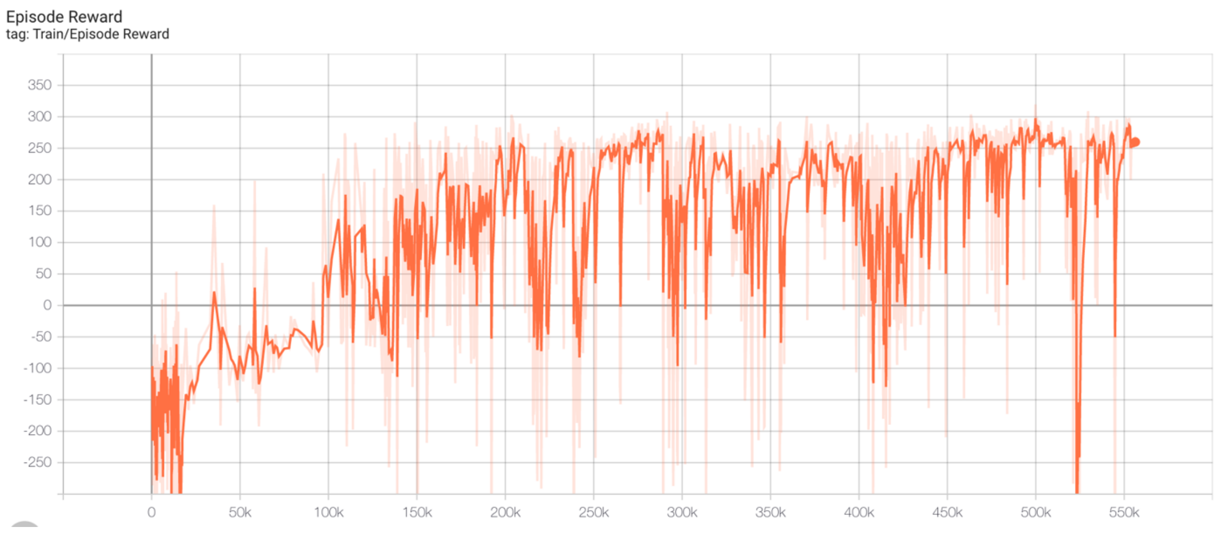
DLP_LAB6_309551124_涂景智

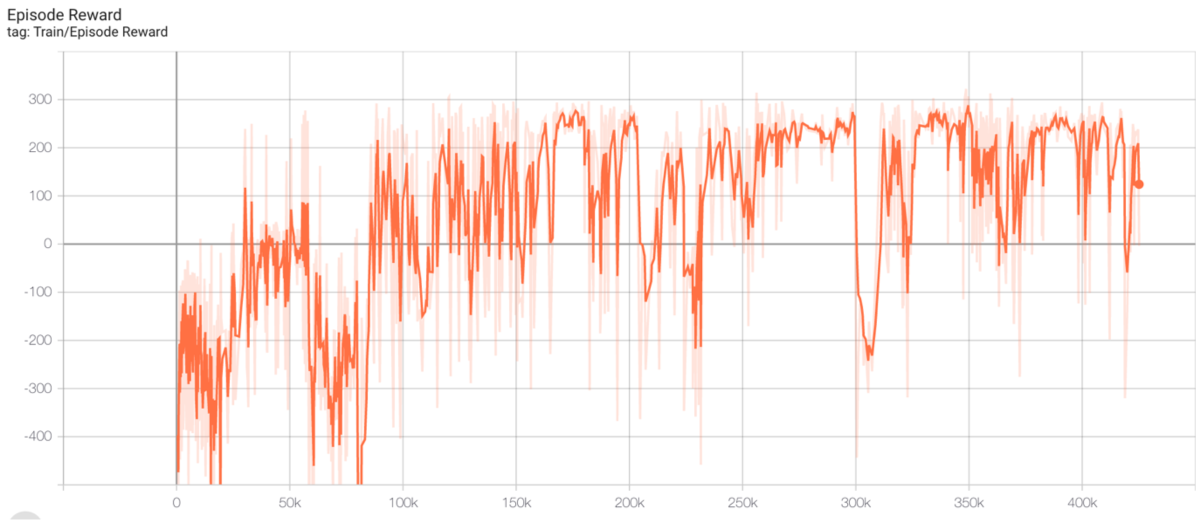
◆ Report

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



上圖是 Training 1500 Episode 的結果。縱軸是 reward，橫軸是 Step 數

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

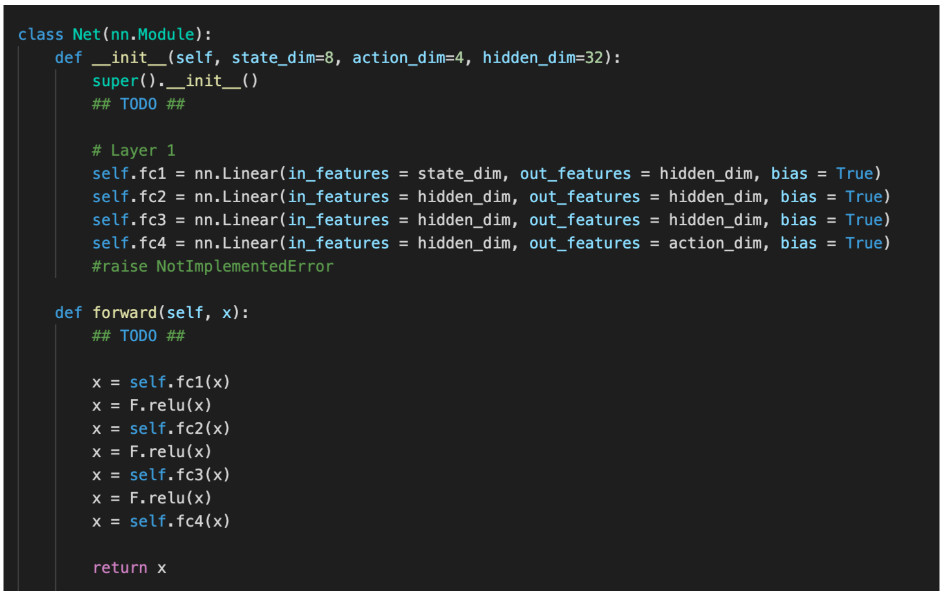


上圖是 Training 1500 Episode 的結果。縱軸是 reward，橫軸是 Step 數

- Describe your major implementation of both algorithms in detail.

  - DQN:

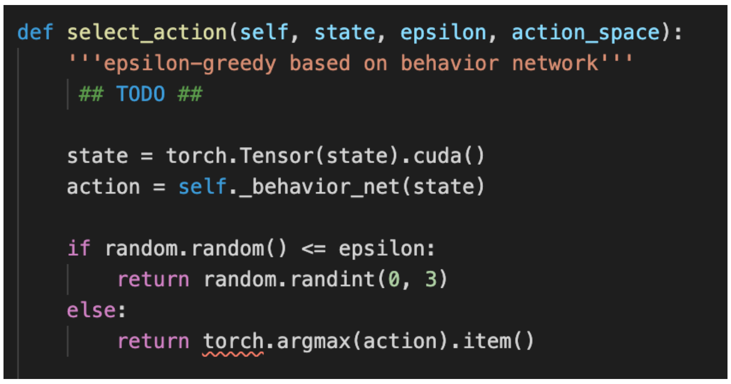
    NET(DQN 有二個 net，Behavior Net 與 Target Net) 的結構如下圖：



```python
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=32):
        super().__init__()
        ## TODO ##

        # Layer 1
        self.fc1 = nn.Linear(in_features = state_dim, out_features = hidden_dim, bias = True)
        self.fc2 = nn.Linear(in_features = hidden_dim, out_features = hidden_dim, bias = True)
        self.fc3 = nn.Linear(in_features = hidden_dim, out_features = hidden_dim, bias = True)
        self.fc4 = nn.Linear(in_features = hidden_dim, out_features = action_dim, bias = True)
        #raise NotImplementedError

    def forward(self, x):
        ## TODO ##

        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.fc4(x)

        return x
```

