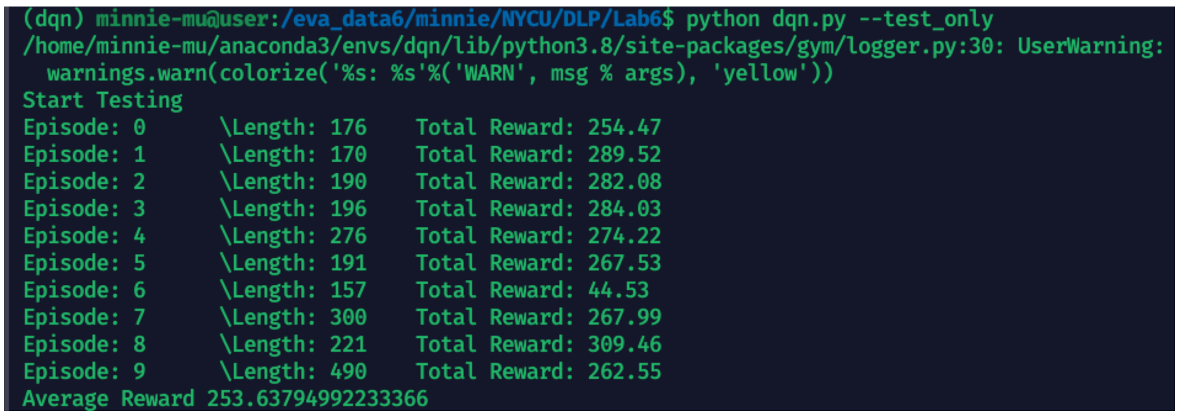
DLP Lab6 Report

▼ 資科工碩 穆冠蓁

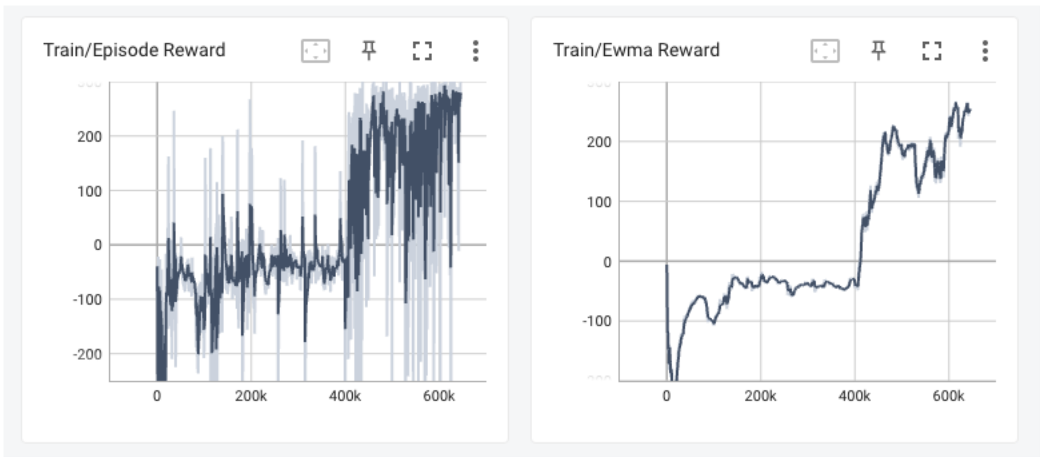
Experiment Result

- LunarLander-v2
    - Testing Result

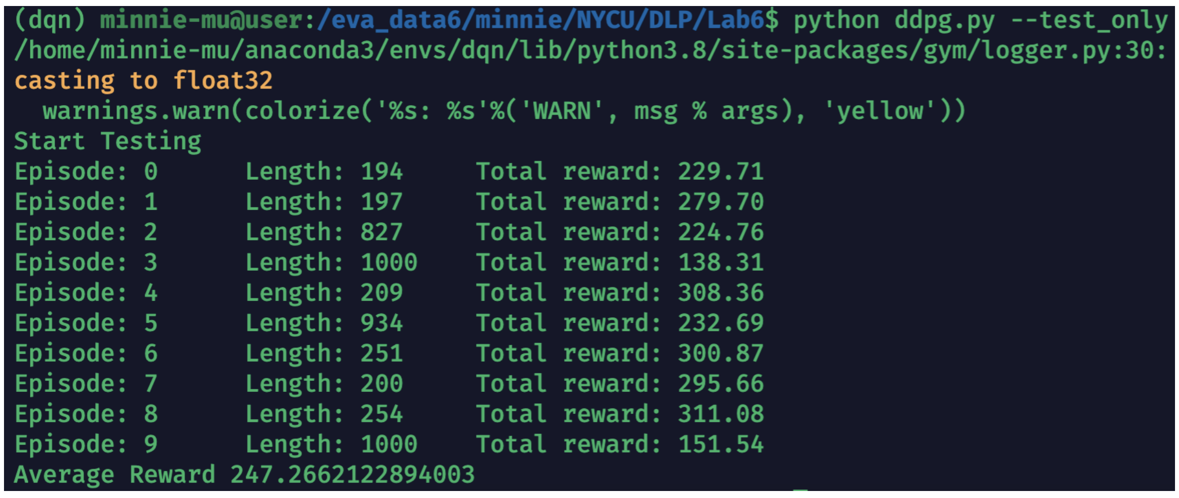


```
(dqn) minnie-mu@user:/eva_data6/minnie/NYCU/DLP/Lab6$ python dqn.py --test_only
/home/minnie-mu/anaconda3/envs/dqn/lib/python3.8/site-packages/gym/logger.py:30: UserWarning:
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode: 0      \Length: 176    Total Reward: 254.47
Episode: 1      \Length: 170    Total Reward: 289.52
Episode: 2      \Length: 190    Total Reward: 282.08
Episode: 3      \Length: 196    Total Reward: 284.03
Episode: 4      \Length: 276    Total Reward: 274.22
Episode: 5      \Length: 191    Total Reward: 267.53
Episode: 6      \Length: 157    Total Reward: 44.53
Episode: 7      \Length: 300    Total Reward: 267.99
Episode: 8      \Length: 221    Total Reward: 309.46
Episode: 9      \Length: 490    Total Reward: 262.55
Average Reward 253.63794992233366
```

