Lab6 Deep Q-Network and Deep Deterministic Policy Gradient

311551040

邱以中

1. Introduction

這次的 lab 我們目標是使用 RL 的方法(DQN、DDPG)，來學習玩 LunarLander 與 Breakout 兩個遊戲，並盡可能取得高分

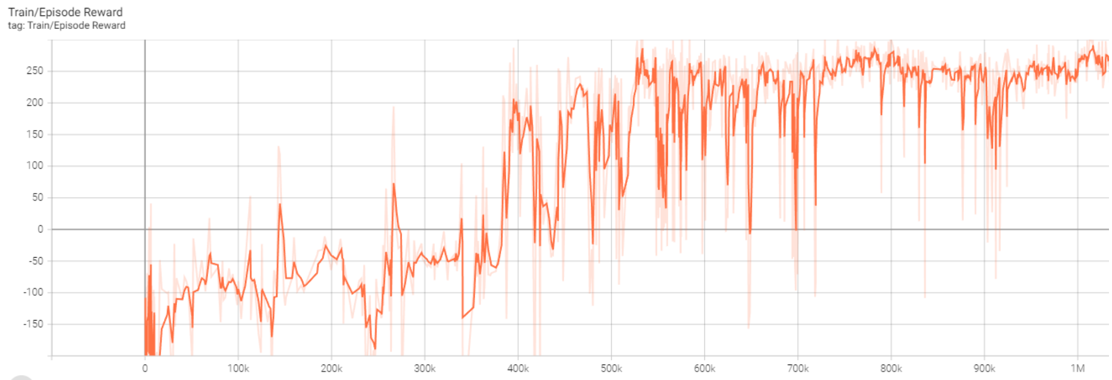
2. Experimental Results

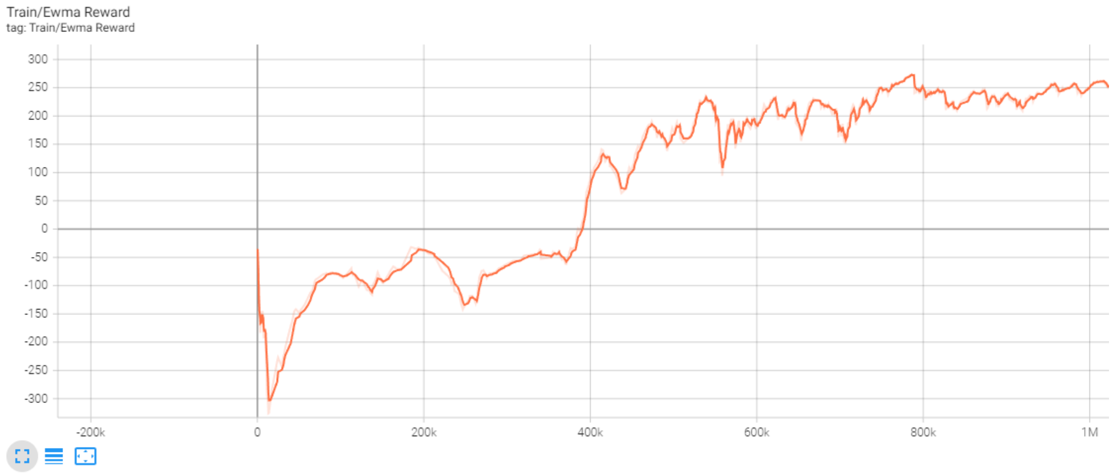
- **LunarLander-v2 (DQN)**

(1) Test results

```
(DL) [redacted]~/NYCU_2023_ML/Lab6$ python dqn-example.py --test_only
[redacted]py:30: UserWarning: WARN: Box bound precision lowered
by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
total reward: 229.89213434039155
total reward: 257.6063929515375
total reward: 273.233228520618
total reward: 262.3758152140892
total reward: 281.80457590446406
total reward: 262.9685830605473
total reward: 284.1101798935341
total reward: 274.7487110527271
total reward: 295.10331511560275
total reward: 258.9064399468252
Average Reward 268.0749376000337
```

