UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ROBOTICS
AND COGNITIVE SYSTEMS

Reinforcement Learning (SS2020)
Honghu Xue
xue@rob.uni-luebeck.de
Handout Date: 11. June 2020
Submission Deadline: 25. June 2020
Submission Deadline for bonus tasks: 16. July 2020

Reinforcement Learning - Assignment IV (15 pt + 10 pt)

The reference link for this assignment is illustrated in the Literature part below. Please fetch the python files for the programming part.
In this assignment, we transit from discrete state space to a continuous one, whereas the action space still remains discrete. To meet these changes, we resort to function approximation approaches, where the value estimate for $V_\pi(s)$ or $q_\pi(s,a)$ are approximated by $\hat{V}(s,w)$ or $\hat{q}(s,a,w)$, and w are the weights to be learned. In the book of *RL, an introduction*, some conventional approaches are introduced, where the raw state or state-action pair are first transformed into some pre-defined feature space $x(s)$ or $x(s,a)$ and then multiplies the weight vector w to get the approximated state value or state-action value. However, the approaches of feature encoding introduced in Chapter 9.5 in *RL, an introduction* can only work in low-dimensional state space, as the feature dimension increases rapidly with the raw input state dimension. In Sergey's lecture, they use more expressive function approximators i.e. *Neural Networks* to directly map the raw states to the approximated V-value or q-value.
In function approximation, more factors need reckoning with. Please make sure you understand Sergey Levine's lecture and then proceed to the tasks. With deep neural networks as function approximators, we move to the field of *Deep Reinforcement Learning*. For this assignment, we focus on value-based approximation methods using neural networks i.e. *Deep Q-Network* and its variants e.g. *fitted Q-iteration* and *Double Deep Q-Network* and their theoretical backgrounds.

Please note: For the bonus task, you can submit it separately from the non-bonus task, and its deadline is July.16th. Hope you enjoy the homework and good luck!

Task I: Theoretical understanding on Function Approximation approaches

1. When using function approximation, what are the conditions for convergence? (Hint: Please consider the case of on/off-policy, linear/non-linear function approximation, bootstrapping and Monte-Carlo estimation) (1 point, from the book of RL)

2. How are *semi-gradient methods* different from true *gradient descent* when updating the weights w in function approximation? (1 point, from the book of RL)

3. (1) In Deep Q-Network (DQN), what kind of loss needs minimizing? Write down the math expression. (2) What is benefit of replay buffer in DQN? What is the purpose of the target network in DQN? (3 points)

4. Explain the difference between DQN and *fitted Q-iteration* in words. (1 point)

5. What are respectively the *hard-update* and *soft-update* (also called *polyak averaging*) of the target network in DQN? Provide mathematical formulae for *soft-update*. (2 points)

6. Explain what is the maximization bias in DQN or q-learning? How to mitigate this problem? (1.5 points)

Task II: Programming part on DQN and DDQN

From this assignment on, we need to use **Pytorch** for function approximation methods, a neuron network package in Python. Please first check if your (Nvidia) graphics card supports any one of the Cuda verison 9.2, 10.1 or 10.2 or not.
If yes, then you can install Pytorch locally. The installation steps are as follows: (1) Download and install Cuda (2GB), which enables the large training acceleration by using GPU. (2) Copy the command of the correct version from the official website `https://pytorch.org/` to Anaconda Powershell to install Pytorch.
If your computer graphics card doesn't support the above Cuda version, please directly use Google Colab to finish all the rest assignments. Don't forget to set 'Hardware accelerator' to be 'GPU' at Colab's side. You need to copy the .py file content into the code blocks, but no need to install Pytorch on Google Colab as it has Pytorch installed by default.

