Reinforcement Learning WS 2018/19
Instructor: Sebastian Blaes `<sblaes@tuebingen.mpg.de>`
Due-date: 09.11.2018 (hand in at beginning of recitation session)
Exercise Sheet 3
November 1, 2018

The goal of this homework is to learn how to use Tensorflow and OpenAI Gym and to get experience iwth imitation learning using behavioral cloning and DAgger.

1 Preparation

The starter code can be found at our website `ex3-immitation_learning.zip`. There are three dependencies described below, which we need for some of the future exercises. The easiers is to install python3 and then install the rest with `pip install`, see the README file. The MuJoCo simulation you have to download ask for a student license.

(a) TensorFlow: Use `pip` or follow the instructions at `https://www.tensorflow.org/get_started/os_setup` to install TensorFlow. We will not require a GPU for our exercises. If you have one, you may want to consider installing the GPU version along with CUDA and CuDNN. However for the small networks we are having here it typically does not bring a speedup.

(b) OpenAI Gym: We will use environments in OpenAI Gym, a testbed for reinforcement learning algorithms. For installation and usage instructions see `https://gym.openai.com/docs` or install via `pip`.

(c) MuJoCo: We will use MuJoCo for physics simulation in this and a few other exercises. Download version 1.50 from `http://mujoco.org/`, and install version 1.50.1.56 from `https://github.com/openai/mujoco-py/` (or via `pip`). Request a student license from `https://www.roboti.us/license.html`.

2 Behavoral Cloning

(a) The starter code provides an expert policy for each of the MuJoCo tasks in OpenAI Gym (See `run expert.py`). Generate roll-outs from the provided policies, and complete the implementation of behavioral cloning in `behavioral_cloning.py`. There are hints in the code. It will be helpful to go through a tensorflow tutorial, e.g. at`http://www.easy-tensorflow.com/`.

(b) Run behavioral cloning (BC) and report results on two tasks one task where a behavioral cloning agent achieves comparable performance to the expert, and one task where it does not. When providing results, report the mean and standard deviation of the return over multiple rollouts in a table, and state which task was used. Be sure to set up a fair comparison, in terms of network size, amount of data, and number of training iterations, and provide these details (and any others you feel are appropriate) in the table caption.

(c) Experiment with one hyperparameter that affects the performance of the behavioral cloning agent, such as the number of demonstrations, the number of training epochs, the variance of the expert policy, or something that you come up with yourself. For one of the tasks used in the previous question, show a graph of how the BC agents performance varies with the value of this hyperparameter, and state the hyperparameter and a brief rationale for why you chose it in the caption for the graph.

3 DAgger

(a) Implement DAgger. See the code provided in `run expert.py` to see how to query the expert policy and perform roll-outs in the environment.

(b) Run DAgger and report results on one task in which DAgger can learn a better policy than behavioral cloning. Report your results in the form of a learning curve, plotting the number of DAgger iterations vs. the policys mean return, with error bars to show the standard deviation. Include the performance of the expert policy and the behavioral cloning agent on the same plot. In the caption, state which task you used, and any details regarding network architecture, amount of data, etc. (as in the previous section).

(c) (Optional) Alternative Policy Architectures: Experiment with a different policy architecture, e.g. using recurrence or changing the size or nonlinearities used. Compare performance between your new and original policy architectures using behavioral cloning and/or DAgger, and report your results in the same form as above, with a caption describing what you did.