Markov Decision Process Notes Part 2

(Math 562)

Reference Artificial Intelligence, Chapter 16/ Chapter 17, Making complex decisions, Russell and Norvig

https://en.wikipedia.org/wiki/Artificial_Intelligence:_A_Modern_Approach

http://aima.cs.berkeley.edu/

Review of Last Class

Definition of Markov decision process, or MDP,

To sum up: a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov decision process, or MDP, and consists of

- a set of states (with an initial state s_0);
- a set ACTIONS(s) of actions in each state;
- a transition model $P\left(s' \mid s, a\right)$; and
- a reward function $R\left(s, a, s'\right)$.

Methods for solving MDPs usually involve **dynamic programming:** simplifying a problem by recursively breaking it into smaller pieces and remembering the optimal solutions to the pieces.

definition of a policy:

Function $\pi(s) \in A(s)$

definition of expected utility of a policy:

Each time a given policy is executed starting from the initial state, the stochastic nature of the environment may lead to a different environment history.

The quality of a policy is therefore measured by the **expected utility** of the possible environment histories generated by that policy.

definition of optimal policy:

An optimal policy is a policy that yields the highest expected utility. (May not be unique)

We use π^* to denote an optimal policy.

How to calculate the utility of state sequences.

Additive discounted rewards:

$$U_h\left([s_0, a_0, s_1, a_1, \ldots, s_{N+k}]\right) = R(s_0, a_0, S_1) + \gamma R(s_1, a_1, S_2) + \gamma^2 R(s_2, a_2, S_3) + \ldots \quad (51)$$

For a discount factor $\gamma \in (0, 1]$

Consequences/Analysis of Additive discounted rewards

finite rewards

If environment history is infinite, but rewards bounded by $|r| \leq \mathbb{R}_{\max}$

$$U_h\left([s_0, a_0, s_1, a_1, \ldots, s_{N+k}]\right) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, S_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma} \quad (52)$$

Optimal policies and the utilities of states

Utility of a state

Given this definition, the true utility of a state is just $U^{\pi^*}(s)$-that is, the expected sum of discounted rewards if the agent executes an optimal policy. We write this as $U(s)$,

$$U(s) = U^{\pi^*}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R\left(S_t, \pi^*\left(S_t\right), S_{t+1}\right)\right] \quad (53)$$

(For any fixed choice of optimal policy)

The utility function $U(s)$ allows the agent to select actions by using

the principle of maximum expected utility.

choose the action that maximizes the reward for the next step plus the expected discounted utility of the subsequent state:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P\left(s' \mid s, a\right)\left[R\left(s, a, s'\right) + \gamma U\left(s'\right)\right] \quad (54)$$

We have defined the utility of a state, $U(s)$, as the expected sum of discounted rewards from that point onwards.

From this, it follows that there is a direct relationship between the utility of a state and the utility of its neighbors:

- the utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action. That is, the utility of a state is given by

$$U(s) = \max_{a \in A(s)} \sum_{s'} P\left(s' \mid s, a\right) \left[R\left(s, a, s'\right) + \gamma U\left(s'\right)\right] \tag{55}$$

This is called the Bellman equation, after Richard Bellman (1957).

Policy from Utility

- Given the utility function, can find the (an) optimal policy $\pi*$ as any element of the argmax in the preceding equation

Action Utility Function

Another important quantity is the **action-utility function, or Q-function:**

- $Q(s, a)$ is the expected utility of taking a given action in a given state. The Q-function is related to utilities in the obvious way:

$$U(s) = \max_{a} Q(s, a) \tag{56}$$

Furthermore, the optimal policy can be extracted from the Q-function as follows:

$$\pi^*(s) = \operatorname*{argmax}_{a} Q(s, a) \tag{57}$$

We can also develop a Bellman equation for Q-functions, noting that the expected total reward for taking an action is its immediate reward plus the discounted utility of the outcome state, which in turn can be expressed in terms of the Q-function:
(17.8)

$$\begin{aligned} Q(s, a) &= \sum_{s'} P\left(s' \mid s, a\right) \left[R\left(s, a, s'\right) + \gamma U\left(s'\right)\right] \\ &= \sum_{s'} P\left(s' \mid s, a\right) \left[R\left(s, a, s'\right) + \gamma \max_{a'} Q\left(s', a'\right)\right] \end{aligned} \tag{58}$$