    - Tensorboard

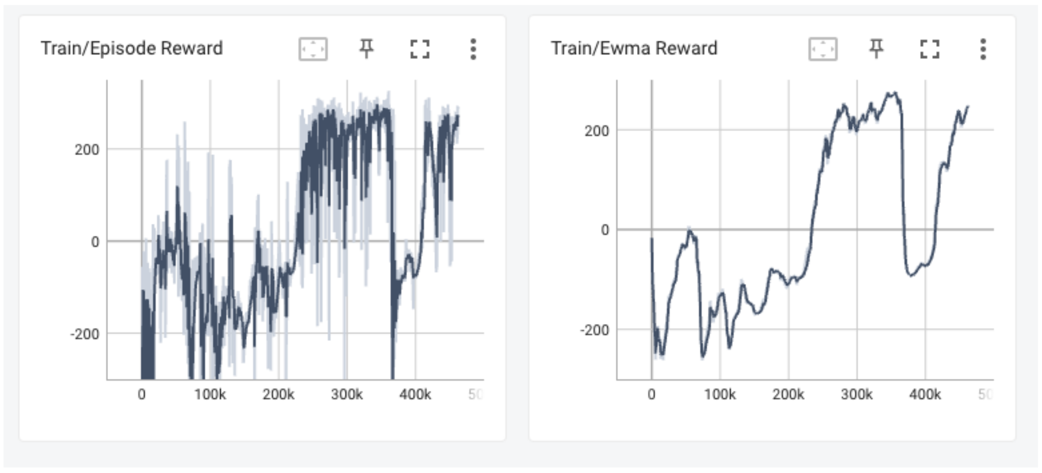


- LunarLander-Continuous
    - Testing Result

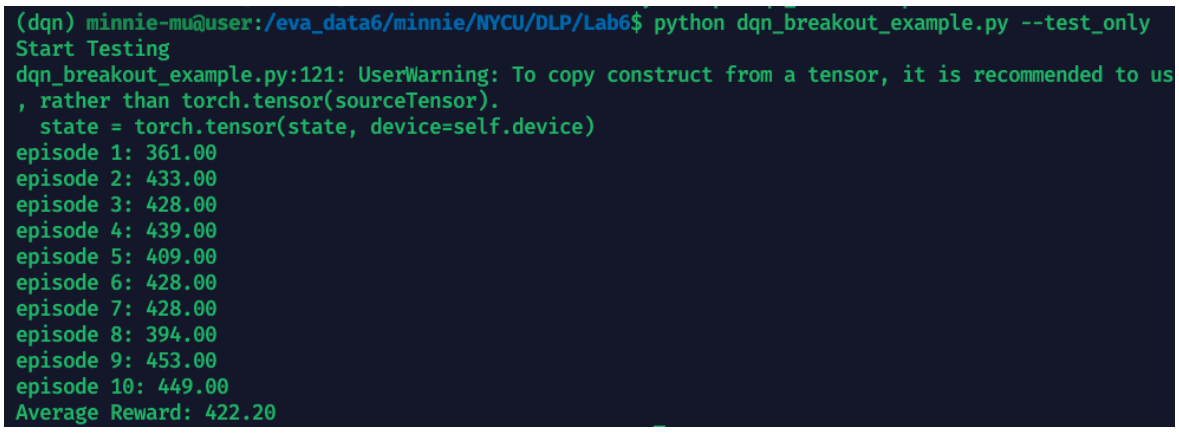


```
(dqn) minnie-mu@user:/eva_data6/minnie/NYCU/DLP/Lab6$ python ddpg.py --test_only
/home/minnie-mu/anaconda3/envs/dqn/lib/python3.8/site-packages/gym/logger.py:30:
casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode: 0      Length: 194     Total reward: 229.71
Episode: 1      Length: 197     Total reward: 279.70
Episode: 2      Length: 827     Total reward: 224.76
Episode: 3      Length: 1000    Total reward: 138.31
Episode: 4      Length: 209     Total reward: 308.36
Episode: 5      Length: 934     Total reward: 232.69
Episode: 6      Length: 251     Total reward: 300.87
Episode: 7      Length: 200     Total reward: 295.66
Episode: 8      Length: 254     Total reward: 311.08
Episode: 9      Length: 1000    Total reward: 151.54
Average Reward 247.2662122894003
```

