Lab 6: Deep Q-Network and Deep Deterministic Policy Gradient

Lab Objective:

In this lab, you will learn and implement two deep reinforcement algorithms by completing the following two tasks: (1) solve LunarLander-v2 using deep Q-network (DQN), and (2) solve LunarLanderContinuous-v2 using deep deterministic policy gradient (DDPG).

Important Date:

1. Experiment report submission deadline: 12/02 (Wed) 12:00
(No demo)

Turn in:

1. Experiment report (.pdf)
2. Source code [NOT including model weights]
Notice: zip all files with name "DLP_LAB6_StudentId_Name.zip",
    e.g.: 「DLP_LAB6_0856032_鄭余玄.zip」
DLP_LAB6_0856032_鄭余玄
├── dqn.py
├── ddpg.py
└── report.pdf

3. Email title: MTK_DLP_LAB6_StudentId_Name
4. Email to: chengscott.cs08g@nctu.edu.tw
(Wrong format deduction: -5pts; Multiple deductions may apply.)

Lab Description:

- Understand the mechanism of both behavior network and target network.
- Understand the mechanism of experience replay buffer.
- Learn to construct and design neural networks.
- Understand "soft" target updates.
- Understand the difference between DQN and DDPG.

Requirements:

- Implement DQN
    - Construct the neural network
    - Select action according to epsilon-greedy
    - Construct Q-values and target Q-values
    - Calculate loss function
    - Update behavior and target network

- Understand deep Q-learning mechanisms
- Implement DDPG
  - Construct neural networks of both actor and critic
  - Select action according to the actor and the exploration noise
  - Update critic by minimizing the loss
  - Update actor using the sampled policy gradient
  - Update target network softly
  - Understand the mechanism of actor-critic

Game Environment – LunarLander-v2:

- Introduction: Rocket trajectory optimization is a classic topic in optimal control. Coordinates are the first two numbers in state vector, where landing pad is always at coordinates (0,0). Reward for moving from the top of the screen to landing pad and zero speed is about 100 to 140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.
- Observation [8]:
  - Horizontal Coordinate
  - Vertical Coordinate
  - Horizontal Speed
  - Vertical Speed
  - Angle
  - Angle Speed
  - If first leg has contact
  - If second leg has contact
- Action [4]: 0 (No-op), 1 (Fire left engine), 2 (Fire main engine), 3(Fire right engine)

Game Environment – LunarLanderContinuous-v2:

- Introduction: same as LunarLander-v2.
- Observation [8]: same as LunarLander-v2.
- Action [2]:
  - Main engine: -1 to 0 off, 0 to +1 throttle from 50% to 100% power. Engine can't work with less than 50% power.
  - Left-right: -1.0 to -0.5 fire left engine, +0.5 to +1.0 fire right engine, -0.5 to 0.5 off

Implementation Details – LunarLander-v2:

Network Architecture

- Input: an 8-dimension observation (not an image)
- 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

Training Hyper-Parameters

- Memory capacity (experience buffer size): 10000
- Batch size: 128
- Warmup steps: 10000
- Optimizer: Adam
- Learning rate: 0. 0005
- Epsilon: $1 \rightarrow 0.1$ or $1 \rightarrow 0.01$
- Gamma (discount factor): 0.99
- Update network evert 4 iterations
- Update target network every 100 iterations

Algorithm – Deep Q-learning with experience replay:

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t = 1,$T **do**
        With probability ε select a random action a_t
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t),a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t,a_t,x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from D

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

        Perform a gradient descent step on $\left(y_j - Q(\phi_j,a_j; \theta)\right)^2$ with respect to the
        network parameters θ
        Every C steps reset $\hat{Q} = Q$
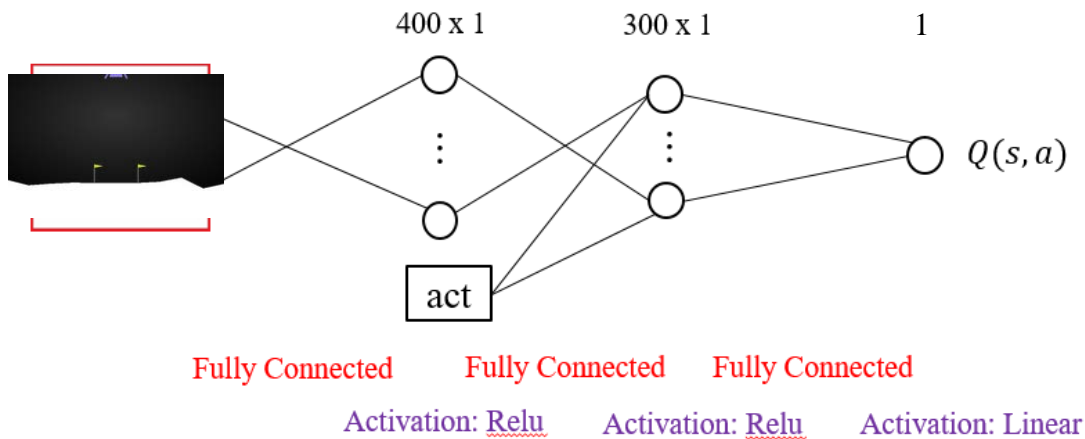   **End For**
End For

