311551148 lab6 Report

1. DQN result

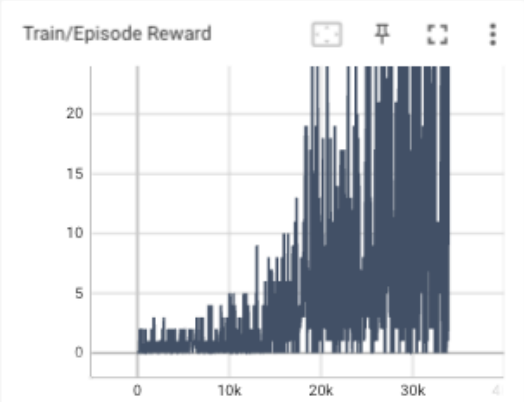
```
Start Testing
Episode: 0      Total reward: 175.08
Episode: 1      Total reward: 198.07
Episode: 2      Total reward: 212.17
Episode: 3      Total reward: 193.85
Episode: 4      Total reward: 234.00
Episode: 5      Total reward: 212.44
Episode: 6      Total reward: 236.50
Episode: 7      Total reward: 209.65
Episode: 8      Total reward: 248.34
Episode: 9      Total reward: 215.50
Average Reward 213.55824876942802
```

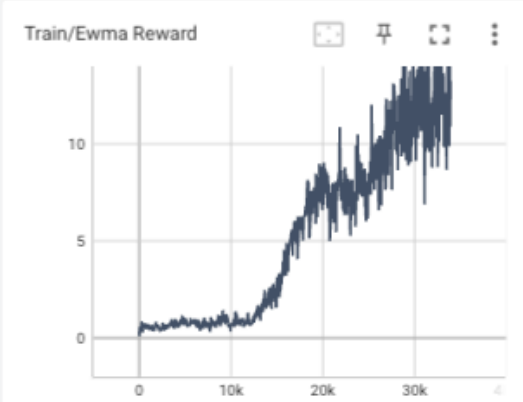
2. DDPG result

```
Start Testing
Episode: 0      Total reward: 289.40
Episode: 1      Total reward: 257.63
Episode: 2      Total reward: 300.38
Episode: 3      Total reward: 253.53
Episode: 4      Total reward: 248.58
Episode: 5      Total reward: 302.22
Episode: 6      Total reward: 255.24
Episode: 7      Total reward: 275.49
Episode: 8      Total reward: 266.07
Episode: 9      Total reward: 251.80
Average Reward 270.0339660135265
(dlp) shlu2240@eva-All-Series:/eva_data2/shlu2240/NCTU_DLP/lab6$
```

3. DQN breakout

```
(dlp) shlu2240@eva-All-Series:/eva_data2/shlu2240/NCTU_DLP/lab6$ python3 dqn_breakout_example.py --test_only
Start Testing
episode 1: 421.00
episode 2: 409.00
episode 3: 425.00
episode 4: 404.00
episode 5: 428.00
episode 6: 399.00
episode 7: 425.00
episode 8: 390.00
episode 9: 399.00
episode 10: 420.00
Average Reward: 412.00
```





Question

1. Describe your major implementation of both DQN and DDPG in detail

甲、Your implementation of Q network updating in DQN.

```
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 = torch.gather(self._behavior_net(state), dim=1, index=action.long())
    with torch.no_grad():
        q_next = torch.max(self._target_net(next_state), dim=1)[0].unsqueeze(dim=1)
        q_target = reward + gamma * q_next * (1 - done)
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)

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

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())
```

first sample a mini-batch of M state transition observations from the replay memory, given i-th observation, the target value is written as follow

$$t_i = \begin{cases} r_i, & \text{if episode terminates at step } i+1 \\ r_i + \gamma \max_{a'} Q_t(s_{i+1}, a'; \theta_t), & \text{otherwise} \end{cases}$$

r is discount factor, Qt is target network, use MSE loss to optimize the behavior network.

乙、Your implementation and the gradient of actor updating in DDPG.

利用 behavior network 的 action network mu，與 critic network Q 可以求出 Q(s,a)，我們想更新 action network mu 使得 Q(s,a)越大越好，定義 loss = -Q(s, mu(s))，backpropagation 時只更新 action network 不更新 critic network

```
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()
```

丙、Your implementation and the gradient of critic updating in DDPG.

利用 target network 所生出來的 q_target 與 behavior network 所生出的 q_value 做 MSE loss，只更新 critic network 不更新 behavior network

```
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()
```

2. Explain effects of the discount factor.
   The discount factor determines the importance of estimated future rewards, i.e., the second term in the second line of equation 10. Smaller discount factor encourages the
   agent focus more on current rewards, and larger discount factor reduces the impact of long-term rewards.
3. Explain benefits of epsilon-greedy in comparison to greedy action selection.
   epsilon -greedy is a simple method that balances exploration and exploitation. At each time step, the agent
   selects an action at random with probability ε and selects the greedy action based on its learned policy
   with probability 1 − epsilon. The reason of adopting ε-greedy action selection is that if we only select the best
   action, there is a chance that we miss a real better one.
4. Explain the necessity of the target network.
   We know that when training DQN and DDPG, the Q-value of each state is estimated by a Q-network
   / critic network. Different from ordinary Q-learning, the exact value function has been replaced by a
   function approximator formed by a deep neural network now. In such manner, Q-values of the same
   state may be different even after a single optimization step on the Q-network, which can possibly make
   the agent's learning process unstable. Hence, using a stable target network should somehow alleviate

this problem. A typical solution is to predict the value of each state by a fixed network, and we only
update its parameters after a pre-defined period.

5. Describe the tricks you used in Breakout and their effects, and how they differ from those used in LunarLander.
   有使用 clip rewrad 以及 frame stack，clip reward 會把 reward 控制在-1 or 0 or 1，縮小離群值對模型的影響，frame stack 會把相連的 4 個 frame concat 在一起，讓我們的模型可以一次看到 4 個 frame，才可以根據球的行徑軌道逕行預測。