    - Tensorboard

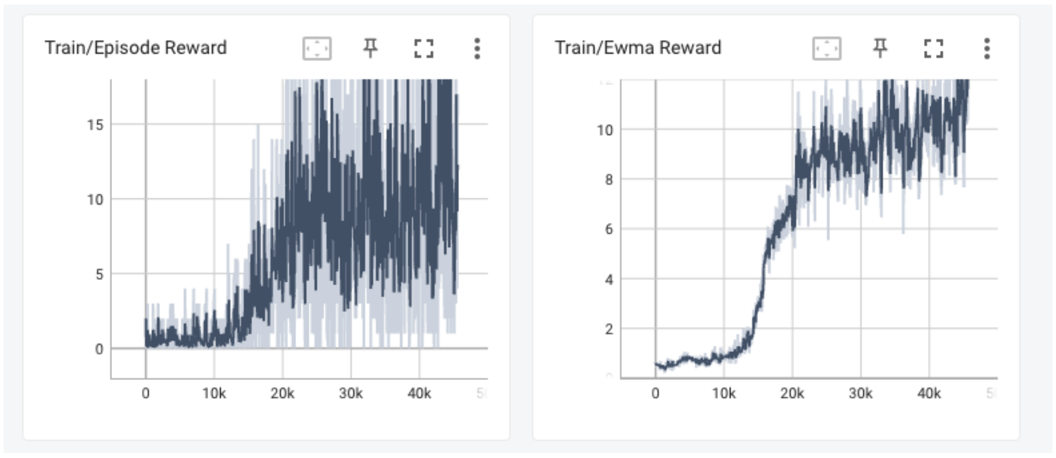


- BreakoutNoFrameskip-v4
  - Testing Result

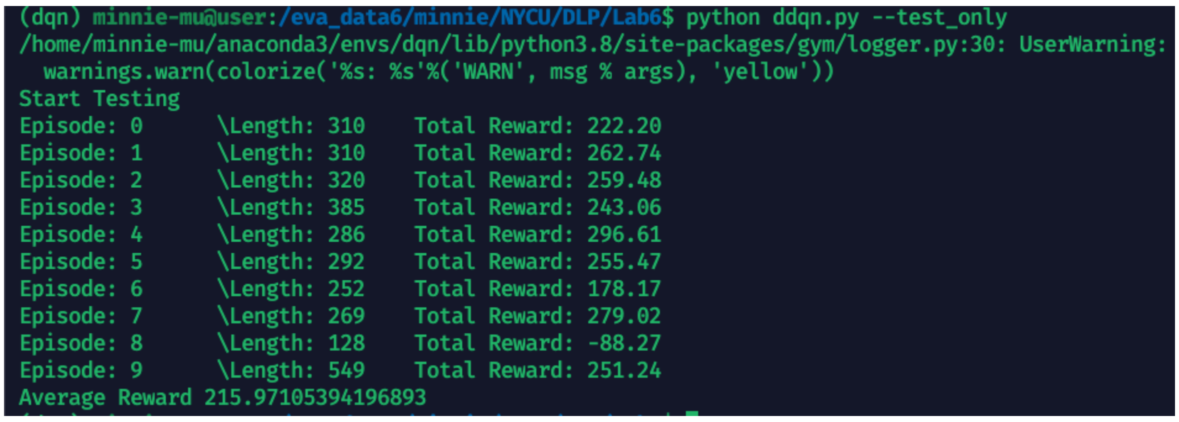


```
(dqn) minnie-mu@user:/eva_data6/minnie/NYCU/DLP/Lab6$ python dqn_breakout_example.py --test_only
Start Testing
dqn_breakout_example.py:121: UserWarning: To copy construct from a tensor, it is recommended to us
, rather than torch.tensor(sourceTensor).
  state = torch.tensor(state, device=self.device)
episode 1: 361.00
episode 2: 433.00
episode 3: 428.00
episode 4: 439.00
episode 5: 409.00
episode 6: 428.00
episode 7: 428.00
episode 8: 394.00
episode 9: 453.00
episode 10: 449.00
Average Reward: 422.20
```

