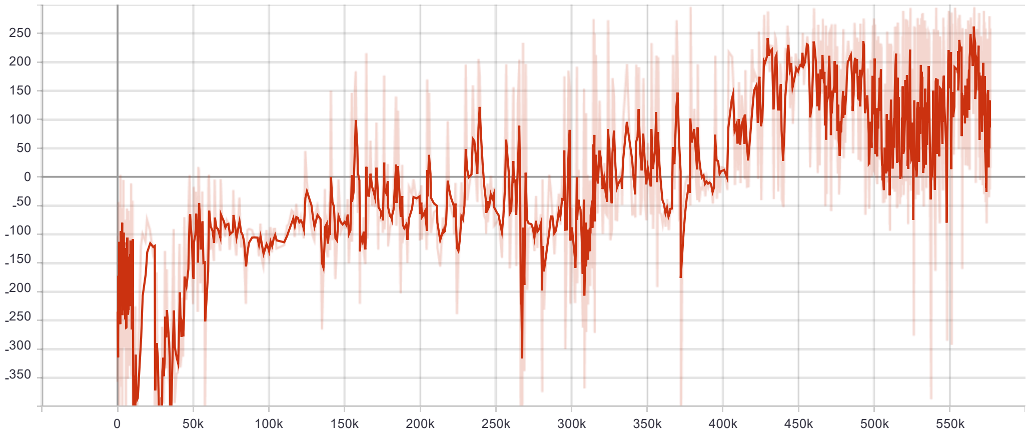
DLP Lab6 Deep Q-Network and Deep Deterministic Policy Gradient

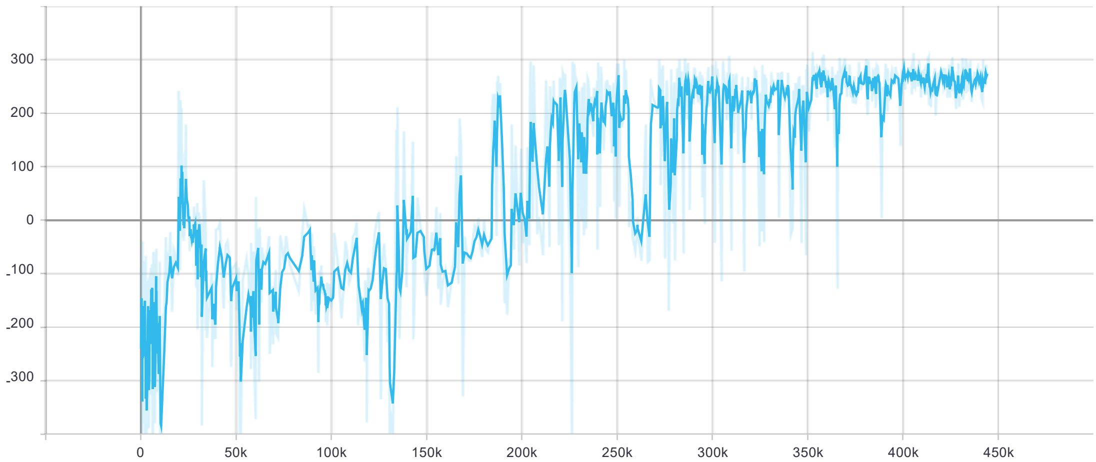
0711529 陳冠儒

Report

- A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLander-v2(5%).
  - Using Soft updating target network



- A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLanderContinuous-v2(5%)



- Describe your major implementation of both algorithms in detail. (20%)
  - ◆ DQN
    - Selection Action with ε -Greedy Exploration

```python
state = torch.from_numpy(state).float().unsqueeze(0).to(device)
self._behavior_net.eval()
with torch.no_grad():
    action_values = self._behavior_net(state)
self._behavior_net.train()

# Epsilon-greedy action selection
if random.random() > epsilon:
    return np.argmax(action_values.cpu().data.numpy())
else:
    return action_space.sample()
```

使用 selection action 來選擇在 training 時要做的動作，會先使用 state 丟到 behavior net 的 eval mode 來得到該 state 各 action 的 value，再依照 ε -Greedy 看是要選擇最大 value 的 action 進行 exploitation，或是進行隨機 action 的 exploration。

- Experience replay & Update the behavior network

```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).gather(1, action.to(dtype=int))
    with torch.no_grad():
        q_next = self._target_net(next_state).detach().max(1)[0].unsqueeze(1)
        q_target = reward + (gamma * q_next * (1 - done))
    loss = F.mse_loss(q_value, q_target)
    # optimize
    self._optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
    self._optimizer.step()
```

