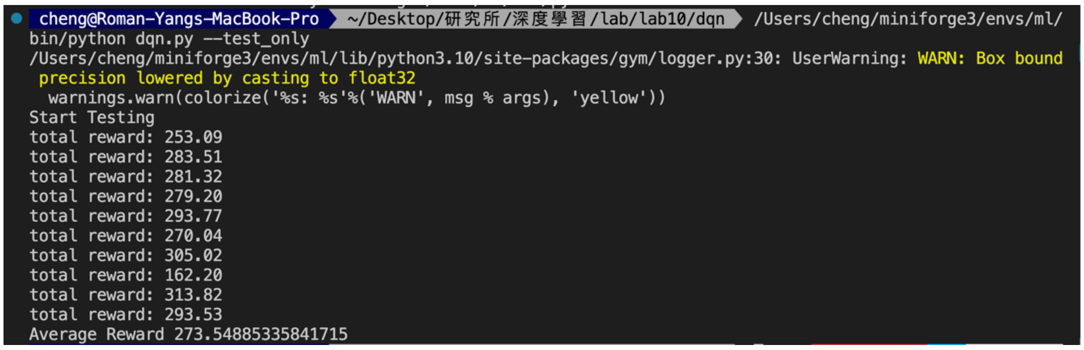
311551180_楊佳誠

Experimental Results

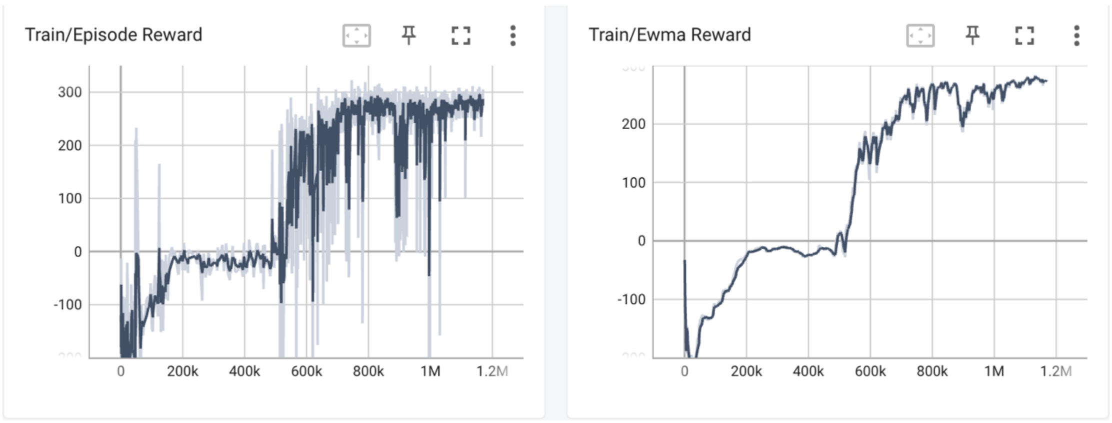
LunarLander-v2 - DQN

- Testing results



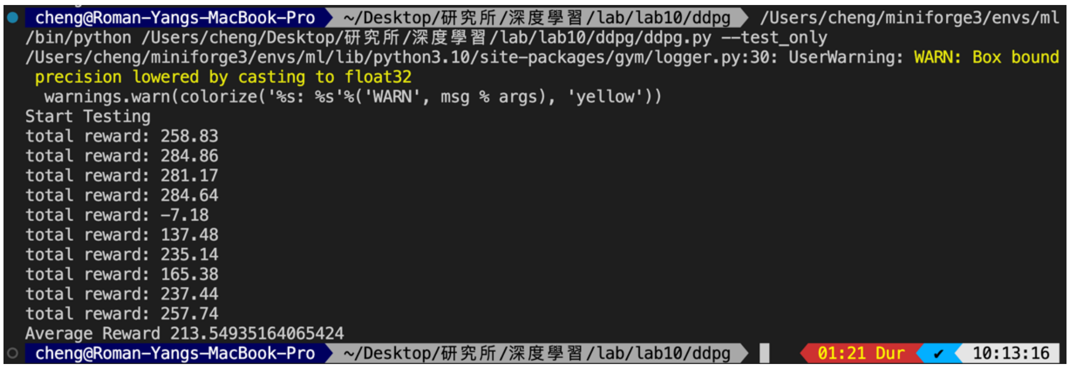
```
cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/研究所/深度學習/lab/lab10/dqn  /Users/cheng/miniforge3/envs/ml/
bin/python dqn.py --test_only
/Users/cheng/miniforge3/envs/ml/lib/python3.10/site-packages/gym/logger.py:30: UserWarning: WARN: Box bound
 precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
total reward: 253.09
total reward: 283.51
total reward: 281.32
total reward: 279.20
total reward: 293.77
total reward: 270.04
total reward: 305.02
total reward: 162.20
total reward: 313.82
total reward: 293.53
Average Reward 273.54885335841715
```

