CS 486/686 Assignment 3 (79 marks)

Alice Gao

Due Date: 11:59 PM ET on Wednesday, July 21, 2021

Changes

- v1.1: Q1a, 1b, 1c, corrected the lexicographical order example to be $(O_0, O_1, O_2, S_0, S_1, S_2)$.
- v1.2: Updated the typo in the recursive case of the backward recursion formula. It should be identical to the formula given in lecture 15.
- v1.3: Updated the backward recursion description to indicate that the b values do not need to be normalized.
- v1.4: Added an example of defining the umbrella environment in the program.

Academic Integrity Statement

I declare the following statements to be true:

- The work I submit here is entirely my own.
- I have not shared and will not share any of my code with anyone at any point.
- I have not posted and will not post my code on any public or private forum or website.
- I have not discussed and will not discuss the contents of this assessment with anyone at any point.
- I have not posted and will not post the contents of this assessment and its solutions on any public or private forum or website.
- I will not search for assessment solutions online.
- I am aware that misconduct related to assessments can result in significant penalties, possibly including failure in the course and suspension. This is covered in Policy 71: https://uwaterloo.ca/secretariat/policies-procedures-guidelines/policy-71.

Failure to accept the integrity policy will result in your assignment not being graded.

By typing or writing my full legal name below, I confirm that I have read and understood the academic integrity statement above.

Instructions

- Submit any written solutions in a file named `writeup.pdf` to the A3 Dropbox on `Learn`. Submit any code to `Marmoset` at https://marmoset.student.cs.uwaterloo.ca/. No late assignment will be accepted. This assignment is to be done individually.
- I strongly encourage you to complete your write-up in Latex, using this source file. If you do, in your submission, please replace the author with your name and student number. Please also remove the due date, the Instructions section, and the Learning goals section. Thanks!
- Lead TAs:
  - Ji Xin (ji.xin@uwaterloo.ca)

  The TAs' office hours will be posted on MS Teams.

Learning goals

Inference in Bayesian Networks

- Define factors. Manipulate factors using the operations restrict, sum out, multiply and normalize.
- Trace the execution of and implement the variable elimination algorithm for calculating a prior or a posterior probability given a Bayesian network.

Inference in Hidden Markov Models

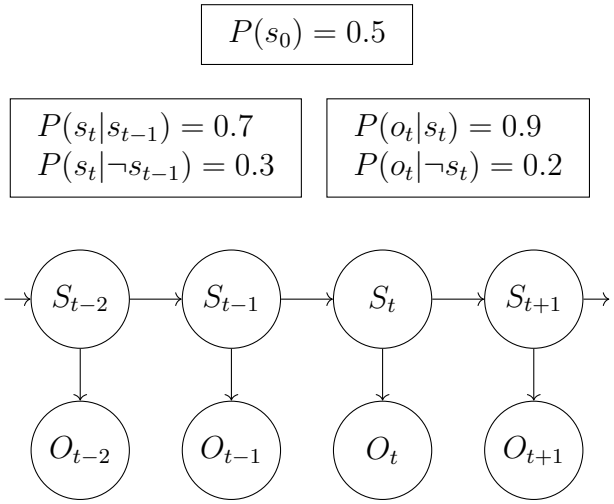
- Construct a hidden Markov model given a real-world scenario.
- Perform filtering and smoothing by executing the forward-backward algorithm.

The Umbrella and Robot Environments

We can model many problems using hidden Markov models. In this section, I include two examples. The first one is the umbrella environment from lectures. The second one is the robot localization problem from a textbook.

The Umbrella Environment

In lectures, I introduced the following umbrella environment. I discussed modeling this problem as a hidden Markov model and calculating the filtered and smoothed probabilities at each time step.



The Full Robot Environment

Let's consider a robot localization problem. This problem is similar to the example on page 389 of the Artificial Intelligence: Foundations of Computational Agents book by Poole and Mackworth.

There is a circular corridor with 16 locations numbered 0 to 15. There are doors at positions 2, 4, 7, and 11. There is no door at any other location.

There is an arbitrary number of time steps, starting at time step $t = 0$. At each time step, the robot is at one of the 16 locations. Let $S_t \in \{0, 1, \ldots, 15\}$ denote the robot's location at time step t.