使用 replay experience 來獲得存在 memory buffer 中以前算過的資料，再帶入 behavior net 和 target net 來求得 Q learning 的 loss function 並 update behavior net。

- Update Target Network

```python
def _update_target_network(self):
    '''update target network by copying from behavior network'''
    self._target_net.load_state_dict(self._behavior_net.state_dict())
```

原始的 algorithm 在訓練了 1000 步後更新 target network，將 behavior network 的參數複製到 target network 中，在下面會將其改成 soft updating 的形式。

- Network Architecture
  - ✓ Input: 8-dimension observation
  - ✓ First layer: fully connected layer (ReLU)
    - ■ input: 8, output: 32
  - ✓ Second layer: fully connected layer (ReLU)
    - ■ input: 32, output: 32
  - ✓ Third layer: fully connected layer
    - ■ input: 32, output: 4
- Hyperparameters
  - ✓ Memory capacity (experience buffer size): 10000
  - ✓ Batch size: 128

- ✓ Warmup steps: 10000
- ✓ Optimizer: Adam
- ✓ Learning rate: 0.0005
- ✓ Epsilon: 1200
- ✓ Gamma (discount factor): 0.99
- ✓ Update network evert 4 iterations
- ✓ Update target network every 100 iterations (updating by copying)

◆ DDPG

- Selection Action with exploration noise

```
def select_action(self, state, env, noise=True):
    '''based on the behavior (actor) network and exploration noise'''
    self._actor_net.eval()
    state = torch.from_numpy(state).float().unsqueeze(0).to(self.device)
    with torch.no_grad():
        action = self._actor_net(state).to("cpu").data.numpy()
    self._actor_net.train()
    if noise:
        action += self._action_noise.sample()
        action = action.clip(env.action_space.low[0], env.action_space.high[0])
    return action[0]
```

跟 DQN ε -Greedy 不一樣的地方在於 DDPG 因為是用於 continuous action spaces，所以他在 exploration 時是用添加 noise 的方式。

- Target Network Updating

```
def _update_target_network(target_net, net, tau):
    '''update target network by _soft_ copying from behavior network'''
    with torch.no_grad():
        for target, behavior in zip(net.parameters(), target_net.parameters()):
            behavior.data.mul_((1.0 - tau))
            behavior.data.add_(tau * target.data)
```

使用 soft updating 的方式，讓他可以緩慢差距不大的更新。
另外 DDPG 的 experience replay 方式跟 DQN 相同故就不再說一次，而 actor 跟 critic updating 的方式將在下面詳述。

- Network Architecture
    - ◆ Actor
        - ✓ Input: 8-dimension observation
        - ✓ First layer: fully connected layer (ReLU)
            - ■ input: 8, output: 400
        - ✓ Second layer: fully connected layer (ReLU)
            - ■ Input: 400, output: 300
        - ✓ Third layer: fully connected layer (tanh)
            - ■ input: 300, output: 2

- Critic
  - Input: 8-dimension observation & action with dim 2
  - First layer: fully connected layer (ReLU)
    - input: 10, output: 400
  - Second layer: fully connected layer (ReLU)
    - Input: 400, output: 300
  - Third layer: fully connected layer (ReLU)
    - input: 300, output: 300
  - Forth layer: fully connected layer
    - input: 300, output: 1
- Hyperparameters
  - Memory capacity (experience buffer size): 50000
  - Batch size: 64
  - Warmup steps: 10000
  - Optimizer: Adam
  - Learning rate (actor): 0.0001
  - Learning rate (critic): 0.0001
  - Gamma (discount factor): 0.99
  - Tau: 0.005

- Describe differences between your implementation and algorithms. (10%)
  - DQN
    - Using Soft Updating for Target Network

```python
def _update_target_network(self):
    '''soft update target network'''
    tau = 1e-3
    for target_param, local_param in zip(self._target_net.parameters(), self._behavior_net.parameters()):
        target_param.data.copy_(tau*local_param.data + (1.0-tau)*target_param.data)
```

從原本的直接 copy 換成了使用 soft updating，原理跟上面所述的 DDPG soft updating target network 一樣。在這邊 DQN 是每隔 4 步更新一次 target network，而更新的方式變成原先的 target network * 0.997 + behavior network * 0.003，緩慢的更新，就不會有像是原本 copy updating 一樣如此劇烈的更動。

