6.036 Spring 2018 : Week 8

March 28, 2018

9 Learning to act

Consider a simple robot navigation problem illustrated in Figure 1 (left). The robot is located in the top left corner and wishes to reach the goal at the bottom right, avoiding the shaded obstacles. If we assume that the robot is aware of the layout (has a map) and that it can accurately sense its position within the map at all times, then it could calculate an ideal path to reach the goal, for example, along the smooth curve in the figure. Of course, it might not be able to actually follow this ideal path if the actions available to it (movement) involve noise and do not deterministically take it where it might want to go. In other words, the robot needs to plan under uncertainty in general. Our goal is to help the robot plan under varying assumptions about what it knows, senses, and how it can act.

To make the problem a bit easier to discuss and solve we will approximate the continuous world as a grid-world shown in Figure 1 (right). The possible locations are now simply the little squares (each square represents a single location) and the robot moves by hopping from one square to another. The path that it might follow in this grid-world is highlighted by the little arrows that show, in each square, the direction (which nearby square) it will try to move into next. We will limit the robot actions such that it can only move up, down, left, or right, and only to an adjacent free square. We will discuss later what happens if it takes an action that bumps it against the wall or the shaded obstacle. If the robot knows the grid-world, and can sense its location, i.e., knows the identity (symbol) of the square it is in, it can again calculate the ideal path to follow. But, similarly to the continuous case, if it's not guaranteed to end up where it wants to move, it has to plan probabilistically.

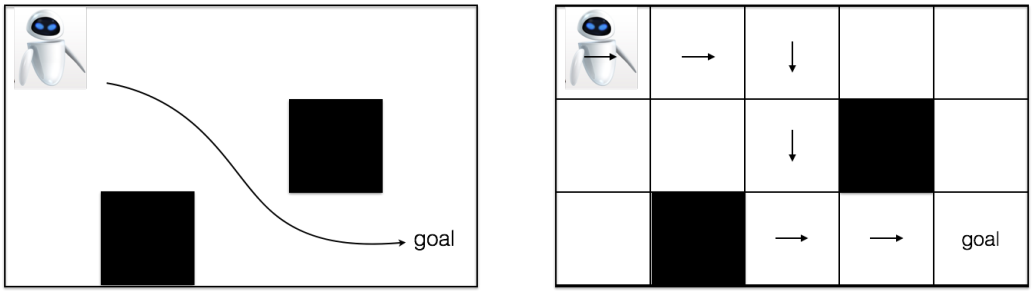

Figure 1: A robot navigation task in the original continuous space (left) and its grid-world approximation (right).

9.1 Markov processes

We will call the grid locations *states* and denote each location by variable s which is simply an integer valued and identifies the location in the grid. We assume that the world is Markov in the sense that what happens next only depends on the current state, not how the robot ended up there. How realistic the Markov assumption is depends on whether state s captures all the relevant information. For example, the robot may have a fuel tank, and moving around consumes fuel. So, if it has been moving around for a while it might end up in state s with low fuel, and this would impact how it can act from there on above and beyond just the location. We would have to include the amount of fuel in the description of the state together with the location in order for the robot's world to be Markov in this scenario.

Let's start by considering a pre-programmed robot with noisy movements. Assuming the world is Markov, we can capture how it moves around by defining *transition probabilities* $T(s, s')$, where s is the current state (location), and s' is a possible next state. Since the robot must end up somewhere at the next step, clearly $\sum_{s' \in S} T(s, s') = 1$, where S is the set of states (set of possible grid locations). Note that T is simply a $|S| \times |S|$ matrix of probabilities where each row sums to one. In other words, each state $s \in S$ has an associated distribution over possible next states after one time step. For example, if the robot moves uniformly at random to an adjacent location, the transition probabilities would be as shown (partially) in Figure 2

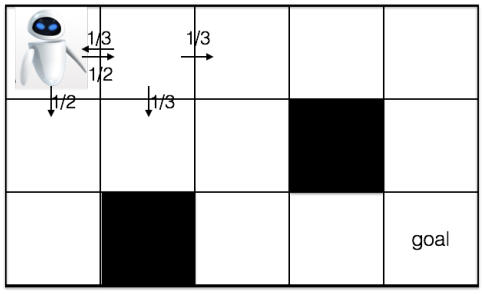

Figure 2: Example transition probabilities in the grid world

If we know the transition probability matrix T, we can calculate where the robot might be after one step, two steps, and so on. If the initial state is s_0, e.g., the top left corner, then after one step the robot would be in state s_1 with probability $T(s_0, s_1)$. Most of these probabilities are exactly zero, and only the two adjacent states have non-zero probabilities ($T(s_0, s_1) = 1/2$ for each of these two states). After two steps, the probability that the robot is in state s_2 is given by $\sum_{s_1 \in S} T(s_0, s_1)T(s_1, s_2)$, and so on. Note that, given how we defined T and s_0, there are only six possible states that the robot could be in after two steps (can you see what they are?).