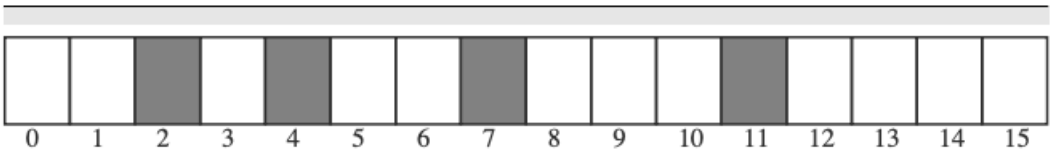


Figure 8.14: Localization domain

Figure 1: The Robot Localization Problem (from the Poole and Mackworth Textbook)

The robot's sensor noisily observes whether it is in front of a door or not. Let $O_t \in \{0, 1\}$ denote the robot's observation at time step t. $O_t = 0$ if the robot observes a door and $O_t = 1$ if the robot does not observe a door. The observation probabilities are given below.

$$P(O_t = 0 | S_t \in \{2, 4, 7, 11\}) = 0.8 \tag{1}$$
$$P(O_t = 0 | S_t \notin \{2, 4, 7, 11\}) = 0.1 \tag{2}$$

At each time step, the robot can choose one of three actions: move left, move right, and stay still. Let $A_t \in \{0, 1, 2\}$ denote the robot's action at time step t. $A_t = 0$ denotes moving left, $A_t = 1$ denotes moving right, and $A_t = 2$ denotes staying still. Taking an action may cause the robot to stay in the same location or move to a different location. The effects of the three actions are shown below.

The effects of moving left at time t:

$$P(S_{t+1} = x | A_t = 0 \wedge S_t = x) = 0.05 \tag{3}$$
$$P(S_{t+1} = x - 1 \,(\text{mod } 16) | A_t = 0 \wedge S_t = x) = 0.90 \tag{4}$$
$$P(S_{t+1} = x - 2 \,(\text{mod } 16) | A_t = 0 \wedge S_t = x) = 0.05 \tag{5}$$

The effects of moving right at time t:

$$P(S_{t+1} = x | A_t = 1 \wedge S_t = x) = 0.15 \tag{6}$$
$$P(S_{t+1} = x + 1 \,(\text{mod } 16) | A_t = 1 \wedge S_t = x) = 0.70 \tag{7}$$
$$P(S_{t+1} = x + 2 \,(\text{mod } 16) | A_t = 1 \wedge S_t = x) = 0.15 \tag{8}$$

The effects of staying still at time t:

$$P(S_{t+1} = x | A_t = 2 \wedge S_t = x) = 1.0 \tag{9}$$

The environment has the following properties:

- The robot's observation at time t depends only on the robot's location at time t. That is, for each t, O_t depends on S_t only.
- The robot's location at time $t+1$ only depends on its location at time t and its action at time t. That is, for each t, S_{t+1} depends on S_t and A_t only.

The robot starts at an unknown location. At any time step, the robot tries to determine its location by estimating the probability of being in each location given its past observations and past actions.

1 Locating the Robot by VEA (24 marks)

You will calculate some probabilities for a simplified robot localization problem by executing the variable elimination algorithm.

The Simplified Robot Localization Problem:

The circular corridor has 2 locations (0, 1). There is no door at location 0 and a door at location 1. The observation probabilities are as follows.

$$P(O_t = 0|S_t = 1) = 0.8 \tag{10}$$
$$P(O_t = 0|S_t = 0) = 0.1 \tag{11}$$

The robot still has three actions, but the dynamics of the actions are simplified.

The effects of moving left at time t:

$$P(S_{t+1} = x|A_t = 0 \wedge S_t = x) = 0.05 \tag{12}$$
$$P(S_{t+1} = x - 1 \,(\text{mod } 2)|A_t = 0 \wedge S_t = x) = 0.95 \tag{13}$$

The effects of moving right at time t:

$$P(S_{t+1} = x|A_t = 1 \wedge S_t = x) = 0.15 \tag{14}$$
$$P(S_{t+1} = x + 1 \,(\text{mod } 2)|A_t = 1 \wedge S_t = x) = 0.85 \tag{15}$$

The effects of staying still at time t:

$$P(S_{t+1} = x|A_t = 2 \wedge S_t = x) = 1.0 \tag{16}$$

We will represent this problem for three time steps (0 to 2) using the graphical model in Figure 2.

