# Lab 6: Deep Q-Network and Deep Deterministic Policy Gradient

**張千祐**
Department of Computer Science
National Yang Ming Chiao Tung University
qianyou.cs11@nycu.edu.tw

## 1 Experimental results

### 1.1 LunarLander-v2

- Testing results

```
❯ python3 dqn.py -d cpu --test_only
/Users/qianyou/Desktop/nctu/assignment/master_1_2/dlp/lab6/venv/lib/python3.9/si
te-packages/gym/logger.py:30: UserWarning: WARN: Box bound precision lowered by
casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode: 0      Length: 282     Total reward: 236.49
Episode: 1      Length: 331     Total reward: 258.36
Episode: 2      Length: 277     Total reward: 269.42
Episode: 3      Length: 160     Total reward: 23.03
Episode: 4      Length: 334     Total reward: 253.08
Episode: 5      Length: 320     Total reward: 247.99
Episode: 6      Length: 339     Total reward: 260.89
Episode: 7      Length: 284     Total reward: 235.05
Episode: 8      Length: 351     Total reward: 259.19
Episode: 9      Length: 290     Total reward: 272.56
Average Reward 231.6070204464924
```

Figure 1: Testing result on LunarLander-v2

- Tensorboard

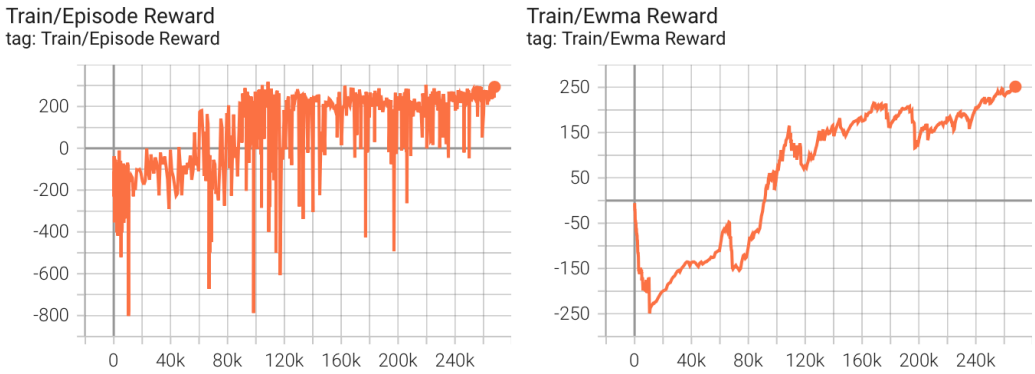


Figure 2: Tensorboard on LunarLander-v2

### 1.2 LunarLanderContinuous-v2

- Testing results

```
❯ python3 ddpg.py -d cpu --test_only
/Users/qianyou/Desktop/nctu/assignment/master_1_2/dlp/lab6/venv/lib/python3.9/si
te-packages/gym/logger.py:30: UserWarning: WARN: Box bound precision lowered by
casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode: 0      Length: 1000    Total reward: 123.11
Episode: 1      Length: 160     Total reward: 233.60
Episode: 2      Length: 231     Total reward: 278.80
Episode: 3      Length: 189     Total reward: 258.07
Episode: 4      Length: 182     Total reward: 232.14
Episode: 5      Length: 182     Total reward: 275.98
Episode: 6      Length: 283     Total reward: 238.84
Episode: 7      Length: 179     Total reward: 247.33
Episode: 8      Length: 195     Total reward: 298.03
Episode: 9      Length: 567     Total reward: 283.72
Average Reward 246.9627286097681
```

Figure 3: Testing result on LunarLanderContinuous-v2

- Tensorboard

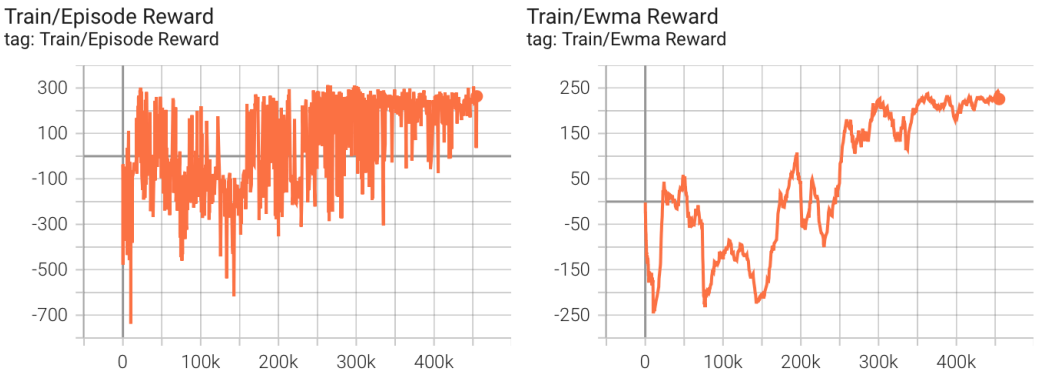


Figure 4: Tensorboard on LunarLanderContinuous-v2

### 1.3 BreakoutNoFrameskip-v4

- Testing results

```
(venv) qianyou@lab708-Default-string:/708HDD/qianyou/dlp/lab6$ python3 dqn_break
out.py --test_only -tmp ckpt/dqn_16400000.pt
Start Testing
episode 1: 341.00
episode 2: 308.00
episode 3: 327.00
episode 4: 286.00
episode 5: 338.00
episode 6: 312.00
episode 7: 312.00
episode 8: 317.00
episode 9: 314.00
episode 10: 317.00
Average Reward: 317.20
```

