ADL A3 report

1. Basic Performance (6%)

Describe your Policy Gradient & DQN model (1% + 1%)

a. Policy gradient:

原則上是使用 vanilla policy gradient。但是沒有使用 baseline。

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

再計算 policy gradient estimate 的時候，我是用

T：total timesteps of 1 trajectory

$$\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)(\sum_{k=t}^{T-1} R(s_k, a_k))$$

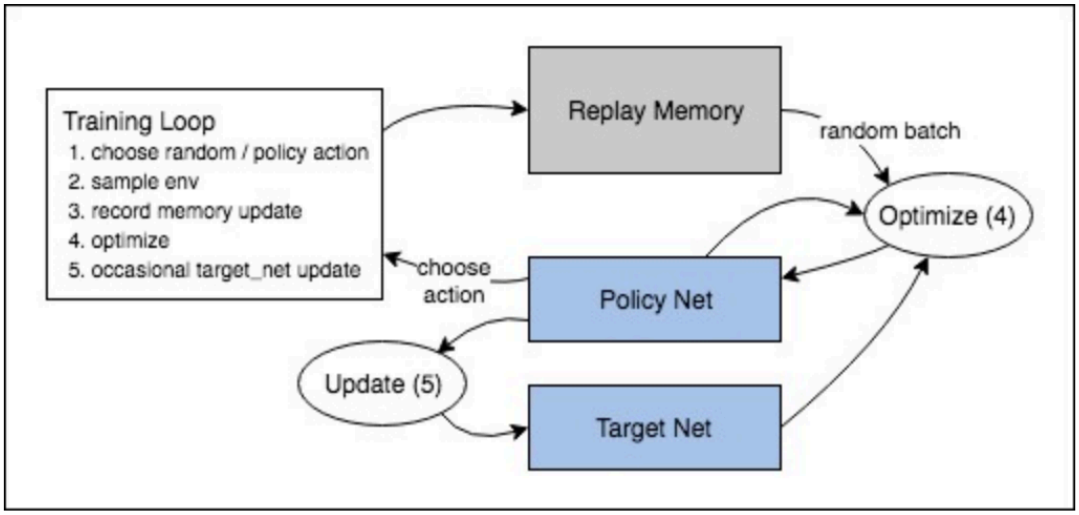
在這裏還沒加上 baseline。

但因為是計算 loss，所以

$$loss = -\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)(\sum_{k=t}^{T-1} R(s_k, a_k))$$

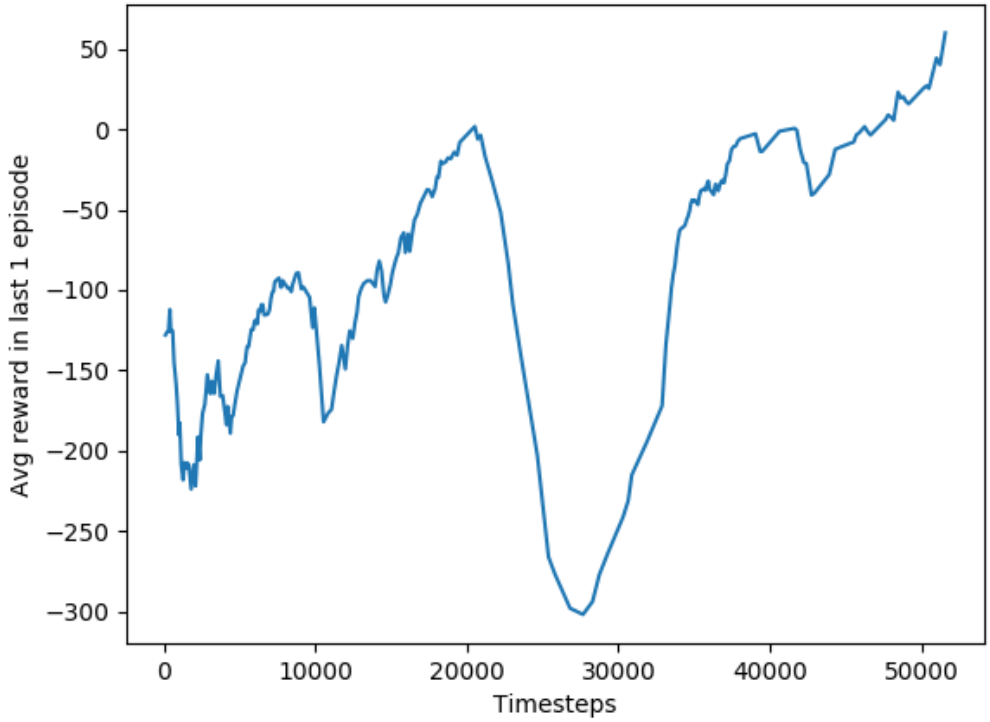
b. DQN:

實作基本上是參考 pytorch 官網的 DQN。下圖中的 policy net 就是 sample code 中的 online_net。



基本上用 policy net 選動作的策略是使用 epsilon greedy policy。如果隨機選出的數字超過了 eps_threshold，就表示我們不要 explore 新的動作，基於原本的 policy 來做動作。其他情況都是隨機選動作，也就是我們要 explore 新的動作。我們會將每個 timestep 的(s_t, a_t, s_{t+1}, r_t)存在 replay memory 裡面。固定一段時間就會 update policy net 中的參數。Optimization 會隨機選一個 batch 的 replay memory 來訓練 policy net。Target net 偶爾也會更新(每 1000 steps 更新一次)。

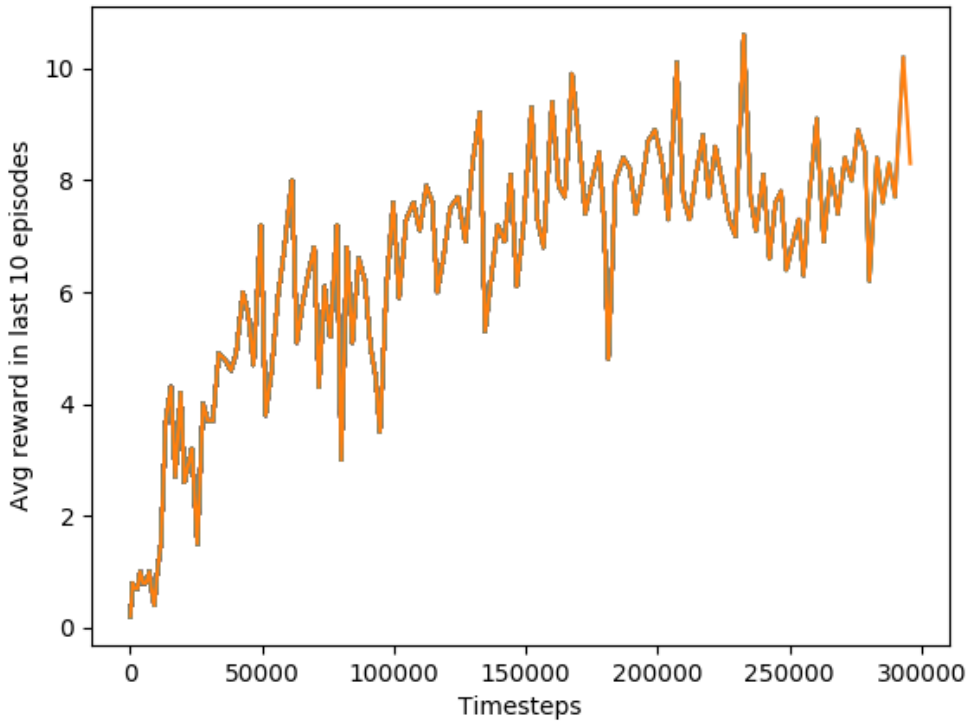
Plot the learning curve to show the performance of your Policy Gradient on LunarLander (2%)



