Deep Learning Lab6 (Deep Q-Network and Deep Deterministic Policy Gradient)

311553060

周睦鈞

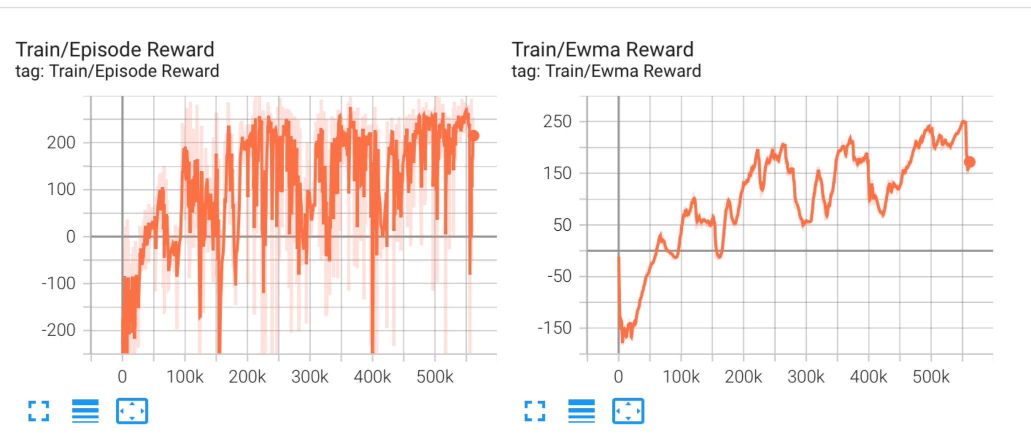
---

1. Experimental Results on LunarLander-v2

(a) Testing results:

```
Start Testing
Episode: 0      \Length: 397    Total Reward: 214.17
Episode: 1      \Length: 362    Total Reward: 256.40
Episode: 2      \Length: 470    Total Reward: 231.55
Episode: 3      \Length: 438    Total Reward: 239.62
Episode: 4      \Length: 297    Total Reward: 293.75
Episode: 5      \Length: 470    Total Reward: 221.48
Episode: 6      \Length: 429    Total Reward: 261.71
Episode: 7      \Length: 362    Total Reward: 255.53
Episode: 8      \Length: 455    Total Reward: 280.04
Episode: 9      \Length: 536    Total Reward: 247.34
Average Reward 250.15945927353354
```

(b) Tensorboard

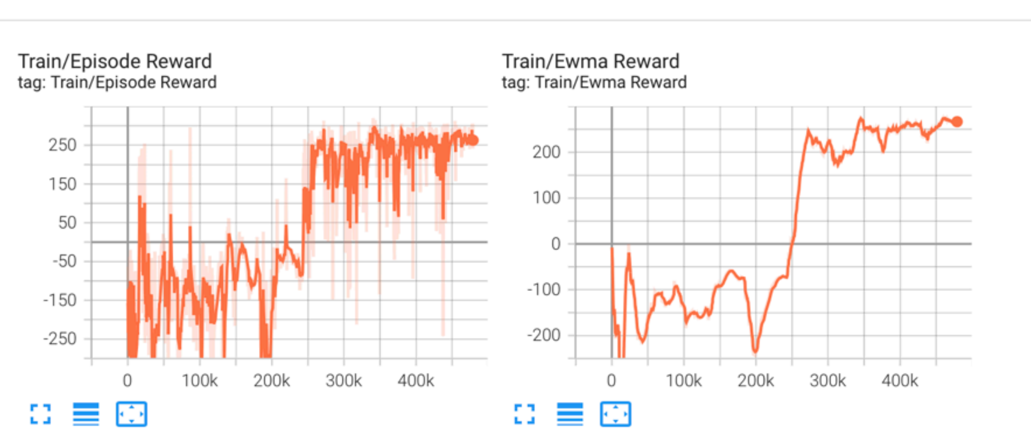


2. Experimental Results on LunarLanderContinuous-v2

(a) Testing results:

```
Start Testing
Episode: 0      Length: 189     Total reward: 250.68
Episode: 1      Length: 164     Total reward: 281.58
Episode: 2      Length: 203     Total reward: 275.07
Episode: 3      Length: 187     Total reward: 277.81
Episode: 4      Length: 190     Total reward: 280.24
Episode: 5      Length: 216     Total reward: 270.17
Episode: 6      Length: 235     Total reward: 275.11
Episode: 7      Length: 285     Total reward: 291.32
Episode: 8      Length: 220     Total reward: 296.96
Episode: 9      Length: 254     Total reward: 294.05
Average Reward 279.29920453594946
```