An important note regarding the actions: The only purpose of observing the action at each time step is to determine the transition probabilities. We do not need to model the actions explicitly since they are not really random variables. For this reason, we have put the actions into the states for the time steps 0 and 1.

Please answer the following questions.

Using the Bayesian network in Figure 2, answer the questions below by executing the variable elimination algorithm.

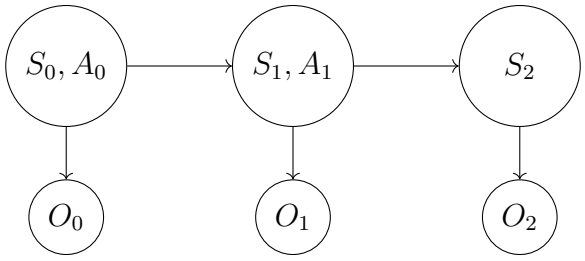


Figure 2: A Graphical Model of Simplified Robot Localization

(a) Answer the question below by executing the variable elimination algorithm using the provided information. Eliminate the hidden variables in **lexicographical order** $(O_0, O_1, O_2, S_0, S_1, S_2)$.

Please provide the final answer only. Make sure that your answer has 4 decimal places.

Problem: At $t = 2$, what is the probability that the robot is in location 0?

You can use the information below.

- At $t = 0$, the robot's prior belief over the two locations is a uniform distribution. That is, $P(S_0 = 0) = 0.5$ and $P(S_0 = 1) = 0.5$.
- At $t = 0$, the robot does not observe the door and decides to move right,
- At $t = 1$, the robot observes the door and decides to move left.
- At $t = 2$, the robot observes the door.

Marking Scheme: (2 marks) Correct answer up to 3 decimal places. All or nothing.

(b) Answer the question below by executing the variable elimination algorithm using the provided information. Eliminate the hidden variables in **lexicographical order** $(O_0, O_1, O_2, S_0, S_1, S_2)$.

Please provide the final answer only. Make sure that your answer has 4 decimal places.

Problem: At $t = 1$, what is the probability that the robot is in location 0?

You can use the information below.

- At $t = 0$, the robot's belief over the two locations is a uniform distribution.

That is, $P(S_0 = 0) = 0.5$ and $P(S_0 = 1) = 0.5$.

- At $t = 0$, the robot does not observe the door and decides to move right,
- At $t = 1$, the robot observes the door and decides to move left.
- At $t = 2$, the robot observes the door.

Marking Scheme: (2 marks) Correct answer up to 3 decimal places. All or nothing.

(c) Answer the question below by executing the variable elimination algorithm using the provided information. Eliminate the hidden variables in **lexicographical order** $(O_0, O_1, O_2, S_0, S_1, S_2)$.

Show all the steps of executing the variable elimination algorithm based on the formats in section 1.1. Make sure that your final answer has 4 decimal places.

Problem: At $t = 1$, what is the probability that the robot is in location 0 and does not observe a door?

You can use the information below.

- At $t = 0$, the robot's belief over the two locations is a uniform distribution. That is, $Pr(S_0 = 0) = 0.5$ and $Pr(S_0 = 1) = 0.5$.
- At $t = 0$, the robot observes a door and decides to move right.
- At $t = 1$, the robot decides to move left.
- At $t = 2$, the robot does not observe the door.

Marking Scheme: (20 marks)

- (2 marks) Correct answer up to 3 decimal places.
- (18 marks) The algorithm execution has the correct steps.

1.1 Showing the execution steps of VEA

Please describe the steps of the variable elimination algorithm using the format specified below. You do not have to follow the spacing exactly. We will mark the steps by hand.

