CS 287, Fall 2015 Optional Extra Credit Problems Policy Search for 2D Locomotion

---

Deliverable: pdf write-up by Wednesday December 16th, 11:59pm, submitted through Gradescope. Late days cannot be used for this extra-credit assignment. **Your pdf should be one page per subquestion, for a total of three pages. Thanks!**

Please refer to the class webpage for the homework policy. Various starter files are provided on the course website: `www.cs.berkeley.edu/~pabbeel/cs287-fa15/`.

Introduction

In this assignment, you will be using the MuJoCo simulator. To install MuJoCo, first download and unzip the starter files. Then add '`/<path-to-folder>/mujoco_linux`' to your library path:

- Linux: `export LD_LIBRARY_PATH=/<path-to-folder>/mujoco_linux:$LD_LIBRARY_PATH`
- Mac: `export DYLD_LIBRARY_PATH=/<path-to-folder>/mujoco_osx:$DYLD_LIBRARY_PATH`
- Windows: `set PATH=/<path-to-folder/mujoco_windows;%PATH%`

The last step is to run `setup_mj.m`.

The files `hopper_policy.m` and `hopper_mdp.m` contain the transition and reward functions for the MuJoCo 2D hopper. Similarly, the files `cartpole_policy.m` and `cartpole_mdp.m` contain the transition and reward functions for the MuJoCo cartpole.

The policies you will be working with in this assignment are represented as neural networks, composed of alternating affine (i.e., fully connected) and hyperbolic tangent layers. Thus, the parameters θ of the policy are the (flattened and concatenated) weights of the affine layers of the neural network. The input to the neural network is the current observation; i.e. a vector of size `observation_dim` x 1. The output of the neural network is a potential action, a vector of size `action_dim` x 1. The policy is stochastic. The chosen action is drawn from a Gaussian distribution centered at the output of the neural network, with covariance `diag(exp(log_`σ^2`))`. You can familiarize yourself with neural networks by doing Question 1 in the Guided Policy Search problem set.

Call `rollout_policy.m` to perform a single rollout of a given policy, starting at the initial state specified by the MDP. Use `p.set_theta_log_sigma(new_theta, new_sigma)` to change the parameters for policy `p` to `new_theta` and `new_sigma`. **Do not** set the policy parameters directly with `p.theta = new_theta`, because that does not update the parameters of the neural net appropriately.

Note: Make sure you leave enough time to run each of these algorithms before the deadline - for the hopper, CEM and CMA may take an hour of training to achieve forward progress, and PPO may take up to several hours.

Note: In order for the MuJoCo visualization to work on Linux, you have to run MATLAB without the Desktop display. To do this, run the command `matlab -nodesktop` from the terminal. Otherwise, MATLAB will crash when you run `mj plot`. Alternatively, you can comment out the `mj plot` line in `loco_starter.m`. The MuJoCo simulator will still work; there just will not be visualization of the cartpole or hopper.

1. Cross Entropy Method (CEM) [10pt]

In this question you get to implement the cross entropy method (CEM) to learn a locomotion (hopping) policy for the hopper. This is a popular gradient-free policy search algorithm, which you may have implemented for Problem Set 2 to do policy search for Tetris. Details on the algorithm are in Thiery and Scherrer[1].

[1] "Improvements on Learning Tetris with Cross Entropy." https://hal.inria.fr/inria-00418930/document

Depending on which reward function you use (there are many ways of measuring forward progress), you may end up with different styles of locomotion.

Since it takes a while for CEM to learn a policy for the hopper, we recommend that you first test your implementation on the cartpole. The interface is exactly the same (see `loco_starter.m`). Using the default hyperparameters in `cross_entropy_starter.m`, the cartpole should be able to learn to balance the pole in 10 iterations of CEM, which takes 30 seconds on a typical desktop computer. This corresponds to a maximum total discounted reward (across all sampled parameters) of 99.99. The average total discounted reward at this point will be lower, around 5, but will keep increasing as CEM continues running. The upper bound for total discounted reward is 99.996 – assuming a reward of 1 for each of the maximum number of timesteps (1000) and a discount factor of 0.99.

Q. Fill in `cross_entropy_starter.m` – look for the `YOUR CODE HERE` sections.

Deliverable: For the hopper, provide plots of the average and maximum total discounted rewards for the samples (`ce_avg_reward_for_samples` and `ce_max_reward_for_samples` in the code) over 100 iterations.

2. Covariance Matrix Adaptation (CMA) [10pt]

In this question you get to implement the covariance matrix adapatation (CMA) algorithm to learn a locomotion policy for the hopper. Like cross entropy, this is a gradient-free policy search algorithm.

The standard CMA algorithm is concisely described in Section 4.1 of Wampler and Popovic[2]. Note that there is a small typo in their paper: in Equation 27, $\mathbf{m}_j$ should be $\mathbf{m}_i$.

As for CEM, we recommend you first test your implementation of CMA on the cartpole. With a discount factor of 0.99, and all other hyperparameters the same as in `covariance_matrix_adaptation_starter.m`, CMA should be able to learn a policy for balancing the pole in 25 iterations, which takes 2 minutes on a typical desktop computer. This corresponds to a maximum total discounted reward (across all sampled parameters) of 99.99. The average total discounted reward will be around 10, and will continue increasing with further iterations of CMA.

Q. Fill in `covariance_matrix_adaptation_starter.m` – look for the `YOUR CODE HERE` sections.

Deliverable: For the hopper, provide plots of the average and maximum total discounted rewards for the samples, (`ce_avg_reward_for_samples` and `ce_max_reward_for_samples` in the code) over 100 iterations.

3. Policy Gradient - Proximal Policy Optimization (PPO) [10pt]

In this question, you get to implement policy gradients to learn a locomotion policy for the hopper.

We covered two types of policy gradient algorithms in class: path derivative and likelihood ratio. Since we do not know the dynamics for the hopper, we cannot use the path derivative method, but we can use the likelihood ratio method. We can improve upon the likelihood ratio method by also restricting the new policy to be not too "far away" from the original policy. One way to quantify this distance is through measuring the Kullbeck-Leibler divergence between the original and new policies.

In addition, the vanilla likelihood ratio policy gradient estimate also has high variance. We can compensate for this by subtracting out a baseline that approximates the value function (as mentioned in lecture).

The method you will implement for this question is called Proximal Policy Optimization (PPO). This is a variant of Trust Region Policy Optimization[3]. PPO is simpler because it solves an unconstrained optimization problem.

We recommend you first test your implementation of PPO on the cartpole. Using the default hyperparameters in `policy_gradient_ppo_starter.m`, PPO should be able to achieve an average total discounted reward of over 3 by iteration 6, and over 4 by iteration 12. Each iteration of PPO takes less than 2 minutes on a typical desktop computer.

[2] "Optimal Gait and Form for Animal Locomotion"
http://grail.cs.washington.edu/projects/animal-morphology/s2009/Optimal_Gait_and_Form_for_Animal_Locomotion.pdf

[3] http://arxiv.org/abs/1502.05477

Note: For PPO, the average over total discounted rewards is taken over the number of rollouts, not the number of timesteps. The total discounted reward for each rollout is calculated as expected: $\sum_{t=1}^{T} \gamma^{t-1} * r_t$, where γ is the discount factor, r_t is the reward gained at timestep t in the rollout, and T is the number of timesteps in the rollout.

Q. Fill in `policy_gradient_ppo_starter.m` – look for the `YOUR CODE HERE` sections.

Deliverable: For the hopper, provide plots of the average total discounted reward of the policy (`policy_theta_avg_reward` in the code) over 100 iterations.