强化学习原理及应用
Reinforcement Learning (RL): Theories & Applications

DCS3015 Autumn 2022

Chao Yu （余超）

School of Computer Science and Engineering
Sun Yat-Sen University

Lecture 5：强化学习-2

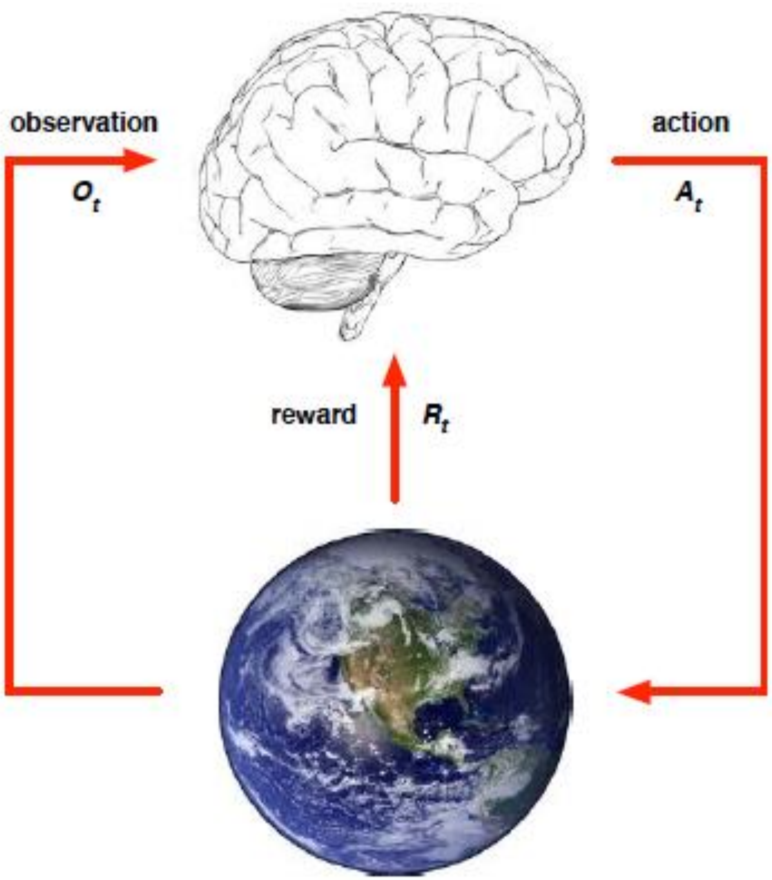
MDP and Dynamic Programming

Recap

- Last lecture:
  - What's RL?
  - Broad applications of RL
  - Why RL?

- This lecture:
  - The formal formulation of an RL problem as a MDP
  - Making good decisions given a MDP

The RL Problem

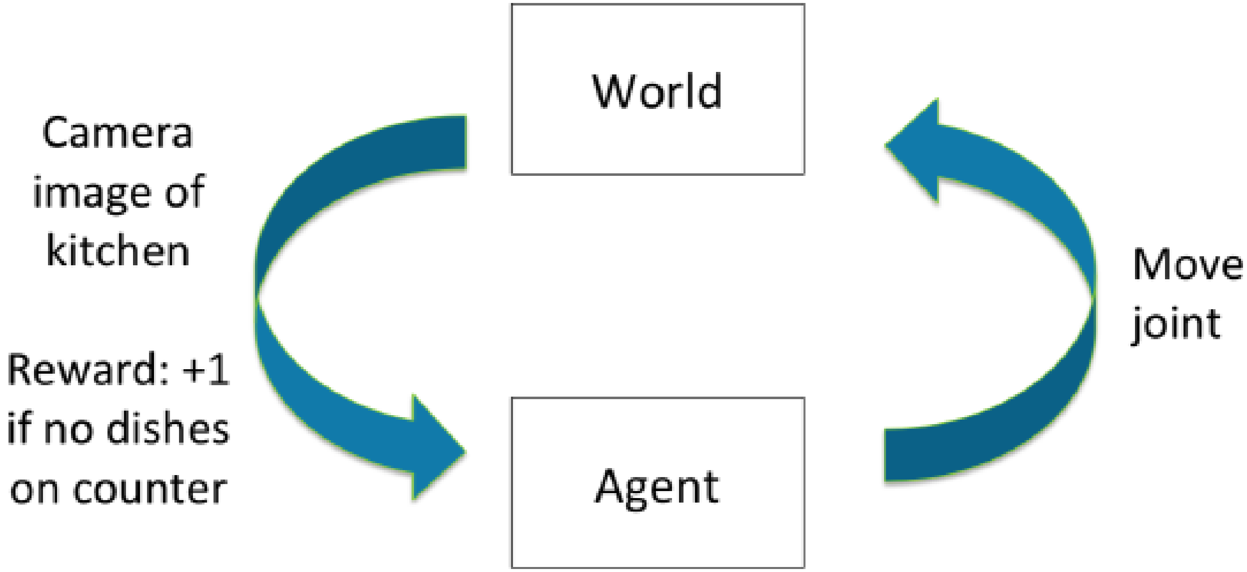


- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}

Goal: learn a policy (*i.e.*, a mapping from observations to actions) to maximise total future reward

Example of RL Problems

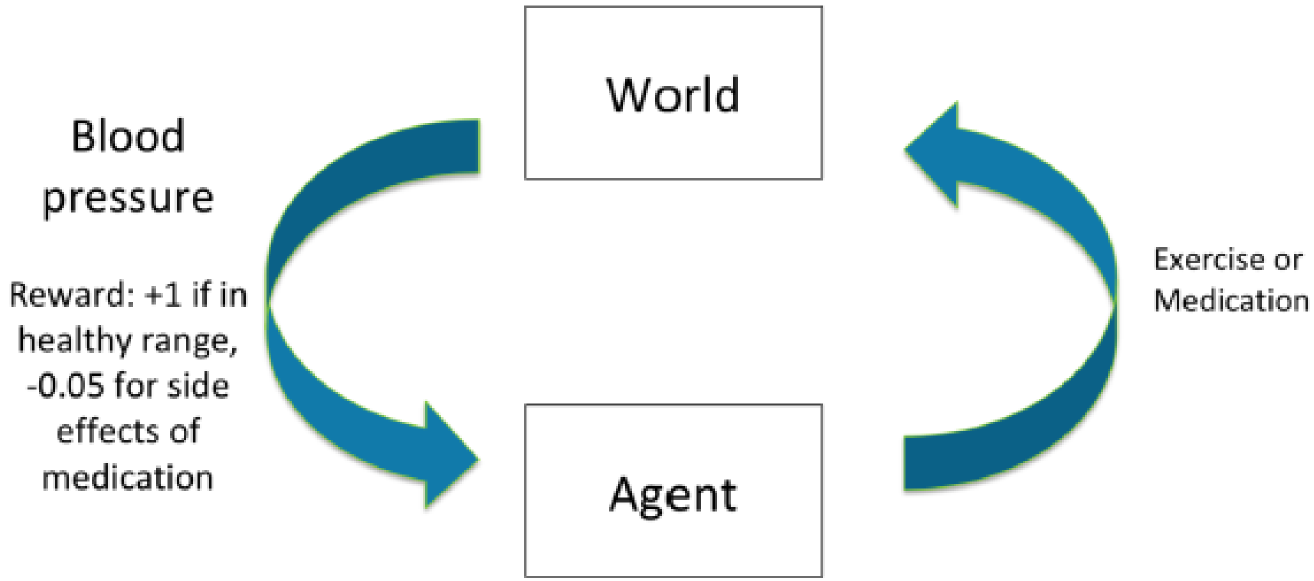
- Robot Unloading Dishwasher



Goal: learn a policy (*i.e.*, a mapping from observations to actions) to maximise total future reward

