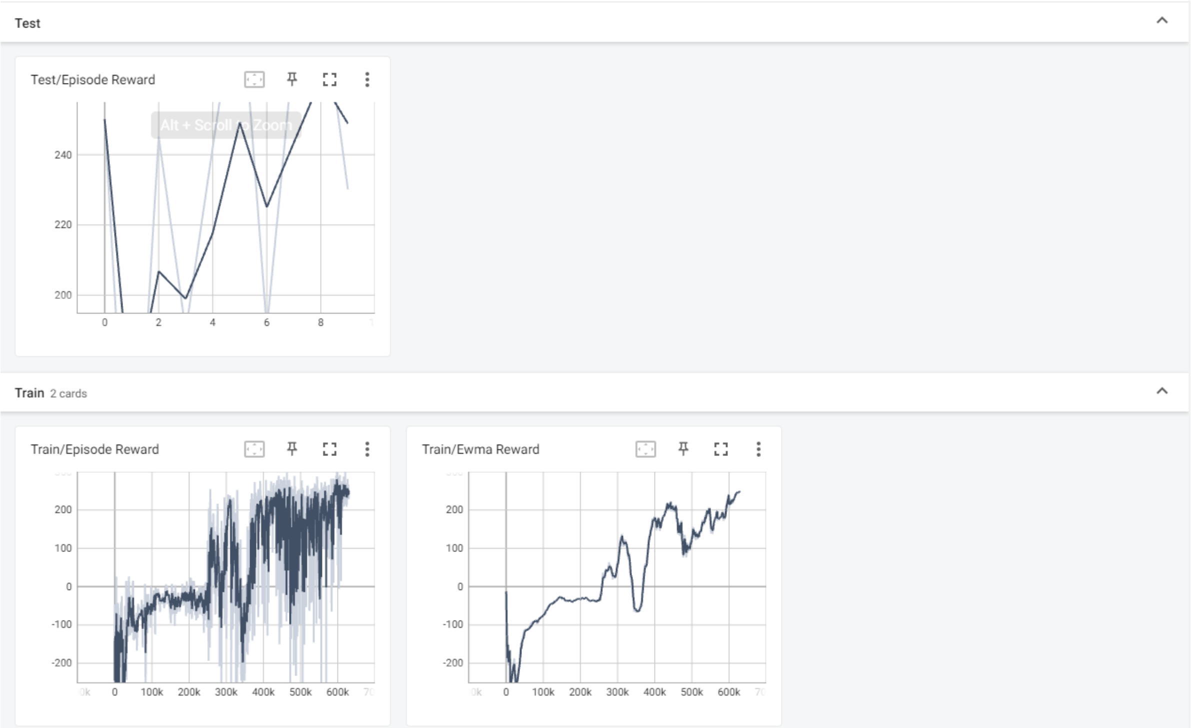
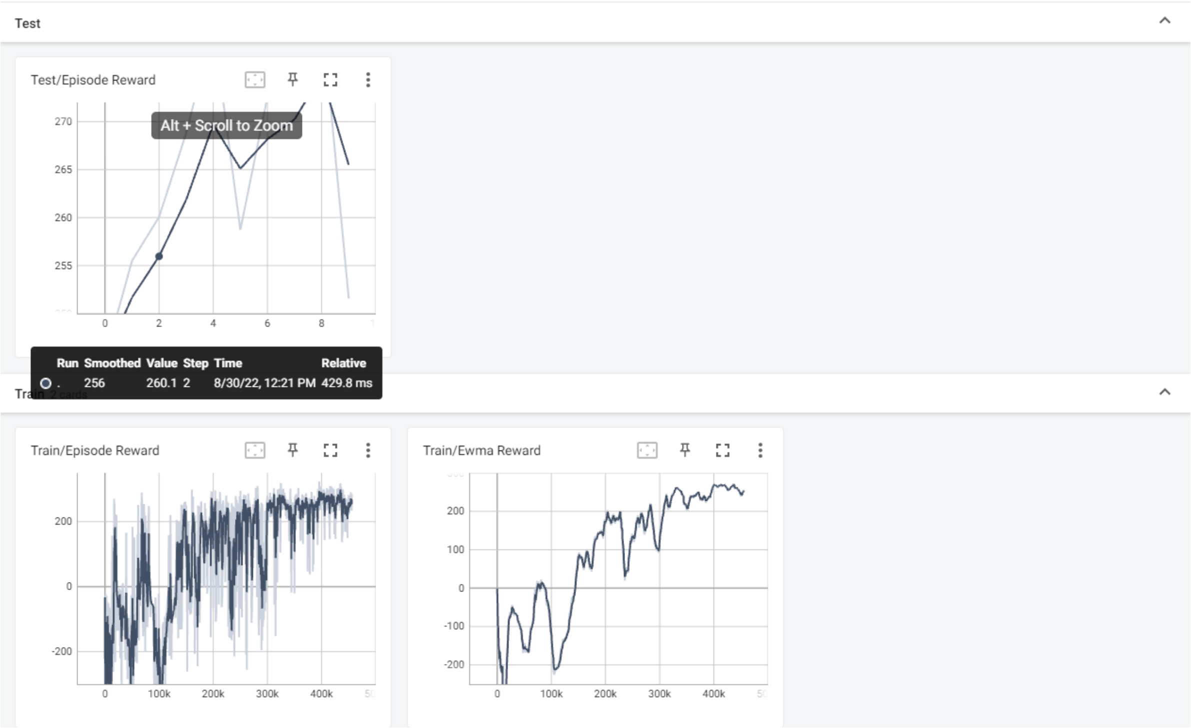
Lab6 - Deep Q-Network and Deep Deterministic Policy Gradient

