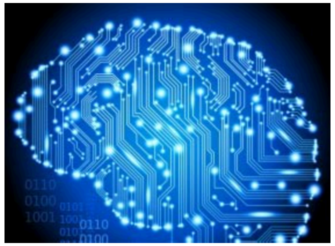
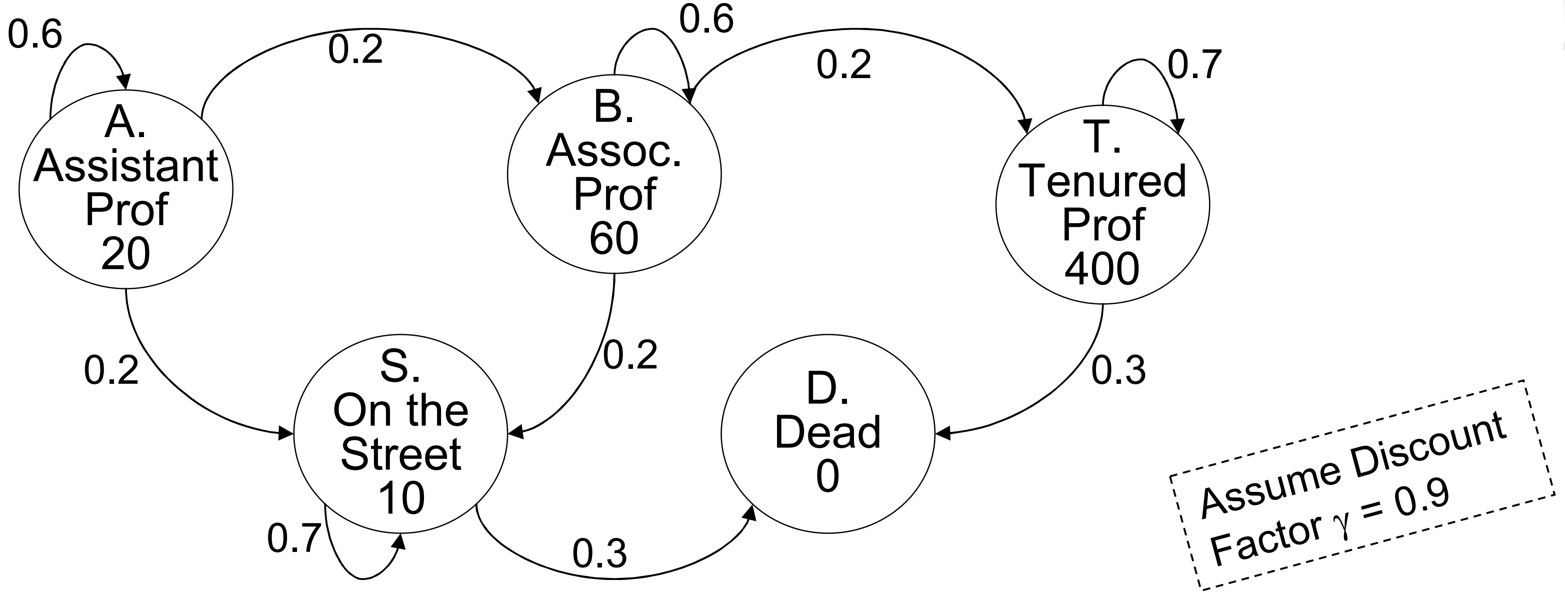
Lecture 18: About Time-MDP

http://cs.nju.edu.cn/yuy/course_ai16.ashx



Markov process

The Academic Life



Define:

J_A = Expected discounted future rewards starting in state A

J_B = Expected discounted future rewards starting in state B

J_T = " " " " " " " T

J_S = " " " " " " " S

J_D = " " " " " " " D

How do we compute J_A, J_B, J_T, J_S, J_D ?

Rewards

"A reward (payment) in the future is not worth quite as much as a reward now."

- Because of chance of obliteration
- Because of inflation

Example:

Being promised \$10,000 next year is worth only 90% as much as receiving \$10,000 right now.

Assuming payment *n* years in future is worth only $(0.9)^n$ of payment now, what is the AP's Future Discounted Sum of Rewards ?

Discount factor

People in economics and probabilistic decision-making do this all the time.

