Probabilistic Artificial Intelligence Task 4: Reinforcement Learning

1 Task Description

Your task is to develop and train a Reinforcement learning agent which will swing up an inverted pendulum from an angle of π (downward position) to 0(upward position) and try to hold it there Figure 1. To swing-up the pendulum, agent has a motor that can apply torques u in range of $[-1, 1]$, i.e., $u \in [-1, 1]$. You will implement an off-policy RL algorithm, such as DDPG or SAC, to solve this control task. Furthermore, we provide you with a simulator of the pendulum, with which your RL agent can interact and learn a control policy for the task.

This '.pdf' is meant to give a task-specific information. For logistics on the submission and setup, see the description on the task webpage. In this document, we will first explain the task environment and scoring. Secondly, we will outline the code structure provided in the code template.

2 Environment and Scoring Details

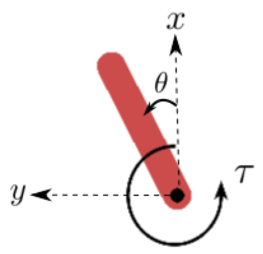

Figure 1: Pendulum from the OpenAI gym control suite.

At each discrete time step t, the controller can control the pendulum by applying a torque u_t. We represent the state $\boldsymbol{x}_t$ of the pendulum with a three dimensional vector.

$$\boldsymbol{x}_t = [\cos(\theta_t), \sin(\theta_t), \dot{\theta}_t]^\top.$$

The goal is to accumulate as much reward as possible in 200 timesteps. During learning, after 200 timesteps the episode ends and the environment is reset to a random initial state. The reward at time step t is given by:

$$r_t = -(\theta_t^2 + 0.1\dot{\theta}_t^2 + 0.001u^2)$$

Since the focus of this task is the implementation of reinforcement learning algorithms, it is not necessary to have a detailed understanding of the pendulum environment beyond the observation and action space sizes (given in the Agent class of 'solution.py'). However, if you are interested you can read more about the environment here.

When you run 'solution.py' your algorithm will have access to the standard pendulum environment that is commonly used in evaluating RL algorithms. To run 'solution.py' you will need to install the packages listed in 'requirements.txt'. You are encouraged to run 'solution.py' for testing.

In a single run of 'runner.sh', your agent will be trained for 50 episodes in order to learn a policy. The pendulum's dynamics in the 'runner.sh' will stay the same for the public score evaluation. However, for private score evaluation, dynamics will be modified from the one we provide you in 'solution.py', to prevent overfitting on the public one and check the generalization ability of your model. Each individual episode will contain exactly 200 transitions. The score is then estimated as the mean return of the learned policy over 300 episodes after training. The final score S will be

$$S =_{i\in\{1,\ldots,300\}} \sum_{t=1}^{T} R_t^i.$$

where R_t^i is the reward obtained at the timestep t, episode i of the evaluation, when using your final policy. During the evaluation, upon completion of each episode i, the environment is reset to the same initial state $\boldsymbol{x}_0 = [-1, 0, 0]^\top$. Your goal is to maximize this final score and beat the given baselines. Public baseline scores are:

- Easy baseline: -739.2
- Medium baseline: -534.0
- Hard baseline: -389.1

Public baselines are just a reference, private ones, which stay hidden, will be used for grading the task.

3 Solution Details

As stated in the task description, the code for this task is contained in 2 files: 'solution.py' containing template code and 'utils.py' where additional methods and functionalities for the task are stored. Classes *NeuralNetwork*, *Actor*, *Critic*, and *Agent*, defined in the main solution file, are supposed to be filled out by you. Changes in other methods of the template file, like *main()* or any method in utils.py WILL NOT take any effect during evaluation.
The core of your solution should be implemented in the *Agent* class, more specifically, methods *train_agent()* and *get_action()*, which will be called by our checker. The definition and arguments for the agent class constructor and two aforementioned methods MUST NOT be changed. Other methods in *Agent* or other classes are provided as additional structures that could guide you through your implementation. However, you are free to introduce your classes and methods or change the existing ones in any way, as that won't interfere with the final evaluation procedure. Still, we strongly recommend using our template classes and code for your implementation.

4 Hints

A successful implementation of an off-policy algorithm such as SAC or its modifications is sufficient to pass the hard baseline.