- Student Info
  1. Student ID: 310555024
  2. Student Name: 林廷翰

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



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

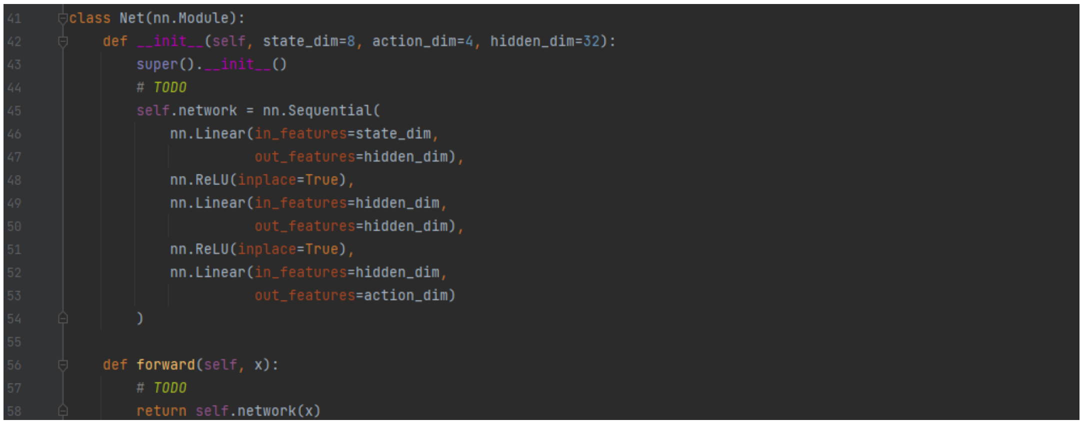


Run	Smoothed	Value	Step	Time	Relative
○ .	256	260.1	2	8/30/22, 12:21 PM	429.8 ms

- Describe your major implementation of both algorithms in detail. (20%)

1. DQN

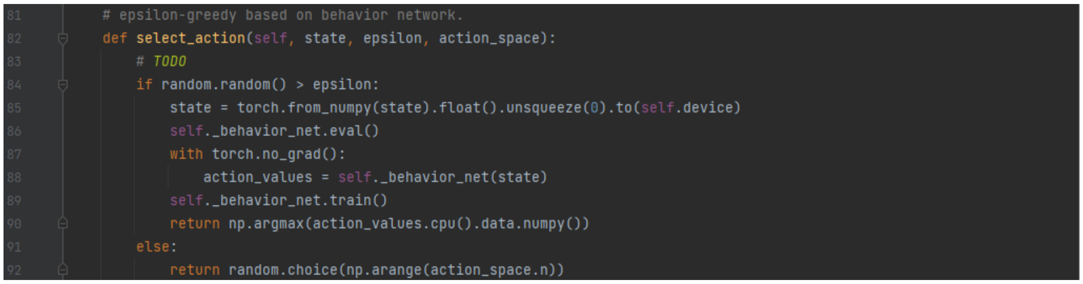
    - Net



```python
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=32):
        super().__init__()
        # TODO
        self.network = nn.Sequential(
            nn.Linear(in_features=state_dim,
                      out_features=hidden_dim),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=hidden_dim,
                      out_features=hidden_dim),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=hidden_dim,
                      out_features=action_dim)
        )

    def forward(self, x):
        # TODO
        return self.network(x)
```