可以觀察到訓練的過程不太穩定。Avg Reward 有急遽下降的情形。

R07922118 資工碩一 翁子騰

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

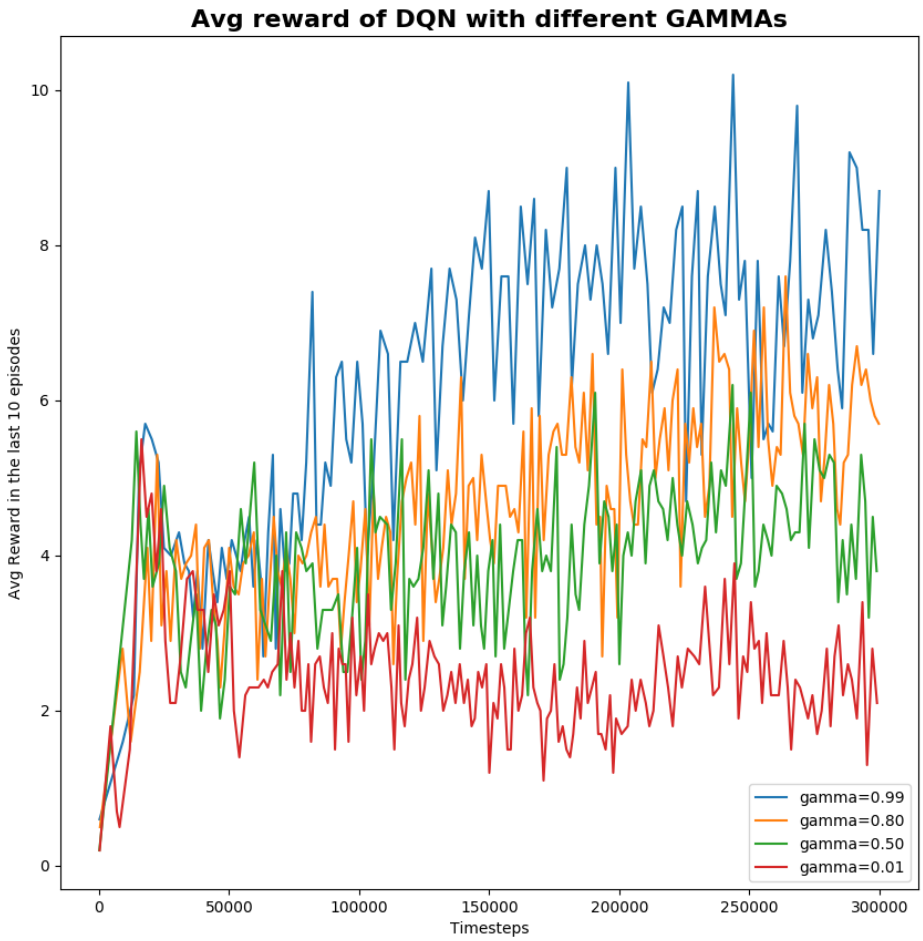


DQN 訓練過程 variance 也是很大，繼續往下訓練的話，avg reward 會介於 9~12 之間。

2. Experimenting with DQN hyperparameters (2%)

選擇 GAMMA 來做實驗。

Plot all four learning curves in the same figure (1%)



Explain why you choose this hyperparameter and how it affect the results (0.5% + 0.5%)

GAMMA 值代表的含義是未來的回報重不重要。如果 GAMMA 值越高，表示未來的回報越重要，反之，如果 GAMMA 值越低，未來的回報越不重要。

為什麼選擇 GAMMA 主要是想看看如果讓 agent 比較重視當前的回報表現會如何？也就是讓 agent 比較短視近利。

結果顯示，如果 GAMMA 值越低(表示 agent 比較重視眼前的利益，不重視未來的利益)，平均的回報會比較差。所以眼光還是要放遠，讓 agent 比較重視未來的利益，最終才能獲得比較高的平均回報。

