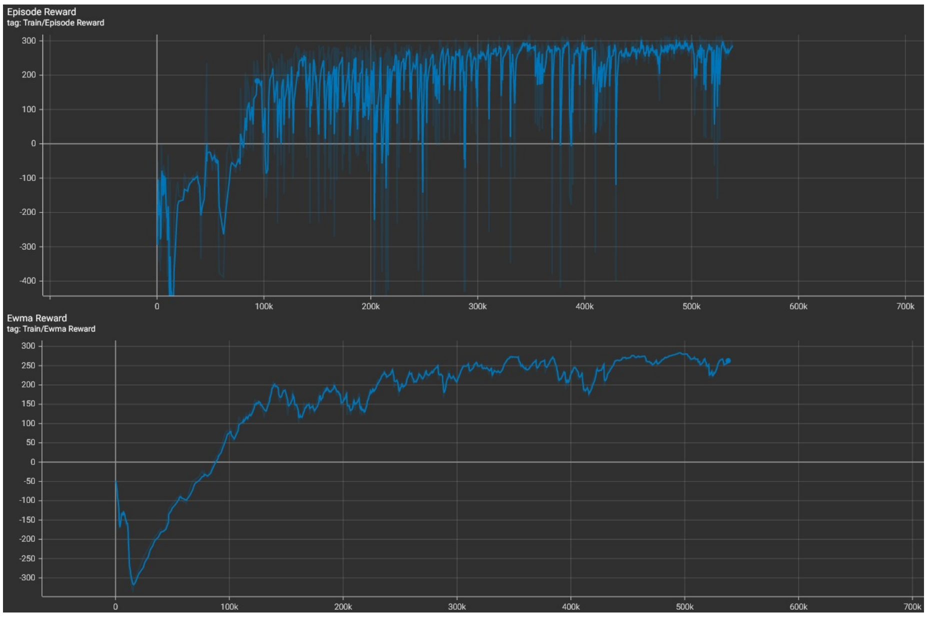
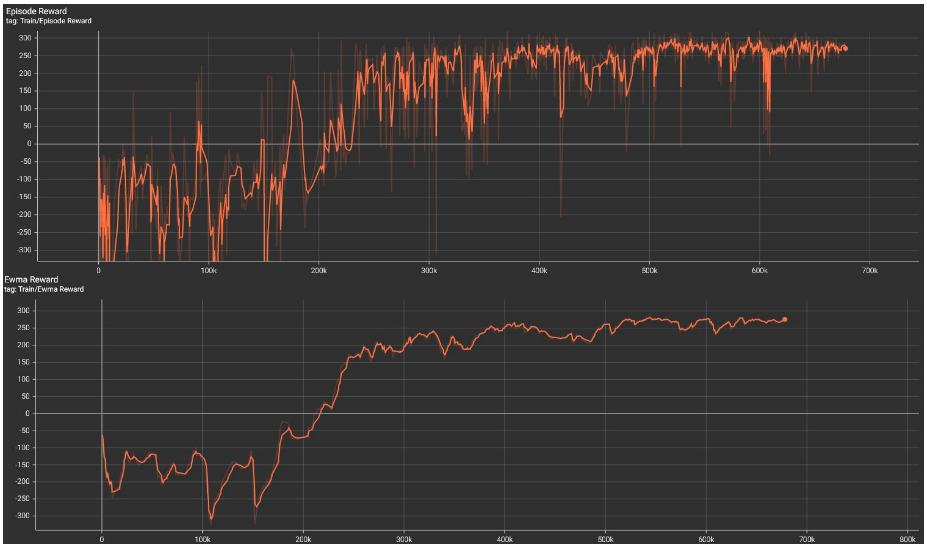
DL Lab6 report - Deep Q-Network and Deep Deterministic Policy Gradient

匯出pdf之後版面變得有點醜QQ
助教不介意的話可以直接到HackMD網址看 感謝～～～
https://hackmd.io/@ZdXM6gDQSTGTtZGr5jTo4Q/BJF0f2jCc (https://hackmd.io/@ZdXM6gDQSTGTtZGr5jTo4Q/BJF0f2jCc)