無論是 Behavior Net 還是 Target Net，架構都很簡單。都是四層 fully connected layer。

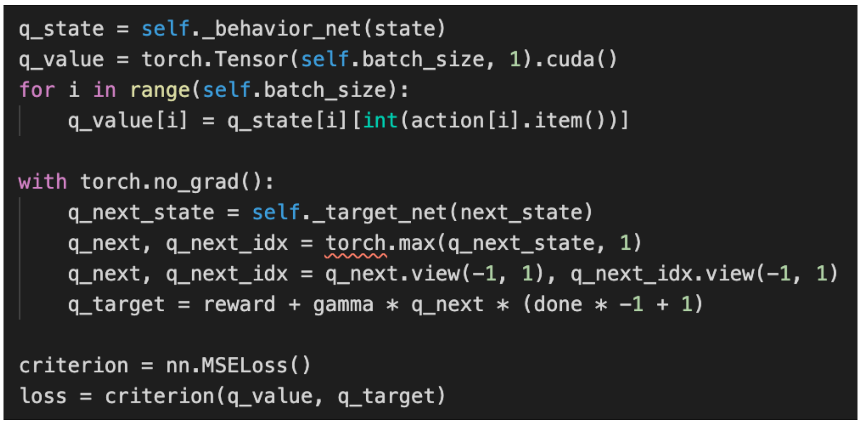
另外，這次 LAB 還需要時做 select_action()：



```python
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
    ## TODO ##

    state = torch.Tensor(state).cuda()
    action = self._behavior_net(state)

    if random.random() <= epsilon:
        return random.randint(0, 3)
    else:
        return torch.argmax(action).item()
```

這樣的實作代表當我們的 model 有 epsilon 的機率做出隨機的 action，這樣才有機會學習到新的經驗。

　　最後，這次 的 LAB 在 DQN 中比較重要的應該是我們還需要實作 Behavior Net 的權重更新：

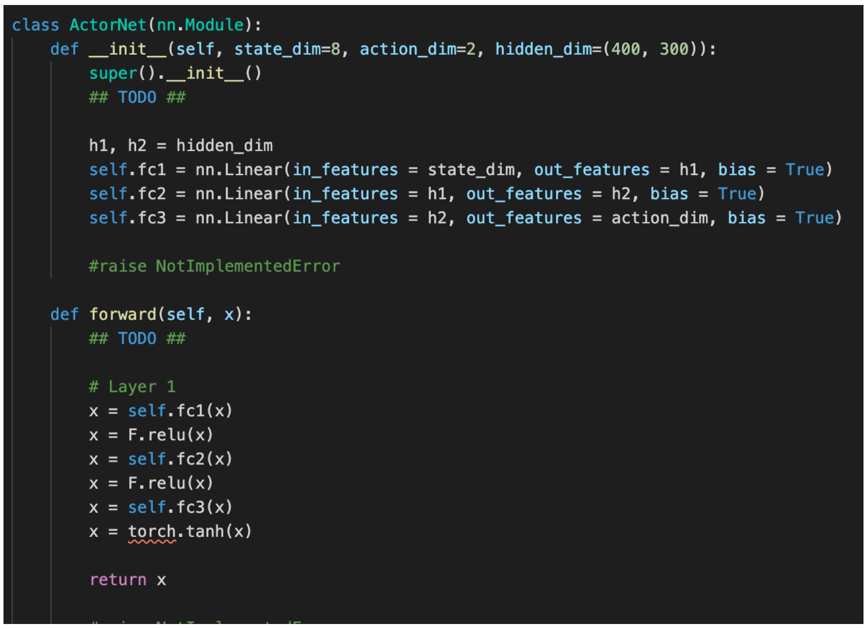


```
q_state = self._behavior_net(state)
q_value = torch.Tensor(self.batch_size, 1).cuda()
for i in range(self.batch_size):
    q_value[i] = q_state[i][int(action[i].item())]

with torch.no_grad():
    q_next_state = self._target_net(next_state)
    q_next, q_next_idx = torch.max(q_next_state, 1)
    q_next, q_next_idx = q_next.view(-1, 1), q_next_idx.view(-1, 1)
    q_target = reward + gamma * q_next * (done * -1 + 1)

criterion = nn.MSELoss()
loss = criterion(q_value, q_target)
```