The sequence of states $s_0, s_1, s_2, \ldots$, generated by the randomly moving robot is a *Markov process*. Since the movements are at each step specified by the same transition probability matrix T, the *Markov model* or *Markov chain* that governs the process is *homogeneous*. The simple Markov process here is also *ergodic* in the sense that there's a non-zero probability that the robot, if it starts from any state s_0, ends up in any other state $s_n \in S$

after some fixed n steps (in our case n can be any $n \geq |S|$). Of course, note that while the exact probability values will depend on the pair (s_0, s_n), ergodicity means that all of them are non-zero.

So, let's explore this Markov model a little further and try to assess how often the robot visits the goal state s^* at the bottom right corner. To this end, we will give the robot a *reward* if it finds itself in the goal state s^*, and nothing otherwise. In other words, we can define a *reward function* $R(s^*) = 1$ and $R(s) = 0$ if $s \neq s^*$. If the robot starts from s_0 and takes a single step, then its reward is simply $V_1(s_0) = R(s_0)$. If it can take two steps, then the *expected reward* (average reward across many trials starting from s_0) would be

$$V_2(s_0) = E\{R(s_0) + R(s_1)|s_0\} = R(s_0) + \sum_{s_1 \in S} T(s_0, s_1)R(s_1) \tag{1}$$

assuming that the reward itself is not a random variable, only the state s_1 that we end up in after one step. We can clearly calculate $V_2(s_0)$ for all the possible starting locations $s_0 \in S$, not only for the top left corner in our example. If we do this, which states in our grid world would have non-zero expected reward after one step? There are only three such states. $V_2(s^*) = 1$, and $V_2(s) = 1/2$ for the two states adjacent to s^*. What about after three steps, i.e., what is the value of $V_3(s_0)$? We can proceed similarly,

$$\begin{aligned}
V_3(s_0) &= E\{R(s_0) + R(s_1) + R(s_2)|s_0\} && (2)\\
&= R(s_0) + \sum_{s_1 \in S} T(s_0, s_1)R(s_1) + \sum_{s_1 \in S}\sum_{s_2 \in S} T(s_0, s_1)T(s_1, s_2)R(s_2) && (3)\\
&= R(s_0) + \sum_{s_1 \in S} T(s_0, s_1)\left[R(s_1) + \sum_{s_2 \in S} T(s_1, s_2)R(s_2)\right] && (4)\\
&= R(s_0) + \sum_{s_1 \in S} T(s_0, s_1)V_2(s_1) && (5)
\end{aligned}$$

It should be clear now that because the process is Markov, we can actually calculate $V_n(s_0)$, $s_0 \in S$, iteratively based on already computed values $V_{n-1}(s_1)$, $s_1 \in S$. If we start with $V_0(s_0) = 0$, $s_0 \in S$, then

$$V_n(s_0) = R(s_0) + \sum_{s_1 \in S} T(s_0, s_1)V_{n-1}(s_1), \tag{6}$$

for all $s_0 \in S$ and $n = 1, 2, \ldots$ however far into the future we wish to look. Given our definition of $R(s)$, the value $V_n(s_0)$ tells us the expected number of times that the process would visit s^* within the first n steps if we started from s_0. Of course, there's nothing in the above calculation that requires us to specify the reward function in this way. We would still be able to calculate the expected reward iteratively as shown above however we defined $R(s)$ provided that process is Markov.

9.2 Markov Decision Processes (MDP)

Our robot was pre-programmed so we didn't have any way of optimizing its behavior. We now turn to the problem of actually selecting which actions the robot should take. In this

scenario, the robot's actions must change the transition probabilities so that we have some way of tailoring where it goes. So, we must write $T(s, a, s')$ for the probability that the robot ends up in state s' if it starts in state s *and* takes action a. Similarly, the reward that we give for the robot may in general depend on the action (even the state s' after one step). More formally, we define a *Markov Decision Process* or MDP for short by specifying

1. The set of states $s \in S$ and actions $a \in A$
2. Action dependent transition probabilities $T(s, a, s')$, where $\sum_{s' \in S} T(s, a, s') = 1$ for all s, a.
3. Reward for starting in state s and taking action a, $R(s, a)$.

