Assignment 1

CPS 590.1: Decision Making for Robots and Autonomous Systems — Fall 2015

Instructions

This assignment consists of 5 questions. Each of them asks you to implement one or more algorithms, and then answer some questions and/or produce some graphs. Your report should contain both *evidence that your implementation is correct* **and** *answers to the questions and/or the graphs*.

You will be graded on the report itself. Please make it readable and easy to understand. Graphs are good, as are tables with a small set of relevant numbers and the best performers for each category in bold. Large tables of numbers (i.e., *data dumps*) are bad. Nice visualizations of value functions and policies are good. Etc.

Submission: Email your PDF and a .zip file containing all your code to `gdk@cs.duke.edu`.
Due date: October 15th 2015

Question 1: Bandits (20 points)

You are to solve a 3-arm bandit problem, where each arm returns rewards as follows:

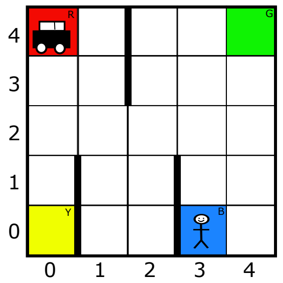
1. $r_1 \sim \mathcal{N}(\mu = 5, \sigma = 1)$.
2. $r_2 \sim \mathcal{N}(\mu = 4.5, \sigma = 2)$.
3. $r_3 \sim \mathcal{N}(\mu = 3, \sigma = 3)$.

a) Implement, and compare the performance of, the ϵ-greedy, optimistically initialized (but still ϵ-greedy), and UCB1 algorithms. This will require you to explore the relevant open parameters.

b) Describe a case (i.e., give an explicit example with actual numbers) of a 2-arm bandit where a greedy ($\epsilon = 0$), optimistically-initialized algorithm fails to find the optimal solution.

Question 2: Dynamic Programming (20 points)

Your task is to implement planning using dynamic programming in the Taxi Domain, depicted below.



A taxi must navigate a 5×5 grid, which contains a few walls, four depots (labeled red, green, blue, and yellow), and a passenger. The taxi may move one square in each direction, or attempt to pick up or drop off a passenger, for a total of six actions. Each movement action succeeds with probability 0.85, and moves in a random other direction with probability 0.15; movements that would cross the grid boundaries or a wall result in no change in state.

Each action execution gets a reward of -1, except delivering the passenger to their goal, which ends the episode and receives a reward of 20, or attempting to pickup the passenger when not on the same square as them, or drop them off when they are not in the taxi, both of which receive a reward of -10. Use $\gamma = 1$.

A state is the Taxi domain is described by 5 state variables: the x and y location of the taxi and the passenger, and whether or not the passenger is in the taxi. This results in a total of 650 states ($25 \times 25 = 625$ states for when the passenger is not in the taxi, plus another 25 for when the passenger is in the taxi and they are constrained to have the same location).

An *instance* of the Taxi domain is obtained by placing the passenger at a depot, selecting a (different) goal depot, and placing the taxi at a start depot. Planning (and learning) should be done for each instance separately; your results in all following questions should be averaged over all $4 \times 3 \times 4 = 48$ possible instances.

a) Describe *prioritized sweeping*, as described in Section 9.4 of Sutton and Barto,[1] and explain how you would implement it for stochastic domains.

b) Implement vanilla dynamic programming and prioritized sweeping and compare their performance in Taxi.

Note: Please keep your domain implementation in a separate file. You ***may*** *compare your* ***domain code*** *(and* ***only*** *your domain code) to that of others in the class to see if it is correct.*

Question 3: Reinforcement Learning (20 points)

In the previous question you could assume knowledge of the domain transition and reward functions. This question asks you to implement learning (i.e., assume that you can only get data by actually executing policies in the (simulated) environment).

a) Implement Sarsa(λ) for the domain in Question 2.

b) Determine which λ is best for an ϵ-greedy policy, with fixed α and $\epsilon = 0.1$.

c) Show learning curves for Sarsa(λ) for $\lambda = \{0, 0.5, 0.9, 1.0\}$ for $\epsilon = 0.1$ and α optimized for *each* setting of λ.

Question 4: RL with Function Approximation (20 points)

Implement Sarsa(λ) for the Mountain Car problem as described in Sutton and Barto.[2] Use linear function approximation with Fourier basis functions.[3]

a) Show learning curves for order 3, 5, and 7 Fourier bases, for a fixed setting of α and ϵ, and $\gamma = 1$, $\lambda = 0.9$.

b) Visualize the cost-to-go function (the negative of the value function) and learned policies after $1,000$ episodes, for the above orders. *(Hint: the cost-to-go should look like the one in Sutton and Barto, but smoother.)*

c) The Mountain Car contains a negative step reward and a zero goal reward. What would happen if γ was less than 1 and the solution was many steps long? What would happen if we had a zero step cost and a positive goal reward, for the case where $\gamma = 1$, and the case where $\gamma < 1$?

Question 5: Least-Squares Methods (20 points)

This question focuses on least-squares methods. Note that, although these methods were presented using linear function approximation, they also work for tabular value functions (by creating an *indicator function* for each state—a function that is 1 only in that state).

a) Implement least-squares TD for the domain in Question 2. Assume a random policy, and that the model is given, so you can perform a full backup at each step.

b) Using this algorithm, perform least-squares policy iteration. Start with a random policy in the first iteration, and assume an ϵ-greedy ($\epsilon = 0.1$) policy thereafter. Again, assume that you have the model and do full backups.

[1] `https://webdocs.cs.ualberta.ca/~sutton/book/ebook/node98.html`
[2] As described in `http://webdocs.cs.ualberta.ca/~sutton/book/8/node9.html`
[3] `http://irl.cs.duke.edu/fb.php`