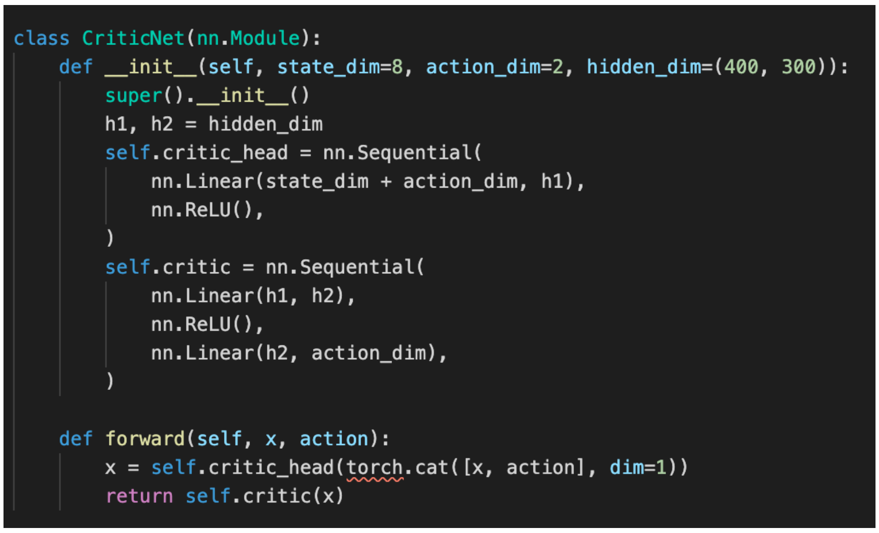
　　大致上就是，根據 Target Net 回傳的 Target Q-value，來與我們 Behavior Net 輸出的 Q-value 計算 Loss，並藉此更新 Behavior Net。

◆ DDPG

DDPG 主要包含 Actor 以及 Critic 組成。實作如下圖：



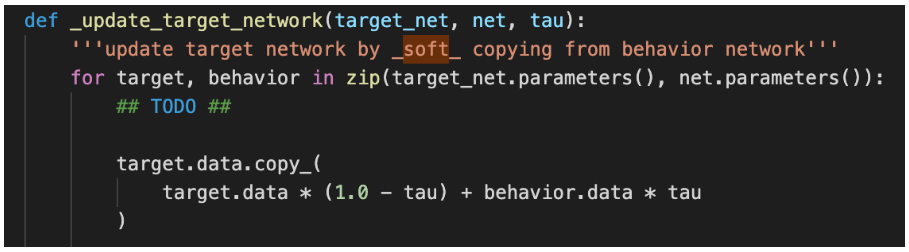
```
class ActorNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        ## TODO ##

        h1, h2 = hidden_dim
        self.fc1 = nn.Linear(in_features = state_dim, out_features = h1, bias = True)
        self.fc2 = nn.Linear(in_features = h1, out_features = h2, bias = True)
        self.fc3 = nn.Linear(in_features = h2, out_features = action_dim, bias = True)

        #raise NotImplementedError

    def forward(self, x):
        ## TODO ##

        # Layer 1
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = torch.tanh(x)

        return x
```



```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, action_dim),
        )

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

Actor Net 以及 Critic Net 的 Architecture (例如 hidden layer 的層數)都是根據助教提供的實作。

另外，與前面 DQN 不同的是，此次的 DDPG 是使用 soft update。每個 step 都會更新 target network，但每次都只更新一點點：



