Markov Decision Processes (II)

University of BRISTOL

EMAT31530/Nov 2020/Xiaoyang Wang

Have a look at ...

... Russell and Norvig (Ch. 17 and Ch. 21)

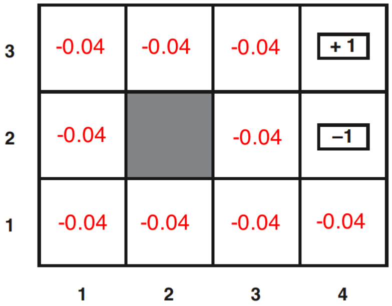
... Sutton and Barto. Reinforcement Learning: An Introduction. MIT press

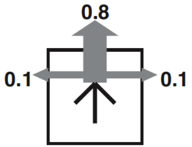
Outline

This lecture continues with the discussion on complex decision making. The objective is to present two alternatives to deal with Markov Decision Processes:

- Value iteration
- Policy iteration

Example: Stochastic Grid World





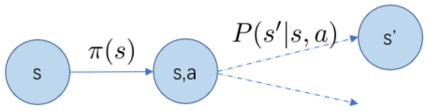
Value

Some remarks

- $R(s)$ is reward for being in s *now*: is the short term reward for being in s
- $U(s)$ is utility of the states that might follow s: $U(s)$ captures long term advantages from being in s
- $U(s)$ reflects what you can do from s; $R(s)$ does not.
- Value $V(s) = E[U(s)]$

Given π, $V_\pi(s) = ?$

Value



$$V_\pi(s) = \begin{cases} 0, & \text{if terminal state} \\ Q_\pi(s,a) & \end{cases}$$

$$Q_\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s') + \gamma V_\pi(s')]$$

Value of s given π is,

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

Bellman equations

There is a direct relationship between the value of a state and the value of its successor states.

Bellman equations (1957)

For V_π: Value of a state is the expectation of immediate reward plus the discounted successor state values

$$V_\pi(s) = E[R(s') + \gamma V_\pi(s')]$$

Optimal policy

Given $V_\pi(s)$, we can easily determine the optimal policy

$$\pi^*(s) = \arg\max_\pi V_\pi(s)$$

Optimal value function is

$$V^*(s) = \max_\pi V_\pi(s)$$

Bellman equations

$$V^*(s) = \max_{\pi} V_\pi(s)$$

Bellman optimality equation

For $V^*(s)$: the value of a state under an optimal policy must equal the expected utility for the best action from that state

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V^*(s')]$$

Value iteration: algorithm

For n states we have n Bellman equations with n unknowns (value of states)

Value iteration is an *iterative* approach to solving the n equations.

Intuition

We start with arbitrary values and update them as follows

$$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V(s')]$$

The algorithm converges to right and unique solution.

Value iteration: algorithm

Value Iteration, for estimating $\pi \approx \pi^*$

```
Parameter: threshold $\theta > 0$ determining accuracy of estimation                      (1)
Init $V(s), \forall s \in \mathcal{S}$ arbitrarily; $V(\text{terminal}) = 0$              (2)
Loop:                                                                                      (3)
    $\Delta \leftarrow 0$                                                                  (4)
    Loop for each $s \in \mathcal{S}$:                                                     (5)
        $v \leftarrow V(s)$                                                                (6)
        $V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V(s')]$                (7)
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$                                       (8)
until $\Delta < \theta$                                                                    (9)

Output $\pi$                                                                               (10)
$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V(s')]$                           (11)
```

Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states

0	0	0.8	0
0	▨	0.476	0
0	0	0.343	0.147

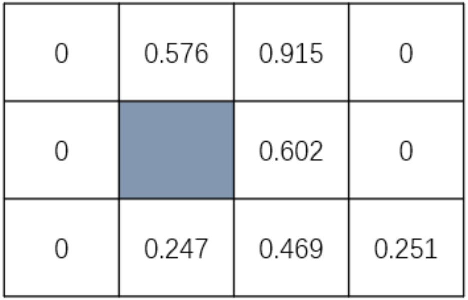
values after 1 iteration

Example: For $(3,3)$, calculate $\sum_{s'} P(s'|s,a)[R(s') + \gamma V(s')]$ for each action

- Up: $0.8 * 0 + 0.1 * 0 + 0.1 * 1 = 0.1$
- Down: $0.8 * 0 + 0.1 * 0 + 0.1 * 1 = 0.1$
- Right: $0.8 * 1 + 0.1 * 0 + 0.1 * 0 = 0.8$
- Left: $0.8 * 0 + 0.1 * 0 + 0.1 * 0 = 0$

Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



0	0.576	0.915	0
0	▨	0.602	0
0	0.247	0.469	0.251

values after 2 iterations

Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states

0.415	0.762	0.936	0
0.299	▨	0.628	0
0.237	0.382	0.509	0.289

values after 3 iterations

Value iteration: example

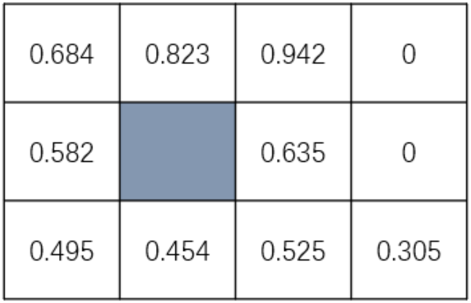
4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states

0.613	0.811	0.941	0
0.495	▨	0.634	0
0.412	0.435	0.522	0.302

values after 4 iterations

Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states

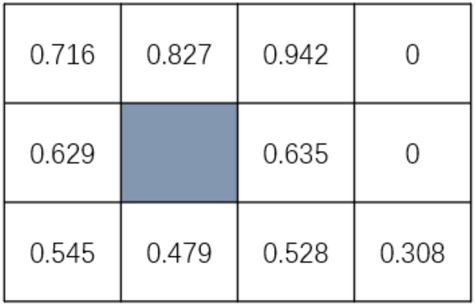


0.684	0.823	0.942	0
0.582	▨	0.635	0
0.495	0.454	0.525	0.305

values after 5 iterations

Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states

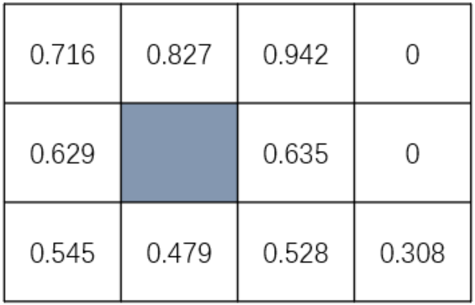


0.716	0.827	0.942	0
0.629	▨	0.635	0
0.545	0.479	0.528	0.308

values after 100 iterations

Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states

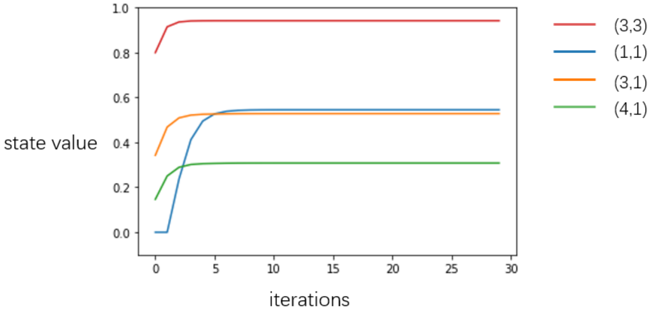


0.716	0.827	0.942	0
0.629	▨	0.635	0
0.545	0.479	0.528	0.308

values after 1000 iterations

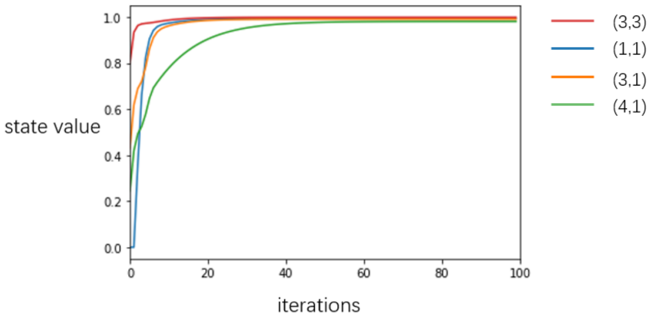
Value iteration: example

4x3 grid world with $\gamma = 0.9$ and $R(s) = 0$ for nonterminal states



Value iteration: example

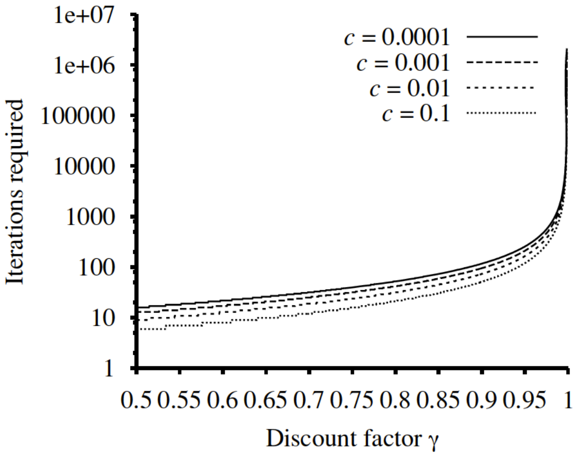
4x3 grid world with $\gamma = 0.999$ and $R(s) = 0$ for nonterminal states



