MDP: Markov Decision Process

① "Actions are indeterminate"

"Action" in "Search": state A —Action a→ state B "Determinate"

Action in "MDP": state A —Action a→ { state B p=0.2; state C p=0.4; state D p=0.4 } "Indeterminate"

② "Cost function depends the starting and ending states of action"

MDP consists of

① a set of states *: what differs from "Search"

② a starting state

③* a set of actions modeled by a global probabilistic transition relation

④* Reward function T(s,a,s'): the probability of s —a→ s'

Reward(s,a,s'): the rewarding for s —a→ s'

⑤ Goal test: End(s)

⑥ A "discount factor": γ, $0 \le \gamma \le 1$. "How much we value the future"

Transition probabilities $T(s,a,s')$

{ 1. if $T(s,a,s') \neq 0$ then s' is a possible successor of s
2. $\sum_{s'} T(s,a,s') = 1$

Policy: A "mapping" from "state" to "action".

$\pi: S \to A$, $\pi(s_1)=a_1$, $\pi(s_2)=a_2$, ...

How to compare Policys: "$\pi(m_1)=m_1$ (?) $\pi(n_1)=n_2$"

Expected utilities $V_\pi(s)$

How to compute V_π?

1. By recursion.

$$V_\pi(s) = \begin{cases} 0, & \text{if } End(s) \\ \sum_{s'} T(s,\pi,s')\left[Reward(s,\pi,s') + \gamma V_\pi(s')\right], & \text{otherwise} \end{cases}$$

Q-value: $Q_\pi(s,a) = \sum_{s'} T(s,a,s')\left[Reward(s,a,s') + \gamma V_\pi(s')\right]$

2. By Iteration

Initialize $V_\pi^0(s) = 0$ for all s

for $t = 1 \dots$ max-iter:

  for each state $s \in S$:

    $V_\pi^t(s) = \sum_{s'} T(s,\pi(s),s')\left[Reward(s,\pi(s),s') + \gamma V_\pi^{t-1}(s')\right]$

Stop when V^t doesn't change much from V^{t-1}

Optimal Policy and Value function

Given a MDP, we want to know the optimal policy and optimal value function

π_{ope}: for every policy π' $V_{\pi_{ope}}(s) \ge V_{\pi'}(s)$ for all s

Hill Climbing Algorithm ⇒ optimal policy

Value iteration ⇒ optimal policy value

Initialize π to a random policy

For i=1 ... Max-iter:

  π = Successor(π) "A better one"

"How to make it better?"

$V_\pi(s) = Q(s,\pi(s))$ ("Q-value" $Q_\pi(s,a) = \sum_{s'} T(s,a,s')\left[Reward(s,a,s') + \gamma V_\pi(s')\right]$)

⇒ Find an action π', $V_{\pi'}(s) = \max_{a \in Action(s)} Q_\pi(s,a)$ ⇒ $V_{\pi'}(s) \ge V_\pi(s)$ since $V_{\pi'}(s)$ is maximum

⇒ π' is as good as π or better than π

⇒ Policy improvement: Given a π ⇒ $V_\pi(s)$, improve the policy to π_{new}

  Compute $Q_\pi(s,a)$ for every s and $a \in Action(s)$

  $\pi_{new}(s) = \arg\max_{a \in Action(s)} Q_\pi(s,a)$

Successor(π) = π_{new}

Policy iterations

$\pi_0 \xrightarrow{\text{policy evaluation}} V_{\pi_0} \xrightarrow{\text{policy improvement}} \pi_1 \xrightarrow{\text{policy evaluation}} V_{\pi_1} \xrightarrow{\text{policy improvement}} \pi_2 \cdots$

Initialization: generate $\pi(s) \in Action(s)$ randomly for all $s \in S$

stable ← false

while not stable:

  policy evaluation

  policy improvement

  if for all $s \in S$, $\pi(s)$ is unchanged:

    stable ← true

Value iteration:

Note: if we have π_{ope}, we can obviously compute $V_{\pi_{ope}}(s)$ for all s.

⇒ How can we compute $V_{\pi_{ope}}(s)$ directly (i.e. without knowing π_{ope})

{ $V_{ope}(s) = 0$ if end(s).
$V_{ope}(s) = \max_{a \in Action(s)} Q_{ope}(s,a)$.
$Q_{ope}(s,a) = \sum_{s'} T(s,\pi_{ope},s')\left[reward(s,\pi_{ope},s') + \gamma V_{\pi_{ope}}(s')\right]$

With recurrence above, we can compute $V_{\pi_{ope}}(s)$ (DP).

Initialize $V_{ope}^0(s) = 0$ for all s

for $t = 1 \dots$ Max-iter "Bellman equation"

$V_{\pi_{ope}}^t(s) = \max_{a \in Action(s)} \sum_{s'} T(s,\pi_{ope},s')\left[reward(s,\pi_{ope},s') + \gamma V_{\pi_{ope}}^{t-1}(s')\right]$

Here, At each step, we record the optimal action, then we can define the policy (policy extraction)

Convergence:

if either the discount factor $\gamma < 1$ or the MDP graph is acyclic

then Both value iterations and policy iteration Converge to the optimal solution

Reinforcement Learning

Goal: Learn a policy to maximize the rewards

Given: a set of actions to do in a state

Assuming: know the states, unlimited memory / computational resource

Agent —action→ environment; environment —state, reward→ Agent

Initialization

Loop:

Choose an action.

Observe the current state and record the reward.

Update parameters: (transition distribution ($T(s,a,s')$)

expected values, expected Q-values)

Exploration / Exploitation

(try something new) (do the best move at current state)

⇒ long-term benefit ⇒ greedy action

"How to balance them?"

Epsilon-Greedy

Epsilon ϵ: the probability of choosing to explore

$(1-\epsilon)$: the probability of choosing to exploit

Epsilon-Greedy

$$\pi(s) = \begin{cases} \arg\max_{a \in Action(s)} \hat{Q}(s,a) & \text{with } p = 1-\epsilon \quad \text{"exploit"} \\ \text{random } a \in Action(s) & \text{with } p = \epsilon \quad \text{"explore"} \end{cases}$$

Representation For Reinforcement learning

"How to update parameters depends on how they are represented"

{ Tabular representation: transition relations, rewards, values, Q-values, policies, All represented by tables

Approximation: Approximated by efficient functions such as neural networks.

Tabular representation

Main idea from Monte-Carlo Method: Average

Sample: $D = [s_1, a_1, r_1, s_2, a_2, r_2, s_3, \dots]$

{ $\hat{T}(s,a,s') = \frac{\#(s,a,s') \in D}{\sum_{s''} \#(s,a,s'') \in D}$

$\hat{Reward}(s,a,s') =$ average of $\{r : (s,a,r,s') \in D\}$.

$\hat{Reward}(s,a) =$ average of $\{r : (s,a,r) \in D\}$

Estimate a Model ⇒ optimal policy / value

"Why not we directly estimate Q-value?"

Utility at s_i: $u_i = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} \dots$

$\hat{Q}_\pi(s,a) =$ average of $\{u_t \mid s_t = s, a_t = a\}$

Temporal Difference

MC Method: requires an entire episode $[s_1, a_1, r_1, s_2, a_2, r_2, \dots]$

Update Q-value at each step (Q-learning)

$$\hat{Q}(s,A) \leftarrow (1-\mu)\hat{Q}(s,A) + \mu\left[r + \gamma \hat{V}(s')\right)$$

Current estimate of Q-value; Learning rate; Having a new transition (s, A, r, s')