```python
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_(
            target.data * (1.0 - tau) + behavior.data * tau
        )
```

圖中的 tau 為 behavior Network 佔更新後 target network 的權重，預設為 0.005。

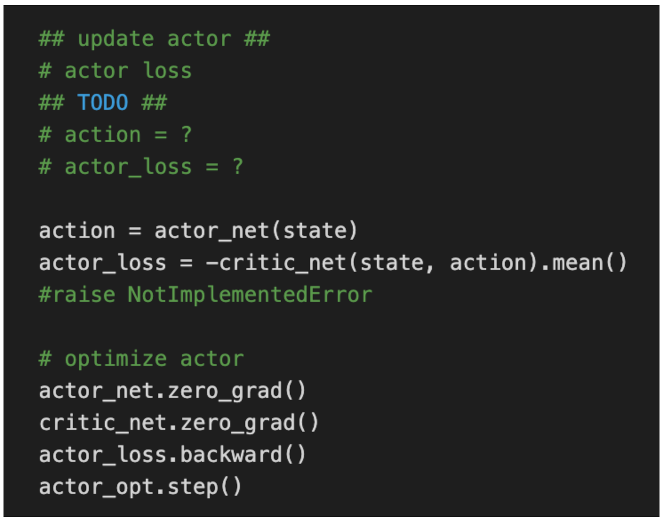
至於 actor 和 critic 的 gradient 的更新方式，會在後面的問題中介紹。

■ Describe your implementation and the gradient of actor updating.

actor network 的更新是採用所謂的 Policy Gradient，其 Gradient 應為：

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

程式的實作：



```
## update actor ##
# actor loss
## TODO ##
# action = ?
# actor_loss = ?

action = actor_net(state)
actor_loss = -critic_net(state, action).mean()
#raise NotImplementedError

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

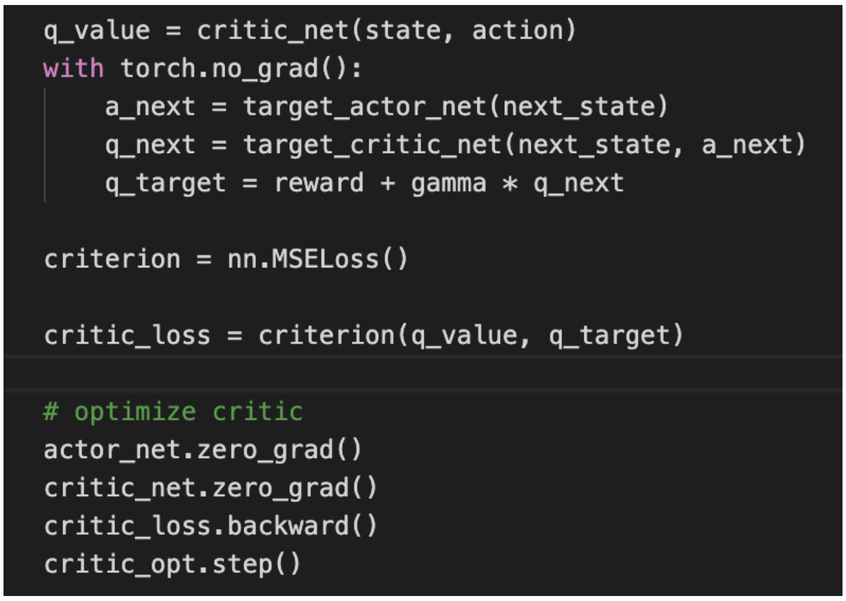
- Describe your implementation and the gradient of critic updating.

critic 的網路更新公式應如下：

$$target_t = R_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1}; \theta^-); \omega^-)$$

$$Loss = \frac{1}{N}\sum_{t=1}^{N}(target_t - Q(S_t, a_t; \omega))^2$$

程式的實作如下：



```
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

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()
```

其實 critic 的更新方式與 DQN 滿類似的。

■ Explain effects of the discount factor.

