Deep Learning
Lab6: DQN and DDPG

Name: 許子駿
Student ID: 311551166

Institute of Computer Science and Engineering

2023 年 5 月 27 日

Table of Contents

Chapter 1

Introduction

This task is to solve *LunarLander-v2* and *BreakoutNoFrameskip-v4* using **deep Q-network (DQN)**, and *LunarLanderContinuous-v2* using **deep deterministic policy gradient (DDPG)**.

Chapter 2

Experiment setups

Experiment setups are listed in this chapter including 2 parts, DQN and DDPG.

2.1 DQN

DQN details are listed in this section including 4 parts, network, action selection, updating behavior network, and updating target network.

2.1.1 Network

Since *LunarLander-v2* includes 4 actions, i.e., 0 (No-op), 1 (Fire left engine), 2 (Fire main engine), 3 (Fire right engine), a network with output dimension 4 is built to predict Q value (See Listing 2.1).

```python
class Net(nn.Module):
    def __init__(self,
        state_dim=8, action_dim=4, hidden_dim=(400, 300)) -> None:
        super().__init__()

        self.layers = nn.Sequential(
            nn.Linear(in_features=state_dim,
                out_features=hidden_dim[0]),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=hidden_dim[0],
                out_features=hidden_dim[1]),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=hidden_dim[1],
                out_features=action_dim)
        )

```

```python
    def forward(self, x: torch.FloatTensor) -> torch.FloatTensor:
        return self.layers(x)
```

Listing 2.1: Python code of **Net** of DQN.

2.1.2 Action selection

During the play, action with maximum Q value with probability ϵ will be selected, or random action is selected, otherwise. It's called ϵ-greedy (See Listing 2.2).

```python
def select_action(self, state, epsilon, action_space):
    if random.random() > epsilon:
        with torch.no_grad():
            # Max element is in 2nd column (dim=1).
            state = torch.from_numpy(state).view(1, -1).to(self.device)
            return self._behavior_net(state).max(dim=1)[1].item()
    else:
        return action_space.sample()
```

Listing 2.2: Python code of **select_action** of DQN.

2.1.3 Updating behavior network

To update behavior network, sample transitions (state, action, reward, next action, done) from replay memory, then do TD-learning, and finally calculate MSE loss from Q value and Q target (See Listing 2.3 and Equation 2.1).

```python
def _update_behavior_network(self, gamma):
    state, action, reward, next_state, 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).max(dim=1)[0].view(-1, 1)
        q_target = reward + gamma * q_next * (1 - done)

    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)

    self._optim.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
    self._optim.step()
```

Listing 2.3: Python code of **_update_behavior_network** of DQN.

$$y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta\right) & \text{otherwise} \end{cases} \tag{2.1}$$

2.1.4 Updating target network

To update target network, copy weights of behavior network (See Listing 2.4).

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

Listing 2.4: Python code of **_update_target_network** of DQN.

2.2 DDPG

DDPG details are listed in this section including 5 parts, actor network, critic network, action selection, updating behavior network, and updating target network.

2.2.1 Actor network

Since *LunarLanderContinuous-v2* includes 2 engines, i.e., main engine with range $[-1, 1]$ and left-right engine with range $[-1, 1]$, a network with output dimension 2 is built (See Listing 2.5).

```python
class ActorNet(nn.Module):
    def __init__(self,
            state_dim=8, action_dim=2, hidden_dim=(400, 300)) -> None:
        super().__init__()

        self.layers = nn.Sequential(
            nn.Linear(in_features=state_dim,
                out_features=hidden_dim[0]),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=hidden_dim[0],
                out_features=hidden_dim[1]),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=hidden_dim[1],
                out_features=action_dim),
            nn.Tanh()
        )

    def forward(self, x: torch.FloatTensor) -> torch.FloatTensor:
```

```
        return self.layers(x)
```

Listing 2.5: Python code of **ActorNet** of DDPG.