1. Show a factor as `f(A B E)`. Use `f` to denote the name of every factor. Show the variables in **lexicographical order**. For example,

   ```
   f(A B E)
   ```

2. After each operation (restrict, multiply, sum out, normalize), show the resulting factor in a table. The columns and rows can be in any order. For example,

   ```
    B,   E,   Prob
    1,   1,  0.9600
    1,   0,  0.9500
    0,   1,  0.2000
    0,   0,  0.0100
   ```

3. Describe the **define** operation as shown below. Show the factors in any order.

   ```
   Define factors f(A G) f(A W)
   ```

4. Describe the **restrict** operation as shown below.

   ```
   Restrict f(B E) to B = 1 to produce f(E)
    E,   Prob
    1,  0.9600
    0,  0.9500
   ```

   If you need to perform the restrict operation on multiple factors and on multiple variables for each factor, go through the factors in **lexicographical order**. For each factor, go through the variables in **lexicographical order**. Show one operation per line.

   The example below performs two restrict operation on `f(A B E)`, one restrict operation on `f(A G)`, and one restrict operation on `f(A W)`.

   ```
   Restrict f(A B E) to A = 1 to produce f(B E)
    B,   E,   Prob
    1,   1,  0.9600
    1,   0,  0.9500
    0,   1,  0.2000
    0,   0,  0.0100
   ```

```
Restrict f(B E) to B = 1 to produce f(E)
 E,   Prob
 1,  0.9600
 0,  0.9500

Restrict f(A G) to A = 1 to produce f(G)
 G,   Prob
 1,  0.4000
 0,  0.6000

Restrict f(A W) to A = 1 to produce f(W)
 W,   Prob
 1,  0.8000
 0,  0.2000
```

5. Describe the **multiply** operation as shown below.

   ```
   Multiply f(A B E) f(B) to produce f(A B E)
   ```

   If you need to multiply more than two factors together, show the operation on a single line. The factors can be in any order. For example,

   ```
   Multiply f(E) f() f(E) f() f() to produce f(E)
    E,   Prob
    1,  0.0000
    0,  0.0001
   ```

6. Describe the **sum out** operation as shown below.

   ```
   Sum out W from f(W) to produce f()
   Prob
   1.0
   ```

7. Describe the **normalize** operation as shown below.

   ```
   Normalize f(E) to produce f(E)
    E,   Prob
    1,  0.0003
    0,  0.9997
   ```

See below for an example output for the Holmes network minus the node R. The network is given in lecture 13.

```
Computing P(E | A and B)

Define factors  f(A B E) f(A G) f(A W) f(B) f(E)

Restrict f(A B E) to A = 1 to produce f(B E)
 B,   E,   Prob
 1,  1,  0.9600
 1,  0,  0.9500
 0,  1,  0.2000
 0,  0,  0.0100

Restrict f(B E) to B = 1 to produce f(E)
 E,   Prob
 1,  0.9600
 0,  0.9500

Restrict f(A G) to A = 1 to produce f(G)
 G,   Prob
 1,  0.4000
 0,  0.6000

Restrict f(A W) to A = 1 to produce f(W)
 W,   Prob
 1,  0.8000
 0,  0.2000

Restrict f(B) to B = 1 to produce f()
Prob
0.0001

Sum out G from f(G) to produce f()
Prob
1.0

Sum out W from f(W) to produce f()
Prob
1.0

Multiply f(E) f() f(E) f() f() to produce f(E)
 E,   Prob
 1,  0.0000
 0,  0.0001

Normalize f(E) to produce f(E)
 E,   Prob
```

```
 1,  0.0003
 0,  0.9997

P(E | A and B) is 0.00030315693739914 51
```

2 Locating the Robot via an HMM (55 marks)

You will implement the forward-backward algorithm to perform inference in an hidden Markov model.

We have provided two python files: `utils_soln.py` and `fba.py`. `utils_soln.py` contains all the provided functions. `fbs.py` contains empty function stubs for you to complete.

Section 2.1 describes the forward-backward algorithm. Section 2.2 describes how you can model an environment using the `Environment` object in `utils_soln.py`.

Please complete the following tasks.

Implement the empty functions in `fba.py`. Submit `fba.py` only on Marmoset.

