Discussion
312551116 黃亭幃

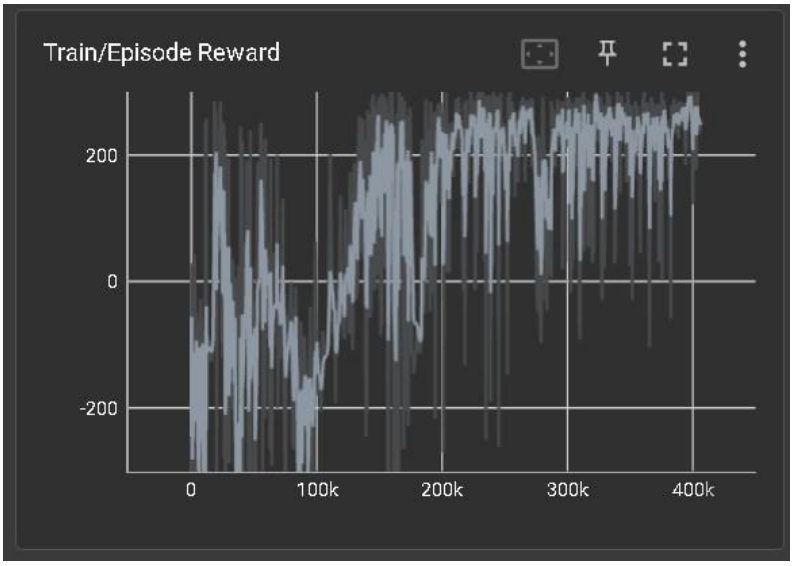
Deep Learning – Lab 4

Deep Q-Network and Deep Deterministic Policy Gradient

I. Experimental Results

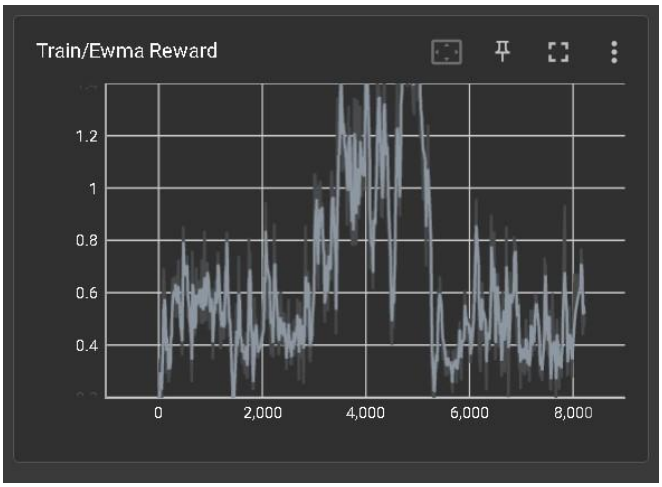
i. DQN_LunarLander:

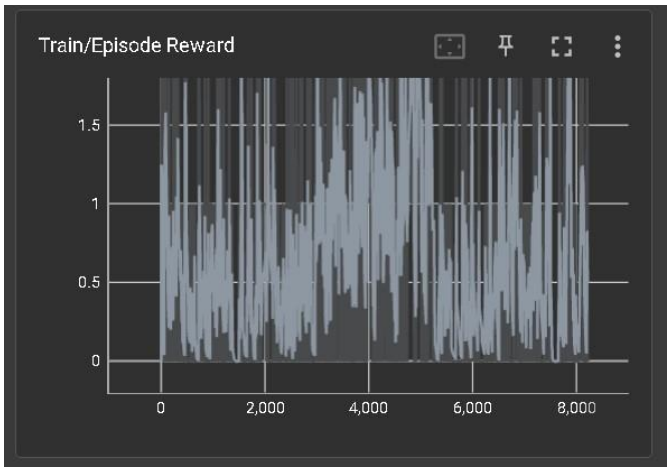
```
Start Testing
Step: 299       Episode: 0      Length: 299     Total reward: 307.99    Epsilon: 0.001
Step: 557       Episode: 1      Length: 258     Total reward: 273.31    Epsilon: 0.001
Step: 770       Episode: 2      Length: 213     Total reward: 235.46    Epsilon: 0.001
Step: 981       Episode: 3      Length: 211     Total reward: 262.84    Epsilon: 0.001
Step: 1226      Episode: 4      Length: 245     Total reward: 271.69    Epsilon: 0.001
Step: 1465      Episode: 5      Length: 239     Total reward: 235.33    Epsilon: 0.001
Step: 1691      Episode: 6      Length: 226     Total reward: 249.52    Epsilon: 0.001
Step: 1904      Episode: 7      Length: 213     Total reward: 272.15    Epsilon: 0.001
Step: 2104      Episode: 8      Length: 200     Total reward: 273.42    Epsilon: 0.001
Step: 2350      Episode: 9      Length: 246     Total reward: 241.72    Epsilon: 0.001
Average Reward 262.34225694518415
```



