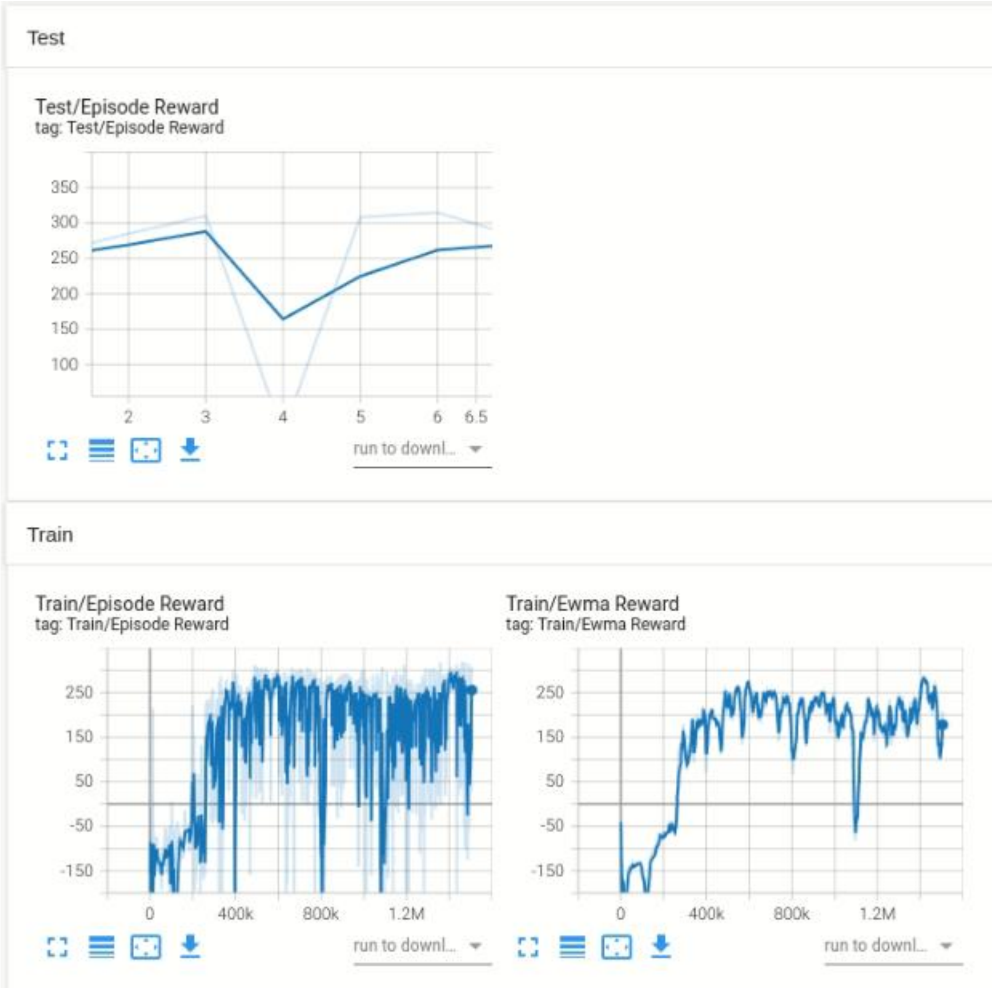
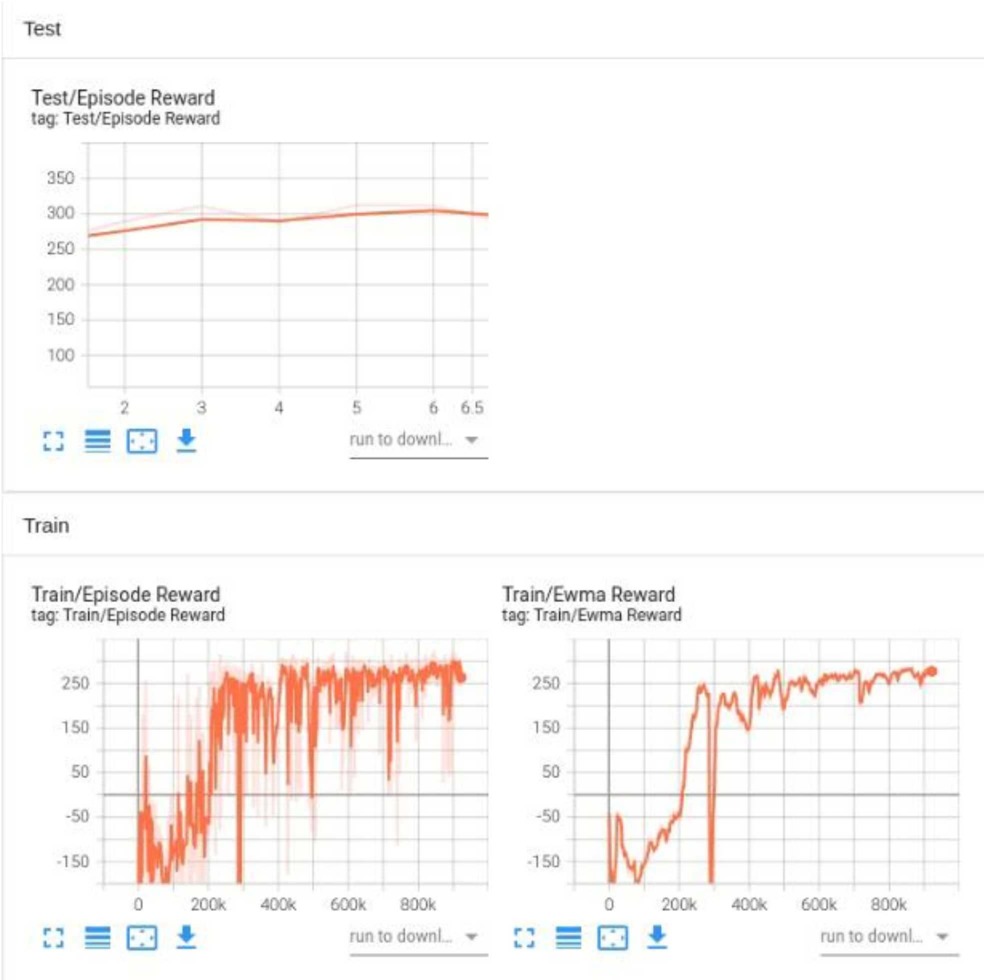
Lab 6:Deep Q-Network and Deep Determinstic Policy Gradient