The "Discounted sum of future rewards" using discount factor γ" is

$$
\begin{aligned}
&\text{(reward now)} + \\
&\gamma\ \text{(reward in 1 time step)} + \\
&\gamma^2\ \text{(reward in 2 time steps)} + \\
&\gamma^3\ \text{(reward in 3 time steps)} + \\
&\quad\vdots \\
&\quad\vdots \qquad \text{(infinite sum)}
\end{aligned}
$$

A Markov process with rewards

- Has a set of states $\{S_1\ S_2 \cdots S_N\}$
- Has a transition probability matrix

$$P = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & & & \\ \vdots & & & \\ P_{N1} & & \cdots & P_{NN} \end{pmatrix} \qquad P_{ij} = \text{Prob}(\text{Next} = S_j \mid \text{This} = S_i)$$

- Each state has a reward. $\{r_1\ r_2 \cdots r_N\}$
- There's a discount factor γ. $0 < \gamma < 1$

On Each Time Step …

0. Assume your state is S_i
1. You get given reward r_i
2. You randomly move to another state
   $P(\text{NextState} = S_j \mid \text{This} = S_i) = P_{ij}$
3. All future rewards are discounted by γ

Value iteration: solve expected reward

Define

$J^1(S_i)$ = Expected discounted sum of rewards over the next 1 time step.

$J^2(S_i)$ = Expected discounted sum rewards during next 2 steps

$J^3(S_i)$ = Expected discounted sum rewards during next 3 steps

:

$J^k(S_i)$ = Expected discounted sum rewards during next *k* steps

$J^1(S_i)$ = (what?)

$J^2(S_i)$ = (what?)

:

$J^{k+1}(S_i)$ = (what?)

Value iteration: solve expected reward

Define

$J^1(S_i)$ = Expected discounted sum of rewards over the next 1 time step.

$J^2(S_i)$ = Expected discounted sum rewards during next 2 steps

$J^3(S_i)$ = Expected discounted sum rewards during next 3 steps

:

$J^k(S_i)$ = Expected discounted sum rewards during next *k* steps

N = Number of states

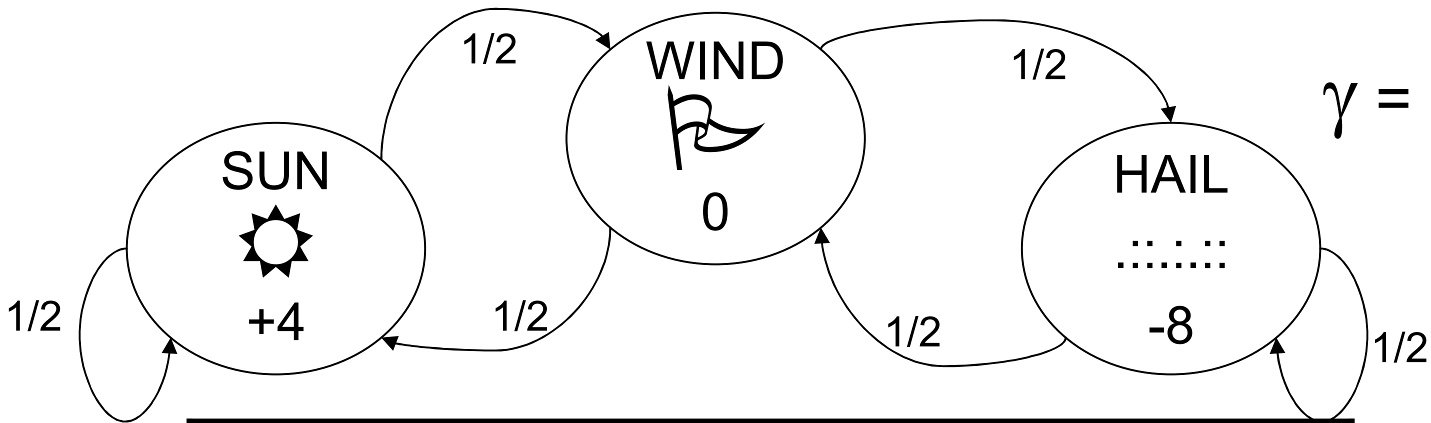
$J^1(S_i) = r_i$ (what?)

$$J^2(S_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^1(s_j)$$ (what?)

:

$$J^{k+1}(S_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^k(s_j)$$ (what?)

