Lab5: Deep Q-Network and Deep Deterministic Policy Gradient

Student id / name: A113599 / 楊淨富

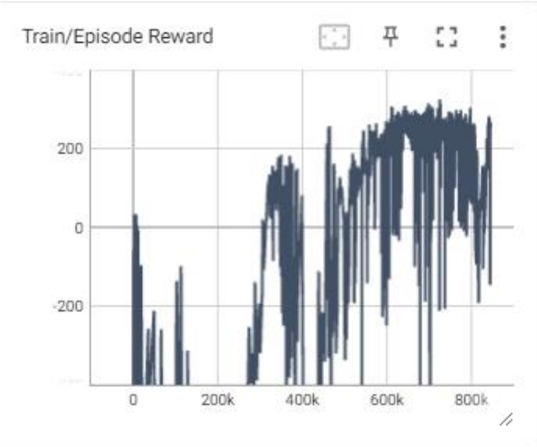
I. Experimental Results

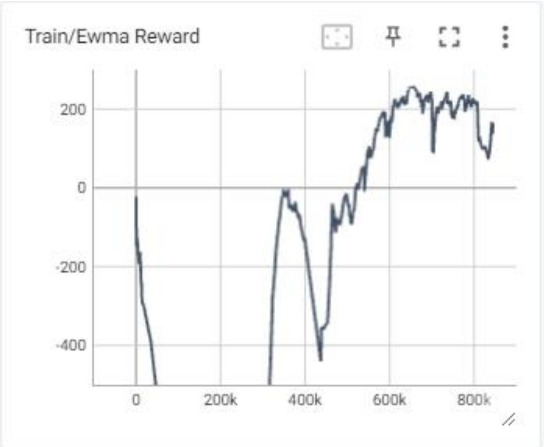
1. LunarLander-v2

- Testing results

```
(gym) PS C:\Users\User\Desktop\NYCU_DLP\Lab5_DQN_DDPG> python .\dqn.py --test_only
Start Testing
C:\Users\User\.conda\envs\gym\lib\site-packages\gym\utils\passive_env_checker.py:233: DeprecationWarning: `np.bool8` is
a deprecated alias for `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
Step : 214, total reward : 258.6304538332647
Step : 291, total reward : 267.9554522121974
Step : 193, total reward : 284.333667722644
Step : 242, total reward : 263.6037968967649
Step : 215, total reward : 249.2889058224383
Step : 216, total reward : 239.9302704567719
Step : 174, total reward : 307.32432807244027
Step : 217, total reward : 261.3129846031714
Step : 199, total reward : 262.7721580904026
Step : 99, total reward : 42.10943471788286
Average Reward 243.72614524279783
```

- Tensorboard



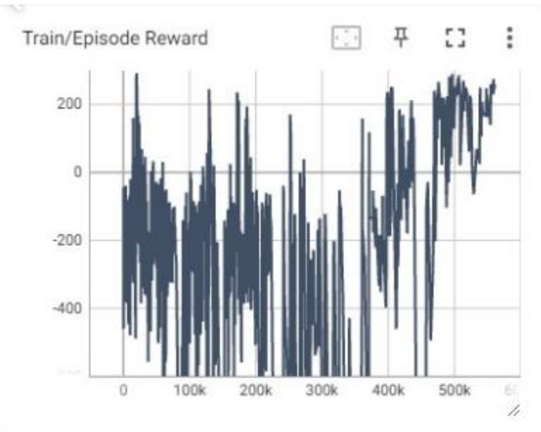


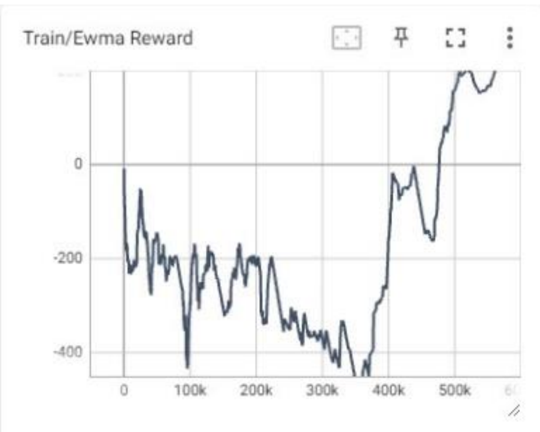
2. LunarLanderContinuous-v2

- Testing results

```
(gym) PS C:\Users\User\Desktop\NYCU_DLP\Lab5_DQN_DDPG> python .\ddpg.py --test_only
Start Testing
C:\Users\User\.conda\envs\gym\lib\site-packages\gym\utils\passive_env_checker.py:233: DeprecationWarning: `np.bool8` is
a deprecated alias for `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
Total reward : 320.7721988352614
Total reward : 274.7602092816601
Total reward : 232.8550529131739
Total reward : 242.43364661548875
Total reward : 205.64132221627125
Total reward : 273.84622384410704
Total reward : 233.53033473122167
Total reward : 166.88104938058
Total reward : 303.58176325078864
Total reward : 233.30635088151843
Average Reward 248.76081519500713
```

