Lecture 24: 18 April, 2024

Madhavan Mukund

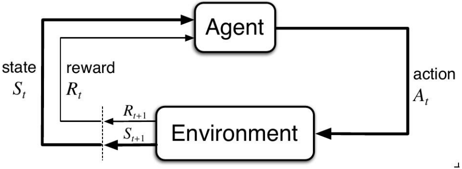
https://www.cmi.ac.in/~madhavan

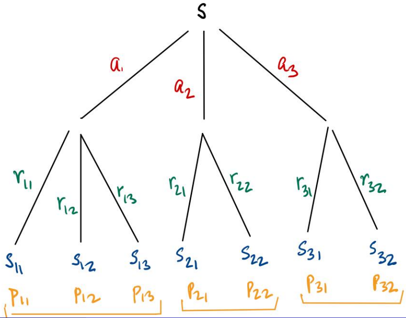
Data Mining and Machine Learning
January–April 2024

Markov Decision Processes

- Set of states S, actions A, rewards R
- At time t, agent in state S_t selects action A_t, moves to state S_{t+1} and receives reward R_{t+1}

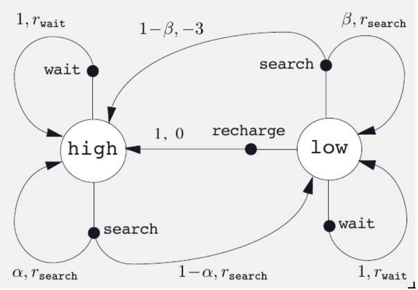
  Trajectory $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$
- Probabilistic transition function: $p(s', r \mid s, a)$
  - Probability of moving to state s' with reward r if we choose a at s
  - For each (s, a), $\sum_{s'} \sum_r p(s', r \mid s, a) = 1$
  - Backup diagram
- Typically assume finite MDPs — S, A and R are finite





MDP Example: Robot that collects empty cans

- State — battery charge: high, low
- Actions: search for a can, wait for someone to bring can, recharge battery
  - No recharge when high
- α, β, probabilities associated with change of battery state while searching
- 1 unit of reward per can collected
- $r_{search} > r_{wait}$ — cans collected while searching, waiting
- Negative reward for requiring rescue (low to high while searching)



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-

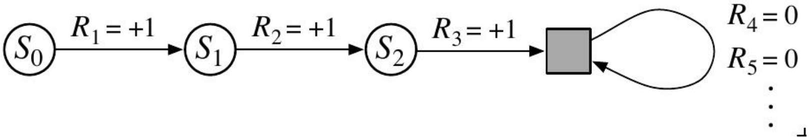
Long term rewards

- How do we formalize long term rewards?
- Assume that each trajectory is a finite episode
- Episode with T steps, expected reward at time t: $G_t \triangleq R_{t+1} + R_{t+2} + \cdots + R_T$
  - Each episode is independent: rewards are reset after each episode
- In some situations, trajectories may be (potentially) infinite
  - Discounted rewards: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, $0 \leq \gamma \leq 1$
- Inductive calculation of expected reward

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Long term rewards

- Can make all episodes infinite by adding a self-loop with reward 0



- Allow $\gamma = 1$ only if sum converges

- Alternatively, $G_t \triangleq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$,

  where we allow $T = \infty$ and $\gamma = 1$, but not both at the same time

- If $T = \infty$, $R_k = +1$ for each k, $\gamma < 1$, then $G_t = \dfrac{1}{1-\gamma}$

Policies and value functions

- A policy π describes how the agent chooses actions at a state
  - $\pi(a \mid s)$ — probability of choosing a in state s, $\sum_a \pi(a \mid s) = 1$
- State value function at s, following policy π

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

- Action value function on choosing a at s and then following policy π

$$q_\pi(s,a) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

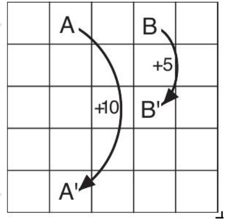
  - Note that $v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s,a)$
- Goal is to find an optimal policy, that maximizes state/action value at every state

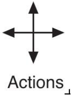
Bellman equation

- $$\begin{aligned} v_\pi(s) &\stackrel{\triangle}{=} \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right] \end{aligned}$$
- Bellman equation relates state value at s to state values at successors of s
- Value function v_π is unique solution to the equation

Gridworld Example

- Actions in each cell are {N,S,E,W}, with usual interpretation
- Reward is 0, except at boundaries
- Colliding with boundary — position unchanged, reward -1
- Special squares A and B — all four actions move as indicated, with rewards $+10$ and $+5$, respectively
- Policy π — choose each action with uniform probability 0.25
- Solving Bellman equations, we obtain v_π for each square
- Values at boundary are negative
- Value at A is less than 10 because next move takes agent to boundary square with negative value
- Value at B is more than 5 because next move is to a square with positive value





3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Optimal policies and value functions

- Compare policies π, π': $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for every state s
- Optimal policy π_*, $\pi_* \geq \pi$ for every π
  - Always exists, but may not be unique
- Optimal state value function, $v_*(s) \stackrel{\triangle}{=} \max_\pi v_\pi(s) = v_{\pi_*}(s)$
- Optimal action value function, $q_*(s, a) \stackrel{\triangle}{=} \max_\pi q_\pi(s, a) = q_{\pi_*}(s, a)$
- Bellman optimality equation for v_*

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s, a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')]
\end{aligned}
$$

Bellman optimality equations

- $$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')] \end{aligned}$$

- Likewise, for action value function

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = t, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \max_{a'} \gamma q_*(s', a')] \end{aligned}$$

- For finite state MDPs, can solve explicitly for v_*
  - n states, n equations in n unknowns, (assuming we know p)
- However, n is usually large, computationally infeasible
  - State space of a game like chess or Go
- Instead, we will explore iterative methods to approximate v_*