CS 330 Autumn 2020/2021 Homework 4

Exploration in Meta-Reinforcement Learning

Due Monday November 9th, 11:59 PM PT

SUNet ID:
Name:
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Overview

In this assignment, we will be exploring meta-reinforcement learning algorithms. In particular, you will:

1. Experiment with black-box meta-RL methods [1,2] trained end-to-end.
2. Implement components of DREAM [3] to replace the end-to-end optimization objective.
3. Compare the performance of end-to-end versus decoupled optimization.

Submission: To submit your homework, submit one PDF report to Gradescope containing written answers/plots to the questions below and the link to the colab notebook. The PDF should also include your name and any students you talked to or collaborated with.

The Colab notebook for this assignment can be found here:
https://colab.research.google.com/drive/1VUsEXSj9cT_PDDH1JPqowQwyShFwbLsj?usp=sharing
You will need to save a copy of the notebook to your Google Drive in order to make edits and complete your submission. **When you submit your PDF responses on Gradescope, please include a clearly-visible link to your Colab notebook so we may examine your code. Any written responses or plots to the questions below must appear in your PDF submission.**

Code Overview: The code consists of a few key sections. You should make modifications to areas where required. Cells where we expect you to make adjustments will be labeled with ⋆.

- ⋆ **embed.py** – This cell contains code for mapping single steps or sequences of experiences to embeddings. **The majority of your modifications will happen here.**
- **dqn.py** – This cell contains several implementations of value-function approximators, largely based around DQN.

- ⋆ **configs.py** – This cell contains the hyperparameters for RL^2 and DREAM. For some problems, you will need to make small modifications to these settings in order to run different agents. **Please only make modifications to this cell as instructed below and leave all other parameters as given in the stencil when submitting your final plots**.
- **rl2.py** – This cell will run RL^2 based on the corresponding configuration in configs.py.
- **dream.py** – This cell will run DREAM based on the corresponding configuration in configs.py.

Included in the code are also various cells for installing requirements and setting up visualizations via Tensorboard. **For all problems below, it is perfectly fine to submit individual screenshots of Tensorboard plots, provided they are clear and include axis labels.**

Dependencies: Based on the mid-quarter feedback, we decided to release this assignment in PyTorch. You should run everything in the first two cells for setup. Please note that the installation of requirements may take some time and may produce some red warning text that you should ignore. If you want to run it locally, we expect code in Python 3.5+ and for you to install necessary dependencies based on the contents of `dream_template.zip` from the first cell of the Colab notebook.

Environment: GridWorld Navigation with Buses

We will be considering a grid world domain illustrated in Figure 1. The **state** consists of the agent's (x, y)-position, a one-hot indicator of the object at the agent's position (none, bus), and a one-hot goal instruction i that corresponds to one of four possible goal locations (shown in green). Note that these goal instructions are part of the state, and hence vary *within* a task. The **actions** are move up, move down, move left, move right, or ride bus, which, at a bus, teleports the agent to the other bus of the same color. Episodes consist of 20 timesteps and the agent receives a reward of -0.1 at each timestep until the agent reaches the correct goal position, at which point the agent receives a $+1$ reward. The task distribution $p(\mu)$ corresponds to different bus arrangements. Hence, **only the dynamics vary across tasks, and not the rewards**. We will use μ to denote the task identifier.

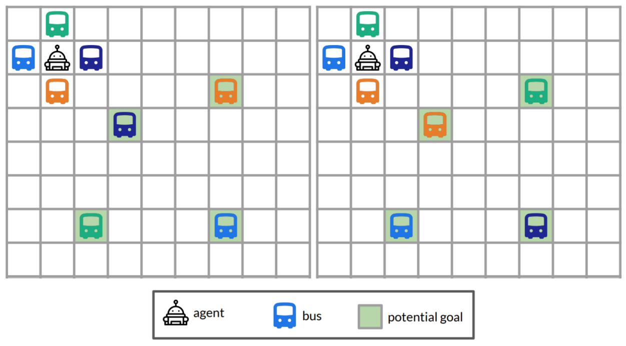

Figure 1: Gridworld environments corresponding to two example tasks.

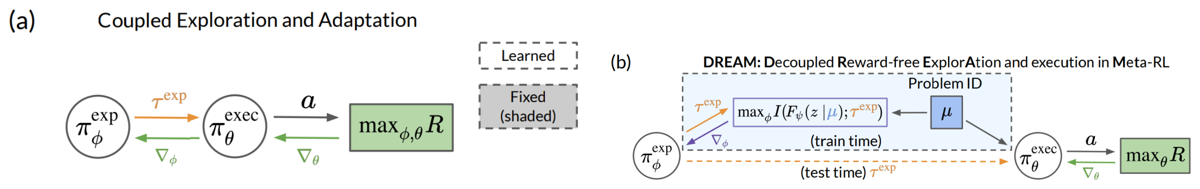

Figure 2: (a) End to end approach for meta-reinforcement learning. (b) DREAM: Decoupled exploration and execution.

Problem 1: Evaluating End-to-End Meta-RL

By default, if you run all cells up to and including **rl2.py**, you will run $10,000$ episodes of a random agent. As there are no parameter updates, you should find this takes only a few minutes to complete. To toggle between running a random agent and RL^2, adjust lines 91-92 of **configs.py** so that the agent type is set to `"learned"` instead of `"random"`.

1. Run a random agent and RL^2 agent for 10,000 episodes on the gridworld navigation task. Provide plots of the train and test rewards for each agent
2. Comment on your observations from the plots generated in the previous part. Based on what you know about RL^2, do these results align with your expectations? Why or why not?