- Tensorboard



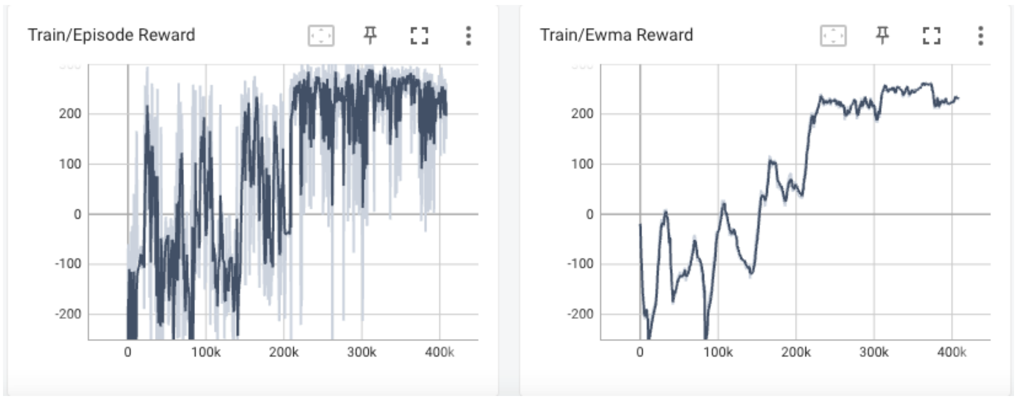
LunarLanderContinuous-v2 - DDPG

- Testing results



```
cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/研究所/深度學習/lab/lab10/ddpg  /Users/cheng/miniforge3/envs/ml
/bin/python /Users/cheng/Desktop/研究所/深度學習/lab/lab10/ddpg/ddpg.py --test_only
/Users/cheng/miniforge3/envs/ml/lib/python3.10/site-packages/gym/logger.py:30: UserWarning: WARN: Box bound
 precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
total reward: 258.83
total reward: 284.86
total reward: 281.17
total reward: 284.64
total reward: -7.18
total reward: 137.48
total reward: 235.14
total reward: 165.38
total reward: 237.44
total reward: 257.74
Average Reward 213.54935164065424
cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/研究所/深度學習/lab/lab10/ddpg                01:21 Dur  ✔  10:13:16
```

