ROB311 - TD2

BRAMBILLA Davide Luigi - GOMES DA SILVA Rafael

23 September 2019

In order to execute the program the command to run is *python TD2.py*.
The code is available on the git lab repository:
https://gitlab.data-ensta.fr/gomesdasilva/rob311.git

Introduction

The Markov decision is a stochastic model where the possible actions that could be taken are random, that is, the current decision takes into account what has already been done in the action before. This kind of method is used in different areas in order to optimise processes: in the area of artificial intelligence it is very often used for reinforcement learning problems, with the help of dynamic programming.

The Markov decision process is composed of 4 main elements (S, A, T_a, R_a) where:

- **S** are all the possibles states, modeled according to the given context;
- **A** are the possible actions that could be taken in order to change from one state to another;
- $\boldsymbol{T_a} = T(S, a, S')$ are the transition functions that give the probability of changing to the state S' from the state S, taking the action a;
- $\boldsymbol{R_a}$ is the reward given when the taken decision leads to a specific state.

Policy π(S), Utility function V(S) and Discounted Factor γ

A *policy* $\pi(S)$ is a function that relates for each states the possible actions that could be taken, that means, $\pi(S) = S \mapsto a$.

The *utility function* of a state shows the reward that a state could get when choosing an action when moving to state S' plus the future utility of the possible states after S'. It is denoted as $V^*(S)$ and the equation is shown in (1).

$$V(S) = R(S) + max_a \gamma \sum_{S'} T(S, a, S') V^*(S') \tag{1}$$

The *optimal utility function* is instead the limit of the *utility function* until the convergence of the optimal value has been reached. It is defined as $V^*(S) = \lim_{t \to \infty} V(s)$.

The *discounted factor* $\gamma \in [0, 1]$ that evaluates the importance of a future event in the computation of the actual utility function. The greater the value of γ, more the future actions will be considered. In other words, we could say that for a value of $\gamma = 0$, we consider only the actual state when choosing an action, while for a $\gamma > 0$ we start to choose an action that will be better for the process considering all the possible states.

The case proposed

The Figure 1 shows the Markovian process given in the TD.

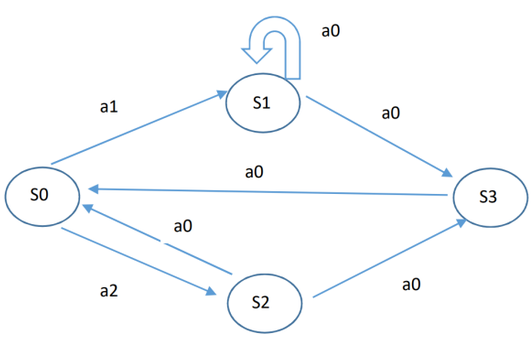

Figure 1: The Markovian process treated in the TD

Q1 - Enumerate all the possible policies

The graph in the figure 1 represents a Markovian process where the future state depends only on the current state. In particular in each state it is possible to take one among the different possible transitions and to move towards a new state. The choice among the available transitions is named *policy*. In the case analysed in the TP we have **4** possible states: *1*, *2*, *3* and *4* and **3** transitions: a_0, a_1 and a_2. Each of the action isn't available in every state: in example in the state *0* we have the possibility to go to state *2* with a_1 or to go to state *2* with a_2. We will report in a table all the possible *policies* that could be taken. The number of real policies is *5* in our model since in the state *1* and in the state *2* we have only one possible transition to choose. In the following table we will report all the possible policies also with the state after that the transition has been accomplished.

State(i)	Transition	State(i+1)
0	a_1	1
0	a_2	2
1	a_0	1
1	a_0	3
2	a_0	3
2	a_0	0
3	a_0	1

We said that the number of policies is **5** but we have reported 7 values in the table: this come from the fact the couples *state 1 - transaction* a_0 and *state 2 - transaction* a_0 constitute only one *policy* but they might have two different results.

Q2 - Equations for each optimal value function

The optimal value function in a *Markov Decision Process* is defined as the limit towards infinite of the maximum value over all the possible utilities.

That means, in order to make a choice between the different policies we have to verify which policy permits to obtain the maximum value and so the biggest reward.