In order for us to actually optimize the robot's behavior, we have to specify the criterion that we are trying to maximize in terms of accumulated rewards. We'll consider in detail the following two scenarios:

(a) *Finite horizon MDP.* The goal is to maximize the expected accumulated sum of rewards after acting for a fixed number H steps.

(b) *Infinite horizon discounted MDP.* The goal is to continue acting (without an end) while maximizing the expected discounted reward. In this setting, reward one step into the future is discounted by a factor $\gamma \in [0, 1)$, two steps ahead by γ^2, and so on. The discounting allows us to focus on near term rewards, and control this focus by changing γ.

Now, before addressing how to actually solve these MDPs, let's adapt our running grid world example to the new definitions. We will have to specify everything that depends on actions, including what the actions are. As alluded to earlier, we only permit simple movements to adjacent squares so the set of actions is $A = \{\text{up}, \text{down}, \text{left}, \text{right}\}$. To keep the running example simple we assume that the transitions are deterministic, i.e., the robot goes where it wants to with probability one so long it aims for an available square. If not, e.g., taking an "impossible" action $a = \text{up}$ in the top left corner, the action keeps the robot where it is. So, for example, in this case $T(s_0, \text{up}, s_0) = 1$ if s_0 is the top left corner. Similarly, $T(s_0, \text{right}, s_0) = 0$ since the robot will move to the right with probability one under this action. It remains to specify what the reward function is. We will mimic the earlier example and set $R(s, a) = 0$ for all states $s \neq s^*$ and actions a aiming for an available square. Any "impossible" action results in a reward -1. Lastly, $R(s^*, a) = 1$ for any possible action $a \in A$ taken in s^* while $R(s^*, a) = -1$ for any "impossible" action similar to other states. Note that since our reward function depends on the action taken in state s, we are effectively awarding the reward for "acting in state s" rather than just ending up there. These definitions are illustrated in Figure 3.

9.2.1 Finite horizon MDPs

Our MDP is assumed to be fully known, including states, actions, transition probabilities, and rewards. The goal is to maximize the expected sum of rewards after exactly H steps. We can solve this problem via *dynamic programming* but need to be a bit careful. The

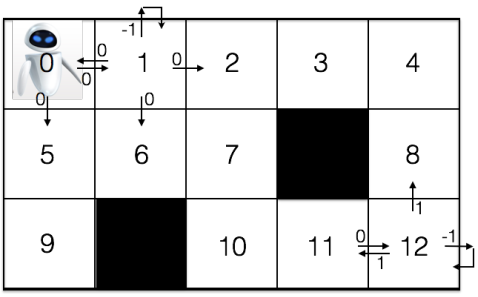

Figure 3: An example MDP in the grid world. Since the state transitions are assumed to be deterministic in this example, the numbers on the little transition arrows indicate the associated reward $R(s,a)$ (only some displayed). The states are numbered from 0 to 12.

difficulty here comes from the fact that we cannot just cue our actions on the state that we are in but also need to consider how many steps are remaining. For example, if we are already at the H^{th} step, there will be no more transitions, and we should just take an action that maximizes the local reward. In contrast, if we have H steps remaining, we should steer the robot towards areas of high reward and can afford to wait for the better reward. So, the optimal mapping from states to actions – *policy* – will also depend on the steps remaining. More formally, we denote $\pi_n(s)$ as the action we should take when there are n steps remaining, for $n = 0, \ldots, H$, and the goal is to find the policy that maximizes

$$V_{H,\Pi}(s_0) = E\{\, R(s_0, \pi_H(s_0)) + R(s_1, \pi_{H-1}(s_1)) + \ldots + R(s_{H-1}, \pi_1(s_{H-1}))|s_0 \,\} \tag{7}$$

for any starting state s_0. Note that there are H mappings $\Pi = \{\pi_n(\cdot), n = 1, \ldots, H\}$ to optimize! The good thing is that the optimal policies Π^*, once found, give the highest value uniformly across states, i.e., $V_{H,\Pi^*}(s_0) \geq V_{H,\Pi}(s_0)$, $s_0 \in S$. Finding the policies is actually a bit easier than it looks.