- Hyperparameters
  - tau = 0.001
  - Update target network every 4 iterations

- Describe your implementation and the gradient of actor updating. (10%)

```python
# actor loss
actor_loss = -critic_net(state, actor_net(state)).mean()
# optimize actor
actor_net.zero_grad()
critic_net.zero_grad()
actor_loss.backward()
actor_opt.step()
```

actor network 用於 policy 參數的更新，使用 policy objective function J(θ)來估測一個 policy 的好壞，並希望去 maximize J(θ)，故將其微分求倒數 ∇J(θ) 也就是 policy gradient。

在這裡使用 Off-Policy Deterministic Actor-Critic，將 actor_net(state)的結果用於 critic net 的 action 來計算，並且因為是希望要 maximize policy objective function 故在前面加負號。

- Describe your implementation and the gradient of critic updating. (10%)

```python
# critic loss
with torch.no_grad():
    q_next = target_critic_net(next_state, target_actor_net(next_state))
    q_target = reward + (1 - done) * gamma * q_next
a_next = critic_net(state, action)
criterion = nn.MSELoss()
critic_loss = criterion(a_next, q_target)
# optimize critic
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

critic 用於 q value function 的近似，他的 update 方式與 DQN 中的類似。先求出下一個 state 的 q value，在將他乘上 gamma 後與 reward 相加得到 target，就可以用原來的 state q value 與 target 進行 MSE 算出 loss 用梯度下降來更新 critic network。

- Explain effects of the discount factor. (5%)

discount factor 是一個介於[0, 1]的 hyperparameters，當 discount factor 小的時候，會較重視及時的 reward；當 discount factor 大的時候會較重視遠程的 reward。以這次的 LunarLander-v2 為例，因為他是一個會結束的遊戲，所以即使 discount factor 很大也不太會有造成總和無限大的問題，且他的最終目標是希望能平穩的降落，因此遠程會是比較好的選擇，因此 discount factor 就會選擇很大的 0.99。

- Explain benefits of epsilon-greedy in comparison to greedy action selection. (5%)

Greedy action selection 就是每次都選擇擁有最大 action value 的選擇，但是這樣會產生 Exploration-Exploitation Dilemma，greedy action selection 沒有 Exploration 的步驟，因此也就是說現在選擇的可能並不是最佳的值；而 epsilon-greedy 會利用 epsilon 當作一個機率 threshold，有一部分的機率會去進行 exploration，另一部分的機率是進行 exploitation，且可以保證經過

epsilon-greedy 後 policy value 一定會增加。

- Explain the necessity of the target network. (5%)
  如果沒有 target network 而只有一個 network 的話，那個 network 要同時 update 和產生 target，因此等號兩邊都會是變數，造成訓練會很不穩定，所以需要讓式子的一邊並成定數，故使用 target network 來當作 update 時的目標。

- Explain the effect of replay buffer size in case of too large or too small. (5%)
  在大多數實驗中，replay buffer 設置為默認容量 10^6。實際上 replay buffer 的大小會嚴重影響到學習速度和 agent 的 quality。如果 replay buffer 太小則違背了他的用意，幾乎沒有用處，因為 replay buffer 是希望能夠儲存下之前跟環境互動後的結果並反覆再利用，但如果太小的話，基本上沒有多少可以儲存。而如果 replay buffer 太大，會需要很大量的 memory 來儲存，並且也會減慢訓練的速度。

Performance

- [LunarLander-v2] Average reward of 10 testing episodes: Average ÷ 30
  - Using Soft updating target network
    Average reward：183.137

```
(base) ubuntu@ec037-108:~/DLP/Lab6$ python3 ./dqn.py --test_only -m="dqn(soft).pth"
Start Testing
Episode: 0      Reward: 232.75
Episode: 1      Reward: 262.64
Episode: 2      Reward: 286.62
Episode: 3      Reward: 287.25
Episode: 4      Reward: 227.23
Episode: 5      Reward: -29.39
Episode: 6      Reward: 163.45
Episode: 7      Reward: 232.92
Episode: 8      Reward: -72.77
Episode: 9      Reward: 240.82
Average Reward 183.15204817845705
```

- [LunarLanderContinuous-v2] Average reward of 10 testing episodes: Average ÷ 30
  Average reward：231.80

```
(base) ubuntu@ec037-108:~/DLP/Lab6$ python3 ./ddpg.py --test_only
Start Testing
Episode: 0      Reward: 257.14
Episode: 1      Reward: 288.37
Episode: 2      Reward: 287.05
Episode: 3      Reward: 283.68
Episode: 4      Reward: 280.10
Episode: 5      Reward: 142.98
Episode: 6      Reward: 219.52
Episode: 7      Reward: 293.66
Episode: 8      Reward: -9.44
Episode: 9      Reward: 274.98
Average Reward 231.8045067606384
```