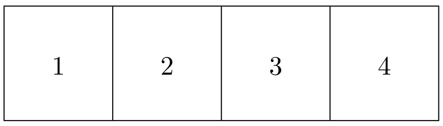
COMP3702 MDP Tutorial

Dimitri Klimenko

26th September 2014

1 Q3 from Tutorial 6/7

Figure 1: The environment



A robot navigates in a very simple environment, as shown in Figure 1. Suppose the robot starts from cell 1 and tries to reach the goal state, which is cell 4. At each time step, the robot can attempt to move one cell to the left, or one cell to the right. However, due to control error, every time the robot moves, it has an 80% chance of ending up in its intended cell and also a 20% chance of staying where it is. If the robot tries to move left from cell 1, it will simply stay in the same place. Each attempted move costs the robot 1 utilon, while reaching the goal gives a reward of 10.

Please model the above navigation problem as an MDP with an infinite horizon and discount factor $\gamma = 0.95$.

2 MDP model

Our MDP model is $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ is the action space.
- T is the transition function.
- R is the reward function.
- $\gamma = 0.95$ is the discount factor.

2.1 State space

Since the environment is static, we only need one state variable to fully describe the problem state—the position of the robot. As such, the state space for this problem can be represented as

$$\mathcal{S} = \{s_1, s_2, s_3, s_4\}$$

As we've said that s_4 is a goal state, this implies that the problem terminates once s_4 is reached. There are multiple ways to do this, but for this problem we will represent this by making s_4 a *terminal state*; after the robot reaches this state it will no longer be able to take any actions. For the purposes of an MDP model, this will be equivalent to stating that all actions will cause it to stay in s_4 and give zero reward. Similarly, since we know that the agent starts in s_0, we can call this state the *initial state*—this would be the root of the tree if we were using Monte-Carlo Tree Search to solve this problem.

2.2 Action space

The robot can only go left or right, so the action space for the problem is simply

$$\mathcal{A} = \{\mathit{Left}, \mathit{Right}\}$$

2.3 Transition function

The transition function $T(s, a, s')$ describes the probabilities for the possible next states s', conditional on the robot being in state s and taking action a, i.e.

$$T(s, a, s') = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$$

where S_t is the state at time t, and A_t is the action at time t. For this particular MDP model, we have

$$T(s_4, \mathit{Left}, s_4) = T(s_4, \mathit{Right}, s_4) = 1$$

since we've stated that the game ends upon reaching s_4, so taking either action from s_4 means that you stay there. We also have

$$T(s_1, \mathit{Left}, s_1) = 1$$

since moving left from s_1 causes us to stay in s_1. Because the robot has an 80% chance of going where it intended to go, we have

$$T(s_2, \mathit{Left}, s_1) = T(s_3, \mathit{Left}, s_2) = T(s_1, \mathit{Right}, s_2) = \\ T(s_2, \mathit{Right}, s_3) = T(s_3, \mathit{Right}, s_4) = 0.8.$$

Additionally, we have

$$T(s_2, \mathit{Left}, s_2) = T(s_3, \mathit{Left}, s_3) = T(s_1, \mathit{Right}, s_1) = \\ T(s_2, \mathit{Right}, s_2) = T(s_3, \mathit{Right}, s_3) = 0.2,$$

as the robot has a 20% chance of staying in the same square when it tries to move to a different one. For all other transitions, we have

$$T(*, *, *) = 0,$$

Table 1: Transition matrix representation of T.

(a) Transition matrix P_{Left}

	s_1	s_2	s_3	s_4
s_1	1	0	0	0
s_2	0.8	0.2	0	0
s_3	0	0.8	0.2	0
s_4	0	0	0	1

(b) Transition matrix P_{Right}

	s_1	s_2	s_3	s_4
s_1	0.2	0.8	0	0
s_2	0	0.2	0.8	0
s_3	0	0	0.2	0.8
s_4	0	0	0	1

because, for example, the robot can never end up in s_3 immediately after being in s_1.

An alternative way to represent this information is with *transition matrices*, one for *Left*, and one for *Right*; those matrices can be seen in Table 1. The rows of the transition matrix represent the *current state* s, and the columns represent the *next state* s'. For example, by looking at the cell in the second row and third column in Table 1b, we can see that the probability of reaching s_3 when going *Right* from s_2 is 0.8, i.e. $T(s_2, \mathit{Right}, s_3) = 0.8$.

2.4 Reward function

Since we have decided that s_4 is a *terminal state*, we are representing this by having the robot get "stuck" in s_4 after reaching it. This means that we must set the reward for taking actions from s_4 to always be zero, or the robot could simply continue to collect rewards in s_4 forever, as we're using an infinite-horizon MDP. As such, we must instead give the robot a reward whenever it reaches s_4. We can't simply give the robot a reward for going *Right* from s_3, as sometimes the robot stays in s_3. This means that our reward function R needs to take the form

$$R(s, a, s'),$$

which represents the reward for reaching s' after taking action a from state s.

