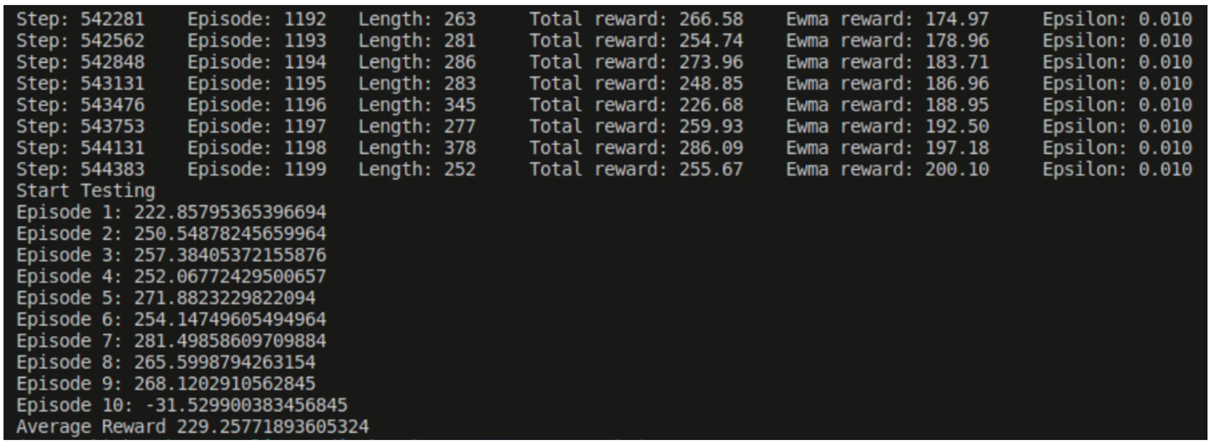
Lab 5 Reinforcement learning

411551007 林穎志

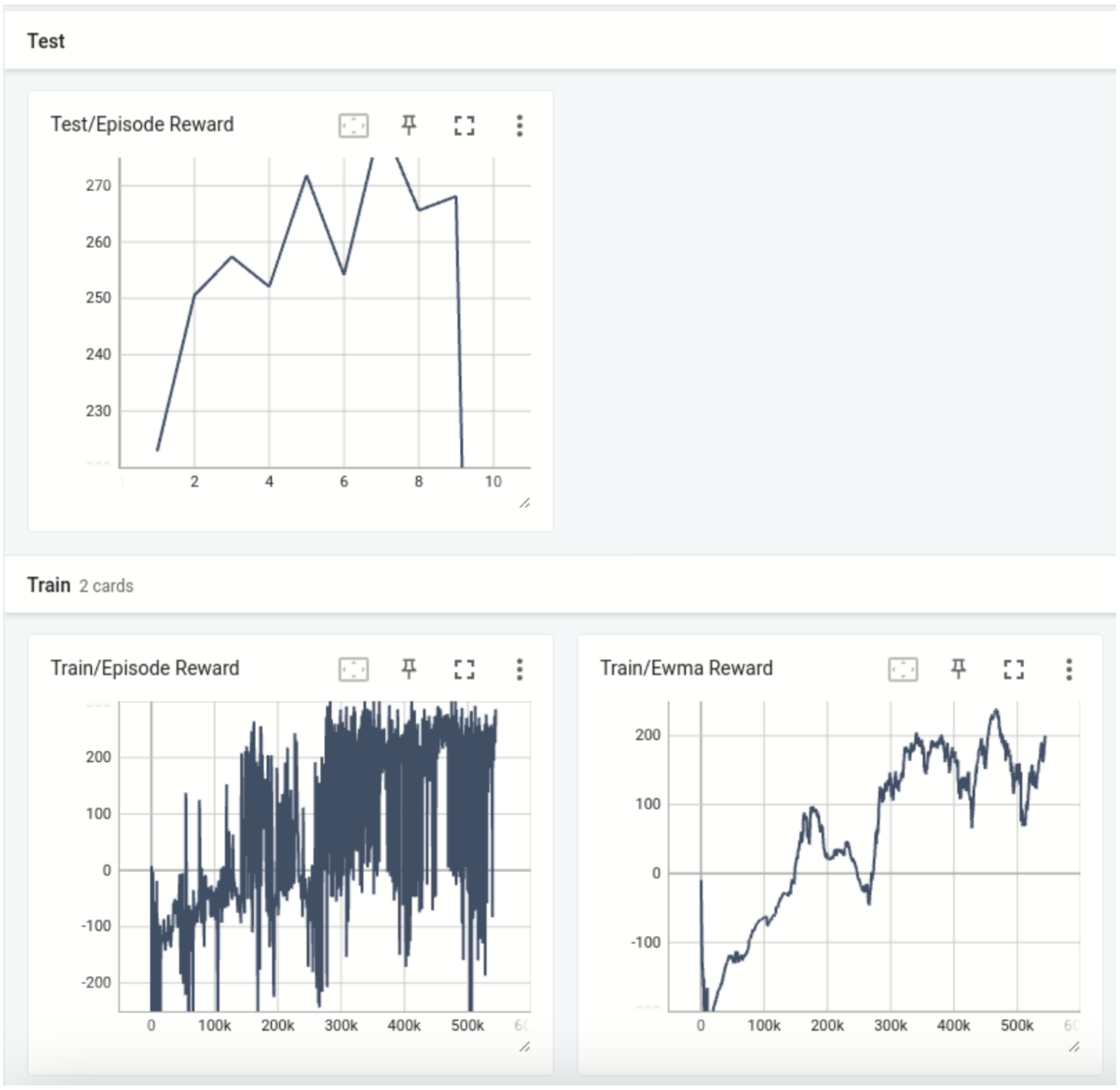
1. Experimental Results

A. DQN (LunarLander)

a. Result

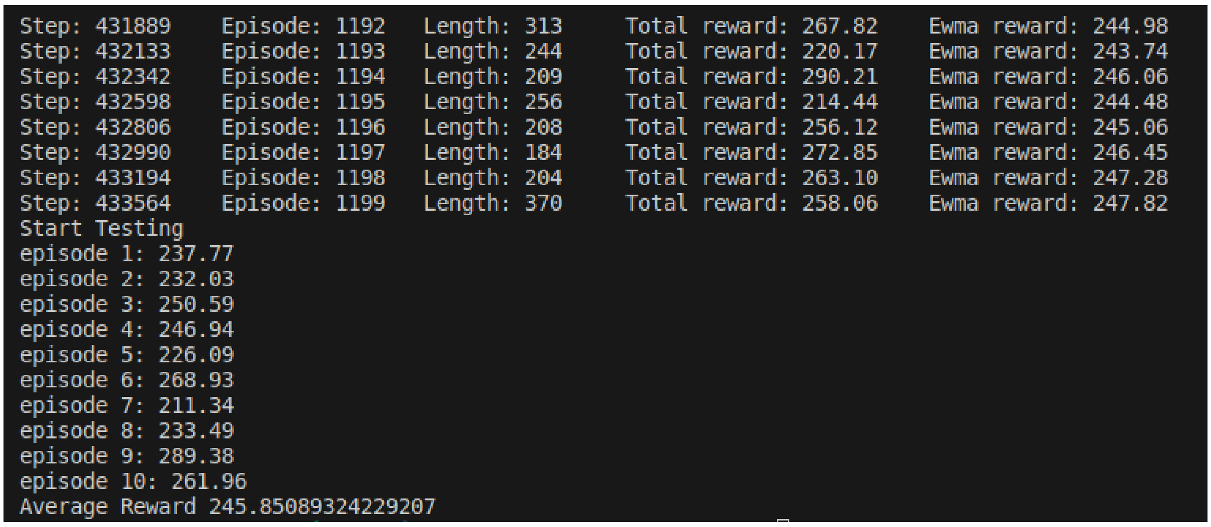


```
Step: 542281    Episode: 1192   Length: 263     Total reward: 266.58    Ewma reward: 174.97     Epsilon: 0.010
Step: 542562    Episode: 1193   Length: 281     Total reward: 254.74    Ewma reward: 178.96     Epsilon: 0.010
Step: 542848    Episode: 1194   Length: 286     Total reward: 273.96    Ewma reward: 183.71     Epsilon: 0.010
Step: 543131    Episode: 1195   Length: 283     Total reward: 248.85    Ewma reward: 186.96     Epsilon: 0.010
Step: 543476    Episode: 1196   Length: 345     Total reward: 226.68    Ewma reward: 188.95     Epsilon: 0.010
Step: 543753    Episode: 1197   Length: 277     Total reward: 259.93    Ewma reward: 192.50     Epsilon: 0.010
Step: 544131    Episode: 1198   Length: 378     Total reward: 286.09    Ewma reward: 197.18     Epsilon: 0.010
Step: 544383    Episode: 1199   Length: 252     Total reward: 255.67    Ewma reward: 200.10     Epsilon: 0.010
Start Testing
Episode 1: 222.85795365396694
Episode 2: 250.54878245659964
Episode 3: 257.38405372155876
Episode 4: 252.06772429500657
Episode 5: 271.8823229822094
Episode 6: 254.14749605494964
Episode 7: 281.49858609709884
Episode 8: 265.5998794263154
Episode 9: 268.1202910562845
Episode 10: -31.529900383456845
Average Reward 229.25771893605324
```

