Homework 3
Graphical Models and MDPs

15-381/781: Artificial Intelligence (Fall 2016)
OUT: Oct 5, 2016
DUE: Oct 14, 2016 at 11:59pm

Instructions

Homework Policy

Homework is due on autolab by the posted deadline. Assignments submitted past the deadline will incur the use of late days.

hYou have 6 late days, but cannot use more than 2 late days per homework. No credit will be given for homework submitted more than 2 days after the due date. After your 6 late days have been used you will receive 20% off for each additional day late.

You can discuss the exercises with your classmates, but you should write up your own solutions. If you find a solution in any source other than the material provided on the course website or the textbook, you must mention the source. You can work on the programming questions in pairs, but theoretical questions are always submitted individually. Make sure that you include a README file with your andrew id and your collaborator's andrew id.

Submission

Please create a tar archive of your answers and submit to Homework 3 on autolab. You should have three files in your archive: a completed `problems.py` for the programming portion, a PDF for your answers to the written component, and a README file with your Andrew ID and your collaborators' Andrew IDs. Your completed functions will be autograded by running through several test cases and their return values will be compared to the reference implementation. There is a `sample.txt` that contains sample inputs and outputs for reference.

1 Written [40 points (Grad 60 points)]

1.1 Choosing the Value of $R_{\max}$ [15 points]

This question refers to the R-MAX algorithm (to be covered in lecture the week of 10/10).

The R-MAX algorithm makes the assumption that we know the maximum possible reward. In the real world, this might not always be the case. In this problem we explore some possible modifications to the algorithm when we don't know the maximum reward.

1.1.1 Setting an upper bound [3 points]

Suppose we have a known upper bound for the reward and we set $R_{\max}$ to be this upper bound. When this bound is very loose, i.e., the bound is much greater than the true maximum possible reward, how will the

algorithm behave?

1.1.2 Working up to the true bound [7 points]

Instead, suppose we initialize $R_{\max} = 0$ (assume this is the minimum possible reward), and every time a state-action pair becomes known with a reward $\rho > R_{\max}$, we set $R_{\max} = \rho$ (and modify our unknown states accordingly). While intuitively this may seem like it should work, in some cases it does not. Give a simple MDP where this fails, i.e., where we will never find the optimal policy. Show that we never find the optimal policy on your example under balanced wandering.

1.1.3 Optimistically increasing $R_{\max}$ [5 points]

Now suppose we use the same algorithm as in (1.1.2), but instead we set $R_{\max} = \rho + \delta$, for some δ, which may depend on the discount factor, γ. Either give a brief description of why this won't fail as it did in (1.2.2), or show that for any value of δ, you can construct an MDP where this fails.

1.2 Q-Learning [15 points]

A robot moves deterministically in a world of 12 states laid out as shown in Table 1. The robot can take four actions at each state, namely, N, E, S, and W. An action against a wall leaves the robot in the same state (note that the walls are not shown in the figure, but you may assume that the outside borders are walls). Otherwise, the outcome of an action deterministically moves the robot in the direction corresponding to the action, e.g., if the robot takes action N and does not hit a wall, it will always end up in the state above it. The robot converged to the Q-table shown in Table 2, where we show only the maximum values for each state, as the other values (represented with a dash) do not matter in terms of the final policy.

S9	S10	S11	S12
S5	S6	S7	S8
S1	S2	S3	S4

Table 1: The World

	N	E	S	W
S1	59	59	-	-
S2	-	66	-	-
S3	-	73	-	-
S4	81	-	-	-
S5	66	-	-	-
S6	-	59	-	59
S7	-	-	66	-
S8	90	-	-	-
S9	-	73	-	-
S10	-	81	-	-
S11	-	90	-	-
S12	100	100	-	-

Table 2: Q-table

1.2.1 Moving in the world [2 points]

Using the learned Q-table, write the sequence of the first six actions and states reached if the robot starts in state S7. If there is more than one best action available, choose one randomly. You may assume for this question that the robot does not encounter any interior walls.

1.2.2 Finding rewards [7 points]

Suppose you are told the discount factor for Q-learning is $\gamma = 0.9$. Suppose you are also told there is a single state, s' such that $r(s', a) > 0$, for all actions, a, and for all other states and actions, s, a, $r(s, a) = 0$. Based on this knowledge and the learned Q-table, what is that state, and what is the reward? Provide an explanation for your answer.