As we've said that the robot eternally recieves no rewards after s_4 is reached, this means that

$$R(s_4, \mathit{Left}, s_4) = R(s_4, \mathit{Right}, s_4) = 0.$$

The robot is given a reward of 10 reaching s_4; we can also see by inspection of the trancition matrices that the only possible way to reach s_4 is by going *Right* from s_3. Since it costs the robot 1 utilon every time it moves, we have

$$R(s_3, \mathit{Right}, s_4) = 9,$$

which incorporates the reward for reaching s_4 as well as the move cost. All other movements of the robot don't reach the goal but still cost one utilon, so finally we have

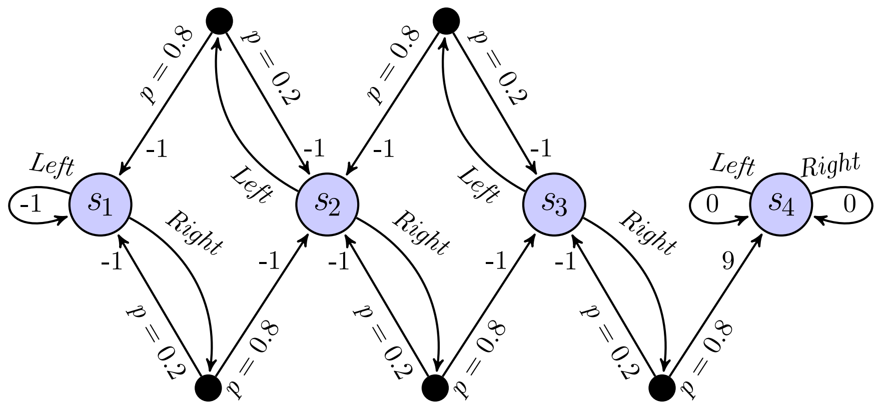
$$R(*, *, *) = -1$$

for all other transitions.

2.5 Graphical representation

This MDP can also be represented via a graph, which can be seen in Figure 2. The four blue nodes in the graph represent the various states, s_1, s_2, s_3 and

Figure 2: A graphical representation of the MDP model



s_4, while the arrows out of those nodes represent the actions (*Left* and *Right*) that can be taken from those states. The filled black nodes are "chance nodes", representing different possible outcomes due to a random event, with labels like $p = 0.2$ representing the probability $T(s, a, s')$ of different state transitions. The integer labels on the ends of the arrows represent the reward function $R(s, a, s')$, which specifies the reward for reaching state s' after taking action a from state s.

3 Policies and value functions

3.1 Policies

A *policy* $\pi : \mathcal{S} \to \mathcal{A}$ is a mapping from states to actions. This is the way we describe how the agent in an MDP will act; in the context of a game we might also call this a *strategy*.

If we associate the states with a specific order, we can also represent a policy as an ordered list of actions *tuple* that lists one action for each state. In this case, the obvious ordering is (s_1, s_2, s_3, s_4), and so an example policy is

$$\pi_L = (\textit{Left}, \textit{Left}, \textit{Left}, \textit{Left}).$$

This is clearly not a very good policy because the robot's goal is to get to the rightmost square, but it's a policy nonetheless! A policy that's clearly better is

$$\pi_R = (\textit{Right}, \textit{Right}, \textit{Right}, \textit{Right});$$

indeed, we will argue that this is one of two optimal policies in this MDP, along with

$$\pi_2 = (\textit{Right}, \textit{Right}, \textit{Right}, \textit{Left});$$

3.2 Value functions

A *value function* $V : \mathcal{S} \to \mathbb{R}$ is a function that associates each state with a *value*, which is a real number that estimates, for each state, how valuable it is for us to be in that state. For example, a relatively silly value function would be

$$V_0(\cdot) \equiv 0$$

which assumes that all states are equally worthless.

3.3 The value of a policy

For a given policy π we can calculate the value function for that policy V_π using equations that look like this:

$$V_\pi(s) = \sum_{s'} T(s, \pi(s), s')\big[R(s, \pi(s), s') + \gamma\, V_\pi(s')\big], \tag{1}$$

where s is the current state, $\pi(s)$ is the action that the policy chooses for that state, and the values s' represent the possible next states. Since there many states, this actually forms a system of equations (one for each state) that can be solved to calculate the value of a given policy.

