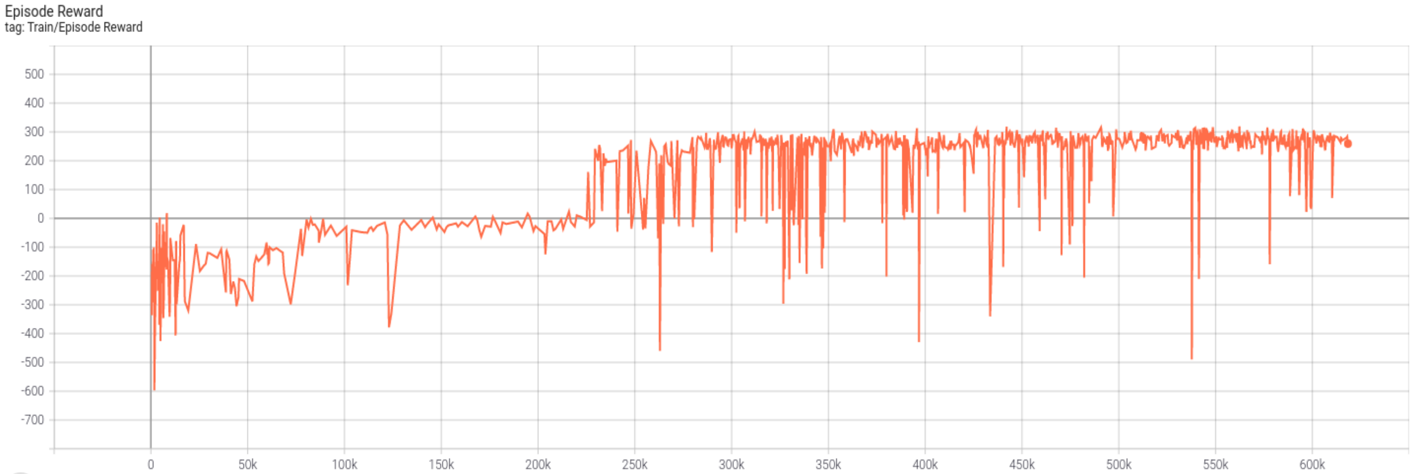
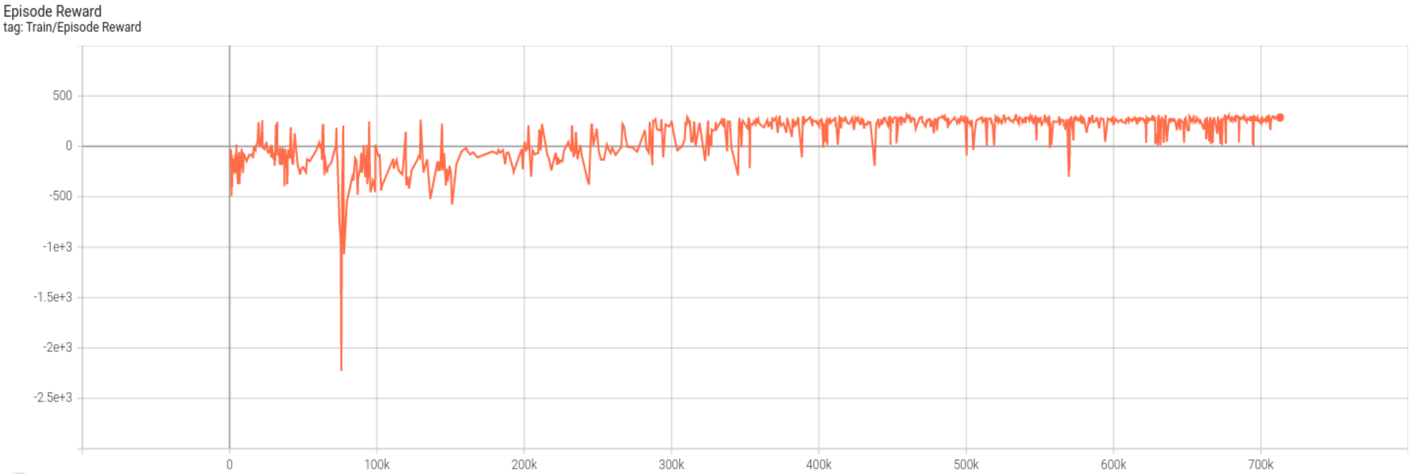
DLP Lab6 Deep Q-Network and Deep Deterministic Policy Gradient

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



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



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

- DQN

  這裡的 action 有 4 種可能 (no-op, fire left engine, fire main engine, fire right engine)，所以 network 最後一層為 4 個 neuron。