3. Improvements to Policy Gradient & DQN / Other RL methods (2% + 2%)

A. Variance reduction by adding a baseline

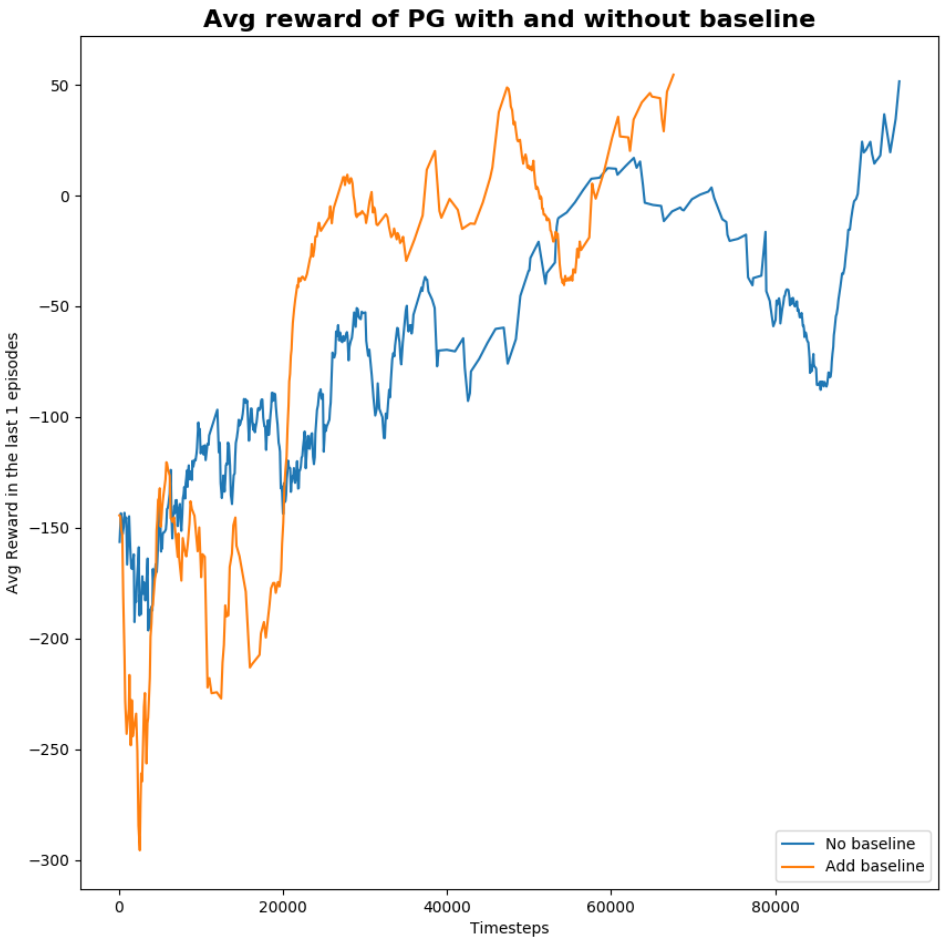
在原本 Q1 中加上 baseline，因爲減少 variance，所以表現比較好。

$$loss = -\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)(\sum_{k=t}^{T-1} R(s_k, a_k) - b)$$

其中 baseline 是 cumulative reward 的平均值。

$$b = \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=t}^{T-1} R(s_k, a_k)$$

由下圖可發現，加上 baseline 後，avg reward 上升的比較快。



B. Double DQN

原本的DQN中的Q value可能被高估，導致會一直選到被高估的動作**:**

$$\max_{a'} \hat{Q}(s', a', w^{-}), \ \ where\ \hat{Q}\ \ is\ the\ target\ network$$