在DQN中用3 layers的network來預測Q(s, a)的值，因為最後分類結果action有4種可 (No-op, Fire left engine, Fire main engine, Fire right engine)，所以最後一層 dim=4。

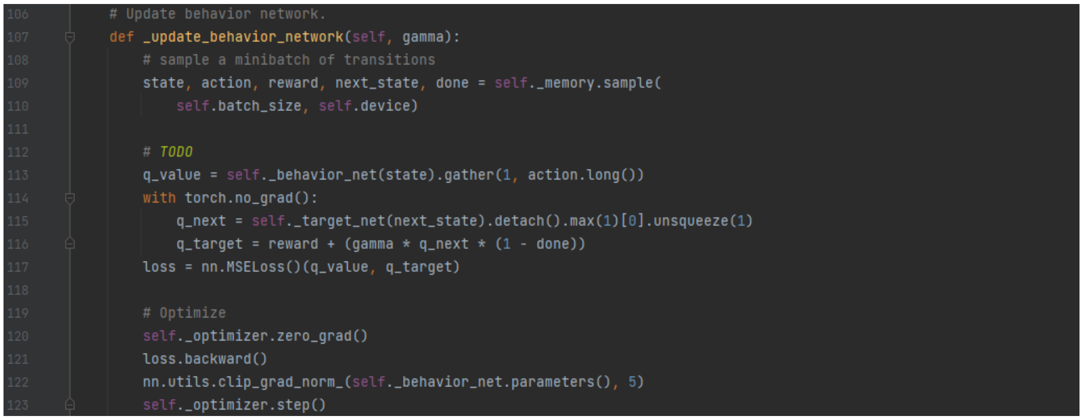
- select_action



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

select action在每輪episode中選擇最大Q(s,ai)的ai或有一定機率選擇。

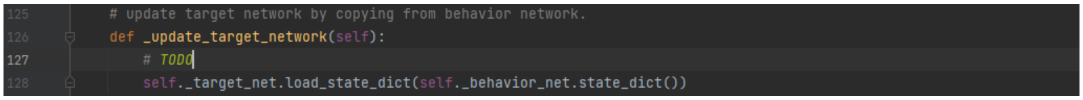
- _update_behavior_network



```
    # Update behavior network.
    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(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)

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

update network是由replay memory中sampling一些遊戲過程來做TD learning，再用Q(s, a)與 r + gamma * maxa’Q’(s’,a’)的差做 MSELoss。

- _update_target_network

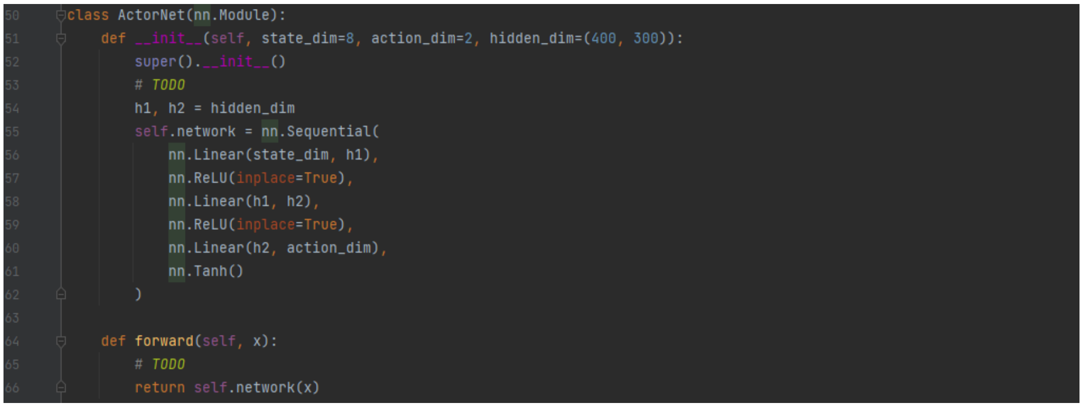


