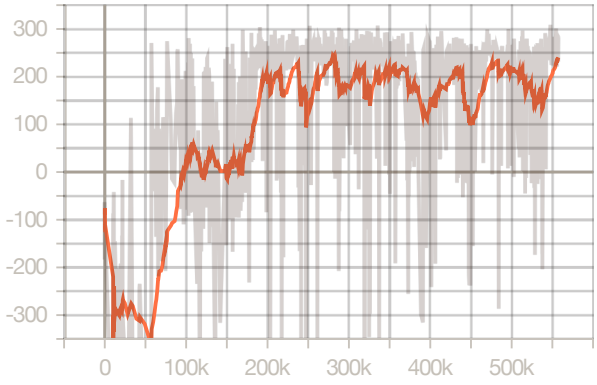
DLP Lab6 309551067 吳子涵

- Report

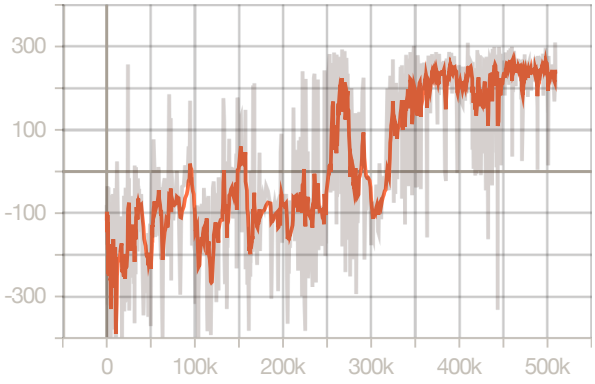
1. A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLander-v2

下圖為 train 1200 episodes 的圖，橫軸為 total steps，縱軸為 rewards。



2. A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLanderContinuous-v2

下圖為 train 1200 episodes 的圖，橫軸為 total steps，縱軸為 rewards。



3. Describe your major implementation of both algorithms in detail.

- DQN

(1) 隨機初始化一個 state、初始化 Replay Memory。

(2) 重複執行多個 episode，而每個 episode 中，會一直往下個 state 走，直到走到終點。

i. 選擇 action：在還沒達到 warmup 的步數前，會先從 Replay Memory 隨機抽取 action，而在達到 warmup 的步數後，就會使用經過訓練的 agent 來選擇 action，而選擇 action 時採用 ε-greedy 的方式，有 ε 的機率隨機選，(1-ε)的機率選最大 Q-value 者。而 ε 會從 1 逐漸下降至 0.01。

ii. 執行 action，得到下一個 state s’ 、reward、done (代表整個 episode 是否結束)。

iii. 將 (state, action, reward, next state) 儲存到 Replay Memory。

iv. 若已超過 warmup 的步數，則代表開始訓練，因此要更新 network，而更新 behavior network 與 target network 的頻率不同。

- 更新 behavior network：
  - 從 Replay Memory 中隨機抽取數據來當作訓練樣本
  - 將目前的 state 輸入 behavior network，得到 Q-value
  - 根據公式算出 Q-target，若 s’ 為終點則 Q(s’,a’)=0

$$\text{target} = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

  - 使用 Q-value 以及 Q-target 計算 MSE loss，訓練神經網絡
- 更新 target network：隔一些步數後，直接複製 behavior network 的參數

v. 判斷整個 episode 是否結束：若結束了，reset 環境。若還沒結束，則往下一個狀態走，也就是將 state 更新為 next state。

- DDPG

(1) 隨機初始化一個 state、初始化 Replay Memory。

(2) 重複執行多個 episode，而每個 episode 中，會一直往下個 state 走，直到走到終點。

i. 選擇 action : 在還沒達到 warmup 的步數前，會先從 Replay Memory 隨機抽取 action，而在達到 warmup 的步數後，就會使用經過訓練的 actor network 來選擇 action，再加上高斯雜訊。

ii. 執行 action，得到下一個 state s’ 、reward、done。

iii. 將(state, action, reward, next state)儲存到 Replay Memory

iv. 若已超過 warmup 的步數，則代表開始訓練，因此要更新 behavior network、target actor network、target critic network。

