Lab 8: Deep Q-Learning and Deep Deterministic Policy Gradient

Lab Objective:

In this project, you need to reconstruct and train two neural networks that are able to play CartPole-v0 and Pendulum. One of the training method is Deep Q-learning (DQN), which is a variation of Q-learning. The other training method is deep deterministic policy gradient (DDPG) that involves an actor and a critic. Actor is used to select action whereas critic is used to estimate $Q(s, a)$.

Important Date:

1. Experiment Report Submission Deadline: 6/6 (Thu) 12:00

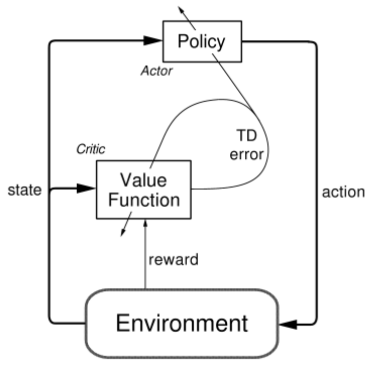
Turn in:

1. Experiment Report (.pdf)
2. Source code [NOT includes your weight]

Notice: zip all files in one file and name it like「DLP_LAB8_your studentID_name.zip」,
ex: 「DLP_LAB8_0586036_何國豪.zip」

Lab Description:

- Learn how to combine Q-learning with neural network to do reinforcement learning
  - Implement Deep Q network
- Understand off-policy reinforcement learning algorithm and the benefits
  - Behavior network and target network
- Learn how to use experience replay mechanism
- Learn how to combine policy gradient with neural network
  - Network design/construction of actor and critic
- Implement Deep Deterministic Policy Gradient (DDPG) algorithm
  - Understand the difference of deep q learning between DDPG.
  - Update the networks by "soft" target updates



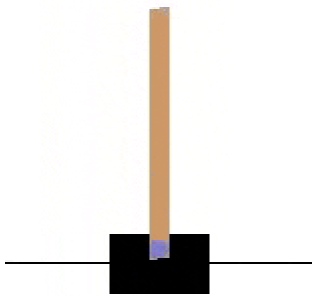
Actor-Critic Architecture

Requirements:

- Implement Deep Q Network
  - Construct the neural network
  - Target value, loss function
  - Action prediction with DQN
- Implement Deep Q-learning algorithm
  - Calculate target Q values
  - Update algorithm for the network
- Understand how actor-critic algorithm works.

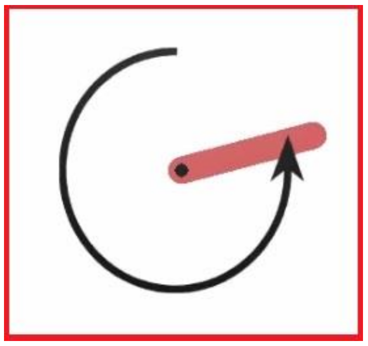
Game Environment – CartPole-v0:

- Introduction: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every time step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.
  - Actions: Left (0) or Right (1)
  - Observation: [Cart Position, Cart Velocity, Pole Angle, Pole Velocity at Tip]
  - Reward: Every timestep +1



Game Environment – Pendulum-v0:

- Introduction: The goal is trying to keep a frictionless pendulum standing up.
- State:
  - cos(theta) min: -1.0 max: 1.0
  - sin(theta) min: -1.0 max: 1.0
  - theta dot min: -8.0 max: 8.0
- Actions:
  - Joint effort min: -2.0 max:2.0
- Reward: -(theta^2 + 0.1*theta_dt^2 + 0.001*action^2)



Implementation Details – CartPole-v0:

Network Architecture

- Input: Observation (4 elements, not images)
- First layer: fully connected layer (ReLU)
  - input: 4, output: 32
- Second layer: fully connected layer
  - input: 32, output: 2

Training Arguments

- Optimizer: Adam or RMSprop
- Learning Rate: 0. 0005
- Epsilon: 1 → 0.1 or 1 → 0.01
- Epsilon decay: 0.995
- Batch Size: 128
- Experience buffer size (Memory capacity): 5000
- Gamma (Discount Factor): 0.95
- Training Episode: 1000
- Update target network every 50 iterations

Methodology:

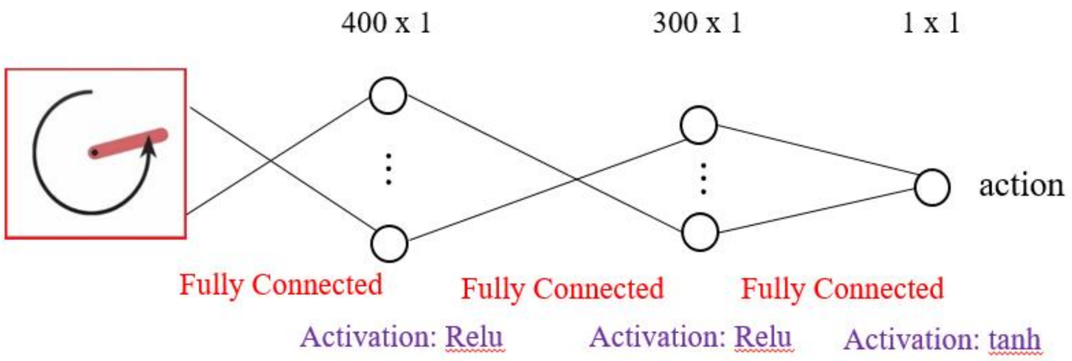
Algorithm - Deep Q-learning with experience replay

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $D$
      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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
      network parameters $\theta$
      Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

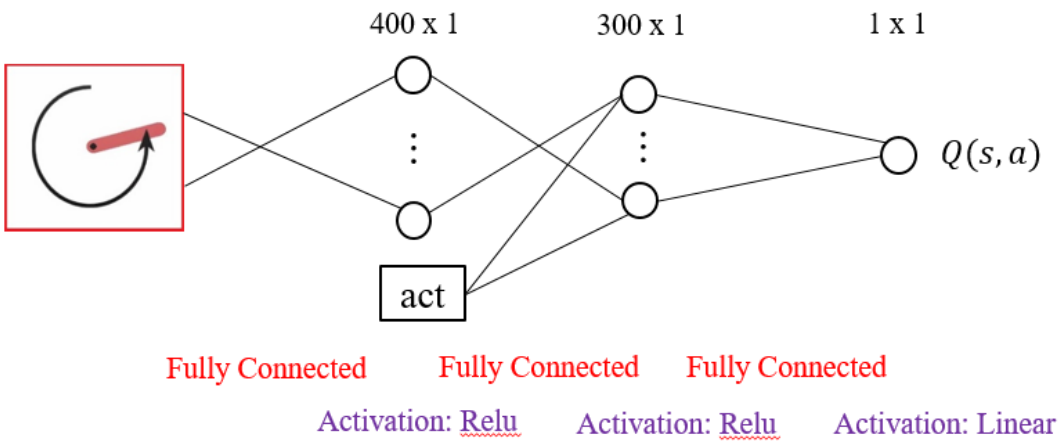
Implementation Details – Pendulum-v0:

Network Architecture

- Actor



- Critic



Training Arguments

- Optimizer: Adam
- Learning Rate (Actor): 0.0001
- Learning Rate (Critic): 0.001
- Tau: 0.001
- Batch Size: 64
- Experience buffer size = 10000
- Gamma (Discount Factor): 0.99
- Total training episode: 3500

Misc.

- Training Time: approx. 2 hours

Methodology:

Algorithm – DDPG algorithm

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize target network $Q'$ and $\mu'$ 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 $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$
        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:
            $\nabla_{\theta^\mu}\mu|s_i \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|s_i$
        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
```

Rule of Thumb:

- [CartPole-v0] The performance should greatly improve after training about 300 episodes. (reaching 100 points or more)
- [Pendulum-v0] The episode score should be able to reach -1000 after training 40~60 episodes.
- Don't set replay buffer size too big. If it costs more memory than your RAM, training process would become very slow.

Scoring Criteria:

- Report (80%)

- A plot shows episode rewards of at least 1000 training episodes in the game CartPole-v0 (5%)
- A plot shows episode rewards of at least 10000 training episodes in the game Pendulum (5%)
- Describe your implement/adjustment of the network structure & each loss function (10%)
- Describe how you implement the training process of deep Q-learning (10%)
- Describe the way you implement of epsilon-greedy action select method (10%)
- Explain the mechanism of critic updating (10%)
- Explain the mechanism of actor updating (10%)
- Describe how to calculate the gradients? (5%)
- Describe how the code work (the whole code) (10%)
- Other study or improvement for the project. (5%)

● Performance (20%)
- [CartPole-v0] Average reward during 100 testing episodes: Average ÷ 2.0
- [Pendulum-v0] Highest episode reward during training: (Highest + 700) ÷ 5

References:

[1] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (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." arXiv preprint arXiv:1509.02971 (2015).

[5] Lever, Guy. "Deterministic policy gradient algorithms." (2014).

[6] ShangtongZhang et al. "Highly modularized implementation of popular deep RL algorithms in PyTorch." Retrieved from Github: https://github.com/ShangtongZhang/DeepRL.

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

[8] OpenAI. "OpenAI Wiki for CartPole v0." Retrieved from Github: https://github.com/openai/gym/wiki/CartPole-v0.

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

[10] pemami4911 et al. "Collection of Deep Reinforcement Learning algorithms." Retrieved from Github: https://github.com/pemami4911/deep-rl.