```
    # update target network by copying from behavior network.
    def _update_target_network(self):
        # TODO
        self._target_net.load_state_dict(self._behavior_net.state_dict())
```

最後每隔一段時間就用bebavior network取代target network。

2. DDPG

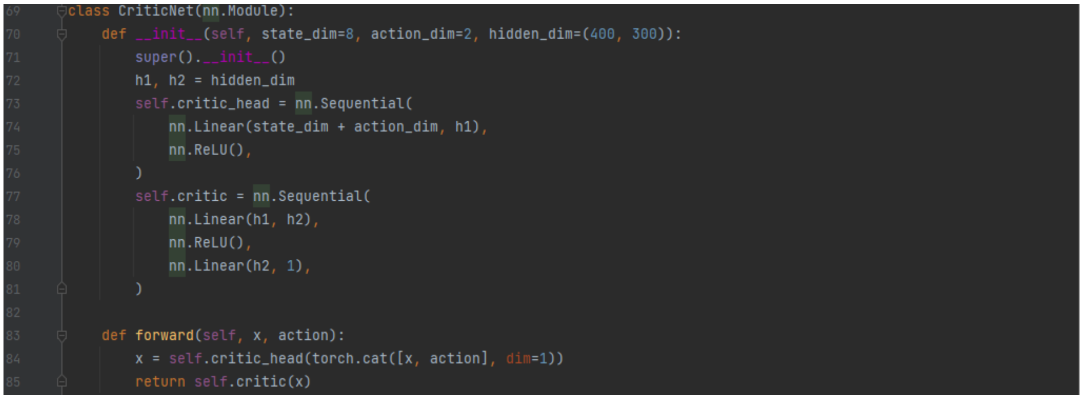
- ActorNet



```python
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.network = nn.Sequential(
            nn.Linear(state_dim, h1),
            nn.ReLU(inplace=True),
            nn.Linear(h1, h2),
            nn.ReLU(inplace=True),
            nn.Linear(h2, action_dim),
            nn.Tanh()
        )

    def forward(self, x):
        # TODO
        return self.network(x)
```

actor network可以根據當前的state決定下個要執行的action, 因為action有2個(main engine, left-right-engine), 所以最後輸出 dim=2。

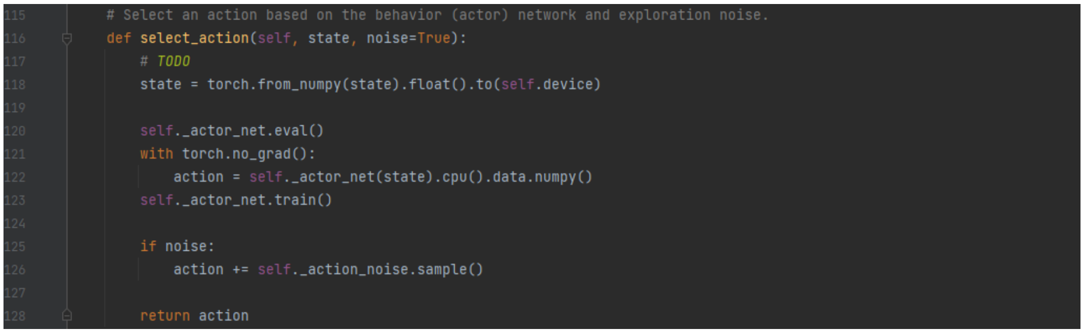
- CriticNet



```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)
```

critic network可以預估Q(s, a), 由於輸出為scalar, 所以輸出 dim=1。

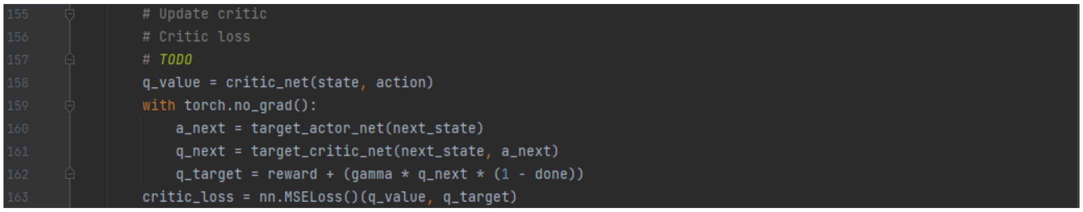
- select_action



```python
    # Select an action based on the behavior (actor) network and exploration noise.
    def select_action(self, state, noise=True):
        # 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
```

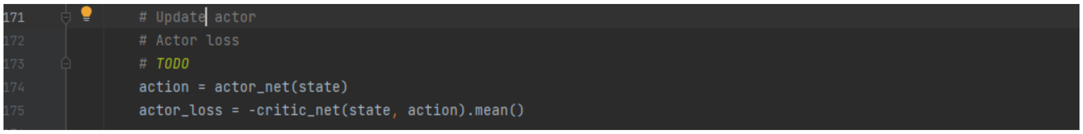
由actor network 選擇action並加上noise。

- _update_behavior_network



