Spring 2020

Stanford
AA 203: Optimal and Learning-based Control
Problem set 5 , due on June 1

Problem 1: In this problem, you will investigate three different approaches to solving the LQR problem with unknown dynamics and reward. In particular, we will assume dynamics of the form

$$\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t + \boldsymbol{w}_t \tag{1}$$

with $\boldsymbol{x}_0 \sim \mathcal{N}(0, \Sigma_0)$ and $\boldsymbol{x}_0 \sim \mathcal{N}(0, \Sigma_w)$, $\boldsymbol{x}_t \in \mathbb{R}^n$, $\boldsymbol{u}_t \in \mathbb{R}^m$ for all t. The cost function is

$$c(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{u}^T R \boldsymbol{u} \tag{2}$$

with Q, R positive definite. We will investigate three approaches to solving this problem: a model-based method; a model-free, value-based method, and a model-free, policy-based method. We will solve the discounted problem for a time-invariant policy,

$$\min_{\pi} \ \mathbb{E}[\sum_{t=0}^{N} \gamma^t c(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t))] \tag{3}$$

All code is provided in the starter code repository.

For all problems, run the method for both the stochastic dynamics and the deterministic dynamics. This can be changed via a boolean variable in the starter code.

Part a) First, implement the optimal controller for the known cost and dynamics via solving the Riccati equation. Although we are considering a finite horizon problem, it is typical in reinforcement learning to use a stationary (non-time-varying) policy to reduce the complexity of the learning problem. Simulate this controller from many random initial states to get an estimate of the average cost. This will be the optimal baseline to which we will compare the performance of other controllers.

Part b) We will now implement a model-based approach to the presented problem. This approach will consist of three steps, iteratively performed:

- Fit a linear dynamics model using least squares regression.

- Fit a model for the cost function using least squares. Let

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \tag{4}$$

  To use linear regression for cost function estimation, rewrite the reward as

$$c(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{q}^T \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ \vdots \\ x_1 x_n \\ x_2^2 \\ \vdots \\ x_n^2 \end{bmatrix} + \boldsymbol{r}^T \begin{bmatrix} u_1^2 \\ u_1 u_2 \\ \vdots \\ u_1 u_m \\ u_2^2 \\ \vdots \\ u_m^2 \end{bmatrix} \tag{5}$$

  for some vectors $\boldsymbol{q}, \boldsymbol{r}$.

- Compute the optimal policy for the estimated dynamics and reward model via Riccati recursion.

Let L^* denote the optimal policy, computed in part (a) with exact model knowledge, and let L_i denote the policy computed at iteration i. Plot $\|L^* - L_t\|_2$ versus time. Also, plot the achieved reward in the episode versus episode number.

Hint: It is convenient to use the iterative formulation of least squares to update the model at every iteration, which can then be interleaved after every timestep with policy optimization. For a system for which we receive input-output pairs $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ at each timestep, which obey the relationship

$$\boldsymbol{y}_t = A\boldsymbol{x}_t + \boldsymbol{w}_t \tag{6}$$

for zero mean normal $\boldsymbol{w}_t$, the iterative update can be written

$$\hat{P}_t = \hat{P}_{t-1} - \frac{\hat{P}_{t-1}\boldsymbol{x}_t\boldsymbol{x}_t^T\hat{P}_{t-1}}{1 + \boldsymbol{x}_t^T\hat{P}_{t-1}\boldsymbol{x}_t} \tag{7}$$

$$\hat{A}_t = \hat{A}_{t-1} + \frac{(\hat{P}_{t-1}\boldsymbol{x}_t)(\boldsymbol{y}_t^T - \boldsymbol{x}_t^T\hat{A}_{t-1})}{1 + \boldsymbol{x}_t^T\hat{P}_{t-1}\boldsymbol{x}_t} \tag{8}$$

with $\hat{A} = 0$, $\hat{P}_0 = I$.

Part c) Implement the policy iteration scheme for LQR presented in [1]. Note that the iterative least square algorithm used in that work is the same as the above.

Plot $\|L^* - L_i\|_2$ versus time, where L_i denotes the policy after improvement after every episode. Also, plot the achieved reward in the episode versus episode number.

Hint: This approach can be quite sensitive to various factors. Make sure your Q function estimate is symmetric by setting

$$\hat{Q}_t = \frac{1}{2}(\hat{Q}_t + \hat{Q}_t^T). \tag{9}$$

It is necessary to add exploration noise to the action selection to ensure convergence. Set the action to

$$\boldsymbol{u}_t = -\hat{L}_t \boldsymbol{x}_t + \boldsymbol{\epsilon}_t \tag{10}$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, I)$.

Part d) Implement Monte Carlo policy gradient (REINFORCE) for a Gaussian policy of the form

$$\pi_W(\boldsymbol{u} \mid \boldsymbol{x}) = \mathcal{N}(W\boldsymbol{x}, \Sigma) \tag{11}$$

for a choice of Σ.

