School of Computer Science
The University of Adelaide

Artificial Intelligence
Assignment 3

Semester 1 2022
Due 11:59pm Sunday 5 June 2022

---

1 Robot localisation (UG and PG)

Robot localization is the process of determining where a mobile robot is located concerning its environment. Robot localization provides an answer to the question: *Where is the robot now*? A reliable solution to the localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robot's own location is an essential precursor to making decisions about future actions.

In a typical robot localization scenario, a map of the environment is available and the robot is equipped with sensors that observe the environment as well as monitor its own motion. The localization problem then becomes one of estimating the robot's position within the map using information gathered from these sensors. The following shows an example of a 2D map drawn using ASCII characters:

```
0 0 0 0 X 0 0 0 0 X
X X 0 0 X 0 X X 0 X
X 0 0 0 X 0 X X 0 0
0 0 X 0 0 0 X 0 0 0
```

The character 'X' denotes an obstacle (the path cannot be traversed), while the character '0' (Zero) represents a traversable positions. Any position within the map is indicated by the coordinates (k, j), where k is the row number (ordered top to bottom) and j is the column number (ordered left to right), starting from 1. For example, the top left position is $(1, 1)$ and the bottom right is $(4, 10)$.

A robot moves in the map and gathers information along the way. In this version, the robot has a single non-deterministic *Move* action and its sensors reports perfectly whether or not obstacles lay immediately to the north, south, east, and west. A sequence

of sensor readings observed is the set of possible blocked directions. The following shows the example of the observed sensor readings at each time step; which means e.g., at time 1, the north (N), south (S) and west (W) of the robot have obstables:

Time Steps	1	2	3	4
Blocked direcition(s)	NSW	NS	N	NE

1.1 Problem formulation

- The state variable X_t represents the location of the robot on the discrete grid; the domain of this variable is the set of traversable points illustrated as '0' points in the map.

- Let NEIGHBORS(i) be the set of traversable points that are adjacent to and let $N(i)$ be the size of that set. Then the transition model for the Move action is defined as follows, which means the robot has equal probability to move to each of its neighbours.

$$P(X_{t+1} = j | X_t = i) = \begin{cases} 1/N(i) & \text{if j} \in \text{NEIGHBORS}(i) \\ 0 & \text{otherwise} \end{cases}$$

  We don't know where the robot starts, so we will assume a uniform distribution over all the sates; that is, $P(X_0 = i) = 1/K$, where K is the number of '0' points in the map.

- The sensor variable E_t has 16 possible values, each a four-bit sequence giving the presence or absence of an obstacle in each of the compass directions (North, East, South and West). **NESW** is the way of specifying the four-bit sequence and **your program must expand each direction in this order**. For example, a four-bit sequence 1010 represents that the sensors report an obstacle on its north and south positions and the east and west positions do not have obstacles.

- The sensors' error rate is ϵ and that errors occur independently for the four sensors. In that case, the probability of getting all four bits right is $(1 - \epsilon)^4$, and the probability of getting them all wrong is ϵ^4. Furthermore, if d_{it} denotes the number of directions are reporting erroneous values, then the probability that a robot at position i would receive a sensor reading e_t (i.e., the observation/emission model) is:

$$P(E_t = e_t|X_t = i) = (1 - \epsilon)^{4-d_{it}} \epsilon^{d_{it}}$$

Using the above problem formulation, you are requested to use the Viterbi forward algorithm provided below to find the Trellis matrix. A trellis matrix gives us the probability of each state (path in this case) given each observation made by the robot's sensors. For the path(s) which cannot be traversed, the probability is 0 (Zero) in the trellis matrix. Therefore, the trellis matrix could reflect the possible positions of the robot.