```
# Update critic
# Critic loss
# TODO
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)
```

在每輪episode裡更新behavior的actor network μ, critic network Q, target 的 actor network μ’, critic network Q’。 再利用target network輸出的q_target和 behavior network輸出的q_value取得MSE loss。



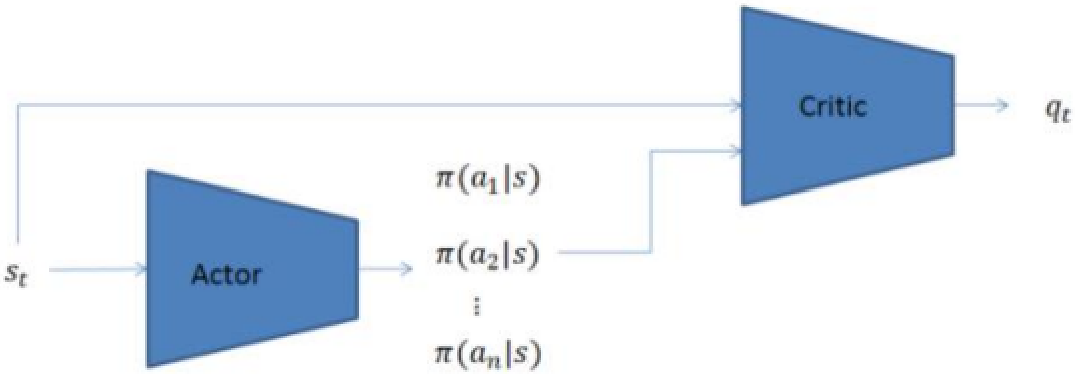
```
# Update actor
# Actor loss
# TODO
action = actor_net(state)
actor_loss = -critic_net(state, action).mean()
```

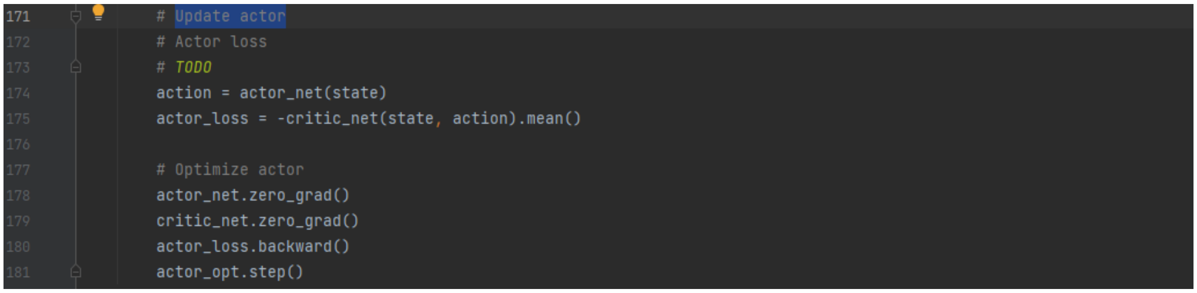
用behavior network的actor network μ 跟critic network Q 求得 Q(s, a),並且定義 loss function E[-Q(a, μ(s))]更新 μ 使得輸出的 Q(s, a)越大越好, 並透過 bp 更新網路。

- Describe the differences between your implementation and algorithms. (10%)

  在剛開始時會有一段時間不更新 network 中的參數而執行遊戲(隨機選action)，並把遊戲過程存到 replay memory 中，且 DQN 部分中每隔 4 個iterations 才更新 behavior network。

- Describe your implementation and the gradient of actor updating. (10%)

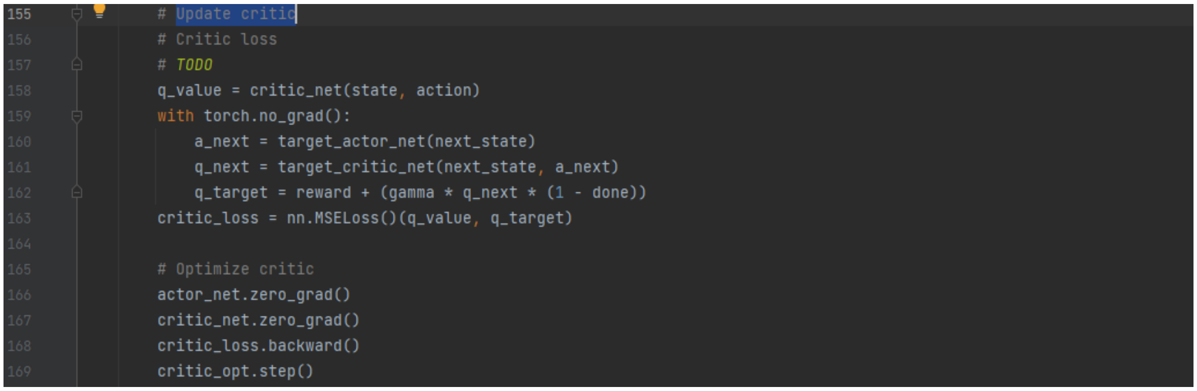




