인공지능

14차시: Markov Decision Processes

서울대학교 컴퓨터공학부
담당 교수: 장병탁
Seoul National University
Byoung-Tak Zhang
Introduction

- Decide what to do today, given that we may decide again tomorrow.

**Sequential decision problems**

- The agent’s utility depends on a *sequence* of decisions (multi-step inferences). NOT one-shot or episodic decision problems. *Search and planning* problems are special cases.

- **MDP**: fully-observable stochastic environments, Markovian transition, additive rewards

- **POMDP**: partially observable MDP

- Finding optimal policies: *Bellman equations*
  - Value iteration
  - Policy iteration

**Decision-theoretic agents**

- Online agents for POMDP

- Dynamic decision networks = dynamic Bayesian net + decision net
Motivating Example: Sequential Decisions for Reaching the Goal

The agent must decide what to do now!

- **Action at time** $t$: $A_t$
- **State (unobservable)**: $X_t$
- **Evidence (observable)**: $E_t$
- **Reward (short-term)**: $R_t$
- **Utility (long-term)**: $U_t$

- **Transition model**: $P(X_{t+1} \mid X_t, A_t)$
- **Sensor model**: $P(E_t \mid X_t)$

**Partially Observable Markov Decision Process (POMDP)**

Figure 17.10  The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is $t$ and the agent must decide what to do—that is, choose a value for $A_t$. The network has been unrolled into the future for three steps and represents future rewards, as well as the utility of the state at the look-ahead horizon.
Outline (Lecture 14)

14.1 Sequential Decision Problems ...................... 5
14.2 Value Iteration ........................................... 11
14.3 Policy Iteration .......................................... 18
14.4 Partially Observable MDPs ............................. 22
Summary ......................................................... 32
Homework ....................................................... 33
1) Decision Process

- The utility function will depend on a sequence of states—an environment history—rather than on a single state.
  - Because the decision problem is sequential
- In each state $s$, the agent receives a reward $R(s)$
  - which may be positive or negative, but must be bounded.
- For example, the reward is $-0.04$ in all states except the terminal states (which have rewards $+1$ and $-1$).
2) Markov Decision Process (MDP)

- A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards

- Consists of
  - a set of states (with an initial state $s_0$)
  - a set $\text{Actions}(s)$ of actions in each state
  - a transition model $P(s \mid s, a)$
  - a reward function $R(s)$

- **Policy** is a solution that must specify what the agent should do for any state that the agent might reach.

- An **optimal policy** is a policy that yields the highest expected utility.
2) Markov Decision Process (MDP) contd.

Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.
3) Utilities over Time

The first question to answer is whether there is a finite horizon or an infinite horizon for decision making.

- **Finite horizon**
  - There is a *fixed time* $N$ after which nothing matters.
  - The optimal policy for a finite horizon is *nonstationary*.

- **Infinite horizon**
  - With no *fixed time limit*, there is no reason to behave differently in the same state at different times.
  - The optimal action depends only on the current state, and the optimal policy is *stationary*.

The most natural assumption is that the agent’s preferences between state sequences are *stationary*.
14.1 Sequential Decision Problems (5/6)

4) Assigning Utilities to Sequences

- **Additive rewards**: \( U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots \)

- **Discounted rewards**: \( U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \)

  *discount factor* \( \gamma \) \((0 < \gamma < 1)\). When \( \gamma \) is close to 0, small weight on distant futures. When \( \gamma \) is 1, discounted rewards equivalent to additive rewards.

- With discounted rewards, the utility of an infinite sequence is finite. If \( \gamma < 1 \) and rewards are bounded by \( \pm R_{\text{max}} \), we have

\[
U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\text{max}} = R_{\text{max}}/(1 - \gamma)
\]

- **Proper policy**: a policy that is guaranteed to reach a terminal state. If the environment contains terminal states and if the agent is guaranteed to get to one eventually, then we will never need to compare infinite sequences.

- Infinite sequences can be compared in terms of the average reward obtained per time step.
5) Optimal Policies and the Utilities of States

The expected utility obtained by executing $\pi$ starting in $s$ is given by

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t) \right]$$

where the expectation is with respect to the probability distribution over state sequences determined by $s$ and $\pi$.

Use $\pi^*_s$ to denote one of these policies:

$$\pi^*_s = \arg\max_{\pi} U^\pi(s)$$

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 expected utility of the subsequent state:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$
1) The Bellman Equation for Utilities

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state,

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

This is called the Bellman Equation.