Example of RL Problems

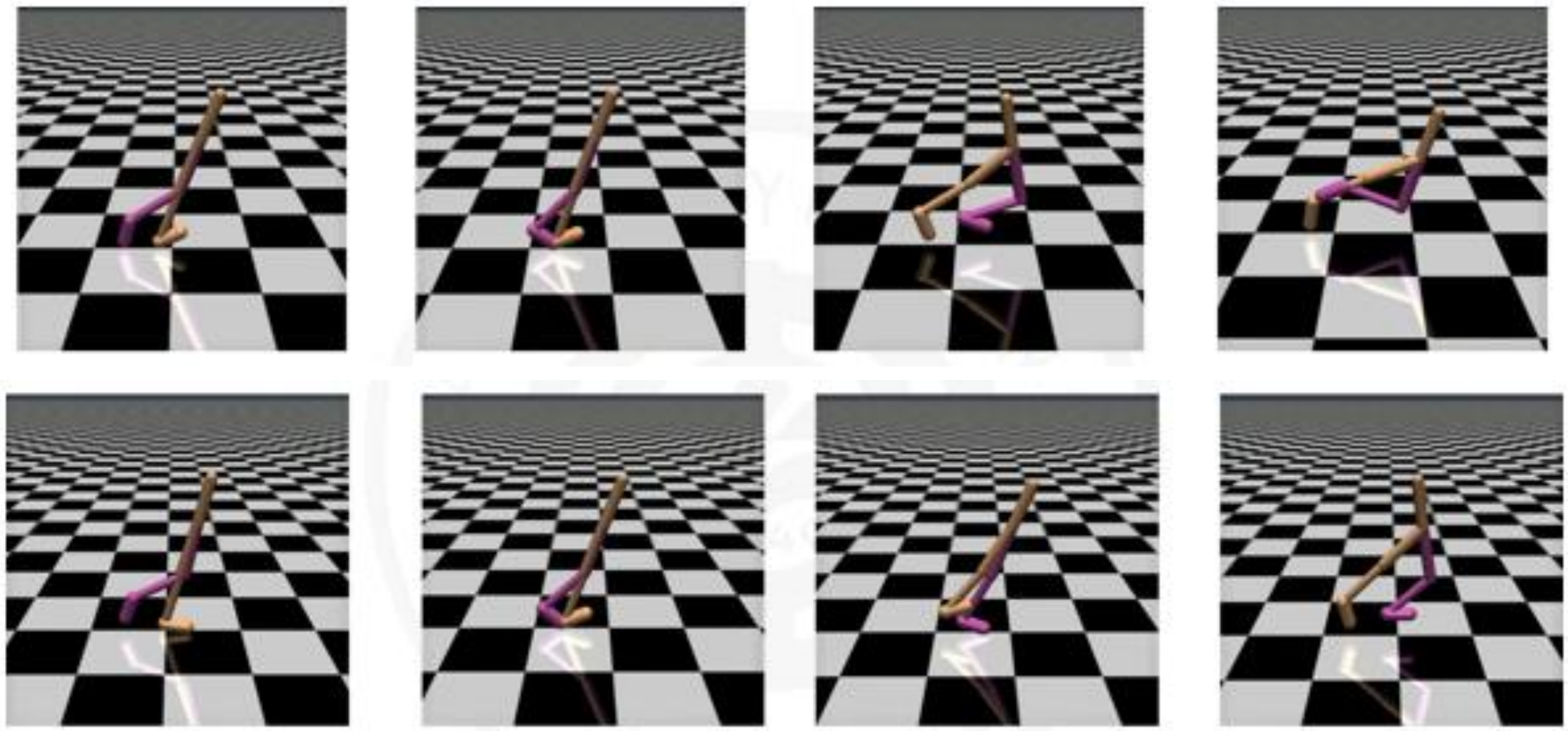
- Blood Pressure Control



Goal: learn a policy (*i.e.*, a mapping from observations to actions) to maximise total future reward

Example of RL Problems

- Robotic control



Goal: learn a policy (*i.e.*, a mapping from observations to actions) to maximise total future reward!

Elements of RL Problems - Reward

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward
- The goal reward and the intermediate reward
  - defeat the world champion at Go
    +1/-1 reward for winning/losing a game
  - Make a humanoid robot walk
    +1 reward for forward motion
    -1 reward for falling over
  - Manage an investment portfolio
    +v reward for each $ in bank
- Reward is the most fundamental component in RL
  - Where is reward from? How to design the best reward? How to address sparse reward problems?
  - Inverse RL, Hierarchical RL, Transfer RL, Knowledge-driven RL, etc.

Elements of RL Problems - State

- The history is the sequence of observations, actions, rewards
  - *i.e. all observable variables up to time t*
  - *i.e. the sensorimotor stream of a robot or embodied agent*
- State is the information used to determine what happens next
- The environment state is its private representation
  - *whatever data to pick the next observation/reward*
  - *not usually visible to the agent*
  - *May contain irrelevant information*
- The agent state is the agent's internal representation
  - *whatever information the agent uses to pick the next action*
  - *it is the information used by RL algorithms*
- An Markov state contains all useful information from the history, i.e., future is independent of past given present

A state S_t is Markov if and only if

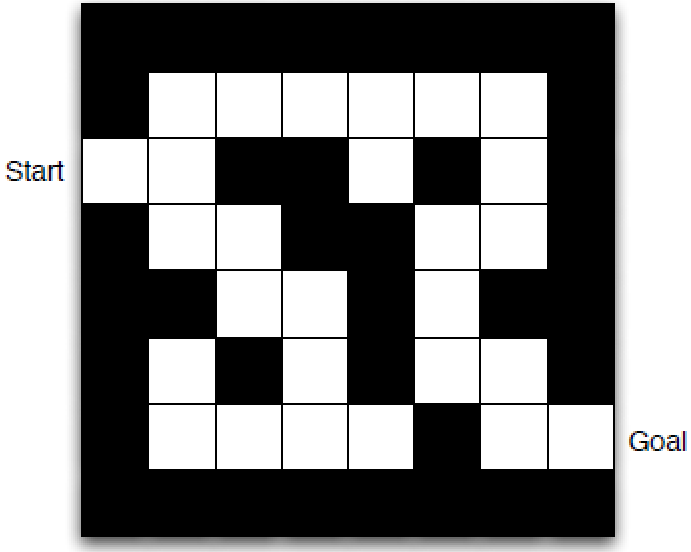
$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

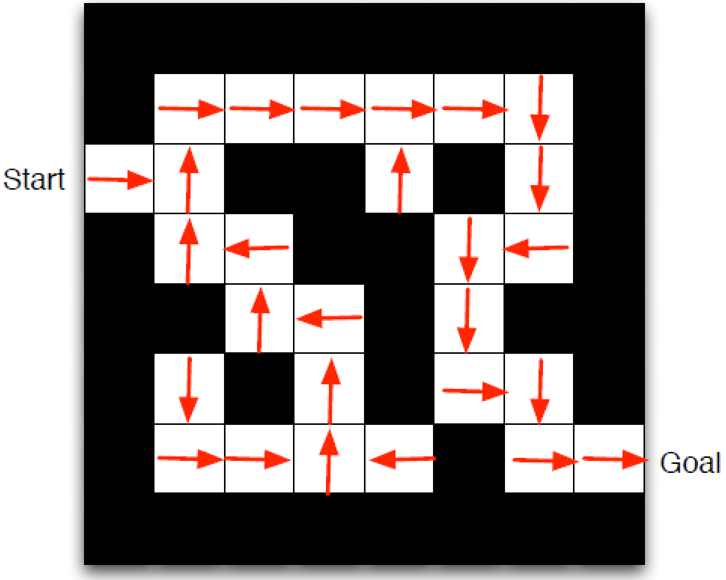
Elements of RL Problems - State

- Full observability: an agent directly observes environment state. Formally, this is a Markov decision process (MDP).
- Partial observability: an agent indirectly observes the environment, e.g.,:
  - *A robot with camera vision isn't told its absolute location*
  - *A trading agent only observes current prices*
  - *A poker playing agent only observes public cards*
  - *Formally, this is a partially observable Markov decision process (POMDP)*

Elements of RL Problems - Policy

- Policy: an agent's behaviour function, i.e., a mapping from state to action
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$





[from David Silver's course]

Elements of RL Problems - Policy

- **Policy**: an agent's behaviour function, i.e., a mapping from state to action
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

