Robot Autonomy - Homework 4

Professor: Oliver Kroemer

1. Introduction

This homework will focus on reinforcement learning for a block pushing robot. We will implement two methods in this homework, the Cross Entropy Method and the Relative Entropy Policy Search method. The block pushing is performed in the MuJoCo simulator. You are provided with two files, one called FrankaRollout.py and another called Main.py. The two files are described below.

2. Modules

2.1 FrankaRollout.py

FrankaRollout.py: This module implements a class called FrankaSim that is specifically designed for the block pushing experiment. The class is instantiated with a path to the MJCF model and a list of joints that we will be using to conduct the experiment. In order to simplify the task, we will be attempting to push the block only using the second and fourth joints. The handle to an instance of this class has already been instantiated for you in Main.py with the correct choice of joints.

For this learning task, we will be attempting to learn the coefficients of a cubic polynomial trajectory for each of the two joints (joint 2 and joint 4). As explained in the lecture, the coefficients of a cubic polynomial can be solved for given the initial position and initial velocity, and the final position and final velocity at the initial and final times respectively for a specific joint. We will choose to represent our state-action policy using a multi-variate normal distribution over these parameters, i.e. in our case for two joints, this will be a multivariate normal distribution over initial position, initial velocity, final position, final velocity and final time for joint 2 and initial position, initial velocity, final position, final velocity and final time for joint 4. This gives us a ten variable vector space.

The attributes of this class have been commented with the their corresponding descriptions. The initial attributes deal with simulation parameters and settings. For the learning task, the attributes to pay attention to are the following:

1. **`policyMu:`** This is where we store the mean that defines our normal distribution. We will be attempting to learn and update this mean by performing roll-outs in simulation and collecting samples.
2. **`policyCov:`** This is where we store the covariance matrix that defines our normal distribution. We will be attempting to learn and update this matrix by performing roll-outs in simulation and collecting samples.

2.2 Main.py

This module instantiates an object of class FrankaSim. This has already been done for you. All you need to do is complete the corresponding method belonging to FrankaSim and un-comment the call to the right method. You will be running Main.py for the homework.

3. MuJoCo Set Up

Creating a python “venv”

To set up MuJoCo,

1. Navigate to the home directory
2. Create a python virtual environment. To do this, run “**`python3 -m venv ./<your_env_name>”.`** Replace `your_env_name` with whatever you’d like.
3. Activate this environment by running the activate script within the bin folder of the venv path you provided, i.e. run “**`source ./<your_env_name>/bin/activate`**”.
4. Within this environment, run “**`which python`**” and ensure that the python path is pointing to a location in the venv path as a sanity check.
5. Install dependencies using the environments pip package by running,
   a) “**`python -m pip install mujoco”`**
   b) **`“python -m pip install quaternion”`**
   c) **`“python -m pip install matplotlib”`**
6. You can now use the packages from the Python venv to run the homework files. Make sure to activate the venv when you run the files.

4. Cross Entropy Method

For this part of the assignment you will need to implement the CEM algorithm in the method called "CEM" in FrankaSim. As mentioned before, we are using a normal distribution to represent our policy for the block pushing task. Our goal is to find the "right" distribution that consistently performs the desired action of block pushing. The pseudocode for this algorithm is shown below:

```
μ ← μ_0
Σ ← Σ_0
Plot_Rewards = []
i ← 1
for i ≤ num_of_updates do
    j ← 1
    Sample_Params = []
    Rewards = []
    for j ≤ num_of_samples do
        sample_param ← sample_policy()
        reward ← policy_rollout()
        Sample_Params.append(sample_param)
        Rewards.append(reward)
        Plot_Rewards.append(reward)
        j ← j + 1
    end for
    top_K_samples ← extract_samples_with_K_best_rewards(K=5, Sample_Params, Rewards)
    μ ← compute_mean(top_K_samples)
    Σ ← compute_covariance(top_K_samples)
    i ← i + 1
end for
```

In your implementation, set **`num_of_updates = 5`** and **`num_of_samples = 15`**. You will make use of some of the methods that we have provided to you in the FrankaSim class in order to implement this function. The **`sample_policy`** function returns a sample drawn from the normal distribution. After calling the sample policy function, you will call the **`policy_rollout`** method which will run the simulator, calculate the reward and return the reward. Finally, we have also provided a function called **`plot_rewards`** in the **`plotter.py`** file that you can pass the Plot_Rewards variable to in order to visualize the overall change in rewards using the policy. When sorting the top **`K`** samples based on the reward, use **`K=5`**, i.e. the top 5 best performing samples.

4 Relative Entropy Policy Search

For this part of the assignment, you will need to complete the method called “REPS” in the FrankaSim class. The algorithm structure is the same as for CEM. The computeETA method defined above the REPS method is meant to optimize the dual function to determine the η ”temperature” parameter. The optimization has largely been set up with scipy’s optimize.minimize function. However, to perform the optimization, the dual function needs to provided. Insert the dual function equation at the point marked TODO. Note that the function takes in eta (the parameter to be optimized), but it also has access to the returns R. Also note that, for numerical stability, we have subtracted the max return from all of the returns, and we should therefore include a +np.max(returns) to the dual function. The optimization will then compute the η value based on the distribution of observed returns from the previous batch of rollouts. Once the η parameter has been computed, the next step is to compute the weights of each of the samples based on their corresponding returns. These weights will subsequently be used for computing the mean and covariance matrix for the new updated policy. The difference in the algorithm between CEM and REPS appears in the re-weighting step. Use the optimal value of η to implement the sample weighting employed by REPS. Use the same plotter function as described in CEM to plot the rewards.

5 Submission

For this homework, submit a pdf page with the following items:

• The reward plot from running the CEM script

• The reward plot from running the completed REPS script

As well as answers to the following questions:

1. What trend do you see in the CEM reward plot? Why do you think this is?

2. Did REPS have improved performance over CEM? Why or why not?

3. What is one advantage and one disadvantage of each of the two methods?