SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.007 Machine Learning, Fall 2015
Lecture Notes for Week 12

Reinforcement Learning (I)

Last update: Thursday 3rd December, 2015 16:05

Learning from Feedback

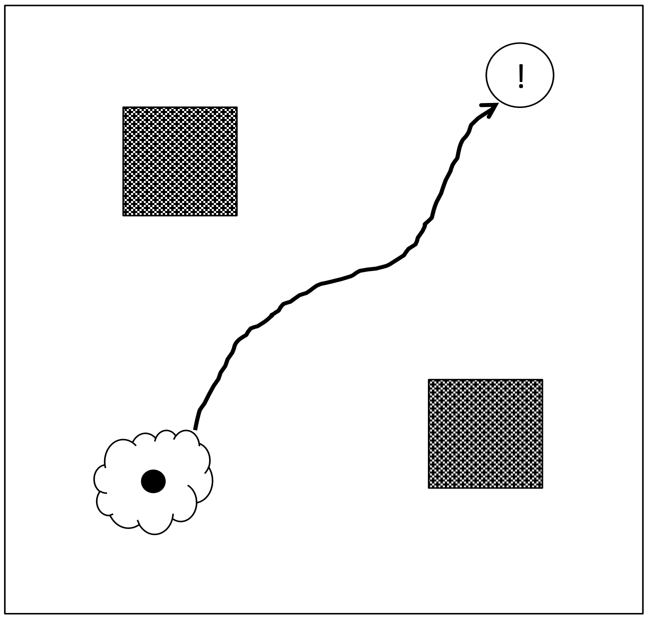

Figure 1: A robot navigation problem

Let's consider a robot navigation problem (see Fig 1 for illustration). At each point in time, our robot is located at a certain position on the grid. Our robot also has sensors which can help predict (with some noise) its position within the grid. Our goal is to bring the robot to its desired final destination (*e.g.*, charging station). The robot can move from one position to another in small increments. However, we assume that the movements are not deterministic. In other words, with some probability the robot moves to the desired next position, but there is also a chance that it ends up in another nearby location (*e.g.*, applying a bit too much power). Let's assume for simplicity that the states are discrete. That is, the positions correspond to *grid blocks*.

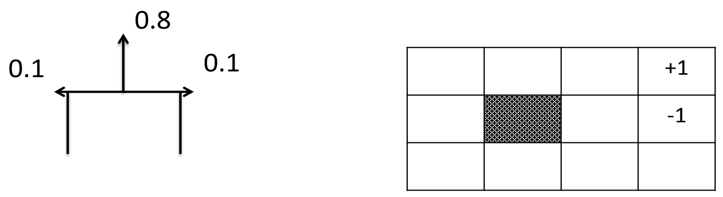

Figure 2: A robot (with actions) and the environment.

An example is shown in Figure 2, where each block is a position. For any direction the robot moves towards, there is a 0.8 probability that the robot will successfully arrive at the desired grid (which is the grid in front of the robot), and a 0.1 probability each that it will arrive at the left and right grid respectively. The robot will get bounced back if the destination grid is a wall and will stay at the current grid.

At the first glance, this problem may resemble HMMs: we use states y_t to encode the position of the robot, and observations x_t capture the sensor readings. The process is clearly Markov in the sense that the transition to the next state is only determined by the current state.

In the case of HMMs, we represent transitions as $T(i, j) = p(y_{t+1} = j | y_t = i)$. What is missing? This parametrization does not account for the fact that the robot can select actions such as the direction that it wants to move in. We will therefore expand the transition probabilities to incorporate selected actions. For example, $T(i, k, j) = p(y_{t+1} = j | y_t = i, a_t = k)$ specifies the probability of transitioning to j from i after taking action k.

In contrast to the more realistic setting discussed above, we will make the model simpler here by assuming that the states are *directly observable*. In other words, at every point, the robot knows its exact location in the grid. Put another way, the observation x_t fully determines y_t, and we will drop x_t as a result. Such modifications to the HMM are illustrated in Figure 3.