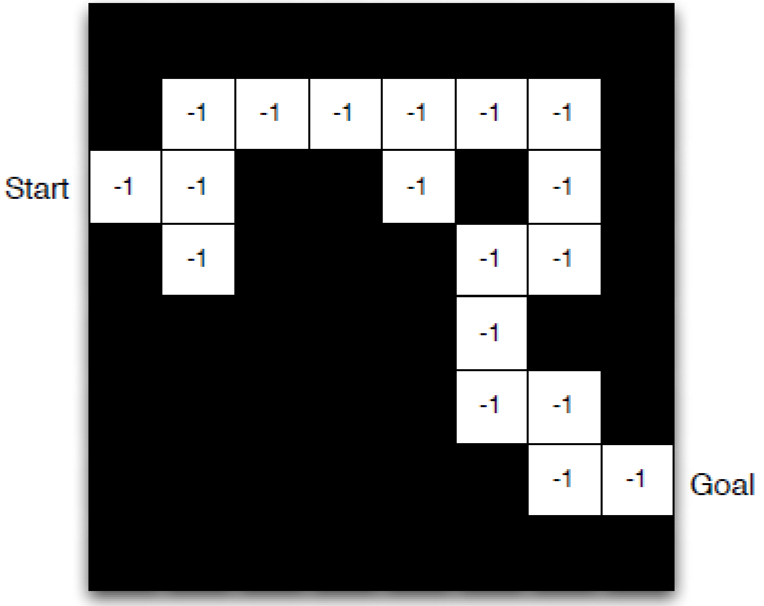
- For the Mars rover example [7 discrete states (location of rover); 2 actions: Left or Right], how many deterministic policies are there?

$$2 \;/\; 14 \;/\; 7^2 \;/\; 2^7$$

Elements of RL Problems - Model

- Model: A model predicts what the environment will do next, i.e., agent's representation of the environment
- P predicts the next state
- R predicts the next (immediate) reward

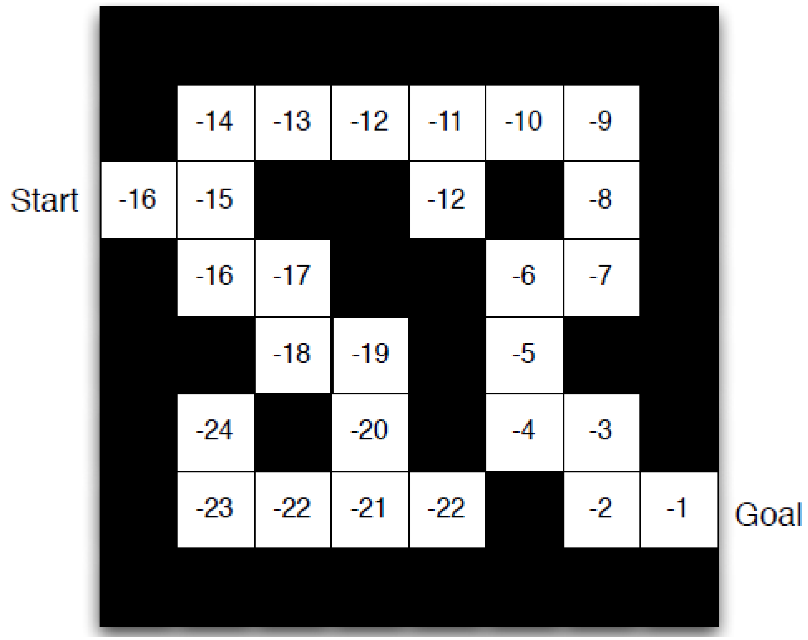
$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$



[from David Silver's course]

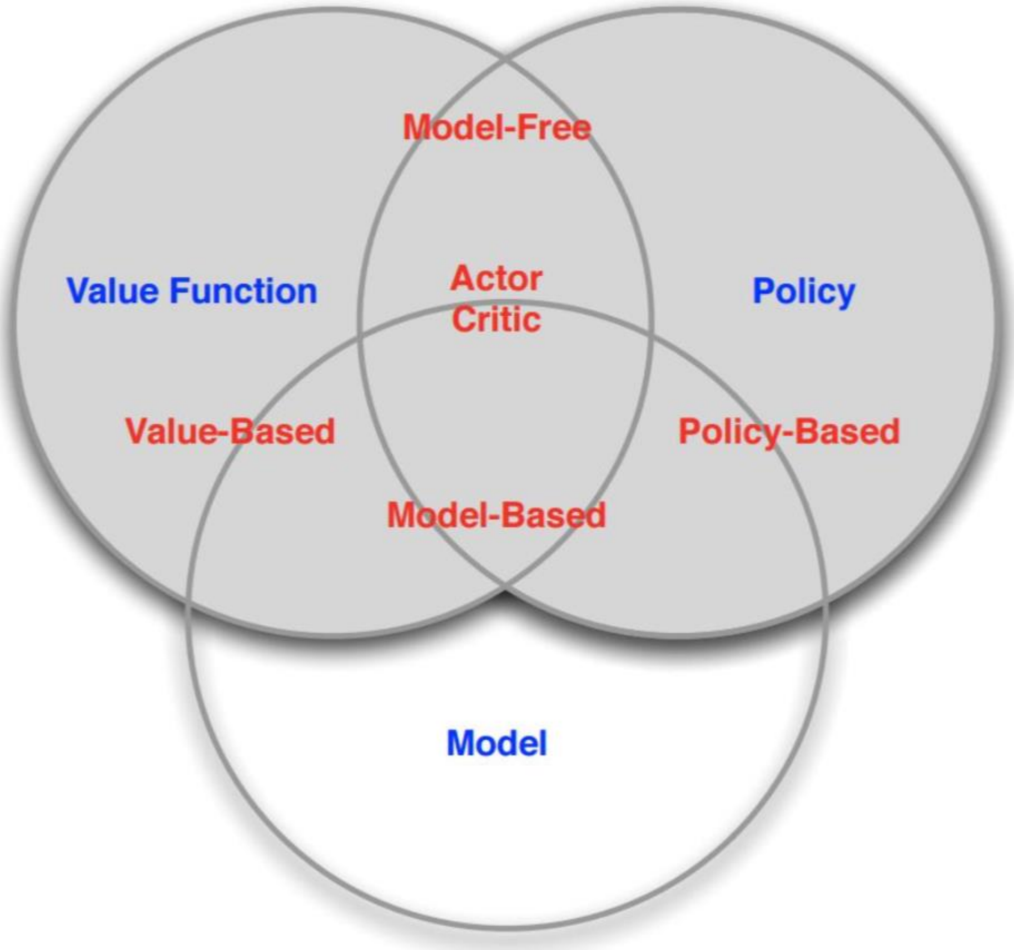
- Value functions: how good is each state and/or action
  - *Value function is a prediction of future reward*
  - *Used to evaluate the goodness/badness of states*
  - *And therefore used to select between actions*

