Is this an MDP? The case of "pick and place" (to be discussed after practice quiz, 2nd week)

Csaba Szepesvári

Monday 6th February, 2023

Abstract

Some controlled stochastic processes are MDPs, some are not. We describe the pick and place task, and then question whether various observation structures lead to an MDP at the observation level.

1 Preliminaries

This will work as follows: We will describe an MDP as usual by just following the definition of what MDPs are. Then we'll derive a POMDP (Partially Observable MDP) from it, by changing what information is observable. The question then would be whether the observations themselves give rise to an MDP?

2 Pick and place

This is the simplest version of the problem: There is a robot, that has a position (start or goal), and a gripper (hand) which can be either open or closed. The actions are opening and closing the gripper, and moving the robot from one position to the other. Objects appear appear at the start position, which, if grabbed by the robot and moved over to the goal position and dropped there, will result in a unit reward. Depending on when and how the objects appear at the start position, we will have different "worlds".

For the formal description, we use an intuitive formalism where states are treated as variables in a programming language (e.g. `Python`) and have member variables that describe the various components of the states. Imagine using a named tuple in `Python`. If `s` is a variable encoding the state, the member variables are `s.open, s.atstart, s.full`, and `s.objatstart`, which all take on the values of `1` (for true) or `0` (for false). Thus, the size of the state space is $2^4 = 16$. The possible actions are `open, close, tostart`, and `tomove`. The transition dynamics and the reward are given by the following code:

```
from dataclasses import dataclass
from enum import Enum
import random

@dataclass
class State:
```

```
    open: int = 0
    atstart: int = 0
    full: int = 0
    objatstart: int = 0

Action = Enum('Action','OPEN CLOSE TOSTART TOEND')

def transition_w1(s,a):
    reward = 1 if a==Action.OPEN and s.full==1 and s.atstart==0 else 0
    snext = State()
    snext.open = 1 if a==Action.OPEN \
        else 0
    snext.atstart = 1 if a==Action.TOSTART or \
            (s.atstart==1 and a!=Action.TOEND) \
        else 0
    snext.full = 1 if \
            (s.atstart==1 and s.objatstart==1 and a==Action.CLOSE) \
           or (s.full==1 and a!=Action.OPEN) \
        else 0
    snext.objatstart = 1
    return snext,reward

def transition_w2(s,a):
    snext,reward = transition_w1(s,a)
    # objatstart computation..

    # the robot just picked up the object
    if a==Action.CLOSE and s.objatstart==1 and \
       s.atstart==1 and s.full==0:
        # replenish randomly
        snext.objatstart = random.randint(0, 1)
    # dropping object at start
    elif a==Action.OPEN and s.full==1 and s.atstart==1:
        snext.objatstart = 1
    else:
        snext.objatstart = s.objatstart
    return snext,reward

if __name__=="__main__":
    s = State()
    a = Action.OPEN
    print(transition_w1(s,a))
```

```
    print(transition_w2(s,a))
```

To be specific, there are two versions of the transition function: In "World 1" there is always an object at the start position. In "World 2", this is not true: If an object is picked up at the start position, a new object appears there with probability 0.5. Otherwise, an object appears there in the next time step if there was already an object there, or if an object is dropped there.

Now, consider a version of the problem, where the observation available on the state is just the open-closed status and the position of the gripper. In another version, the observation available on the state is as above (that is, the open-closed status and the position), and additionally the status of whether the gripper is full. The following code shows these two versions:

```
from pp import *

@dataclass
class State1:
    open: int = 0
    atstart: int = 0

    def __init__(self, s):
        self.open = s.open
        self.atstart = s.atstart

@dataclass
class State2:
    open: int = 0
    atstart: int = 0
    full: int = 0

    def __init__(self, s):
        self.open = s.open
        self.atstart = s.atstart
        self.full = s.full


if __name__=="__main__":
    s = State()
    print(s)
    print(State1(s))
    print(State2(s))
```

The question is whether the controlled stochastic process where the agent has access only to the restricted states (either `State1(s)` or `State2(s)`) and not the full state `s`, gives rise to a Markov Decision Process? We thus have four different cases to consider:

1. Using `State1` and World 1: Yes/No/It depends?
2. Using `State1` and World 2: Yes/No/It depends?
3. Using `State2` and World 1: Yes/No/It depends?
4. Using `State2` and World 2: Yes/No/It depends?

