Markov Decision Process

Weizi Li

Department of Computer Science
University of Memphis

Outline

- Markov Property
- Markov Process
- Markov Reward Process
- Markov Decision Process

Markov Property

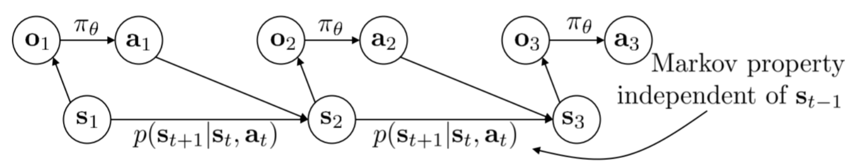
- Markov Property: current state completely characterizes all information from the history,

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \ldots, S_{t-1}, S_t].$$

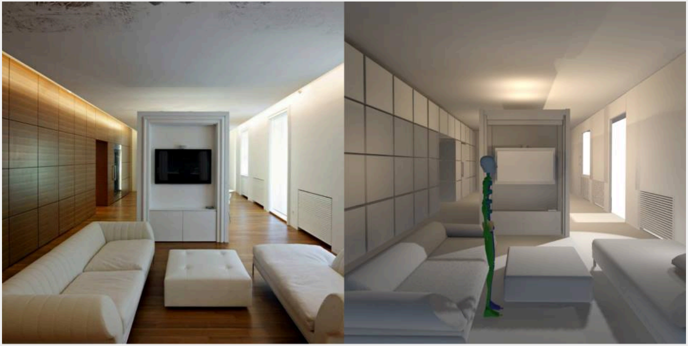
- The Markov state is a sufficient statistics of the future.

- The future is independent of the past given the present, e.g., playing a leftover game of Chess by someone else.

- Graphical model view



- In real-world applications
  - state: information of real objects
- In virtual/simulated environments
  - one level of abstraction
  - state: information of the 3D models of a subset of real objects



- In RL formulation
  - state: information of a subset of 3D models
  - observation: interpreted information from “sensor” readings of 3D models

Markov Process

- A Markov process is a memoryless random process, which consists of a sequence of random Markov states S_1, S_2, ...
- A Markov process (or Markov chain) is a tuple $\langle S, P \rangle$:
  - S: a finite set of states
  - P: state transition matrix

- Transition probability from a state s to its successor state s':

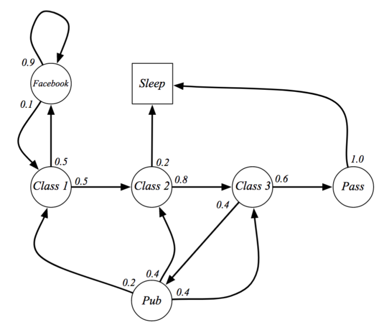
$$P_{ss'} = P[S_{t+1} = s' | S_t = s].$$

- State transition matrix P defines transition probabilities from all states to all successor states (each row sums to 1):

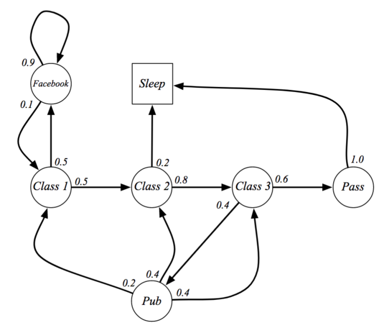
$$\begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \dots & \dots & \dots & \dots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{bmatrix}$$

Markov Process

	C1	C2	C3	Pass	Pub	FB	Sleep
C1		0.5				0.5	
C2			0.8				0.2
C3				0.6	0.4		
Pass							1.0
Pub	0.2	0.4	0.4				
FB	0.1					0.9	
Sleep							1