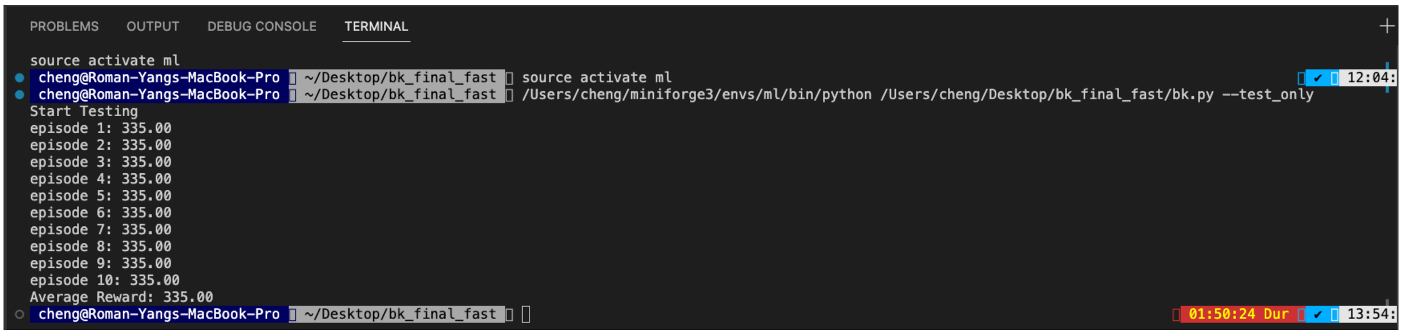
- Tensorboard



BreakoutNoFrameskip-v4 - DQN

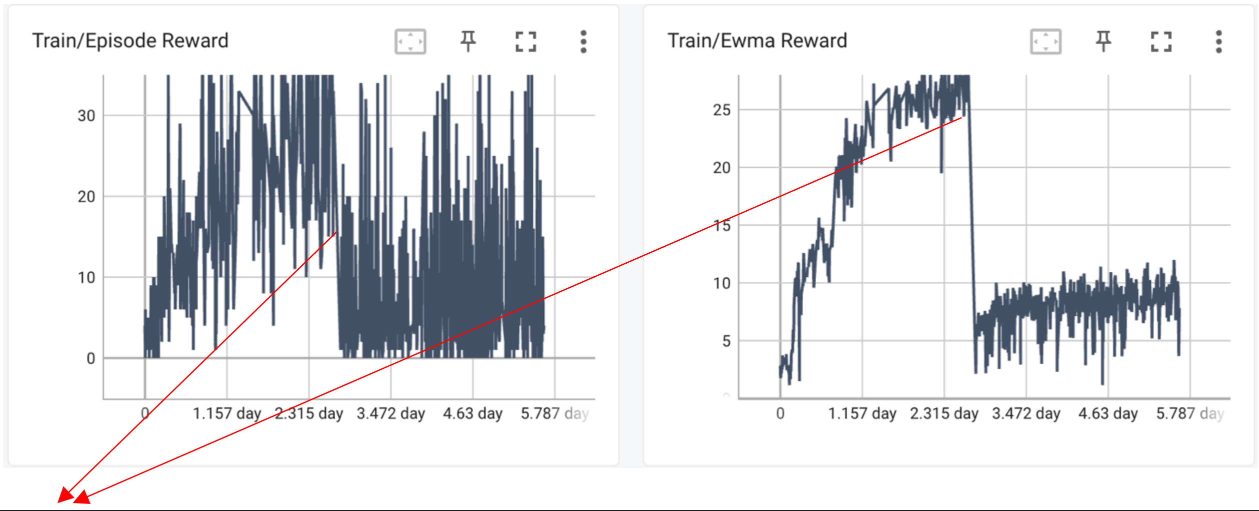
- Testing results

I set test epsilon to 0.0008 to make sure the agent play game in its best condition. But it is time consuming to wait the agent to "fire" the ball, so it takes one hour to finish ten episodes.



```
source activate ml
cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/bk_final_fast  source activate ml                                          12:04:
cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/bk_final_fast  /Users/cheng/miniforge3/envs/ml/bin/python /Users/cheng/Desktop/bk_final_fast/bk.py --test_only
Start Testing
episode 1: 335.00
episode 2: 335.00
episode 3: 335.00
episode 4: 335.00
episode 5: 335.00
episode 6: 335.00
episode 7: 335.00
episode 8: 335.00
episode 9: 335.00
episode 10: 335.00
Average Reward: 335.00
cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/bk_final_fast                                                01:50:24 Dur  13:54:
```

- Tensorboard



From here start using `episode_life=True, clip_rewards=True, frame_stack=True, scale=True`

This setting dramatically accelerate the testing score!

- Score : Breakout: min(5 * sqrt(average testing score), 100) * 0.4.