在 RL 中，定義目前策略或狀態的好壞的方式通常是用「這個狀態對未來的期望」來定義。如下圖：

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

Future Reward

式子中的 Lambda 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。另外，discount factor 越小，也代表未來 reward 的影響會越小

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

如果單純用 greedy action selection，則 model 都只會選擇當下最有利的選擇。然而，當下最有利的選擇並不一定代表對未來最有利的選擇，因此我們需要用 epsilon-greedy，使 model 有一定的機率隨機選擇要做的 action，如此才能做出新的嘗試。

■ Explain the necessity of the target network.

target network 的目的在於使整個 Deep Q-Learning 的訓練穩定性提高。其中 target network 久久更新一次，更新時直接把 behavior network 的參數整組複製過來。若沒有 target network，整個訓練會無法趨於穩定。

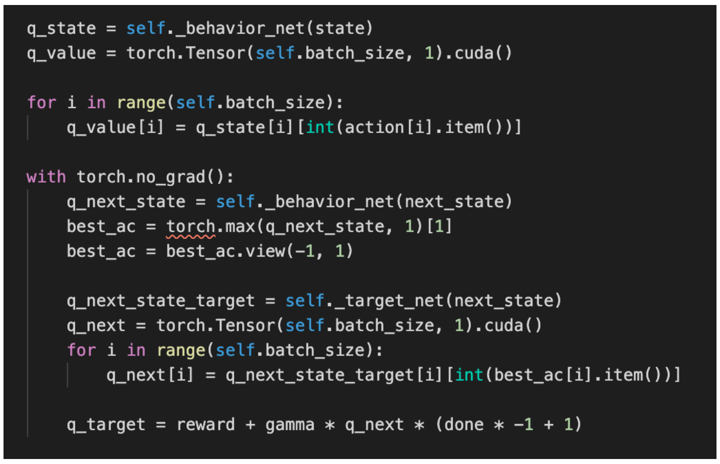
■ Explain the effect of replay buffer size in case of too large or too small.

replay buffer 的機制可以使 model 將 experience 存在 memory 中，訓練時隨機從中抽樣。若 replay buffer size too large ，則浪費記憶體空間且可能使訓練速度變慢。若 replay buffer size too small，我們訓練的 model 可能會 too overfit to recent data。

◆ Report Bonus

■ Implement and experiment on Double-DQN

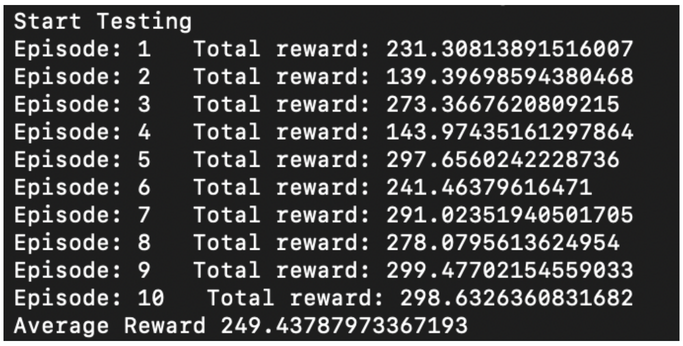
在此次的 LAB 中，我也有實作了 Double-DQN。與 DQN 實作最大的差異，是要 behavior network 的更新：



```
q_state = self._behavior_net(state)
q_value = torch.Tensor(self.batch_size, 1).cuda()

for i in range(self.batch_size):
    q_value[i] = q_state[i][int(action[i].item())]

with torch.no_grad():
    q_next_state = self._behavior_net(next_state)
    best_ac = torch.max(q_next_state, 1)[1]
    best_ac = best_ac.view(-1, 1)

    q_next_state_target = self._target_net(next_state)
    q_next = torch.Tensor(self.batch_size, 1).cuda()
    for i in range(self.batch_size):
        q_next[i] = q_next_state_target[i][int(best_ac[i].item())]

    q_target = reward + gamma * q_next * (done * -1 + 1)
```