Bellman equations for the 4×3 world (for the state (1,1)):

$$U(1, 1) = -0.04 + \gamma \max \left[ 0.8U(1, 2) + 0.1U(2, 1) + 0.1U(1, 1), \right.$$

$$0.9U(1, 1) + 0.1U(1, 2),$$

$$0.9U(1, 1) + 0.1U(2, 1),$$

$$0.8U(2, 1) + 0.1U(1, 2) + 0.1U(1, 1) \right].$$
2) The Value Iteration Algorithm

```
function VALUE-ITERATION(mdp, ϵ) returns a utility function
    inputs: mdp, an MDP with states S, actions A(s), transition model P(s' | s, a),
            rewards R(s), discount γ
            ϵ, the maximum error allowed in the utility of any state
    local variables: U, U', vectors of utilities for states in S, initially zero
                    δ, the maximum change in the utility of any state in an iteration
    repeat
        U ← U'; δ ← 0
        for each state s in S do
            U'[s] ← R(s) + γ \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']
            if |U'[s] - U[s]| > δ then δ ← |U'[s] - U[s]|
        until δ < ϵ(1 - γ)/γ
    return U
```

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

Bellman update

\[ U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U_i(s') \]

← equilibrium
2) The Value Iteration Algorithm

\[\begin{array}{c}
\text{Utility estimates} \\
\text{Number of iterations} \\
\text{(a)}
\end{array}\]

\[\begin{array}{c}
\text{Iterations required} \\
\text{Discount factor } \gamma \\
\text{(b)}
\end{array}\]

**Figure 17.5** (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations \(k\) required to guarantee an error of at most \(\epsilon = c \cdot R_{\text{max}}\), for different values of \(c\), as a function of the discount factor \(\gamma\).
3) Convergence of Value Iteration

The basic concept used in showing that value iteration converges is the notion of a contraction. A contraction is a function of one argument that, when applied to two different inputs in turn, produces two output values that are “closer together,” by at least some constant factor, than the original inputs.

E.g. the function “divide by two” is a contraction. This function has a fixed point, namely zero.

Two important properties of contractions:

1. 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.
2. 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.
3) Convergence of Value Iteration cont.

- **$B$**: the Bellman update operator that is applied simultaneously to update the utility of every state.
- **$U_i$**: the vector of utilities for all the states at the $i$-th iteration.
- The **Bellman update equation** can be written as $U_{i+1} \leftarrow BU_i$.
- The **max norm** to measure distances between utility vectors: $||U|| = \max_s |U(s)|$.
- Let $U_i$ and $U'_i$ be any two utility vectors. Then we have:
  $||BU_i - BU'_i|| \leq \gamma||U_i - U'_i||$
  - Bellman update is a **contraction** by a factor of $\gamma$ on the space of utility vectors.
- In particular, we can replace $U'_i$ with the true utilities $U$, for which $BU = U$.
- Then we obtain the inequality:
  $||BU_i - BU|| \leq \gamma||U_i - U||$
  - 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 $\gamma$ on each iteration.
- This means that value iteration converges exponentially fast.
3) Convergence of Value Iteration contd.

We can calculate the number of iterations required to reach a specified error bound as follows:

First, the utilities of all states are bounded by $\pm R_{max}/(1 - \gamma)$.

Maximum initial error is $\|U_0 - U\| \leq 2R_{max}/(1 - \gamma)$.

Because the error is reduced by at least $\gamma$ each time, $\gamma^N \cdot 2 R_{max}/(1 - \gamma) \leq \epsilon$.

Taking logs, $N = \left\lceil \log \left( \frac{2R_{max}}{\epsilon(1 - \gamma)} \right) / \log(1/\gamma) \right\rceil$

From the contraction property, it can be shown that if the update is small, then the error, compared with the true utility function, also is small.

If $\|U_{i+1} - U_i\| < \epsilon(1 - \gamma)/\gamma$ then $\|U_{i+1} - U\| < \epsilon$. 
3) Convergence of Value Iteration contd.

- $U^\pi_i(s)$ is the utility obtained if $\pi_i$ is executed starting in $s$.
- The policy loss $\|U^\pi_i - U\|$ is the most the agent can lose by executing $\pi_i$ instead of the optimal policy $\pi^*$. 
- The policy loss of $\pi_i$ is connected to the error in $U_i$ by the following inequality:
  - If $\|U_i - U\| < \epsilon$ then $\|U^\pi_i - U\| < 2\epsilon\gamma/(1 - \gamma)$

The policy $\pi_i$ is optimal (policy loss = 0) when $i = 4$, even though the maximum error in $U_i$ is still 0.46.
We have seen that it is possible to get an optimal policy even when the utility function estimate is inaccurate. If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise. This insight suggests an alternative way to find optimal policies, i.e. policy iteration algorithms.