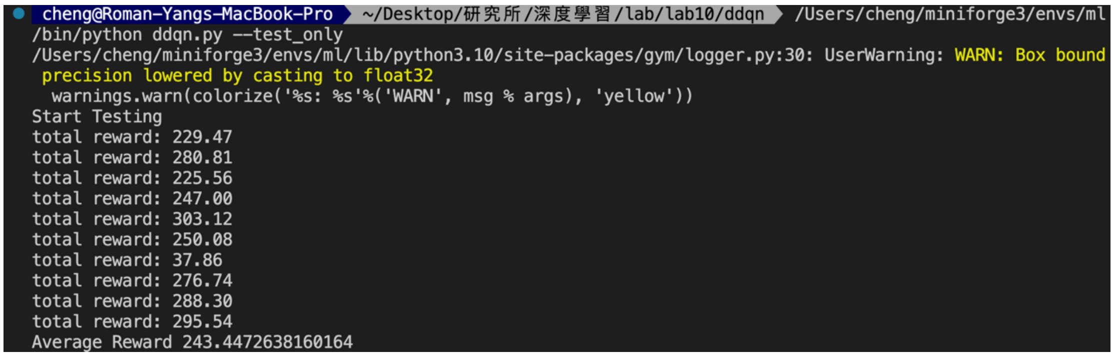
$5 * \sqrt{335} = 91.515$

$91.515 * 0.4 = 36.6$

Experimental Results of bonus parts

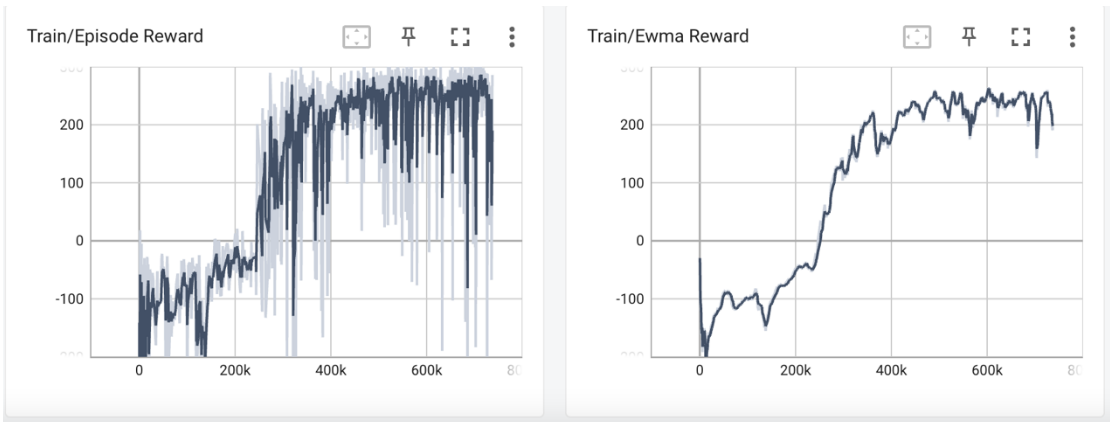
LunarLander-v2 - DDQN

- Testing results



```
● cheng@Roman-Yangs-MacBook-Pro  ~/Desktop/研究所/深度學習/lab/lab10/ddqn  /Users/cheng/miniforge3/envs/ml
/bin/python ddqn.py --test_only
/Users/cheng/miniforge3/envs/ml/lib/python3.10/site-packages/gym/logger.py:30: UserWarning: WARN: Box bound
 precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
total reward: 229.47
total reward: 280.81
total reward: 225.56
total reward: 247.00
total reward: 303.12
total reward: 250.08
total reward: 37.86
total reward: 276.74
total reward: 288.30
total reward: 295.54
Average Reward 243.4472638160164
```

- Tensorboard



Questions

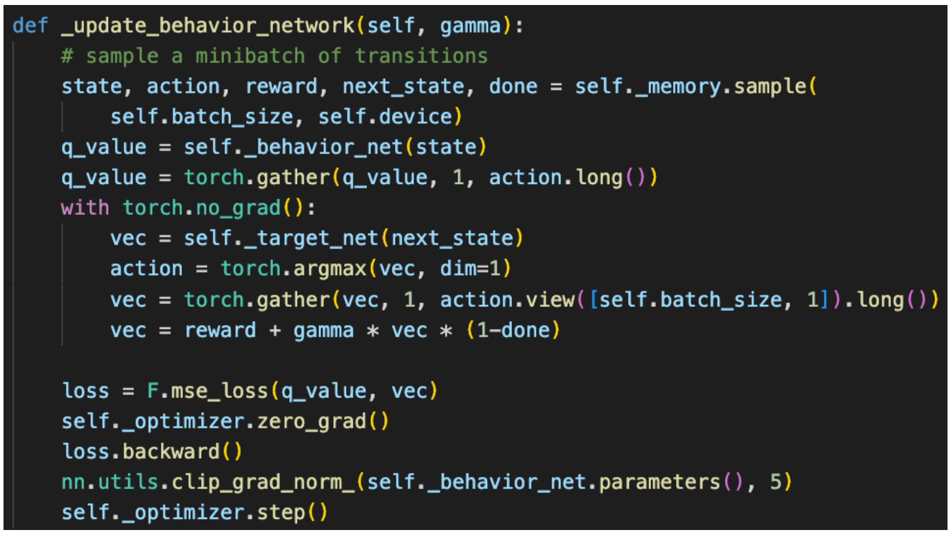
1. Describe your major implementation of both DQN and DDPG in detail.

