Deep Learning Lab 6

311552004 李明倫

1. Introduction

In this lab, you will learn and implement two deep reinforcement algorithms by completing the following three tasks:

- Solve LunarLander-v2 using deep Q-network (DQN)
- Solve LunarLanderContinuous-v2 using deep deterministic policy gradient (DDPG)
- Solve BreakoutNoFrameskip-v4 using deep Q-network (DQN)

Requirements

- Implement DQN
  - Construct the neural network
  - Select action according to epsilon-greedy
  - Construct Q-values and target Q-values
  - Calculate loss function
  - Update behavior and target network
  - Understand deep Q-learning mechanisms
- Implement DDPG
  - Construct neural networks of both actor and critic
  - Select action according to the actor and the exploration noise
  - Update critic by minimizing the loss
  - Update actor using the sampled policy gradient
  - Update target network softly
  - Understand the mechanism of actor-critic

Result

Solve LunarLander-v2 using DQN

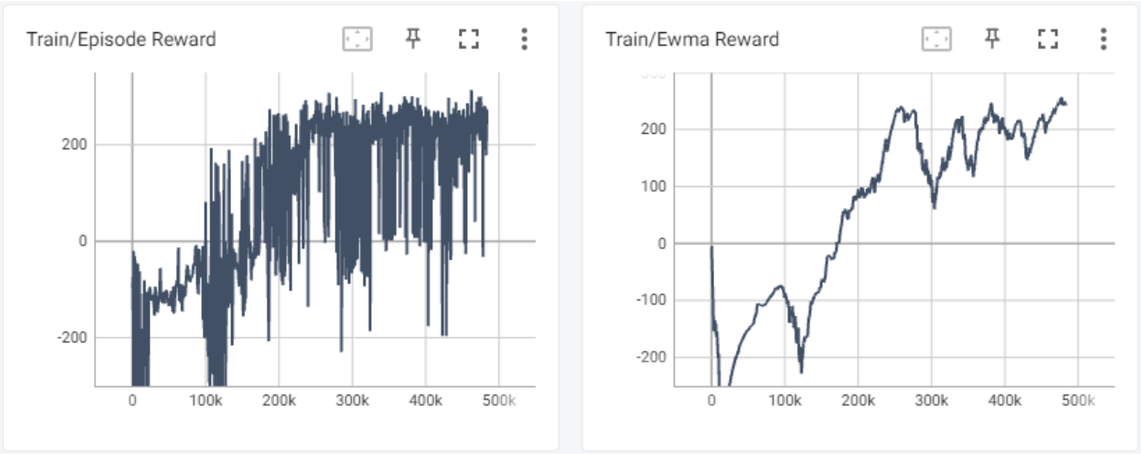
- Tensorboard

- Testing

```
PS C:\Users\MichaelLee\Desktop\NYCU\Deep Learning\Labs\Lab6\code> python .\dqn-example.py --test_only
C:\Users\MichaelLee\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWar
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode 1: 255.51223553395567
Episode 2: 258.1296164433185
Episode 3: 279.4680307714741
Episode 4: 267.73292317317043
Episode 5: 296.9268756801771
Episode 6: 246.68304317150796
Episode 7: 216.6991917657565
Episode 8: 259.1539662280297
Episode 9: 72.9977888850079
Episode 10: 226.9086844407501
Average Reward 238.02123560931483
```