1. Implement the 'classic' *DQN* on 'MountainCar-v0' with the version of no episodic length limit. Run the algorithm 5 times (one run per group member on average) and **plot the learning curves (loss not needed) in 5 subplots with the title of each subplot to be team member name who does the experiment**. The trained policy should register the episodic reward of around -150. A successful training requires around 70000 frames, which roughly takes 7 minutes on Colab. (3 points)

2. We now move to a more interesting gym[box2d] environment 'LunarLander-v2'. Here, you need to implement the *Double-DQN* (*DDQN*) on 'LunarLander-v2'. Run the algorithm 5 times (one run per team member on average) and **plot the learning curves (loss not needed) in 5 subplots in the form of scatter plot with the title of each subplot to be team member name who does the experiment**. You can expand the code directly in the '.py' file above. Answer the following question: in practice, how are the two q-networks defined in *DDQN*. For Colab users, you can only see the progress of improvement. If your program finally converges to the episodic reward larger than 250, then the training is regarded successful. (Hint: **First test your *DDQN* algorithm on the easy task of MountainCar to check if it is correct implementation, then on LunarLander**). A successful training requires around 100000 frames, which roughly takes 12 minutes on Colab. (2.5 points)

Bonus Tasks (10 pt)

We now extend the *DQN* to raw-pixel image input, which is definitely of higher state dimension than the above tasks.

1. Here are some empirical questions. Even if you fail the subsequent programming task, you can still answer this part. (1) How to process the raw-pixel images to achieve better performance ? How to reduce the replay buffer storage space for images? (2) Can one frame of image satisfy the Markovian property and reason why. If not, how is the state defined? (3) Mention the network architecture for mapping raw-pixel image input to the q-value. (3 points)

2. Implement *DQN* on one OpenAI Atari Game 'PONG-v0' (with image input), show the learning curve. For this you don't need to plot mean and variance scheme, instead, one subplot for each student who does the task. **You can refer to any online resources, please only use Pytorch implementations**. Be patient with training, since around 500K of frames are needed to see improvements. For debugging purposes: If in 300K samples, the agent doesn't show any stable improvement. Probably your implementation is wrong.
   Hint: (1) Try to avoid using pooling layer in the task, as some work shows that pooling layer used in Deep RL reduces information and finally fails to learn the task. Downscale can be achieved using conv layer with strides. (2) Deep RL is highly sensitive to hyper-parameter settings, e.g. learning rate, network design, reward design and so on. (luckily, rewards are already processed from gym) To guarantee convergence, please do refer to the settings of online resources. (3) The 'wrappers.py' provides a way to partially pre-process the images (set the frame-skip rate, and define the end of an episode as losing one life or losing complete lives...) Please refer to the last literature for details. And you also need to modify some parts in 'wrappers.py'. (7 points)

Literature

The following reference material is relevant for this assignment. Each reference have some **overlapping content** with the other, so you don't need to go through all of the reference material below, just choose wisely.

- Sergey Levine's online *Deep Reinforcement Learning* lecture 7, 8 (highly-recommended, you can skip the part of Q-learning with continuous actions in lec 8 and some part you already knew in lec 7)

Reinforcement Learning (SS2020)
Honghu Xue
xue@rob.uni-luebeck.de
Handout Date: 11. June 2020
Submission Deadline: 25. June 2020
Submission Deadline for bonus tasks: 16. July 2020

- **Reinforcement Learning - an introduction, second edition** Chapter 9.1-9.4, Chapter 11.1-11.3, Chapter 6.7 (maximization bias), Chapter 9.7 (optional, simple introduction to *Artificial Neural Network*)
- Online material on *DQN*, *DDQN* (for Task I, Q3 and Q5)
  `https://www.efavdb.com/dqn`
  `https://greentec.github.io/reinforcement-learning-second-en/` (only the subsection : DQN)
  `https://greentec.github.io/reinforcement-learning-third-en/` (subsection: Hyperparameter adjustment, weight initialization, soft update (target network), Double DQN)
- Parametric ReLU (PReLU, optional)
  `https://medium.com/@shoray.goel/prelu-activation-e294bb21fefa`
- Pytorch basic operations (for programming part)
  `https://jhui.github.io/2018/02/09/PyTorch-Basic-operations/`
- Explanation for Gym wrappers on Atari Games (only for bonus task)
  `https://books.google.de/books?id=BAlnDwAAQBAJ&pg=PA124&lpg=PA124&dq=NoopResetEnv(gym.Wrapper)&source=bl&ots=zgSwjdNN5E&sig=ACfU3U2QdEnLh3Jd5-ZcXUYNW-iccREuGQ&hl=de&sa=X&ved=2ahUKEwia68rN2ILmAhWQy6QKHReODwsQ6AEwAnoECAgQAQ#v=onepage&q=NoopResetEnv(gym.Wrapper)&f=false`