Lecture 2: Bandit Problem and MDP

Lei Zhang

Outline

- Bandit problem
- Markov Process
- Markov Reward Process
- Markov Decision Process

*Materials of MDP are modified from David Silver's RL lecture notes

Outline

- Bandit problem
- Markov Process
- Markov Reward Process
- Markov Decision Process

Multi-Armed Bandit Problem

- You need to repeatedly take a choice among different options/actions
- You receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected
- Your objective is to maximize the expected total reward over some time period
- Invented in early 1950s by Robbins to model decision making under uncertainty when the environment is unknown

Multiple Slot Machines

- Each machine has a different distribution for rewards with unknown expectation
- Assume independence of successive plays and rewards across machines
- A policy is an algorithm that chooses the next machine to play based on the sequence of the past plays and obtained rewards

Questions

- If the expected reward from each machine is known, we are done: just pull the lever with the highest expected reward
- However, expected reward is unknown
- What should we do?
- Basic idea: estimate the expectation from the average of the reward received so far

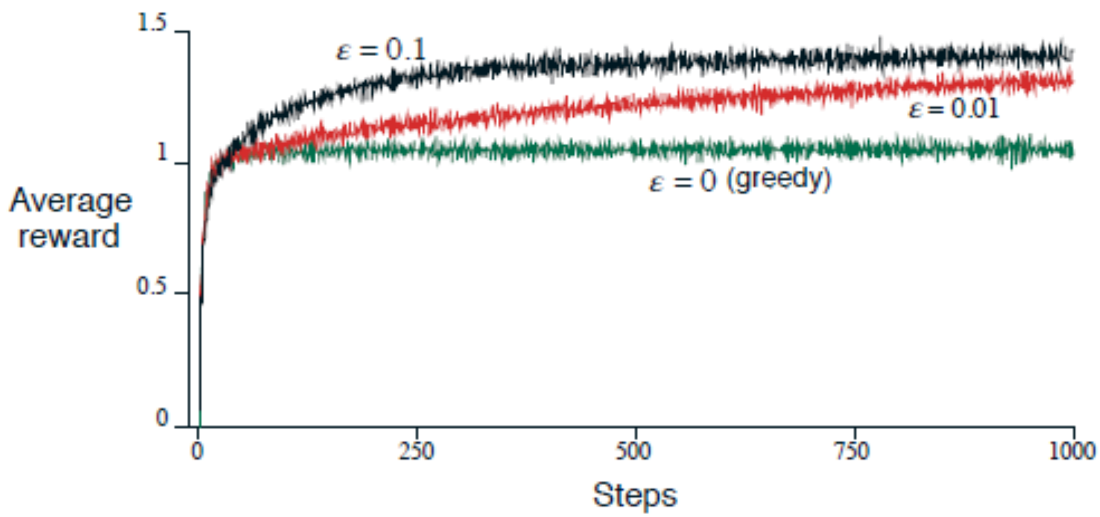
Action-Value Methods

- $Q_t(a)$: estimated value of action "*a*" at the "*t*-th" play
- K_a : times that action "*a*" has been chosen prior to *t*
- $R_1, R_2, \ldots, R_{K_a}$: reward received from each play
- Then, we define the action value as

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{K_a}}{K_a}$$

Policies

- Greedy policy
  - Always choose the machine with current best expected reward $Q_t(a)$
- ϵ-greedy policy
  - Choose machine with current best expected reward with probability $1 - \epsilon$
  - Choose a random machine with probability ϵ/n (total n machines)



Discussions

- The greedy method performs significantly worse in the long run because it often gets stuck performing suboptimal actions
- *Exploitation vs. exploration* dilemma:
  - Should you exploit the information you have learned or explore new options in the hope of greater payoff?
  - Still an on-going research topic in RL
  - Examples:
    - Restaurant selection: go to your favorite restaurant or try a new one?
    - Oil drilling: drill at the best known location or at a new location?
    - Online banner advertisements: show the most successful advert or show a different advert?

Softmax

- Another policy to balance exploration and exploitation
- Use Gibbs (or Boltzmann) distribution to choose action "*a*" at "*t*-th" play with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}}$$

- τ is a positive parameter called the *temperature*
  - *Large temp. → all (nearly) equiprobable selection*
  - *Small temp. → greedy action selection*

Algorithm Implementation

- In applications, do we need to save all rewards to compute Q_k ? *NO!*
- Incremental implementation to save memory: let Q_k be the average of its first $k-1$ rewards,

$$
\begin{aligned}
Q_{k+1} &= \frac{1}{k}\sum_{i=1}^{k} R_i \\
&= \frac{1}{k}\left(R_k + \sum_{i=1}^{k-1} R_i\right) \\
&= \frac{1}{k}\Big(R_k + kQ_k - Q_k\Big) \\
&= Q_k + \frac{1}{k}\Big[R_k - Q_k\Big],
\end{aligned}
$$

