DLP Lab6

Deep Q-Network and Deep Deterministic Policy Gradient

311551170 林琨堯

1. Introduction

   這次的 lab 要實作 DQN、DDPG 遊玩”LunarLander-v2”操控飛行器降落在旗幟中間。另外還需要訓練 DQN 玩” BreakoutNoFrameskip-v4” 盡可能擊破更多磚塊。

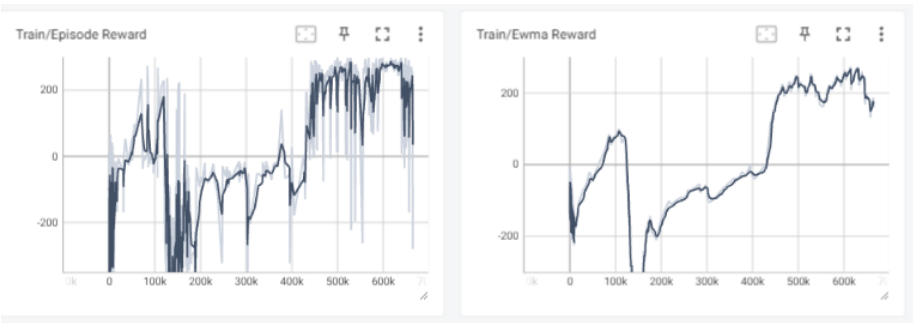
2. Experimental Results

   LunarLander-v2 with DQN

   (1) Testing results

```
Start Testing
total reward: 248.55
total reward: 283.99
total reward: 272.14
total reward: 276.44
total reward: 308.67
total reward: 258.36
total reward: 309.77
total reward: 256.60
total reward: 318.73
total reward: -56.18
Average Reward 247.70539762443636
```

   (2) Tensorboard

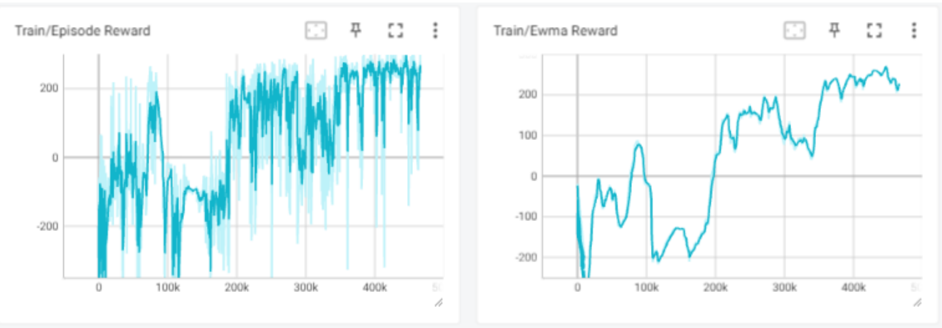


   LunarLander-v2 with DDPG

   (1) Testing results

```
Start Testing
Episode: 0      Length: 552     Total reward: 232.38
Episode: 1      Length: 146     Total reward: 250.05
Episode: 2      Length: 192     Total reward: 283.57
Episode: 3      Length: 215     Total reward: 267.59
Episode: 4      Length: 244     Total reward: 259.03
Episode: 5      Length: 197     Total reward: 267.65
Episode: 6      Length: 330     Total reward: 217.01
Episode: 7      Length: 920     Total reward: 285.75
Episode: 8      Length: 242     Total reward: 29.16
Episode: 9      Length: 282     Total reward: 264.76
Average Reward 235.69493596541474
```

(2) Tensorboard

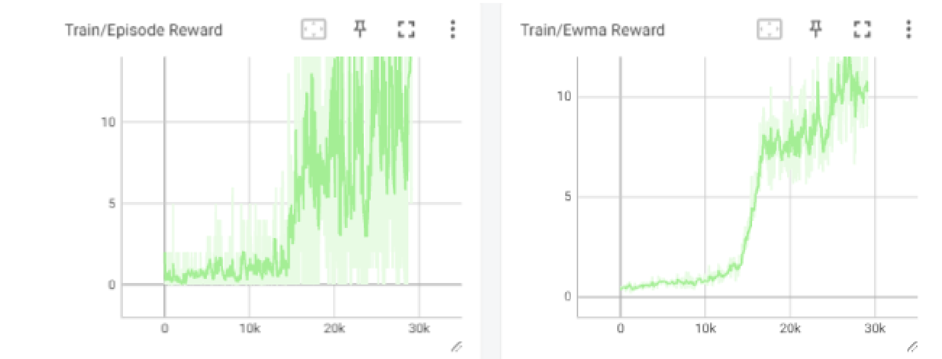


BreakoutNoFrameskip-v4 with DQN

(1) Testing results

```
Start Testing
episode 1: 350.00
episode 2: 435.00
episode 3: 428.00
episode 4: 428.00
episode 5: 417.00
episode 6: 382.00
episode 7: 424.00
episode 8: 412.00
episode 9: 418.00
episode 10: 361.00
Average Reward: 405.50
```