Based on (1) and on the transition functions $T(S, a, S')$ given on TP subject and each reward $R(S)$, it is possible to write the equation for each optimal value function for each state as shown below: we are going to report the optimal policy for each state:

$$V^*(S_0) = R(S_0) + max_a \gamma \left[T(S_0, a_1, S_1')V^*(S), T(S_0, a_2, S_2')V^*(S)\right] = \gamma\, max_a \left[V^*(S_1), V^*(S_2)\right] \tag{2}$$

$$V^*(S_1) = R(S_1) + \gamma \left[T(S_1, a_0, S')V^*(S)\right] = \gamma\ \ \left[(1-x)V^*(S_1) + xV^*(S_3)\right] \tag{3}$$

$$V^*(S_2) = R(S_2) + \gamma \left[T(S_2, a_0, S')V^*(S)\right] = 1 + \gamma\ \left[(1-y)V^*(S_0) + yV^*(S_3)\right] \tag{4}$$

$$V^*(S_3) = R(S_3) + \gamma \left[T(S_3, a_0, S')V^*(S)\right] = 10 + \gamma\ \left[V^*(S_0)\right] \tag{5}$$

It is possible to notice here that the 4 equations are reported for the policies: one directly linked to each state. It is evident also that for the state zero the policy is not fixed and can vary between a_1 and a_2. Finally we found the same number of policies reported in Q1.

Q3 - x for $\pi^*(S_0) = a_2$

Is there a value for x, that for all $\gamma \in [0, 1)$*, and* $y \in [0, 1]$ *, so* $\pi^*(S_0) = a_2$ *?*

Yes. Knowing that $\pi^*(s) = argmax_a \sum_{S'} T(S, a, S')V^*(S')$, it is possible to write the optimal policy of S_0, $\pi^*(S_0)$, as shown in (6):

$$\pi^*(S_0) = max\{V^*(S_1), V^*(S_2)\} \tag{6}$$

Thus, according to (6), $\pi^*(S_0) = a_2$ is true only if (7) is true.

$$V^*(S_2) > V^*(S_1) \tag{7}$$

- **Analyse 1: $\gamma = 0$**

  If $\gamma = 0$, we have from (3) and (4) that $V^*(S_1) = 0$, and $V^*(S_2) = 1$, that means that the equation (7) is true and x can take any value in this case.

- **Analyse 2: $0 < \gamma < 1$**

  Having as hypothesis that $x = 0$ to start, we have from (3) that: $V^*(S_1) = \gamma\ V^*(S_1)$.
  Since in this case $\gamma \neq 0$, in order to verify the previous equation the only possibility is to have $V^*(S_1) = 0$.
  On the other hand, from (4) we will always have that: $V^*(S_2) \geq 1$. This is given by the fact that γ and y are defined as positive and the utilities are all positives since all the initial rewards associated to the possible state are positive. In other words we will have that $\gamma\ [(1-y)V^*(S_0) + yV^*(S_3)] \geq 0$.
  In the end, since $V^*(S_1) = 0$ and $V^*(S_2) \geq 1$, (7) is also true for $0 < \gamma < 1$.

In this way we can conclude that for $x = 0$, for $\gamma \in [0, 1)$, and $y \in [0, 1]$ (y does not influence in this case), it is possible to find an optimal policy such that $\pi^*(s_0) = a_2$. This result can also be verified using the Algorithm 1 implemented later, as shown in Figure 2.

```
Error value: 0.000100
Number of iterations: 51

    |    S0    |    S1    |    S2    |    S3    |
-------------------------------------------------
V*  | 13.9110 | 0.0000  | 15.4569 | 22.5199 |
-------------------------------------------------
pi* |    a2    |    a0    |    a0    |    a0    |
```

Figure 2: Result of the Algorithm 1 for $x = 0$

Q4 - y for $\pi^*(S_0) = a_1$

Is there a value for y, that for all $x > 0$, and $y \in [0, 1]$, $\pi^*(S_0) = a_1$?

No. Thus, according to (6), $\pi^*(S_0) = a_1$ is true only if (8) is true.

$$V^*(S_1) > V^*(S_2) \tag{8}$$

- **Analyse 1: $\gamma = 0$**

  If $\gamma = 0$, we have from (3) that $V^*(S_1) = 0$, and from (4) $V^*(S_2) = 1$, that reveal that (8) is false since $V^*(S_2) > V^*(S_1)$.

