CSC2626: Assignment 2
Due October 18th at 6pm ET
25 points

Setup

Starter Code Run `git clone https://github.com/florianshkurti/csc2626w22.git` to get the starter code for this assignment. This should create the directory `csc2626w22a2`. All of your work for this assignment will be under `csc2626w22a2/assignments/A2`.

Virtualenv This is optional but encouraged. Set up a virtualenv to encapsulate the dependencies of this project. You can do this with `virtualenv -p python3 myenv` or, if you use anaconda, `conda create -n myenv python=3.7`.

Dependencies Follow the instructions in the `README.md` file contained under `csc2626w22a2/assignments/A2` in order to install remaining dependencies.

1 Conservative Q-Learning (10 points)

The goal of this question is to help you get familiar with offline reinforcement learning, and in particular with the implementation of Conservative Q-Learning `https://arxiv.org/abs/2006.04779`.

Provide a summary, up to 2 pages, of the main ideas and assumptions in the paper. Ensure that you provide a description of the main equations and theorems and describe the significance of the experimental results with respect to related baseline methods. In addition, make sure to address the following questions in your summary, by looking at the reference implementation[1].

- Which equation in the paper was used to implement the CQL update rule in the reference implementation?
- How was this equation derived in the paper?
- How is the equation above implemented for discrete systems and continuous systems?
- Do the CQL lower bounds hold upon convergence of the CQL update rule, or also during each Q iteration?
- How is the policy learned? Is there another way it could have been computed?
- (Bonus Question) What directions for future work would you consider in order to improve, extend, or make use of this method or the lower bounds that it guarantees?

2 Dynamic Movement Primitives (15 points)

In this question, you will use Dynamic Movement Primitives (DMPs), to learn a policy which enables a 6DOF robot arm (UR10) to catch a ball. See Figure 1.
You will be provided with a set of demonstration trajectories collected from an expert policy, which you will use to train the parameters of the DMP.

[1] `https://github.com/aviralkumar2907/CQL/blob/master/d4rl/rlkit/torch/sac/cql.py`

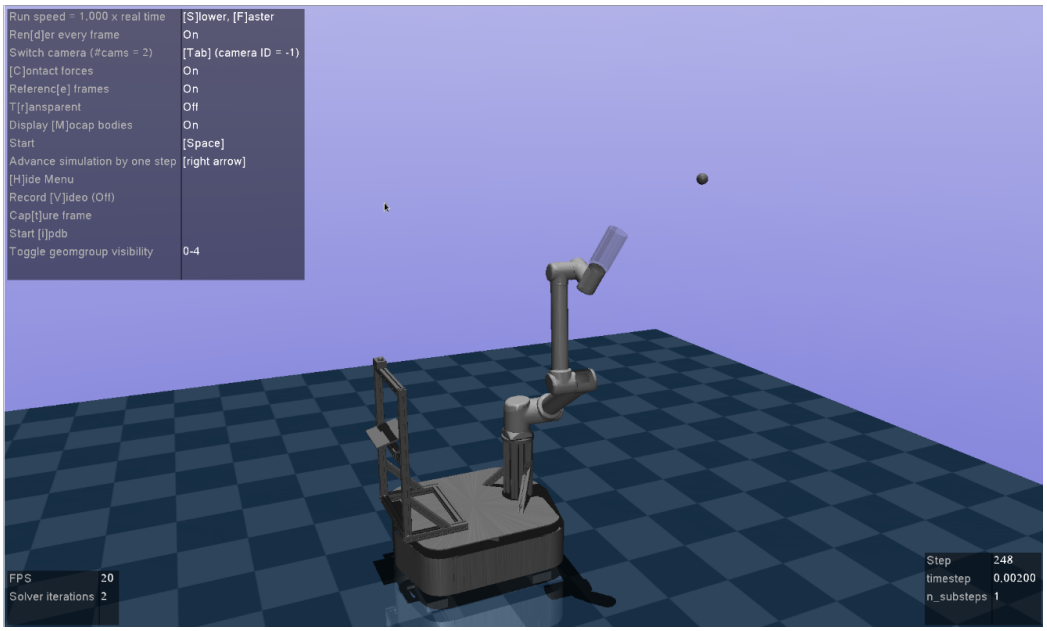

Figure 1: A screenshot from the `ballcatch-v0` environment.