ii. DQN_BreakoutNoFrameskip:

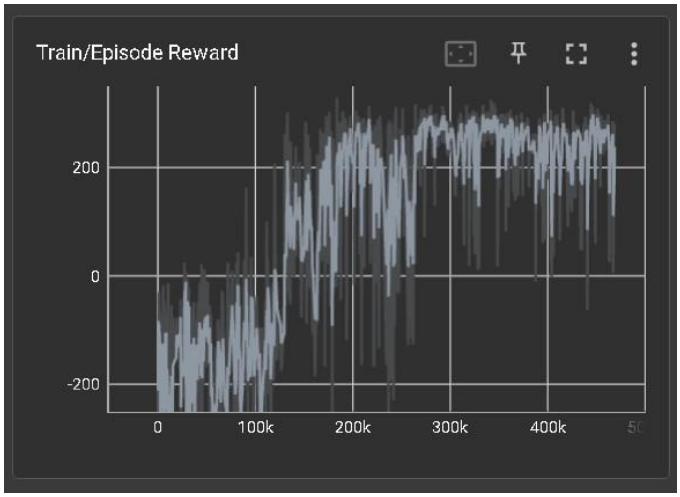
```
episode 1: 113.00
episode 2: 285.00
episode 3: 143.00
episode 4: 162.00
episode 5: 160.00
episode 6: 125.00
episode 7: 113.00
episode 8: 133.00
episode 9: 113.00
episode 10: 80.00
Average Reward: 142.70
```





iii. DDPG_LunarLanderContinuous:

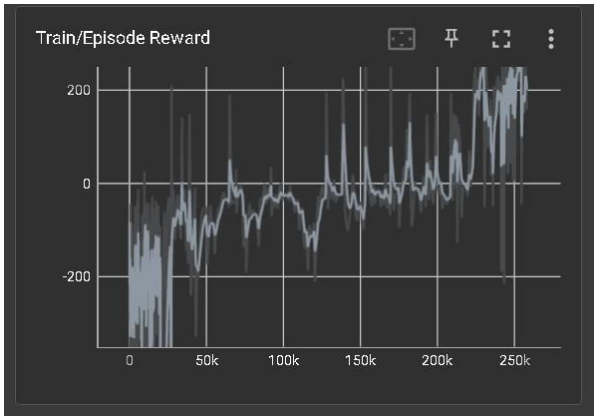
```
Start Testing
episode 0 : 268.65
episode 1 : 263.75
episode 2 : 232.75
episode 3 : 247.84
episode 4 : 255.93
episode 5 : 235.03
episode 6 : 256.17
episode 7 : 281.90
episode 8 : 287.06
episode 9 : 275.85
Average Reward 260.49261739143594
```

