CS 277: Control and Reinforcement Learning (Winter 2021)
Assignment 1

Due date: Fridat, January 22, 2021 (Pacific Time)

Roy Fox
https://royf.org/crs/W21/CS277/

In the following questions, a formal proof is not needed (unless specified otherwise). Instead, briefly explain informally the reasoning behind your answer.

Part 1 Relations between horizon settings (25 points)

1. A Markov Decision Process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r \rangle$, where

   - $\mathcal{S}$ is the space of environment states,
   - $\mathcal{A}$ is the space of agent actions,
   - p: $\mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the environment dynamics, such that $p(s'|s,a)$ is the distribution of the state s' that follows state s when action a is taken (here $\Delta(\cdot)$ is the space of distributions over a set), and
   - r: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function.

   The MDP together with a stochastic agent policy π: $\mathcal{S} \to \Delta(\mathcal{A})$ induce a Markov process over states, with the distribution of the transition from a state s to the next state s' given by

   $$p_\pi(s'|s) = \sum_{a \in A} \pi(a|s) p(s'|s,a) \qquad \forall s, s' \in \mathcal{S}.$$

   This question is concerned with a *stationary distribution* $p(s)$ of the process, which is defined as a "fixed point" of the transition p_π. This means that, if the distribution $p(s_t)$ of the state in time t is a stationary distribution, then $p(s_{t+1}) = p(s_t)$.

   Assume that the initial distribution $p(s_0)$ is a stationary distribution. Show that the expected return of the policy π is the same, up to some multiplicative constant, for a T-step finite horizon, an infinite horizon, and a discounted horizon (these settings were defined in Lecture 1). (10 points)

2. What is the "effective finite horizon" of the discounted horizon with discount γ, i.e. the finite horizon T that would gives the same multiplicative constant in the previous question? (5 points)

3. In an episodic horizon, there exists an *absorbing state* $s_f \in \mathcal{S}$. A state is absorbing if any action $a \in \mathcal{A}$ taken in it remains in it $p(s_f|s_f,a) = 1$, and gets no reward $r(s_f,a) = 0$. Once the trajectory gets to an absorbing state, nothing interesting can ever happen again, and we can say that the process has terminated. This allows us to define the return in the episodic horizon as $R = \sum_t r(s_t,a_t)$, although there's generally no bound on how many terms the series has before reaching $s_T = s_f$. We will restrict our attention to processes and policies that get to the absorbing state in finite *expected* time, so that the expected return is finite.

   What is the stationary distribution with the policy thus restricted? (5 points)

4. Show that the discounted horizon is a special case of the episodic horizon. Namely, for any MDP $\mathcal{M}_1$ with a discounted horizon, construct another MDP $\mathcal{M}_2$ with an episodic horizon such that the two problems are quite obviously equivalent (i.e. finding good policies in one is equivalent to finding good policies in the other). What state do you need to add to $\mathcal{M}_1$? What are the rewards and transition probabilities involving that state in $\mathcal{M}_2$, such that the discount is emulated? (5 points)

Part 2 Imitation Learning algorithms — theory (25 points)

In this part we consider a rough abstraction of a lane-following task, in which a car needs to follow a curved lane on the road. The car starts in the center of the lane. In each time step, the lane either curves left ($x = -1$), right ($x = +1$), or keeps straight ($x = 0$), with equal probabilities ($1/3, 1/3, 1/3$). The agent can turn left or right slowly ($a = -1$ or $a = +1$) or quickly ($a = -2$ or $a = +2$), or keep straight ($a = 0$). The car's position in the lane is then changed by $x - a$; for example, turning left quickly when the lane curves left adds $-1 - (-2) = +1$ to the position. If the position becomes $\leqslant -2$ or $\geqslant +2$, the car crashes (terminating the episode). Otherwise, the reward is 1 for being in the center of the lane (position 0), and 0 for other positions.

