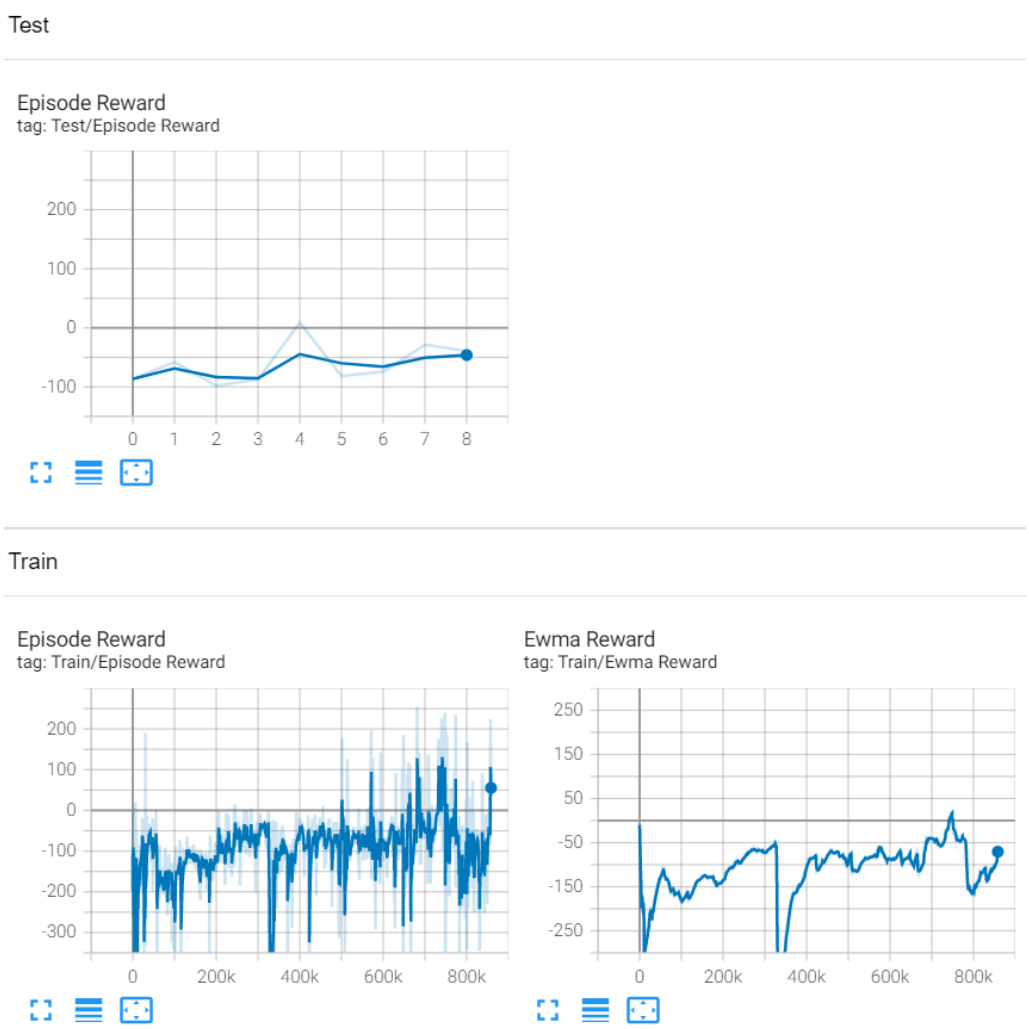
Deep Learning and Practice Lab 6

309552007

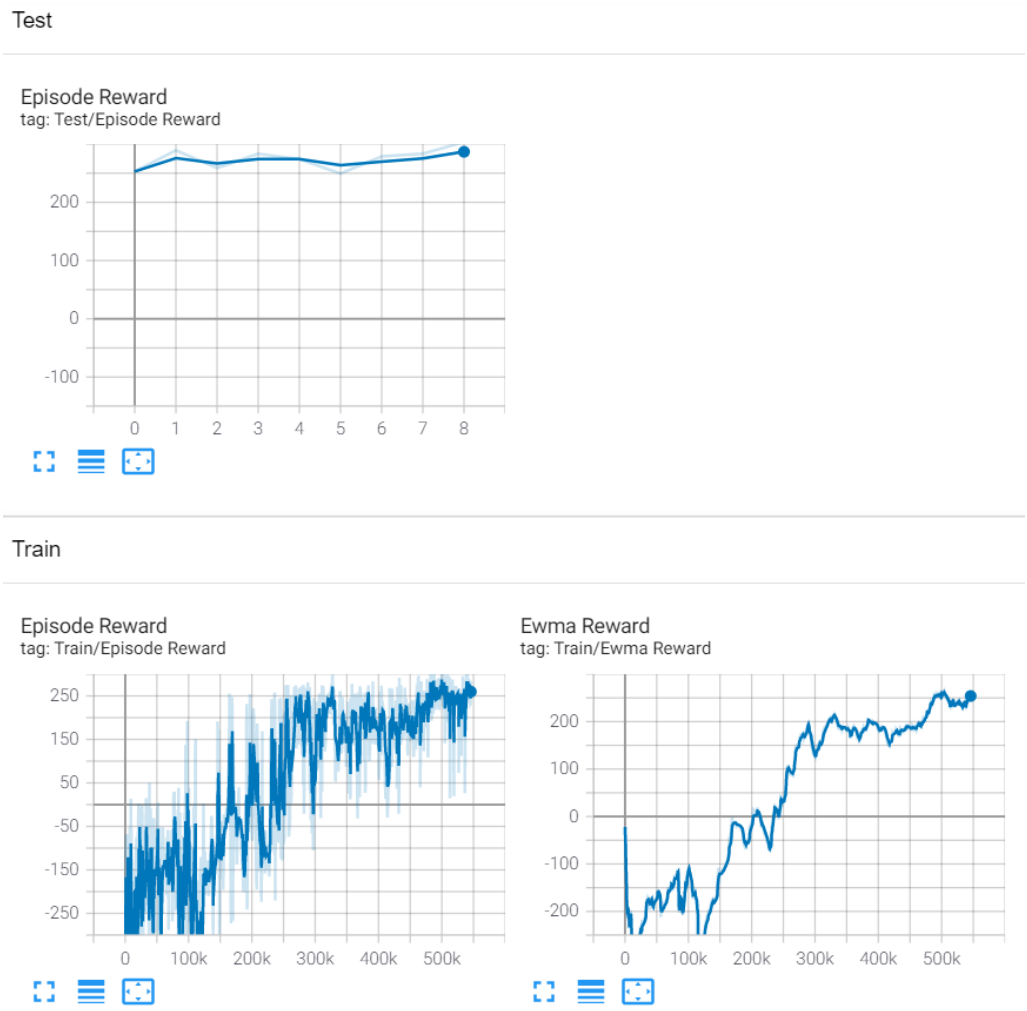
袁鈺勛

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



B. A tensorboard plot shows episode rewards of at least 800

training episodes in LunarLanderContinuous-v2



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

1. DQN

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

```
# 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))
```

上圖的程式碼便是根據上面的 algorithm 中的其中一段所做出來的，會根據當下 epsilon 的機率來選擇要採取 random action 還是從 behavior net 取得最好的 action。

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

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
# q_value = ?
# with torch.no_grad():
#     q_next = ?
#     q_target = ?
# criterion = ?
# loss = criterion(q_value, q_target)
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))
loss = nn.MSELoss()(q_value, q_target)
```

上面的程式碼也是根據上面的 algorithm 做出，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。

Every C steps reset $\hat{Q} = Q$

```
# TODO
self._target_net.load_state_dict(self._behavior_net.state_dict())
```

上面的程式碼也是根據上面的 algorithm 做出，他就是要將 target net 中的參數換成當下 behavior net 中的參數。

2. DDPG

Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy 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
```

上面的程式碼是根據上面的 algorithm 所做出，他會從 actor net 取出 action，並且加上 noise。

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

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

```
# Update critic
# critic loss
# TODO
# q_value = ?
# with torch.no_grad():
#     a_next = ?
#     q_next = ?
#     q_target = ?
# criterion = ?
# critic_loss = criterion(q_value, q_target)
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))
critic_loss = nn.MSELoss()(q_value, q_target)
```

上面的程式碼也是根據上面的 algorithm 做出，他會從 critic net 取得 q value，然後將從 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 來更新 critic net。

Update the actor policy using the sampled gradient:

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

```
# Update actor
# actor loss
# TODO
# action = ?
# actor_loss = ?
action = actor_net(state)
actor_loss = -critic_net(state, action).mean()
```

上面的程式碼也是根據上面的 algorithm 做出，他是取得當下 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去變化 parameter。

Update the target networks:

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

```
for target, behavior in zip(target_net.parameters(), net.parameters()):
    # TODO
    target.data.copy_(tau * behavior.data + (1.0 - tau) * target.data)
```