  - Input: an 8-dimension observation (not an image)
  - First layer: fully connected layer (ReLU)
    - input: 8, output: 400
  - Second layer: fully connected layer (ReLU)
    - input: 400, output: 400
  - Third layer: fully connected layer
    - input: 400, output: 4

```
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=400):
        super().__init__()
        ## TODO ##
        self.fc1 = nn.Linear(state_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, 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，選擇最大 Q(s, a_i)的 a_i 或有一定的機率 epsilon 隨機選擇 action，我們稱為 epsilon-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():
            return self._behavior_net(torch.from_numpy(state).view(1,-1).to(self.device)).max(dim=1)[1].item()
```

在 update behavior network 中，由 replay memory 去 sample 一些遊戲過程(state, action, reward, next_state, done)來做 TD-learning，再用 Q(s, a)與 r + gamma * $\max_{a'}$Q'(s', a')的差做 MSELoss。

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

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

```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()
```

在 update target network 中，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
  建一個可以依據目前 state 決定要執行哪個 action 的 actor network，由於 action 有兩種(main engine:-1 ~ +1, left right engine: -1 ~ +1)，所以最後一層有 2 個 neuron。

```python
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
```

建一個可以預估 Q(s, a)的 critic network，由於輸出的是純量，所以最後一層 neuron 數為 1。

```python
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，由 actor network 選擇 action 並加上一個 noise。

```python
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，利用 target network 生出的 q_target 與 behavior network 生出的 q_value 做 MSELoss，然後更新 Q。

```python
# 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 = 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)

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

利用 behavior network 的 actor network mu 和 crttic network Q 可以求出 Q(s, a)，想要更新 mu 來使輸出 Q(s, a)愈大愈好，因此定義 Loss value = E[-Q(s, mu(s))]，並利用 backpropagation 更新。

```python
## update actor ##
# actor loss
## TODO ##
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()
```

4. Describe differences between your implementation and algorithms.

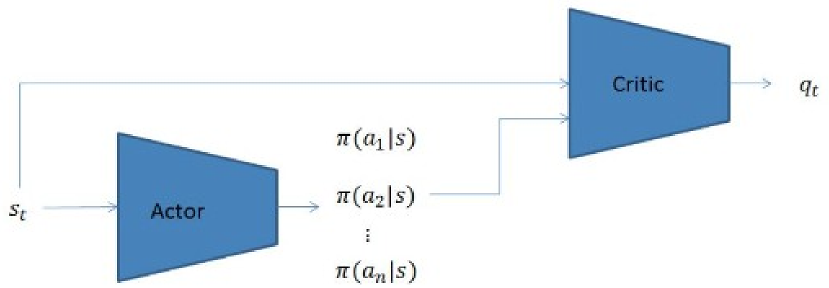
在一開始 training 時，會有一段暖身時間，在這段時間裡，不會去 update network 的參數，只會隨機選擇 action，並把遊玩過程儲存到 replay memory 中。在 DQN，並不是每個 iteration 都要更新 behavior network，而是一段時間才更新一次(ex: 4 個 iteration)。

5. Describe your implementation and the gradient of actor updating.

透過 behavior network 中的 actor network mu 和 critic network Q 可以求出 Q(s, a)，想更新 actor network mu 來使輸出的 Q(s, a)愈大愈好，因此定義 Loss value = -Q(s, mu(s))，然後在 backpropagation 時不更新 critic，只更新 actor。

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

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




```
## update actor ##
# actor loss
## TODO ##
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()
```

6. Describe your implementation and the gradient of critic updating.

利用 target network 生出的 Q_{target} 與 behavior network 生出的 Q(s, a)做 MSE 來更新 Q network。

$$L = \frac{1}{N}\sum(Q_{target} - Q(s_t, a_t|\theta_Q))^2$$

```
# 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 = 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)

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

7. Explain effects of the discount factor.

epsilon 就是 discount factor，也就是，當愈未來所給 reward 影響是愈小的，當下的 reward 影響是最大的。

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

8. Explain benefits of epsilon-greedy in comparison to greedy action selection.
為了在 explore 和 exploit 之間取得平衡，因此在 greedily 選擇 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.
target network 與 behavior network 的搭配下，可以使 training 時更穩定，因為生出的 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.
若 replay buffer size 愈大，在 training 過程中可以更穩定，但同時會降低 training 的速度。但若 replay buffer size 愈小，則會一直著重在最近玩的 episode 上，容易造成 overfitting，甚至整個 train 壞。

Performance :

- DQN
  train 了 2000 個 episode

```
Start Testing
total reward: 246.41
total reward: 280.55
total reward: 283.94
total reward: 276.47
total reward: 295.84
total reward: 272.12
total reward: 302.61
total reward: 292.09
total reward: 306.86
total reward: 304.05
Average Reward 286.09438894059195
```

- DDPG
  train 了 2000 個 episode

```
Start Testing
total reward: 248.58
total reward: 281.11
total reward: 282.11
total reward: 265.43
total reward: 275.96
total reward: 274.45
total reward: 270.45
total reward: 292.30
total reward: 298.97
total reward: 303.05
Average Reward 279.2414680082608
```