Bonus: Implement DDQN

- Tensorboard

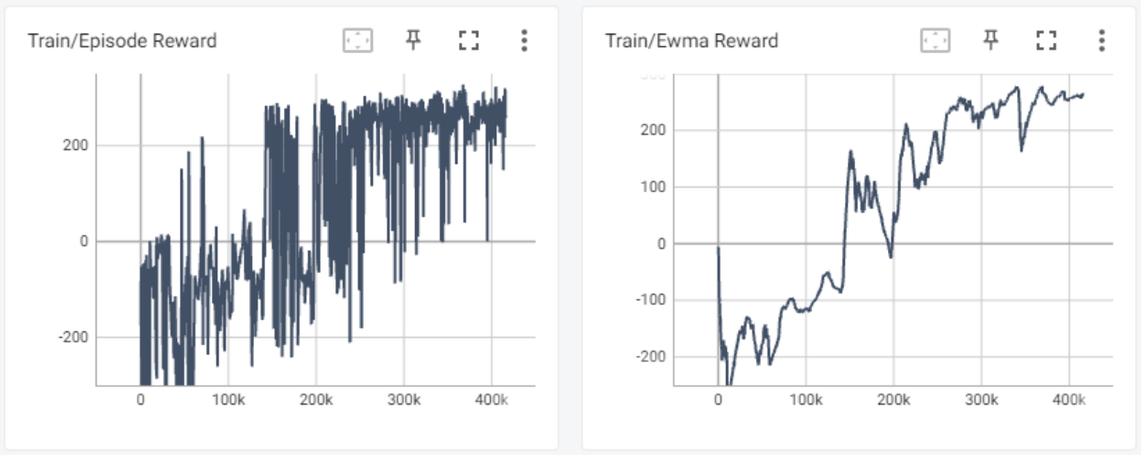


- Testing

```
PS C:\Users\MichaelLee\Desktop\NYCU\Deep Learning\Labs\Lab6\code> python .\double-dqn.py --test_only
C:\Users\MichaelLee\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWa
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode 1: 249.21381668362736
Episode 2: 211.01693425344666
Episode 3: 276.8116209325742
Episode 4: 247.44252174228387
Episode 5: 259.27862183089394
Episode 6: 246.89013299336216
Episode 7: 196.47380926241163
Episode 8: 273.711518185553
Episode 9: 300.7640247400824
Episode 10: 273.125387187638
Average Reward 253.47283878118728
```

Solve LunarLanderContinuous-v2 using DDPG

- Tensorboard



- Testing

```
PS C:\Users\MichaelLee\Desktop\NYCU\Deep Learning\Labs\Lab6\code> python .\ddpg-example.py --test_only
C:\Users\MichaelLee\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWarn
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
episode 1: 247.70
episode 2: 239.90
episode 3: 282.06
episode 4: 263.08
episode 5: 245.77
episode 6: 267.67
episode 7: 262.54
episode 8: 237.77
episode 9: 289.61
episode 10: 240.45
Average Reward 257.65491185909525
```

Bonus: Implement TD3

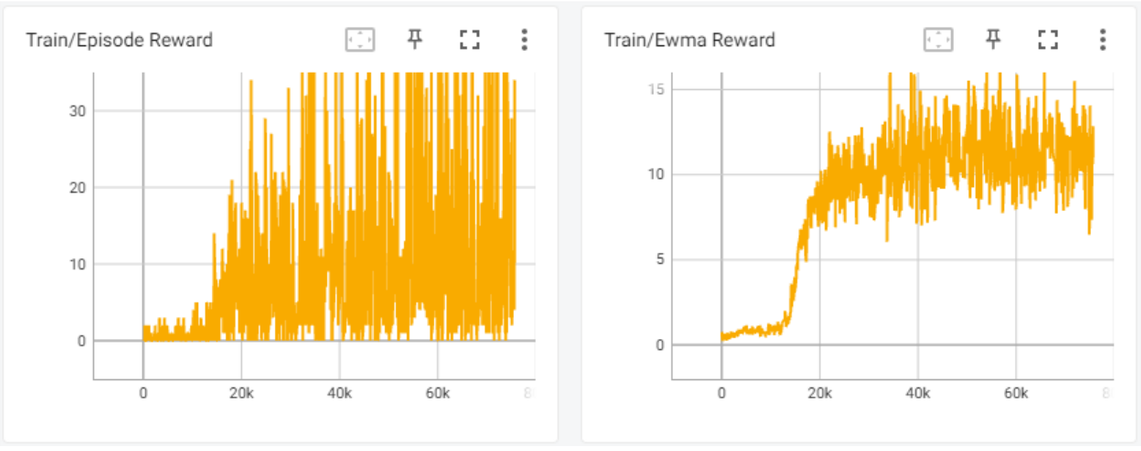
- Tensorboard

- Testing

```
PS C:\Users\MichaelLee\Desktop\NYCU\Deep Learning\Labs\Lab6\code> python .\td3-example.py --test_only
C:\Users\MichaelLee\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWar
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
episode 1: 253.58
episode 2: 289.06
episode 3: 280.88
episode 4: 283.52
episode 5: 304.17
episode 6: 254.72
episode 7: 299.45
episode 8: 297.86
episode 9: 185.98
episode 10: 286.92
Average Reward 273.61308632219897
```