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



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



Describe your major implementation of both algorithms in detail

- DQN

```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
```

建立一個network來預測Q(s, a)的value，因為action有四種，所以最後一層為四個neuron。

```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()
```

在遊戲過程中，選擇最大Q(s, ai)的ai，或有一定的機率ε隨機選擇action。

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

由replay memory中sampling一些遊戲的過程來做td-learning，再做MSELoss。

```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())
```

每隔一段時間，就用behavior network取代target network。

- DDPG

```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
```

建立一個根據目前state來判斷要執行的action的network，因為有兩種action，所以最後一層是兩個neuron。

```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)
```

建立一個可以預測Q(s, a)的network，因為輸出是一個純量，所以最後一層是一個neuron。

```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()
```

在遊戲過程中，由actor network選擇action然後加上noise。

```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()
```

在遊戲過程中，也要更新behavior的actor network μ跟critic network Q，還有target的actor network μ'跟critic network Q'。再利用target network產生的q_target跟behavior network產生的q_value做MSELoss。

```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()
```

利用behavior network的actor network μ跟critic network Q求出Q(s, a)，並且希望更新μ
來使輸出的Q(s, a)越大越好。

Describe differences between your implementation and algorithms

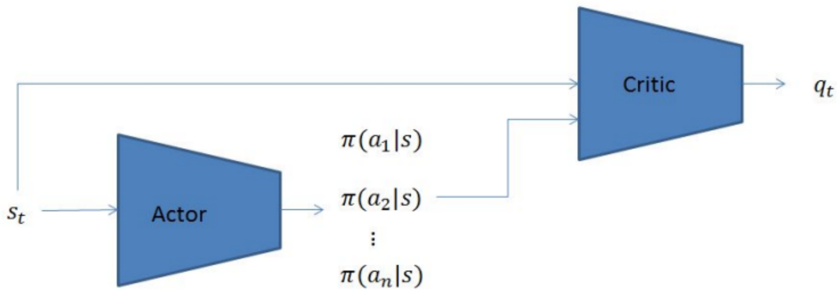
在training的時候，最初有一段warmup的時間不會update network的參數，只會隨機選擇action，並把遊戲過程儲存到replay memory裡。DQN的部份，並不是每個iteration都會更新behavior network，而是每隔幾個iteration才會更新一次。

Describe your implementation and the gradient of actor updating

利用behavior network的actor network μ與critic network Q可以求出Q(s,a)，利用更新
actor Network μ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。

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

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



Describe your implementation and the gradient of critic updating

利用target network生出的Qtarget與behavior network生出的Q(s, a)做MSE來更新Q Network。

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

Explain effects of the discount factor

λ就是discount factor，越以後的reward影響是越來越小的，當下的reward是最大的。

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

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

要在explore與exploit之間取得平衡，所以在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。

Explain the necessity of the target network

target network與behavior network的搭配可以使training更穩定，因為產生Q_target的target network每隔一段時間才會改變一次。

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

buffer size越大，training過程可以越穩定，但會降低

training的速度。buffer size越小，會著重於越近的episode，容易造成overfitting，甚至整個training效果差勁。

Result

- DQN

```
  "You are calling render method, "
total reward: 250.62
total reward: 244.73
total reward: 266.93
total reward: 279.94
total reward: 316.87
total reward: 265.40
total reward: 280.07
total reward: 309.63
total reward: 259.00
total reward: 267.24
Average Reward 274.04184706796457
```

  episode = 2000

  random seed = 2

- DDPG

```
  "You are calling render method, "
total reward: 268.98
total reward: 280.29
total reward: 304.10
total reward: 269.25
total reward: 283.17
total reward: 303.86
total reward: 262.77
total reward: 273.50
total reward: 280.24
total reward: 165.02
Average Reward 269.1167979251321
```

  episode = 2000

  random seed = 4