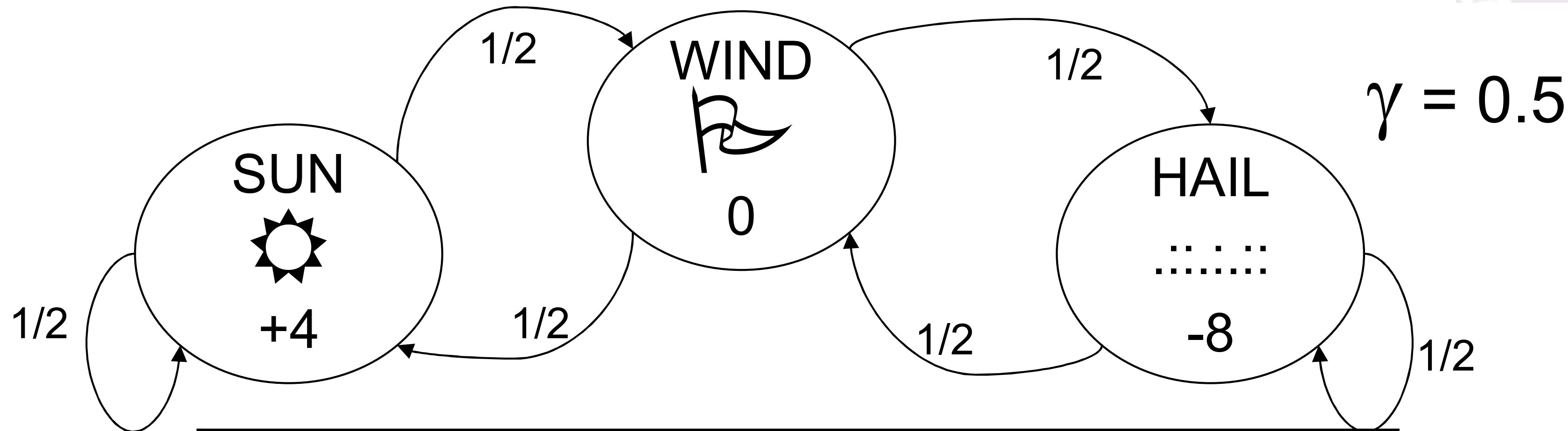
Example



$\gamma = 0.5$

k	J^k(SUN)	J^k(WIND)	J^k(HAIL)
1			
2			
3			
4			
5			

Example



k	J^k(SUN)	J^k(WIND)	J^k(HAIL)
1	4	0	-8
2	5	-1	-10
3	5	-1.25	-10.75
4	4.94	-1.44	-11
5	4.88	-1.52	-11.11

Value iteration: solve expected reward

- Compute $J^1(S_i)$ for each j
- Compute $J^2(S_i)$ for each j

  :
- Compute $J^k(S_i)$ for each j

As $k \to \infty$ $J^k(S_i) \to J^*(S_i)$. Why?