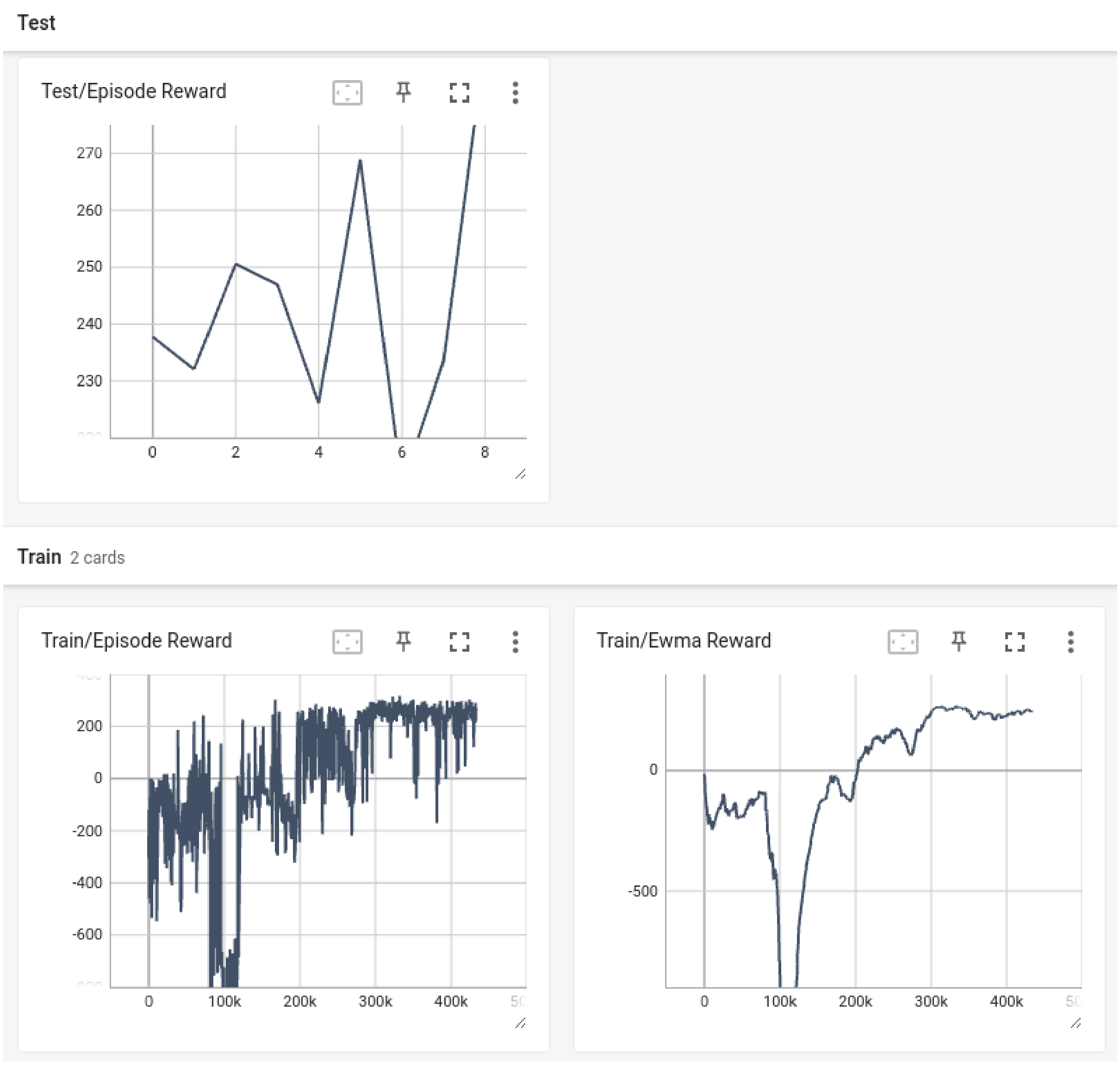


B. DDPG (LunarLander)

a. Result

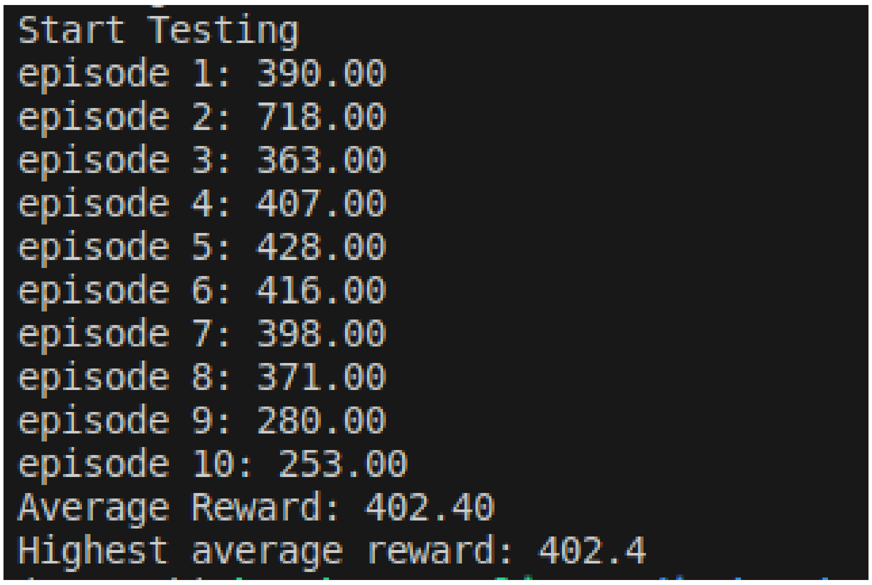


```
Step: 431889    Episode: 1192   Length: 313     Total reward: 267.82    Ewma reward: 244.98
Step: 432133    Episode: 1193   Length: 244     Total reward: 220.17    Ewma reward: 243.74
Step: 432342    Episode: 1194   Length: 209     Total reward: 290.21    Ewma reward: 246.06
Step: 432598    Episode: 1195   Length: 256     Total reward: 214.44    Ewma reward: 244.48
Step: 432806    Episode: 1196   Length: 208     Total reward: 256.12    Ewma reward: 245.06
Step: 432990    Episode: 1197   Length: 184     Total reward: 272.85    Ewma reward: 246.45
Step: 433194    Episode: 1198   Length: 204     Total reward: 263.10    Ewma reward: 247.28
Step: 433564    Episode: 1199   Length: 370     Total reward: 258.06    Ewma reward: 247.82
Start Testing
episode 1: 237.77
episode 2: 232.03
episode 3: 250.59
episode 4: 246.94
episode 5: 226.09
episode 6: 268.93
episode 7: 211.34
episode 8: 233.49
episode 9: 289.38
episode 10: 261.96
Average Reward 245.85089324229207
```