程式碼是

next_state_values[non_final_mask]=

self.target_net(non_final_next_states).max(1)[0].detach()

為了減少target network高估的Q值，我們用原本的online network來估計target network中的a’。

即將上述式子改為

$$\hat{Q}\left(s', arg\max_{a'} Q(s', a', w), w^{-}\right),$$

$$where\ \hat{Q}\ is\ the\ target\ network\ and\ Q\ is\ the\ online\ network$$

實作上不難，只要將原本的 DQN中改成

next_state_actions =self.online_net(non_final_next_states).max(1)[1].unsqueeze(1)

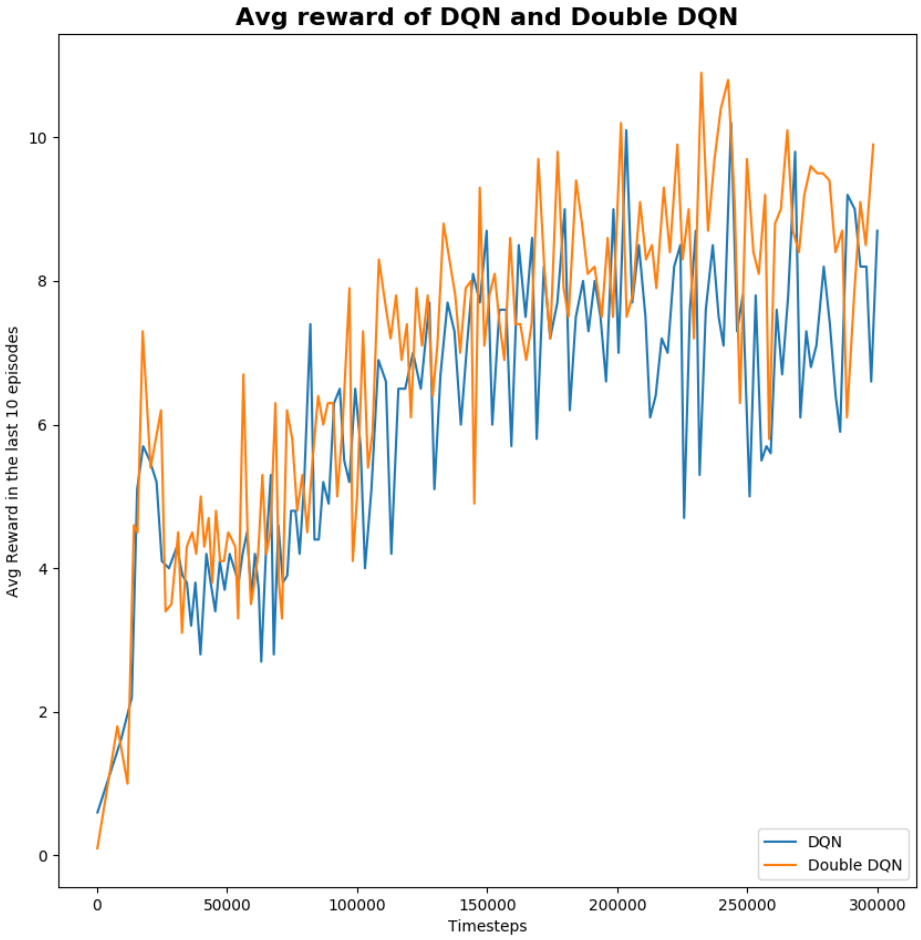
#以上這步是$arg\max_{a'} Q(s', a', w)$

next_state_values[non_final_mask]

=self.target_net(non_final_next_states).gather(1, next_state_actions).squeeze().detach()

R07922118 資工碩一 翁子騰

Environment: AssaultNoFrameskip-v0



由上圖中可以觀察到 Double DQN 的平均回報會一直比 DQN 好。

Gamma=0.99, trained for 300,000 steps; Run 100 episodes when testing

	DQN	Double DQN
Mean Reward	166.1	170.3