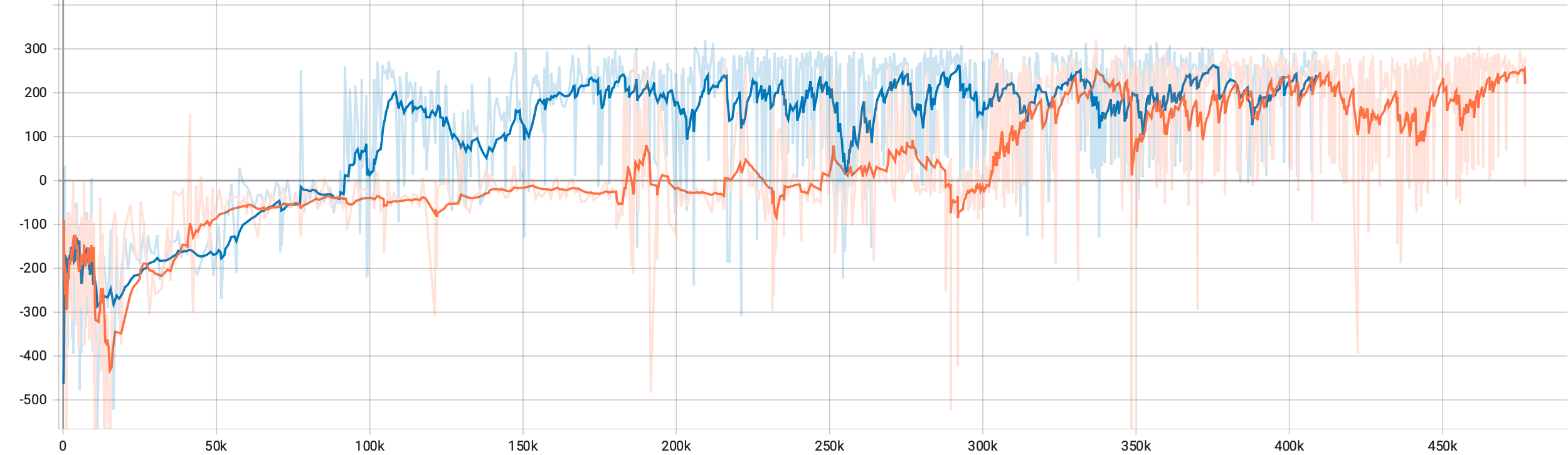
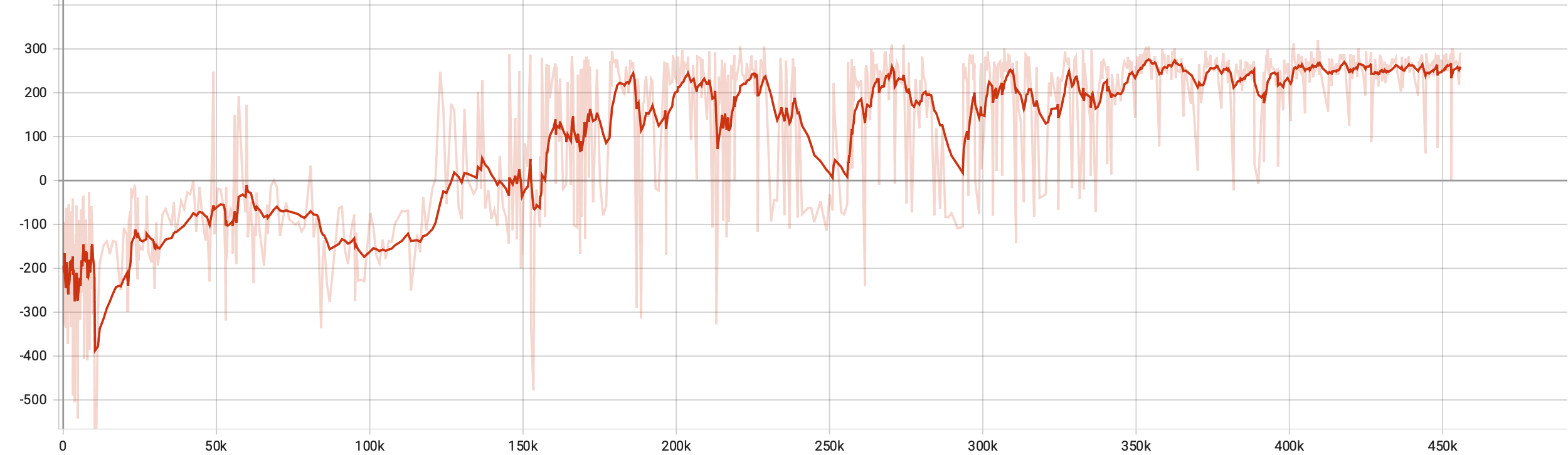
* Report:
* A tensorboard plot show episode rewards of at least 800 training episodes in LunarLander-v2:



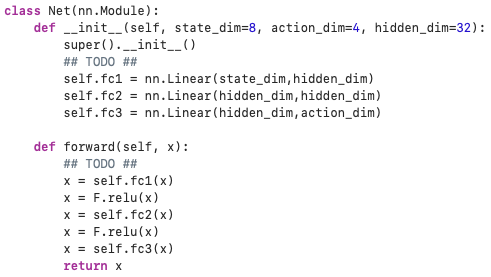
* A tensorboard plot show episode rewards of at least 800 training episodes in

LunarLanderContinuous-v2:

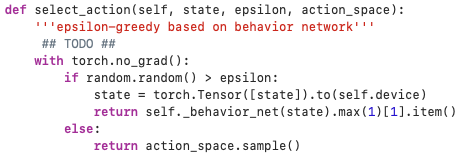


* Describe your major implementation of both algorithms in detail:

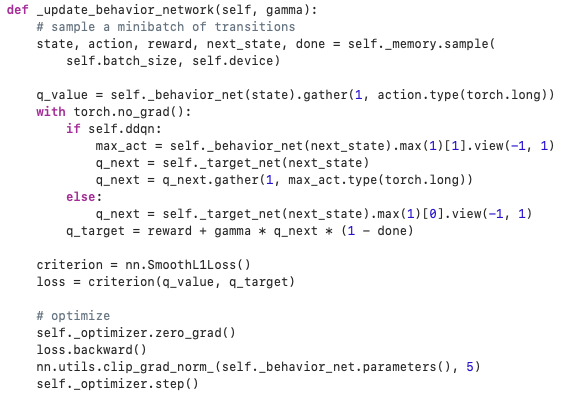
DQN:



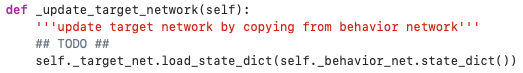
在 LunarLander-v2 中，env會給出8個observation，例如(Horizonal Coordinate , Vertical Coordinate, Horizontal Speed等等)，並不是給整個engine的畫面，所以我們可以用fully-connect來處理，而Lander會有4個動作，先經過hidden layer，最後再給4 dim作為輸出。



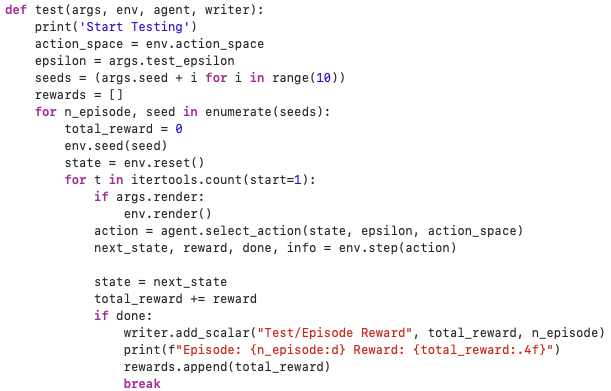
在選擇action的部分，這邊使用epsilon greedy的策略，避免陷入local的最佳解，先random出一個 0~1的數字，如果大於就使用behavior\_net所給出的action，反之則random給一個動作出來。



在這裡我們要用q\_value跟q\_target來計算loss，這邊q\_value是給當下這個時態的state給behavior\_net所給出來的值，而q\_target則是當下這個時態的state所做出的action所做出來的rewards，加上gamma乘以下一個時態的state：next\_state，預測下一個時態target\_net所遞迴推下去的直。利用兩者之差來計算loss。

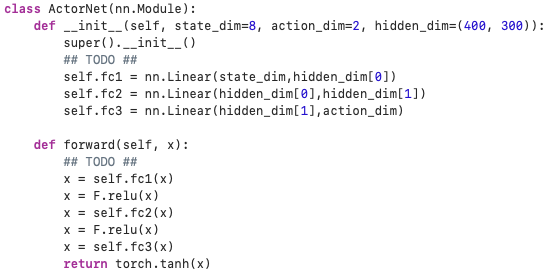


整個DQN有兩個網路一個是現在這一時態的網路和下一個時態延遲的網路，而這邊是單純是時態往後移，把在時間t的behavoir\_net交給時間t+1的target，就等於單純平移過去。