- Require memory only for k, Q_k, and the new reward
- The general form (frequently used in RL)

$$NewEstimate \leftarrow OldEstimate + StepSize \Big[Target - OldEstimate\Big]$$

Discussions on step size

- $\alpha_t(a)$: the step-size parameter of action "*a*" at time step "*t*"
- The following conditions (in stochastic approximation theory) are required to assure convergence of the incremental implementation with probability 1:

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty.$$

- Choose constant step size for nonstationary problem
  - Weight recent reward more heavily than long-past ones

Outline

- Bandit problem
- Markov Process
- Markov Reward Process
- Markov Decision Process

Markov Decision Process

- MDP formally describe an environment for reinforcement learning, where the environment is fully observable
- Almost all RL problems can be formalized as MDPs
  - Bandits are MDPs with one state
  - Optimal control primarily deals with continuous MDPs

Markov Property

- Definition: the future is independent of the past given the present

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

The state captures all relevant information from the history

Once the state is known, the history may be thrown away

i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

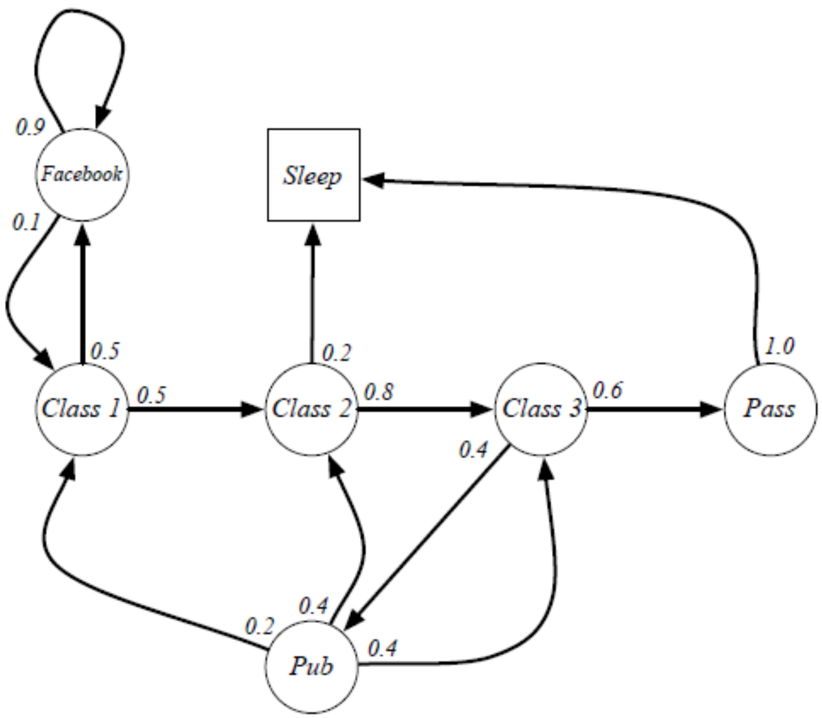
$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} \quad = \begin{array}{cc} & \textit{to} \\ \textit{from} & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

where each row of the matrix sums to 1.

Example: State Transition Matrix



$$
\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array}
\begin{array}{c}
\begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\
\left[\begin{array}{ccccccc}
 & 0.5 & & & & 0.5 & \\
 & & 0.8 & & & & 0.2 \\
 & & & 0.6 & 0.4 & & \\
 & & & & & & 1.0 \\
0.2 & 0.4 & 0.4 & & & & \\
0.1 & & & & & 0.9 & \\
 & & & & & & 1
\end{array}\right]
\end{array}
$$

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \ldots$ with the Markov property.

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$

Outline

- Bandit problem
- Markov Process
- Markov Reward Process
- Markov Decision Process