```
# Update actor
# Actor loss
# TODO
action = actor_net(state)
actor_loss = -critic_net(state, action).mean()

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

可以利用behavior network中的actor network μ 和critic network Q 求出 Q(s, a)，這時定義loss function為-Q(s, μ(s))來更新actor network μ, bp時只更新actor，不更新critic, 使得輸出 Q(s, a)越大越好。

- Describe your implementation and the gradient of critic updating. (10%)



```
# Update critic
# Critic loss
# TODO
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)

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

critic是利用target network輸出的Qtarget和behavior network輸出的Q(s, a)做MSE來更新Q network。

- Explain the effects of the discount factor. (5%)

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

  λ 即為 discount factor, 意即越後面的 reward 的影響越小, 當前 reward 的影響最大。

- Explain the benefits of epsilon-greedy in comparison to greedy action selection. (5%)

  用 epsilon-greedy 的方式因為偶爾會選擇其他隨機 action，可以避免陷入局部最佳解的困境。

- Explain the necessity of the target network. (5%)

  有 target network 和 behavior network 搭配使 training 更穩定, 因為 target network 每隔一段時間才會更新。

- Explain the effect of replay buffer size in case of too large or too small. (5%)

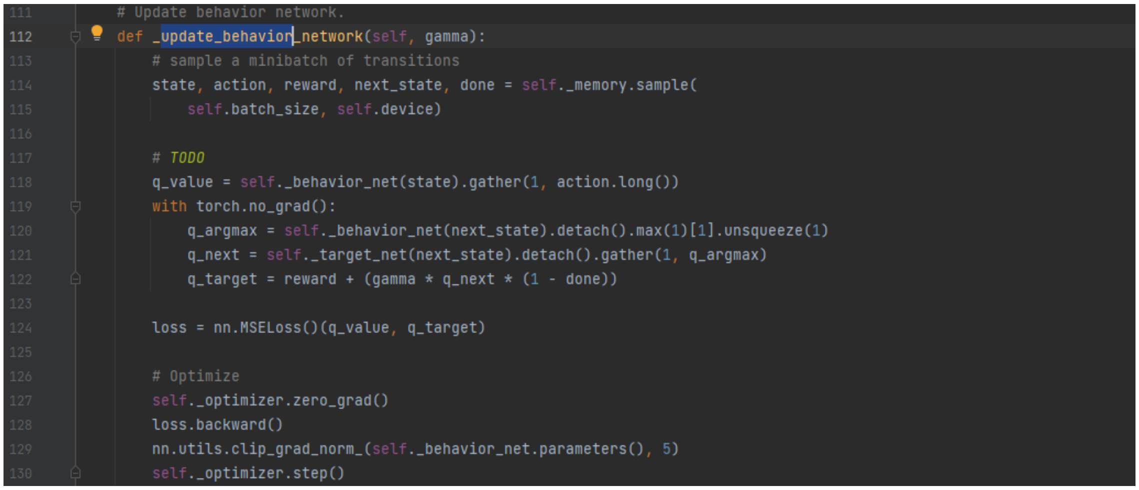
  當 replay buffer size 越大, training 可以更穩定, 但相對會降低速度。當它太小時, 會一直著重在最近幾次 episode 中, 造成 overfitting。

- Implement and experiment on Double-DQN

  1. Overview

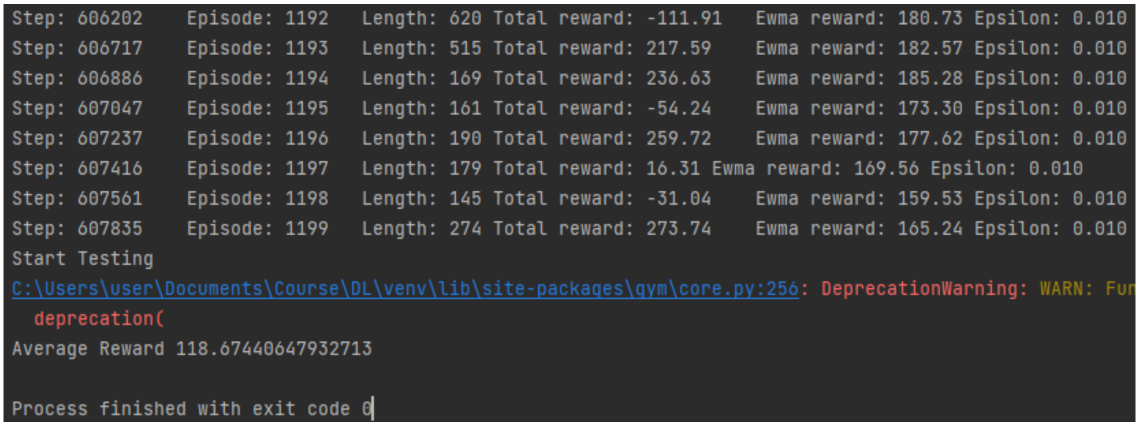
     DDQN 和 DQN 在實際上差不多, 差別只在更新 behavior network 時是如何決定 q_target 的；DDQN 是用 maxarg i Q(s, ai)作為找 Q’(s,ai)的 index, 而不是直接取 max Q’(s,ai)。

  2. _update_behavior_network



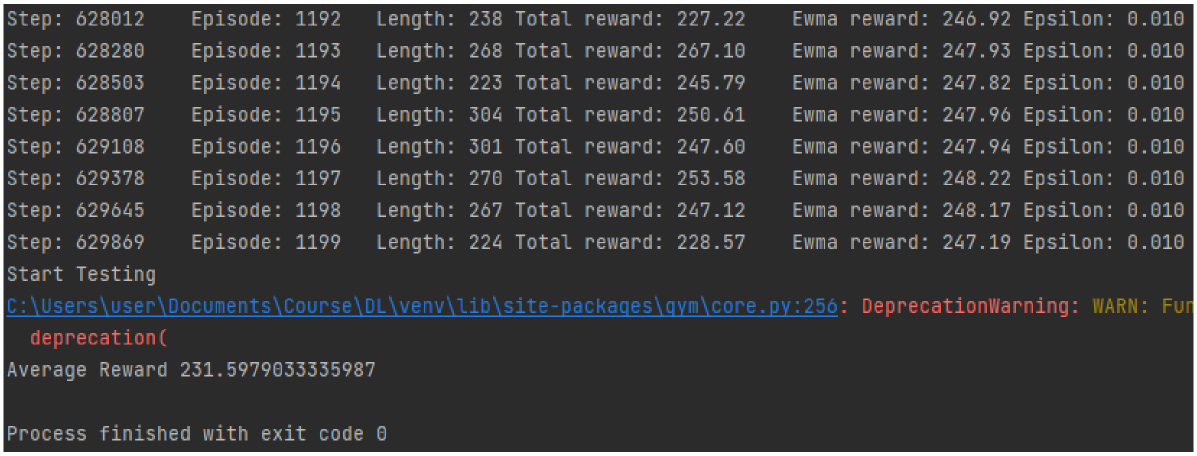
```python
# Update behavior network.
def _update_behavior_network(self, gamma):
    # sample a minibatch of transitions
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size, self.device)