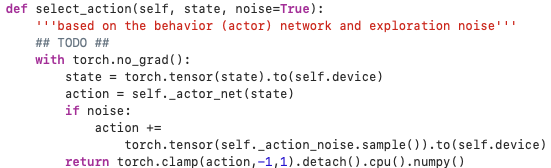


這邊就是單純的test，每次給出來的action交給env，直到整個episode結束。重複執行。

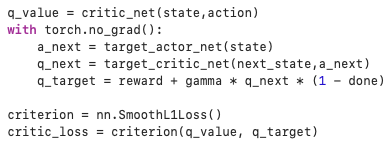
DDPG:



DDPG在網路架構與DQN最大的不同就是DDPG有ActorNet和CriticNet，而ActorNet的部分則與DQN一樣，相較於離散的問題，連續的網路hidden\_dim比較大。



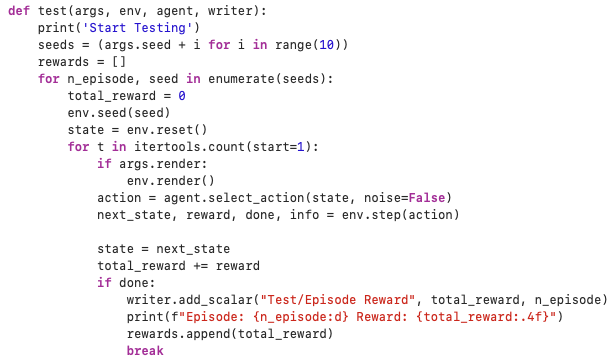
在這邊select\_action的時候，除了把當下的state交給網路外，在訓練的時候在給出的action vector加上noise可以幫助network更好的探索，否則可能會陷入網路自身的loacl 最佳解。在test時則不需要加上noise



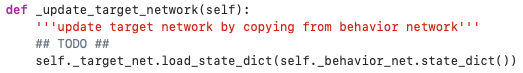
在DDPG中，q\_value的值是要由critic\_net來做評估的，action\_next則如DQN一樣是延遲的target\_actor\_net給予state所出來的，action\_next和下一個時態的state再交給延遲的target\_critic\_net所計算出，最後在用出來的q\_next用Q learning的公式和DQN一樣算出q\_target，再用兩者算出loss。



Action的部分則是單純的交給criticNet做評斷，將出來的數值做平均。



Test的部分上基本上跟DQN一樣，唯一的不同是，select\_action有noise的選項，在測試時需要關掉。



與DQN完全一致，將當前的網路傳給target(延遲)網路

* Describe differences between your implementation and algorithms:

DQN:

在DQN的演算法是在每一個時態都更新兩個不同的網路，但其實可以利用freq參數來改變不同的更新頻率。另外在一開始model什麼都不會和不清楚state、

action、rewards的關係時可以利用warmup的參數，產生隨機的動作供model做參考，這也是為什麼在訓練時，前面幾個episode會特別快的原因。

DDPG:

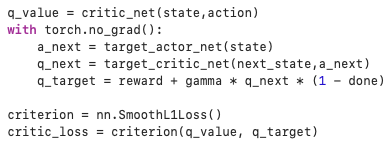
給予actor\_loss負的critic所給出得值可以令actor做小化loss就如同actor需要去學習如何才是最好的action。

* Describe your implementation and the gradient of actor updating:



如先前所述，actor\_loss是由critic\_net做 評斷，就如同GAN中的discriminator，為genernator做評斷一樣，actor和critic的關係就如同一個時根據經驗做出回饋而一個旁觀者依照當前的state和做出的action 所得到的reward做出評價，這就是critic所做的事情。

* Describe your implementation and the gradient of critic updating:



如同上所述，critic\_loss的計算就如同傳統Q-learning和DQN一樣，需要算出q\_target，q\_target是下一個時態所預估出來的值，我們想要逼近的，要利用q\_next乘上discount factor加上這次action應該要得到的reward，而q\_next則是需要下一個時態的state和action所利用target\_critic\_net所算出，利用還尚未更新的延遲網路做評估。當中所需的action\_next是利用target\_actor\_net所算出，一樣也是延遲網路。最後再利用兩個不同時態的q\_value做loss的計算。

* Explain effects of the discount factor:

可以給不同階段的rewards不同的權重，換句話說給越後面的權重越低，較注重越較近的狀態。

* Explain benefits of epsilon-greedy in comparison to greedy action selection:

如果沒有使用epsilon-greedy的話，network會一直給予最greedy的action，這樣子就很容易陷入到一個local 最佳的誤區，但如果使用epsilon-greedy的話就會給予亂走action的機會，這樣相較之下就可以嘗試更多不同的動作得到不同的反饋，多多探索可能整體會比較好。

* Explain the necessity of the target network:

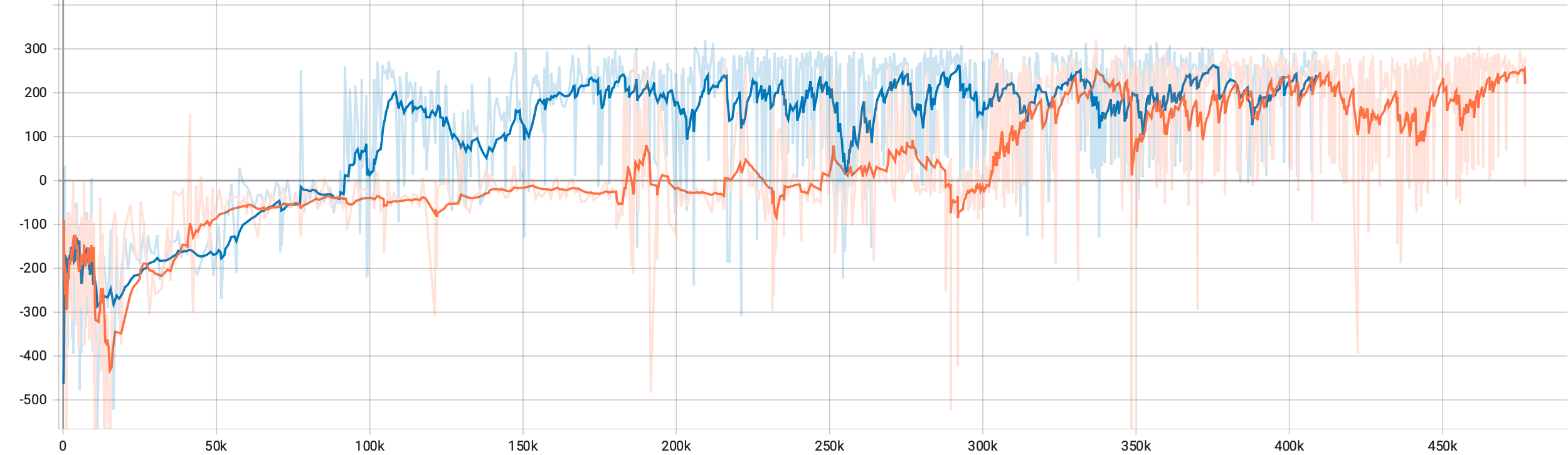
如果只有單一網路，用一個網路的參數做預測及更新權重，會造成以下問題，如果用一個網路做預測，在當你更新參數的時候，你更新之後的網路預測會做改動，這樣下來，整體就不會穩定，數值會一直變動。這時候如果有一個網路做延遲一個時態的話，相較之下就會穩定很多。

* Explain the effect of replay buffer size in case of too large or too small:

Buffer size會影響模型的訓練速度和結果的成效，buffer太大的話抽出來的memory可能會與當下的state不大相關，而且需要久久一次才能使用到離現在較近的記憶。Buffer太小則記憶的目的就不顯著，且可能會造成overfitting。

* Report Bonus:
* Implement and experiment on Double-DQN:

在原生的DQN的training時的reward經常會overestimate，有點高估所得到的reward的現象，如下圖，藍色的部分是DQN、紅色則是DDQN，在前三分之二的地方可以觀察到，藍色是明顯高於紅色的，原因在於當在算q\_value時是使用舊的參數做運算，舊的參數尚未更新至新的時態，所以會比正確的q\_value有高估的情況。造成整體結果的過於樂觀。

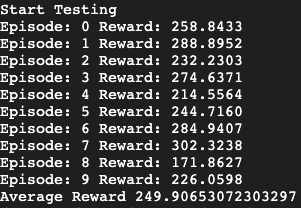


而DDQN則是用來解決這overestimate的問題，想法就是利用DQN原本就現有的behavior\_net和target\_net做雙層的prediction，原本q\_next只需要一個target\_net做決策，DDQN的q\_next則需要先求用當前時態的behavior\_net決定出action再給q\_next共同做出參考的依據。

* Extra hyperparameter tuning, e.g., Population Based Training:

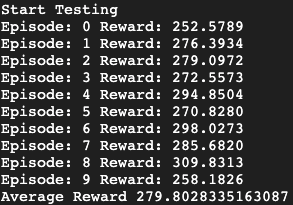
沒做。

* Performance:
* [LunarLander-v2] Average reward of 10 testing episodes: Average ÷ 30:



249.9065/30=8.330

* [LunarLanderContinuous-v2] Average reward of 10 testing episodes: Average ÷ 30:



279.8028/30=9.326