And if the answer is yes, what is the transition probability kernel corresponding to the restricted states?

3 Solution

We now discuss the solutions to the four problems described above.

1. **`State1`, World 1**: Recall this observation is missing whether the hand is full, and whether there is an object at the start position. The answer in this case is **no**.

   This is because the "reward is not Markov". Indeed, consider the case when the robot executes the following sequence of actions: `TOEND, OPEN, TOSTART, CLOSE, TOEND, OPEN`. This will lead to a reward of *one*: First, we empty the hand, then pick up an object and then drop it. Further, the last observation incurred before taking the last action is so that `s.open=0, s.atstart=0`. Now, consider the longer sequence of actions: `TOEND, OPEN, TOSTART, CLOSE, TOEND, OPEN, CLOSE, OPEN`. Again, the last observation incurred is `s.open=0, s.atstart=0`, the same as before. However, in this case, deterministically, the reward received is *zero*.

   Hence, the information about the history of interaction changes the distribution of the reward, and as such, the controlled random process does not give rise to an MDP.[1]

2. **`State1`, World 2**: This is a bit more challenging, since moving the gripper to the start position will not necessarily result in the robot picking up an object. Nevertheless, the answer stays the same and almost the same example works (just prepend both action sequences with the first one, `TOEND, OPEN, TOSTART, CLOSE, TOEND, OPEN`). The difference is that with the new first action sequence an object is picked up at the start position with probability 0.5, which can be shown using a case analysis (omitted). This in turn gives a reward with probability 0.5. Now, with the new longer action sequence, zero reward is received at the end with probability one. The last observations are the same in the two cases. Hence, the history "matters" and we do not have an MDP and the answer is **no** again.

3. **`State2`, World 1**: This is the simple world and the case when the observation includes whether the hand is full, but is missing whether there is an object at the start position.

   This is a funny case: apart from the very first time step ($t = 0$), we are guaranteed that there is always an object at the start position. Hence, if we are guaranteed that even at this time step there will be an object at the start position, the information whether there is an object at the start position is redundant and we could leave `s.objatstart==1` out from the definition of `transition_w1` and we would still get the same outcomes for any policy. However, we cannot know this, hence, we need

to investigate the case when this is not the case. Thus, assume that initially `s.objatstart==0`. Furthermore, assume that initially the robot is at the start position with its gripper open and with no object in it. The initial observation is thus `s` so that `s.open=0, s.atstart=1, s.full=0`. Take the action `CLOSE`. Then, for the next state `snext`, we will have `snext.full==0`. Now, consider the action sequence `CLOSE, CLOSE`. The observation before the last close action is still `s.open=0, s.atstart=1, s.full=0`. Yet, in this case, because we are in World 1, for the next state `snext`, we will have `snext.full==1`. Thus, in one case we get that `snext.full==0` and in the other case we get that `snext.full==1`, while keeping the last observation and last action the same. Hence, this is not an MDP and the correct answer is **it depends**.

(Note that, according to the `transition_w1` function given before, the gripper will pick an object, i.e. `s.full==1`, if the gripper is in the start position, there is an object in the start position, and the action is to close the gripper. This happens irrespective of whether the gripper was open or not in the previous timestep. Even though this transition dynamics is counter-intuitive, we use it nevertheless due to its simplicity.)

4. **`State2`, World 2**: The same argument as for World 1 works except that in this case, this is not an MDP unless the initial state is randomly chosen so that there is an object at the start position with probability 0.5. Yet again, the correct answer is **it depends**.