2.2.2 Critic network

Critic network is used to predict Q value. Since the output is scalar, output dimension 1. (See Listing 2.6).

```
class CriticNet(nn.Module):
    def __init__(self,
            state_dim=8, action_dim=2, hidden_dim=(400, 300)) -> None:
        super().__init__()

        h1, h2 = hidden_dim
        self.critic_head = nn.Sequential(
            nn.Linear(state_dim + action_dim, h1),
            nn.ReLU(),
        )
        self.critic = nn.Sequential(
            nn.Linear(h1, h2),
            nn.ReLU(),
            nn.Linear(h2, 1),
        )

    def forward(self,
            x: torch.FloatTensor, action: torch.FloatTensor) -> torch.FloatTensor:
        x = self.critic_head(torch.cat([x, action], dim=1))
        return self.critic(x)
```

Listing 2.6: Python code of **CriticNet** of DDPG.

2.2.3 Action selection

During the play, select action with actor network and add extra noise (See Listing 2.7).

```
def select_action(self, state, noise=True):
    with torch.no_grad():
        if noise:
            re = self._actor_net(torch.from_numpy(state).view(1, -1).to(self.device)) + \
                    torch.from_numpy(self._action_noise.sample()).view(1, -1).to(self.device)
```

```
        else:
            re = self._actor_net(torch.from_numpy(state).view(1, -1).to(self.
    device))

    return re.to('cpu').numpy().squeeze()
```

Listing 2.7: Python code of **select_action** of DDPG.

2.2.4 Updating behavior network

To update behavior network, sample transitions (state, action, reward, next action, done) from replay memory, and calculate MSE loss from Q value of behavior network and Q target of target network to update critic network. Also, to maximize Q value by updating actor network μ, calculate negative Q value expectation $\mathbb{E}[-Q(s, \mu(s))]$ (See Listing 2.8 and Equation 2.2).

```
def _update_behavior_network(self, gamma):
    # Sample transitions batch.
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size, self.device)

    # Update critic.
    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)

    self._actor_net.zero_grad()
    self._critic_net.zero_grad()
    critic_loss.backward()
    self._critic_opt.step()

    # Update actor.
    action = self._actor_net(state)
    actor_loss = -self._critic_net(state, action).mean()

    self._actor_net.zero_grad()
    self._critic_net.zero_grad()
    actor_loss.backward()
```

```
    self._actor_opt.step()
```

Listing 2.8: Python code of **__update__behavior__network** of DDPG.

$$y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}) \tag{2.2}$$

2.2.5 Updating target network

To update target network, **soft copy** weights of behavior network (See Listing 2.9).

```
def _update_target_network(target_net, net, tau):
    for target, behavior in zip(target_net.parameters(), net.parameters()):
        target.data.copy_((1 - tau) * target.data + tau * behavior.data)
```

Listing 2.9: Python code of **__update__target__network** of DDPG.

Chapter 3

Experimental results

Results are listed in this chapter including 3 parts, DQN for *LunarLander-v2*, DDPG for *LunarLanderContinuous-v2*, and DQN for *BreakoutNoFrameskip-v4*.

3.1 DQN for *LunarLander-v2*

From Figure 3.2, we can see that the average testing reward is **255.89**.