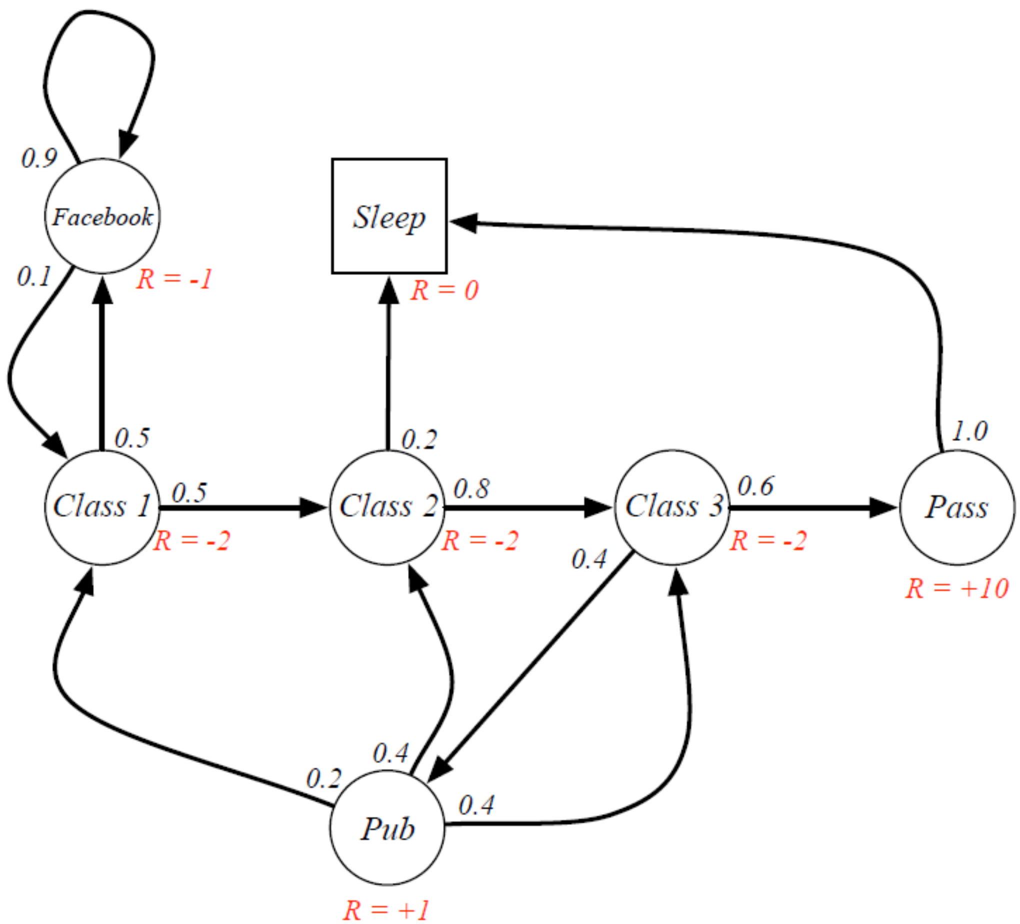
Markov Reward Process

- A Markov reward process is a Markov process with reward values

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Example: MRP



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

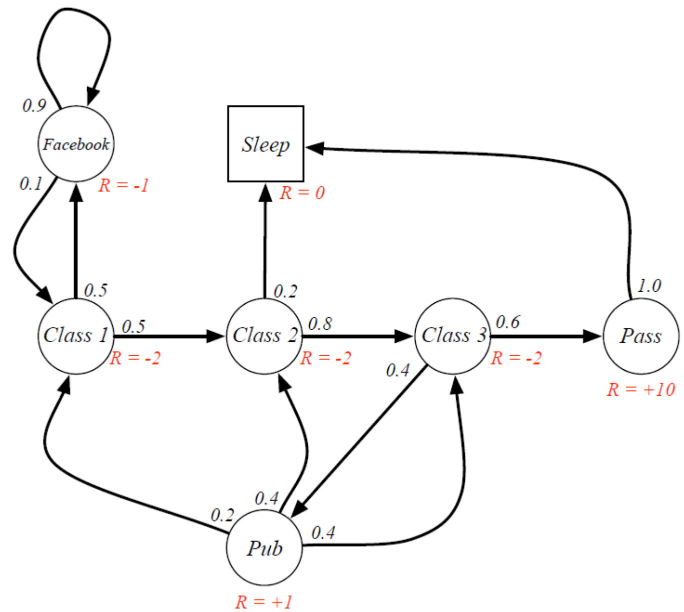
Return

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

State-Value Function



Sample returns for Student MRP:
Starting from $S_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