學號:311605015 張哲源

1.A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLander-v2 (5%)



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.

3.1DQN :

創建模型，輸入 STATE 輸出 action

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

        self.fc1 = nn.Linear(state_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim * 2)
        self.fc3 = nn.Linear(hidden_dim * 2, action_dim)

        #raise NotImplementedError


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

        return x
        #raise NotImplementedError
```

選擇 action，透過 greedy policy 選擇當下能產生最大 Q 值得 action，但也有一定機率隨機選擇行動

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

    rnd = random.random()
    if rnd < epsilon:
        return np.random.randint(action_space.n)
    else:
        state = torch.from_numpy(state).float().unsqueeze(0).cuda()
        with torch.no_grad():
            actions_value = self._behavior_net.forward(state)#state a
        action = np.argmax(actions_value.cpu().data.numpy()) #take ma

    return action
    #raise NotImplementedError
```

Batch_size 設為 64，discount factor 設為 0.99，target_net 更新率為 behavior net 走 1000 步便將參數複製到 target net，optimizer 使用 Adam，並設計 replay memory 將過往的遊玩記錄存起來。

```python
class DQN:
    def __init__(self, args):
        self._behavior_net = Net().to(args.device)
        self._target_net = Net().to(args.device)
        # initialize target network
        self._target_net.load_state_dict(self._behavior_net.state_dict())
        ## TODO ##
        # self._optimizer = ?
        self._optimizer = torch.optim.Adam(self._behavior_net.parameters(), lr=arg

        #raise NotImplementedError
        # memory
        self._memory = ReplayMemory(capacity=args.capacity)

        ## config ##
        self.device = args.device
        self.batch_size = args.batch_size
        self.gamma = args.gamma #0.99
        self.freq = args.freq
        self.target_freq = args.target_freq
```

```
def _update_behavior_network(self): #change the gamma rate
    # sample a minibatch of transitions
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size, self.device)
    q_value = self._behavior_net(state).gather(1, action.long())

    with torch.no_grad():
        q_next = self._target_net(next_state)  # Not Backpropagate
        q_target = reward +self.gamma * q_next.max(1)[0].view(self.batch_size,
    #print('reward:',reward,'q_value:',q_value,'q_target:',q_target)
    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()
```