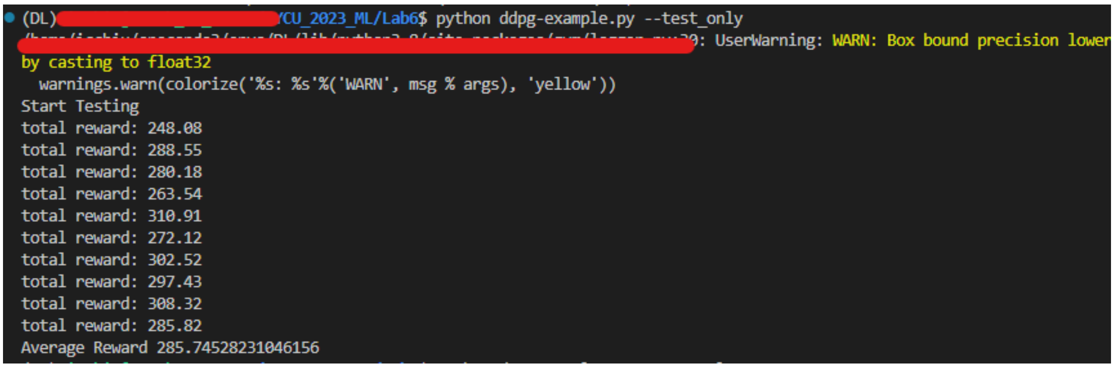
(2) Tensorboard





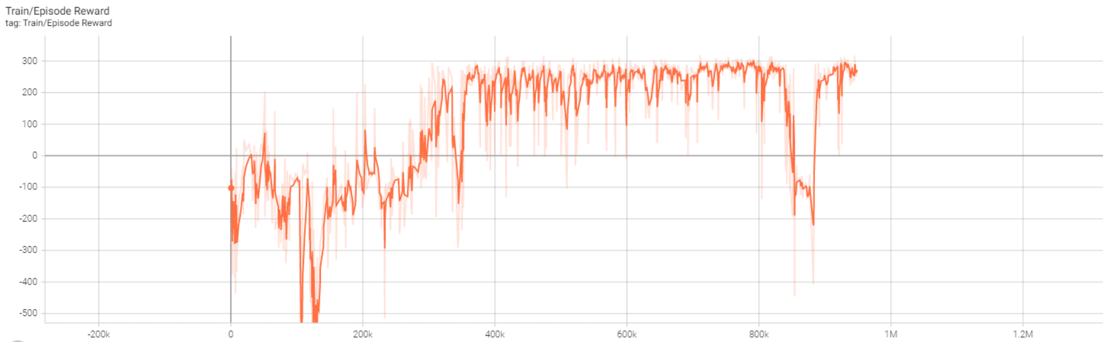
- **LunarLanderContinuous-v2 (DDPG)**

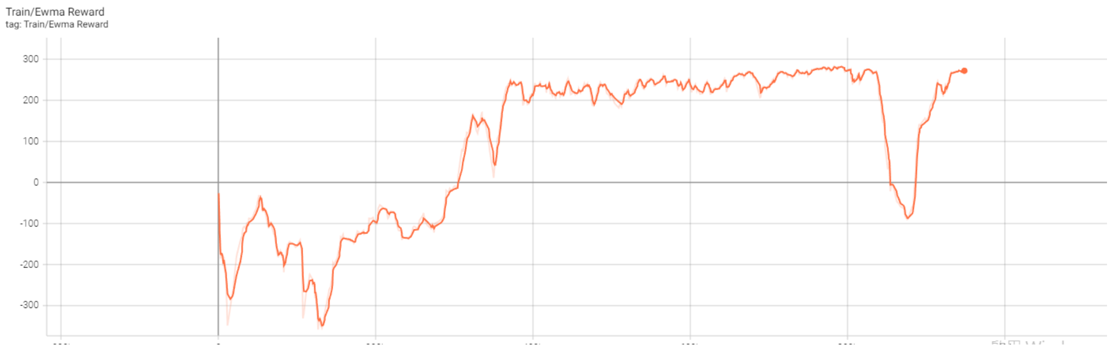
  (1) Test results



```
(DL) [redacted]YCU_2023_ML/Lab6$ python ddpg-example.py --test_only
[redacted]: UserWarning: WARN: Box bound precision lower
by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
total reward: 248.08
total reward: 288.55
total reward: 280.18
total reward: 263.54
total reward: 310.91
total reward: 272.12
total reward: 302.52
total reward: 297.43
total reward: 308.32
total reward: 285.82
Average Reward 285.74528231046156
```