Implementation Details – LunarLanderContinuous-v2:

Network Architecture

- Actor
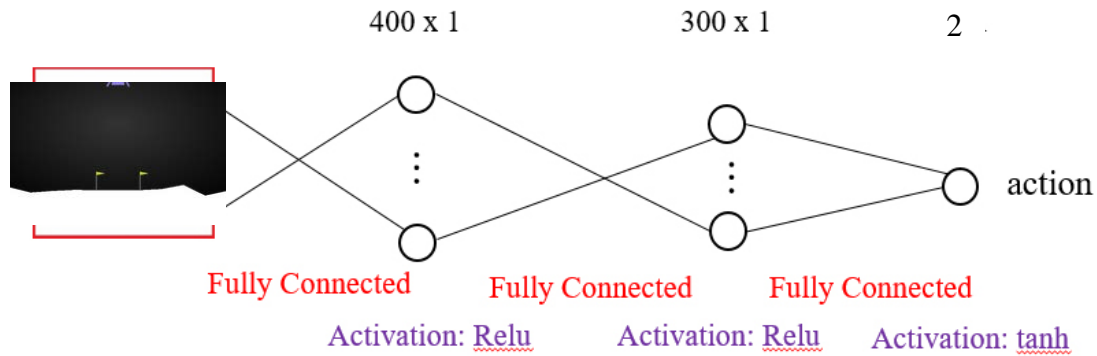
- Critic

Training Hyper-Parameters

- Memory capacity (experience buffer size): 500000
- Batch size: 64
- Warmup step: 10000
- Optimizer: Adam
- Learning rate (actor): 0.001
- Learning rate (critic): 0.001
- Gamma (discount factor): 0.99
- Tau: 0.005

Algorithm – DDPG algorithm:

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

for $episode = 1, M$ **do**

   Initialize a random process N for action exploration

   Receive initial observation state s_1

   for $t = 1, T$ **do**

     Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise

     Execute action a_t and observe reward r_t and observe new state s_{t+1}

     Store transition (s_t, a_t, r_t, s_{t+1}) in R

     Sample random minibatch of N transitions (s_j, a_j, r_j, s_{j+1}) from R

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

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

     Update the actor policy using the sampled gradient:

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

     Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$$

   end for

end for

Scoring Criteria:

Show your work, otherwise no credit will be granted.

- Report (80%)
  - A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLander-v2 (5%)
  - 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%)
  - Describe differences between your implementation and algorithms. (10%)
  - Describe your implementation and the gradient of actor updating. (10%)
  - Describe your implementation and the gradient of critic updating. (10%)
  - Explain effects of the discount factor. (5%)
  - Explain benefits of epsilon-greedy in comparison to greedy action selection. (5%)
  - Explain the necessity of the target network. (5%)
  - Explain the effect of replay buffer size in case of too large or too small. (5%)
- Report Bonus (20%)
  - Implement and experiment on Double-DQN (10%)
  - Extra hyperparameter tuning, e.g., Population Based Training. (10%)
- Performance (20%)
  - [LunarLander-v2] Average reward of 10 testing episodes: Average ÷ 30
  - [LunarLanderContinuous-v2] Average reward of 10 testing episodes: Average ÷ 30

References:

[1] Mnih, Volodymyr et al. "Playing Atari with Deep Reinforcement Learning." ArXiv abs/1312.5602 (2013).

[2] Mnih, Volodymyr et al. "Human-level control through deep reinforcement learning." Nature 518 (2015): 529-533.

[3] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." AAAI. 2016.

[4] Lillicrap, Timothy P. et al. "Continuous control with deep reinforcement learning." CoRR abs/1509.02971 (2015).

[5] Silver, David et al. "Deterministic Policy Gradient Algorithms." ICML (2014).

[6] OpenAI. "OpenAI Gym Documentation." Retrieved from Getting Started with Gym: https://gym.openai.com/docs/.

[7] PyTorch. "Reinforcement Learning (DQN) Tutorial." Retrieved from PyTorch Tutorials: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.