First, find

$$\nabla_W \log \pi_W(\boldsymbol{u} \mid \boldsymbol{x}). \tag{12}$$

Then, implement the Monte Carlo policy gradient algorithm.

Plot $\|L^* - L_i\|_2$ versus time, where L_i denotes the policy at the end of each episode. Also, plot the achieved reward in the episode versus episode number.

Hint: Naïve policy gradient is quite unstable on this problem, so we will provide some hyperparameters that should enable you to get a basic version working. First, initialize your policy as the zero matrix, and set $\Sigma = 0.1I$. *As a step size parameter* α, *we recommend values in the neighborhood of* 10^{-13}. *These hyperparameters should yield a functional if unimpressive policy gradient algorithm, if used on top of the standard REINFORCE policy gradient. Expect this approach to take at least 5000 episodes to yield visible performance improvement. This approach will perform one to two orders of magnitude worse than the other methods; just report the performance, do not try to optimize this method to compete with the others.*

There are numerous ways to improve the performance of the policy gradient method. One simple approach is to whiten the reward signal. This can be done by maintaining the mean of the received rewards, as well as the empirical standard deviation, and subtracting off this mean and dividing this difference by the standard deviation. This approach shifts and rescales the reward to have mean zero and unit covariance, and should result in substantially improved performance. With this modification, choose a step size in the neighborhood of $\alpha = 10^{-8}$. *This modification is not required.*

Part e) Each of the previous methods required access to some amount of problem-specific information. For each of the four methods above, describe the problem information that was used in the design of the algorithm. How does the amount of assumptions for each of the methods compare to the performance of the methods?

Problem 2: As seen in lecture and in Part (d) of Problem 1, a naïve Monte Carlo policy gradient method is quite unstable, due to its high variance. To address this, we will implement an actor-critic method, in which a value function is used as baseline (referred to as the critic, which predicts the cost associated with actions), with the policy referred to as the actor (which selects the actions). In particular, we will implement advantage-based actor-critic algorithm that was discussed in lecture 14. Combining the policy gradient Q-function with the value function baseline we obtain the following formulation:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\delta^\pi \nabla_\theta \log \pi(\boldsymbol{u}_\tau \mid \boldsymbol{x}_\tau)\right] \tag{13}$$

with: $\delta^\pi = \sum_{\tau=t}^{N} r_\tau - V^\pi(\boldsymbol{x}_t)$ and $\mathbb{E}[\delta^\pi \mid \boldsymbol{x}, \boldsymbol{u}] = A^\pi(\boldsymbol{x}, \boldsymbol{u})$ the advantage.

To implement this method in practice, we fit our value function (for the critic) to target values:

$$y_t = \sum_{\tau=t}^{N} r_\tau \tag{14}$$

and use gradient descent to regress onto these target value with the following objective:

$$\min_w \sum_{i,t} (V_w^\pi(\boldsymbol{x}_{it}) - y_{it})^2. \tag{15}$$

Implementation: We have provided starter code and added **TODO**s in the following functions that are required to be implemented:

- `forward`: Sets up the Neural Network
- `select-action`: Selects the action
- `finish-episode`: Training code, computes actor and critic loss

Results: We will implement the method on the `LunarLanderContinuous-v2` environment. To get good results, you should run the algorithm for approximately 5000 episodes. You should provide a plot of the total reward versus episode. Run the script as follows (render can be toggled to True or False (default) to view the performance of the agent):

```
python run_actor_critic.py --render True
```

Improvements: Many improvements over the method implemented above are possible and can improve performance. A non-exhaustive list is given below, feel free to experiment with these (you can also try them out on more complicated environments such as `BipedalWalker-v2`):

- Experience replay: Stores transitions (x_t, u_t, r_t, x_{t+1}) in a buffer. Old examples are deleted as we store new transitions. To update the parameters of our network, we sample a (mini-)batch from the buffer and perform the stochastic gradient update on this batch.
- Bootstrap with value function, instead of using the sum of the tail rewards to estimate advantage.
- Iterate between computing target values and updating the value function to regress to these target values. You can add parameters that control the number of target value updates and the number of gradient updates.
- Using different policy or value networks.

Submission Submit your code and the plots of the evaluation reward for both environments listed above.

Learning goals for this problem set:

Problem 1: Gain experience implementing several different reinforcement learning algorithms on a simple linear system, and gain intuition for their relative strengths and weaknesses.

Problem 2: Gain experience implementing a neural network based actor-critic algorithm, to give insight into modern reinforcement learning practice.

References

[1] Steven J Bradtke, B Erik Ydstie, and Andrew G Barto. Adaptive linear quadratic control using policy iteration. In *Proceedings of 1994 American Control Conference (ACC)*. IEEE, 1994.