Marking Scheme: (55 marks)

Unit tests:

- `create_observation_matrix`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `create_transition_matrices`
  (1 public test + 2 secret tests) * 1 mark = 3 marks
- `forward_recursion`
  (1 public test + 6 secret tests) * 2 marks = 14 marks
- `backward_recursion`
  (1 public test + 6 secret tests) * 3 marks = 21 marks
- `fba`
  (1 public test + 6 secret tests) * 2 marks = 14 marks

2.1 The Forward-Backward Algorithm

There are t time steps in total, from 0 to $t-1$. The algorithm makes use of both forward and backward recursion.

Step 1 (Forward Recursion): Given the observations from time 0 to time $t-1$, forward recursion calculates the values $f_{0:k} = P(S_k|o_{0:k})$ for $0 \leq k \leq t-1$ using the recurrence relation below.

$$\text{Base case: } f_{0:0} = \alpha\, P(o_0|S_0)P(S_0) \tag{17}$$

$$\text{Recursive case: } f_{0:k} = \alpha\, P(o_k|S_k) \sum_{s_{k-1}} P(S_k|s_{k-1})\, f_{0:(k-1)}, \quad (1 \leq k \leq t-1) \tag{18}$$

The `forward_recursion` function returns a 2D `numpy` array of shape (t, number of states). The vector at index i where $0 \leq i \leq t-1$ is the normalized vector $f_{0:i}$.

Step 2 (Backward Recursion): Given observations from time 0 to time $t-1$, backward recursion calculates the values $b_{(k+1):(t-1)} = P(o_{(k+1):(t-1)}|S_k)$ for $0 \leq k \leq t-1$ using the recurrence relation below.

$$\text{Base case:} \tag{19}$$

$$b_{t:t-1} = \vec{1} \tag{20}$$

$$\text{Recursive case:} \tag{21}$$

$$b_{(k+1):t-1} = \sum_{s_{k+1}} P(o_{k+1}|s_{k+1})\, b_{(k+2):t-1}\, P(s_{k+1}|S_k), \quad (0 \leq k \leq t-2) \tag{22}$$

The `backward_recursion` function returns a 2D `numpy` array of shape (t+1, number of states). The vector at index 0 is not used by the algorithm and can be an arbitrary vector (our tests will ignore this vector). The vector at index i where $1 \leq i \leq t$ is $b_{i:(t-1)}$.

Step 3 (Forward-Backward Algorithm): Given the observations from time 0 to time $t-1$, the forward-backward algorithm calculates the values $P(S_k|O_0, \ldots, O_{t-1})$ for $0 \leq k \leq t-1$ using the equation below.

$$P(S_k|O_0, \ldots, O_{t-1}) = \alpha\, f_{0:k}\, b_{k+1:t-1}, \quad (0 \leq k \leq t-1) \tag{23}$$

The `fba` function returns a 2D `numpy` array of shape (t, number of states). The vector at index i where $0 \leq i \leq t-1$ is the normalized vector $P(S_i|O_0, \ldots, O_{t-1})$.

2.2 Modeling the umbrella, robot, or another environment

We can use the `Environment` object to model a variety of environment including the umbrella and robot environments. We highly recommend that you test your program using a variety of environments.

2.2.1 Modeling states

- The state types. (`num_state_types`, `state_types`)

  The states have different types. The state type determines the probabilities of generating different observations.

  For each environment, we will specify the number of state types (`num_state_types`) and a list of state types (`state_types`).

  For example,

  - The umbrella environment has 2 types of states: rain (type 0), or no rain (type 1). We can initialize it as follows.

    ```
    num_state_types = 2
    state_types = [0, 1]
    ```

  - The robot environment has 2 types of states: has door (type 0) or does not have door (type 1). We can initialize it as follows.

    ```
    num_state_types = 2
    state_types = [1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1]
    ```

- The number of states. We do not need to specify the number of states explicitly. The number of states is set to be the length of `state_types`.

  For example,

  - The umbrella environment has 2 states.
  - The robot environment has 16 states.

2.2.2 Modeling observation probabilities

The state type determines the probabilities of generating different observations. If two states have the same type, they generate the observations with the same probabilities. It is sufficient to specify the observation probabilities for each state type instead of for each state.

