Homework 6

CMU 10-703: Deep Reinforcement Learning (Fall 2019)
OUT: November 18, 2019
DUE: December 1, 2019 by 11:59pm[1]

Instructions: START HERE

- **Collaboration policy:** You may work in groups of up to ~~two~~ **three 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[2].

- **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/703website/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 6." Additionally, upload your code (as a zip file) to the GradeScope assignment titled "Homework 6: Code." Each team should only upload one copy of each part. Regrade requests can be made within one week of the assignment being graded.
  - **Autolab:** Autolab is not used for this assignment.

[1] You are welcome to use late days for this assignment, but you must submit your assignment before 12pm (noon) on Dec 4.

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

Problem 0: Collaborators

Please list your name and Andrew ID, as well as those of your collaborators. You may collaborate in groups of at most ~~two~~ **three people**.

Problem 1: Crowd-Sourcing Quiz Questions (16 pts)

This problem is aimed at helping you prepare for the Quiz 2. Propose four great quiz questions that cover material from the entire course. Please submit your four questions and answers to each using the following form (once per question):

`https://forms.gle/xKqhqaxNaXHYRS2c7`

You can view the proposed questions from all teams here:

`https://docs.google.com/spreadsheets/d/1lNGqHV7TQ7P-3RX42UMOmHWZ-QmKTQ5msSAOy2vQNeg`

This question will be graded based on completion, not content, but we nonetheless encourage you to think carefully about it, as it will help you prepare for the quiz. Try your best to make sure that your proposed questions do not repeat any questions that have already been proposed. We recommend doing this problem early, before most of the "good" questions have been proposed.

Problem 2: Types of Uncertainty (25 pts)

This problem studies the two types of uncertainty and how they can be captured. Understanding these two types of uncertainty is useful for designing exploration algorithms and model-based RL algorithms.

2.1 Combined Variance (10 pts) As a warmup, prove the following identity:

$$\mathrm{Var}(\mathrm{y}) = \mathrm{Var}\left(\mathbb{E}[\mathrm{y} \mid \mathrm{x}]\right) + \mathbb{E}\left[\mathrm{Var}(\mathrm{y} \mid \mathrm{x})\right], \tag{1}$$

where x and y are random variables. Note what this identity says: the uncertainty over one random variable can be factored into two constituent terms that capture uncertainty.

2.2 Combined Variance and Dynamics Models (15 pts) Now, consider learning a dynamics model of the environment (i.e., a "forward" model): $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Our model f_θ will have parameters θ, which we will treat as a random variable. After learning this dynamics model, we will care about the uncertainty over our predictions: $\mathrm{Var}[\mathrm{s}_{t+1} \mid \mathrm{s}_t, \mathrm{a}_t]$. Parts 2.2.2, 2.2.3, and 2.2.4 can all be answered independently.

2.2.1 (6 pts) Use the identity from above to write this uncertainty as a combination of aleatoric and epistemic uncertainty. Hint: you will use $\mathrm{Var}[\mathrm{s}_{t+1} \mid \mathrm{s}_t, \mathrm{a}_t]$ as the left hand side of Equation 1.

2.2.2 (3 pts) In practice, how would you estimate each of the terms?

2.2.3 (3 pts) How is this decomposition related to model-based RL (e.g., [2])?

2.2.4 (3 pts) How is this decomposition related to exploration bonuses (e.g., [4, 3])?

Problem 3: Bayesian Neural Networks (30 pts)

The aim of this problem is to understand how we can capture uncertainty using Bayesian Neural Networks, and how these types of networks can be trained. We will use $p(\mathbf{y} \mid \mathbf{x})$ to represent the true conditional probability distribution. Using θ to represent the parameters of our neural network, we will define $q(\mathbf{y} \mid \mathbf{x}, \theta)$ as the corresponding conditional probability distribution of this neural network. To make this neural network Bayesian we will learn a *distribution over parameters*, $q(\theta)$, rather than learning a fixed parameter θ.

3.1 Uncertainty in BNNs (5 pts) Write the mean and variance of the network's predictions y for a fixed input x, given the parameter distribution $q(\theta)$. This question shouldn't take more than a few lines. You are welcome to write your answer in terms of expectations, rather than integrals. Problem 2.1.1 might be helpful for writing the variance.