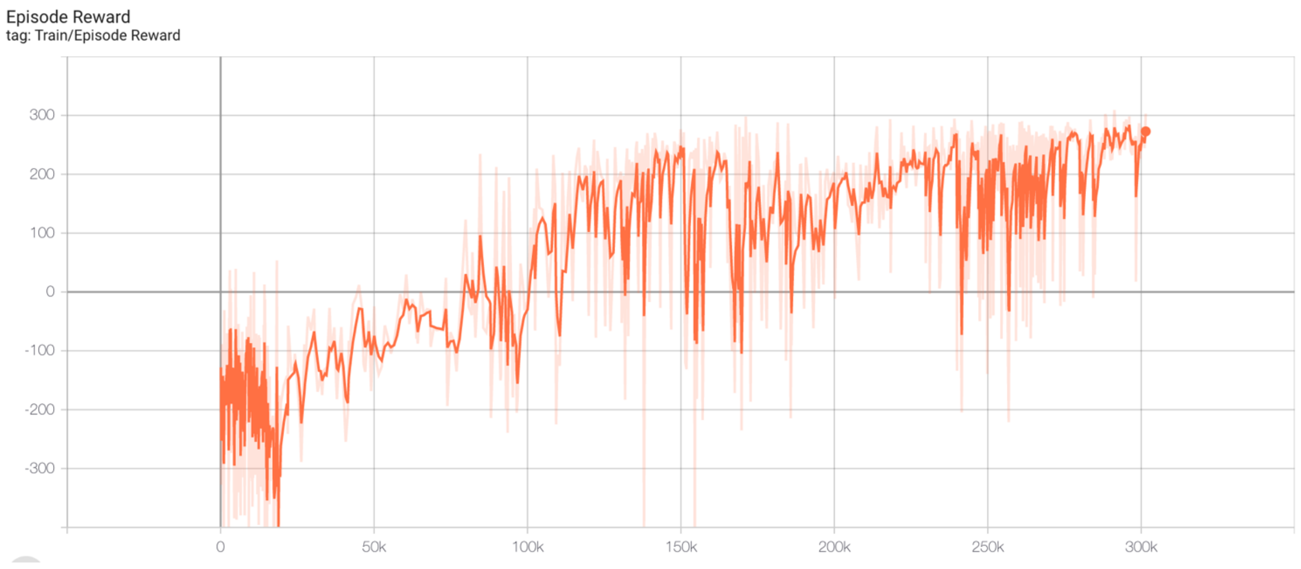
下圖附上 DDQN 的實作結果：

（訓練 800 epochs）



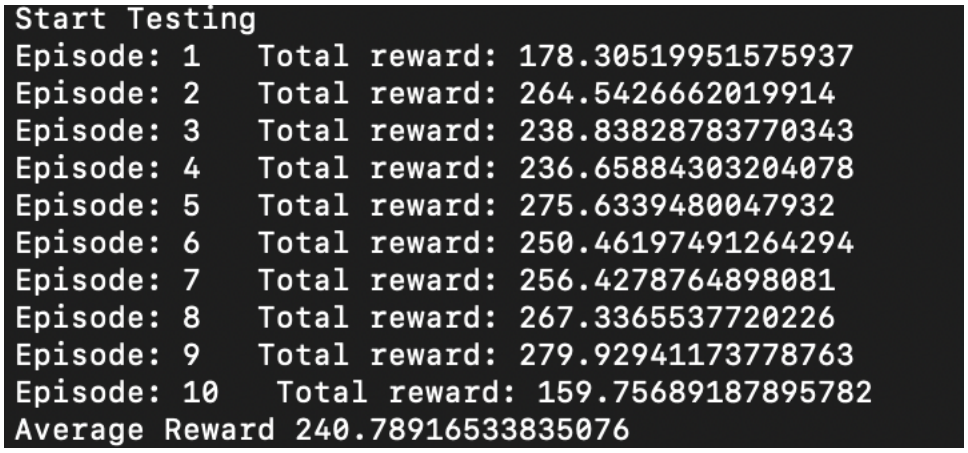
```
Start Testing
Episode: 1   Total reward: 231.30813891516007
Episode: 2   Total reward: 139.39698594380468
Episode: 3   Total reward: 273.3667620809215
Episode: 4   Total reward: 143.97435161297864
Episode: 5   Total reward: 297.6560242228736
Episode: 6   Total reward: 241.46379616471
Episode: 7   Total reward: 291.02351940501705
Episode: 8   Total reward: 278.0795613624954
Episode: 9   Total reward: 299.47702154559033
Episode: 10   Total reward: 298.6326360831682
Average Reward 249.43787973367193
```