Solve BreakoutNoFrameskip-v4 using DQN

- Tensorboard



- Testing

```
PS C:\Users\MichaelLee\Desktop\NYCU\Deep Learning\Labs\Lab6\code> python .\dqn_breakout_example.py --test_only
Start Testing
episode 1: 388.00
episode 2: 450.00
episode 3: 394.00
episode 4: 427.00
episode 5: 428.00
episode 6: 380.00
episode 7: 416.00
episode 8: 444.00
episode 9: 405.00
episode 10: 431.00
Average Reward: 416.30
```

Questions

Describe your major implementation of both DQN and DDPG in detail

1. **Your implementation of Q network updating in DQN**

   我設計了三層的 neural network，input 為 state (state_dim=8)，output 為 action (action_dim)，每一層之間有再加入 ReLU 做非線性的轉換。

```
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=32):
        super().__init__()
        ## TODO ##
        self.block1 = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
        )

        self.block2 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
        )

        self.block3 = nn.Sequential(
            nn.Linear(hidden_dim, action_dim),
        )

    def forward(self, x):
        ## TODO ##          x.shape = [batch_size, 8]
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        return x
```

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

   透過 _actor_net 去找出當前的 action a_t 是甚麼，希望計算出的 action 能夠讓拿到的 expected cumulative reward 也就是 $Q(s_t, a_t)$ 越大越好，因此就變成去 minimize 這個負的值，來更新 actor network。

```
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()
```

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

   先透過 target_actor_net 去找出 state s_{t+1} 的 action a_{t+1} 是甚麼，接著再透過 target_critic_net 找出 $Q(s_{t+1}, a_{t+1})$ 為何，再接著用 reward 以及找出來的 q value 去計算 q_target 是多少。最後透過 `nn.SmoothL1Loss()` 計算 q_value 與 q_target 的差異，去更新 critic network。

```
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 * (1.0 - done)

criterion = nn.SmoothL1Loss()
critic_loss = criterion(q_value, q_target)
# optimize critic
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

Explain effects of the discount factor

能夠讓 actor 考慮比較近期的 reward，比較遠的 reward 的關聯性與重要性就比較低。

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

能夠讓 agent 在選擇 action 的時候，會有部分的機率去做 exploration，對環境進行探索，而非像 greedy selection 每次都選擇 q value 最大的 action，有可能因此錯過找到更好 action 的機會。

Explain the necessity of the target network

因為在計算 q_value 與 q_target 中的 q_next_state 如果都用同一個 network 的話，當 network 的參數更新的時候，這兩個數值都會改變，就會讓整個過程變得不太穩定。因此在計算 q_target 的時候，我們會需要一個 freeze 的 target network 計算 q_next_state 的值，用來更新原本的 behavior network，過了一段時間 c 之後，再把 behavior network 更新給 target network，就可以讓訓練比較穩定。

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

- Reward 沒有除以 10 + Clip Reward = True

  因為每個 episode 獲得的 reward 都很小，如果再除以 10 就會讓計算 q value 的時候的值更小，因此認為說用原本拿到的 reward 直接去計算 q value 就好。接著因為 train 到後面有時候拿到的 reward 會忽大忽小，所以加上 clip reward 去讓訓練更穩定。

- Episode Life = True

  目的是要讓 agent 知道死掉是不好的行為，所以在訓練的時候要讓 agent 死掉就是結束一個 episode；而在 lunarlander 中如果 agent crash 掉就會直接結束並且得到負的 reward。

- Stack Frame = True

  因為是以圖片當作訓練資料，所以需要一次以連續四個 frame 當作 input，這樣才能夠知道球的移動方向是甚麼，讓 agent 更好的判斷目前的 state ，以做出比較好的 action；而在 lunarlander 中的輸入 (horizontal coordinate, vetical speed, …) 即為當前 agent 完整的 state。