CSCI 4160/6963, ECSE 4965/6965
Reinforcement Learning
Homework 5

Overview

This assignment is an introductory assignment to Markov Decision Processes (MDPs), value functions and policies. Consider the following grid:

3	6	9	12	15	18	21	24	27	30	33	36
4	7	10	13	16	19	22	25	28	31	34	37
5	8	11	14	17	20	23	26	29	32	35	38
1											2

You can think of this is a cliff environment. You start in state 1, and your goal is to get to state 2. If you fall in the shaded cells (i.e., the cliff), you get a reward of -100 and you go back to state 1 (e.g., if you go down from state 8, you transition to state 1 with a reward of -100). Otherwise, you get a reward of -1. If you hit a wall from any state, you remain in that state and also get a reward of -1. Once you get to state 2, you get a reward of 0 and cannot transition to any other state (i.e., it's a terminal state).

Your first task is to encode this environment as an MDP. Your MDP should have 38 states as shown above. The MDP has four actions (fix this order for consistency): `[left, up, down, right]`. For each action/state pair, you need to store the probability of going to any other state (this environment is deterministic but you may still want to set up your code to handle probabilistic environments, for future assignments). Furthermore, your MDP needs to store the rewards for each transition.

Once you have the MDP, you will need to implement the optimal policy, π, for each state. You don't need to use any learning algorithms – just hardcode the optimal policy. To make things

interesting, we will also explore two ϵ-greedy variations of the optimal policy: $\pi_{0.1}$ and $\pi_{0.2}$, which take the optimal policy's action with probability 0.9 and 0.8, respectively (and take a random action otherwise). Once you have the policies, set the discount factor to 0.9 (i.e., $\gamma = 0.9$) and i) compute the state values for each state under each policy, and ii) simulate 1000 trajectories in the environment under each policy and record your average discounted return.

Logistics

All coding assignments in this course will be in Python. If you need help with Python, please talk to me ahead of time, so we can discuss the best way to get familiar with it.

For this assignment, you will be using your own computer, which should be sufficient. The code should not require more than 5 minutes to run on any standard laptop.

You are provided with skeleton starting code. Please use the Python libraries in the `requirements.txt` file that's provided on LMS.

Grading

For a full score, you need to implement the provided functions, `get_cliff_mdp`, `get_optimal_policy`, `get_pi_e_greedy`, `get_P_R`, `get_v`, `gen_episode`. Your MDP data structure should allow for probabilistic transitions (since the environment is deterministic, only one transition will get a probability of 1). Your policy should also be probabilistic – the optimal policy will always select the optimal action with probability 1. The other policies select the optimal policy with probability $1 - \epsilon$ and select a random other action with probability ϵ (where $\epsilon = 0.1$ for $\pi_{0.1}$ and $\epsilon = 0.2$ for $\pi_{0.2}$).

The easiest way to calculate the state values is to convert your MDP to an MRP (given your policy) and then compute the infinite-horizon value (using the `get_P_R` and `get_v` functions), as we did in class, for a discount factor of $\gamma = 0.9$.

Make sure your code prints out some indication of progress as well as all required information – we should not have to insert any print statements in order to find out whether your code works!

Finally, please provide brief answers (2-3 sentences each) to each of the questions below:

1) The average simulation (discounted) return is going to be similar to the value of one of your MDP's states. Which state is that? What statistical property (i.e., theorem) that we saw in class ensures that the average reward will converge to that state's value?

2) Why does policy $\pi_{0.2}$ result in lower returns?

3) [**Graduate students only**] Suppose you use the matrix form of the Bellman equation in order to calculate the state values. What would happen to the calculation if we set the discount factor to 1, i.e., $\gamma = 1$?

Hints and Tips

- It is up to you to decide what data structure you want to use for the MDP. The easiest is to probably use a dictionary, where each key is an MDP state; the corresponding value is also a dictionary where each key is an action – the corresponding value is a tuple: 1) a list of probabilities (effectively a row of your transition matrix) and 2) a list of corresponding rewards. The other option is to use a numpy array, which tends to be faster than dictionaries.
- Similarly, it is up to you what data structure you want to use for the policy. The easiest is to probably use a dictionary (or a numpy array), where each key is an MDP state; the corresponding value is a list of the probability of each action being taken. Again, for the deterministic policy, this list would have a 1 for the optimal action and 0 for other actions.

Submission

Please use `LMS` to submit a zip file containing **only the following files**: 1) your **.py** code, along with instructions on how to run it and 2) a **.pdf** file containing your answers to the above questions. The deadline is **11:59pm, Thursday, Oct. 16**.