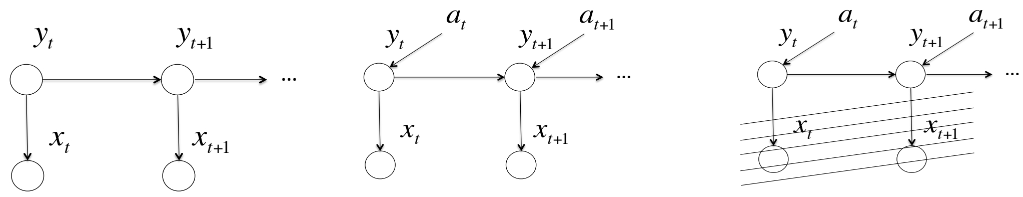

Figure 3: HMM $\rightarrow$ with actions $\rightarrow$ without observations.

The remaining piece in our robot navigation problem is to specify costs or rewards. Rewards can be associated with states: $R(s_t)$ (*e.g.*, the target charging station has a high reward) or they can also take into account the action that brings the robot to the follow-up state $R(s_{t-1}, a_{t-1}, s_t)$. Here is a simple example of a reward function:

$$R(s_t) = \begin{cases} 1 & \text{if on target} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Utility Function The utility function describes the *long term reward* for a robot when it takes actions. Intuitively, a utility function would be the sum of all the rewards that the robot accumulates. However, this definition may result in acquiring infinite reward (for instance, when the robot loops around). Also, it may be better to acquire high rewards sooner than later. One possible setting is to assume that the robot has a finite horizon: after N steps, the utility value does not change at all (here we assume the robot transit from a particular state to the following sequence of states: s_0, then s_1,...):

$$U([s_0, s_1, \dots, s_N, \dots, s_{N+k}]) = U([s_0, s_1, \dots, s_N]) \quad \forall k \geq 1 \tag{2}$$

However, under this assumption in certain cases it would be difficult to decide what can be the optimal action to take.

An alternative approach is to use so called *discounted rewards*. Even for infinite sequences, this utility function is guaranteed to have a finite value so long as the individual rewards are finite. For $0 \leq \gamma < 1$, we define it as

$$U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots = \sum_{t=0}^{\infty} \gamma^t R(s_t) \tag{3}$$

$$\frac{R_{min}}{1-\gamma} = \sum_{t=0}^{\infty} \gamma^t R_{min} \leq U([s_0, s_1, s_2, \dots]) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma} \tag{4}$$

This formulations assigns higher utilities to rewards that come early. The discounting also helps us solve for these utilities (important for convergence of the algorithms we will cover later in the lecture).

Another way to think of the discounted utility is by augmenting our state space with an additional *halt* state. Once the agent reaches this state, the experience stops. At any point, we will transition to this halt state with probability $1 - \gamma$. Thus the probability that we continue to move on is γ. Larger γ means longer horizon.

Policy A policy π is a function that maps from each state to an action. Our goal is to compute an optimal policy that maximizes the expected utility, *i.e.*, the action that the robot takes in any state is chosen with the idea of maximizing the discounted future rewards. As illustrated in Fig 4, an optimal policy depends heavily on the reward function. In the first case, all grids are assigned a reward -0.01, and in the second case, all grids are assigned a reward -2.0 (except for the two grids at the top right corner which have rewards +1 and -1 respectively). In fact, it is the reward function that specifies the goal.

We will consider here two situations where we would like to learn the optimal policy:

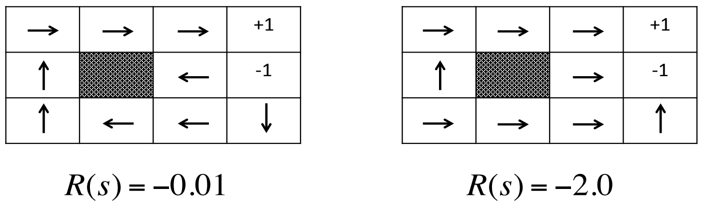

Figure 4: We obtain different optimal policies when different rewards are considered. $R(s)$ is the reward for any state s, except for the two top-right states.

