Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.3800 Introduction to Inference
Fall 2024

Computational Lab III

Issued: Friday, October 11 **Due:** Friday, November 1 at 11:59pm

Robot Localization

Your pet robot is stuck on Mars somewhere on a 12×8 grid. You need to figure out where your robot is over time so that you can rescue it.

Let random variable $Z_i \in \{0, 1, \ldots, 11\} \times \{0, 1, \ldots, 7\}$ represent the robot's position at time i. For example, $Z_2 = (5, 4)$ means that at time step 2, the robot is in column 5, row 4. Let $A_i \in \{\texttt{stay}, \texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$ be the robot's most recent action at time i. The robot's action at any time step depends on its previous action. In particular, if the robot's previous action was a movement, it moves in the same direction with probability 0.9 and stays put with probability 0.1. If the robot's previous action was to stay put, it chooses an action at random (i.e., all 5 options are equally likely). On the boundary of the grid, the robot's behavior must adjust but is consistent with above. For instance, at the top of the grid, the robot cannot go any higher. When the robot is at one of the boundary locations, the probabilities of the possible actions are re-normalized so that they sum to 1. For example, if $Z_i = (0, 0)$ and $A_i = \texttt{left}$, then $A_{i+1} = \texttt{stay}$ with probability 1.

Such boundary cases suggest that the transition probabilities depend on the robot's previous action *and* its current location. Thus, we model the robot's hidden state at time i as a super variable $X_i = (Z_i, A_i)$ as depicted in Fig. 1. The robot's initial position Z_0 is equally likely to be any of the grid locations, and its initial action $A_0 = \texttt{stay}$.

Unfortunately, you cannot directly observe the robot's hidden state X_i. Instead, you have access to a noisy sensor that puts a uniform distribution on valid grid positions within one grid cell of the robot's current true position. The sensor cannot tell you

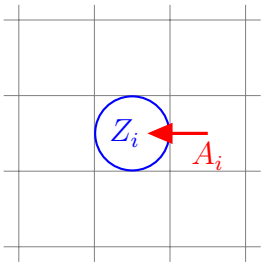


Figure 1: Robot's hidden state $X_i = (Z_i, A_i)$ where the blue circle indicates robot's position Z_i and the red arrow indicates its most recent action A_i.

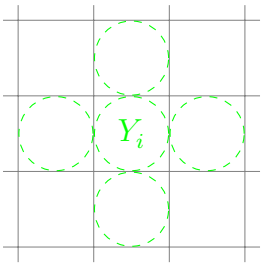
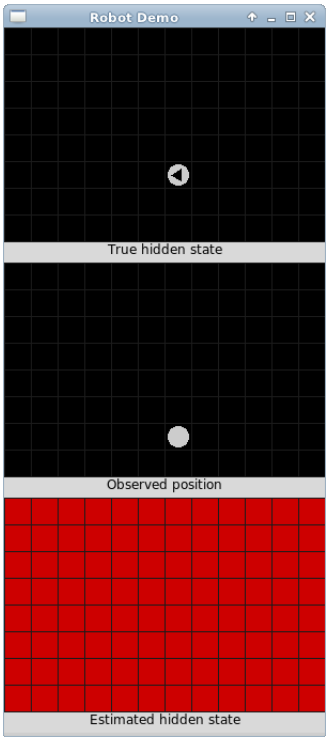


Figure 2: Distribution of Y_i given the true location Z_i shown in Fig. 1. Each of the 5 locations shown in green is equally likely.

what actions the robot takes. In other words, at time i we observe random variable $Y_i \in \{0, 1, \ldots, 11\} \times \{0, 1, \ldots, 7\}$ as depicted in Fig. 2. Y_i is uniformly distributed over the possible locations determined by Z_i.

Code, Data and Visualization. File `robot.zip` contains the following files:

- File `inference.py` will contain all of your code. It also shows how to generate HMM samples. This is the only file you will modify.
- File `robot.py` contains functions for generating the initial distribution, the transition probabilities given a current hidden state, and the observation probabilities given a current hidden state, so you need not re-implement these.
- File `test.txt` contains data for part (b).
- File `test_missing.txt` contains data for part (c).
- File `graphics.py` provides graphics code. You do not need to read this file.



When you run `python inference.py`, you should be able to see your robot move around as shown on the right. The top pane shows the true (hidden) state of the robot, which includes location and the most recent action, represented by an arrow. The middle pane shows the observed position of the robot. The bottom pane shows the estimated state of the robot, with red signifying missing data. After you implement your inference algorithms, the bottom pane will automatically show your estimate. This visualization will help you see what your code does. You can also turn the visualization off by setting variable `enable_graphics` to `false` in `inference.py` file. Whether you use visualization or not will *not* affect grading.