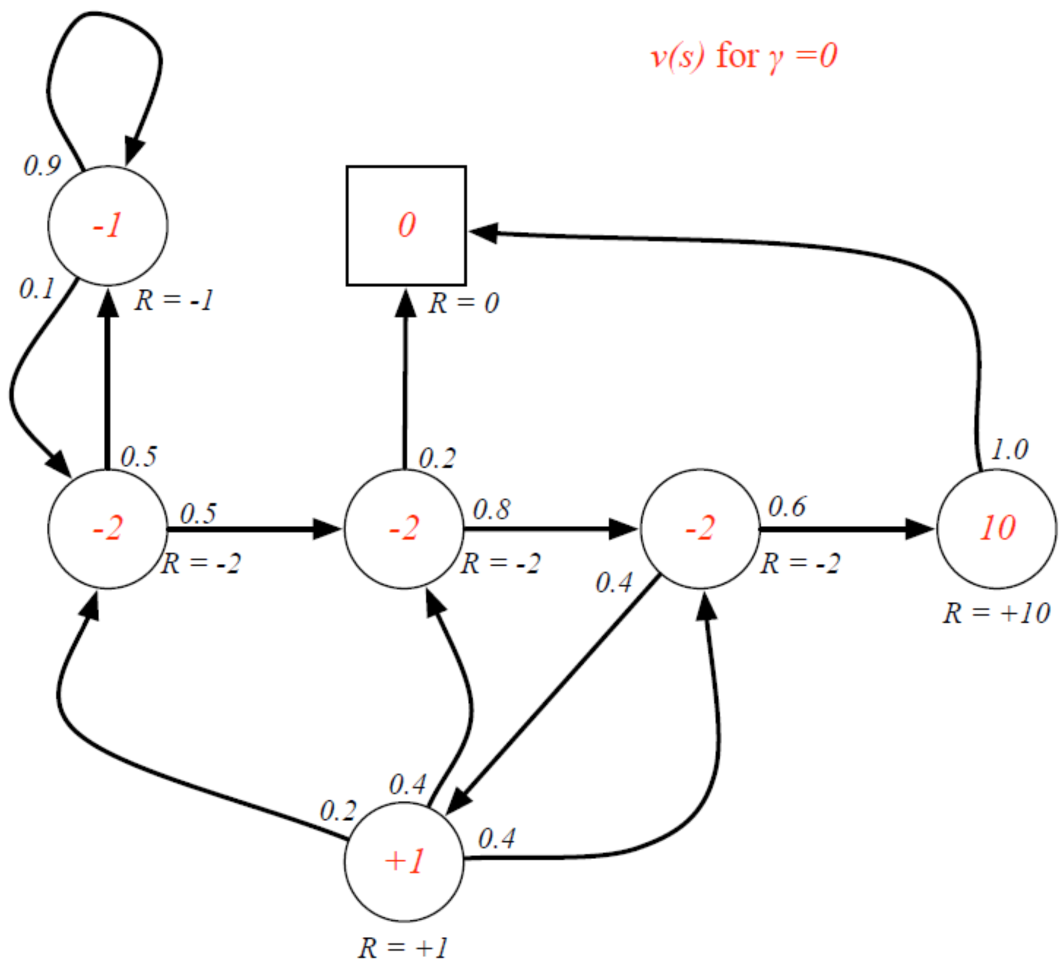
$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	$=$	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	$=$	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	$=$	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots$		
FB FB FB C1 C2 C3 Pub C2 Sleep		$=$	-3.20

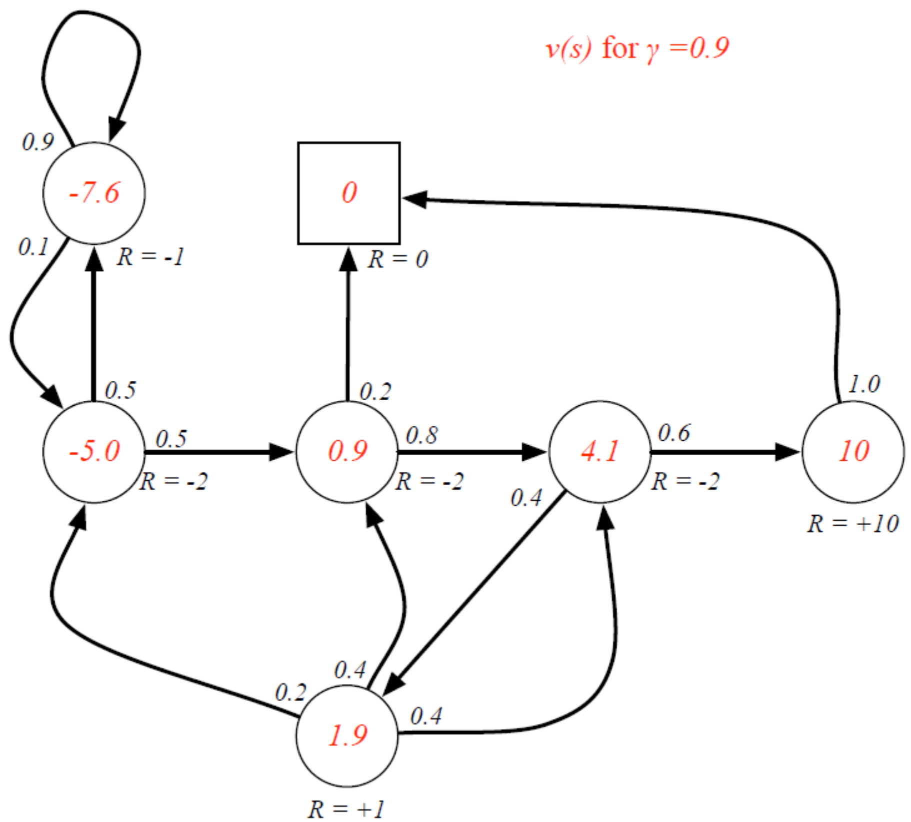
The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Example: state-value function



Example: state-value function



- A simple way to compute state-values ?

