Deep RL Assignment 1: Imitation Learning

Fall 2018

due September 5th, 11:59 pm

The goal of this assignment is to experiment with imitation learning, including direct behavior cloning and the DAgger algorithm. In lieu of a human demonstrator, demonstrations will be provided via an expert policy that we have trained for you. Your goals will be to set up behavior cloning and DAgger, and compare their performance on a few different continuous control tasks from the OpenAI Gym benchmark suite. Turn in your report and code as described in Section 5.

Section 1. Getting Set Up

The starter code can be found at `https://github.com/berkeleydeeprlcourse/homework/tree/master/hw1`. There are three dependencies described below. For this assignment and others in the course, you may use whatever deep learning library you wish, but your code should be in Python. We would strongly appreciate avoiding unusual dependencies (beyond Theano, TensorFlow, and other standard DL libraries). If your code does have particular additional dependencies, please be sure to document these in your submission email so that we can run your code.

1. **TensorFlow**: Follow the instructions at `https://www.tensorflow.org/get_started/os_setup` to install TensorFlow. We will not require a GPU for assignments in this course, but if you have one, you may want to consider installing the GPU version along with CUDA and CuDNN. Note that you don't have to use TensorFlow for your implementation, but you do need it to run the expert policy for imitation.

2. **OpenAI Gym**: We will use environments in the OpenAI gym, a testbed for reinforcement learning algorithms. For installation and usage instructions see `https://gym.openai.com/docs`. Download version 0.9.1.

3. **MuJoCo**: We will use MuJoCo for physics simulation in this assignment. Download version 1.31 from `mujoco.org` and obtain a license from the course instructors.

Section 2. Behavioral Cloning

1. 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 implement behavioral cloning.

2. 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.

3. 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 agent's 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.

Section 3. DAgger

1. 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.

2. 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 policy's 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).

Section 4. Bonus: Alternative Policy Architectures

1. (Optional) 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.

Section 5. Turning it in.

1. Your report should be a PDF with 1 or 2 pages, containing 3 things: Table 1 for a table of results from question 2.2, Figure 1 for question 2.3, and Figure 2 for question 3.2. Optionally, you may include a table or figure for question 4.1.

   You do not need to write anything else in the report, just include the figures with captions as described in each question above. See the handout at `http://rail.eecs.berkeley.edu/deeprlcourse/static/misc/viz.pdf` for notes on how to generate plots.

2. In addition to the pdf, you should turn in one zip file with your code any any special instructions we need to run it to produce each figure or table below (e.g. "run python myassignment.py -sec2q1" to generate the result for Section 2 Question 1).

3. Turn in your assignment on Gradescope.

Answer Key for Exam A

Section 1. Getting Set Up

The starter code can be found at `https://github.com/berkeleydeeprlcourse/homework/tree/master/hw1`. There are three dependencies described below. For this assignment and others in the course, you may use whatever deep learning library you wish, but your code should be in Python. We would strongly appreciate avoiding unusual dependencies (beyond Theano, TensorFlow, and other standard DL libraries). If your code does have particular additional dependencies, please be sure to document these in your submission email so that we can run your code.

1. **TensorFlow**: Follow the instructions at `https://www.tensorflow.org/get_started/os_setup` to install TensorFlow. We will not require a GPU for assignments in this course, but if you have one, you may want to consider installing the GPU version along with CUDA and CuDNN. Note that you don't have to use TensorFlow for your implementation, but you do need it to run the expert policy for imitation.

2. **OpenAI Gym**: We will use environments in the OpenAI gym, a testbed for reinforcement learning algorithms. For installation and usage instructions see `https://gym.openai.com/docs`. Download version 0.9.1.

3. **MuJoCo**: We will use MuJoCo for physics simulation in this assignment. Download version 1.31 from `mujoco.org` and obtain a license from the course instructors.

Section 2. Behavioral Cloning

1. 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 implement behavioral cloning.

2. 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. TODO

3. 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 agent's 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. TODO

Section 3. DAgger

1. 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.

2. 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 policy's 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). TODO

Section 4. Bonus: Alternative Policy Architectures

1. (Optional) 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.

Section 5. Turning it in.

1. Your report should be a PDF with 1 or 2 pages, containing 3 things: Table 1 for a table of results from question 2.2, Figure 1 for question 2.3, and Figure 2 for question 3.2. Optionally, you may include a table or figure for question 4.1.

   You do not need to write anything else in the report, just include the figures with captions as described in each question above. See the handout at `http://rail.eecs.berkeley.edu/deeprlcourse/static/misc/viz.pdf` for notes on how to generate plots.

2. In addition to the pdf, you should turn in one zip file with your code any any special instructions we need to run it to produce each figure or table below (e.g. "run python myassignment.py -sec2q1" to generate the result for Section 2 Question 1).

3. Turn in your assignment on Gradescope.