  (2) Tensorboard



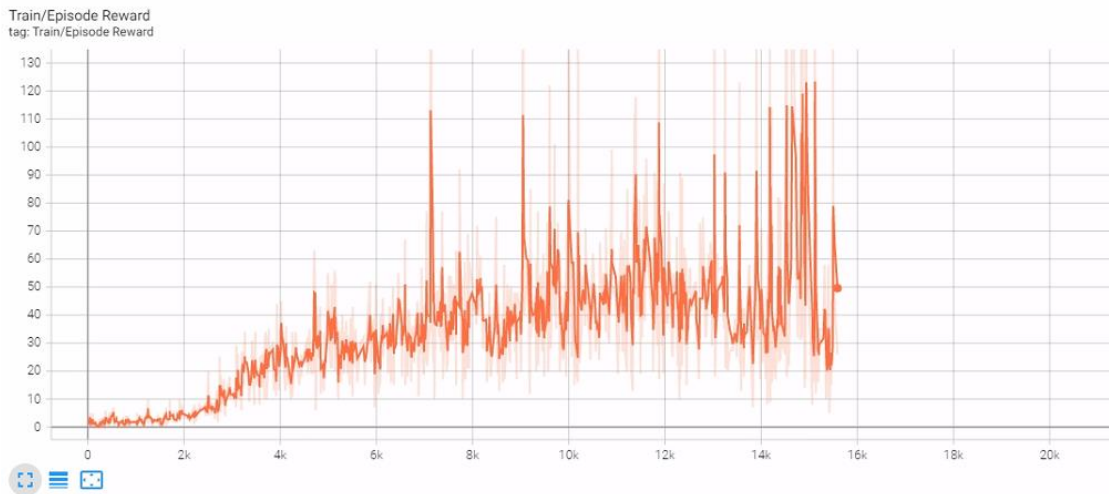


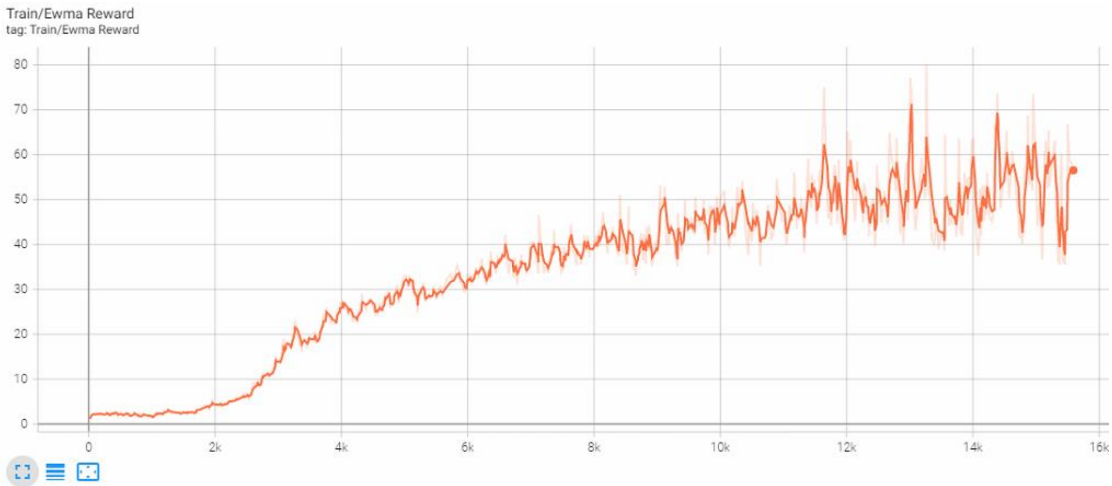
- **BreakoutNoFrameskip-v4 (DQN)**

(1) Test results

```
(DL) [redacted]$ python dqn_breakout_example.py --test_only
Start Testing
episode 1: 224.00
episode 2: 266.00
episode 3: 266.00
episode 4: 266.00
episode 5: 266.00
episode 6: 50.00
episode 7: 266.00
episode 8: 48.00
episode 9: 266.00
episode 10: 261.00
Average Reward: 217.90
```