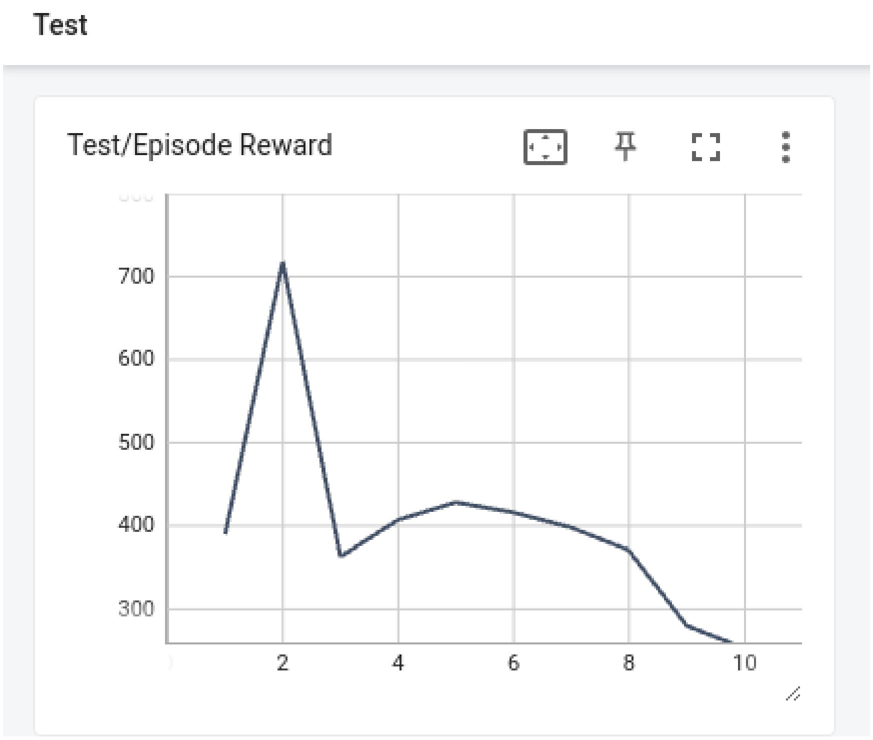


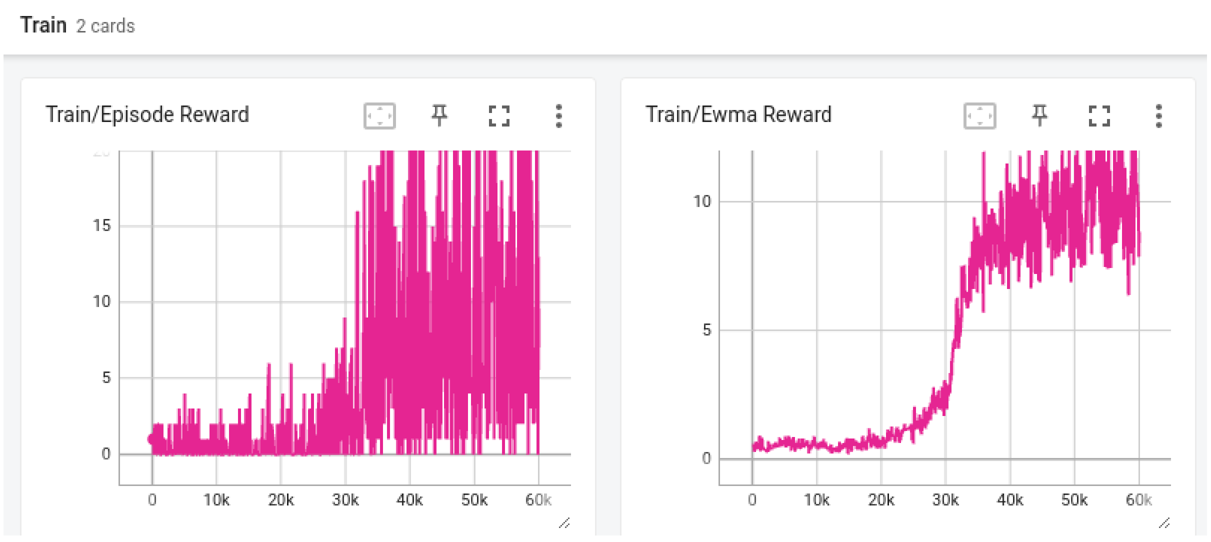
C. DQN (Breakout)

a. Result



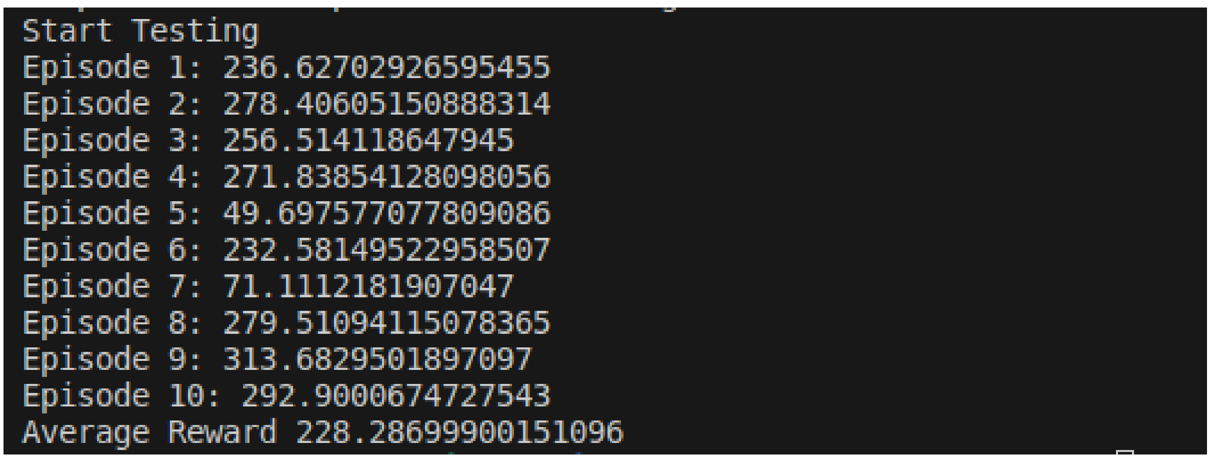
```
Start Testing
episode 1: 390.00
episode 2: 718.00
episode 3: 363.00
episode 4: 407.00
episode 5: 428.00
episode 6: 416.00
episode 7: 398.00
episode 8: 371.00
episode 9: 280.00
episode 10: 253.00
Average Reward: 402.40
Highest average reward: 402.4
```