1. Model this task as a MDP. What are the states? What is the dynamics? What is the reward function? (5 points)
2. An expert demonstrates the task with perfect performance. We then use Behavior Cloning to learn a lane-following policy. Assume that the learned policy agrees with the expert with probability 0.9 on states in the training set, choosing each of the 4 other actions with probability 0.025. On states missing in the training set, the policy is uniform. What is the expected (undiscounted) return of this policy? (10 points)
3. We now use DAgger to learn a policy, with expert annotation of learner rollouts. Using the same assumption on policy quality, what is the expected (undiscounted) return of the policy learned by DAgger? (10 points)

Part 3 Imitation Learning algorithms — practice (50 points)

In this part, you will install a Deep RL framework, RLlib), and use it to evaluate Deep IL algorithms. The OS we use in this assignment is Linux / MacOS. If you're using Windows or another OS, please use a local or remote virtual machine.

Installation

Make sure you have a recent version of Python installed, such as the latest Python 3.8; but **not Python 3.9**, as RLlib doesn't seem to support it at this time. It is recommended that you create a new conda environment for this assignment.

RLlib can use either of two Deep Learning library, TensorFlow (TF) and PyTorch. Most of the algorithms already implemented in RLlib support both these libraries. In this assignment we will use TF for CPU:

```
pip install tensorflow
```

It is also possible to install TF for GPU, and this would allow much faster execution, but is not needed in this assignment.

We will also need a simulator of the environment, and the one we will use is Box2D:

```
pip install box2d-py
```

This may require you to first install swig.

RLlib is implemented on top of the Ray distributed execution library:

```
pip install "ray[rllib]"
```

1. Use RLlib to train an agent to play Lunar Lander. The algorithm we will use is called PPO (we'll learn about it in a later lecture). The environment is implemented by OpenAI Gym, and is called LunarLanderContinuous-v2 (in this version, the action space is continuous). Run the algorithm for

1000 "iterations" (what iteration means in RLlib is algorithm-dependent) and create a checkpoint (a save of the agent's trained parameters) every 10 iterations.

```
rllib train
  --run PPO
  --env LunarLanderContinuous-v2
  --checkpoint-freq 10
  --stop '{"training_iteration": 1000}'
```

2. Roll out the agent that you trained to see how well it performed. The agent checkpoints are by default in a path named `~/ray_results/default/PPO_LunarLanderContinuous-v2_<experiment_id>`, where `experiment_id` is an automatically assigned identifier of the experiment you ran in the previous step, ending with the date and time. Roll out for 5000 steps, which should be about 20 episodes.

```
rllib rollout ~/ray_results/default/PPO_LunarLanderContinuous-v2_<experiment_id>/→
    checkpoint_1000/checkpoint-1000
  --run PPO
  --env LunarLanderContinuous-v2
  --steps 5000
```

You can also roll out an earlier checkpoint to see the difference in performance. RLlib will output the return of each episode, which should be well above 200 for successful episodes. If you are not seeing consistently good episodes (most should get above 200 return), rerun the training in the previous question.

3. Use the trained agent to generate demonstrations for an imitation learning agent. Roll out for 250000 steps. The demonstrations will be saved in a file named `rollouts.pkl`. The `-no-render` flag prevents rendering the episode to screen, thus saving much time.

```
rllib rollout ~/ray_results/default/PPO_LunarLanderContinuous-v2_<experiment_id>/→
    checkpoint_1000/checkpoint-1000
  --run PPO
  --env LunarLanderContinuous-v2
  --steps 250000
  --out rollouts.pkl
  --no-render
```

4. Convert the demonstrations into the format used by RLlib for training from offline data. Download the utility https://royf.org/crs/W21/CS277/A1/prepare_dataset.py and run it. The results will be saved in a directory named `LunarLanderContinuous-v2`.

5. Train a Behavior Cloning agent. The agent will be trained on input from the data generated in the previous step, and evaluated in new simulations of the environment.

```
rllib train
  --run BC
  --env LunarLanderContinuous-v2
  --checkpoint-freq 1000
  --stop '{"training_iteration": 10000}'
  --config='{"input": "LunarLanderContinuous-v2", →
    "input_evaluation": ["simulation"]}'
```

6. Repeat step 2 for the trained BC agent, and report the mean and standard deviation of the returns of 5 episodes.