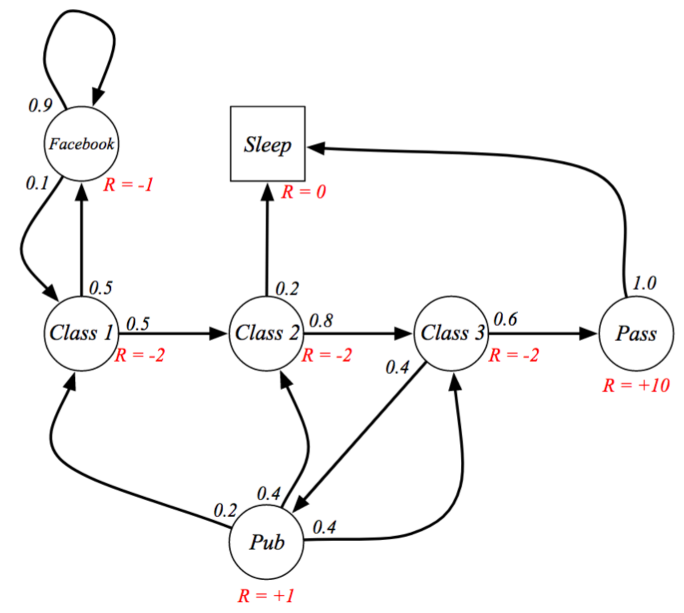


- Sleep is the *terminal state*
- Assumption: all trajectories/episodes are finite and end at the terminal state, e.g., C1, C2, C3, Pass, Sleep.



Markov Reward Process

- A Markov reward process is a Markov process with rewards, and can be represented by a tuple $\langle S, P, R, \gamma \rangle$:
  - S: a finite set of states
  - P: state transition matrix, $P_{ss'} = P[S_{t+1} = s' | S_t = s]$
  - R: reward function, $R_s = \mathbb{E}[R_t | S_t = s]$
  - γ: discount factor, $\gamma \in [0, 1]$


Facebook
R = -1
0.9
0.1
Sleep
R = 0
0.5
Class 1
R = -2
0.5
0.2
Class 2
R = -2
0.8
Class 3
R = -2
0.6
0.4
Pass
R = +10
1.0
0.2
0.4
0.4
Pub
R = +1

- The return G_t is the total discounted reward starting from timestep t:

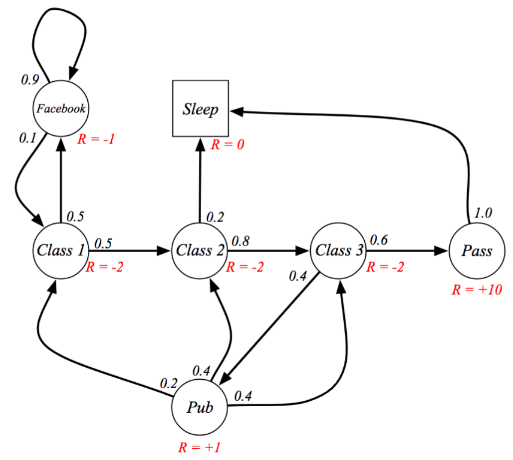
$$G_t \equiv R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i R_{t+i}$$

- Most Markov reward and decision processes are discounted:
  - Model uncertainty about the future (e.g., non-stationary environment).
  - Animal/human favor immediate rewards, as well as many real-world problems.
  - Mathematical convenience, i.e., for convergence.
- It is possible to use undiscounted Markov reward processes (i.e., $\gamma = 1$). Undiscounted rewards can still be used to solve MDP.

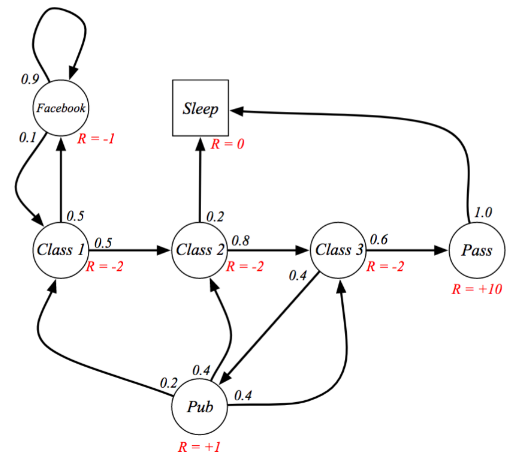
- The value function $V(s)$ gives the long-term, average value of state s.
- $V(s)$ of an MRP is the expected return starting from s:

$$
\begin{aligned}
V(s) &\equiv \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s]
\end{aligned}
$$

- $V(\mathrm{C1}) = \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = \mathrm{C1}]$, i.e., the average from all trajectories originated from C1.



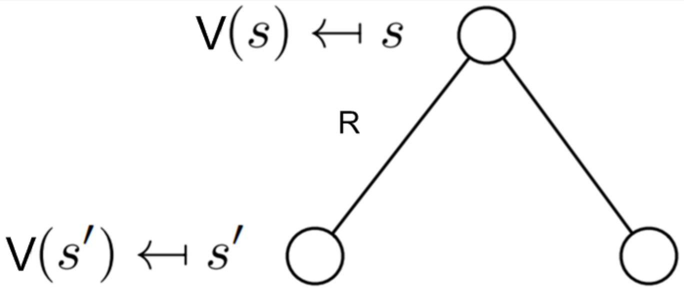
- Estimate from a single trajectory, e.g., trajectory: C1, C2, C3, Pass, Sleep, $\hat{V}(\mathrm{C1}) = -2 - 2\gamma - 2\gamma^2 + 10\gamma^3$.