Let's try to evaluate the value corresponding to the optimal policy as it will give us the way to construct the optimal mappings from states to actions as well. As already argued, with no steps remaining, the value of finding ourselves (our robot) in state s is simply $V_0^*(s) = 0$. With one step remaining, $V_1^*(s) = \max_{a \in A} R(s,a)$. This is clearly the best we can do, and the optimal action is $\pi_1^*(s) = \text{argmax}_{a \in A} R(s,a)$, where $\pi_1^*(s)$ is the final policy. If we have two steps remaining, the optimal value of being in state s is to take the action that maximizes the expected sum of rewards where we also take the maximizing action in the state we end up after one step:

$$\begin{aligned} V_2^*(s) &= \max_{a \in A} E\{\, R(s,a) + \max_{a' \in A} R(s',a')|s \,\} & (8) \\ &= \max_{a \in A}\{\, R(s,a) + \sum_{s' \in S} T(s,a,s') \max_{a' \in A} R(s',a') \,\} & (9) \\ &= \max_{a \in A}\{\, R(s,a) + \sum_{s' \in S} T(s,a,s') V_1^*(s') \,\} & (10) \end{aligned}$$

The optimal policy with two steps remaining is then just the maximizing action or

$$\pi_2^*(s) = \text{argmax}_{a \in A}\{\, R(s,a) + \sum_{s' \in S} T(s,a,s') V_1^*(s') \,\} \tag{11}$$

By continuing this way we have specified the **value iteration** algorithm: for any $n = 1, 2, \ldots$

$$V_n^*(s) = \max_{a \in A}\{ R(s,a) + \sum_{s' \in S} T(s,a,s') V_{n-1}^*(s') \} \quad (12)$$

$$\pi_n^*(s) = \text{argmax}_{a \in A}\{ R(s,a) + \sum_{s' \in S} T(s,a,s') V_{n-1}^*(s') \} \quad (13)$$

which we can solve by first calculating $V_0^*(s) = 0$, $s \in S$, then $V_1^*(s)$, $s \in S$, on the basis of $V_0^*(s)$, and so on. The optimal policies at each horizon follow naturally as the maximizing actions.

Let's be concrete and see what happens in our grid world illustrated in Figure 3. If the horizon is one, we only calculate $V_1^*(s)$, for $s = 0, \ldots, 12$. What are these values? For all states other than the goal, $V_1^*(s) = 0$ (we simply avoid taking an "impossible" action resulting in a zero reward). $V_1^*(12) = 1$ since the maximum reward is acquired by aiming to move to any available square. In terms of policies, any policy that moves the robot to an available square (avoids "impossible" moves) will achieve these values. It will get a bit more interesting as the horizon increases. When $H = 1$, we will have non-zero values also for states 8 and 11 that are one step away from the goal. The associated values are $V_2^*(8) = V_2^*(11) = 1$ with maximizing actions given by $\pi^*(8) = \text{down}$ and $\pi^*(11) = \text{right}$. All other non-goal state values remain at zero with policies that simply avoid "impossible" actions. Note that we still have $V_2^*(12) = 1$ since it takes two steps to get back to the goal and act there; we can only accumulate reward 1 with horizon 2. What about $H = 3$?

9.3 Infinite horizon discounted MDPs

One way to think about the infinite horizon problem is that the discount factor tells us the probability that we will continue acting after each step. So we would halt the robot with probability $1 - \gamma$ without the possibility of accruing any more reward. So, the larger the value of γ, the longer the effective horizon we wish to optimize. In terms of finding the optimal actions to take, the problem is a bit simpler than the finite horizon case. This is because when we land on a particular state, say s, we then again have an infinite horizon discounted problem to solve from there on. As a result, the optimal action to take in state s at the beginning will be the same as the action we should take if we later land on this state. So, the optimal policy simply maps each state to a single action. Our problem is then to learn policy $\pi(s)$ so as to maximize

$$\begin{aligned}
V_\pi(s_0) &= E\{ R(s_0, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \ldots | s_0 \} & (14)\\
&= E\{ R(s_0, \pi(s_0)) + \gamma\{ R(s_1, \pi(s_1)) + \gamma R(s_2, \pi(s_2)) + \ldots \} | s_0 \} & (15)\\
&= E\{ R(s_0, \pi(s_0)) + \gamma E\{ R(s_1, \pi(s_1)) + \gamma R(s_2, \pi(s_2)) + \ldots | s_1 \} | s_0 \} & (16)\\
&= E\{ R(s_0, \pi(s_0)) + \gamma V_\pi(s_1) | s_0 \} & (17)\\
&= R(s_0, \pi(s_0)) + \gamma \sum_{s' \in S} T(s_0, \pi(s_0), s') V_\pi(s') & (18)
\end{aligned}$$