(b) Tensorboard

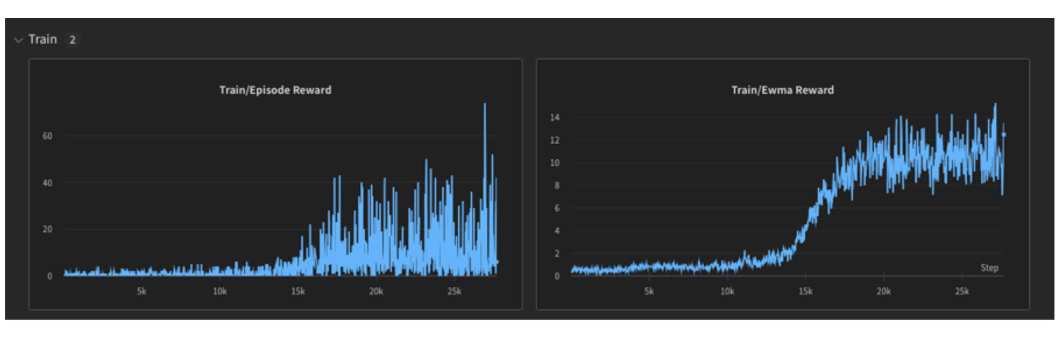


3. Experimental Results on BreakoutNoFrameskip-v4

(a) Testing results

```
(VLLab) mcchou@GS203:~/Lab$ CUDA_VISIBLE_DEVICES=3 python dqn_breakout.py --test_only -tmp '
dqn6.pth'
Start Testing
episode 1: 424.00
episode 2: 372.00
episode 3: 363.00
episode 4: 395.00
episode 5: 412.00
episode 6: 404.00
episode 7: 424.00
episode 8: 440.00
episode 9: 424.00
episode 10: 392.00
Average Reward: 405.00
```

(b) Tensorboard



4. Experimental Results of bonus parts(DDQN)

(a) Testing results

```
Start Testing
Episode: 0      \Length: 165    Total Reward: 249.37
Episode: 1      \Length: 218    Total Reward: 281.94
Episode: 2      \Length: 211    Total Reward: 277.64
Episode: 3      \Length: 210    Total Reward: 278.65
Episode: 4      \Length: 282    Total Reward: 299.91
Episode: 5      \Length: 504    Total Reward: 218.04
Episode: 6      \Length: 242    Total Reward: 301.44
Episode: 7      \Length: 230    Total Reward: 294.23
Episode: 8      \Length: 262    Total Reward: 310.54
Episode: 9      \Length: 409    Total Reward: 265.17
Average Reward 277.69245569083256
```

(b) Tensorboard

5. Question

(a) Describe your major implementation of both DQN and DDPG in detail

DQN

a-1-1. Select action

使用 epsilon-greedy 的方式去決定 action, 如果 random 的 number 小於 epsilon 就會隨機 sample 其中一個 action，否則我們會使用 behavior network 去選擇 maximum q-value

```
def select_action(self, state, epsilon, action_space):
    """
    epsilon-greedy based on behavior network
    :param state: current state
    :param epsilon: probability
    :param action_space: action space of current game
    :return: an action
    """
    ## TODO ##
    if random.random() > epsilon:
        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()
    else:
        return action_space.sample()
```

a-1-2. Update behavior network

從 Replay memory 中 sample minibatch 個 transition $(\phi_j, a_j, r_j, \phi_{j+1})$，並且計算 target value y_j 且用 MSE loss 來更新 behavior network

- $\hat{Q}$: target net
- $\hat{\theta}$: weights of target net
- γ: discount factor

$$y_j = \begin{cases} r_j, & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma * max_a \hat{Q}(\phi_{j+1}, a; \hat{\theta}), & otherwise \end{cases}$$

```
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(dim = 1, index = 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()
```

a-1-3. Update target network

```
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())
```

DDPG

a-2-1. Select action

會將 ActorNet 的 output 加上 noise 來當作下一個 action，但如果在 testing 則會直接使用 ActorNet 的 output

```
def select_action(self, state, noise=True):
    '''based on the behavior (actor) network and exploration noise'''
    ## TODO ##
    with torch.no_grad():
        state = torch.tensor(state, device=self.device).view(1, -1)
        outputs = self._actor_net(state)
        exploration_noise = torch.tensor(self._action_noise.sample(), device=self.device).view(1, -1)
        if noise:
            return (outputs + exploration_noise).squeeze(0).cpu().numpy()
        else:
            return outputs.squeeze(0).cpu().numpy()
```

a-2-2. Update behavior network

Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$

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

Update the actor policy using the sampled gradient:

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

根據公式，會從 Critic net 取得 q_value，之後會從 target actor net 得到 action 並且將其放入 target critic net 得到 q_target 來計算 MSE loss

```
## update critic ##
# critic loss
## TODO ##
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
## TODO ##
action = actor_net(state)
actor_loss = -critic_net(state, action).mean()
```

a-2-3. Update target network

利用 soft update 去 update target network

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

(b) Explain effects of the discount factor

當 Discount factor 越小，agent 會更專注在 current reward；反之，當 discount factor 越大，agent 會更專注在未來的 reward

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

(c) Explain benefits of epsilon-greedy in comparison to greedy action selection

model 如果都去選擇當前最好的 action，結果可能不一定對未來最好的，因此透過 epsilon-greedy 來增加隨機性說不定能讓 model 找到對未來最好的 action

(d) Explain the necessity of the target network

Target network 可以讓 behavior network 根據過去的經驗來比較新的 action 是否有往好的方向來走，因此沒有 target network 的話可能會造成訓練困難且不穩定

(e) Describe the tricks you used in Breakout and their effects, and how they differ from those used in LunarLander

Breakout 使用的是把四個 frame stack 再一起當作一個 state，和 LunarLander 一次只把 observation 當作 state 不一樣，因為 Network 沒辦法只看一張 frame 就決定 action，需要透過多個 frame 當作參考，因此才會將多個 frame stack 再一起當作一個 state

在 breakout 中，我們可以透過調整 wrap_deepmind 的參數來測試看看訓練結果，我發現如果在訓練中使用 scale 會訓練不太起來，因此我最後是把 scale 這個參數設成 false 去訓練