(2) Tensorboard



3. (Bonus) Experimental Results of LunarLander-v2 with DDQN

(1) Testing Results

```
Start Testing
Episode: 0      \Length: 259   Total Reward: 230.85
Episode: 1      \Length: 274   Total Reward: 270.93
Episode: 2      \Length: 281   Total Reward: 254.92
Episode: 3      \Length: 324   Total Reward: 254.14
Episode: 4      \Length: 248   Total Reward: 297.88
Episode: 5      \Length: 290   Total Reward: 251.08
Episode: 6      \Length: 357   Total Reward: 230.73
Episode: 7      \Length: 233   Total Reward: 275.54
Episode: 8      \Length: 344   Total Reward: 271.13
Episode: 9      \Length: 490   Total Reward: 233.52
Average Reward 257.0705101553232
```

(2) Tensorboard

4. (Bonus) Question

■ Describe your major implementation of both DQN and DDPG in detail

(1) DQN

Select action: 用 epsilon-greedy 選 action。當 random number 小於 epsilon，random 一個 action，否則使用 behavior net 選可以最大化 expected q value 的 action。

```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():
        state = torch.tensor(state, device=self.device).view(1, -1)
        outputs = self._behavior_net(state)
        _, best_action = torch.max(outputs, 1)
        return best_action.item()
```

Update behavior network: sample random minibatch of transition (ϕj , aj , rj , ϕj+1) from ReplayMemory. Compute target value y_j. Then use MSE as loss function.

```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)

    ## TODO ##
    q_value = self._behavior_net(state).gather(1, action.long())
    with torch.no_grad():
        q_next = torch.max(self._target_net(next_state), 1)[0].view(-1, 1)
        q_target = reward + q_next * gamma * (1.0 - 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()
```

Update target network: 複製 behavior net 的 weight 更新 target network。

```python
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) DDPG

ActorNet: 輸出 main engine 和 left-right engine 的 actions

```python
class ActorNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        ## TODO ##
        h1, h2 = hidden_dim
        self.fc1 = nn.Sequential(
            nn.Linear(state_dim, h1),
            nn.ReLU(inplace=True)
        )
        self.fc2 = nn.Sequential(
            nn.Linear(h1, h2),
            nn.ReLU(inplace=True)
        )
        self.fc3 = nn.Sequential(
            nn.Linear(h2, action_dim),
            nn.Tanh()
        )

    def forward(self, x):
        ## TODO ##
        x = self.fc1(x)
        x = self.fc2(x)
        out = self.fc3(x)
        return out
```

Select action: ActorNet 的 output 加入 noise 來生成 next action，可以獲得 exploitation 和 exploration 的影響。另外，只在 training 的時候加入 noise，testing 直接使用 ActorNet 的 output。

```
def select_action(self, state, noise=True):
    '''based on the behavior (actor) network and exploration noise'''
    ## TODO ##
    with torch.no_grad():
        state = torch.tensor(state, device=self.device).view(1, -1)
        outputs = self._actor_net(state)
        exploration_noise = torch.tensor(self._action_noise.sample(), device=self.device).view(1, -1)
        if noise:
            return (outputs + exploration_noise).squeeze(0).cpu().numpy()
        else:
            return outputs.squeeze(0).cpu().numpy()
```

Update behavior network: sample 和 loss 與 DQN 相同，update behavior actor network by $loss = \frac{-1}{N}\sum(\mathrm{Q}(\mathrm{si}\,,\mathrm{u}(\mathrm{si}))$

```
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 + gamma * q_next * (1 - done)
    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()
```

Update target network:

```
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 ##
        target.data.copy_(tau * behavior.data + (1 - tau) * target.data)
```

- Explain effects of the discount factor
  下圖是 Q learning algo., γ 是 discount factor. γ 越小，agent 越關注 current reward, 反之則越關注 future reward

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

- Explain benefits of epsilon-greedy in comparison to greedy action selection
  當前最好的選擇不一定對未來也是最好的選擇。為了讓模型探索其他動作，用 epsilon-greedy 設定一個機率 ε ，讓模型可以隨機選擇其他 action。
- Explain the necessity of the target network
  用 behavior q value 和 behavior target q value 計算 loss 可能會導致訓練

困難，所以需要固定從 behavior network 複製 weight 到 target network，用 target network 計算 loss

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

  用 atari_wrappers 把 4 張 frame 堆疊成一份，目的是檢測球的運動方向，以此計算 board 該如何移動，而 LunarLander 不需要的原因是，只需要知道當下 spaceship 與安全區之間的偏移量就可以知道要如何控制 spaceship。