3.2 DDQN

```
class DDPG:
    def __init__(self, args):
        # behavior network
        self._actor_net = ActorNet().to(args.device)
        self._critic_net = CriticNet().to(args.device)
        # target network
        self._target_actor_net = ActorNet().to(args.device)
        self._target_critic_net = CriticNet().to(args.device)
        # initialize target network
        self._target_actor_net.load_state_dict(self._actor_net.state_dict())
        self._target_critic_net.load_state_dict(self._critic_net.state_dict())
        ## TODO ##
        self._actor_opt = torch.optim.Adam(self._actor_net.parameters(), lr=args.l
        self._critic_opt = torch.optim.Adam(self._critic_net.parameters(), lr=args
        # self._actor_opt = ?
        # self._critic_opt = ?
        # raise NotImplementedError
        # action noise
        self._action_noise = GaussianNoise(dim=2)
        # memory
        self._memory = ReplayMemory(capacity=args.capacity)

        ## config ##
        self.device = args.device
        self.batch_size = args.batch_size
        self.tau = args.tau
        self.gamma = args.gamma
```

Soft update:每次更新網路一點點。

Actor net:將 State 輸入進去並返回個個 actor 的輸出

Critic net:將 State 及 action 輸入並返回預測的 Q 值

```python
def _update_behavior_network(self, gamma):
    actor_net, critic_net, target_actor_net, target_critic_net = self._actor_n
    actor_opt, critic_opt = self._actor_opt, self._critic_opt

    # sample a minibatch of transitions
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size, self.device)
    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 + (self.gamma * q_next * (1 - done))

    criterion = nn.MSELoss()
    critic_loss = criterion(q_value, q_target)
    # raise NotImplementedError
    # optimize critic
    actor_net.zero_grad()
    critic_net.zero_grad()
    critic_loss.backward()
    critic_opt.step()

    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()
```

4.Describe differences between your implementation and algorithms.

在 TRANING 的時候有設計一個 WARMUP，在這段時間，系統會隨便玩(不按照 GREEDY Policy)選擇 action，並將資料存在 replay memory 裡，另外在 DQN 裡，並不是每個 iteration 都要更新 Behavior Network ，而是每隔一段時間(4 iteration) 才會更新一次。

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

利用 behavior Network 的 s 及 a 可以求出 Q(s,a)，我們想要更新 Actor Network 史的輸出的 Q(s,a)越大越好，因此定義 Loss Value=-Q(s,u(s)),backpropagation 的時候不更新 critic，只更新 Actor

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

    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()
```

6. Describe your implementation and the gradient of critic updating

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

方法使用兩個網路來達成學習動作，一為 Actor 網路，主要用來輸出動作，policy gradient 的網路長的很像， Actor 網路就是從 policy gradient 演化而來的，主要是改進 policy gradient 回合更新制的缺點，加了 Critic 網路之後就可以使用 TD error 當作 advantage function 做每步更新的步驟了

```
state, action, reward, next_state, done = self._memory.sample(
    self.batch_size, self.device)
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 + (self.gamma * q_next * (1 - done))


criterion = nn.MSELoss()
critic_loss = criterion(q_value, q_target)
# raise NotImplementedError
# optimize critic
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

7. Explain effects of the discount factor.

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

從的公式中可得知，discount factor (γ)可控制 agent 較重視眼前 的 reward 或是從歷史資料計算出的長期利益，若 γ 越小，則 agent 越短視近 利，只重視眼前 reward；若 γ 越大，則 agent 越重視長期利益。

8. Explain benefits of epsilon-greedy in comparison to greedy action selection.
我們在 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.
因為 DQN 是以 neural network 來取代 table，所以這變成一個預測 output Q(s, a) 的 regression 問題，可是因為 Q-learning 中 Q 值的遞迴關係，加上 Q 值一直在 變動，而導致模型訓練難以穩定，因此，我們再製作一個 target

network，作為 另一個一段時間才更新一次的 Q network，更新時就直接把實際在訓練的 Q network 權重複製給 target network 即可。加上 target network 的做法能使模型訓 練更加穩定。

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

buffer size 如果太大，會使需要的記憶體空間過大、訓練時間過長，而且 buffer 中較近期的資料會被早期的資料稀釋，較難從新的資料中學習，但如果 buffer size 過小，則會使隨機採樣後的資料相關性過低，因為 buffer 中大多是近期剛 加入的資料，而導致神經網路的訓練不穩定

Report Bonus(20%)