Solving the Bellman equations for U (or for Q) gives us what we need to find an optimal policy. The Q-function shows up again and again in algorithms for solving MDPs, so we shall use the following definition:

$$\begin{aligned} &\text{function } \mathrm{Q-VALUE}(mdp, s, a, U) \text{ returns a utility value} \\ &\text{return } \sum_{s'} P\left(s' \mid s, a\right) \left[R\left(s, a, s'\right) + \gamma U\left[s'\right]\right] \end{aligned} \tag{59}$$

Reward scales

an affine transformation transformation of rewards will leave the optimal policy unchanged in an MDP:

$$R'\left(s,a,s'\right)=mR\left(s,a,s'\right)+b \tag{60}$$

- Optimal policies unchanged.

It turns out, however, that the additive reward decomposition of utilities leads to significantly more freedom in defining rewards. Let $\Phi(s)$ be any function of the state s. Then, according to the **shaping theorem**, the following transformation leaves the optimal policy unchanged:
(17.9)

$$R'\left(s,a,s'\right)=R\left(s,a,s'\right)+\gamma\Phi\left(s'\right)-\Phi(s) \tag{61}$$

Shaping theorem

To show that this is true, we need to prove that two MDPs, M and M', have identical optimal policies as long as they differ only in their reward functions as specified above.

[Proof by showing the bellman equation is invariant under transformation]

Algorithms for MDPs

- Value Iteration
- Policy Iteration

Policy Valuation

For a fixed (non-optimal) Policy $\pi(s)$, we can measure the corresponding state value function

$U(s)=U^{\pi}(s)$ = Expected discounted rewards starting from state s

$$U(s)=U^{\pi^*}(s)=E\left[\sum_{t=0}^{\infty}\gamma^{t}R\left(S_t,\pi^*\left(S_t\right),S_{t+1}\right)\right] \tag{62}$$

Satisfies Linear Equation

$$U(s)=\sum_{s'}P\left(s'\mid s,a\right)\left[R\left(s,a,s'\right)+\gamma U\left(s'\right)\right],\qquad a=\pi(s) \tag{63}$$

$$U(s)=Q(s,a=\pi(s)) \tag{64}$$

Linear equation with one equation for each state s. (Need to know the transition prob and rewards)

Define the bellman operator

$$B(R,U)(s) = B(U)(s) = \max_{a \in A(s)} L(U,a)(s) \tag{65}$$

$$L(U,a)(s) = \sum_{s'} P\left(s' \mid s,a\right)\left[R\left(s,a,s'\right) + \gamma U\left(s'\right)\right] \tag{66}$$

$$\pi(U)(s) = \arg\max_{a \in A(s)} L(U,a)(s) \tag{67}$$

So, can also write

$$B(U)(s) = L(U, \pi(U)(s))(s) \tag{68}$$

Value iteration comes from iterating the Bellman Operator

$$U^{n+1} = B(U^n) \tag{69}$$

Corresponds to choosing the best policy at each s according to the current value estimate U^n

Convergence of Value Iteration

Value iteration eventually converges to a unique set of solutions of the Bellman equations.

- we obtain some methods for assessing the error in the utility function returned when the algorithm is terminated early; this is useful because it means that we don't have to run forever.
- A contraction has only one fixed point; if there were two fixed points they would not get closer together when the function was applied, so it would not be a contraction.
- When the function is applied to any argument, the value must get closer to the fixed point (because the fixed point does not move), so repeated application of a contraction always reaches the fixed point in the limit.

Next, we need a way to measure distances between utility vectors. We will use the max norm, which measures the length of a vector by the absolute value of its biggest component:

$$\|U\| = \|U\|_\infty = \max_s |U(s)| \tag{70}$$

With this definition, the "distance" between two vectors, $\|U - U'\|$, is the maximum difference between any two corresponding elements. The main result of this section is the following: Let U_i and U_i' be any two utility vectors. Then we have
(17.11)

$$\left\|BU_i - BU_i'\right\| \leq \gamma \left\|U_i - U_i'\right\| \tag{71}$$

That is, the Bellman update is a contraction by a factor of γ on the space of utility vectors. Hence, from the properties of contractions in general, it follows that value iteration always converges to a unique solution of the Bellman equations whenever $\gamma < 1$.