1) Policy Iteration Algorithm

- Alternates the following two steps, beginning from some initial policy $\pi_0$:
  - Policy evaluation: given a policy $\pi_i$, calculate $U_i = U^{\pi_i}$, the utility of each state if $\pi_i$ were to be executed.
  - Policy improvement: Calculate a new MEU policy $\pi_{i+1}$, using one-step look-ahead based on $U_i$.

- The algorithm terminates when the policy improvement step yields no change in the utilities.
14.3 Policy Iteration (2/4)

1) Policy Iteration Algorithm contd.

Policy iteration uses a **simplified** (linear, no max operation) version of the **Bellman equation** relating the utility of $s$ (under $\pi_i$) to the utilities of its neighbors:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) U_i(s')$$

For example, suppose $\pi_i$ is the policy shown in Figure 17.2(a) $\rightarrow$

Then we have $\pi_i(1, 1) = U_1$, $\pi_i(1, 2) = U_1$, and so on,

and the simplified Bellman equations are

$$U_i(1, 1) = -0.04 + 0.8U_i(1, 2) + 0.1U_i(1, 1) + 0.1U_i(2, 1),$$

$$U_i(1, 2) = -0.04 + 0.8U_i(1, 3) + 0.2U_i(1, 2),$$

$$\vdots$$
2) Modified Policy Iteration

- We can perform some number of simplified value iteration steps (simplified because the policy is fixed, no max operation) to give a reasonably good approximation of the utilities.

- The simplified (linear) Bellman update for this process is

\[ U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') \]

  and this is repeated \( k \) times to produce the next utility estimate.

- The resulting algorithm is called **modified policy iteration**.
  - It is often much more efficient than standard policy iteration or value iteration.
2) Modified Policy Iteration

function POLICY-ITERATION(mdp) returns a policy
  inputs: mdp, an MDP with states S, actions A(s), transition model P(s' | s, a)
  local variables: U, a vector of utilities for states in S, initially zero
                  π, a policy vector indexed by state, initially random

repeat
  U ← POLICY-EVALUATION(π, U, mdp)
  unchanged? ← true
  for each state s in S do
    if \( \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s'] > \sum_{s'} P(s' | s, \pi[s]) U[s'] \) then do
      \( \pi[s] \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s'] \)
    unchanged? ← false
  until unchanged?
return π

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.

Updating the utility or policy (policy improvement or simplified value iteration)
- for all states at once
- pick any subset of states (asynchronous policy iteration)
1. Definition of POMDPs

- When the environment is only \textit{partially observable}, the situation is, one might say, much less clear.

- A \textit{POMDP} has the same elements as an MDP
  - \textit{transition} model $P(s | s, a)$,
  - \textit{actions} $A(s)$,
  - \textit{reward} function $R(s)$

- But, like the partially observable search problems of Section 4.4, it also has a \textit{sensor model} $P(e | s)$.

- The sensor model specifies the probability of perceiving evidence $e$ in state $s$. 
1) Definition of POMDPs contd.

For POMDPs, we also have an action to consider, but the result is essentially the same.

If \( b(s) \) was the previous belief state, and the agent does action \( a \) and then perceives evidence \( e \), then the new belief state is given by

\[
b'(s') = \alpha P(e|s') \sum_s P(s'|s, a) b(s)
\]

where \( \alpha \) is a normalizing constant that makes the belief state sum to 1.

\( b' = \text{FORWARD}(b, a, e) \)
2) Cycle of a POMDP Agent

- The optimal action depends only on the agent’s current belief state.
  - That is, the optimal policy can be described by a mapping $\pi^*(b)$ from belief states to actions.
- It does not depend on the actual state the agent is in.
- The decision cycle of a POMDP agent is like the following three steps:
  1. Given the current belief state $b$, execute the action $a = \pi^*(b)$
  2. Receive percept $e$.
  3. Set the current belief state to $\text{FORWARD}(b, a, e)$ and repeat.
- The POMDP belief-state space is continuous.
3) Outcome of Actions

Calculate the probability that an agent in belief state $b$ reaches belief state $b'$ after executing action $a$.

The probability of perceiving $e$, given that $a$ was performed starting in belief state $b$, is given by summing over all the actual states $s$ that the agent might reach:

$$P(e|a, b) = \sum_{s'} P(e|a, s', b)P(s'|a, b)$$

$$= \sum_{s'} P(e|s')P(s'|a, b)$$

$$= \sum_{s'} P(e|s')\sum_{s} P(s'|s, a)b(s).$$
3) Outcome of Actions cont.

Let us write the probability of reaching \( b \) from \( b' \), given action \( a \), as \( P(b' \mid b, a) \).

\[
P(b' \mid b, a) = P(b' \mid a, b) = \sum_e P(b' \mid e, a, b) P(e \mid a, b)
= \sum_e P(b' \mid e, a, b) \sum_{s'} P(e \mid s') \sum_s P(s' \mid s, a) b(s)
\]

where \( P(b' \mid e, a, b) \) is 1 if \( b' = \text{FORWARD}(b, a, e) \) and 0 otherwise.