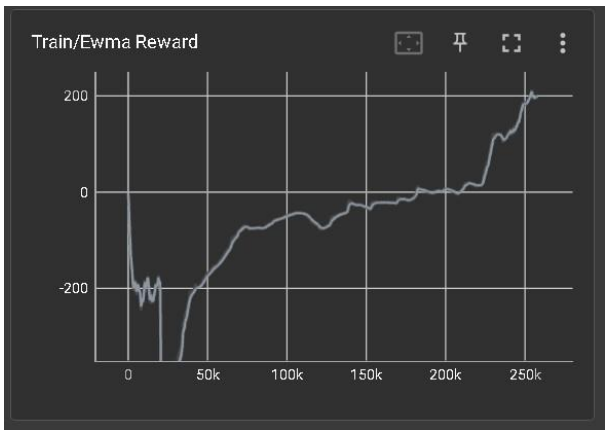


II. Experimental Results of bonus parts (TD3, DDQN):

i. TD3

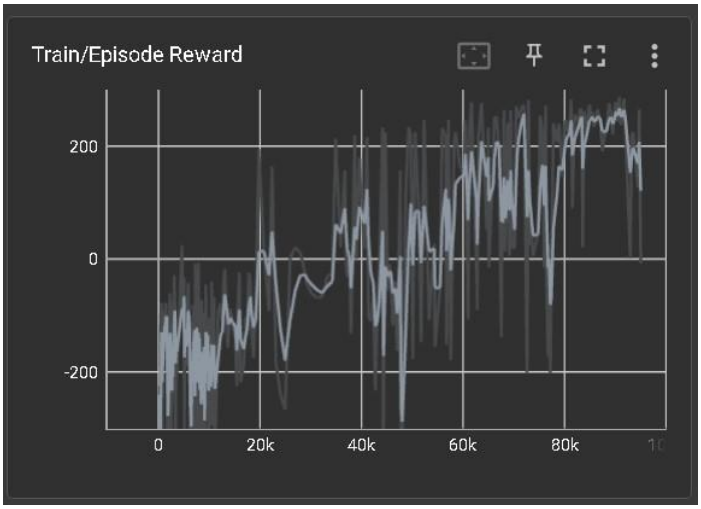
```
Start Testing
episode 0 : 276.39
episode 1 : 275.80
episode 2 : 240.93
episode 3 : 48.70
episode 4 : 266.82
episode 5 : 26.96
episode 6 : 256.06
episode 7 : 237.42
episode 8 : 182.01
episode 9 : 264.18
Average Reward 207.5274527424664
```

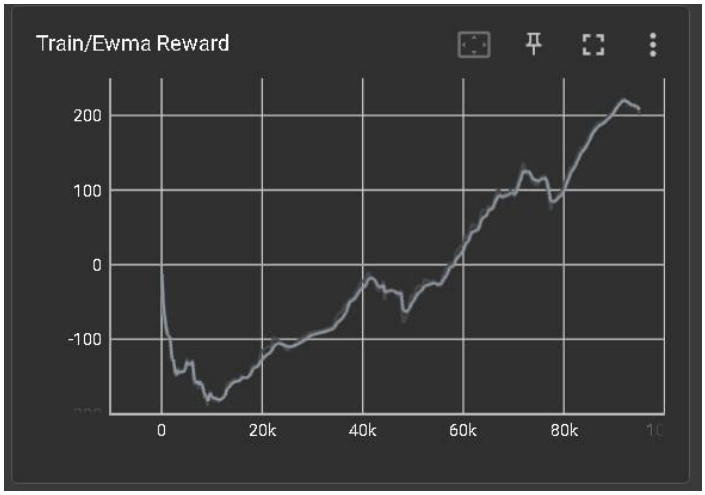




ii. DDQN:

```
Start Testing
Step: 468       Episode: 0      Length: 468     Total reward: 268.64    Epsilon: 0.001
Step: 913       Episode: 1      Length: 445     Total reward: 175.28    Epsilon: 0.001
Step: 1259      Episode: 2      Length: 346     Total reward: 200.36    Epsilon: 0.001
Step: 1549      Episode: 3      Length: 290     Total reward: 250.82    Epsilon: 0.001
Step: 1842      Episode: 4      Length: 293     Total reward: 275.20    Epsilon: 0.001
Step: 2842      Episode: 5      Length: 1000    Total reward: 98.53     Epsilon: 0.001
Step: 3117      Episode: 6      Length: 275     Total reward: 249.08    Epsilon: 0.001
Step: 3451      Episode: 7      Length: 334     Total reward: 251.11    Epsilon: 0.001
Step: 4451      Episode: 8      Length: 1000    Total reward: 153.16    Epsilon: 0.001
Step: 4737      Episode: 9      Length: 286     Total reward: 212.85    Epsilon: 0.001
Average Reward 213.5025620384127
```