3.2 An Objective for BNNs (5 pts) For learning our BNN, we will use $q_\phi(\theta)$ as our distribution over the parameters of the network, where ϕ is the parameters of the distribution.[3] We can obtain the conditional distribution of y given x under our BNN by marginalizing over the parameters θ:

$$q_\phi(\mathrm{y} \mid \mathrm{x}) = \int q(\mathrm{y} \mid \mathrm{x}, \theta) q_\phi(\theta) d\theta = \mathbb{E}_{\theta \sim q_\phi(\theta)}[q(\mathrm{y} \mid \mathrm{x}, \theta)]. \tag{2}$$

We will attempt to learn ϕ such that the BNN distribution $q_\phi(\mathrm{y} \mid \mathrm{x})$ equals the true distribution, $p(\mathrm{y} \mid \mathrm{x})$. To do this, we will maximize the log likelihood of data sampled from the true distribution:

$$\max_\phi \mathbb{E}_{x,y \sim p} \left[\log q_\phi(\mathbf{y} \mid \mathbf{x})\right]. \tag{3}$$

Relate the objective ~~on the LHS~~ to a KL divergence.[4]

3.3 REINFORCE for BNNs (10 pts) Derive an algorithm for optimizing Equation 3 that uses a likelihood-ratio gradient (i.e., the REINFORCE trick [5]).

3.4 Variational Inference for BNNs (10 pts) Derive an algorithm for optimizing a lower bound on Equation 3. Hint: try substituting Equation 2 into Equation 3 and applying Jensen's inequality.

[3] For example, $q_\phi(\theta)$ might be a Gaussian distribution, so ϕ would represent the mean and variance of each neural net parameter θ_i.

[4] Many BNNs (e.g., [1]) include a prior on the weights, $p(\theta)$. For simplicity, we are ignoring this prior for this problem.

Problem 4: LQR and iLQR (29 pts)

In this problem you will implement Linear Quadratic Regulation (LQR) and iterative Linear Quadratic Regulator (iLQR). We have provided you a coding template; ou do not have to use the provided code. You should be able to run all parts of this problem on a laptop (no need for AWS or GPUs).

Environment: You will be controlling a simple 2-link planar arm. It is based on the OpenAI gym API with a couple of additions you will need to approximate the dynamics. The environment comes with rendering code so that you can visually see what the arm is doing. The environments themselves define a cost matrix Q and a cost matrix R. Use these when calculating your trajectories. The rewards returned by the environment are computed using the LQR cost function. The `step` function includes an additional argument `dt`. When calculating the finite differences your dt will be much smaller than the dt that the simulator normally steps at when calling `step`. So when executing a command you should just use `step` with an action. When you are trying to approximate the dynamics using finite differences you should use `step` and override the `dt` argument as well. You can also explicitly set the state of this simulator using the `state` attribute. You will need this when doing finite differences. Just set this attribute equal to the q and $\dot{q}$ values you want before calling `_step`.

4.1 LQR (12pts) Implement LQR. Test your LQR implementation on the `TwoLinkArm-v0` environment. Record the total reward and number of steps to reach the goal. Also plot q, $\dot{q}$, and your control inputs u.

4.2 iLQR [**12pts**] Implement iLQR. Test your iLQR implementation on the `TwoLinkArm-v0` environment. Plot the total cost (intermediate cost + final cost) respect to iterations and record the total reward. Also plot q, $\dot{q}$, your control inputs u.

4.3 Comparing LQR and iLQR [**5pts**] In 2-3 sentences, describe *how* LQR and iLQR performed similarly/differently in your experiments, and use your knowledge of LQR/iLQR to suggest *why* they performed similarly/differently.

Hyperparameters: Set number of control steps to $T = 100$. The intermediate cost function of intermediate control steps $1 \cdots T-1$ is $\|u\|^2$, where u is the control parameters. The final cost function for the final step is $10^4 \times \|x_T - x^*\|^2$, where x_T is the final state generated from your algorithm and x^* is the target state. Set the maximum number of optimization iterations to be 10^5. You can add any heuristic stopping criteria to early stop if the algorithm converges.

Suggestions: iLQR will take a lot longer to run than LQR, but you should still be able to run it on a regular laptop. If you face the numerical issue when you are doing the inverse operation, try to add λI to make the matrix full-rank. You are free to use any $\lambda > 0$ (e.g., $\lambda = 1$). We recommend using simple tasks for debugging.

References

[1] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[3] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[4] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[5] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.