CS395T: Robot Learning
Assignment 2: MaxEnt IRL

In this assignment, you will be implementing maximum entropy inverse reinforcement learning, as described by Ziebart et al. (2008) in"Maximum Entropy Inverse Reinforcement Learning".

Your task is to write three functions, as specified in the provided file (the file is in python, but you can use a language of your choice, as long as you keep the same interface). The three functions are:

1) calcMaxEntPolicy(trans_mat, horizon, r_weights, state_features):

Description: For a given reward function and horizon, calculate the MaxEnt policy that gives equal weight to equal reward trajectories

Parameters:

- trans_mat: an S x A x S' array of transition probabilities from state s to s' if action a is taken
- horizon: the finite time horizon (int) of the problem for calculating state frequencies
- r_weights: a size F array of the weights of the current reward function to evaluate
- state_features: an S x F array that lists F feature values for each state in S

Return: an S x A policy in which each entry is the probability of taking action a in state s

2) calcExpectedStateFreq(trans_mat, horizon, start_dist, policy):

Description: Given a MaxEnt policy, begin with the start state distribution and propagate forward to find the expected state frequencies over the horizon

Parameters:

- trans_mat: an S x A x S' array of transition probabilities from state s to s' if action a is taken
- horizon: the finite time horizon (int) of the problem for calculating state frequencies
- start_dist: a size S array of starting start probabilities - must sum to 1
- policy: an S x A array array of probabilities of taking action a when in state s

Return: a size S array of expected state visitation frequencies

3) maxEntIRL(trans_mat, state_features, demos, seed_weights, n_epochs, horizon, learning_rate):

Description: Compute a MaxEnt reward function from demonstration trajectories. This will invoke the above two functions.

Parameters:

- trans_mat: an S x A x S' array that describes transition probabilities from state s to s' if action a is taken
- state_features: an S x F array that lists F feature values for each state in S

- demos: a list of lists containing D demos of varying lengths, where each demo is series of states (ints)
- seed_weights: a size F array of starting reward weights
- n_epochs: how many times (int) to perform gradient descent steps
- horizon: the finite time horizon (int) of the problem for calculating state frequencies
- learning_rate: a multiplicative factor (float) that determines gradient step size

Return: a size F array of reward weights

The transition matrix for a 5x5 gridworld domain is provided (again, in Python, but feel free to re-implement carefully in another language), with states laid out as follows:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

0 is the start state and 24 is a special goal state that always transitions to a special terminal state, 25. There are 4 actions: up, down, left, and right, which move the agent accordingly (and deterministically), unless the agent would move past the edge of the world, in which it stays in place. All actions at state 24 lead to state 25 deterministically. All actions at state 25 lead to each possible next state with probability zero, making it terminal. There is a hazard at state 8 that should be avoided.

A very simple feature space can be used for this problem. There are 25 features, one for each state. Thus, the features corresponding to the i^{th} state are all zeros, except for a one in the i^{th} position. The 25th state (terminal) has an all zero feature vector, forcing a zero reward at all time. This feature space is simple, but very expressive—it can represent any reward function over states of the form R(s).

There are four provided demonstrations that take 4 different paths to get from state 0 to state 24 as quickly as possible, while also avoiding the hazard at state 8. By chance, the demonstrations happen to avoid state 16 as well, despite there being no hazard—you will analyze the effects of this later:

[0, 1, 2, 3, 4, 9, 14, 19, 24, 25]

[0, 5, 10, 15, 20, 21, 22, 23, 24, 25]

[0, 5, 6, 11, 12, 17, 18, 23, 24, 25]

[0, 1, 6, 7, 12, 13, 18, 19, 24, 25]

Notes:

- $\tilde{f}$ should be averaged over all demonstrations trajectories (i.e. it is an expected value that needs to be divided by the number of trajectories). The paper isn't totally clear about this.
- For all questions, seed the reward weights with all zeros unless instructed otherwise.

Answer the following questions:

1. Use a principled or well-justified method to choose a static step size / learning rate for gradient descent (i.e. a scalar value to multiply the gradient by, in order to decide on the magnitude of the weight update at each epoch). Describe how you chose an appropriate learning rate.

2. Run MaxEntIRL for 100 epochs with a horizon of 10 and describe the shape of the reward function. What does it appear to be trying to do, qualitatively? How does it behave at state 8 and 16? Does it appear to underfit or overfit the demos? If so, how might this issue be addressed, while still using the same underlying algorithm? Include a graph of the reward function in your writeup.

3. Call calcMaxEntPolicy() with a specific reward function: +10 at state 24 and -1 everywhere else with a horizon of 10. Do NOT perform gradient descent or call any other parts of the MaxEntIRL algorithm—just call the calcMaxEntPolicy() function with the given reward functions. Compare and contrast the returned policy with the optimal policy under the same reward function. In what ways is it different and why? In particular, examine the most likely action at State 19 under the MaxEnt policy and explain the rationale for it.

4. Increase the horizon to 100 for the same reward function as above and call calcMaxEntPolicy() again. What happens to the MaxEnt policy and why? If it is difficult to interpret all the probabilities of actions at each state, try just looking at the most likely action at each state.

5. Show the gradient calculation to get to equation 6 from the likelihood function $L(\theta)$ directly above it. If you are having difficulty, you might want to refresh your memory on how to take derivatives of logs, exponentials, and how to use the chain and product rule.

6. Compare equations 5 and 7. Intuitively, why does the MaxEnt policy of equation 5 equally weight paths in the example in Figure 2, while the action based distribution model of equation 7 weights them differently? Why is the weighting of equation 7 problematic?

7. Explain in your own words what $Z_{a_{ij}}$ and Z_{s_i} represent over a particular horizon.

8. Why is Z_s set to one for the terminal state? What is interpretation of this in terms of both 1) total probability of trajectories that end at the terminal state and 2) the reward gained for being in a terminal state? (hint: think of the equation for $Z_{a_{ij}}$ as a state-action value function)

As always, be sure to include all relevant code as an appendix to your writeup.