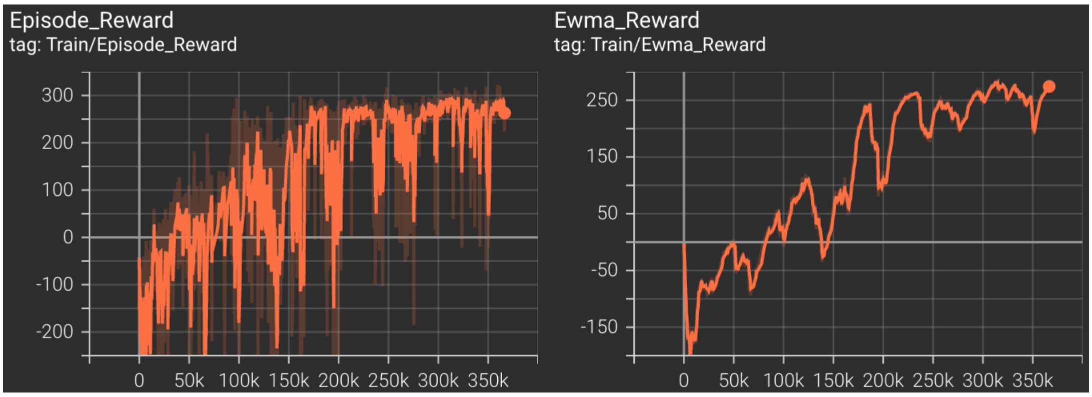
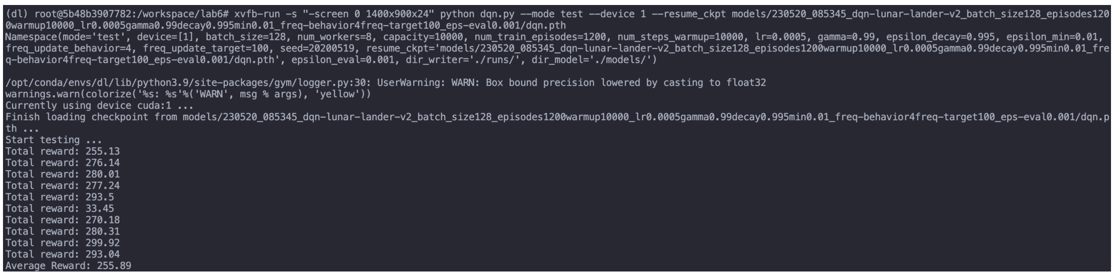


Figure 3.1: Rewards of DQN for *LunarLander-v2* during training.



```
(dl) root@5b48b3907782:/workspace/lab6# xvfb-run -s "-screen 0 1400x900x24" python dqn.py --mode test --device 1 --resume_ckpt models/230520_085345_dqn-lunar-lander-v2_batch_size128_episodes120
0warmup10000_lr0.0005gamma0.99decay0.995min0.01_freq-behavior4freq-target100_eps-eval0.001/dqn.pth
Namespace(mode='test', device=[1], batch_size=128, num_workers=8, capacity=10000, num_train_episodes=1200, num_steps_warmup=10000, lr=0.0005, gamma=0.99, epsilon_decay=0.995, epsilon_min=0.01,
freq_update_behavior=4, freq_update_target=100, seed=20200519, resume_ckpt='models/230520_085345_dqn-lunar-lander-v2_batch_size128_episodes1200warmup10000_lr0.0005gamma0.99decay0.995min0.01_fre
q-behavior4freq-target100_eps-eval0.001/dqn.pth', epsilon_eval=0.001, dir_writer='./runs/', dir_model='./models/')

/opt/conda/envs/dl/lib/python3.9/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'))
Currently using device cuda:1 ...
Finish loading checkpoint from models/230520_085345_dqn-lunar-lander-v2_batch_size128_episodes1200warmup10000_lr0.0005gamma0.99decay0.995min0.01_freq-behavior4freq-target100_eps-eval0.001/dqn.p
th ...
Start testing ...
Total reward: 255.13
Total reward: 276.14
Total reward: 280.01
Total reward: 277.24
Total reward: 293.5
Total reward: 33.45
Total reward: 270.18
Total reward: 280.31
Total reward: 299.92
Total reward: 293.04
Average Reward: 255.89
```

Figure 3.2: Rewards of DQN for *LunarLander-v2* during evaluation.

3.2 DDPG for *LunarLanderContinuous-v2*

From Figure 3.4, we can see that the average testing reward is **206.39**.