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$$



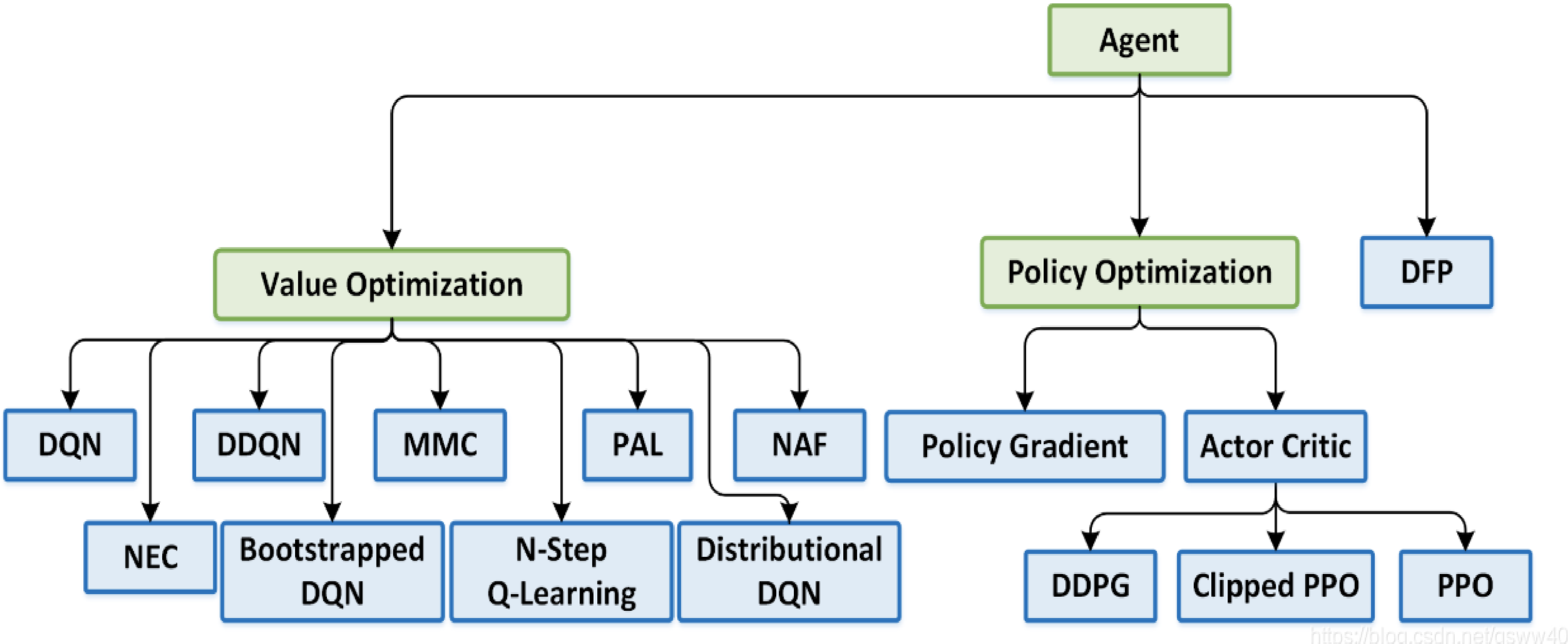
[from David Silver's course]

Categorizing RL Algorithms



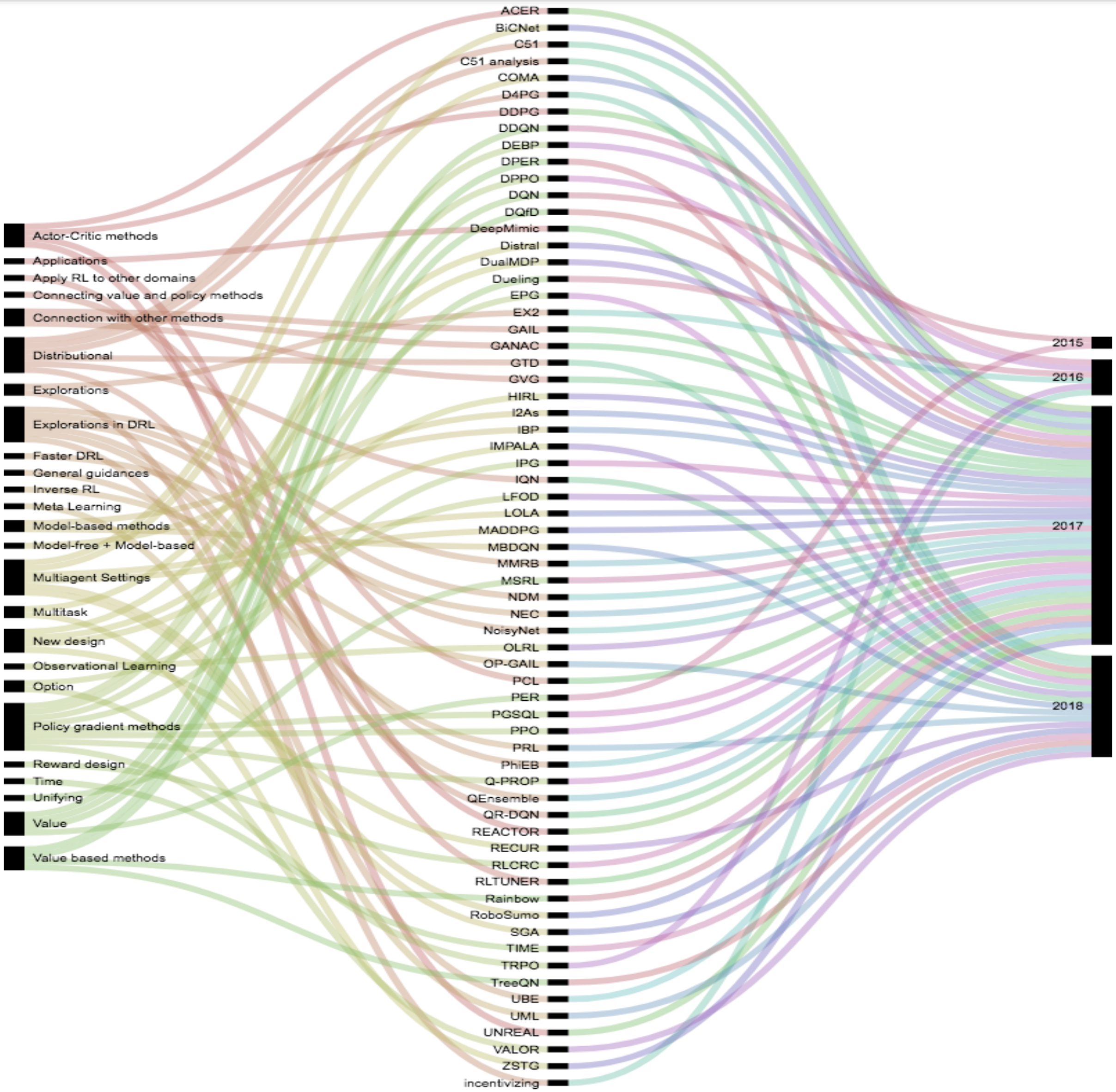
[from David Silver's course]

Categorizing RL Algorithms



[from https://blog.csdn.net/gsww404/article/details/103074046]

Categorizing RL Algorithms



[from https://blog.csdn.net/gsww404/article/details/103074046]

Formulation of an RL Problem

- Markov Process
- Markov Reward Process (MRP)
- Markov Decision Process (MDP)
- Evaluation/Prediction and Improvement/Control in MDP

Recall: Markov Property

- A Markov state contains all useful information from the history, i.e., future is independent of past given present

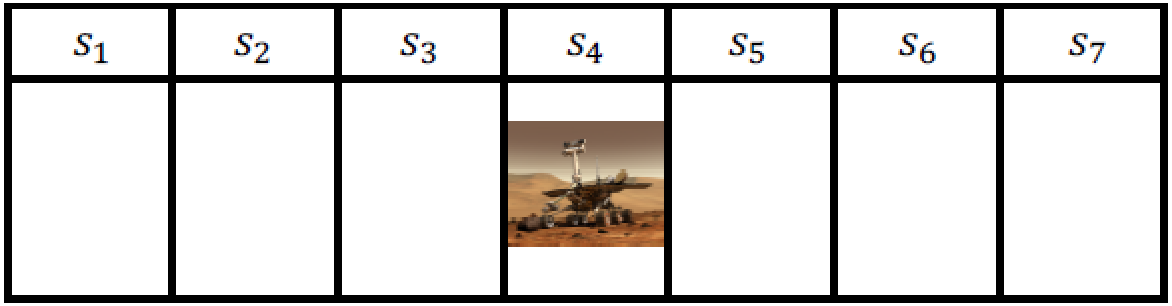
A state S_t is Markov if and only if

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

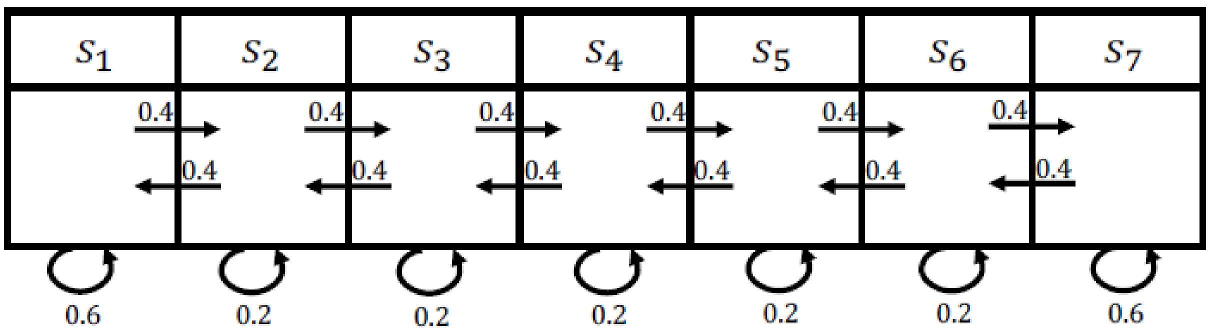
- Markov Process or Markov Chain
  - Sequence of random states with Markov property
  - P is dynamics/transition model that specifies
    $$p(s_{t+1} = s' | s_t = s)$$
  - no rewards, no actions
  - P can be expressed as a matrix

$$P = \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \cdots & P(s_N|s_N) \end{pmatrix}$$

