LAB 6
Deep Reinforcement Learning

TA 施囿維

Deadline: 2022/5/24(Tue) 23:59

No Demo

In this lab,

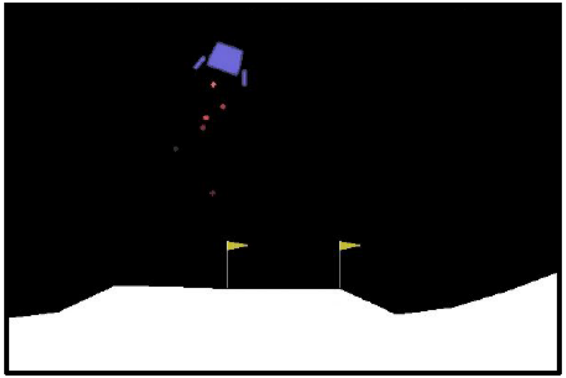
**Must use sample code,
otherwise no credit.**

Outline

1. Solve LunarLander-v2 using DQN
2. Solve LunarLanderContinuous-v2 using DDPG
3. Modify and Run Sample Code
4. Scoring Criteria
5. Reminders

LunarLander-v2

- Observation [8]
  1. Horizontal Coordinate
  2. Vertical Coordinate
  3. Horizontal Speed
  4. Vertical Speed
  5. Angle
  6. Angle Speed
  7. If first leg has contact
  8. If second leg has contact
- Action [4]
  1. No-op
  2. Fire left engine
  3. Fire main engine
  4. Fire right engine



- Action [2] (Continuous)
  - 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

Deep Q-Network (DQN)

Algorithm 1 – 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
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

TODO:

- 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

Deep Deterministic Policy Gradient (DDPG)

Algorithm 2 – Deep Deterministic Policy Gradient 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_i, a_i, r_i, s_{i+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

TODO:

- 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

3. Modify Sample Code

1. Find a #TODO comment with hints
2. remove the raise NotImplementedError

3. Run Sample Code

- Simply train and test: python dqn.py
- Only test and render: python dqn.py --test_only --render

4. Scoring Criteria

Show your work, otherwise no credit will be granted.

- Report (80%)
  - (DO explain; do not only copy and paste your codes.)
- Report Bonus (25%)
  - Implement and Experiment on Double-DQN (10%)
  - Implement and Experiment on TD3 (Twin-Delayed DDPG) (10%)
  - Extra hyperparameter tuning, e.g., Population Based Training. (5%)
- Performance (20%)
  - [LunarLander-v2] Average reward of 10 testing episodes: Average ÷ 30
  - [LunarLanderContinuous-v2] Average reward of 10 testing episodes: Average ÷ 30

5. Reminders

- Your network architecture and hyper-parameters can differ from the defaults.
- Ensure the shape of tensors all the time especially when calculating the loss.
- Be aware of the indentation of hints.
- When testing DDPG, action selection need NOT include the noise.

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 DoubleQ-Learning." AAAI. 2016.
4. Lillicrap, Timothy P. et al. "Continuous control with deep reinforcement learning." CoRRabs/1509.02971 (2015).
5. Silver, David et al. "Deterministic Policy Gradient Algorithms." ICML (2014).
6. Fujimoto, S., Hoof, H.V., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. ArXiv, abs/1802.09477.
7. OpenAI. "OpenAI Gym Documentation." Retrieved from Getting Started with Gym: https://gym.openai.com/docs/ .
8. OpenAI. "OpenAI Wiki for Pendulum v0." Retrieved from Github: https://github.com/openai/gym/wiki/Pendulum-v0 .
9. PyTorch. "Reinforcement Learning (DQN) Tutorial." Retrieved from PyTorch Tutorials: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html .