tensorboard plot:



◆ Performance

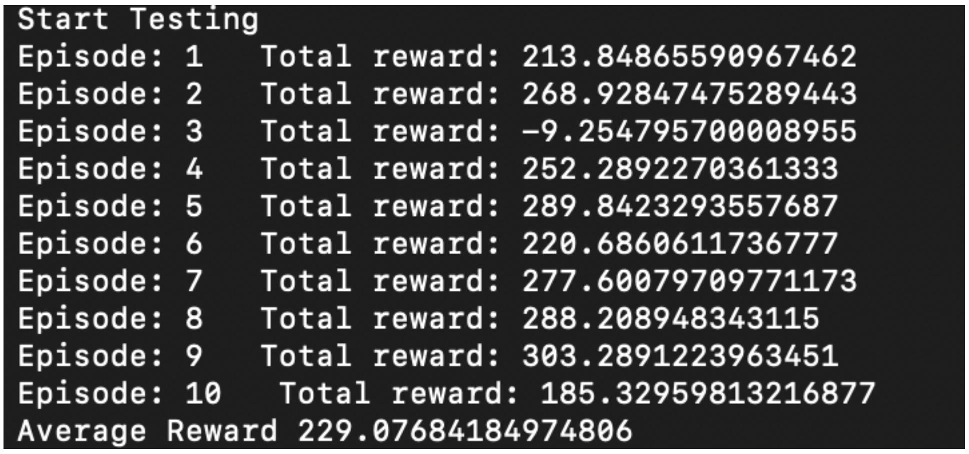
■ [LunarLander-v2] Average reward of 10 testing episodes:



```
Start Testing
Episode: 1   Total reward: 178.30519951575937
Episode: 2   Total reward: 264.5426662019914
Episode: 3   Total reward: 238.83828783770343
Episode: 4   Total reward: 236.65884303204078
Episode: 5   Total reward: 275.6339480047932
Episode: 6   Total reward: 250.46197491264294
Episode: 7   Total reward: 256.4278764898081
Episode: 8   Total reward: 267.3365537720226
Episode: 9   Total reward: 279.92941173778763
Episode: 10   Total reward: 159.75689187895782
Average Reward 240.78916533835076
```

Average Reward: 240

■ [LunarLanderContinuous-v2] Average reward of 10 testing episodes:



```
Start Testing
Episode: 1   Total reward: 213.84865590967462
Episode: 2   Total reward: 268.92847475289443
Episode: 3   Total reward: -9.254795700008955
Episode: 4   Total reward: 252.2892270361333
Episode: 5   Total reward: 289.8423293557687
Episode: 6   Total reward: 220.6860611736777
Episode: 7   Total reward: 277.60079709771173
Episode: 8   Total reward: 288.208948343115
Episode: 9   Total reward: 303.2891223963451
Episode: 10   Total reward: 185.32959813216877
Average Reward 229.07684184974806
```

Average Reward: 229