Markov Process



The Mars rover problem



$$P = \begin{pmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

e.g., Sample episodes starting from *S4*

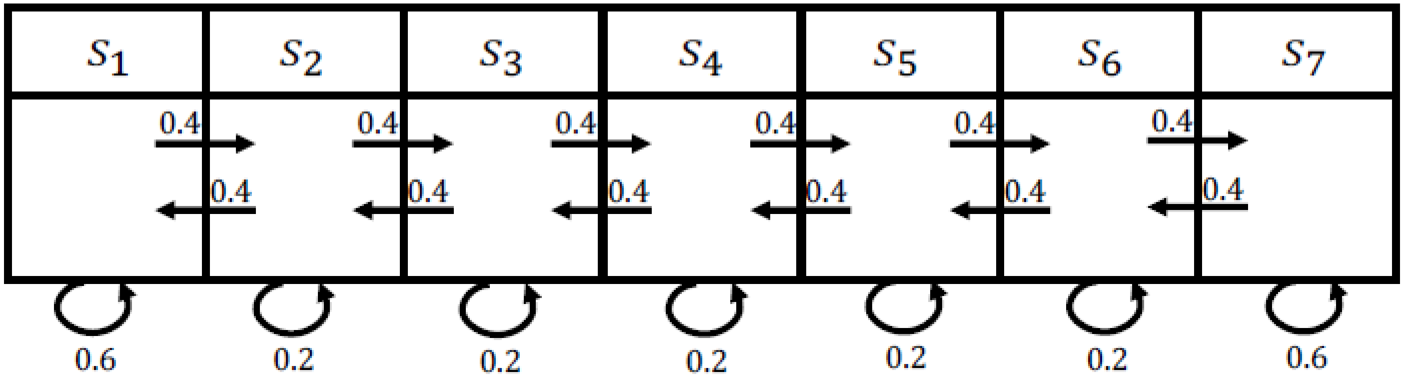
- $s_4, s_5, s_6, s_7, s_7, s_7, \ldots$
- $s_4, s_4, s_5, s_4, s_5, s_6, \ldots$
- $s_4, s_3, s_2, s_1, \ldots$

Markov Reward Process (MRP)

- Markov Reward Process is a Markov Process with rewards
  - P is dynamics/transition model that specifies

$$p(s_{t+1} = s' | s_t = s)$$

  - R is a reward function $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$
  - Discount factor $\gamma \in [0, 1]$
  - No actions



Reward: +1 in s_1, +10 in s_7, 0 in all other states

Return & Value Function

- Definition of Horizon (*H*)
  - Number of time steps in each episode
  - Can be infinite or finite
- Definition of Return
  - Discounted sum of rewards from time step *t* to horizon *H*

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1}$$

- Definition of State Value Function V(s)
  - Expected return from starting in state s

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1} | s_t = s]$$

Discount Factor

- Mathematically convenient
  - avoid infinite returns and values
- Model humans' behaviors
  - $\gamma = 0$: only care about immediate reward
  - $\gamma = 1$: future reward is with the same importance

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1}$$

Computing the Value of MRP

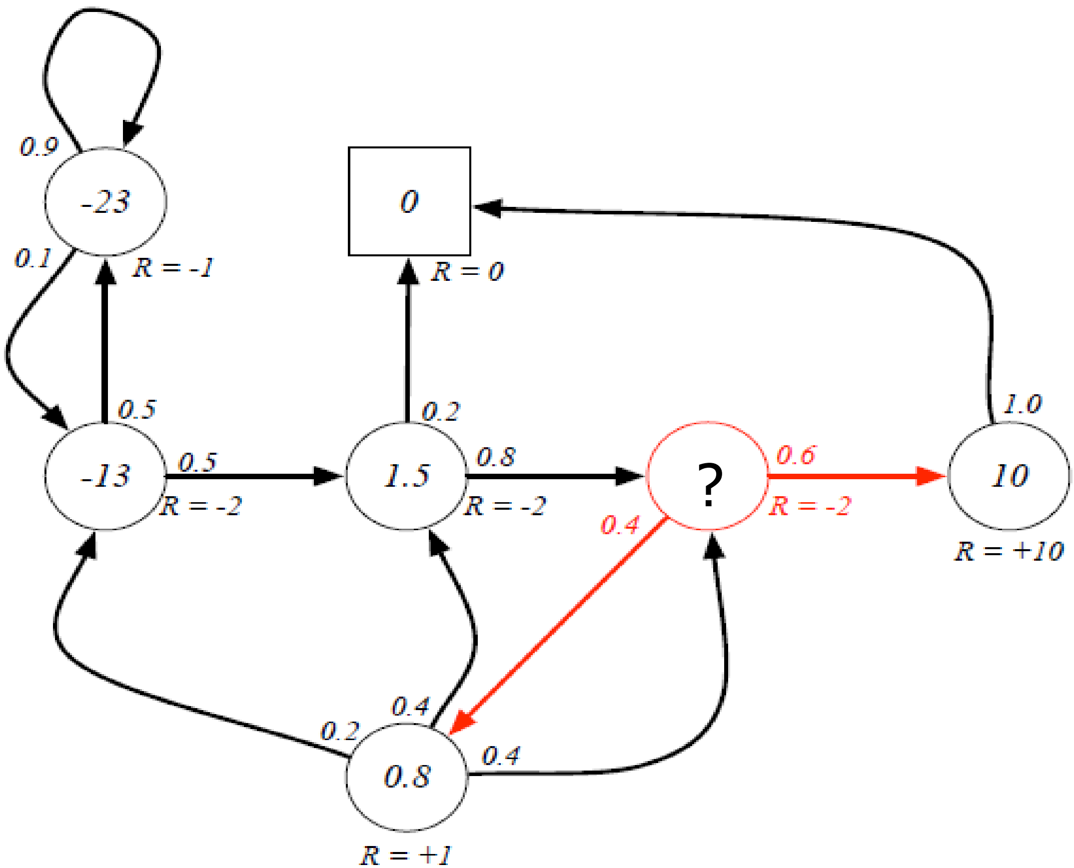
- MRP value function satisfies the Bellman Equation