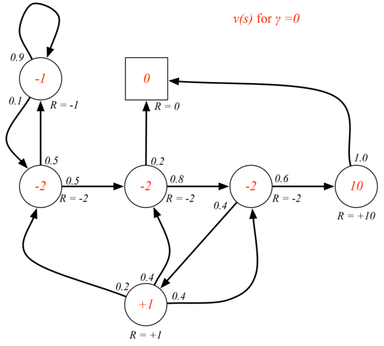
- The value function can be decomposed into two parts:
  - immediate reward R_t
  - discounted value of successor state $\gamma V(S_{t+1})$

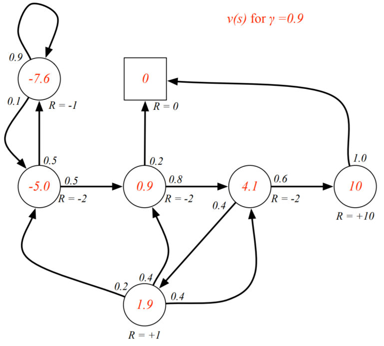
$$\begin{aligned} V(s) &\equiv \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots |S_t = s] \\ &= \mathbb{E}[R_t + \gamma (R_{t+1} + \gamma R_{t+2} + \dots)|S_t = s] \\ &= \mathbb{E}[R_t + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_t + \gamma V(S_{t+1})|S_t = s] \end{aligned}$$

$$V(s) = \mathbb{E}[R_t + \gamma V(S_{t+1}) | S_t = s] = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s')$$



- Example 1 ($\gamma = 0$): $-2 = -2 + 0(\ldots)$
- Example 2 ($\gamma = 0.9$): $4.1 = -2 + 0.9(0.6 * 10 + 0.4 * 1.9)$





- Bellman expectation equation is linear and can be solved using either direct method or iterative method.
- Direct method (i.e., the normal equation) has $O(n^3)$ complexity thus only works for small MRPs:

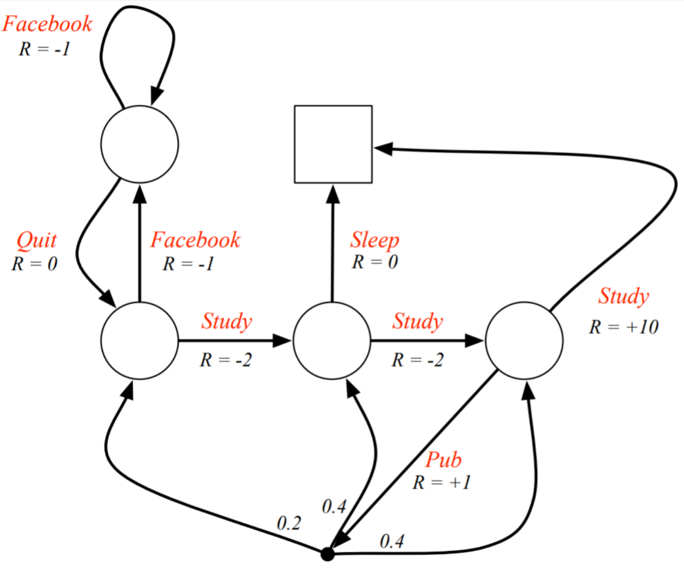
$$\begin{aligned} V &= R + \gamma P V \\ (1 - \gamma P) V &= R \\ V &= (1 - \gamma P)^{-1} R \end{aligned}$$

- Iterative methods for large MRPs:
    - Dynamic Programming
    - Monte-Carlo Evaluation
    - Temporal-Difference Learning