上面的程式碼也是根據上面的 algorithm 做出，他會 soft update target net。

D. Describe differences between your implementation and algorithms.

```
def _soft_update_target_network(self, tau=.005):
    """
    Update target network by _soft_ copying from behavior network
    :param tau: weight
    :return: None
    """
    for target, behavior in zip(self._target_net.parameters(), self._behavior_net.parameters()):
        target.data.copy_(tau * behavior.data + (1.0 - tau) * target.data)
```

因為要做 DDQN，所以在 dqn.py 中多了一個 soft update，在實驗 DDQN 的時候會採用，在每次 update target net 時會小幅度的更新他。

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

Update the actor policy using the sampled gradient:

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

```
# Update actor
# actor loss
# TODO
# action = ?
# actor_loss = ?
action = actor_net(state)
actor_loss = -critic_net(state, action).mean()
```

上面的程式碼便是更新 actor net，他會取得當下 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去變化 parameter。

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

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

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

```
# Update critic
# critic loss
# TODO
# q_value = ?
# with torch.no_grad():
#     a_next = ?
#     q_next = ?
#     q_target = ?
# criterion = ?
# critic_loss = criterion(q_value, q_target)
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))
critic_loss = nn.MSELoss()(q_value, q_target)
```

上面的程式碼便是更新 critic net，他會從 critic net 取得當下的 q value，然後將從 target actor net 取得的 action 放進 target critic net 進而取得 q next 並進而得到 q target，經由 q value 和 q target 便可以計算 loss，以此 loss 來更新 critic net。

G. Explain effects of the discount factor.

讓 model 用來決定要多在意愈遠的 future，如果 discount factor 愈大，代表愈要考慮較遠的 future，如果愈小，代表比較考慮較近的 future。

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

如果只採用 greedy action selection 總是選擇最好的 action，有可能會找不到真正最好的，因為他可能會是要在某一步採取較不好的 action，所以用 epsilon-greedy 上 model 有時候會 random 選一個 action，以此達到 exploration 的功能。

I. Explain the necessity of the target network.

如果有 target network 的話就可以根據過往的經驗來看現在 behavior network 在嘗試的新 action，才不會讓 behavior network 反而往不好的方向一直走下去，藉此讓 behavior network 會修正去往較好的 action 走。

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

如果 replay buffer 太小，那 model 只會考慮到最近的 data，那就有可能 overfit。如果 replay buffer 太大，那就會佔用太多的 memory 空間，同時也會拖慢 training。

K. Implement and experiment on Double-DQN

```
# TODO 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))

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

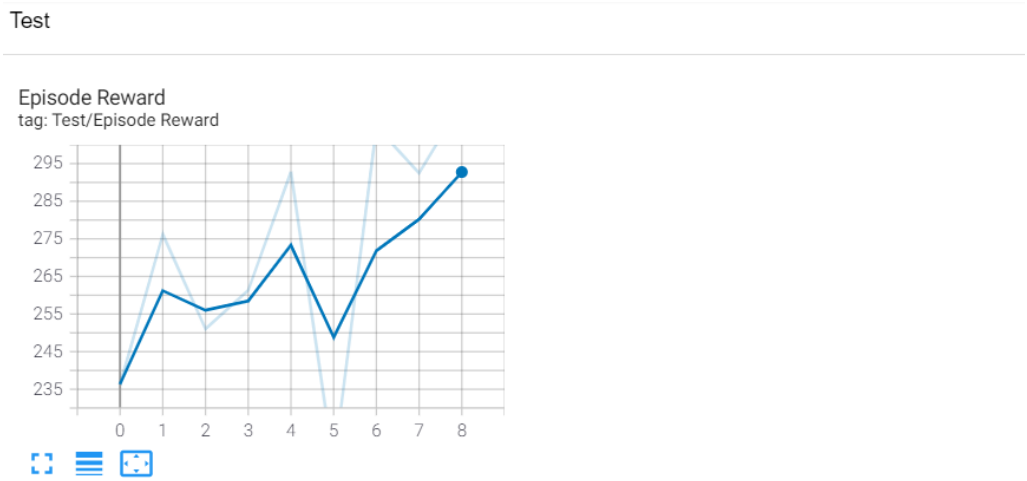
上面便是 DDQN 的實作，和 DQN 不同的是他會先經由 behavior net 取得 action 再帶到 target net 取得 q next，才以此取得 q target。

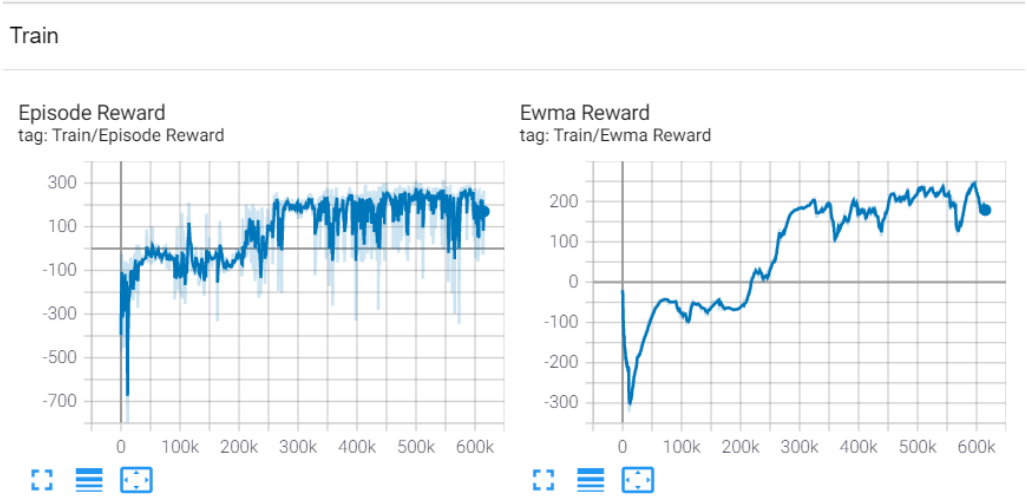
```
# TODO DDQN
self._soft_update_target_network()
```