1.2.3 Finding walls [6 points]

Suppose in addition to the facts from (1.2.2), you are told there are interior walls, besides the outside walls. Where are the walls? Provide an explanation for your answer.

1.3 Domain Representation in Policy Learning [10 points]

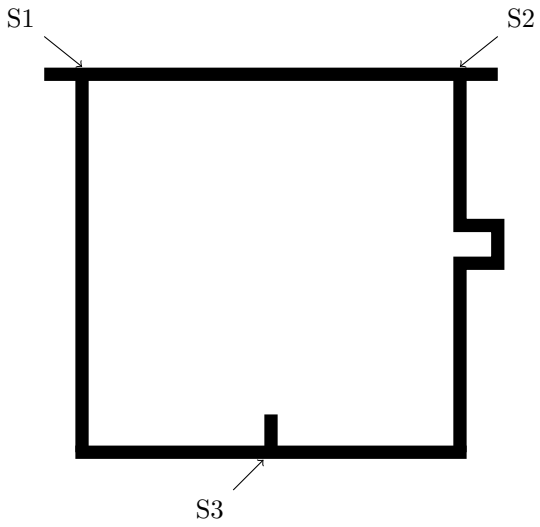

Figure 1: The three T-shaped intersections

In this problem we explore how domain representation affects performance and computational time when learning policies for MDPs.

A robot moves around in a hallway with three T-shaped intersections, S1, S2, and S3, shown in the figure (1). At each intersection, the robot has the choice to turn left or right, and then it follows the hallway until it reaches the next intersection. At each dead end is a portal that teleports the robot back to S3 facing south, where it must chose again to turn right or left.

The true transition matrix and reward function are shown in table 3. All actions are deterministic. At both S1 and S2, turning right gives positive reward and turning left gives negative reward, but the positive reward in S2 is slightly higher. In S3, both right and left give negative reward, but turning left gives a slightly more negative reward.

Action "left"

	S1	S2	S3
S1	0	0	1
S2	0	0	1
S3	0	1	0

Action "right"

	S1	S2	S3
S1	0	0	1
S2	0	0	1
S3	1	0	0

Reward function

	right	left
S1	+0.7	-1.0
S2	+1.0	-1.3
S3	-0.5	-0.7

Table 3: The State Transition Matrix and Reward Function

We consider two representations of the domain, represented by two robots, one (robot one) which has a sensor that tells it whether it is in S1, S2, or S3 (allowing it to fully represent the true state). The other robot (robot two) has a simpler representation, having only a compass allowing it to tell if it's facing north or south (thus it can distinguish between S3 and S1 or S2, but can't distinguish between S1 and S2).

1.3.1 Robot One [2 points]

Using a discount factor of $\gamma = 0.9$, robot one learns the Q-table shown in table 4. Answer the following questions:

	right	left
S1	+1.64	-0.06
S2	+1.94	-0.36
S3	+0.96	+1.03

Table 4: Robot one's Q-table

(a) What does the optimal policy look like for robot one?

(b) What is the value of executing this policy?

1.3.2 Robot Two [2 points]

Robot two uses a uniform exploration strategy while learning, so the observed reward in the "north" state is the average of the rewards in S1 and S2, as each are visited roughly equally. Under this exploration strategy, with a discount factor of $\gamma = 0.9$, robot two learns the Q-table shown in table 5.

	right	left
North	+2.08	+0.08
South	+1.38	+1.18

Table 5: Robot two's Q-table

(a) What is the optimal policy for robot two? How is this policy represented in the original domain, i.e., how would robot one represent this policy?

(b) What is the value of executing this policy in the original domain, i.e., what would the value be if this policy were executed by robot one?

1.3.3 The Importance of Representation [6 points]

Based on the results from robot one and robot two, answer the following questions:

(a) Which of the policies performs better on the original domain? Can the optimal policy for the original domain be expressed in both domains? Discuss why the optimal policies of the robots end up being different.

(b) How does the computational cost of value iteration compare between the two domains?

(c) What implications does this have for policy learning in large spaces? What are the tradeoffs in chosing the domain? How might we know if the policy we learn with our chosen domain is close to optimal?