To use the visualization functions, make sure to have Python 3.10 or above installed. You will also need `tkinter` module installed. On MAC-OS, you can install it by running `pip install tk`. You can install it on Linux by running `sudo apt-get install python python3-tk`. If you are using `ssh` to login to Athena, be sure to use the `-X` flag with `ssh` to enable visualization on your computer.
Our solution runs in 2 seconds. While we are not grading the speed of your code, please double check your code if it runs significantly faster or slower.

(a) Suppose we get two observations, $y_0 = (5, 4)$ and $y_1 = (6, 5)$. Compute the forward message $m_{0\to1}(\cdot)$ and the backward message $m_{1\to0}(\cdot)$ in the forward-backward algorithm, and the marginal distributions of X_0 and X_1 given these

observations. We recommend you solve this part by hand and use the results later to verify that your answers here and in part (b) are consistent.

(b) Complete the function `forward_backward()` to implement the forward-backward algorithm that takes as input observations $y_0, \ldots y_{n-1}$ and computes the marginal distribution $p_{X_i|Y_0,\ldots,Y_{n-1}}(\cdot|y_0,\ldots,y_{n-1})$ for $i = 0,\ldots,n-1$. Run `python inference.py --load=test.txt` and determine the marginal distribution $p_{X_0|Y_0,\ldots,Y_{99}}(\cdot \mid y_0,\ldots,y_{99})$ of the robot's initial state and the marginal distribution $p_{X_{99}|Y_0,\ldots,Y_{99}}(\cdot \mid y_0,\ldots,y_{99})$ of the robot's final state for this sequence of observations. Only include state values (i.e., position-action configurations) with non-zero probability in your answer.

Hint: You may find it helpful to work with log probabilities to avoid arithmetic underflow and to add special handling of state values whose probability is zero.

Sanity checks: 1) If you run your code on test_small.txt, you should obtain the answer you got in part (a).

2) For the full data file test.txt, the marginal distribution of the robot's state at time $i = 1$ is

$$p_{X_1|Y_0,\ldots,Y_{99}}(x|y_0,\ldots,y_{99}) = \begin{cases} 0.5 & \text{if } x = \texttt{(6, 5, down)}, \\ 0.5 & \text{if } x = \texttt{(6, 5, right)}, \\ 0 & \text{otherwise.} \end{cases}$$

3) The distribution of X_{99} assigns non-zero probability to exactly three states and has a mode at `(11, 0, stay)`. The value of the distribution at the mode is 0.81.

(c) Some of the observations were lost when they were transmitted from Mars to Earth. Modify the `forward_backward()` function so that it can handle missing observations, i.e., where at some time steps, there is no observation. In a list of observations, a missing observation is listed as `None` rather than the usual tuple `(x, y)`. Run `python inference.py --load=test_missing.txt` and determine the marginal distributions for X_0 and X_{99} given the available observations. Only include state values (i.e., position-action configurations) with non-zero probability in your answer. Explain your modifications to the `forward_backward()` function.

Sanity check: The mode of the distribution of X_{99} should be `(3, 0, 'right')`, which has probability 0.9.

(d) Unsatisfied with maximizing marginal distributions, you now seek the most likely *sequence* of states the robot has visited given the observations of its location. Complete the function `Viterbi()` to implement the Viterbi algorithm. Your implementation should be able to handle missing observations. Run `python`

`inference.py --load=test_missing.txt` and determine the last ten states in the MAP estimate.

Sanity check: The first state of the MAP estimate should be `(0, 5, up)`. The last 3 states of the MAP estimate should be: `(1, 0, right)`, `(2, 0, right)`, `(3, 0, right)`.

Hint: You might find it helpful to first implement the Viterbi algorithm that assumes we have all observations and test it by running `python inference.py --load=test.txt`

(e) **(Optional, not graded – just for your enjoyment!)** Implement a modified version of the Viterbi algorithm that outputs the second-best solution to the MAP problem rather than the best solution by completing the function `second_best()`. Run `python inference.py --load=test.txt` and determine the last ten states in the second-best sequence.

What to hand in: Upload to Gradescope your writeup as a single PDF file and your code as `inference.py`. The writeup should include your answers for parts (a)-(d). Scans/images of handwritten work are fine but please write neatly — we can't grade what we can't decipher!

Gradescope will run a number of tests which are visible to you. These are to make sure your code runs correctly in the Gradescope system. Please make sure you pass these tests as we will use the autograder to check your submission.