Bellman Equation for MRP

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + ...\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Bellman Equation: $v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$

Bellman Equation in Matrix Form

The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving Bellman Equation

The Bellman equation is a linear equation

It can be solved directly:

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(I - \gamma \mathcal{P})\, v &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

Computational complexity is $O(n^3)$ for n states

Direct solution only possible for small MRPs

There are many iterative methods for large MRPs, e.g.

- Dynamic programming
- Monte-Carlo evaluation
- Temporal-Difference learning

Example: Solving Bellman Equation

- Transition Matrix =

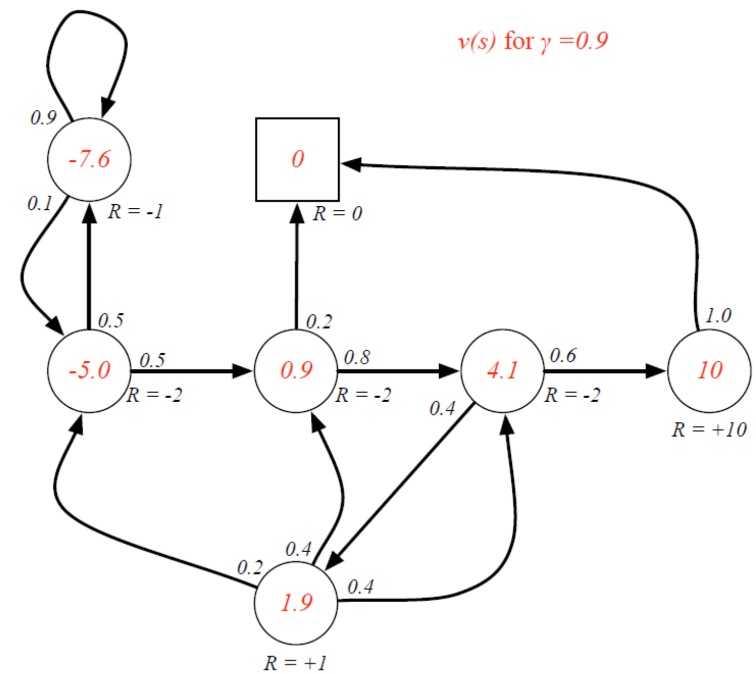
FB	0.9	0.1	0.	0.	0.	0.	0.
Class 1	0.5	0.	0.5	0.	0.	0.	0.
Class2	0.	0.	0.	0.8	0.	0.	0.2
Class 3	0.	0.	0.	0.	0.6	0.4	0.
Pass	0.	0.	0.	0.	0.	0.	1.
Pub	0.	0.2	0.4	0.4	0.	0.	0.
sleep	0.	0.	0.	0.	0.	0.	0.

- Reward Vector =

FB	-1
Class 1	-2
Class2	-2
Class 3	-2
Pass	10
Pub	1
sleep	0

- State Value =

FB	-7.63760843
Class 1	-5.01272891
Class2	0.9426553
Class 3	4.08702125
Pass	10.
Pub	1.90839235
sleep	0.



Outline

- Bandit problem
- Markov Process
- Markov Reward Process
- Markov Decision Process

MDP

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Policies

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)

  i.e. Policies are *stationary* (time-independent),
  $A_t \sim \pi(\cdot|S_t), \forall t > 0$

Policies

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^{\pi} \rangle$
- The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$

  where

$$\mathcal{P}^{\pi}_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^{a}_{s}$$

Value Functions for MDP

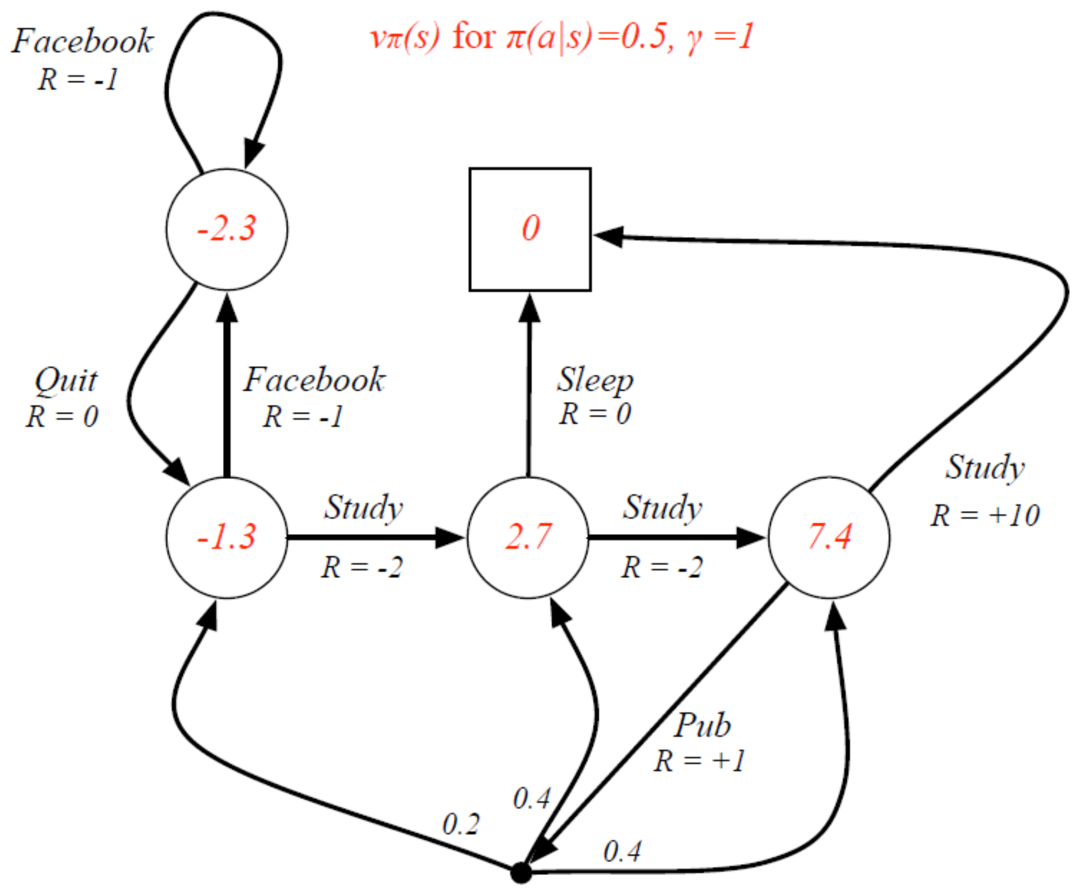
The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Example: State-Value Function for MDP



