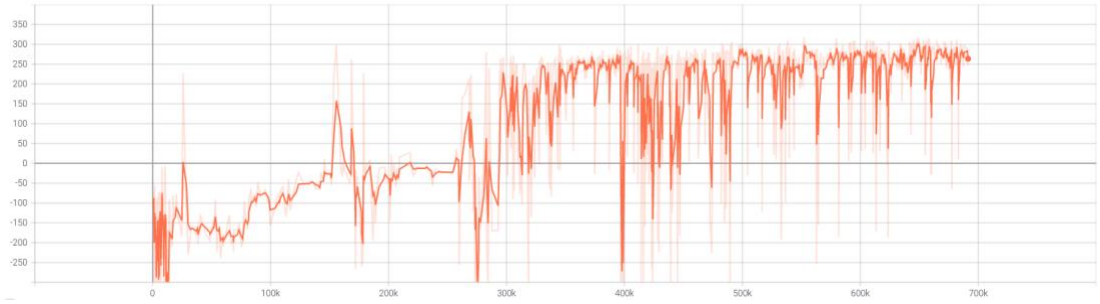
Lab6

機器人碩 309605014 王麒睿

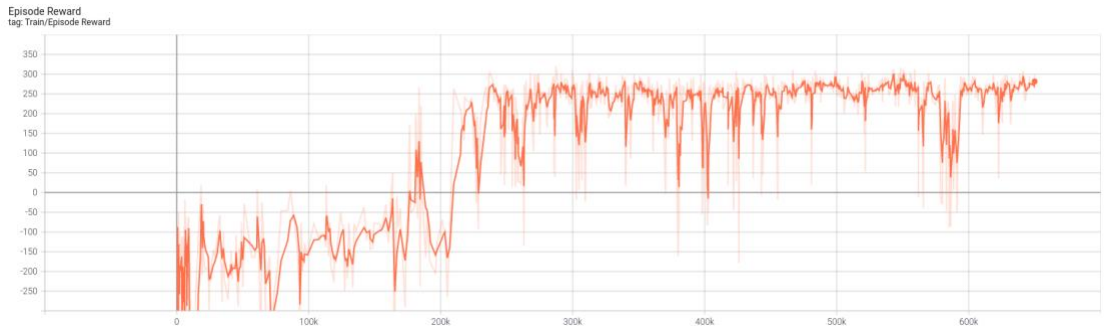
1.

LunarLander-v2 (action 離散) episode rewards



2.

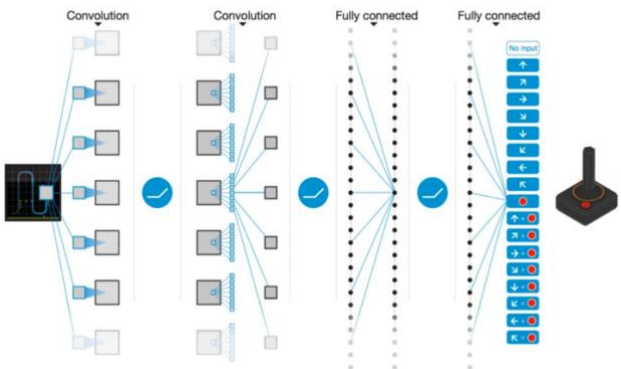
LunarLanderContinuous-v2 (action 連續) episode rewards



3. Describe your major implementation of both algorithms in detail

DQN:

建立一個 network 來預測 Q(s,a)的 value，這裡的 action4 有種可能(No-op,Fire left engine,Fire main engine,Fire right engine)，所以網路最後一層為 4 個 neuron



```python
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=(400,300)):
        super().__init__()
        ## TODO ##
        self.fc1=nn.Linear(state_dim,hidden_dim[0])
        self.fc2=nn.Linear(hidden_dim[0],hidden_dim[1])
        self.fc3=nn.Linear(hidden_dim[1],action_dim)
        self.relu=nn.ReLU()

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

在 episode 中(玩遊戲的過程中)，選擇最大 Q(s,ai)的 ai 或者有一定的機率 ε 隨機選擇 action (called ε-greedy)