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1} | s_t = s]$$

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future rewards}}$$

Computing the Value of MRP

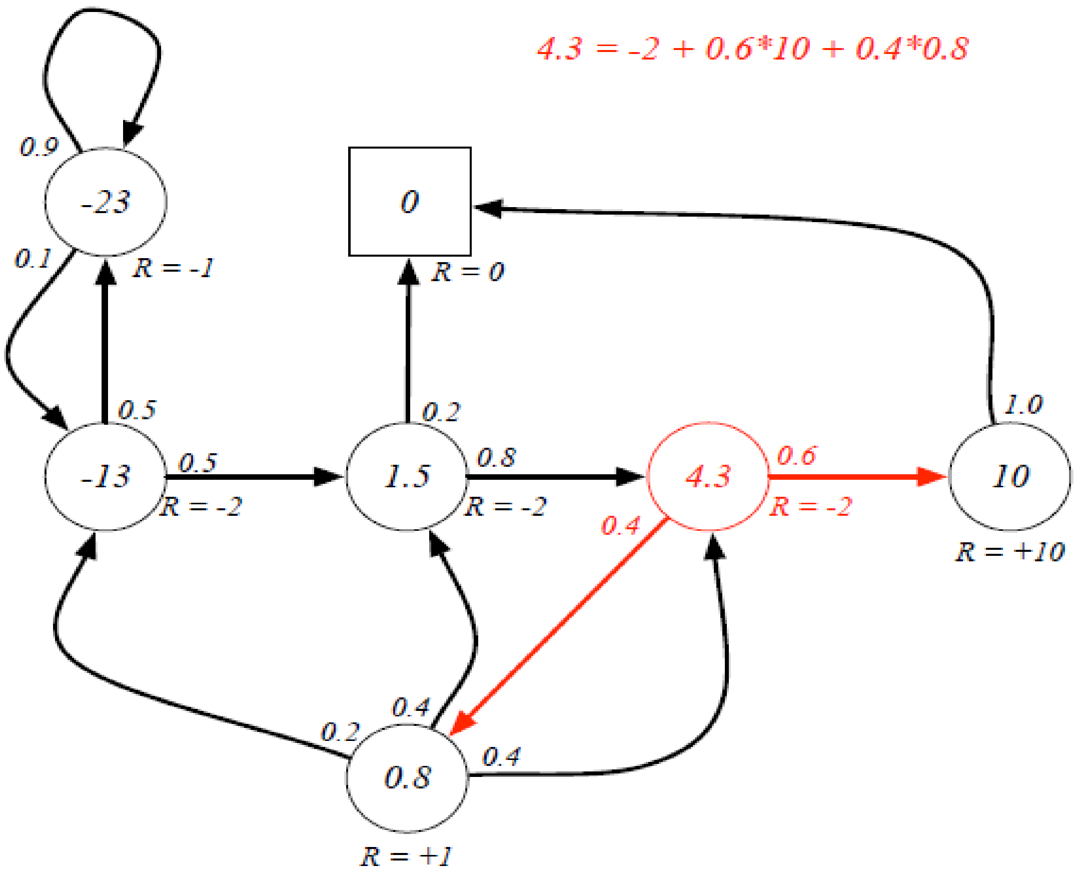
$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted sum of future rewards}}$$



[from David Silver's course]

Computing the Value of MRP

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted sum of future rewards}}$$



[from David Silver's course]

Computing the Value of MRP

- For finite state MRP, we can express *V*(s) in a matrix form

$$\begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ \vdots \\ R(s_N) \end{pmatrix} + \gamma \begin{pmatrix} P(s_1|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \cdots & P(s_N|s_N) \end{pmatrix} \begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix}$$

$$V = R + \gamma PV$$

$$V - \gamma PV = R$$

$$(I - \gamma P)V = R$$

$$V = (I - \gamma P)^{-1} R$$

- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Markov Decision Process (MDP)

- Markov Decision Process is Markov Reward Process with actions
  - P is dynamics/transition model for each action that specifies $P(s_{t+1} = s' | s_t = s, a_t = a)$
  - R is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t = s, a_t = a]$
  - Discount factor $\gamma \in [0, 1]$
  - MDP is a tuple: (S, A, P, R, γ)

$$P(s'|s, a_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad P(s'|s, a_2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The transition matrix for the Mars rover problem (a_1 means moving left, and a_2 means moving right)

MDP Policies

- Policy specifies what action to take in each state
  - Can be deterministic or stochastic
  - Usually is a distribution over actions given states
    $\pi(a|s) = P(a_t = a|s_t = s)$
  - Given an MDP and a policy, then

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

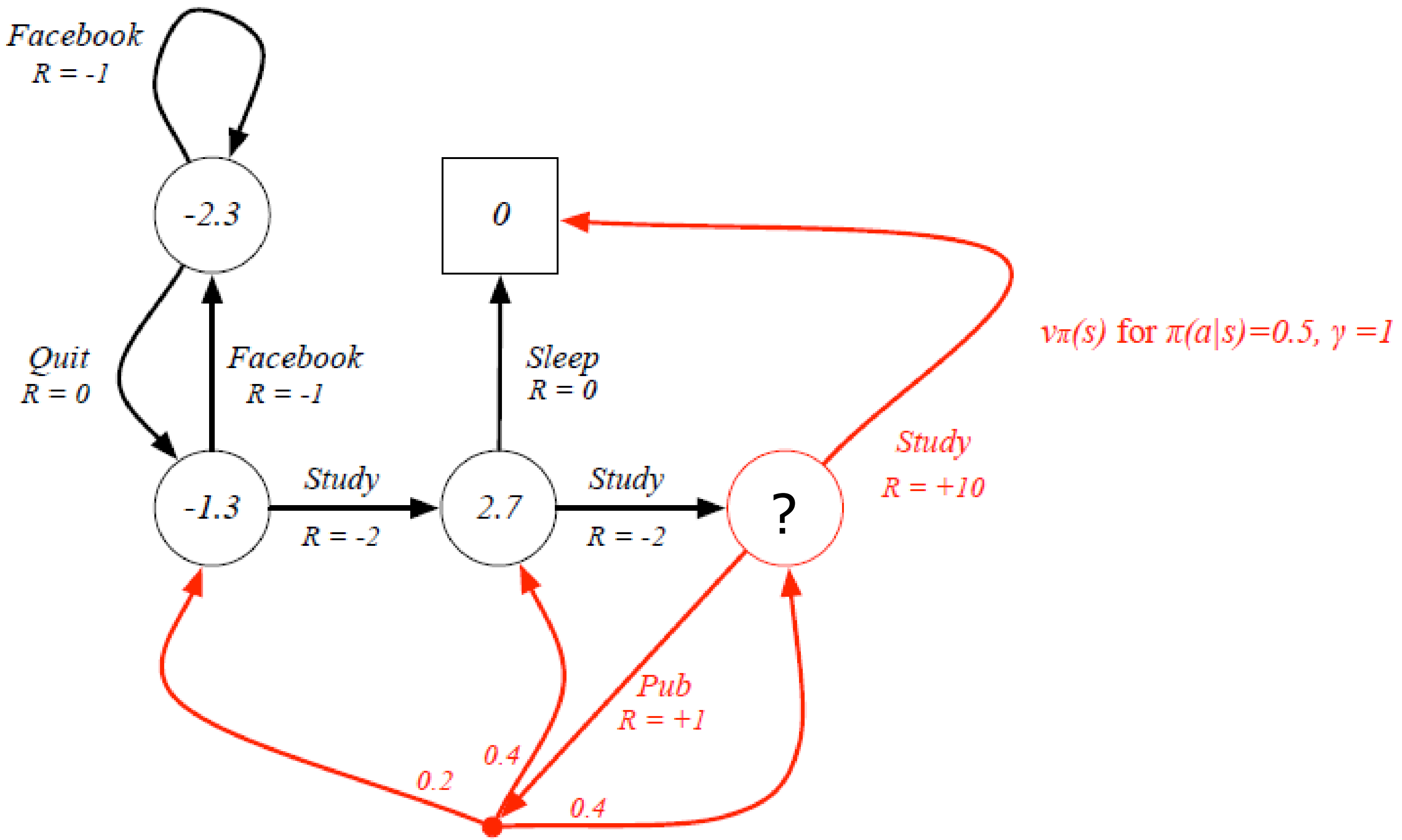
$$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s, a)$$