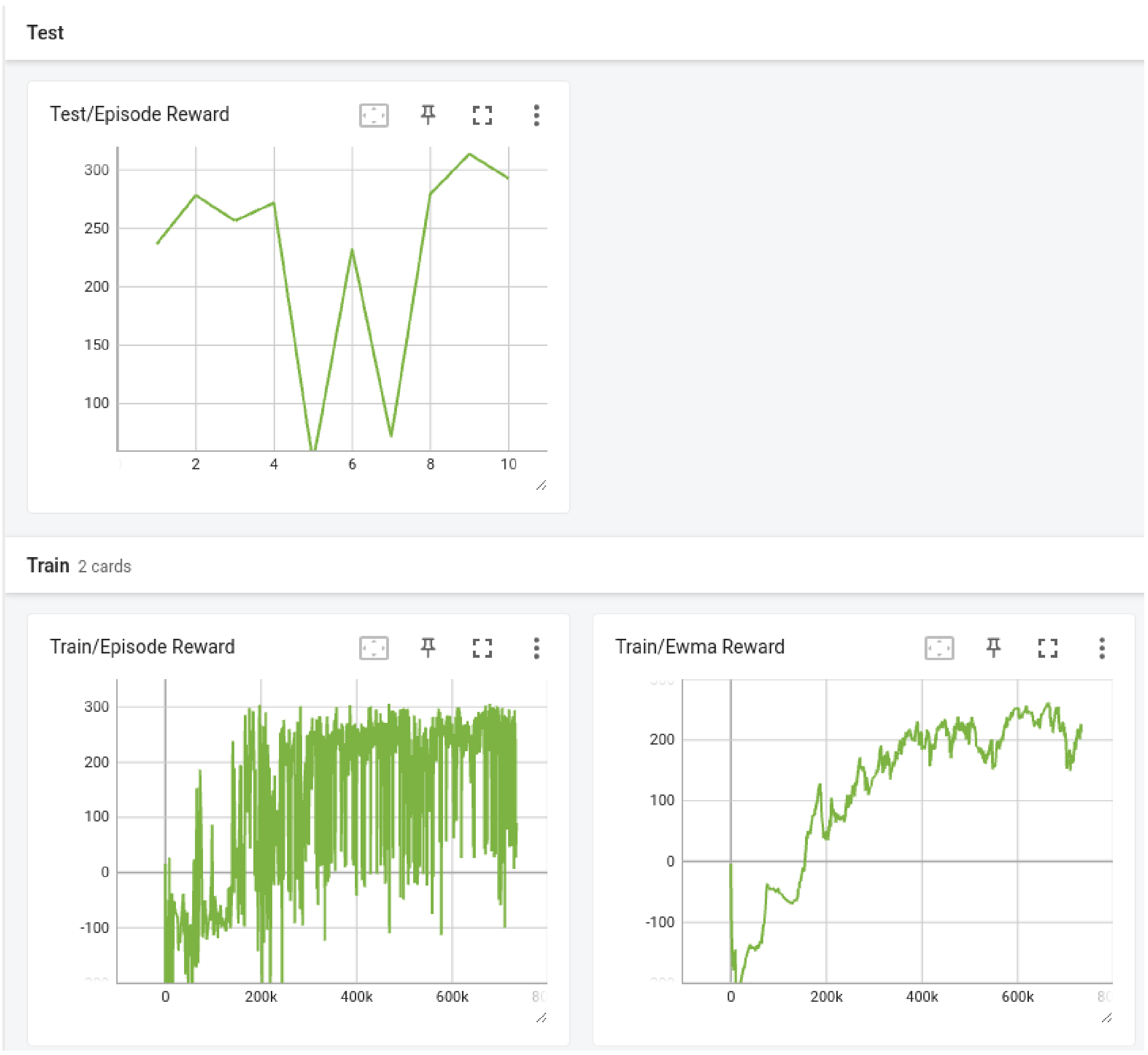


2.DDQN (Bonus)