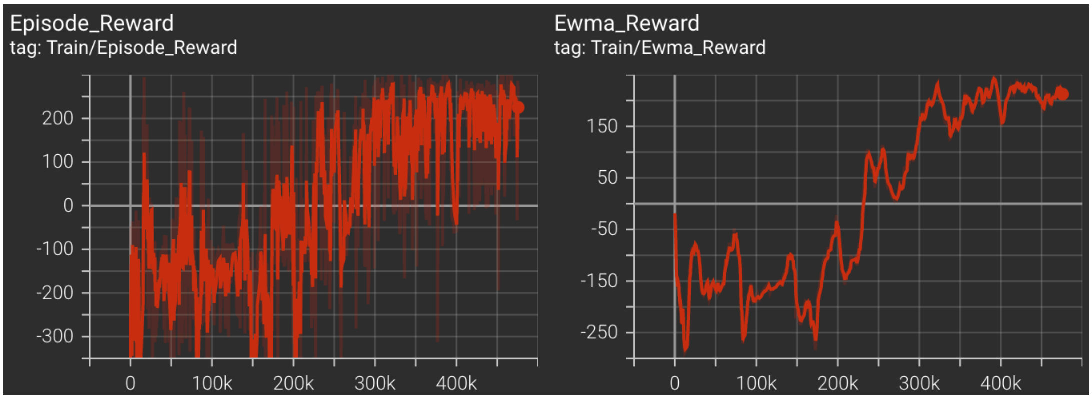


Figure 3.3: Rewards of DDPG for *LunarLanderContinuous-v2* during training.

```
(dl) root@5b48b3907782:/workspace/lab6# xvfb-run -s "-screen 0 1400x900x24" python ddpg.py --mode test --device 1 --resume_ckpt models/230521_063715_ddpg-lunar-lander-continuous-v2_batch_size64
_episodes1200warmup10000_lra0.001lrc0.001gamma0.99tau0.005/ddpg.pth
Namespace(mode='test', device=[1], batch_size=64, num_workers=8, capacity=500000, num_train_episodes=1200, num_steps_warmup=10000, lr_actor=0.001, lr_critic=0.001, gamma=0.99, tau=0.005, seed=2
0200519, resume_ckpt='models/230521_063715_ddpg-lunar-lander-continuous-v2_batch_size64_episodes1200warmup10000_lra0.001lrc0.001gamma0.99tau0.005/ddpg.pth', dir_writer='./runs/', dir_model='./m
odels/')

/opt/conda/envs/dl/lib/python3.9/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'))
Currently using device cuda:1 ...
Finish loading checkpoint from models/230521_063715_ddpg-lunar-lander-continuous-v2_batch_size64_episodes1200warmup10000_lra0.001lrc0.001gamma0.99tau0.005/ddpg.pth ...
Start testing ...
Total reward: 251.21
Total reward: 153.31
Total reward: 282.03
Total reward: 145.49
Total reward: 229.51
Total reward: 264.50
Total reward: 219.92
Total reward: 297.10
Total reward: -44.63
Total reward: 265.44
Average Reward: 206.39
```

Figure 3.4: Rewards of DDPG for *LunarLanderContinuous-v2* during evaluation.

3.3 DQN for *BreakoutNoFrameskip-v4*

From Figure 3.6, we can see that the average testing reward is **227.06**.