```python
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
    ## TODO ##
    if random.random() < epsilon:  # explore
        return action_space.sample()
    else: # exploit
        with torch.no_grad():
            # t.max(1) will return largest column value of each row.
            # second column on max result is index of where max element was
            # found, so we pick action with the larger expected reward.
            return self._behavior_net(torch.from_numpy(state).view(1,-1).to(self.device)).max(dim=1)[1].item()
```

update network 的方法是由 replay memory 中 sampling 一些遊戲的過程: (state,action,reward,next_state,done)來做 td-learning，再用 Q(s,a)與 r + gamma * maxa'Q'(s',a')的差做 MSELoss

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

每隔一段時間，就用 behavior network 取代 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())
```

DDPG:

建立一個可以依據目前 state 決定要執行哪個 action 的 Actor Network, 由於 action 有 2 個(Main engine:-1~+1,Left-right-engine:-1~+1)，所以最後一層有 2 個 neuron

```
class ActorNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        ## TODO ##
        self.fc1=nn.Linear(state_dim,hidden_dim[0])
        self.fc2=nn.Linear(hidden_dim[0],hidden_dim[1])
        self.fc3=nn.Linear(hidden_dim[1],action_dim)
        self.relu=nn.ReLU()
        self.tanh=nn.Tanh()

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

建立一個可以預估 Q(s,a)的 Critic Network，由於輸出的是一個純量，所以最後一層 neuron 數 為 1

```
class CriticNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        h1, h2 = hidden_dim
        self.critic_head = nn.Sequential(
            nn.Linear(state_dim + action_dim, h1),
            nn.ReLU(),
        )
        self.critic = nn.Sequential(
            nn.Linear(h1, h2),
            nn.ReLU(),
            nn.Linear(h2, 1),
        )

    def forward(self, x, action):
        x = self.critic_head(torch.cat([x, action], dim=1))
        return self.critic(x)
```

在 episode 中(玩遊戲的過程中)，由 Actor Network 選擇 action 並加上一 noise

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

在 episode 中(玩遊戲的過程中)，也要更新 Behavior 的 Actor Network μ, Critic Network Q, Target 的 Actor Network μ', Critic Network Q'。

利用 Target Network 生出的 q_target 與 Behavior Network 生出的 q_value 做 MSELoss 更新 Q。

```
## update critic ##
# critic loss
## TODO ##
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()
critic_loss = criterion(q_value, q_target)

# bp
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

利用 Behavior Network 的 Actor Network μ 與 Critic Network Q 可以求出 Q(s,a)，我們想要更新 μ 來使輸出的 Q(s,a)越大越好，因此定義
Loss Value = E[-Q(s, μ(s))]，並透過 backpropagation 更新。

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

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

4.

Describe differences between your implementation and algorithms

在 training 的時候，一開始會有一段 warmup 的時間，在這段時間中，不會去 update network 的 參數，只會隨便亂玩(隨機選擇 action)，並把遊戲過儲存到 replay memory 裡。另外在 DQN 的 部份，並不是每個 iteration 都要更新 Behavior Network，而是每隔一段時間(Ex: 4 個 iteration)才 會更新一次。

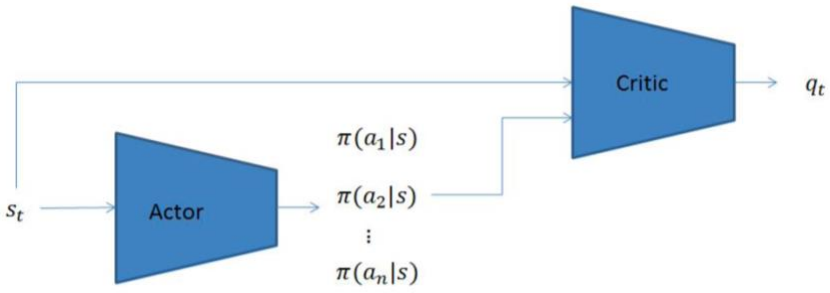
5.

Describe your implementation and the gradient of actor updating in DDPG

利用 Behavior Network 的 Actor Network μ 與 Critic Network Q 可以求出 Q(s,a)，
我們想更新 Actor Network μ 來使輸出的 Q(s,a)越大越好，
因此定義 Loss Value = -Q(s, μ(s))，backpropagation 的時候不更新 Critic、只更新 Actor。

$$L = -Q(s, a|\theta_Q),\ a = u(s|\theta_u)$$

$$\frac{\nabla L}{\nabla \theta_u} = -\frac{\nabla Q(s,a|\theta_Q)}{\nabla a}\frac{\nabla a}{\nabla u(s|\theta_u)}\frac{\nabla u(s|\theta_u)}{\nabla \theta_u}$$
$$= -\frac{\nabla Q(s,a|\theta_Q)}{\nabla u(s|\theta_u)}\frac{\nabla u(s|\theta_u)}{\nabla \theta_u}$$




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

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

6.