The `Environment` object's attribute `observe_matrix` stores the observation probabilities.

To create `observe_matrix`, we first define the number of observation types (`num_observe_types`). We assume that the observation types are `[0, 1, ..., num_observe_types - 1]`.

Next, we will define `observe_probs`: the probabilities of generating each observation type for each state type. `observe_probs` is a 2D list. Entry (i, j) in `observe_probs` specifies the probability of generating observation type j by state type i.

For example,

- For the umbrella environment, there are two observation types: has umbrella (type 0) and no umbrella (type 1).

  ```
  num_observe_types = 2
  observe_probs = [[0.9, 0.1], [0.2, 0.8]]
  ```

- For the robot environment, there are two observation types: observes door (type 0) and does not observe door (type 1).

  ```
  num_observe_types = 2
  observe_probs = [[0.8, 0.2], [0.1, 0.9]]
  ```

Given `num_observe_types` and `observe_probs`, you need to implement `create_observation_matrix` to create `observe_matrix` using `observe_probs`.

The `observe_matrix` is a 2D numpy array. Entry (i, j) in the array specifies the probability of generating observation type j in state i (not state type i). All the values in each row should sum to 1 since they represent a distribution of observation probabilities.

2.2.3 Modeling transition probabilities

State transitions occur for different reasons in different environments. In the umbrella environment, the state transitions follow a stochastic process that the agent cannot control. In the robot environment, the robot's action at each time step causes a state transition to occur.

The `Environment` object's attribute `transition_matrices` defines the transition probabilities. To accommodate the two types of environments, we will create `transition_matrices` in two different ways.

Creating `transition_matrices` *for the umbrella environment*

In the umbrella environment, state transitions occur based on a stochastic process. We initialize the environment with `action_effects = None` and a valid `transition_matrices` attribute.

`transition_matrices` is a 3D numpy array. The entry (i, j, k) represents the probability of transitioning from state j to state k given action i. Since the umbrella environment has no actions, we will assume that there is a single dummy action. Thus, `transition_matrices` is a list containing a single 2D numpy array in it. See an example below.

```
transition_matrices = [[[0.7, 0.3], [0.3, 0.7]]]
```

Creating `transition_matrices` *for the robot environment*

In the robot environment, different actions cause different state transitions. We initialize the environment with a valid `action_effects` attribute and `transition_matrices = None`. You need to implement `create_transition_matrices` to create `transition_matrices` using `action_effects`.

`action_effects` is a list of dictionaries. The i-th dictionary represents the state transition probabilities as a result of executing the i-th action. See an example below.

```
action_effects[0] = {0: 0.1, -1: 0.8, -2: 0.1}
action_effects[1] = {0: 0.1, 1: 0.8, 2: 0.1}
action_effects[2] = {0: 1.0}
```

The keys in the i-th dictionary are relative state offsets after taking action i. The state offsets are relative to the current state; they may be positive, negative or zero. For example, a state offset of -2 means that the target state is two locations to the left, and a state offset of 4 means that the target state is four locations to the right. Since the environment is circular, there are many ways of specifying the relative state offsets. For examples, for the moving left action, two possible set of keys are $[0, -1, -2]$ and $[0, 14, 15]$. Make sure that your code handles all the cases (by computing modulo the number of states).

You need to implement `create_transition_matrix` to create `transition_matrices` by using `action_effects`. The `transition_matrix` is a 3D numpy array. The entry (i, j, k) in the array specifies the probability of transitioning from state j to state k by taking the i-th action. All the values in the (i, j)-th list should sum to 1 since they represent a distribution of transition probabilities.

2.2.4 The Umbrella Environment

See below for an example of defining the umbrella environment in the program.

```
ue_num_state_types = 2
ue_state_types = [0, 1]
ue_num_observe_types = 2
ue_observe_probs = [[0.9, 0.1], [0.2, 0.8]]
ue_transition_matrices = np.array([[[0.7, 0.3], [0.3, 0.7]]])

ue = Environment(ue_num_state_types, ue_state_types, \
                 ue_num_observe_types, ue_observe_probs, \
                 None, ue_transition_matrices)
ue_init_probs = [1. / ue.num_states] * ue.num_states
```