- 更新 behavior network
  - update critic network: 從 Replay Memory 中隨機抽取一些數據來當作訓練樣本，將目前的(state, action)輸入 critic network，會得到 Q-value，再根據下列公式計算出 Q-target，最後使用 Q-value 以及 Q-target 計算 MSE loss，訓練神經網絡。

    $$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

    ps. 若 s_{i+1} 為終點，則 Q’ =0
  - update actor network: 將目前的 state 輸入 actor network，會得到 action，再將 (state, action) 放入 critic network，會得到 Q-value，希望這個值越大越好，因此可以把 loss 設成 (-1) * Q-value，等同於希望 loss 越小越好。
- 更新 target network：使用 soft update 的方式，較穩定。

v. 判斷整個 episode 是否結束：若結束了，reset 環境。若還沒結束，則往下一個狀態走，也就是將 state 更新為 next state。

4. Describe differences between your implementation and algorithms.

我的 implementation 比原本的 algorithm 多加了 warm-up 的機制，也就是在正式 training 之前，先儲存一些記憶。在 warm-up 的階段，用 random 的方式選擇 action，並儲存這次的 experience，但不訓練網絡。

5. Describe your implementation and the gradient of actor updating.

- The gradient of actor updating:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \underbrace{\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)}}_{\text{第一項}} \underbrace{\nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}}_{\text{第二項}}$$

- 第一項：從 critic network 來的，希望能找出：這次 actor 要採取怎麼樣的動作，才能獲得更大的 Q 值。
- 第二項：從 actor network 來的，希望能找出：actor 要怎麼樣自身修改參數，使得 actor 更有可能做這個動作。
- Implementation:
  - 將目前的 state 輸入 actor network，會得到 action。
  - 將 (state, action) 放入 critic network，會得到 Q-value，希望這個值越大越好。
  - 因此可以把 loss 設成 (-1) * Q-value，將問題轉換為：希望 loss 越小越好。

```
action = actor_net(state)
actor_loss = -torch.mean( critic_net(state, action) )
```

6. Describe your implementation and the gradient of critic updating.

- The gradient of critic updating: ∇L

$$\text{Set } y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$
$$\text{Update critic by minimizing the loss: } L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

- 上式中，y_i即為 Q-target，計算 Q-target 與 Q-value 的 MSE loss，並對 θ 取偏微分，∇L 即為 critic updating 的 gradient。
- Implementation
  - 將目前的 state 輸入 critic network，會得到 Q-value。
  - 根據上述公式算出 Q-target。
  - 計算 Q-target 與 Q-value 的 MSE loss，進行 back-propagation，訓練神經網絡。

7. Explain effects of the discount factor.

因為 discount factor γ∈[0,1]，由下式可知，當下的 reward 是影響最大的，越是未來所給的 reward，影響越小。

$$G_t = R_{t+1} + \lambda R_{t+2} + \ldots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

discount factor 定義了未來的重要性，若 γ=0 代表只考慮當下的 reward(短期)，若 γ=1 代表更重視長期的 reward。

8. Explain benefits of epsilon-greedy in comparison to greedy action selection.

epsilon-greedy 有兩個模式：exploration 以及 exploitation。

- exploitation: (1-ε)的機率選最大 Q-value 者。
- exploration: 有 ε 的機率隨機選擇 action，這是 greedy action selection 沒有的部分。有了隨機選 action 的機制，才能學到新的經驗，且能即時改變策略，避免卡在非最佳的狀態。

9. Explain the necessity of the target network.

若沒有使用 target network，會同時計算 Q-target 和 Q-value，同一個 network 在更新，Q-target 也在更新，因此會有自己在跟自己的 Q-target 做比較的情況，導致不穩定。