- Tensorboard





3. BreakoutNoFrameskip-v4

- Testing results

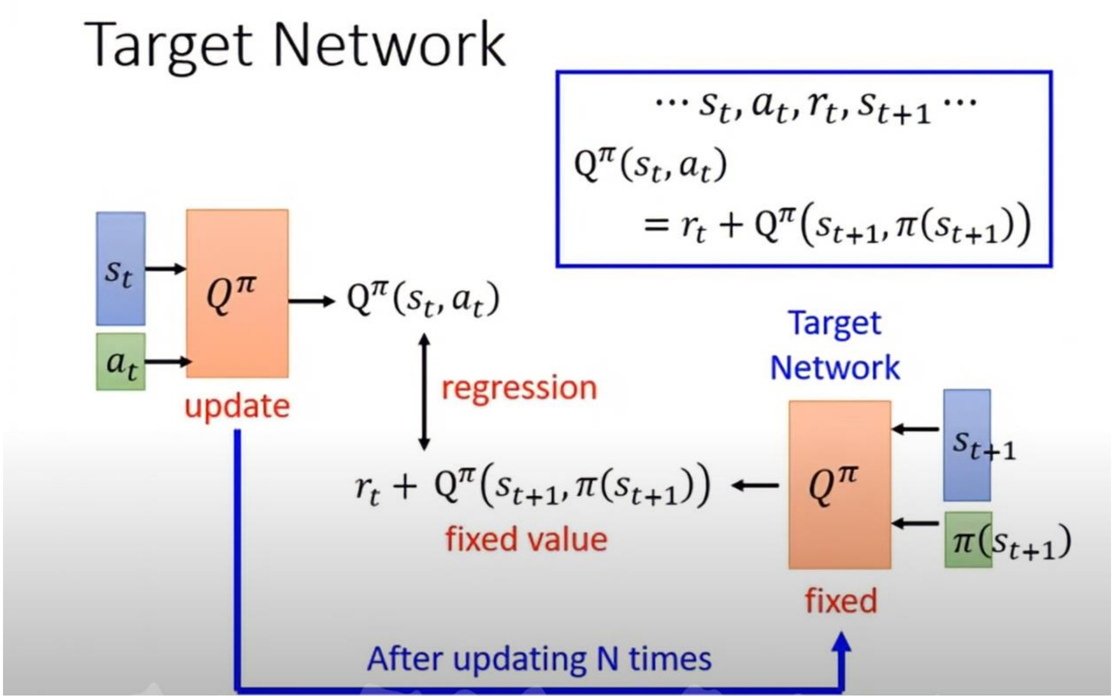
```
(br) PS C:\Users\User\Desktop\NYCU_DLP\Lab5_DQN_DDPG> python .\dqn_breakout_v2.py --test_only
Start Testing
episode 1: 39.00
episode 2: 61.00
episode 3: 1.00
episode 4: 4.00
episode 5: 2.00
episode 6: 37.00
episode 7: 20.00
episode 8: 46.00
episode 9: 1.00
episode 10: 2.00
Average Reward: 21.30
```

- Tensorboard

II. Questions

1. 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.



```python
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
    ## TODO ##
    if random.random() < epsilon:
        return action_space.sample()

    with torch.no_grad():
        q_values = self._behavior_net(torch.from_numpy(state).view(1, -1).to(self.device))
        _, best_action_index = q_values.max(dim=1)
    return best_action_index.item()
```

用epsilon-greedy來取得當前的a_t值(即action)，一開始epsilon設為1，目的是在初期進行exploration，若隨機數>= epsilon，則會丟到Q Net計算Q(s_t, a_t)的值。

```python
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)

    q_value = self._behavior_net(state) # 用model對當前state，得到預測的Q值:(batch_size, num_actions)
    q_value = torch.gather(input=q_value, dim=1, index=action.long()) # 在預測的Q值tensor中選取相應的動作索引的元素
    with torch.no_grad():
        q_next = self._target_net(next_state)
        q_next, _ = torch.max(q_next, dim=1)
        q_next = q_next.reshape(-1, 1) # 轉換成(batch_size, 1)
        q_target = reward + gamma * q_next * (1 - done)
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)
```

從Replay buffer裡面sample出一個minibatch，即過去的experience。並計算他們的Q value用來更新behavior network，此舉能夠減少actor和環境互動的次數，提升training process的效率。

```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()
```

每args.freq步，就將behavior network進行更新。

而每args.target_freq步，就將target network進行更新，具體做法就是直接把

behavior network copy到target network。

(2) Your implementation and the gradient of actor updating in DDPG.