Algorithm 1 Viterbi forward algorithm

input: **O**, observation space O = {o_1, o_2,...,o_N}.
S, state space S = {s_1,s_2,...,s_K} // Here, K refers to the traversable positions.
$\prod$, array of initial probabilities $\prod = (\pi_1,\pi_2,...,\pi_K)$ // Here, $\pi_1 = \pi_2 =, ..., \pi_K$.
Y, a sequence of observations Y = (y_1,y_2,...,y_T).
Tm, transition matrix of size K x K.
Em, emission matrix of size K x N.
output: Trellis matrix
for each position $i = 1, 2, ..., K$ **do**
    trellis[i,1] $\leftarrow \pi_i$ * Em_{iy_1}
end for
for each observation $j = 2, 3, ...T$ **do**
    for each state $i = 1, 2, ...K$ **do**
        trellis[i,j] $\leftarrow \max_k$(trellis[k, j - 1] * $Tm_{ki} * Em_{iy_j}$) // Here, k is the most likely prior positions and y_j is the observation at time j.
    end for
end for
return trellis

1.2 Deliverables

Write your robot localisation program in Python 3.6.9 in a file called `viterbi.py`. Your program must be able to run as follows:

```
$ python viterbi.py [input]
```

Input: Here, `[input]` specifies the path to a text file. In GradeScope, the file path will be provided. Your program need to be able to read the file in the following format:

```
4 10
0 0 0 0 X 0 0 0 0 X
X X 0 0 X 0 X X 0 X
X 0 0 0 X 0 X X 0 0
0 0 X 0 0 0 X 0 0 0
4
1011
1010
1000
1100
0.2
```

The description of the input file to the program is as follows:

- The first line indicates the size of the map (rows by columns).
- The map data then follows for the specified number of rows, where all values are either 0 or X.
- Next line specifies the number of sensor observation(s).
- The observed values then follows for the specified number of rows corresponding to each time step.
- The last line specifies the sensor's error rate ϵ.

Given the inputs, your program must compute a trellis matrix (following the prescribed Algorithm 1) using the input provided. Your program must then build a map representations of the probabilities of each position at each time step using the trellis matrix. Remember, the path(s) which cannot be traversed, the probability is 0 (Zero) in the trellis matrix.

Output: Your program is required to output the map representations in a file called **output.npz**. We will assess the content of this output file. A sample of the output file is depicted below.

Each map representation is of the same size as of the input map. Below shows 2 map representations, representing the probabilities of all the positions present in the map at each time step (sensor observations). Since, we have 4 observations in the above example input file, your program will output 4 such map representations.

$$\begin{bmatrix} 0.01575 & 0.00394 & 0.00098 & \dots & 0.00000 \\ 0.00000 & 0.00000 & 0.00098 & \dots & 0.00000 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.01575 & 0.00025 & 0.00000 & \dots & 0.00025 \end{bmatrix}$$

$$\begin{bmatrix} 0.00020 & 0.00645 & 0.00020 & \dots & 0.00000 \\ 0.00000 & 0.00000 & 0.00000 & \dots & 0.00000 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.00001 & 0.00040 & 0.00000 & \dots & 0.00001 \end{bmatrix}$$

And so on..

Hint: For saving these maps in a **output.npz** file. You can use Numpy's `savez()` official documentation.

```
maps = list of numpy matrices at each time step.
np.savez("output.npz", *maps)
```

1.3 Python libraries

You are allowed to use the Python standard library to write your decision tree learning program (see `https://docs.python.org/3/library/` for the components that make up the Python v3.6.9 standard library). In addition to the standard library, you are allowed to use NumPy. Note that the marking program will not be able to run your program to completion if other third party libraries are used.

1.4 Submission

You must submit your program files on Gradescope. Instructions on accessing Gradescope and submitting assignments are provided at `https://help.gradescope.com/article/5d3ifaeqi4-student-canvas`. Please use the course code **X3ZJZE** to enrol into the course.

For undergraduates, please submit your robot localisation program (`viterbi.py`) to **Assignment 3 - Undergraduates**. If there are any questions or issues with Gradescope, please contact Arpit Gole via email at arpit.gole@adelaide.edu.au.

1.5 Expected run time

Your program must be able to terminate within 300 seconds for the given 2D map.

1.6 Assessment

We will compile and run your code on several test problems. If it passes all tests, you will get **15%** (undergrads) or **12%** (postgrads) of the overall course mark.