We can also use the contraction property to analyze the rate of convergence to a solution. In particular, we can replace U_i' in Bellman equation, with the true utilities U, for which $BU = U$. Then we obtain the inequality

$$\|U_{i+1} - U\| = \|BU_i - U\| \leq \gamma \|U_i - U\| \tag{72}$$

If we view $\|U_i - U\|$ as the error in the estimate U_i, we see that the error is reduced by a factor of at least γ on each iteration. Thus, value iteration converges exponentially fast.

We can calculate the number of iterations required as follows:

- First, recall from Equation (17.1)! that the utilities of all states are bounded by $\pm R_{\max}/(1-\gamma)$.
- This means that the maximum initial error $\|U_0 - U\| \leq 2R_{\max}/(1-\gamma)$.
- Suppose we run for N iterations to reach an error of at most ϵ. Then, because the error is reduced by at least γ each time, we require $\gamma^N \cdot 2R_{\max}/(1-\gamma) \leq \epsilon$. Taking logs, we find that

$$N = \lceil \log\left(2R_{\max}/\epsilon(1-\gamma)\right)/\log(1/\gamma) \rceil \tag{73}$$

iterations suffice.

From Update Error to convergence error

if $\|U_{i+1} - U_i\| < \epsilon(1-\gamma)/\gamma$ then $\|U_{i+1} - U\| < \epsilon$.

Proof: Handwritten notes

What about the error in the policy?

So far, we have analyzed the error in the utility function returned by the value iteration algorithm. What the agent really cares about, however, is how well it will do if it makes its decisions on the basis of this utility function.

- Suppose that after i iterations of value iteration, the agent has an estimate U_i of the true utility U and obtains the maximum expected utility (MEU) policy π_i based on one-step look-ahead using U_i
-

$$\pi(U)(s) = \arg\max_{a \in A(s)} L(U, a)(s) \tag{74}$$

- Will the resulting behavior be nearly as good as the optimal behavior? This is a crucial question for any real agent, and it turns out that the answer is yes.

- $U^{\pi_i}(s)$ is the utility obtained if π_i is executed starting in s, and the policy loss $\|U^{\pi_i} - U\|$ is the most the agent can lose by executing π_i instead of the optimal policy π^*. The policy loss of π_i is connected to the error in U_i by the following inequality:

if $\|U_i - U\| < \epsilon$ then $\|U^{\pi_i} - U\| < 2\epsilon$.

Policy Iteration

The policy iteration algorithm alternates the following two steps, beginning from some initial policy π_0 :

- POLICY EVALUATION: given a policy π_i, calculate $U_i = U^{\pi_i}$, the utility of each state if π_i were to be executed.
- POLICY IMPROVEMENT: Calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i

The algorithm terminates when the policy improvement step yields no change in the utilities.

at this point, U_i is a fixed point of the Belllman equation, so π_i must be optimal.

How do we implement POLICY-EVALUATION? It turns out that doing so is simpler than solving the standard Bellman equations (which is what value iteration does), because the action in each state is fixed by the policy. At the i th iteration, the policy π_i specifies the action $\pi_i(s)$ in state s. This means that we have a simplified version of the Bellman equation relating the utility of s (under π_i) to the utilities of its neighbors:

$$U_i(s) = \sum_{s'} P\left(s' \mid s, \pi_i(s)\right)\left[R\left(s, \pi_i(s), s'\right) + \gamma U_i\left(s'\right)\right] \tag{75}$$

or

$$U_i(s) = L(U, \pi_i(s))(s) = \sum_{s'} P\left(s' \mid s, a = \pi_i(s)\right)\left[R\left(s, a = \pi_i(s), s'\right) + \gamma U\left(s'\right)\right] \tag{76}$$

- Can solve this linear equation (when state space is small)
- Can approximately solve this linear equation by iterating a few times.