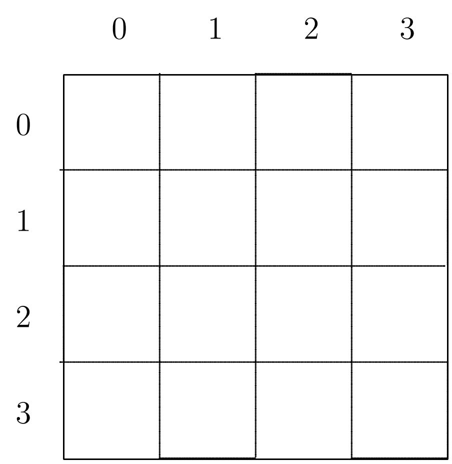
18 Optimal cyclic decision making

What if the state space has cycles?

18.1 Example: Optimal behavior strategy for a mouse

Assignment 4

- A mouse and a cat live in a 4×4 grid



- Cheese is sitting at the corners $(0, 0)$ and $(3, 3)$
- The mouse's goal is to eat cheese while avoiding the cat

$$\texttt{Reward}(s) = \begin{cases} 0 & \text{if mouse not on cheese and cat not on mouse} \\ 1 & \text{if mouse on cheese and cat not on mouse} \\ -3 & \text{if mouse not on cheese but cat on mouse} \\ -2 & \text{if mouse on cheese but cat also on mouse} \end{cases}$$

- A state can be described by four numbers (m_i, m_j, c_i, c_j)
  - (m_i, m_j) gives the coordinates of the mouse
  - (c_i, c_j) gives the coordinates of the cat
  - Thus, there are a total of $4^4 = 256$ states

- The cat and the mouse can execute one of five actions

(0, -1)	move left
(-1, 0)	move up
(0, 0)	stay
(1, 0)	move down
(0, 1)	move right

To make implementation easier:

- state vectors recoded into state numbers, 1 to 256
- action vectors recoded into numbers, 1 to 5
- Matlab functions statedecode.m, statencode.m, actdecode.m, actencode.m convert between numerical and vector-based representations

Two types of environment

Environment R Cat is blind and moves randomly

Given

- 256×1 matrix `Reward`
- $256 \times 256 \times 5$ matrix `Probs` where

$$\text{Probs}(sg, sn, a) = \text{P}(sn|sg, a)$$

- `Gamma` (default value 0.95)

Environment A Cat can see and is clairvoyant (can guess your move!)

Given

- 256×1 matrix `Reward`
- $256 \times 256 \times 5$ matrix `Possibs` where

$$\text{Possibs}(sg, sn, a) = \begin{cases} 1 & \text{if } sn \text{ is possible from taking } a \text{ in } sg \\ 0 & \text{otherwise} \end{cases}$$

- `Gamma` (default value 0.95)
- `rho` (probability cat takes random move, specified in `Probs`)

In each case return an optimal `Policy`: 256×1 vector with entries in $\{1, 2, 3, 4, 5\}$

18.2 Optimizing reward in a cyclic R-environment

How to define utility of a policy π in a given state s?

Problem total expected future reward can be infinite (summing total future expected reward diverges)

Two standard criteria

1. Maximize asymptotic *rate* of expected reward

$$\lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} R(time_i)$$

2. Maximize expected *discounted* reward

$$R(time_0) + \gamma R(time_1) + \gamma^2 R(time_2) + \gamma^3 R(time_3) + \cdots$$

   for a discount factor $\gamma < 1$

We will use discounted reward, because it is actually much easier

18.3 Value function

$$\begin{aligned} V_\pi(s) &= \text{expected discounted reward of following } \pi \text{ from } s \\ &= R(s) + \gamma \sum_{s'} \mathrm{P}(s'|s, \pi(s))\; V_\pi(s') \end{aligned}$$

Objective

Given $R(s)$, $\mathrm{P}(S'|s,a)$, γ
  Calculate π^* that maximizes $V_{\pi^*}(s)$ for all s

Two ways to do this ("generalized dynamic programming")

1. Policy iteration

2. Value iteration

18.4 Policy evaluation

Given a policy π, how to calculate value function $V_\pi(s)$?

Value function is defined to be

$$V_\pi(s) = R(s) + \gamma \sum_{s'} V_\pi(s') \ \mathrm{P}(s'|s, \pi(s))$$

Strategy 1 Solve linear system of equations (256 equations, 256 unknowns) for 256×1 vector V_π

$$V_\pi = R + \gamma \mathrm{P}_\pi V_\pi$$

Strategy 2 Iterative procedure

- Initialize V_π arbitrarily
- Iterate
$$V_\pi^{new}(s) = R(s) + \gamma \sum_{s'} V_\pi^{old}(s') \ \mathrm{P}(s'|s, \pi(s))$$
for each s
- Halt when V_π^{new} and V_π^{old} are sufficiently close

Guaranteed to converge to correct value function

18.5 Policy iteration

Iterative procedure to calculate optimal policy

- Initialize π arbitrarily and evaluate V_π
- Iterate
$$\begin{aligned} \pi^{new}(s) &= \arg\max_a \ R(s) + \gamma \sum_{s'} V_{\pi^{old}}(s') \ \mathrm{P}(s'|s, a) \\ &= \arg\max_a \ \sum_{s'} V_{\pi^{old}}(s') \ \mathrm{P}(s'|s, a) \end{aligned}$$
for each s
- Use policy evaluation to calculate $V_{\pi^{new}}$ for π^{new}
and repeat policy update
- Halt when $\pi^{new} = \pi^{old}$ (or more generally when $V_{\pi^{new}} = V_{\pi^{old}}$)

Guaranteed to converge to an optimal solution

18.6 Value iteration

Iterative procedure to calculate value function of optimal policy
(without explicitly calculating the optimal policy!)

A less obvious approach than policy iteration, but often works better

- Initialize V arbitrarily
- Iterate

$$V_{new}(s) \quad = \quad R(s) + \gamma \max_a \sum_{s'} V^{old}(s') \; \mathrm{P}(s'|s,a)$$

  for each s

- Halt when V^{new} and V^{old} are sufficiently close

Guaranteed to converge to the value function of an optimal policy
(but does not explicitly use optimal policy!)

Each iteration is cheaper than policy iteration because no policy evaluation is required, only value update

18.7 Recovering greedy policy

Given a value function V, the one-step greedy policy π for V can be recovered

$$\begin{aligned} \pi(s) \quad &= \quad \arg\max_a \quad R(s) + \gamma \sum_{s'} V(s') \; \mathrm{P}(s'|s,a) \\ &= \quad \arg\max_a \quad \sum_{s'} V(s') \; \mathrm{P}(s'|s,a) \end{aligned}$$

for each s

If V is the value function for an optimal policy then π guaranteed optimal

Readings

Russell and Norvig: Chapter 17