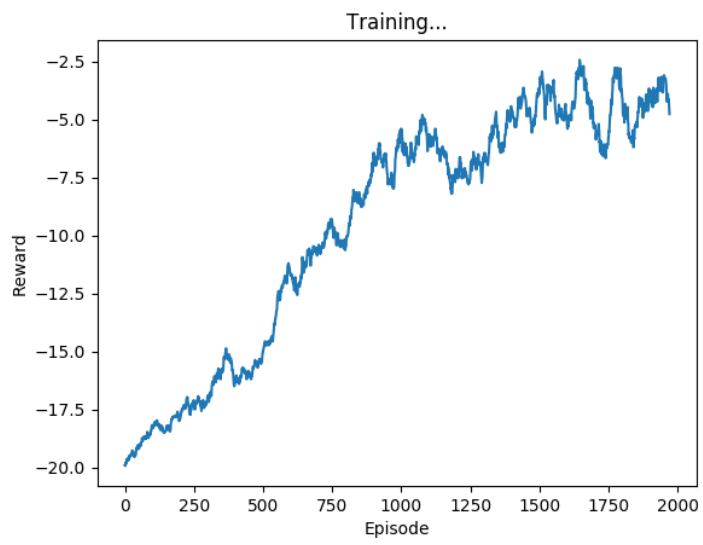
Input 是前處理後的灰階 state,
output 是在這個 state 的條件之下各個 action 的機率值

Optimizer : RMSProp

Learning rate = 0.0001

GAMMA = 0.99

訓練表現:



(2) 實驗:

選擇 **exploration schedule** 做為實驗調整的參數, 試著調整隨機選擇 **action** 的機率, 分別試了從 90%遞減到 5%、99%遞減到 5%、99%遞減到 1%、99%遞減到 10%的設定。之所以選擇這個設定是因為一開始這個 **exploration rate** 設得較低時, 發現訓練不太起來, 推測是因為一開始的模型由於尚未經過訓練, 所以給的預測通常不太有用, 且一開始由於很容易就死掉了, 能觀察到的 **state** 也有限, 若使用模型的預測, 感覺得到的 **(state, action) pair** 可能變化太少了, 無法讓 **agent** 對這個環境有更多的認識, 因此一開始的 **exploration rate** 應該要設定高一點, 藉由 **random** 決定動作再觀察 **reward**, 讓 **agent** 更容易學到怎樣的 **action** 的 **reward** 應該要比較高。另外, 關於 **exploration rate** 最後應該固定到多少較合適, 從圖中可以看到從 99%降到 1%的設定(淺橘色線)的表現最好, 因為到了後期用 **random** 選動作比起用 **model** 預測動作容易死掉, 所以 **reward** 比較難上升, 我覺得可能因為這個遊戲並不是太複雜, 所以不需要花太多力氣的 **exploration** 上面, 所以最後固定的 **exploration rate** 不需要太高, 讓 **model** 盡量在已知的情況下得到最高的 **reward** 可能比較好。