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

DDPG(Deep Deterministic Policy Gradient)適用於continuous action space，主要是用使用兩個神經網路(actor和critic)以及Experience Replay的方法，來有效學習Policy和Action value function。其中Policy可以是deterministic或是機率分布(對於每個狀態都有一個機率分布來選擇動作)，而Action value function，即Q function，是用來estimate在特定狀態下執行某個動作的價值。它表示agent從某個狀態開始，在選擇某個動作後，與其可以獲得的累積reward。

實作上就是在behavior network和target network上，都使用actor和critic，前者負責學習policy(生成動作)，後者負責估計reward(動作的價值)。

```python
def _update_behavior_network(self, gamma):
        actor_net, critic_net, target_actor_net, target_critic_net =
self._actor_net, self._critic_net, self._target_actor_net, self._target_critic_net
        actor_opt, critic_opt = self._actor_opt, self._critic_opt

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

        ## update critic ##
        # critic loss
        ## TODO ##
        state = state.to(torch.float32)
        action = action.to(torch.float32)
        next_state = next_state.to(torch.float32)

        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()
        q_value = q_value.to(torch.float32)
        q_target = q_target.to(torch.float32)
        critic_loss = criterion(q_value, q_target)

        # optimize critic
        actor_net.zero_grad()
        critic_net.zero_grad()
        critic_loss.backward()
        critic_opt.step()

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

        # optimize actor
        actor_net.zero_grad()
        critic_net.zero_grad()
        actor_loss.backward()
        actor_opt.step()
```

在update過程中，首先，透過計算q_value(用self._critic_net計算)，然後利用target network(self._target_actor_net)來預測目標狀態的下一步動作a_next，再用target network(self._target_critic_net)來計算目標狀態和目標動作的Q value(q_next)，並計算q_target，利用這兩者來算MSE loss，從而逼近真實的Q value。針對critic_loss進行back propagation，並update critic network。

計算actor的loss，目標是maximize actor network在當前狀態下所生成的

action經過critic network後的價值，即actor_loss = -self._critic_net(state, action).mean()。針對actor_loss進行back propagation，並update actor network。

2. Explain effects of the discount factor.

Discount factor，或稱gamma，這個值介於0到1之間，通常會設定比較接近1，比如0.99。當gamma接近1時，代表更重視長期回報，因為在訓練過程中，model更傾向於將未來的獎勵也考慮在內，這可以上訓練過程更穩定，並在長期內做出更好的策略。然後，過高的gamma可能會導致過於保守，忽略了眼前的獎勵，因為未來獎勵的價值被過度強調。

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

最主要的區別是讓 agent 在挑選 action 時，會有機會去執行 exploration，而非像 greedy action selection 每次都只挑選 Q value 最大的 action，這樣在某種狀態下，從來沒執行過某個 action，可能會錯過更好的 action。這就好像你去一家陌生的餐廳，點了打拋豬，覺得很好吃，所以每次你來都點打拋豬來吃，但是其實它的椒麻雞更好吃，而你這輩子都吃不到這麼好吃的椒麻雞了。

4. Explain the necessity of the target network.

因為在計算q_value與q_target中的q_next_state時，如果都用同一個network的話，當network在更新參數時，這兩個數值都會改變，代表想要fit的target

一直在變，會使得整個訓練過程很不穩定，不太好train。因此實際上，在計算q_target時，會fix target network以用來計算q_next_state的值，用來更新原本的behavior network，等過了一段時間， 再把behavior network複製給target network來進行更新。

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

	LunarLander	Breakout
Input state	輸入的 8 項 observation 即為 state	一次使用 4 幀當作 input state，即 stacked frames
Reward	除以 10，方便計算 Q value。	由於本身 reward 已經很小，沒有額外除以 10。
episode_life	無特別設定	Training 時，episode_life 設為 True，即讓 agent 在 env 失敗後，會強制重新開始一個新的 episode，即使在遊戲中間的某個狀態失敗也會被認為是 episode 結束，可以模擬現實中的遊戲畫面。
Clip_rewards	無特別設定	Training 時，clip_rewards 設為 True，則 reward 會被截斷 (clipped)。在某些情況下，遊戲會給極大或極小的獎勵，大幅度的獎勵可能導致網路不穩定。透過設定 clip_rewards，獎勵會被截斷到一個較小的範圍內，通常是 [-1, 1]的區間內，從而緩解這種問題。