DLP Lab6

310555008 曾信彥

A. A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLander-v2

(DDQN)

B. A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLanderContinuous-v2

(DDPG)

C. Describe your major implementation of both algorithms in detail.

1. DQN

(1) With probability ε select a random action a_t
otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

```
class DQN:
    def __init__(self, args): ...

    def select_action(self, state, epsilon, action_space):
        '''epsilon-greedy based on behavior network'''
        ## TODO ##
        if random.random() > epsilon:
            state = torch.from_numpy(state).float().unsqueeze(0).to(self.device)
            self._behavior_net.eval()
            with torch.no_grad():
                action_values = self._behavior_net(state)
            self._behavior_net.train()
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(action_space.n))
```

以上會固定 epsilon 在每次決定 action 時若 random 出比較大的值就會選擇從 behavior net 中取出最好的 action(也就是說 epsilon 的值對應到機率)

(2)

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

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

```
## TODO DQN##
if self.mth == 'DQN':
    q_value = self._behavior_net(state).gather(1,action.long())
    with torch.no_grad():
        q_next = self._target_net(next_state).detach().max(1)[0].unsqueeze(1)
        q_target = reward + (gamma * q_next * (1-done))
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)
```

以上會取得當下的 q value 和 target net 中的 q target 做 loss，並以此 loss 來更新 model

(3) Every C steps reset $\hat{Q} = Q$

```
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())
```

以上會根據 target_frequency 將 target net 中的參數定期換回 behavior net 中的參數

2. DDPG

(1)

Select action $a_t = \mu(s_t|\theta^{\mu}) + N_t$ according to the current policy and exploration noise

```
def select_action(self, state, noise=True):
    '''based on the behavior (actor) network and exploration noise'''
    ## TODO ##
    state = torch.from_numpy(state).float().to(self.device)

    self._actor_net.eval()
    with torch.no_grad():
        action = self._actor_net(state).cpu().data.numpy()
    self._actor_net.train()

    if noise:
        action += self._action_noise.sample()

    return action
```

以上會從 actor net 中取出 action 並加上 noise

(2)

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

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

```
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)
```

以上會從 critic net 中取出 q value，將從 target actor net 取得的 action 放進 target critic net 進而取得 q target 並計算 loss，以此 loss 更新 critic net

(3)

Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|s_i \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)|s_i$$

```
action = actor_net(state)
actor_loss = -critic_net(state, action).mean()
```

以上會取得當下 state 要採取的 action，並且放入 critic net 來取得 actor net 的 loss，要加負號是因為要讓他反向變化 parameter

(4)

Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$$

```
@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 ##
        target.data.copy_(tau * behavior.data + (1.0-tau)*target.data)
```

以上為更新 target network 的 method，會根據 tau 這個參數調整 target net 的參數

D. Describe differences between your implementation and algorithms.

```
def _soft_update_target_network(self, tau=.9):
    for target, behavior in zip(self._target_net.parameters(), self._behavior_net.parameters()):
        target.data.copy_(tau * behavior.data + (1.0 - tau) * target.data)
```

```
## TODO DDQN##
if self.mth == 'DDQN':
    q_value = self._behavior_net(state).gather(1,action.long())
    with torch.no_grad():
        q_argmax = self._behavior_net(next_state).detach().max(1)[1].unsqueeze(1)
        q_next = self._target_net(next_state).detach().gather(1,q_argmax)
        q_target = reward + (gamma * q_next * (1-done))
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)
```

跟原本的 DQN 不同，要 implement DDQN 會用到 soft_update 來小幅度更新 update target net

而下圖則是微調計算 loss 以更新 model 的 algorithm

E. Describe your implementation and the gradient of actor updating.
Actor updating
同 C 2.DDPG (3)
將 state 以及 action 放入 critic net 中並取負值以取得 loss

F. Describe your implementation and the gradient of critic updating.
同 C 2.DDPG (2)
從 critic net 中取得 q value，然後將取得自 target actor net 中的 action 放入 target critic net 而取得 q_next 和 q_target，以此計算 loss 並更新 critic net

G. Explain effects of the discount factor.
Discount factor 可以決定 model 要多注重 future，若 discount factor 越大代表要考慮越遠的 future，如果較小則考慮較近的 future

H. Explain benefits of epsilon-greedy in comparison to greedy action selection.
如果只採用 greedy action selection，也就是說每次都找最好的 action 會有可能 over all 找不到真正最好的，因為最好的 action 很可能在某一不看起來不是最好的甚至是很糟的 action，所以會使用 epsilon-greedy 的方式有時 random action，以達到 exploration 的功能

I. Explain the necessity of the target network.
如果有 target network 才可以根據過往經驗來看現在 behavior network 在嘗試的新 action，才不會讓 behavior network 反而往不好的方向走下去，藉此讓 behavior network 有修正的機會

J. Explain the effect of replay buffer size in case of too large or too small.
若 replay buffer 太小，model 只會考慮最近的 data，很可能 overfit。如果 replay buffer 太大，就會占用過多的 memory 空間並可能拖慢 training。

K. Implement and experiment on Double-DQN
主要分為三個部分
1. 計算 loss

```
## TODO DDQN##
if self.mth == 'DDQN':
    q_value = self._behavior_net(state).gather(1,action.long())
    with torch.no_grad():
        q_argmax = self._behavior_net(next_state).detach().max(1)[1].unsqueeze(1)
        q_next = self._target_net(next_state).detach().gather(1,q_argmax)
        q_target = reward + (gamma * q_next * (1-done))
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)
```

和 DQN 不同的是，會先經由 behavior net 取得 action 再代到 target net 取得 q_next，並取得 q_target

2.

```
# DDQN
elif(self.method == 'DDQN'):
    self._soft_update_target_network()
```

在 update 時使用 soft update

3.

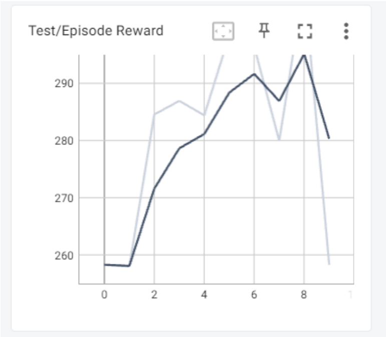
```
def _update_target_network(self):
    '''update target network by copying from behavior network'''
    ## TODO ##
    self._target_net.load_state_dict(self._behavior_net.state_dict())

def _soft_update_target_network(self, tau=.9):
    for target, behavior in zip(self._target_net.parameters(), self._behavior_net.parameters()):
        target.data.copy_(tau * behavior.data + (1.0 - tau) * target.data)
```

和一般的 update target network 不太一樣，會根據 tau 值來較低幅度的更新 target network

L. [LunarLander-v2] Average reward of 10 testing episodes

For DDPG,




```
Average Reward 281.248133813327 06
```

M. [LunarLanderContinuous-v2] Average reward of 10 testing episodes

For DDQN,


```
Average Reward 257.61390870379284
```