1 Introduction

The goal of this assignment is to get experience with model-learning in the context of RL, and to use simple model-based methods (particularly, Model Predictive Control (MPC)) for controlling agents. The experiments you will run are based on (Nagabandi, 2017). \[1\]

2 Algorithm and Implementation

2.1 Algorithm

The algorithm you will implement is described in Algorithm 1. The exact rule for the MPC action-selection is described in Algorithm 2.

2.2 Code Setup

The following files are ones you are expected to modify:

- `main.py`
  - Contains the main loop which calls the rollout sampler, fits the dynamics model, and aggregates data.
  - You will implement the entire main loop. (Some structure is provided to guide you.)

---

Algorithm 1 Model-Based Control with On-Policy Data Aggregation

Sample a random set of $N_{\text{rand}}$ trajectories $D_{\text{rand}}$ from environment $\mathcal{E}$

Initialize dataset $D$ to $D_{\text{rand}}$

for $k = 0, 1, 2, \ldots$ do
  • Fit dynamics model $f_{\theta}$ according to
    \[
    \theta_k = \arg \min_{\theta} \frac{1}{N} \sum_{(s,a,s') \in D} \| f_{\theta}(s,a) - s' \|_2^2
    \]
    using the Adam optimization algorithm, starting from initial parameters $\theta_{k-1}$ (or if $k=0$, starting from random initial parameter values).
  • Sample a set of $N_{\text{rl}}$ on-policy trajectories $D_{\text{rl}}$ from $\mathcal{E}$ using a policy which selects actions according to Algorithm 2.
  • Aggregate data: $D = D \cup D_{\text{rl}}$.
end for

Algorithm 2 MPC Action Selection Using Dynamics Model $f_{\theta}$

input Initial state $s$, number of simulated rollouts $K$, path length (horizon) for simulated rollouts $H$, cost function on trajectories $C$, dynamics model $f_{\theta}$.

• Sample $K$ sequences of $H_{\text{mpc}}$ actions, $\{a^j_1, \cdots, a^j_H\}_{j=1,\ldots,K}$
• Use dynamics model $f_{\theta}$ to generate associated simulated rollouts:
  \[
  s^j_{i+1} = f_{\theta}(s^j_i, a^j_i),
  \]
  where for all $j$, $s^j_0 = s$.
• Use $C$ to evaluate fictitious trajectories $\tau^j = (s^j_0, a^j_0, \ldots, s^j_H, a^j_H, s^j_{H+1})$. Find the best trajectory, $j^* = \arg \min_j C(\tau^j)$.
• Return $a^j_0$.

• dynamics.py
  − Contains the dynamics model code.
  − The dynamics model object has two key methods: fit, which runs an iteration of the optimization algorithm, and predict, which performs inference using the learned model.
  − You will implement both of these.

• controllers.py
  − Contains the MPC controller code.
  − To produce an action for a given state, the MPC controller uses the learned dynamics model to generate imaginary rollouts using random actions, uses a cost
function to determine the best imaginary rollout, and selects the first action of
the best imaginary rollout.

– You will implement the action-selection process.

The file `cost_functions.py` contains functions you will use to evaluate the imaginary
rollouts generated with your learned dynamics model.

The file `cheetah_env.py` contains the environment (a half-cheetah robot) you will be
testing your code with.

The files `logz.py` and `plots.py` are utility files which you have used before (in homework
2), and you will not modify them.

After you fill in the blanks, you should be able to just run `python main.py` with some
command line options to perform the experiments. To visualize the results, you can run
`python plot.py path/to/logdir`. (Full documentation for the plotter can be found
in `plot.py`).

### 2.3 Implementation Details

- **When implementing `compute_normalization` in `main.py`:**
  - Make sure to produce **vector-valued means and stds** for the various quantities.
  - That is, you should have means and stds for each component of each of those
    vectors.

- **Use the AdamOptimizer to train the dynamics model.** For details on how many steps
  of gradient descent to take, we recommend that you study the experimental details in
  (Nagabandi, 2017).

- **When implementing the dynamics model:**
  - Pay careful attention to the keyword args for the dynamics model. The normal-
    ization vectors are inputs here, and you need these for normalizing inputs and
    denormalizing outputs from the model.
  - You want the neural network for your dynamics model to output **differences in
    states**, instead of outputting next states directly. Then using the estimated state
    difference $\hat{\Delta}$ and the current state $s$, you will predict the estimated next state $\hat{s}'$ according to:
    $\hat{s}' = s + \hat{\Delta}$.
  - How to use the normalization statistics: given a state $s$ and an action $a$, and
    normalization statistics $\mu_s, \sigma_s, \mu_a, \sigma_a, \mu_\Delta, \sigma_\Delta$ (where $\Delta = s' - s$), you want your
network to compute an estimate of the state difference $\hat{\Delta}$ according to

$$\hat{\Delta} = \mu_\Delta + \sigma_\Delta \odot f_\theta \left( \frac{s - \mu_s}{\sigma_s + \epsilon}, \frac{a - \mu_a}{\sigma_a + \epsilon} \right),$$

where $\odot$ is an elementwise vector multiply and $\epsilon$ is a small positive value (to prevent divide-by-zero).

- When implementing the MPC controller:
  - To evaluate the costs of imaginary rollouts, use `trajectory_cost_fn`, which requires a per-timestep `cost_fn` as an argument. Notice that the MPC controller gets a cost function as a keyword argument—this is what you should use!
  - When generating the imaginary rollouts starting from a state $s$, be efficient and **batch the computation**. At the first step, broadcast $s$ to have shape (number of fictional rollouts, observation dim), and then use that as an input to the dynamics model prediction to produce the batch of next steps.
  - The cost functions are also designed for batch computations, so you can feed the whole batch of trajectories at once to `trajectory_cost_fn`. For details on how, read the code.

### 3 Experiments

- Fit a dynamics model to random data alone and use the learned dynamics model in your MPC controller to control the cheetah robot. Report your performance (copy/paste the log output into your report).
- Run the full algorithm, including on-policy data aggregation, for 15 iterations. Make a graph of the performance (average return) at each iteration. How does performance change when the on-policy data is included?

### 4 Bonus

Choose any (or all) of the following:

- Use this method to get another robot to move forward - could be the swimmer, the ant or anything else.
- Implement a better way of choosing actions during MPC than random sampling, and show the difference in performance with this method.
- Any other algorithmic improvements to the dynamics model or the controller to improve sample complexity or performance.
5 Submission

Your report should be a one or two page document containing the results for your experiments from section 4 and all command line expressions you used to run your experiments.

Also provide a zip file including all of the files in your code, along with any special instructions needed to exactly duplicate your results.

Turn this in by October 18th 11:59pm by emailing your report and code to berkeleydeeprlcourse@gmail.com, with subject line “Deep RL Assignment 4”.