    # TODO
    q_value = self._behavior_net(state).gather(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)

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

3. Average reward



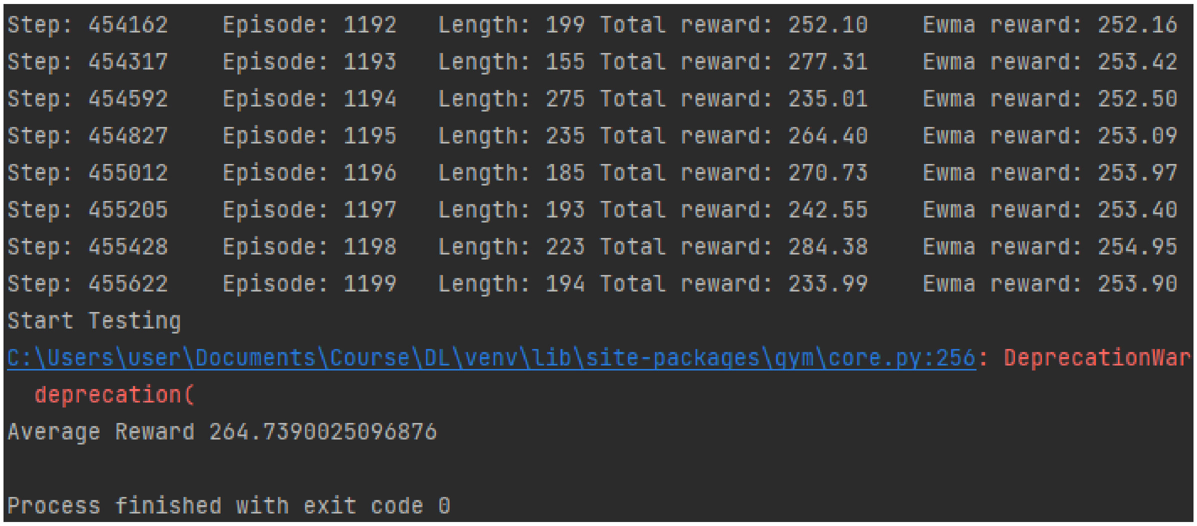
```
Step: 606202    Episode: 1192   Length: 620 Total reward: -111.91    Ewma reward: 180.73 Epsilon: 0.010
Step: 606717    Episode: 1193   Length: 515 Total reward: 217.59     Ewma reward: 182.57 Epsilon: 0.010
Step: 606886    Episode: 1194   Length: 169 Total reward: 236.63     Ewma reward: 185.28 Epsilon: 0.010
Step: 607047    Episode: 1195   Length: 161 Total reward: -54.24     Ewma reward: 173.30 Epsilon: 0.010
Step: 607237    Episode: 1196   Length: 190 Total reward: 259.72     Ewma reward: 177.62 Epsilon: 0.010
Step: 607416    Episode: 1197   Length: 179 Total reward: 16.31 Ewma reward: 169.56 Epsilon: 0.010
Step: 607561    Episode: 1198   Length: 145 Total reward: -31.04     Ewma reward: 159.53 Epsilon: 0.010
Step: 607835    Episode: 1199   Length: 274 Total reward: 273.74     Ewma reward: 165.24 Epsilon: 0.010
Start Testing
C:\Users\user\Documents\Course\DL\venv\lib\site-packages\gym\core.py:256: DeprecationWarning: WARN: Fun
  deprecation(
Average Reward 118.67440647932713

Process finished with exit code 0
```

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



```
Step: 628012    Episode: 1192   Length: 238 Total reward: 227.22     Ewma reward: 246.92 Epsilon: 0.010
Step: 628280    Episode: 1193   Length: 268 Total reward: 267.10     Ewma reward: 247.93 Epsilon: 0.010
Step: 628503    Episode: 1194   Length: 223 Total reward: 245.79     Ewma reward: 247.82 Epsilon: 0.010
Step: 628807    Episode: 1195   Length: 304 Total reward: 250.61     Ewma reward: 247.96 Epsilon: 0.010
Step: 629108    Episode: 1196   Length: 301 Total reward: 247.60     Ewma reward: 247.94 Epsilon: 0.010
Step: 629378    Episode: 1197   Length: 270 Total reward: 253.58     Ewma reward: 248.22 Epsilon: 0.010
Step: 629645    Episode: 1198   Length: 267 Total reward: 247.12     Ewma reward: 248.17 Epsilon: 0.010
Step: 629869    Episode: 1199   Length: 224 Total reward: 228.57     Ewma reward: 247.19 Epsilon: 0.010
Start Testing
C:\Users\user\Documents\Course\DL\venv\lib\site-packages\gym\core.py:256: DeprecationWarning: WARN: Fun
  deprecation(
Average Reward 231.5979033335987

Process finished with exit code 0
```

- [LunarLanderContinuous-v2] Average reward of 10 testing episodes: Average ÷ 30 (10%)



```
Step: 454162    Episode: 1192   Length: 199 Total reward: 252.10    Ewma reward: 252.16
Step: 454317    Episode: 1193   Length: 155 Total reward: 277.31    Ewma reward: 253.42
Step: 454592    Episode: 1194   Length: 275 Total reward: 235.01    Ewma reward: 252.50
Step: 454827    Episode: 1195   Length: 235 Total reward: 264.40    Ewma reward: 253.09
Step: 455012    Episode: 1196   Length: 185 Total reward: 270.73    Ewma reward: 253.97
Step: 455205    Episode: 1197   Length: 193 Total reward: 242.55    Ewma reward: 253.40
Step: 455428    Episode: 1198   Length: 223 Total reward: 284.38    Ewma reward: 254.95
Step: 455622    Episode: 1199   Length: 194 Total reward: 233.99    Ewma reward: 253.90
Start Testing
C:\Users\user\Documents\Course\DL\venv\lib\site-packages\gym\core.py:256: DeprecationWar
  deprecation(
Average Reward 264.7390025096876

Process finished with exit code 0
```

- Reference

1. https://www.youtube.com/watch?v=mv0kfiepn3s&ab_channel=NextDayVideo