加入了 target network 後，因為 target network 的參數 θ'不會更新那麼頻繁，而是經過一定時間才從 behavior network θ 里複製過來一次，這樣能降低 Q-value 與 Q-target 的關聯性，因此可以保證 target 的穩定性。

10. Explain the effect of replay buffer size in case of too large or too small.

若 replay buffer 太小，則 replay buffer 幾乎沒有作用，沒辦法打亂樣本之間的相關性，影響收斂效果。

若 replay buffer 太大，agent 可能需要經過很長時間才能學到新的 experience，training 的速度較慢，並且需要很大的 memory。

- Performance
  - LunarLander-v2

    Average Reward: 257.97

```
Start Testing
total reward:  284.8618638718656
total reward:  294.5516456748393
total reward:  248.4260547929432
total reward:  248.69059795986354
total reward:  244.13470937007074
total reward:  267.83041367417104
total reward:  240.82858926725564
total reward:  276.56329456306923
total reward:  257.67197632598925
total reward:  216.20496064462935
Average Reward 257.9764106144697
```

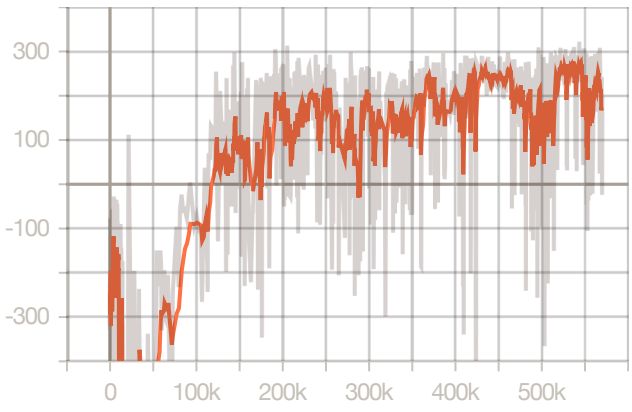
- LunarLanderContinuous-v2

  Average Reward: 262.12

```
Start Testing
total reward:  253.48114583072575
total reward:  278.59516905149394
total reward:  285.43396419667937
total reward:  268.1827027733932
total reward:  215.9502325730664
total reward:  270.7488275390715
total reward:  212.61258070388794
total reward:  288.4135052207918
total reward:  295.1694051875742
total reward:  252.63708426843073
Average Reward 262.1224617345115
```

- Report Bonus
  - Double DQN:
    - Episode Reward

      下圖為 train 1200 episodes 的圖，橫軸為 total steps，縱軸為 rewards。



- Performance

  Average Reward: 259.07

```
Start Testing
total reward:  248.48518249532478
total reward:  284.3887252491537
total reward:  244.83176340640094
total reward:  284.9055540378313
total reward:  251.57540116786242
total reward:  236.79056216388986
total reward:  264.33572715521495
total reward:  267.40075002145886
total reward:  274.0597634809037
total reward:  233.9808086141155
Average Reward 259.0754237792156
```

- Implementation

  Double DQN 是 DQN 的改良版，希望改善 over-estimation 的問題。而實作改變的地方在於，計算 Q-target 的算法不同。

  - DQN: 取 target network 輸出的 Q-value 中最大者。
  - Double DQN: 取 behavior network 輸出的 Q-value 最大值的 index，然後使用 target network 算出 Q-value，再用 index 取出對應的 Q 值。
  - 以下是 DQN 與 Double DQN 的公式比較。

Basic Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Double Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \boxed{Q'(s_{t+1}, \boxed{a})} - Q(s_t, a_t))$$

estimated/expected Q-value

$$\boxed{a} = \max_a Q(s_{t+1}, a)$$

$$q_{estimated} = \boxed{Q'(s_{t+1}, \boxed{a})}$$