  - State-Action Value Q for a policy

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s')$$

Computing the Value of MDP

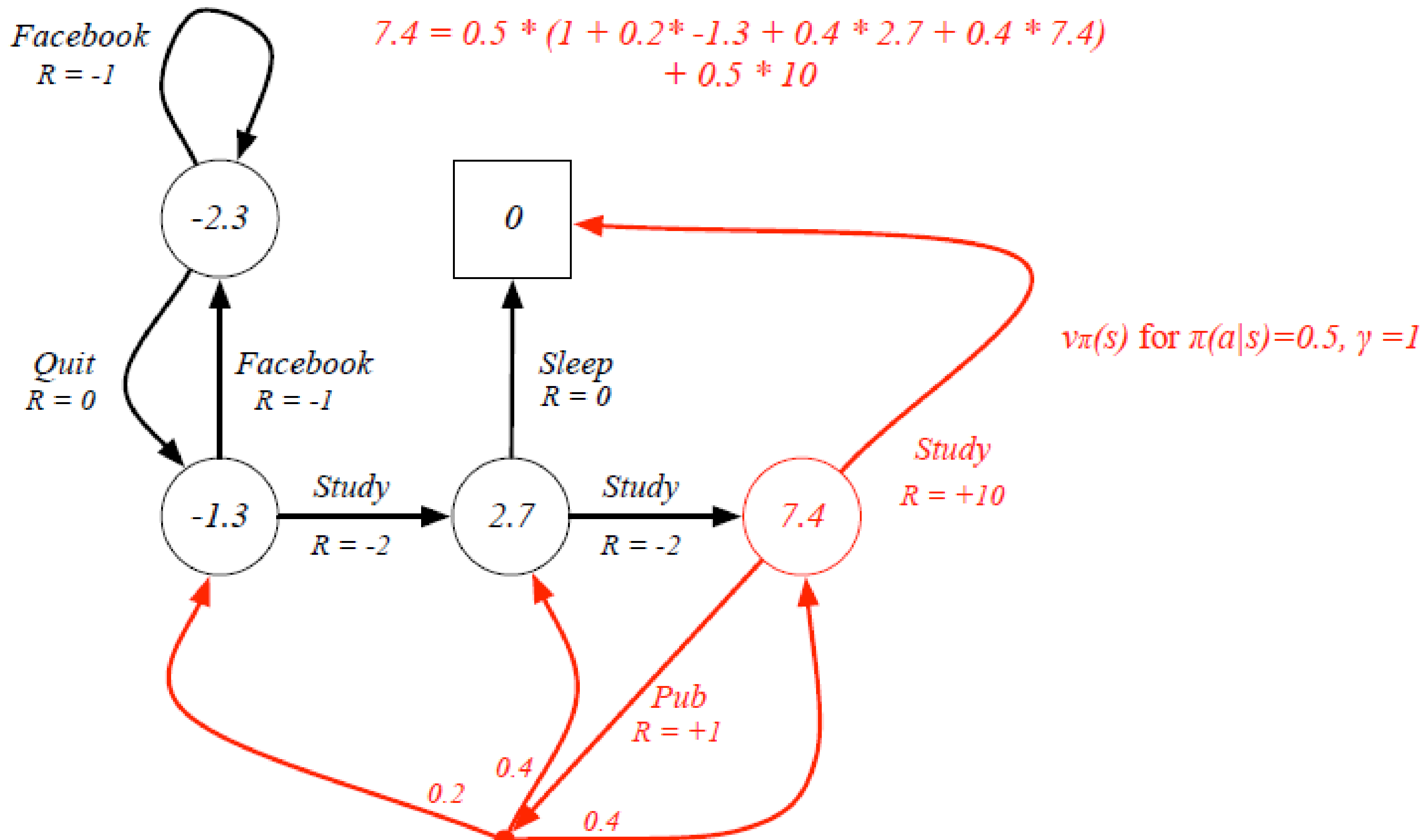
- The value of a state is the value of expected next state plus the reward expected along the way



[from David Silver's course]

Computing the Value of MDP

- The value of a state is the value of expected next state plus the reward expected along the way



[from David Silver's course]

Optimal Policy

- Compute the optimal policy

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

- There exists a unique optimal value function, which specifies the best possible performance in the MDP
- Optimal policy for an MDP in an infinite horizon problem is deterministic, but not necessarily unique
- One option is searching to compute best policy
- Number of deterministic policies is $|A|^{|S|}$
- Policy iteration is generally more efficient than enumeration

What's Dynamic Programming (DP)?

- *Dynamic*: sequential or temporal component to the problem
- *Programming:* optimizing a "program", i.e. a policy
- A method for solving complex problems by
  - breaking them down into subproblems
  - combining solutions of subproblems
- DP is a general solution method for problems with two properties:
  - Optimal solution can be decomposed into subproblems
  - Subproblems recur many times and solutions can be cached and reused
- MDP satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted sum of future rewards}}$$

Solving MDP using DP

- DP assumes full knowledge of the MDP for planning
- DP is an iterative solution method to MDP
  - Policy Iteration (PI)
  - Value Iteration (VI)
- PI iterates between the following processes
  - Policy evaluation (prediction): Estimate/predict the expected rewards from following a given policy
  - Policy improvement (control): find a better policy
- VI iterates between the estimation of value functions and policy optimization, without explicit policy

Value Function for MDP

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

$$v^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

where $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$

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

$$q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, A_t = a]$$

- We have the relation

$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s) q^{\pi}(s, a)$$

Bellman Expectation Equation

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

$$v^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma v^{\pi}(s_{t+1}) | s_t = s]$$

- The action-value function can similarly be decomposed

$$q^{\pi}(s, a) = E_{\pi}[R_{t+1} + \gamma q^{\pi}(s_{t+1}, A_{t+1}) | s_t = s, A_t = a]$$

Bellman Expectation Equation for V and Q

$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s) q^{\pi}(s, a)$$

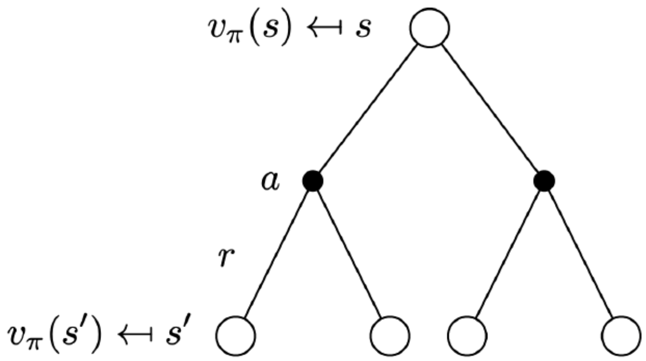
$$q^{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s, a) v^{\pi}(s')$$

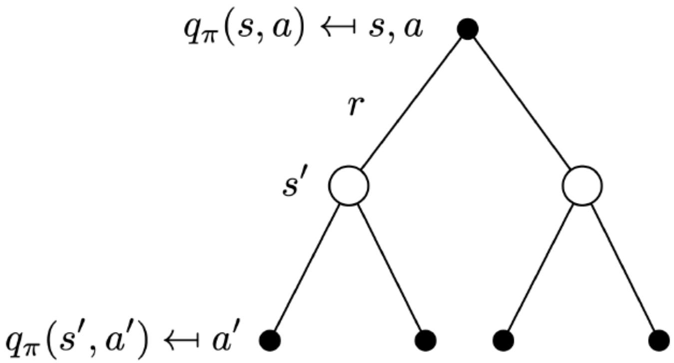
Thus

$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s)(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^{\pi}(s'))$$

$$q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') q^{\pi}(s', a')$$

Backup Diagram for V and Q



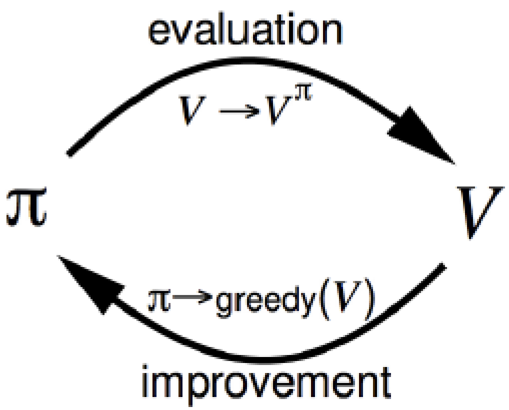


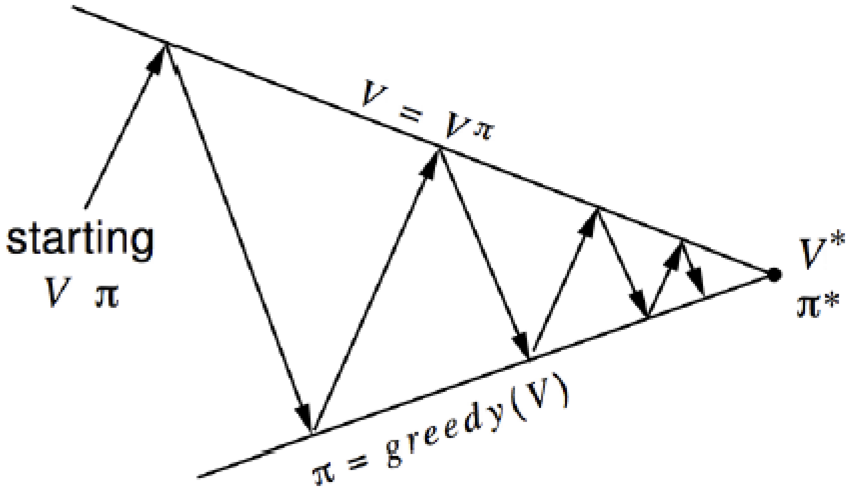
$$v^{\pi}(s) = \sum_{a\in A} \pi(a|s)(R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) v^{\pi}(s'))$$

$$q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \sum_{a'\in A} \pi(a'|s') q^{\pi}(s',a')$$