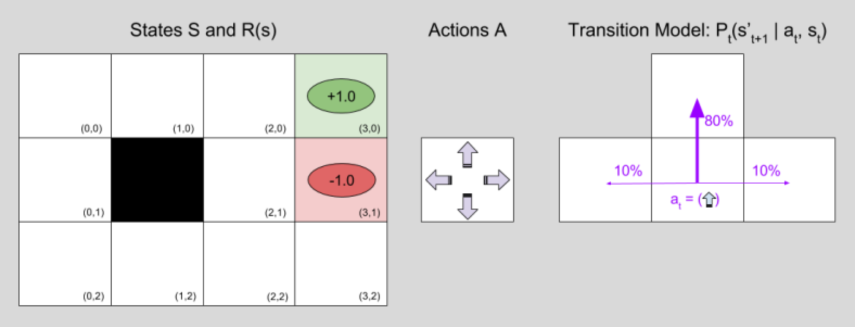
Markov Decision Process

- MDP is a Markov reward process with actions, and can be represented as a tuple $\langle S, A, P, R, \gamma \rangle$:
  - S: a finite set of states
  - A: a finite set of actions
  - P: state transition matrix, $P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$
  - R: reward function, $R^a_s = \mathbb{E}[R_t | S_t = s, A_t = a]$ (or R_s, $R^a_{ss'}$)
  - γ: discount factor, $\gamma \in [0, 1]$

Markov Decision Process



Markov Decision Process



- Classical formulation of sequential decision making.
- Almost all RL problems can be formulated as MDPs:
  - bandits $\rightarrow$ MDPs with one state
  - partially observable problems $\rightarrow$ POMDPs
  - optimal control $\rightarrow$ continuous MDPs
- We will focus on finite MDPs (states, actions, and rewards are finite).

- A policy defines the behavior of an agent by mapping states to actions.
- A (stochastic) policy is a distribution over actions given a state $\pi(a|s) = P[A_t = a|S_t = s]$.
- We will focus on stationary (time-independent) polices.

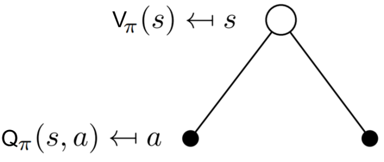
- The *state-value function* of an MDP is the expected return starting from state s, and then following policy π:

$$V^\pi(s) \equiv \mathbb{E}_\pi[G_t | S_t = s].$$

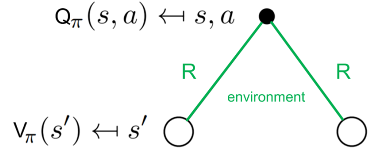
- The *action-value function* of an MDP is the expected return starting from state s, taking action a, and then following policy π:

$$Q^\pi(s, a) \equiv \mathbb{E}_\pi[G_t | S_t = s, A_t = a].$$

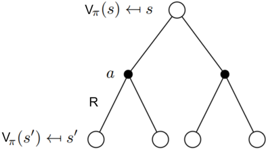
Value Function Connections



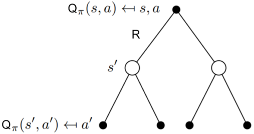
$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a)$$



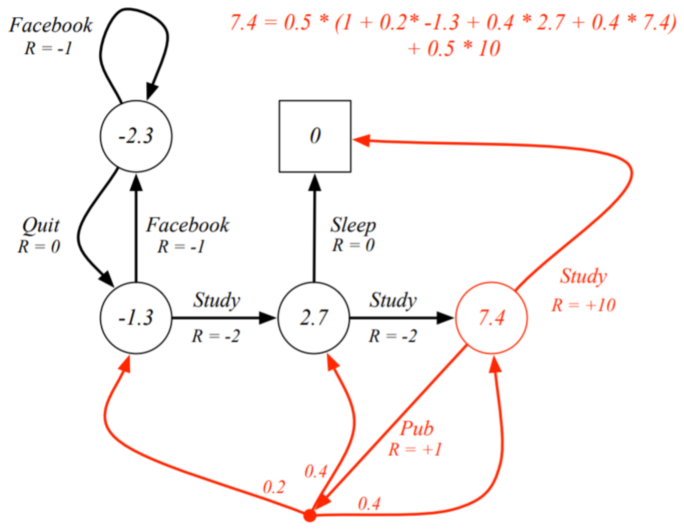
$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$



$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a')$$


7.4 = 0.5 * (1 + 0.2* -1.3 + 0.4 * 2.7 + 0.4 * 7.4)
+ 0.5 * 10
Facebook
R = -1
-2.3
0
Quit
R = 0
Facebook
R = -1
Sleep
R = 0
-1.3
Study
R = -2
2.7
Study
R = -2
7.4
Study
R = +10
Pub
R = +1
0.2
0.4
0.4