(2) Tensorboard





3. Questions

- **Describe your major implementation of both DQN and DDPG in detail. Your description should at least contain three parts**

  (1) Your implementation of Q network updating in DQN.

在 DQN 當中，我會根據參數的定義來每隔一段時間更新一次 behavior network 與 target network。
在更新 behavior network 時，我會使用 td(0)的方式進行更新

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the

在更新 target network 時，我則會直接將 behavior network 的參數複製過去。

```python
    def update(self, total_steps):
        if total_steps % self.freq == 0:
            self._update_behavior_network(self.gamma)
        if total_steps % self.target_freq == 0:
            self._update_target_network()

    def _update_behavior_network(self, gamma):
        # sample a minibatch of transitions
        state, action, reward, next_state, done = self._memory.sample(
            self.batch_size, self.device)

        ## TODO ##
        # 根據過去的action做選擇
        q_value = self._behavior_net(state).gather(dim=1,index = action.long())
        with torch.no_grad():
           q_next = self._target_net(next_state).max(dim=1)[0].unsqueeze(-1)
           q_target = reward + gamma*q_next*(1-done)
        criterion = nn.MSELoss()
        loss = criterion(q_value, q_target)

        # optimize
        self._optimizer.zero_grad()
        loss.backward()
        nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
        self._optimizer.step()

    def _update_target_network(self):
        '''update target network by copying from behavior network'''
        ## TODO ##
        self._target_net.load_state_dict(self._behavior_net.state_dict())
```

(2) Your implementation and the gradient of actor updating in DDPG.
我們希望 actor 在 critic 得到的分數最大化，因此 loss 就是加上一個負號，代表分數越大時 loss 越小。

```python
## update actor ##
# actor loss
## TODO ##
action = self._actor_net(state)
actor_loss = -self._critic_net(state,action).mean()
```

(3) Your implementation and the gradient of critic updating in DDPG.

更新 critic 的方式與 dqn 相同都是使用 TD(0)，不過因為在 ddpg 當中的 actor 是連續的值，所以會先利用 actor network 預測出一個 actor，再把 actor 與 state 輸入 critic network 來預測 reward。

Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

```
## update critic ##
# critic loss
## TODO ##
q_value = self._critic_net(state,action)
with torch.no_grad():
    a_next = self._target_actor_net(next_state)
    q_next = self._target_critic_net(next_state,a_next)
    q_target = reward + gamma*q_next*(1-done)
criterion = nn.MSELoss()
critic_loss = criterion(q_value, q_target)
```

- **Explain effects of the discount factor**

  我們要計算的是當前 actor 得到 reward 的期望值，Discount factor 代表的是未來的 reward 對於現在的重要程度，與當前時間越遠的 reward 的重要程度就會越低。

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

  使用 greedy action selection 只會取最好的 actor，這樣可能會導致有些 actor 從來沒被選擇到，因此會使用 epsilon greedy 的方法來增加一些隨機性，讓每個 actor 都有被選擇到的機會。

- **Explain the necessity of the target network**

  因為我們的目標是讓 qvalue 與 qtarget 越接近越好，如果使用同一個 network 的話，在每次更新參數後，會導致 target 的值也不斷變動，讓訓練變得很不穩定，因此使用一個 target network，一段時間更新一次，讓 target 可以保持相對穩定會有助於模型的訓練。

- **Describe the tricks you used in Breakout and their effects, and how they differ from those used in LunarLander**

在 breakout 當中，我們會將多個 state stack 在一起當作輸入，這樣可以讓模型學到 temporal 的資訊，更好的預測下一時刻的動作。