- **Analyse 2: $0 < \gamma < 1$**

  Since in this case $\gamma \neq 0$, from (4) we can see that $V^*(S_2) \geq 1$, which yields, from (8), that $V^*(S_1) \geq 1$. Thus, we can reorganize (3) in this way:

$$V^*(S_1) = \gamma\,[(1-x)V^*(S_1) + xV^*(S_3)]$$

$$(1 - \gamma\,(1-x))V^*(S_1) = \gamma\ x\ V^*(S_3)$$

  and obtain (9):

$$V^*(S_1) = \frac{\gamma\ x\ V^*(S_3)}{1 - \gamma\ (1-x)} \tag{9}$$

  that must be greater than 1.

  However, from (9) it is possible to choose a value of $\gamma \in [0, 1)$, in example a γ very small, that will lead to:

$$\frac{\gamma\ x\ V^*(S_3)}{1 - \gamma\ (1-x)} < 1$$

  We can also notice that y doesn't appear either in the previous formula.

Thus, we can conclude that there is no y, no matter what value of x we take (x does not influence in this case), so that $\pi^*(S_0) = a_1$.

Q5 - Calculate the π^* and V^* for all states

In order to calculate the π^* and V^* for all states, the *value iteration algorithm* (Algorithm 1) was implemented in *python*, having as a parameter to stop the iterations the *RMS* error (10) between the utility function of the actual iteration and the utility function of the last iteration. This error is limited by a small positive number (ϵ), which in this case is defined as $\epsilon = 10^{-4}$.

$$RMS = \frac{1}{|S|}\sqrt{\sum_{i=1}^{|S|}(V(i) - V'(i))^2} \tag{10}$$

Algorithm 1 The Value Iteration Algorithm. Source: [**Slides from the class**]

$T \leftarrow$ Transition model
$R \leftarrow$ Reward function on states
$V \leftarrow$ Utility function initially identical to R
$V' \leftarrow$ Utility function initially identical to R
$\epsilon \leftarrow$ Small positive number (error)

while $RMS < \epsilon$ **do**
  for each state i **do**
    $V'(i) = R(i) + \gamma \max_a \sum_{S'} T(S, a, S')V^*(S')$
  end for
end while
return V

Listing 1: Implementation of the Algorithm 1

```
#Definition of the parameters
x = 0.25
y = 0.25
#Discounted factor
gamma = 0.9
#Maximum error expected
epsilon = 1e-4
#Transition Matrices
T1 = [[0, 0, 0, 0],[0, 1-x, 0, x],[1-y, 0, 0, y],[1, 0, 0, 0]]
T2 = [[0, 1, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]
T3 = [[0, 0, 1, 0],[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]
#Rewards
R = [0, 0, 1, 10]
V = [R[0], R[1], R[2], R[3]]
V_prime = [R[0], R[1], R[2], R[3]]
action = [0, 0, 0, 0]
iterations = 0

while (1):
    for i in range (len(V)):
            summ = np.matmul(T1[i][:], V_prime), np.matmul(T2[i][:], V_prime),
                np.matmul(T3[i][:], V_prime)
            max_summ = np.amax(summ)

            #Saves the value of pi
            action[i] = summ.index(max(summ))

            #Saves the latest value of V' to compare the error
            V[i] = V_prime[i]

            #Updates the Value of V'
            V_prime[i] = R[i] + gamma*max_summ
    iterations += 1
    #Stop criterion
    if (math.sqrt(np.sum((np.array(V) - np.array(V_prime))**2))/4 < epsilon):
        V = V_prime
        break
```

Number of iterations until reaching the desired error: 68.

```
Error value: 0.000100
Number of iterations: 68

    |    S0   |    S1   |    S2   |    S3   |
-------------------------------------------
V*  | 14.1841 | 15.7603 | 15.6965 | 22.7657 |
-------------------------------------------
pi* |    a1   |    a0   |    a0   |    a0   |
```

Figure 3: Result of the Algorithm 1 for $x = y = 0.25$ and $\gamma = 0.9$