III. Question

i. Describe your major implementation of both DQN and DDPG in detail. Your description should at least contain three parts:

➢ (1) Your implementation of Q network updating in DQN.

```
def _update_behavior_network(self, gamma):
    state, action, reward, next_state, is_done = self._memory.sample(self.batch_size, self.device)
    q_value = self._behavior_net(state).gather(dim = 1, index = action.long())

    with torch.no_grad():
        #q_next= self._target_net(next_state).argmax(dim = 1).item()
        q_next= torch.max(self._target_net(next_state), 1)[0]
        q_next = q_next.unsqueeze(1)
        q_target = reward + gamma * q_next * (1 - is_done)

    loss = self._criterion(q_value, q_target)
    self._optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
    self._optimizer.step()

def _update_target_network(self):
    self._target_net.load_state_dict(self._behavior_net.state_dict())
```

- Here is my implementation of updating Q network.
- First, I will get the Qvalue of the action in this step. And then I will calculate the V value in the next state. In DQN, the next V value is assume as the maximum Q value of all the action in next step. In this part, Q value is calculate by the target network. And final we will use the reward plus V value(* gamma for discount factor) to be the expect Q value(Qtarget).
- And then we update the behavior by MSE(Qvalue, Qtarget)

➢ (2) Your implementation and the gradient of actor updating in DDPG.

```
def _update_behavior_network(self, gamma):
    actor_net, critic_net, target_actor_net, target_critic_net = \
    self._actor_net, self._critic_net, self._target_actor_net, self._target_critic_net
    actor_opt, critic_opt = self._actor_opt, self._critic_opt

    state, action, reward, next_state, done = self._memory.sample(self.batch_size, self.device)

    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)

    critic_loss = self._criterion(q_value, q_target)
    # raise NotImplementedError
    # optimize critic
    actor_net.zero_grad()
    critic_net.zero_grad()
    critic_loss.backward()
    critic_opt.step()

    action = self._actor_net(state)
    actor_loss = -self._critic_net(state,action).mean()
    actor_net.zero_grad()
    critic_net.zero_grad()
    actor_loss.backward()
    actor_opt.step()

@staticmethod
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()):
        target.data.copy_((1-tau)*target.data + tau*behavior.data)
```

Update the actor policy using the sampled gradient:

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

- **The main idea is to calculate the Q-value through the updated critic. However, since we want the Q-value to be as large as possible, we minimize the negative of the Q-value.**

➢ (3) Your implementation and the gradient of critic updating in DDPG.

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

- **First, I will calculate Q-value by the critic network. In theis time, critic will use state and action to be the input. That is, It will output just one action Qvalue(QDN will output all Qvalue, and we choose largest Qvalue action).**
  - And the just like DQN, use MSE to update it.

ii. Explain effects of the discount factor.

- **The significance of the discount factor lies in the emphasis placed on future rewards. Assuming a low discount factor, it indicates that the model focuses more on the immediate rewards generated by current actions. Conversely, a higher discount factor reflects a greater concern for expected future rewards.**

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

- **The significance of epsilon-greedy is to prevent the model from constantly being in a greedy state. We provide it with a certain chance to explore randomly. This avoids situations where certain paths are never taken, causing the model to not update its expectations of potentially higher rewards that it might not have encountered before.**

iv. Explain the necessity of the target network.

- **During the process of adjusting the neural network (NN) to align the Q-value with the Q-target, it simultaneously affects the computation of the Q-target (as they share the same NN). To address this, a target network is used to compute the Q-target separately from the value network, aiming to minimize interference between target and value calculations. And later, the parameters from the training network are copied to the target network. This approach enhances the stability of network training.**

v. Describe the tricks you used in Breakout and their effects, and how they differ from those used in LunarLander.

- **The primary and most significant difference lies in the LunarLander game, where the**

model can determine actions based solely on the current state. However, in Breakout, the state consists of an image, which alone does not convey information about the ball's movement, making it unsuitable for direct training.

- **To address this, I grouped every five frames into a single state for Breakout. During sampling, the first four frames serve as the current state, and the subsequent four frames are treated as the next state. This approach enables the model to utilize a sequence of consecutive frames for state assessment.**