Policy Iteration (PI)

- Set $i = 0$
- Initialize $\pi_0(s)$ randomly for all states s
- While $i == 0$ or $\|\pi_i - \pi_{i-1}\|_1 > 0$ (L1-norm, measures if the policy changed for any state):
  - $V^{\pi_i} \leftarrow$ MDP V function policy **evaluation** of π_i
  - $\pi_{i+1} \leftarrow$ Policy **improvement**
  - $i = i + 1$





Policy Evaluation

- Objective: evaluate a given policy π for a MDP
- Output: the value function under policy π
- Solution: iteration on Bellman expectation backup
- Algorithm: Synchronous backup

1. At each iteration t+1
   update $v_{t+1}(s)$ from $v_t(s')$ for all states $s \in \mathcal{S}$ where s' is a successor state of s

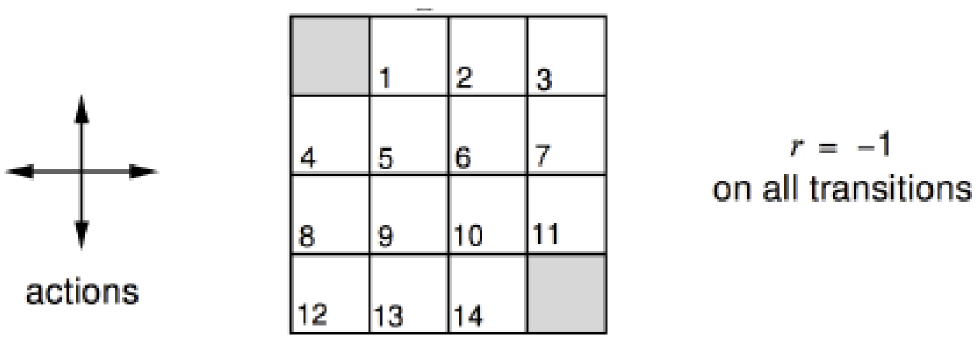
$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_t(s'))$$

- Convergence: $v_1 \to v_2 \to \ldots \to v^\pi$

Policy Evaluation

☐ Example 4.1 in the Sutton RL textbook.

- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy



$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_t(s'))$$

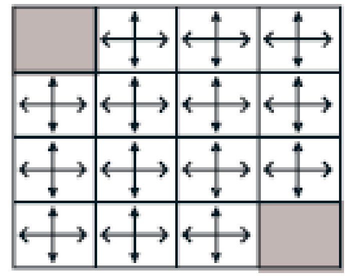
Policy Evaluation

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

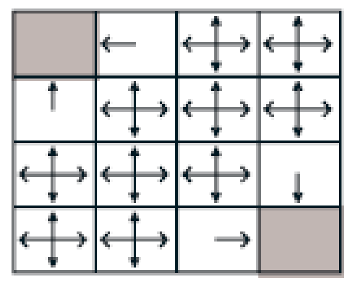
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

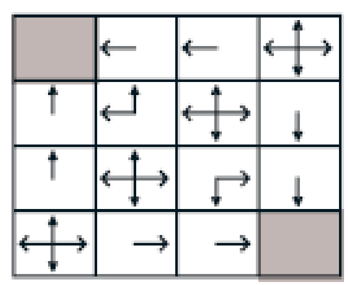
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Policy Evaluation

$k = 3$

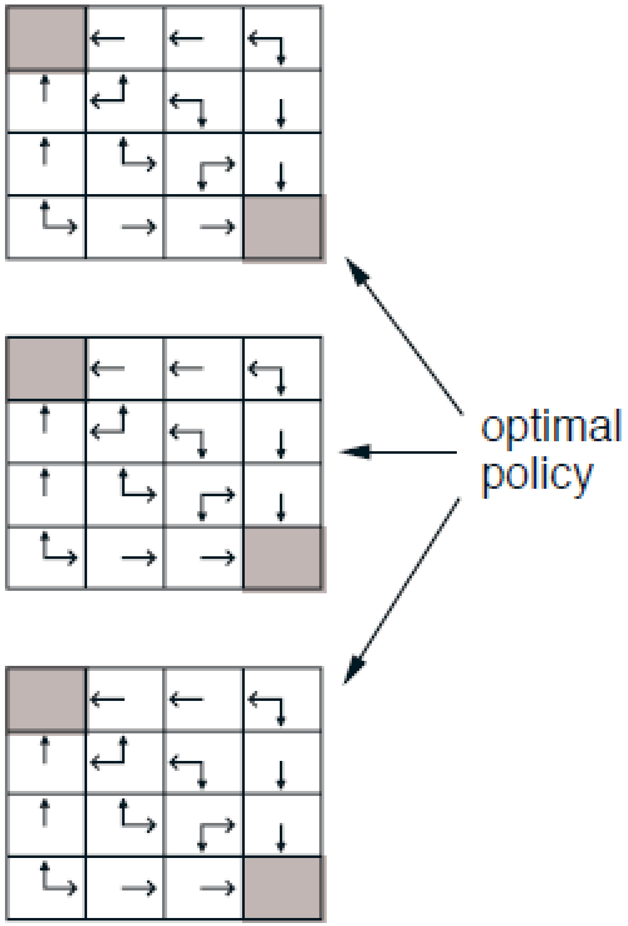
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Improvement

- Consider a determinisitc policy $a = \pi(s)$
- We improve the policy through

$$\pi'(s) = \arg\max_a q^\pi(s, a)$$

- This improves the value from any state *s* over one step

$$q^\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^\pi(s, a) \geq q^\pi(s, \pi(s)) = v^\pi(s)$$

- It therefore improves the value function $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned}
v_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)
\end{aligned}$$

Bellman Optimality Equation

- The optimal value functic B^{π} are reached by the Bellman optimality equations

$$v^*(s) = \max_a q^*(s,a)$$

$$q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^*(s')$$

thus

$$v^*(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^*(s')$$

$$q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} q^*(s',a')$$

Value Iteration (VI)

- Policy iteration computes optimal value and policy
- Value iteration is another technique
  - Maintain optimal value of starting in a state *s* if having a finite number of steps *k* left in the episode
  - Iterate to consider longer and longer episodes
- In other word, we assume we know the solution to subproblems and then find the optimal solution by one-step lookahead

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \right)$$

Value Iteration (VI)

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration (VI)

1. Objective: find the optimal policy π
2. Solution: iteration on the Bellman optimality backup
3. Value Iteration algorithm:
    1. initialize $k = 1$ and $v_0(s) = 0$ for all states s
    2. For $k = 1 : H$
        1. for each state s

$$q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_k(s')$$

$$v_{k+1}(s) = \max_a q_{k+1}(s,a)$$

        2. $k \leftarrow k + 1$
    3. To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{k+1}(s')$$

Value vs Policy Iteration

- Policy iteration includes: policy evaluation + policy improvement, and the two are repeated iteratively until policy converges
- Value iteration includes: finding optimal value function + one policy extraction. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
- Finding optimal value function can also be seen as a combination of policy improvement (due to max) and truncated policy valuation (the reassignment of v(s) after just one sweep of all states regardless of convergence).

Summary of DP methods

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2n^2)$ per iteration

[from David Silver's course]

Conclusion

- Define MP, MRP, MDP, Bellman operator, contraction, model
- Know how to compute value function, optimal policy
- Be able to implement Value Iteration and Policy Iteration
- Give pros and cons of different policy evaluation approaches