Problem 2: Implementing Components of DREAM

The goal of the DREAM algorithm is to avoid issues with end-to-end learning of exploration and exploitation policies by decoupling the two objectives. This model has three main components, a learned task encoder, an execution policy, and an exploration policy, described as follows:

1. A **learned task encoder** $F_\psi(z|\mu)$, which extracts only the information necessary to execute instructions in the task ID μ, and an **execution policy** π_θ^{exec} are trained jointly using the objective:

$$\max_{\psi,\phi} \mathbb{E}_{\mu\sim p(\mu), z\sim F_\psi(z|\mu), i\sim p_\mu(i)} \left[V^{\pi_\theta^{\text{exec}}}(i, z; \mu) \right] - \lambda I(z; \mu)$$

where $V^{\pi_\theta^{\text{exec}}}(i, z; \mu)$ corresponds to the expected return of the policy on task μ for instruction i with task representation z. That is, the execution policy and the task embedding are optimized jointly, to train the encoder to represent task-relevant information. The information bottleneck $I(z; \mu)$ is can be estimated using a variational upper bound:

$$I(z; \mu) \leq \mathbb{E}_\mu[\mathbb{D}_{KL}[F_\psi(z|\mu)||p(z)]$$

where $p(z)$ is any prior distribution.

Code: The execution policy in DREAM π^{exec} produces a distribution over actions at each timestep given the current observation, current instruction, and an exploration trajectory τ^{exp}. In the `InstructionPolicyEmbedder`, complete the `forward` function which applies the `self._obs_embedder`, `self._instruction_embedder`, and `self._trajectory_embedder` to the state observation, instruction, and τ^{exp} respectively. Once you've collected each of those embeddings, concatenate them together so they can be fed into the hidden layer.

2. While task ids are available at train time, they are either not available during meta-testing or not useful (e.g. because they correspond to a new one-hot vector). For this reason we need to train an **exploration policy** that efficiently obtains task-informative data. The key approach is to train the policy during meta-training by maximizing the mutual information

$$\max_{\phi} I(F_{\psi}(z|\mu), \tau^{\text{exp}})$$

where τ^{exp} is a trajectory obtained by rolling out the exploration policy. Notice that the above does not optimize over ψ. That is, once we have obtained a suitable task representation is step 1, we would like to separately train a policy that produces a high very informative trajectories. This objective can be optimized using a variational lower bound of:

$$H(z) + \mathbb{E}_{\mu, z\sim F_{\psi}}[\log q_{\omega}(z)] + \mathbb{E}_{\mu, z\sim F_{\psi}, \tau^{\text{exp}}\sim\pi^{\text{exp}}}\Big[\sum_{t=1}^{T} q_{\omega}(z|\tau^{\text{exp}}_{:t}) - q_{\omega}(z|\tau^{\text{exp}}_{:t-1})\Big]$$

where $q_{\omega}(z|\tau^{\text{exp}})$ is a learned trajectory embedder. This objective is optimized in two steps:

(a) The trajectory embedder $q_{\omega}(z|\tau^{\text{exp}})$ is trained using supervised learning to predict the task representation from the exploration trajectory. If we parameterize $q_{\omega}(z|\tau^{\text{exp}})$ as a normal distribution with a fixed variance $\mathcal{N}(\mu_{\omega}(\tau^{\text{exp}}), \sigma^2 I)$, this loss reduces to an ℓ_2 regression of z on $\mu_{\omega}(\tau^{\text{exp}})$.

Code: Fill in the `_compute_losses` method of the `TrajectoryEmbedder` to compute the squared ℓ_2-distance between the trajectory embedding and task encoding.

(b) To optimize the objective with respect to the exploration policy parameters we use regular Q-learning using rewards:

$$r^{\text{exp}}_t(a_t, s_{t+1}, \tau^{\text{exp}}_{t-1}; \mu) = \mathbb{E}_{z\sim F(z|\mu)}[q_{\omega}(z|[s_{t+1}, a_t, \tau^{\text{exp}}_{:t-1}]) - q_{\omega}(z|\tau^{\text{exp}}_{:t-1})] \tag{1}$$

This quantity represents the "information gain" from the additional step. That is, the exploration policy maximizes the total information gain about the MDP.

Code: Fill in the `label_rewards` method of the `TrajectoryEmbedder` to provide the exploration rewards for π^{exp}.

Problem 3: Analysis of DREAM

1. Run **dream.py** and examine the plots of train and test reward. How do these results compare with RL^2 or random behavior? **Note:** while neither method will solve the task within $10,000$ episodes, you should be able to make some inferences about the performance of each algorithm in the alotted time. If this happens to be difficult to see on your first run, try one or two more random seeds by modifying line 35 of **rl2.py** and **dream.py**.

2. The efficiency of DREAM stems from the decoupled optimization of exploration and execution policies, with a carefully chosen objective for each. Adjust lines 51-52 of **configs.py** to run DREAM with a random exploration policy. Comment on the difference between the learning curves generated by these two instances of DREAM for $10,000$ episodes. Based on what you know about DREAM, do these results align with your expectations? Why or why not?

References

[1] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, Pieter Abbeel. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. https://arxiv.org/abs/1611.02779

[2] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, Matt Botvinick. Learning to Reinforcement Learn. The Annual Meeting of the Cognitive Science Society. (CogSci) 2017. https://arxiv.org/abs/1611.05763

[3] Evan Liu, Aditi Raghunathan, Percy Liang, Chelsea Finn. Explore then Execute: Adapting without Rewards via Factorized Meta-Reinforcement Learning. https://arxiv.org/abs/2008.02790v1