- Your implementation of Q network updating in DQN.

  There are two network, behavior_net and target_net.

  Behavior net is used to compute every action’s Q value for a given state, and generally choose the max value as it’s action.

  Target network plus reward is used as the ground truth to train the behavior network. After several episodes, the target network will copy the behavior network. I use MSE loss to train my model.

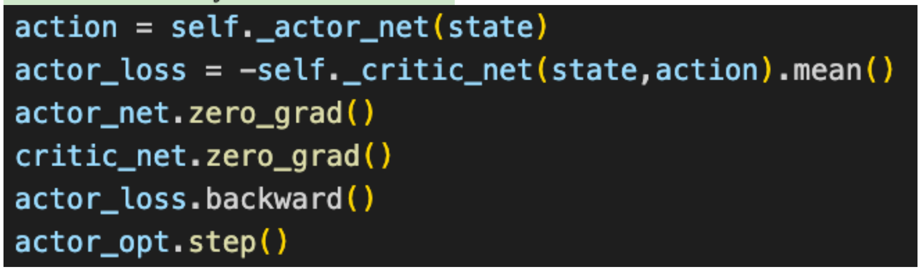


```python
def _update_behavior_network(self, gamma):
    # sample a minibatch of transitions
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size, self.device)
    q_value = self._behavior_net(state)
    q_value = torch.gather(q_value, 1, action.long())
    with torch.no_grad():
        vec = self._target_net(next_state)
        action = torch.argmax(vec, dim=1)
        vec = torch.gather(vec, 1, action.view([self.batch_size, 1]).long())
        vec = reward + gamma * vec * (1-done)

    loss = F.mse_loss(q_value, vec)
    self._optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
    self._optimizer.step()
```

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

  To maximize the probability of the action which leverage the most Q value, I use the equation shown bellowed to be my loss function of actor net.

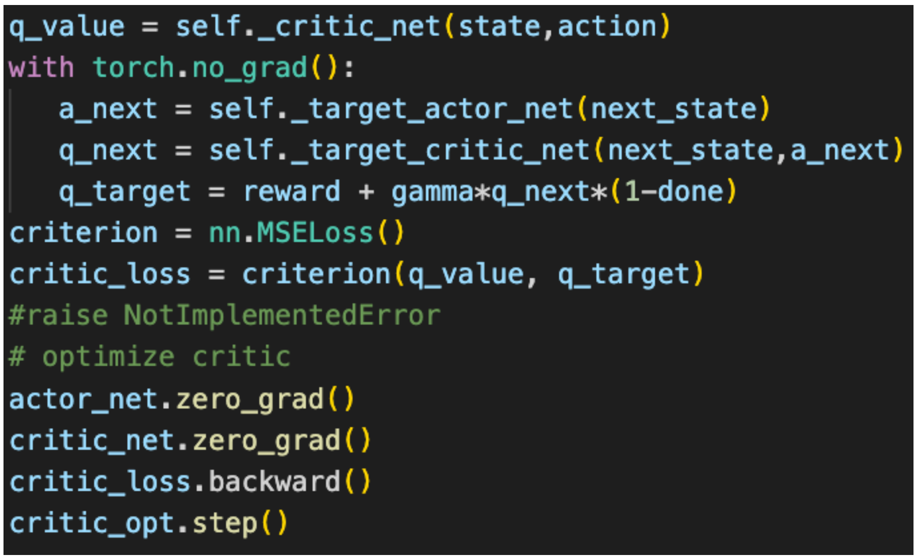
$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{m} Q(s_i, a_i, w)$$



```python
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()
```