Note that after landing on s_1 we have the same problem to solve from there on. We can therefore take the expectation of this tail of discounted rewards as $E\{\cdot | s_1\}$ for each possible value of state s_1 after the first step, yielding $V_\pi(s_1)$ by definition. The above iterative

relation holds for any policy, including the optimal. Clearly, the optimal policy must select the maximizing action in s_0 and follow with this policy from there on. In terms of values then,

$$V^*(s_0) = \max_{a \in A} \left\{ R(s_0, a) + \gamma \sum_{s' \in S} T(s_0, a, s') V^*(s') \right\} \tag{19}$$

This is known as the *Bellman equation*. It relates the optimal values (same values on both sides). As before, the optimal action is just the maximizing action or

$$\pi^*(s) = \text{argmax}_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right\} \tag{20}$$

It remains to show how we can actually calculate the values $V^*(s)$. It turns out that we can just initially set these values to zero, i.e, $V_0(s) = 0$ for all $s \in S$, and iteratively update them according to

$$V_1(s) = \max_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_0(s') \right\} \tag{21}$$

and so on. As k increases, $V_k(s)$ values will converge to $V^*(s)$ values provided that $\gamma < 1$. To see this, consider the case with a single action and a single state. We denote the values just as V_k without referring to the single state, and the reward as R. In this case, the value iteration update boils down to

$$V_k = R + \gamma V_{k-1} \tag{22}$$

We also know that the optimal values satisfy $V^* = R + \gamma V^*$. By subtracting, we get

$$V_k - V^* = \gamma (V_{k-1} - V^*) \tag{23}$$

Therefore, after each value iteration step, the new values V_k are a factor γ closer to V^*.

General value iteration, summary

Value iteration is an algorithm for finding the (an) optimal policy for a given MDP. The algorithm iteratively estimates the value of each state; in turn, these values are used to compute the optimal policy. We will use the following notations:

- $V^*(s)$ – value of state s, i.e., the expected discounted reward when starting in state s and acting optimally.
- $Q^*(s, a)$ – Q value of state s and action s. It is the expected discounted reward when we start in state s, take action a, and act optimally thereafter.
- $\pi^*(s)$ –optimal policy. $\pi^*(s)$ specifies the action we should take in state s. Following policy π^* would, in expectation, result in the highest expected discounted reward.

These definitions are clearly related. We express them here in the more general case where the reward depends also on the next state.

$$V^*(s) = \max_{a \in A} Q^*(s,a) = Q^*(s, \pi^*(s)) \tag{24}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \tag{25}$$

$$V^*(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right) \tag{26}$$

$$= R(s, \pi^*(s)) + \gamma \sum_{s'} T(s, \pi^*(s), s') V^*(s') \tag{27}$$

The Value Iteration Algorithm

- Start with $V_0^*(s) = 0$, for all $s \in S$
- Given $V_k^*(s)$, calculate the values for all states $s \in S$ (depth $k+1$):

$$V_{k+1}^*(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_k^*(s') \right) \tag{28}$$

- Repeat until convergence (until $V_{k+1}(s)$ is nearly $V_k(s)$ for all states)

The convergence of this algorithm is guaranteed and the error between $V^*(s)$ and $V_k^*(s)$ decreases by a factor γ after each iteration.

Once the values are computed, we can turn them into the optimal policy:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \tag{29}$$

$$\pi^*(s) = \text{argmax}_{a \in A} Q^*(s,a) \tag{30}$$

As we rely on the Q-values to get the policy, we could alternatively compute these values directly:

The Q-Value Iteration Algorithm

- Start with $Q_0^*(s,a) = 0$ for all $s \in S$, $a \in A$.
- Given $Q_k^*(s,a)$, calculate updated Q-values for all states and actions a (depth $k+1$):

$$Q_{k+1}^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a' \in A} Q_k^*(s',a') \tag{31}$$

This algorithm has the same convergence guarantees as its value iteration counterpart.