Equation 1 simply represents a calculation of the expected value over different possibilities for the next state. This is the sum of two terms: the value of a given state is the expected value of the immediate reward,

$$\mathbb{E}\left[R(s, a, \cdot)\right] = \sum_{s'} T(s, \pi(s), s')R(s, \pi(s), s') \tag{2}$$

added to the discounted expected value of the next state,

$$\gamma \sum_{s'} T(s, \pi(s), s')\, V_\pi(s'). \tag{3}$$

If the reward was independent of the next state, we could represent our reward with a function $R(s, a)$ instead, with $R(s, a, s') \equiv R(s, a)$. Consequently, the expected reward over all possible next states would simply be

$$\mathbb{E}\left[R(s, \pi(s), \cdot)\right] \equiv R(s, \pi(s))$$

and Equation 1 would simplify to give

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s')\, V_\pi(s'). \tag{4}$$

3.3.1 An example

As an example, we can calculate the value of our not-very-good policy

$$\pi_L = (\mathit{Left}, \mathit{Left}, \mathit{Left}, \mathit{Left}).$$

For convenience, we can turn Equation 1 into a matrix form. As you may recall, the transition function can represented by *transition matrices*; this also applies when we have a fixed policy. Since π_L says we should always go left, the appropriate transition matrix is simply $P_L = P_{\mathit{Left}}$ from Table 1a, i.e.

$$P_L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.8 & 0.2 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Since moving left always has a cost of -1, we know that the expected immediate reward will always be -1, except at s_4 where the reward is always zero. Consequently, we can represent the expected reward by the vector

$$\mathbf{r} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 0 \end{pmatrix}.$$

Using this, we can transform Equation 1 to the matrix form

$$\mathbf{v}_L = \mathbf{r} + \gamma P_L \mathbf{v}_L. \tag{5}$$

I've switched to the notation $\mathbf{v}_L$ for the value function to emphasize that it is a *vector* in this equation. Some basic algebra gives

$$(I - \gamma P_L)\mathbf{v}_L = \mathbf{r}$$

which is simply a system of linear equations in matrix form! If we substitute the numbers in, we have

$$\left[\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - 0.95 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.8 & 0.2 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right] \mathbf{v}_L = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 0.05 & 0 & 0 & 0 \\ -0.76 & 0.81 & 0 & 0 \\ 0 & -0.76 & 0.81 & 0 \\ 0 & 0 & 0 & 0.05 \end{pmatrix} \begin{pmatrix} V_L(s_1) \\ V_L(s_2) \\ V_L(s_3) \\ V_L(s_4) \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 0 \end{pmatrix}$$

If we solve the equations, we get

$$V_L(s_4) = 0,$$

and

$$V_L(s_1) = V_L(s_2) = V_L(s_3) = -20.$$

This would mean that, from our initial state of s_1, the expected value of our policy π_L is -20. Clearly not a very good policy…

3.4 The one-step best policy for a value function

Just as we calculated the value function for a given policy, we can start with a value function and work out what the best policy would be if those values were actually correct. To make this simpler, we introduce the notion of a *Q-value*, which represents the estimated value of taking an action a from state s. The one-step Q-value for a given value function V is, similarly to Equation 1,

$$Q_V(s, a) = T(s, a, s')\big[R(s, a, s') + \gamma V(s')\big], \tag{6}$$

Similarly, we can work out a *one-step* best policy for a given value function by looking one step ahead for every state, as per the above equation, and choosing the action that has the best one-step value for each state. Consequently, the one-step best policy can be defined like this:

$$\pi(s) = \arg\max_a Q_V(s,a) = \arg\max_a \sum_{s'} T(s,a,s')\Big[R(s,a,s') + \gamma V_\pi(s')\Big]. \quad (7)$$

which simply states that policy always takes the action that has the largest expected value, when we look one step ahead with our given value function V.

3.4.1 An example

As an example, let's start with a value function V_σ (σ for silly), which we will represent by the vector

$$\mathbf{v}_\sigma = (1,0,0,0).$$

In order to find the one-step best policy, we first calculate the values of all the actions from looking one step ahead, which gives us the following:

$$\begin{aligned}
Q_\sigma(s_1, \mathit{Left}) &= -1 + 1 && = 0\\
Q_\sigma(s_1, \mathit{Right}) &= -1 + 0.95(0.8\cdot 0 + 0.2\cdot 1) && = -0.81\\
Q_\sigma(s_2, \mathit{Left}) &= -1 + 0.95(0.8\cdot 1 + 0.2\cdot 0) && = -0.24\\
Q_\sigma(s_2, \mathit{Right}) &= -1 + 0 && = -1\\
Q_\sigma(s_3, \mathit{Left}) &= -1 + 0 && = -1\\
Q_\sigma(s_3, \mathit{Right}) &= 0.8\cdot(9 + 0.95 * 0) && =\\
Q_\sigma(s_4, \mathit{Left}) &= 0 + 0 && = 0\\
Q_\sigma(s_4, \mathit{Right}) &= 0 + 0 && = 0
\end{aligned}$$

3.5 Optimal policies and optimal value functions

4 Value iteration

5 Policy iteration