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

  The behavior of critic network is similar to DQN. First, use target_actor_net to get the action.Second, use target_critic_net to get the Q value. Finally, use the update equation to compute target Q value. The loss also use MSE, which is same as DQN.



```python
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)
#raise NotImplementedError
# 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 is denoted as gamma shown bellowed:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots =$$

   It will affect whether the agent learn the strategy. If the discount factor equals to one, the agent will consider the long-term future value. If the discount factor equals to zero, the agent will consider the "immediate" reward. To sum up, the discount value affect the strategy of agent.

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

   The epsilon-greedy method is not always choose the best action as greedy. It reserves some probability to explore the environment. In contrast, greedy method can lead to suboptimal outcomes if the initial estimates are inaccurate or if there are multiple actions with similar value.

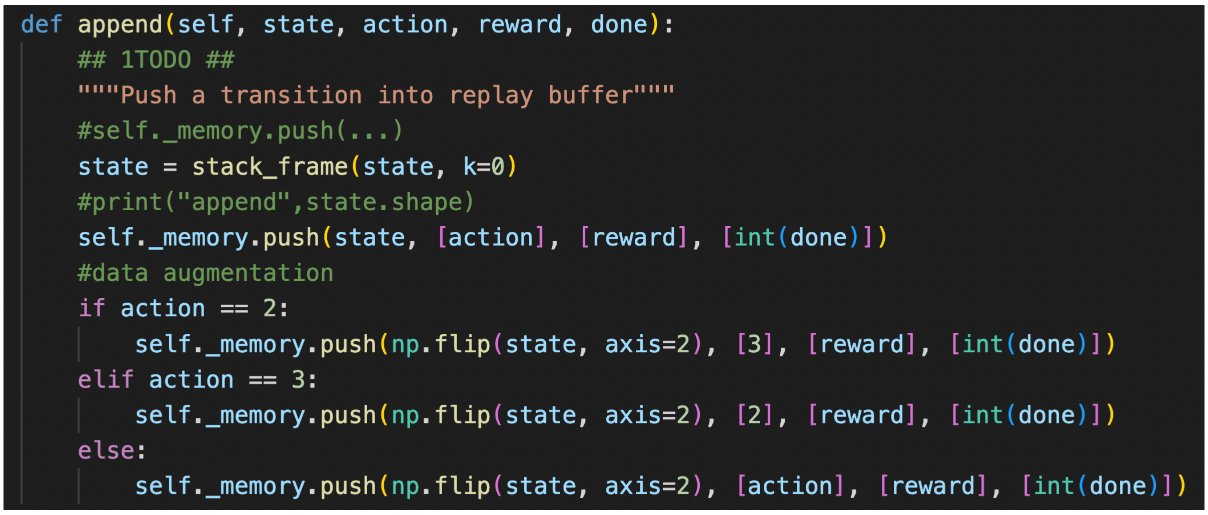
```
p = random()

if p < ε:
   pull random action
else:
   pull current-best action
```

4. Explain the necessity of the target network.

   Target network is used as the target(label) for behavior net to predict more accurate by using the information in the Replay memory. However, directly using behavior network as target network leads to the instability during training process. In general, we use target network as ground truth and replace it by behavior network after several episodes.

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

   - Data augmentation
     Although having epsilon greedy method, the network train really slow in the beginning. The stick keep stuck at the corner. So, when append data into replay memory, I extra append the mirror flip version of experience to the memory to make the model learn faster.



```python
def append(self, state, action, reward, done):
    ## 1TODO ##
    """Push a transition into replay buffer"""
    #self._memory.push(...)
    state = stack_frame(state, k=0)
    #print("append",state.shape)
    self._memory.push(state, [action], [reward], [int(done)])
    #data augmentation
    if action == 2:
        self._memory.push(np.flip(state, axis=2), [3], [reward], [int(done)])
    elif action == 3:
        self._memory.push(np.flip(state, axis=2), [2], [reward], [int(done)])
    else:
        self._memory.push(np.flip(state, axis=2), [action], [reward], [int(done)])
```

- Stack frame

In LunarLander, we don’t actually need to know how spaceship move continuously. But in the breakout task, it is crucial to know the state by stacking it together. After we stack it together, we can finally know how the ball moves and take the correct respond to the environment.