Bellman Expectation Equation (1)

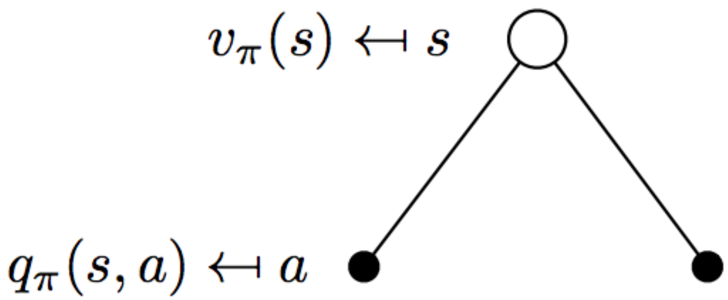
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

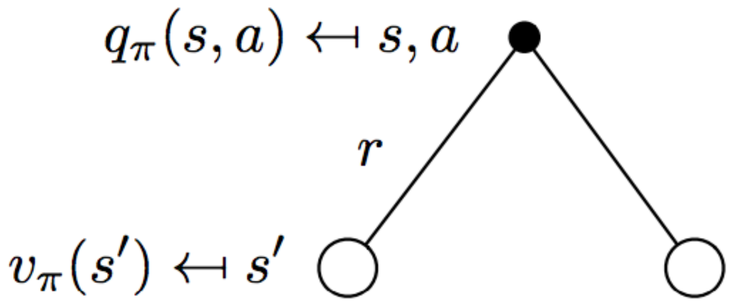
The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

Bellman Expectation Equation (2)

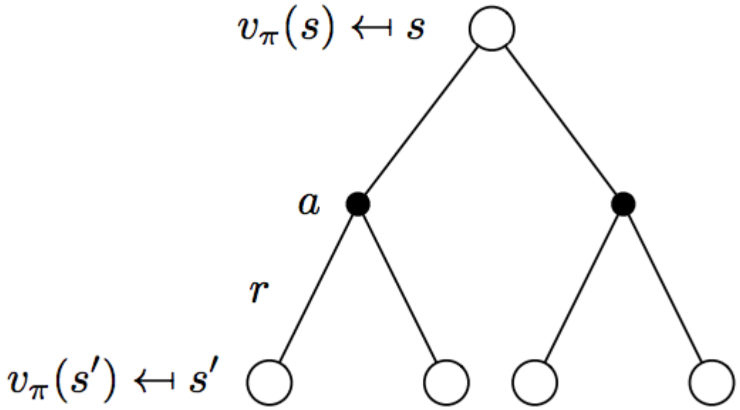


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

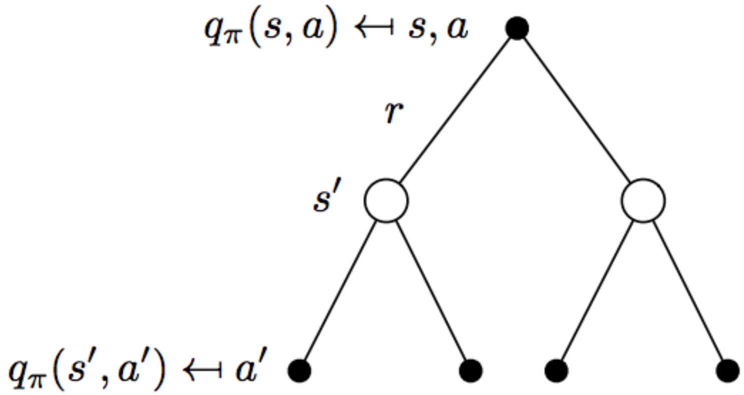


$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expectation Equation (3)