There will be no further manual inspection/grading of your program to award marks on the basis of coding style, commenting or "amount" of code written.

1.7 Using other source code

You should not use other source code(s) for this assignment. All submitted code must be your own work written from scratch. Only by writing the solution yourself will you fully understand the concept of the Viterbi algorithm.

1.8 Due date and late submission policy

This assignment is due by 11:59pm Sunday 5 June 2022. If your submission is late the maximum mark you can obtain will be reduced by 25% per day (or part thereof) past the due date or any extension you are granted.

Continues next page for postgraduate section.

2 Forward-backward algorithm for smoothing (PG only)

For postgraduate students, completing this section successfully will give you the remaining 3% of the marks.

The problem settings remain the same as described in Sec. 1.1. In this task, you will implement the Forward-backward algorithm to smooth a sequence as shown in Algorithm 2. For more backgrounds on this algorithm, please refer to Figure 14.4 in Stuart Russell, Peter Norvig - Artificial Intelligence A Modern Approach (4th Edition).

It is important to note that the Algorithm 1 finds the sequence by considering the joint probabilities over all the time steps and Algorithm 2 computes posterior probabilities by smoothing over single time steps.

Algorithm 2 Forward-backward algorithm for smoothing

input: **ev**, a sequence of observations for steps 1,...,t.
prior, the prior distribution on the initial state, $P(X_0)$.
output: A vector of probability distributions.
local variables: **fv**, a vector of forward messages for steps 0,...,t.
b, a representation of the backward message, initially all 1s.
sv, a vector of smoothed estimates for steps 1,...,t.
$fv[0] \leftarrow prior$
for $i = 1$ to t **do**
    $fv[i] \leftarrow$ FORWARD($fv[i-1]$, $ev[i]$)
end for
for $i = t$ down to 1 **do**
    $sv[i] \leftarrow$ NORMALIZE($fv[i]$ * b)
    $b \leftarrow$ BACKWARD(b, $ev[i]$)
end for
return sv

Algorithm 2 computes posterior probabilities of each position in a sequence, given a sequence of sensor observations . The posterior probabilities computed by smoothing are distributions over single time steps. The FORWARD, NORMALIZE and BACKWARD operators are defined as follows:

- FORWARD

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha \underbrace{\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})}_{\text{sensor model}} \sum_{\mathbf{x}_t} \underbrace{\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)}_{\text{transition model}} \underbrace{\mathbf{P}(\mathbf{x}_t|\mathbf{e}_{1:t})}_{\text{recursion}}$$

- NORMALIZE: It is used to make probabilities sum up to 1.
- BACKWARD

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \sum_{\mathbf{x}_{k+1}} \underbrace{\mathbf{P}(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})}_{\text{sensor model}} \underbrace{\mathbf{P}(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})}_{\text{recursion}} \underbrace{\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)}_{\text{transition model}}$$

For more detailed understanding please follow lecture slides and the textbook.

2.1 Deliverables

Write your Forward-backward for smoothing program in Python 3.6.9 in a file called `forward-backward.py`. Your program must be able to run as follows:

```
$ python forward-backward.py [input]
```

Input: Here, `[input]` specifies the path to a text file. In GradeScope, the file path will be provided. The input file format is exactly the same as defined in the above Sec. 1.2.

Given the inputs, your program must compute a smoothed estimates for each time step (sensor observations), following Algorithm 2 using the input provided. Your program must then build a map representations of the probabilities of each position at each time step using the smoothed estimates.

Output: Your program is required to output the map representations in a file called **output.npz**. We will assess the content of this output file. The content of this file is exactly the same as defined in the above Sec. 1.2.

Submit your program in the same way as the submission for Sec. 1. **For postgraduates**, please submit your robot localisation programs (`viterbi.py` and `forward-backward.py`) to **Assignment 3 - Postgraduates**. The due date, late submission policy, python libraries, execution run time and code reuse policy are also the same as in Sec. 1.

$\sim\sim\sim$ The End $\sim\sim\sim$