4 Comments

A few comments are in order:

1. Some POMDPs are MDPs at the observation level.

   This was the case using `State2` as the observation in both worlds under some special conditions. In World 1, we needed that the initial state was such that, either the robot was not at the start position, or its hand was full, or there was an object at the start position. In World 2, it was sufficient if the probability of having an object at the start position was 0.5 for the first time step. One expects that there will be examples when we have an MDP at the observation level, but solving this MDP will result in worse performance than solving the POMDP. Is this true? These two examples did not show this.

2. Reversely, one does not need an MDP to get good performance.

   In World 1, the optimal policy executes a fixed action sequence. Indeed, this world is deterministic. An object always appears at the start, the hand can be moved to the start position, the object can be picked up, delivered to the goal position, and dropped off there for a unit reward. This works even in the absence of any information. The only possibly extra (redundant) step is moving the robot hand to the start position. For long horizons, the opportunity cost of such redundant steps quickly washes out.

   If we modified the environment so that the success rate of holding or grabbing the object is only 0.5, performance in the case of lack of the information regarding whether the hand is full would

quickly deteriorate. The same holds for World 2, and `State1`, which lacks this information. In this modified environment, sensing is crucial to maintain performance.

3. One can turn a problem into a Markov one by removing reward.

   If reward was removed from World 1, even `State1` would give rise to an MDP. But this is of little comfort, as then all the policies would look equally good. This just underlines the importance of including the reward in the description and talks about the dangers of solely worrying about the state-dynamics with no concern for the reward.

4. One can turn a problem into a Markov one by changing the meaning of actions.

   Indeed, if in the case of `State2`, World 1, we redefined the actions to be `TOSTART, CLOSE` (call this compound action `PICK`) and `TOEND, OPEN` (call this compound action `DROP`), the problem would become Markov again. After `PICK`, the robot is at the start position and has an object, with the gripper closed. This is true regardless of the state (and hence, regardless of the last and previous observations, or actions). Similarly, after `DROP`, the robot is at the end position, does not have the object, the gripper is open. Furthermore, if the robot had the object, a reward is received. Again, this is true regardless of the state (and hence, regardless of the last and previous observations, or actions). The indeterminacy that was the result from not knowing whether at time $t = 0$ there is an object at the start position is removed. Moreover, with the above action sequence, the only relevant state variable to be kept around is whether the object is in the hand of the robot. With this observation, the resulting problem is Markov. Thus, the state space is reduced. Furthermore, there is almost no loss in performance.

   If we were in World 2, how would we need to modify `PICK` to guarantee that there is (almost) no loss of performance? You may assume that `State2` is available.

5. The full history is always a Markov state, but occasionally one can get away with remembering less.

   Consider the version of World 1 when the initial state is guaranteed to have an object at the start position. In this case, using `State1` does not lead to an MDP because it does not allow us to predict the reward regardless of the history. But if we let the robot remember what it did before, the reward becomes predictable! If the robot was moved to the start position, and object was picked up there, and was not yet dropped and we are opening the hand, a reward will be incurred! This looks complicated and a lot of things seem to matter. A simpler approach is to construct an internal memory and used that for prediction. The internal memory in this case is to just encode whether the hand is full! The `amend_state` function from the code below could be used to create this extra bit **inside** the agent. This function would be called once an action is chosen to just compute (predict) whether the hand will be full in the next time step. When the next time step information arrives, this can then be amended by the pre-computed "is the hand full?" information.

```
from pp import *
from pp_obs import *

def amend_state(s: State1, a:Action):
    '''
```

```
    Takes an observation of type State1 and an action, and returns
    whether the object is in the hand in the next time step
    '''
    return 1 if a==Action.CLOSE and s.atstart==1 else 0


if __name__=="__main__":
    s = State(1,1,0,1) # open atstart full objatstart
    a = Action.CLOSE
    snext,r = transition_w1(s,a)
    print("Hand full?", snext.full)
    print("Hand predicted to be full?", amend_state(State1(s), a))
```

Notes

1. To reason formally, one can use the last proposition from this document.