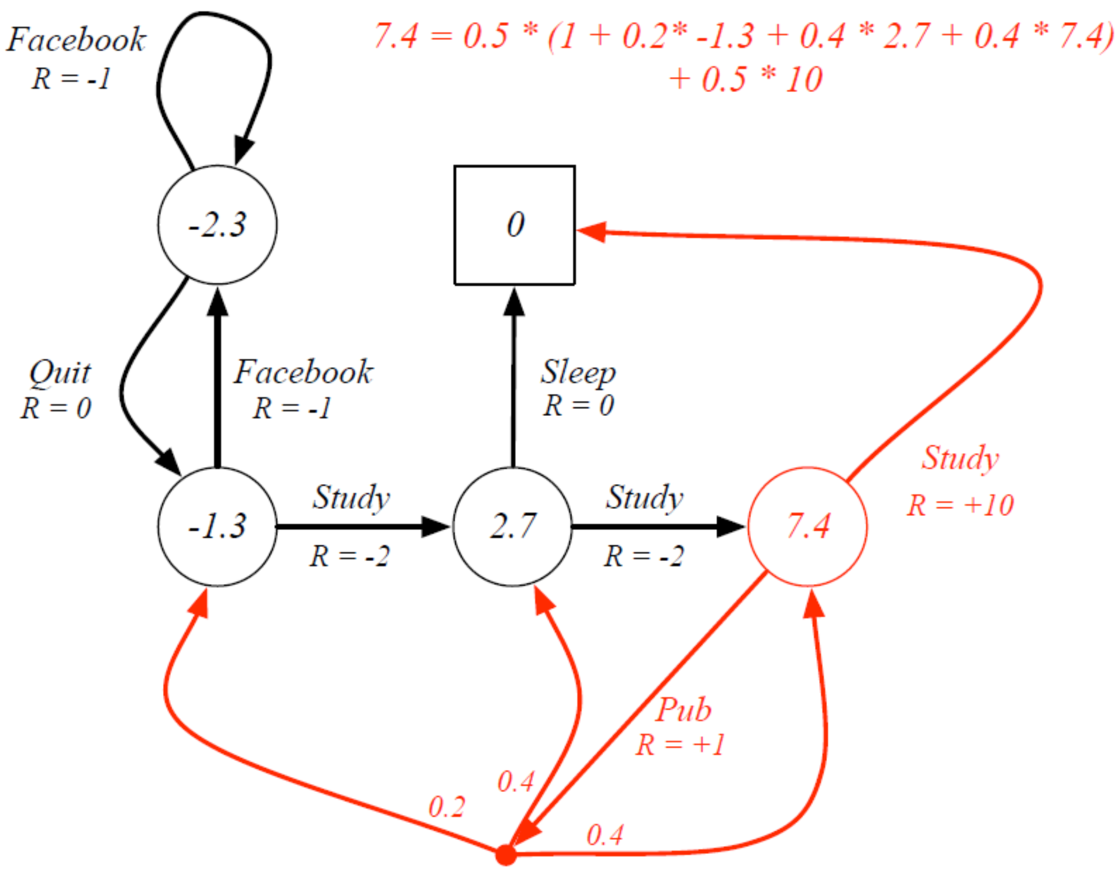


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$



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

Example: Bellman Expectation Equation for MDP



Bellman Expectation Equation in Matrix Form

The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

$$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

Optimal Value Function

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

For any Markov Decision Process

- *There exists an optimal policy* π_* *that is better than or equal to all other policies,* $\pi_* \geq \pi, \forall \pi$
- *All optimal policies achieve the optimal value function,* $v_{\pi_*}(s) = v_*(s)$
- *All optimal policies achieve the optimal action-value function,* $q_{\pi_*}(s, a) = q_*(s, a)$

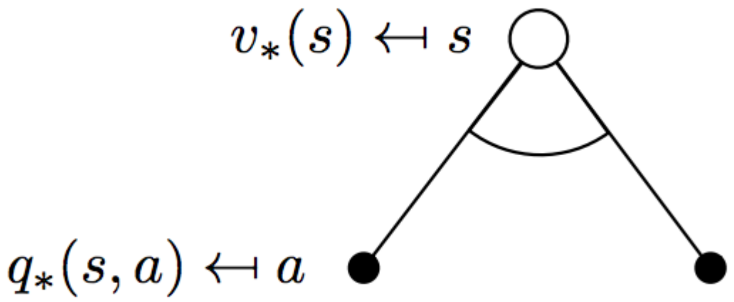
Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

