Homework 1

CMU 10-403: Deep Reinforcement Learning and Control
(Spring 2020)
OUT: Jan. 31, 2020
DUE: Feb. 15, 2020 by 11:59pm

Instructions: START HERE

- **Collaboration policy:** You may work in groups of up to two people for this assignment. It is also OK to get clarification (but not solutions) from books or online resources after you have thought about the problems on your own. You are expected to comply with the University Policy on Academic Integrity and Plagiarism[1].
- **Late Submission Policy:** You are allowed a total of 10 grace days for your homeworks. However, no more than 3 grace days may be applied to a single assignment. Any assignment submitted after 3 days will not receive any credit. Grace days do not need to be requested or mentioned in emails; we will automatically apply them to students who submit late. We will not give any further extensions so make sure you only use them when you are absolutely sure you need them. See the Assignments and Grading Policy here for more information about grace days and late submissions: `https://cmudeeprl.github.io/Spring202010403website/logistics/`
- **Submitting your work:**
  - **Gradescope:** Please write your answers and copy your plots into the provided LaTeX template, and upload a PDF to the GradeScope assignment titled "Homework 1." Additionally, export the code from your Colab notebook ([File → Export .py]) and upload it the GradeScope assignment titled "Homework 1: Code." Each team should only upload one copy of each part. Make sure you mark your partner as a collaborator on Gradescope and that both names are listed in the writeup. Regrade requests can be made within one week of the assignment being graded.
  - **Autolab:** Autolab is not used for this assignment.

[1] `https://www.cmu.edu/policies/`

Problem 2: Bayesian Optimization (25 pts)

In this section, you will implement Bayesian Optimization using Gaussian Processes and compare the average regret over time for three acquisition functions: GP-greedy, GP-UCB, and GP-Thompson.

In this section **your solution will consist of a single plot**, with each curve in that plot being described in a subsection below. This allows you to get partial credit if you are not able to implement all 3 acquisition functions.

We provide you with the squared exponential kernel (also known as the RBF kernel) you will use for this assignment, defined as:

$$K(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x_i - x_j)^T(x_i - x_j)\right)$$

In the function templates we provide, we describe the parameters:

1. the parameter l is the scale of the RBF kernel.
2. *sigma_f* is the vertical variation parameter σ_f of the RBF kernel.
3. *sigma_y* is the standard deviation of the intrinsic noise in y.

Feel free to experiment with values for these, but **do not change the default GP arguments (arguments l, sigma_f, sigma_y) for the final plot**.

Use the following GP update equations for the posterior:

$$\begin{aligned}
K_y &= K + \sigma_y^2 I_N \\
\begin{bmatrix} y \\ f_* \end{bmatrix} &= \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_y & K_* \\ K_*^T & K_{**} \end{bmatrix}\right) \\
p(f_*|X_*, X, y) &= \mathcal{N}(f_*|u_*, \Sigma_*) \\
\mu_* &= K_*^T K_y^{-1} y \\
\Sigma_* &= K_{**} - K_*^T K_y^{-1} K_*
\end{aligned}$$

Review slide 193 of `https://cmudeeprl.github.io/Spring202010403website/assets/lectures/s20_lecture3.pdf` for more details. These equations differ from the ones in the slides because here we omit the offset in μ_* since we assume zero mean.

For all questions in this section, you only need to take into account the domain defined by the python constants $[MIN\_X, MAX\_X]$. The terms "posterior" and "posterior predictive" can be considered synonyms for the purposes of this question.

1. [**5pts**] The first acquisition function you will implement is the simplest: GP-greedy. This function will take a training dataset, find the maximum mean value of the posterior distribution, and output the x-value corresponding to that mean. Since the GPs we are dealing with here are 1-dimensional, we recommend using grid search as an effective way to estimate the maximizer.

   More specifically, to do grid search on a function f, construct a grid of uniformly spaced points using numpy's linspace function, evaluate the function f at all such points, and return the grid point with maximum value. We recommend you use at least 100 such grid points when finding the maximum.

2. **[10pts]** Next, implement the GP-UCB acquisition function described in class. This function is similar to the greedy one but you will need to take into account the standard deviation of each point you evaluate in the grid search. Use the following optimization process to choose the next x_t to evaluate:

$$x_t = \arg\max_x \mu(x) + 1.96\sigma(x)$$

   where $\mu(x)$ represents the GP's mean at x and $\sigma(x)$ represents the standard deviation at x. Notice here that we provide a constant 1.96 instead of the $\beta_t^{1/2}$ described in class, which is a function of time. While having a time-varying constant is important for some applications, we choose a constant here to make convergence faster.

3. **[10pts]** Finally, implement Thompson sampling using the GP. This involves sampling a function from the GP and then taking the argmax of this sampled function.

Once you have completed the subsections, **plot a graph of the regrets of the acquisition functions you implemented using the provided function** *bayes_opt(n_trials=200, n_steps=50)*. This might take a while, so try to debug with fewer *n_trials*. Finally, **include a short description of the regret curve of each acquisition function and why you believe the regret behaves the way it does for that function (does it reach exactly zero regret? If it doesn't, why not? etc.).**

Problem 3: CMA-ES (25 pts)

For this problem, you will be working with the Cartpole (`Cartpole-v0`) environment from OpenAI gym. Please take a look at the implementation to learn more about the state and action spaces: `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`.

For Cartpole, we have continuous states and hence we cannot use policy learning methods for finite MDPs. In class, we learned about covariance matrix adaptation evolutionary strategy (CMA-ES), a black box, gradient-free optimization method. In this problem, you will implement CMA-ES to find a policy for the Cartpole environment. There is template code provided for CMA-ES in the Colab notebook. Start by filling in the missing pieces in the `CMAES` class. For the CMA-ES update equations, follow the hints mentioned in the Colab notebook comments.

1. [**15 pts**] Run CMA-ES on Cartpole for 200 iterations (or until it reaches a reward of 200), using the provided hyperparameters. Plot the best point reward and average point reward throughout training.

2. [**10 pts**] Run your CMA-ES algorithm with population size of 20, 50, and 100. For each of the population sizes, run CMA-ES for 200 iterations (or until it reaches a reward of 200) and plot the best point reward at each iteration. In another plot, plot the best point reward as a function of the number of weights evaluated (# of iterations $\times$ population size).

3. (Optional) We provided most of the hyperparameters to you for this problem. However, in practice, significant time is spent tuning hyperparameters. Play around with the dimensionality of your network as well as other parameters of CMA-ES to get a feel for how the algorithm behaves.