Reward function for belief states is

\[
\rho(b) = \sum_s b(s) R(s)
\]
14.4 Partially Observable MDPs (6/10)

4) Value Iteration for POMDPs

Consider an optimal policy $\pi^*$ and its application in a specific belief state $b$. The policy is exactly equivalent to a conditional plan, as defined in Chapter 4 for nondeterministic and partially observable problems.

We make two observations:

- $\alpha_p(s)$: utility of executing a fixed conditional plan (= policy) $p$ starting in physical state $s$
- Expected utility of executing $p$ in belief state $b$ is $\sum_s b(s) \alpha_p(s)$, or $b \cdot \alpha_p$

Expected utility of $b$ under the optimal policy: $U(b) = U^{\pi^*}(b) = \max_p b \cdot \alpha_p$

$\alpha_{[\text{stay}]}(0) = R(0) + \gamma(0.9R(0) + 0.1R(1)) = 0.1$
$\alpha_{[\text{stay}]}(1) = R(1) + \gamma(0.9R(1) + 0.1R(0)) = 1.9$
$\alpha_{[\text{go}]}(0) = R(0) + \gamma(0.9R(1) + 0.1R(0)) = 0.9$
$\alpha_{[\text{go}]}(1) = R(1) + \gamma(0.9R(0) + 0.1R(1)) = 1.1$
14.4 Partially Observable MDPs (7/10)

- Undominated plans
- Hyperplane
- Piecewise linear
- Convex

(a) Utility of two one-step plans as a function of the initial belief state $b(1)$ for the two-state world, with the corresponding utility function shown in bold.

(b) Utilities for 8 distinct two-step plans.

(c) Utilities for four undominated two-step plans.

(d) Utility function for optimal eight-step plans.
4) Value Iteration for POMDPs

```python
function POMDP-VALUE-ITERATION(pomdp, ε) returns a utility function
inputs: pomdp, a POMDP with states S, actions A(s), transition model P(s' | s, a),
sensor model P(ε | s), rewards R(s), discount γ
ε, the maximum error allowed in the utility of any state
local variables: U, U', sets of plans p with associated utility vectors α_p

U' ← a set containing just the empty plan [], with α_1(s) = R(s)
repeat
    U ← U'
    U' ← the set of all plans consisting of an action and, for each possible next percept,
        a plan in U with utility vectors computed according to Equation (17.13)
    U' ← REMOVE-DOMINATED-PLANS(U')
until MAX-DIFFERENCE(U, U') < ε(1 − γ)/γ
return U
```

Figure 17.9 A high-level sketch of the value iteration algorithm for POMDPs. The REMOVE-DOMINATED-PLANS step and MAX-DIFFERENCE test are typically implemented as linear programs.
5) Online Agents for POMDPs

The basic elements of the agent design for partially observable, stochastic environments:

- The transition and sensor models are represented by a dynamic Bayesian network (DBN) (Chapter 15).
- The dynamic Bayesian network is extended with decision and utility nodes, as used in decision networks in Chapter 16. The resulting model is called a dynamic decision network, or DDN.
- A filtering algorithm is used to incorporate each new percept and action and to update the belief state representation.
- Decisions are made by projecting forward possible action sequences and choosing the best one.

Action at time $t$: $A_t$

Reward: $R_t$

Utility: $U_t$

Transition model: $P(X_{t+1} \mid X_t, A_t)$

Sensor model: $P(E_t \mid X_t)$
5) Online Agents for POMDPs

Figure 17.11 Part of the look-ahead solution of the DDN in Figure 17.10. Each decision will be taken in the belief state indicated.
Sequential decision problems in uncertain environments, also called Markov decision processes, or MDPs, are defined by a transition model specifying the probabilistic outcomes of actions and a reward function specifying the reward in each state.

The solution of an MDP is a policy that associates a decision with every state that the agent might reach. An optimal policy maximizes the utility of the state sequences encountered when it is executed.

The value iteration algorithm for solving MDPs works by iteratively solving the equations relating the utility of each state to those of its neighbors.

Policy iteration alternates between calculating the utilities of states under the current policy and improving the current policy with respect to the current utilities.

A decision-theoretic agent can be constructed for POMDP environments. The agent uses a dynamic decision network to represent the transition and sensor models, to update its belief state, and to project forward possible action sequences.
Homework

- 17.8 (The 3x3 World, Figure 17.14(a))
- 17.13 (The 4x3 POMDP)
출처

사진