ESE 650, Spring 2019 Assignment 5

Due on Friday May 3rd, 2019,11:59PM

This assignment is split into two parts.

1 Value Iteration (10 Points)

Fill in code in tester.py in function *value_iteration*. Environment comes from lake_env.

2 Policy Evaluation (5 Points)

Fill in code in tester.py in function *evaluate_policy*.

3 Policy Iteration (5 Points)

Using the policy evaluation function above write code for *policy_iteration.py*

4 Policy Gradient (30 Points)

In *policy_gradient_test* implement a policy parametrized by a 2 layer neural network implemented in pytorch. This model must be loaded once in tester.py in the constructor (init)

The network is trained on your local computer and when uploaded to the server must call a saved model. **It is important to note that NO training must be done on the server. There is a timeout condition and your code will exit if you attempt to train on the server.**

5 Model Based Learning (50 points)

For this part of the assignment, compile your results into a pdf and submit on canvas. There is no autograder for this part. This part of the assignment is designed such that the solution cannot be found on Google or in a textbook. It is designed to help you think through the problem at hand from first principles and arrive at a reasonable solution. It is hard but we hope it will extremely rewarding. Have fun.

Consider a pendulum with a some unknown mass and unknown dynamics as your system (see Fig 1). Your system takes in as input a torque (clockwise or counter clockwise). Your goal is to compute a sequence of actions that ensure the pendulum can stay in an upright position. This is a classic model based problem and can be broken down to 2 pieces :

1. Learn the model of the system.
2. Using the learned model, use a planner to compute the sequence of states required to achieve the desired position.

Part a) Implement a system that learns the model of the pendulum.

The state of the pendulum at any time t is given as $s(t) = [cos(\theta), sin(\theta), \theta, \dot{\theta}]$. Your output action space is a continuous action space corresponding to the torque τ applied at the hinge $\tau \in (-2, 2)$. To get started with this task :

1. Install the openai-gym package and run through the tutorial on how to get set up with a pendulum environment.

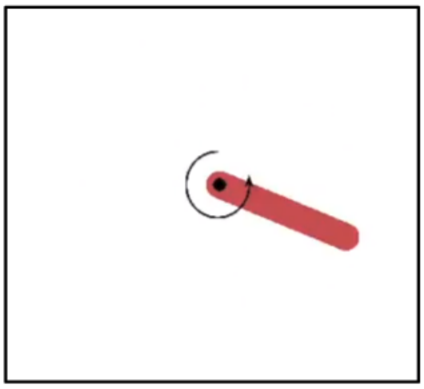

Figure 1: Pendulum

2. Once you have the simulation set up, learn the dynamics model of the system with any parametric function. For example, if attempting to use a neural network to learn a model, the input to the model is the current state s_t and the current action a_t (the current action is a real valued number representing torque) and the model predicts the next state $\hat{s}_{t+1}$. The neural network attempts to minimize the loss between the predicted next state $\hat{s}_{t+1}$ and the true next state s_{t+1} which can be obtained from the simulator. (Randomly sample actions and states as inputs. Learn a model and adapt it such that the input is $[s_t, a_t]$ and output is s_{t+1})

Plot a training curve for your model. Report training error. Also report test error.

Part b Using the learned model from Part a) and implement a graph based planner that can compute the most optimal solution. An example of a graph based planner can be A^* (reads as A-star). For example, A-star takes in as input the current state, evaluates reachable states from the current state by computing a heuristic and then picks the best possible next state according to this heuristic. Thus, to do this subpart :

1. You first need to discretize your state space. Uniformly discretizing your state space is a good strategy ($-1 \leq \theta \leq 1$ and $-8 \leq \dot{\theta} \leq 8$).

2. Sweep through all your states and for every state find the best possible state by evaluating a heuristic. (a good choice for a heuristic $h = -(\theta^2 + \dot{\theta}^2 + 0.001 * {a_t}^2)$)

If you have gotten so far, you are almost there. To recap :

You have a pendulum with some initial state s_0. You wish to output a sequence of actions $a_1, a_2, \ldots a_n$ such that you reach a desired s_{n+1}.

In the first step, you learned what the next state is going to be if fed in information about the current state and an action. Thus, you know all reachable states from your current state. Say you start with s_0, actions possible in s_0 are a_1 and a_2. a_1 takes you to state s_1 and action a_2 takes you to state s_2. Now using your planner you can evaluate a cost for s_1 and s_2 and evaluate which state has higher value (say it is s_2). Thus, the best action to take in s_0 is a_2 and end up in s_2. Now repeat this process till we reach desired state s_n. Thus, you successfully learned how to control an unknown object.

Report a graph depicting number of steps required by the planner vs discretization resolution.

Submit on Canvas

We will use ENIAC to submit assignments. After you have uploaded your submission to the ENIAC according to instructions in the above link, you may submit your estimate_rot.py **AND ALL FILES IT DEPENDS ON** for grading by running the following command on the ENIAC. For example, if estimate_rot depends on bias.py and motion.py then turn your assignment in using

turnin -c ese650 -p project3 tester.py policy$_m$*odel.pt*