2.1 Implementing Dynamic Movement Primitives (10 points)

You will be implementing the version of DMPs that is described in Pastor et al. [2009].

1. **Learning a DMP**: Fill in the missing code for the learning procedure `DMP.learn` in `dmp.py`. This instance method takes as input a set of trajectories and solves a least squares problem to set the weight parameters of a DMP.

   The input to this method is two numpy arrays, one with shape (N, T, D) and the second with shape (N, T) where N is the number of demonstrations, T is the number of timesteps, and D is the number of degrees of freedom in the trajectory. Note: that each degree of freedom is to be treated as a different DMP with a different set of weights, but all of them share the same phase variable.

2. **Querying a DMP:** Fill in the code for the planning procedure `DMP.execute` in `dmp.py`.

2.2 Using Dynamic Movement Primitives (5 points)

You will be working with a small dataset of expert demonstrations provided under `data/demos.pkl`. Each element i in the dataset is a numpy array of shape $(T_i, 6)$, where T_i is the number of timesteps in trajectory i. For each timestep, the demo contains the joint angles for the 6DOF robot in figure 1. Note that each timestep corresponds to a duration of 0.04 seconds.

1. **Interpolation of Variable Length Trajectories**: Each trajectory in the dataset has a different length T_i. Fill in the code in `dmp._interpolate` in `dmp.py` to perform a linear interpolation of the trajectories such that all of them will have the same length. Note: you can use `scipy.`$_{i}nterpolate for this$.

2. **Reconstruction using DMP**: Use your implementation from above to train a DMP **only** on the first demonstration in the dataset (i.e. $i = 0$). (For this question, keep all the hyperparameters in their default setting.) Then try to reconstruct the trajectory. You can use `DMP.rollout` function for this, setting the initial position x_0 to the first timestep of the trajectory, and the goal to the final timestep. Visualize the reconstruction by plotting the demo trajectory and reconstruction for each degree of freedom separately. You will find boilerplate for this in `q2_recons()` in `main.py`. To run your code, use `python main.py recons`

3. **Tuning**: Train a DMP on the full set of trajectories. Experiment with different settings of the number of basis functions `DMP.nbasis` and the gain parameter `DMP.K_vec`. You can evaluate the fit quantitatively using RMSE against the demos. Once you have determined appropriate settings, fill in the relevant code under `q2_tuning()` in `main.py`. Finally, run `python main.py tuning`. Note: This will save your DMP to disk, so it can be used in the remaining questions.

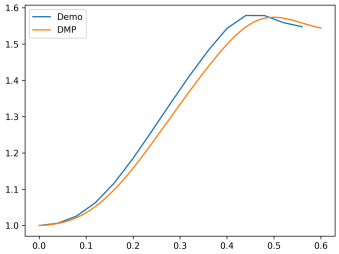

Figure 2: DMP reconstruction of demo0 degree of freedom 0.

4. **Closed Loop Policy**: So far, we have used the DMP in to plan open loop trajectories. This assumes that the robot's motion is perfect, and ignores the effects of inertia, torque limits, etc. When we try to execute the policy on a real (or in this case, simulated) robot, we may end up veering off the planned trajectory. DMPs provide a natural way of representing closed loop policies which correct themselves based on the position at the current time. Fill in the missing lines in `BallCatchDMPPolicy.select_action` in `policy.py` to do this.

5. **Catch some balls**: Now, you will plug your DMP into the Mujoco simulator for ball catching, and see how well it does. To do this, simply run `python main.py`. This should launch the mujoco simulator, and use your saved DMP and closed loop policy to drive the arm. Report the average reward and success rate across 20 episodes. Record a video of the results to submit along with your report.

Submission:

Submit a file called `a2_q1_firstname_lastname_studentid.zip` on Quercus. This should contain the following:

- a file called `cql.pdf`, containing your summary of CQL.
- `dmp.py`
- `main.py`
- `policy.py`
- `recons[1-6].png` - the visualization from the reconstructed trajectory
- `ballcatch.mp4` - the video from the final question in the report

References

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768, 2009. doi: 10.1109/ROBOT.2009.5152385.