Figure 5: Testing result on BreakoutNoFrameskip-v4

- Tensorboard

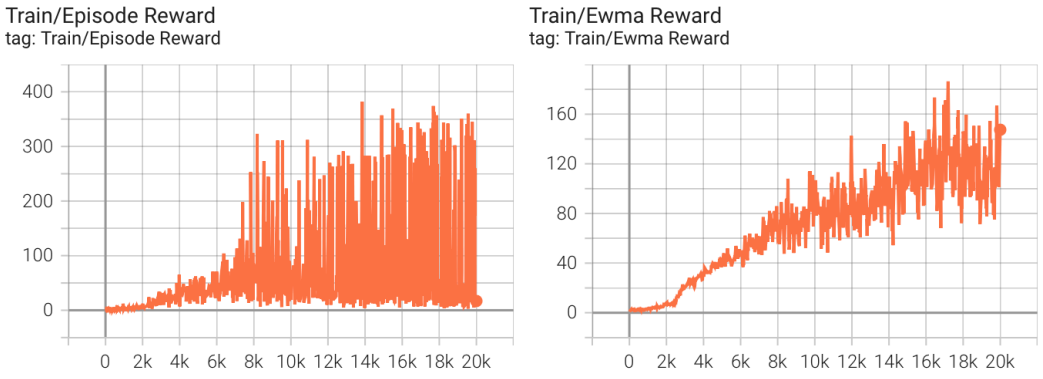


Figure 6: Tensorboard on BreakoutNoFrameskip-v4

## 2 Questions

### 2.1 Describe your major implementation of both DQN and DDPG in detail.

#### 2.1.1 Your implementation of Q network updating in DQN.

- Behavior network：首先會從 replay buffer 中取出 batch size 的 transitions，接著分別計算 Q 的 value 和 target，value 會取 behavior network 的 output，另外會取 target network 的 max 當作 target，接著便可取兩者的 MSELoss 當作 loss 作更新。
- Target network：更新較爲單純，每隔幾步後會進行一次 hard update 成當前的 behavior network。

```
    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 ##
        q_value = self._behavior_net(state).gather(1, action.long())
        with torch.no_grad():
            q_next = self._target_net(next_state).max(dim=1)[0].view
    (-1, 1)
            q_target = reward + gamma * (1-done) * q_next
        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
    ())
```

Listing 1: DQN update

### 2.1.2 Your implementation and the gradient of actor updating in DDPG.

$$\nabla_{\theta^\mu} J_\mu = E[\nabla_\mu Q(s, \mu(s)) \nabla_{\theta^\mu} \mu(s)]$$

這是 actor 部分的 gradient，因此這邊我們取負的 Q value 的平均當作我們的 actor loss。

### 2.1.3 Your implementation and the gradient of critic updating in DDPG.

Critic 部分跟 DQN 雷同，我們直接拿 TD error 當作 critic loss，差別只在於 ddpg 這邊 next state 的 action 是直接取 target actor network 的 output，不用取 argmax。

```
    def update(self):
        # update the behavior networks
        self._update_behavior_network(self.gamma)
        # update the target networks
        self._update_target_network(self._target_actor_net, self._actor_net, self.tau)
        self._update_target_network(self._target_critic_net, self._critic_net, self.tau)

    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 ##
        q_value = critic_net(state, action)
        with torch.no_grad():
           a_next = target_actor_net(next_state)
           q_next = target_critic_net(next_state, a_next)
           q_target = reward + self.gamma * (1-done) * q_next
        criterion = nn.MSELoss()
        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 = actor_net(state)
        actor_loss = -critic_net(state, action).mean()

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

    @staticmethod
    def _update_target_network(target_net, net, tau):
        '''update target network by _soft_ copying from behavior network'''
        for target, behavior in zip(target_net.parameters(), net.parameters()):
            ## TODO ##
```

```
49                 with torch.no_grad():
50                     target.copy_(tau * behavior + (1-tau) * target)
```

Listing 2: DDPG update

### 2.2 Explain effects of the discount factor.

1. 使得 value 不會發散
2. 越重視 Imediate reward 時，可將 discount factor 調整越接近 0，反之，則越接近 1。

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

Epsilon-greedy 是在 exploration 跟 exploitation 兩者間的平衡機制，在能夠兼顧 exploitation 的同時，確保有 epsilon 的機率能夠採取隨機的策略以達到 exploration 的目的。

### 2.4 Explain the necessity of the target network.

Target network 的作用是使得 training 的表現更穩定，可以用最後一道防線來比喻 target network，在訓練的過程中，我們會希望 behavior network 可以多嘗試略爲不同的策略，同時我們希望這個策略可以越來越好，但問題是這件事本身不保證會使得策略越來越好，因此我們用一個 target network 來確保 behavior network 在嘗試新的策略時，同時確保這個新的策略不會變差。

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

在 BreakoutNoFrameskip-v4 的環境中，我們會將 4 個 frame 堆疊起來作爲 1 個 state，原因是因爲在這環境我們只有影像的輸入，因此我們需要一小段的連續影像才能夠讓我們判斷球移動的方向和速度。不像是 LunarLander 的環境裡，我們會獲得位置、速度、角速度等 observation 的資訊，因此不需要另外將多個 transitions 的 observation 堆疊起來。