- The optimal *state-value function* gives the maximum value over all policies:

$$V^*(s) \equiv \max_\pi V^\pi(s).$$

- The optimal *action-value function* gives the maximum value over all policies:

$$Q^*(s,a) \equiv \max_\pi Q^\pi(s,a).$$

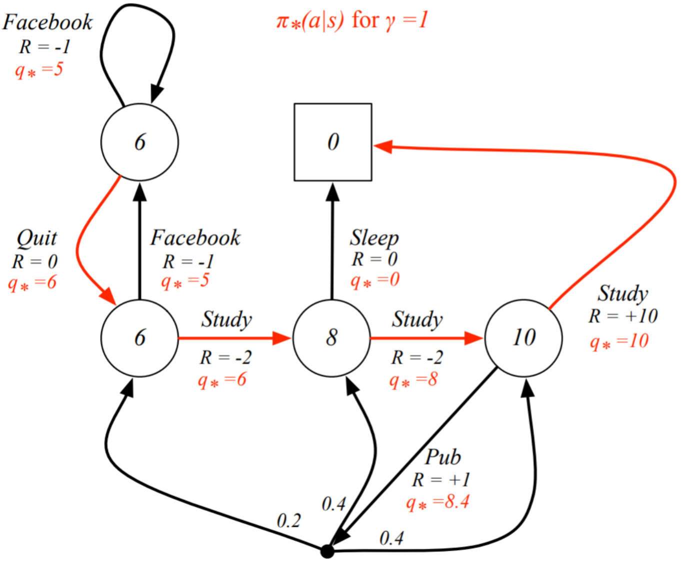
- The optimal value function specifies the best possible performance in the MDP.
- Optimal policy can be obtained by acting greedily to the optimal value function.

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s)$$

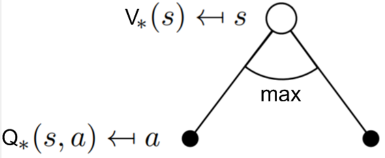
or

$$\pi^* \equiv \arg\max_{\pi} Q^{\pi}(s, a).$$

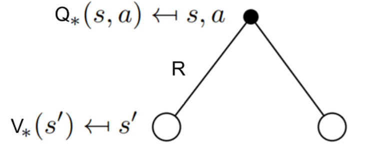
- Optimal policy allows the agent, from any initial state, to choose actions to maximize the cumulative reward.
- Optimal value function is unique, but optimal policy may not be unique (they all have the same optimal value function).


$\pi_*(a|s)$ for $\gamma = 1$
Facebook
R = -1
$q_* = 5$
6
0
Quit
R = 0
$q_* = 6$
Facebook
R = -1
$q_* = 5$
Sleep
R = 0
$q_* = 0$
Study
R = +10
$q_* = 10$
6
Study
R = -2
$q_* = 6$
8
Study
R = -2
$q_* = 8$
10
Pub
R = +1
$q_* = 8.4$
0.2
0.4
0.4

- Agent takes the action that corresponds to the maximum Q value. Then, the environment takes over.

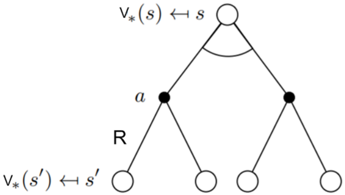


$$v_*(s) = \max_a Q_*(s,a)$$

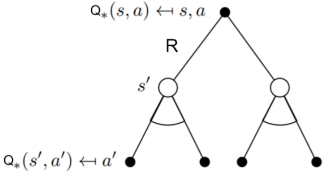


$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation



$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$



$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

- Bellman optimality equation is non-linear (due to the *max* operation).
- No direct method (closed-form solution) in general.
- Iterative methods:
  - Policy Iteration
  - Value Iteration
  - Q-learning
  - Sarsa

- Solving the Bellman optimality equation = finding the optimal value function (and hence optimal policy).
- Similar to an exhaustive search, looking ahead at all possibilities.
- Finding a solution requires:
  - dynamics of the environment, i.e., the model
  - sufficient computational resources, e.g., memory
  - Markov states
- Optimality helps us understand the theory.
- Real-world (interesting) tasks rarely satisfy all solution requirements.

- In practice, approximation is inevitable, even given the environment's dynamics (model).
- A critical aspect is the available computational power.
    - We need to compute for a single time step in order to solve the Bellman equations exactly.
    - In many cases, the number of states is too large to put into a table and the functions must be approximated using a more compact parameterized representation.
- Approximation can be sufficient when selecting suboptimal actions has little impact on the cumulative reward, e.g., "bad" moves made by AlphaGo do not affect its win.