Describe your implementation and the gradient of critic updating in DDPG

利用 Target Network 生出的 Q_{target} 與 Behavior Network 生出的 Q(s,a)做 mean square error 來更 新 Q Network。

```
## update critic ##
# critic loss
## TODO ##
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()
critic_loss = criterion(q_value, q_target)

# bp
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

7. Explain effects of the discount factor

$$G_t = R_{t+1} + \lambda R_{t+2} + \ldots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

λ 就是 discount factor，意思就是說，越是未來所給的 reward 影響是越來越小的，當下的 reward 是最大的

8.

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

我們必須在 explore 與 exploit 之間取得平衡，因此在 greedily choosing action 的基礎上，必須偶爾選擇其他的 action 來 explore 那些未知但可能是最佳的 action。

9. Explain the necessity of the target network

有 Target Network 與 Behavior Network 的搭配可以使 training 的時候更穩定， 因為生出 Q_target 的 Target Network 每隔一段時間才會改變一次。

10.

Explain the effect of replay buffer size in case of too large or too small

如果 replay buffer size 越大，training 過程可以更穩定，但會降低 training 的速度。如果 replay buffer size 越小，會一直著重於最近玩的 episode 的狀況，容易造成 overfitting、甚至整個 train 壞掉。

Bonus

1. Implement and experiment on Double-DQN

DDQN 與 DQN 其實差不多，就只差在 update Behavior Network 時是如何決定 q_target 的， DDQN 在決定 q_target 時，不是直接取 max Q'(s,ai)，而是用 Q(s,ai)中最大值的 i 作為 t 查找 Q'(s,ai)的 index。

```
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(dim=1,index=action.long())
    with torch.no_grad():
        action_index=self._behavior_net(next_state).max(dim=1)[1].view(-1,1)
        q_next = self._target_net(next_state).gather(dim=1,index=action_index.long())
        q_target = reward + gamma*q_next*(1-done)
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)

    # bp
    self._optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
    self._optimizer.step()
```

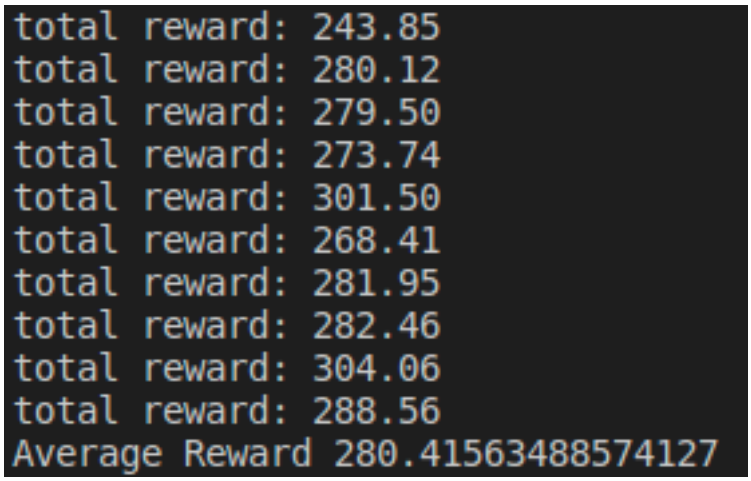
train 出來的結果跟 DQN 差不多。(train 2000 個 episode)

```
total reward: 254.97
total reward: 274.73
total reward: 284.31
total reward: 286.18
total reward: 52.09
total reward: 274.79
total reward: 311.56
total reward: 289.20
total reward: 318.51
total reward: 300.98
Average Reward 264.7320360204013
```

Performance

1.

[LunarLander-v2] Average reward of 10 testing episodes: Average ÷ 30 DQN: 2000 episode



2.

[LunarLanderContinuous-v2] Average reward of 10 testing episodes: Average ÷ 30 DDPG: 2000 episode