A. Result



```
Start Testing
Episode 1: 236.62702926595455
Episode 2: 278.40605150888314
Episode 3: 256.514118647945
Episode 4: 271.83854128098056
Episode 5: 49.697577077809086
Episode 6: 232.58149522958507
Episode 7: 71.1112181907047
Episode 8: 279.51094115078365
Episode 9: 313.6829501897097
Episode 10: 292.9000674727543
Average Reward 228.28699900151096
```



3. Questions (10%)

A. 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.

1. We design a 3-layer net which input dimension is 8 and output dimension is 4.

```
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=32):
        super().__init__()
        ## TODO ##
        self.block1 = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
        )

        self.block2 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
        )

        self.block3 = nn.Sequential(
            nn.Linear(hidden_dim, action_dim),
        )
```

2. We use epsilon-greedy strategy to select action. If random number less than the epsilon, we will random sample an action, otherwise we will use behavior net to choose which action can get the maximum expected Q value.

```
def select_action(self, state, epsilon, action_space):
        '''epsilon-greedy based on behavior network'''
        ## TODO ##
        rand = random.random()
        if rand < epsilon:
            action = action_space.sample()
        else:
            with torch.no_grad():
                state = torch.Tensor(state).view(1, -1).to(self.device)
                action = torch.argmax(self._behavior_net(state), 1).item()

        return action
```

3. We will sample random minibatch of transition (ϕj , aj , rj , ϕj+1) from replay memory.And compute target value yj .

```
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).max(dim=1).values.view(-1, 1)
            q_target = reward + gamma * q_next * (1.0 - done)

        criterion = nn.SmoothL1Loss()
        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()
```

4. Copy weight of behavior net to update target network every C steps.

```
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())
```

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

The function of actor is to output an action that can gain the maximum Q value when inputting into critic. Hence, we gave a negative value to the loss to let it do gradient ascend.

```
## update actor ##
# actor loss
## TODO ##
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()
```

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

To find the a_{t+1} from s_{t+1} by target actor net. Then get the q_{t+1} from target critic net and calculate the q_{target}. Last, we calculate the SmoothL1Loss between q value and q target.

```
# critic loss
## TODO ##
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 * (1.0 - done)
```

B. Explain effects of the discount factor.

A: The discount factor reflects on short-term / long-term rewards:

- **High Discount Factor (γ close to 1)**: When γ is close to 1, the agent heavily values long-term rewards. It prioritizes maximizing cumulative rewards over the entire episode or task horizon. In this case, the agent is willing to make sacrifices in the short term to achieve greater rewards in the long term.
- **Low Discount Factor (γ close to 0)**: When γ is close to 0, the agent prioritizes short-term rewards and is myopic in its decision-making. It focuses on immediate gains and does not consider future consequences as much. This can lead to a more opportunistic, short-sighted strategy.

C. (1%) Explain benefits of epsilon-greedy in comparison to greedy action selection.

A: Greedy action selection always selects the action with the highest estimated value (exploitation), which can cause the agent to get stuck in suboptimal policies if it doesn't explore other actions. Epsilon-greedy, on the other hand, allows the agent to choose a random action (exploration) with a small probability (epsilon, ε), ensuring that all actions are considered over time.

D. (1%) Explain the necessity of the target network.

A: The target network is necessary because it can makes the model train stably and it plays a crucial role in achieving better convergence and learning in deep RL. During training, as the Q-values are updated, the target Q-values also change. This leads to a moving target problem, where the target Q-values constantly shift, making it difficult for the learning process to stabilize.

**E. (2%) Describe the tricks you used in Breakout and their effects, and how they differ
from those used in LunarLander.**

A: The difference between Breakout and LunarLander is the input, the input of Breakout include 4 frames and the LunarLander has just one frame.