- **Markov Decision Process (MDP)** We assume that reward function and transition probabilities are known and available to the robot. More specifically, we are provided with
  - a set of states S
  - a set of actions A
  - a transition probability function $T(s, a, s') = p(s'|s, a)$
  - a reward function $R(s, a, s')$ (or just $R(s')$)
- **Reinforcement Learning** The reward function and transition probabilities are unknown (more precisely: their forms are known, but values are unknown), and the robot only knows
  - a set of states S
  - a set of actions A

Value Iteration

Value iteration is an algorithm for finding *an* optimal policy (there can be more than one) 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:

- $\pi(s)$ – a particular policy that specifies the action we should take in state s.
- $V^\pi(s)$ – The *value* of state s under policy π, *i.e.*, the expected utility of starting in state s and act based on policy π thereafter.
- $Q^\pi(s, a)$ – The *Q-value* of state s and action a under policy π. It is the expected utility of starting in state s, taking action a and acting based on policy π thereafter.

We also introduce the following notations related to the optimal policy:

- $\pi^*(s)$ – The optimal policy $\pi^*(s)$ specifies the action we should take in state s. Following policy π^* would, in expectation, result in the highest expected utility.

- $V^*(s)$ – The *value* of state s under the optimal policy π^*, *i.e.*, the expected utility of starting in state s and acting optimally thereafter.
- $Q^*(s,a)$ – The *Q-value* of state s and action a under the optimal policy π^*. It is the expected utility of starting in state s, taking action a and acting optimally thereafter.

The above quantities are clearly related:

$$\begin{aligned} V^*(s) &= \max_a Q^*(s,a) = Q^*(s,\pi^*(s)) && (5)\\ Q^*(s,a) &= \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] && (6)\\ V^*(s) &= \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] && (7)\\ &= \sum_{s'} T(s,\pi^*(s),s')[R(s,\pi^*(s),s') + \gamma V^*(s')] && (8)\end{aligned}$$

For example, in Eq. (6), we evaluate the expected reward of taking action a in state s. We sum over the possible next states s', weighted by the transition probabilities corresponding to the state s and action a. Each summand combines the immediate reward with the expected utility we would get if we started from s' and followed the optimal policy thereafter. The future utility is discounted by γ as discussed above. In other words, we have simple one-step lookahead relationship among the utility values.

Based on these equations, we can recursively estimate $V_k^*(s)$, the optimal value considering next k steps. As $k \to \infty$, the values converges to the optimal values $V^*(s)$.

The Value Iteration Algorithm

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

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')] \quad (9)$$

- Repeat the above until convergence (until $V_{i+1}(s)$ is nearly $V_i(s)$ for all states)

The convergence of this algorithm is guaranteed. We will not show a full proof here but just illustrate the basic idea. Consider a simple MDP with a single state and a single action. In such a case:

$$V_{i+1} = R + \gamma V_i \quad (10)$$

We also know that for the optimal V^* the following must hold:

$$V^* = R + \gamma V^* \quad (11)$$

By subtracting these two expressions, we get:

$$(V_{i+1} - V^*) = \gamma(V_i - V^*) \tag{12}$$

Thus, after each iteration, the difference between the estimate and the optimal value decreases by a factor $\gamma < 1$. Hence the value for V converges to V^*.

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

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{13}$$

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{14}$$

As we rely on the Q-values to get the policy, we could alternatively compute these Q-values directly. We can develop a Q-value iteration algorithm instead. How do we do that? We'll talk about this in the next class.

Learning Objectives

You need to know:

1. What is a Markov decision process, reinforcement learning, and what are the components required
2. What are the basic concepts such as states, actions, policy, reward functions and utility functions and their roles in MDP problems
3. What does value iteration algorithm do and how to perform value iteration for a given simple MDP problem