  - Tensorboard

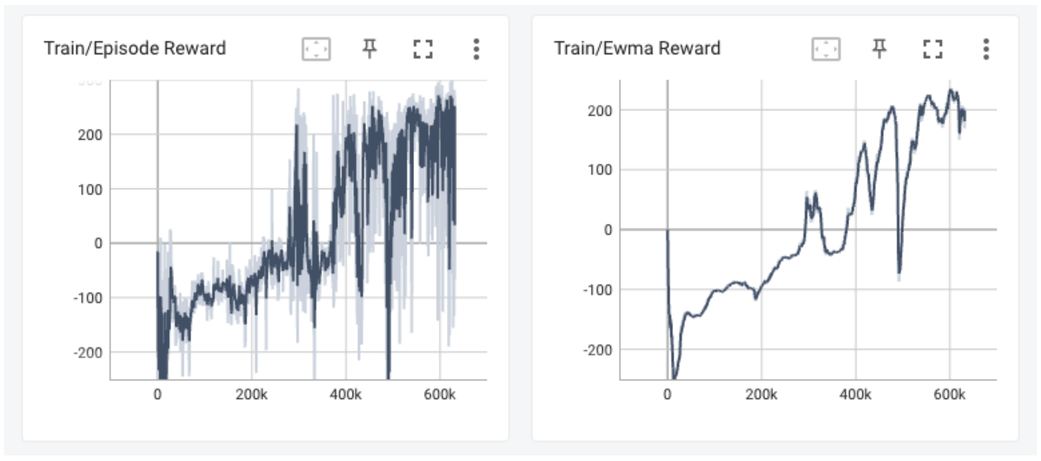


- Experimental Results of bonus parts (DDQN)
  - Testing Result



```
(dqn) minnie-mu@user:/eva_data6/minnie/NYCU/DLP/Lab6$ python ddqn.py --test_only
/home/minnie-mu/anaconda3/envs/dqn/lib/python3.8/site-packages/gym/logger.py:30: UserWarning:
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode: 0      \Length: 310    Total Reward: 222.20
Episode: 1      \Length: 310    Total Reward: 262.74
Episode: 2      \Length: 320    Total Reward: 259.48
Episode: 3      \Length: 385    Total Reward: 243.06
Episode: 4      \Length: 286    Total Reward: 296.61
Episode: 5      \Length: 292    Total Reward: 255.47
Episode: 6      \Length: 252    Total Reward: 178.17
Episode: 7      \Length: 269    Total Reward: 279.02
Episode: 8      \Length: 128    Total Reward: -88.27
Episode: 9      \Length: 549    Total Reward: 251.24
Average Reward 215.97105394196893
```

  - Tensorboard



- Questions
  - 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.

   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):
  if random.random() < epsilon:
      return action_space.sample()

  with torch.no_grad():
      state = torch.tensor(state, device=self.device).view(1, -1)
      outputs = self._behavior_net(state)
      _, best_action = torch.max(outputs, 1)
      return best_action.item()
```

   We will sample random minibatch of transition $(\phi_j, a_j, r_j, \phi_{j+1})$ from replay memory.And compute target value y_j.

```
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)
  q_value = self._behavior_net(state).gather(1, action.long())
  with torch.no_grad():
     q_next = torch.max(self._target_net(next_state), 1)[0].view(-1, 1)
     q_target = reward + q_next * gamma * (1.0 - 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()
```

   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'''
  self._target_net.load_state_dict(self._behavior_net.state_dict())
```

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

   We want to let behavior actor network predict an action which can get maximum q value by critic network.Use actor network to generate actions, and then get q value by critic network.Compute the mean q value and back-propogation to update actor network.Therefore we define actor loss as follows.

   - s : current state
   - Q : behavior critic network.
   - u : behavior actor network

$$\text{Actor Loss} = -Q(s, u(s))$$

   Update the actor policy using the sampled gradient:

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

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

   First sample random minibatch from replay memory, and use actions of batch to calculate corresponding q value by critic network.We also need to calculate target value.First, get action of next state by target actor network, and then generate q value of next state by target critic network.Multiply discount factor and plus reward, we can get the target q value.

   Set $y - i = \gamma_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$ Update critic by minimizeing the loss; $L = \frac{1}{N}\sum(y_i - Q(s_i, a_i|\theta^Q))^2$

4. Update behavior network: We will sample random minibatch(s_i, a_i, r_i, s_{i+1}) from replay memory.And compute target value y_i, the formulation is as follows.
   - Q: behavior critic network
   - u: behavior critic network
   - $\hat{Q}$: target critic network
   - $\hat{u}$: target actor network
   - γ: discount factor
   - $y_i = r_i + \gamma * \hat{Q}(s_{i+1} + \hat{u}(s_{i+1}))$

   And use mean square error as loss function to update behavior critic network.

$$\text{MSE Loss} = \frac{1}{N}\sum(y_i - Q(s_i, a_i))^2$$

   And update behavior actor network by following formulation.

$$\text{Actor Loss} = -\frac{1}{N}\sum(Q(s_i), u(s_i))$$

```
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._tar
  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)

  ## update critic ##
  # critic loss
  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)
  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()

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

- Explain effects of the discount factor.

  By the Q learning algorithm,γ is the discount factor.the smaller discount factor is, the agent will focus on current reward.The larger discount factor is ,the agent will focus on future reward.

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_{a'} Q(S', a') - Q(S, A))$$

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

  The model choose the action which is the best for current state, but the result is not necessarily the best for the future. So, to let model explore other action, we use epsilon-greedy to set a probability ε which can let model to randomly choose other actions.

- Explain the necessity of the target network.

  If we don't have target network, the loss is calculated with behavior q value and behavior target q value. This may cause difficulty for training. So, to avoid this situation, we need target network and copy weights to target network from behavior network every constant steps.

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

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