```
def _soft_update_target_network(self, tau=.005):
    """
    Update target network by _soft_ copying from behavior network
    :param tau: weight
    :return: None
    """
    for target, behavior in zip(self._target_net.parameters(), self._behavior_net.parameters()):
        target.data.copy_(tau * behavior.data + (1.0 - tau) * target.data)
```

上面便是在做 DDQN 的時候加上得 soft update 來提升 reward，在每次 update target net 的時候都會小幅度更新他。

L. LunarLander-v2 performance



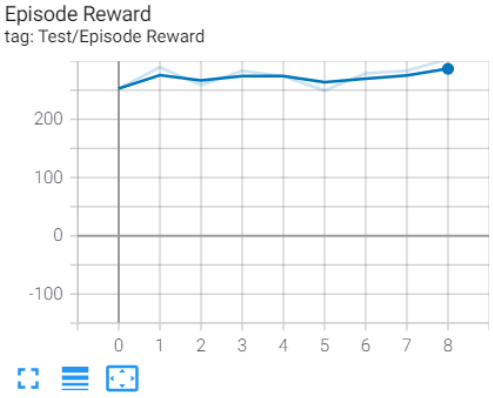



```
Step: 614159    Episode: 1197   Length: 196     Total reward: -28.77    Ewma reward: 172.81  E
psilon: 0.010
Step: 614469    Episode: 1198   Length: 310     Total reward: 264.29    Ewma reward: 177.39  E
psilon: 0.010
Step: 614822    Episode: 1199   Length: 353     Total reward: 193.15    Ewma reward: 178.18  E
psilon: 0.010
Start Testing
Average Reward 267.1178147316768
```

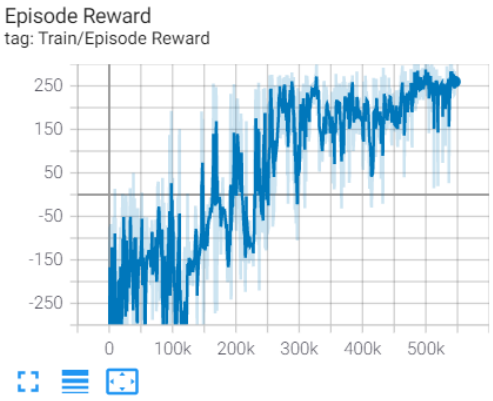
上圖得到的結果是設定 capacity 為 100000，並且利用 soft update，但 tau 是設定 0.9。

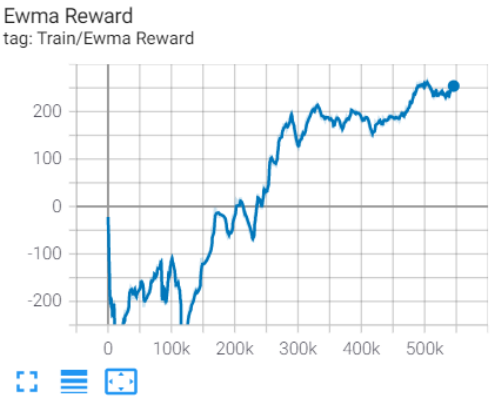
M. LunarLanderContinous-v2 performance

Test



Train






```
Average Reward 270.2179943079367
```

上圖得到的結果是用預設的 hyperparameters。