Value iteration: algorithm

4x3 grid world with $R(s) = -0.04$ for nonterminal states

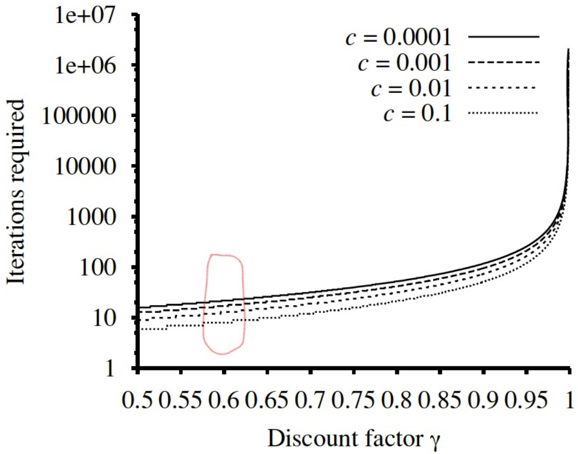
estimation accuracy $\theta = c \cdot R_{max}$



Value iteration: algorithm

4x3 grid world with $R(s) = -0.04$ for nonterminal states

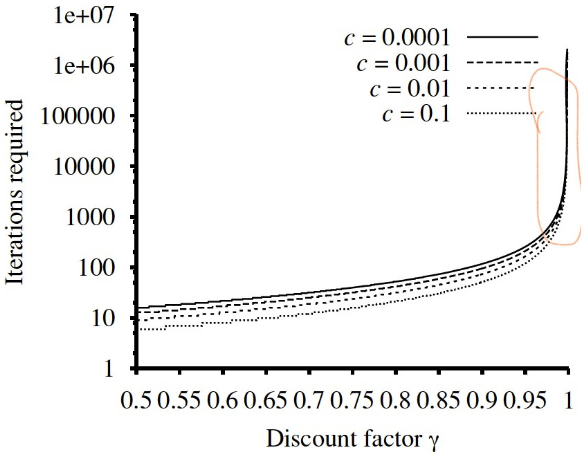
estimation accuracy $\theta = c \cdot R_{max}$



Value iteration: algorithm

4x3 grid world with $R(s) = -0.04$ for nonterminal states

estimation accuracy $\theta = c \cdot R_{max}$



Policy iteration

Idea: if one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.

Algorithm

Policy iteration alternates two steps, beginning from some initial policy π_0:

1. **Policy evaluation**: given a policy π_i, calculate $V_i = V_{\pi_i}$, the value of each state if π_i were to be executed.
2. **Policy improvement**: Calculate a new policy π_{i+1}, using one-step look-ahead based on V_i.

$$\begin{aligned}\pi_{\text{new}} &= \arg\max_a Q_\pi(s,a) \\ &= \arg\max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V_i(s')]\end{aligned}$$

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

Policy iteration: policy evaluation

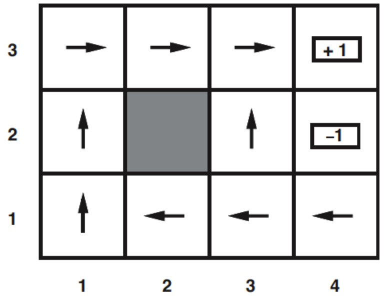
Policy improvement is easy, but policy evaluation?

Policy evaluation

- Bellman equation for $V_\pi(s)$
- At the ith iteration, the policy π_i specifies the action $\pi_i(s)$ in state s:

$$V_{\pi_i}(s) = \sum_{s'} P(s'|s,a)[R(s') + \gamma V_{\pi_i}(s')]$$

Policy iteration: example



E.g. $\pi_i(1,1) = Up$, $\pi_i(1,2) = Up$, ...

The simplified Bellman equations are

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

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

$$\cdots$$

Policy iteration: algorithm

Policy Iteration for estimating $\pi \approx \pi^*$

```
Init $V(s)$ and $\pi(s)$ arbitrarily for $s \in \mathcal{S}$
Repeat
    Policy evaluation using $V_\pi(s) = \sum_{s'} P(s'|s,a)[R(s') + \gamma V_\pi(s')]$
    Policy improvement using $\pi(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V(s')]$
Until policy is stable
Return $\pi$
```

Policy iteration vs Value iteration

The equations are now **linear**: the max operator has been removed.

For n states, we have n linear equations with n unknowns, which can be solved exactly in time $O(n^3)$ by standard linear algebra methods.

When to use Policy iteration?

- For small state spaces: **policy evaluation** using exact solution methods is often the most efficient approach, typically very fast and converges quickly.
- For large state spaces, $O(n^3)$ time might be prohibitive. **Value iteration** is preferred.
- For very large state spaces: use an **approximation** but optimality guarantee is lost.

Summary

- Bellman equations
- Value iteration
- Policy iteration

MDPs are great, if

... we know the state transition function $P(s'|a, s)$

... we know the reward function $R(s)$

But what if we don't?

Reinforcement Learning