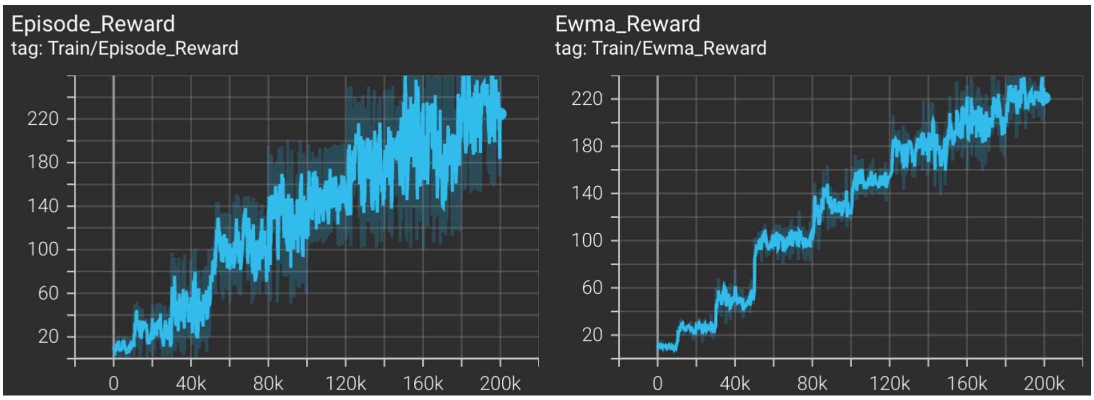


Figure 3.5: Rewards of DQN for *BreakoutNoFrameskip-v4* during training.

```
(dl) root@5b48b3907782:/workspace/lab6# xvfb-run -s "-screen 0 1400x900x24" python dqn_breakout.py --mode test --device 1 --resume_ckpt models/230526_120821_dqn-breakout_batch_size32_episodes20
000warmup20000_lr6.25e-05gamma0.99decay1000000min0.1_freq-behavior4freq-target10000freq-eval200000_eps-eval0.01episodes-eval10/dqn-breakout.pth
Namespace(mode='test', device=[1], batch_size=32, num_workers=8, capacity=100000, num_train_episodes=20000, num_steps_warmup=20000, lr=6.25e-05, gamma=0.99, epsilon_decay=1000000, epsilon_min=0
.1, freq_update_behavior=4, freq_update_target=10000, freq_eval=200000, resume_ckpt='models/230526_120821_dqn-breakout_batch_size32_episodes20000warmup20000_lr6.25e-05gamma0.99decay1000000min0.
1_freq-behavior4freq-target10000freq-eval200000_eps-eval0.01episodes-eval10/dqn-breakout.pth', epsilon_eval=0.01, num_eval_episodes=10, dir_writer='./runs/', dir_model='./models/')

Currently using device cuda:1 ...
Finish loading checkpoint from models/230526_120821_dqn-breakout_batch_size32_episodes20000warmup20000_lr6.25e-05gamma0.99decay1000000min0.1_freq-behavior4freq-target10000freq-eval200000_eps-ev
al0.01episodes-eval10/dqn-breakout.pth ...
Start testing ...
Episode 1: 234.48
Episode 2: 239.47
Episode 3: 232.12
Episode 4: 223.81
Episode 5: 246.23
Episode 6: 229.97
Episode 7: 212.54
Episode 8: 207.03
Episode 9: 223.01
Episode 10: 221.91
Average Reward: 227.06
```

Figure 3.6: Rewards of DQN for *BreakoutNoFrameskip-v4* during evaluation.

Chapter 4

Discussion

Four questions and their answers are listed in this chapter.

4.1 Explain effects of the discount factor

In Equation 4.1, λ is discount factor, and the future effects getting smaller with the degree of λ getting larger.

$$G_t = R_{t+1} + \lambda R_{t+2} + \cdots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1} \tag{4.1}$$

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

It's better to balance between explore and exploit with greedy action selection, and sometimes choosing other action to explore may be best action.

4.3 Explain the necessity of the target network

With target network and behavior network, the training process can be more stable, since Q target is output from target network and it's updated with lower frequency.

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

For *LunarLander-v2*, **MSE loss** is used, but for *BreakoutNoFrameskip-v4*, **smooth L1 loss** is used instead (See Listing 4.1).

```
def _update_behavior_network(self, gamma):
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size)

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

    loss = F.smooth_l1_loss(q_value, q_target.unsqueeze(1))

    self._optim.zero_grad()
    loss.backward()
    for param in self._behavior_net.parameters():
        param.grad.data.clamp_(-1, 1)
    self._optim.step()
```

Listing 4.1: Python code of **__update__behavior__network** of DQN for *BreakoutNoFrameskip-v4*.