1.4 Graduate Student Question: Gibbs Sampling and its Limitations [20 points]

As we saw in lecture, Gibbs sampling is an extremely useful tool for scenarios where it is infeasible to sample from the joint distribution. However, as with most heuristic algorithms, Gibbs sampling also has its limitations. In this problem we will explore these limitations.

1.4.1 Getting Stuck [8 points]

Consider the following distribution over four states:

$$p(x=0,y=0) = p(x=1,y=1) = 0.5; \qquad p(x=0,y=1) = p(x=1,y=0) = 0 \ .$$

(a) [4 points] Fill out the following conditional probabilities:

$$p(x=1|y=0) = ?$$
$$p(x=1|y=1) = ?$$

(b) [4 points] Given starting state $(x=1,y=1)$, write the probability distribution over the states returned from Gibbs sampling, and give a brief explanation of why Gibbs sampling behaves the way it does here.

1.4.2 Convergence issues [12 points]

Now consider the set of all boolean strings of length 100:

$$\{x \,|\, \forall 1 \leq i \leq 100,\, x_i \in \{0,1\}\} = \{0,1\}^{100} \ .$$

Let 0^{100} denote the all-zero string. Consider the distribution given by

$$p(0^{100}) = 0.5; \qquad p(x) = \frac{1}{2(2^{100}-1)} \ \ \forall x \neq 0^n \ .$$

Consider the Gibbs sampling algorithm on this distribution, where we iterate over all the variables from 1 to 100, and sample a value for each variable in turn.

(c) [4points] Given the initial state $x_i = 0$ for all $1 \leq i \leq 100$, what is the probability of sampling 1 at step j of Gibbs sampling, assuming that you sampled 0 at all steps $k < j$. (Your solution should be "simple enough" that we can directly plug your solution into a calculator.)

(d) [4 points] Now compute the probability of updating to any state where $x_i = 1$ for one or more i. (The same remarks regarding the "simplicity" of your solution apply for this problem as they did for the previous problem.)

(e) [4 points] Briefly explain why this shows that convergence to the actual distribution will take a large amount of time in expectation. As a sidenote, it can be shown that it is similarly improbable to reach the all-zero state from any other state, but this requires somewhat more tricky probability calculations.

2 Programming [30 points]

Amayon, the online shipping conglomerate, is having issues with their drones. As a result of compass interference the drones do not always move in the desired direction. Instead they attempt to move in the desired direction with probability 0.8, the direction to the left with probability 0.1, and the direction to the right with probability 0.1. If an attempted move is into an obstacle or off the grid, that attempt would result in staying in-place. To prevent the company from losing its drones you must develop an optimal policy to direct the dysfunctional drones back to a company warehouse.

You will be given a 2d map of the surrounding area, a list of no-fly-zone locations, and a list of warehouse locations. On the 2d map obstacles are marked as None. Free spaces, warehouses, and no-fly-zones are marked with their corresponding rewards (the reward in each state is the same for all actions). Each square is free, an obstacle, a warehouse, or a no-fly-zone. Drones can only move North, South, West or East from their current position (encoded as N,S,W,E respectively). The warehouses and no-fly-zones are terminal states (the drone crashes by moving into a no-fly-zone).

The following is a sample input.

1	-0.1	-2
-2	-0.1	-0.1
1	-0.1	None

Warehouses = [(0,0), (2,0)] No-Fly-Zones = [(0,2), (1,0)]

Note that (2,2) is a wall and all the positions with reward -0.1 are free spaces.

2.1 Value Iteration [15 points]

Implement the `valueiteration(rewards,ware,nofly)`, function by performing value iteration on the input MDP. Return the utilities following each iteration and the final optimal policy. Use a discount factor of 0.9 and terminate the iteration when $\delta <= 0.001 * (1 - discount) * discount$ where δ is the maximum absolute difference of utilities for a single state between consecutive iterations. Break ties in N,S,W,E order (you should allow ties with a small error).

2.2 Policy Iteration [15 points]

Implement the `policyiteration(rewards, ware, nofly)` function by performing policy iteration on the input MDP. Return the policies following each iteration and the final optimal policy. Use a discount factor of 0.9 and a starting policy of all N. When performing policy evaluation use the same termination condition as value iteration. Break ties in N,S,W,E order (you should allow ties with a small error).