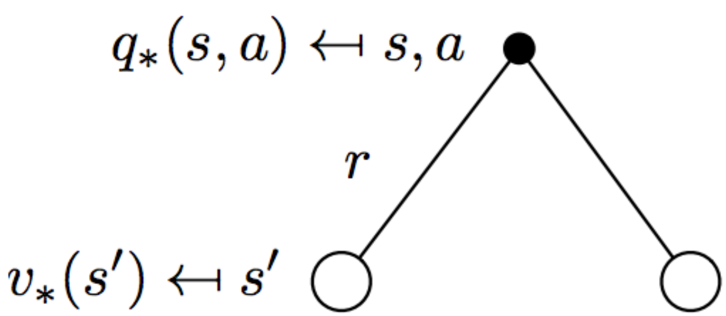
$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Bellman Optimality Equation (1)

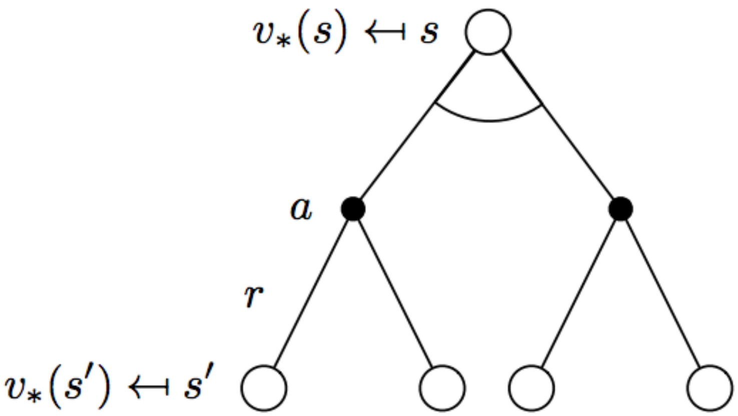


$$v_*(s) = \max_a q_*(s,a)$$

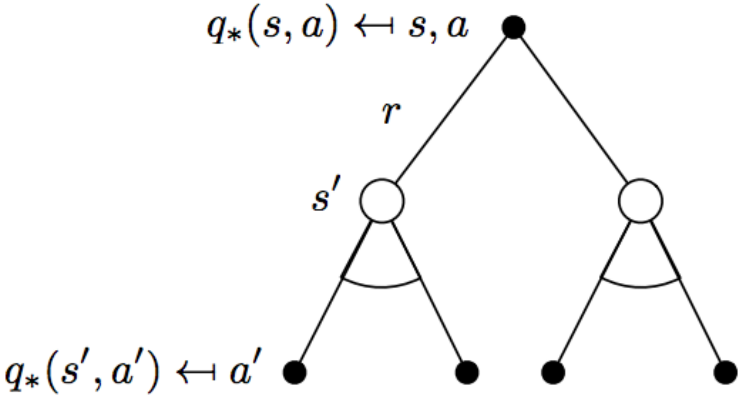


$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation (2)



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



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

Solving Bellman Optimality Equation

Bellman Optimality Equation is non-linear

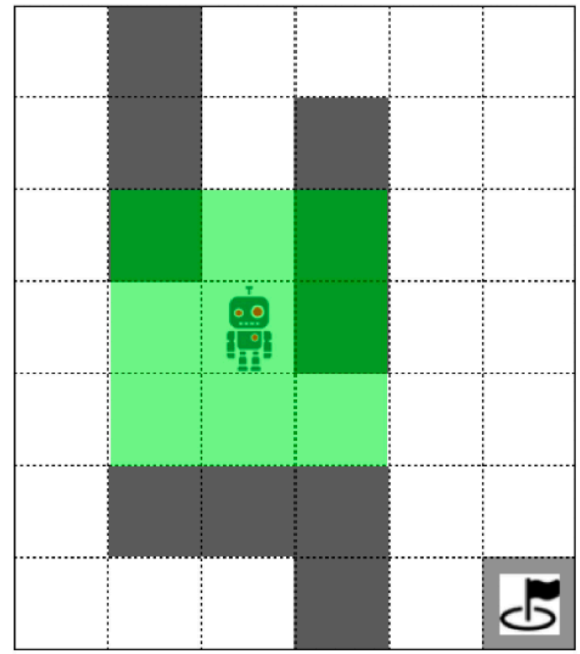
No closed form solution (in general)

Many iterative solution methods

- Value Iteration
- Policy Iteration
- Q-learning
- Sarsa

Extended MDPs

- Partially Observable MDP (POMDP)
  - Example: sailing at night w/o any navigation instruments
  - System dynamics are determined by an MDP, but the agent cannot directly observe the underlying state
  - Solved by introducing belief MDP (see wiki for more details)
- Continuous state MDP
  - Example: position, orientation and velocity of a car
  - Discretization on the continuous-state
  - Value function approximation



Given the green area as the only observable neighboring grids, the robot does not know the state, i.e., which grid on the map it is standing