Implement and experiment on Double-DQN

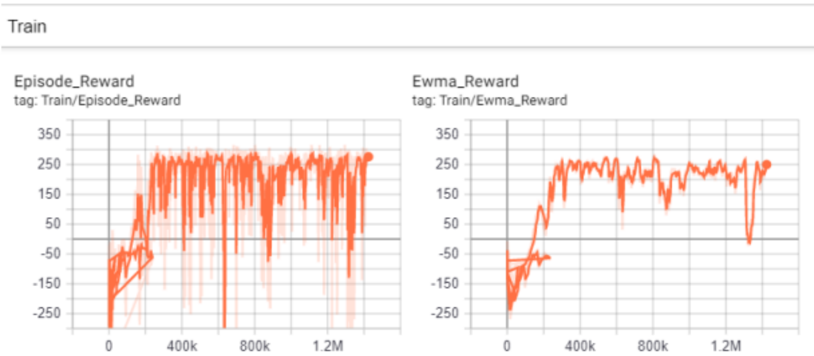
以 Double-DQN 的概念，嘗試使用另一個 Q function 來一起計算 Q value

```python
q_value = self._behavior_net(state).gather(1, action.long())  # shape (batch, 1)

with torch.no_grad():
    q_next = self._behavior_net(next_state)
    q_target_next = self._target_net(next_state)
    s_value =  q_target_next.gather(1, torch.max(q_next, 1)[1].unsqueeze(1))
    q_target = reward + gamma * s_value * (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()
```

```
Step: 1418637   Episode: 4980   Length: 265    Total reward: 249.42   Ewma reward: 245.18   Epsilon: 0.010
Step: 1418830   Episode: 4981   Length: 193    Total reward: 286.07   Ewma reward: 247.22   Epsilon: 0.010
Step: 1419039   Episode: 4982   Length: 209    Total reward: 263.68   Ewma reward: 248.05   Epsilon: 0.010
Step: 1419852   Episode: 4983   Length: 813    Total reward: 265.16   Ewma reward: 248.90   Epsilon: 0.010
Step: 1420010   Episode: 4984   Length: 158    Total reward: 262.63   Ewma reward: 249.59   Epsilon: 0.010
Step: 1420195   Episode: 4985   Length: 185    Total reward: 315.58   Ewma reward: 252.89   Epsilon: 0.010
Step: 1420411   Episode: 4986   Length: 216    Total reward: 293.13   Ewma reward: 254.90   Epsilon: 0.010
Step: 1420580   Episode: 4987   Length: 169    Total reward: 248.11   Ewma reward: 254.56   Epsilon: 0.010
Step: 1420860   Episode: 4988   Length: 280    Total reward: 278.73   Ewma reward: 255.77   Epsilon: 0.010
Step: 1421037   Episode: 4989   Length: 177    Total reward: 264.21   Ewma reward: 256.19   Epsilon: 0.010
Step: 1421299   Episode: 4990   Length: 262    Total reward: 304.03   Ewma reward: 258.58   Epsilon: 0.010
Step: 1421511   Episode: 4991   Length: 212    Total reward: 286.74   Ewma reward: 259.99   Epsilon: 0.010
Step: 1421846   Episode: 4992   Length: 335    Total reward: 261.44   Ewma reward: 260.06   Epsilon: 0.010
Step: 1422041   Episode: 4993   Length: 195    Total reward: 321.19   Ewma reward: 263.12   Epsilon: 0.010
Step: 1422729   Episode: 4994   Length: 688    Total reward: 247.29   Ewma reward: 262.33   Epsilon: 0.010
Step: 1423234   Episode: 4995   Length: 505    Total reward: 231.73   Ewma reward: 260.80   Epsilon: 0.010
Step: 1423396   Episode: 4996   Length: 162    Total reward: 289.90   Ewma reward: 262.25   Epsilon: 0.010
Step: 1423640   Episode: 4997   Length: 244    Total reward: 276.51   Ewma reward: 262.97   Epsilon: 0.010
Step: 1423942   Episode: 4998   Length: 302    Total reward: 246.53   Ewma reward: 262.14   Epsilon: 0.010
Step: 1424149   Episode: 4999   Length: 207    Total reward: 291.93   Ewma reward: 263.63   Epsilon: 0.010
Start Testing
Average Reward 173.6489995914115
```



Performance (20%)

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

```
  deprecation(
total_reward: 287.488163244048
total_reward: 294.9382685008662
total_reward: 263.90903876025243
total_reward: 308.40285070248393
total_reward: 286.6580726374177
total_reward: 277.7306440284532
total_reward: 264.6687985452663
total_reward: 290.40457115950574
total_reward: 267.04176915187975
total_reward: 304.4015569676286
Average Reward 284.5643733697801
```

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

```
  deprecation(
total_reward:  260.25527144275145
total_reward:  288.88118963049567
total_reward:  311.96954882213015
total_reward:  288.18176354919444
total_reward:  311.96893752575306
total_reward:  312.990006010671
total_reward:  284.23728438849105
total_reward:  306.1980933701955
total_reward:  263.49355362604996
Average Reward 289.07899546932435
```