When to stop? When

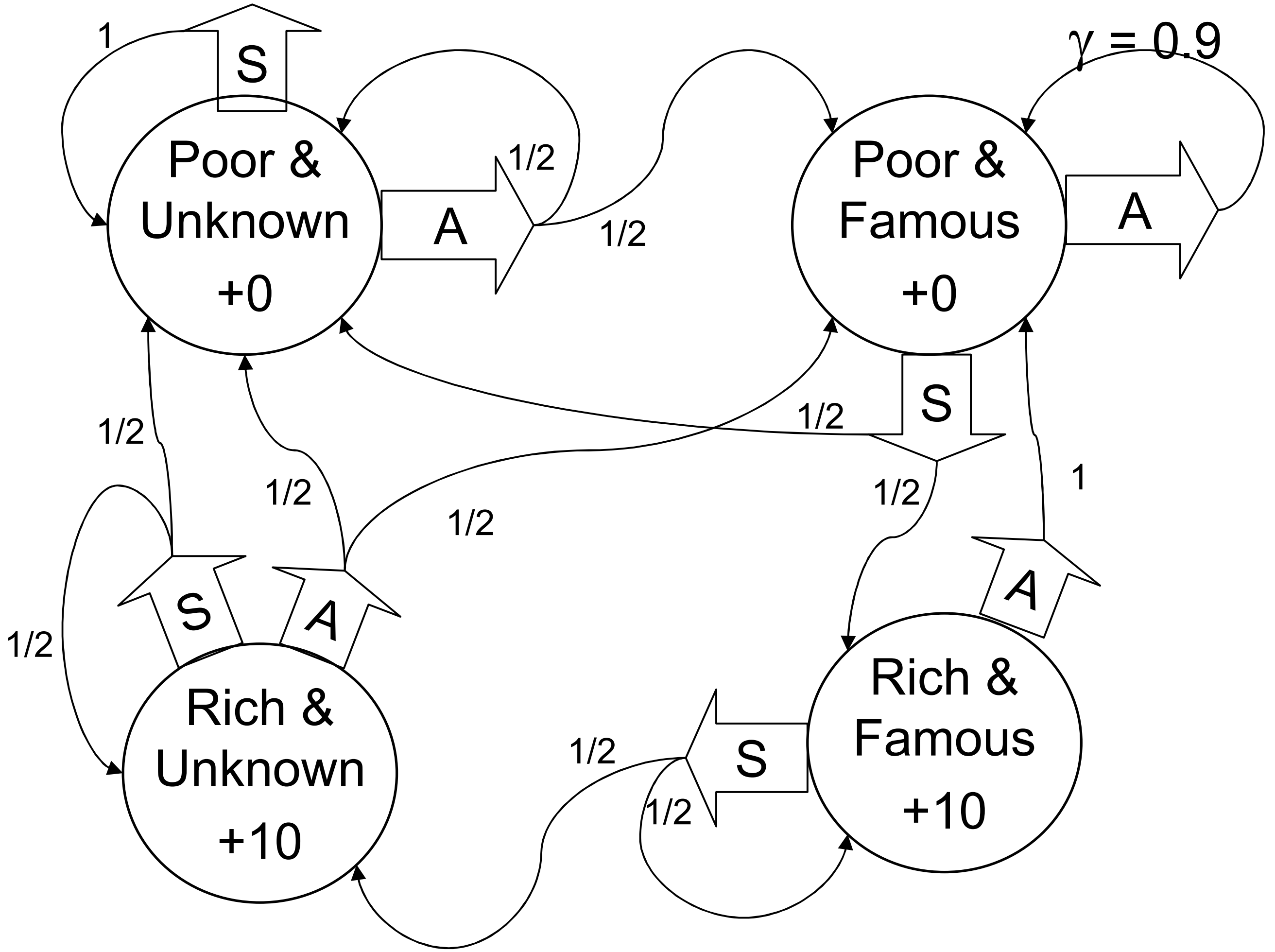
$$\max_i \left| J^{k+1}(S_i) - J^k(S_i) \right| < \xi$$

Markov process -> Markov **decision** process

A Markov decision process (MDP)

You run a startup company.

In every state you must choose between Saving money or Advertising.



Markov decision process

An MDP has…

- A set of states $\{s_1 \cdots S_N\}$
- A set of actions $\{a_1 \cdots a_M\}$
- A set of rewards $\{r_1 \cdots r_N\}$ (one for each state)
- A transition probability function

$$P_{ij}^{k} = \text{Prob}\left(\text{Next} = j \middle| \text{This} = i \text{ and I use action } k\right)$$

On each step:

0. Call current state S_i
1. Receive reward r_i
2. Choose action $\in \{a_1 \cdots a_M\}$
3. If you choose action a_k you'll move to state S_j with probability P_{ij}^{k}
4. All future rewards are discounted by γ

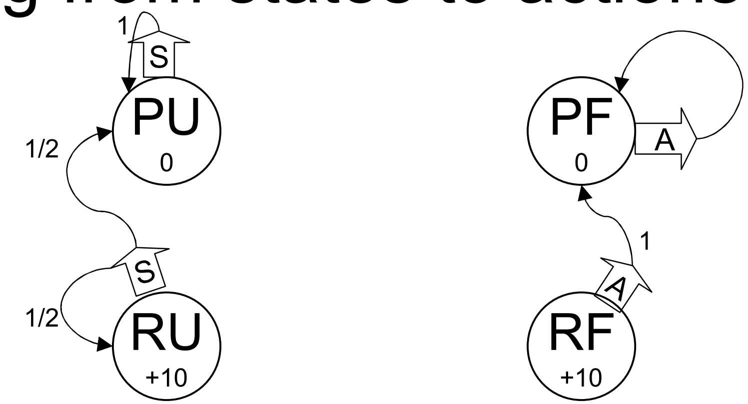
Policy

A policy is a mapping from states to actions.

Examples

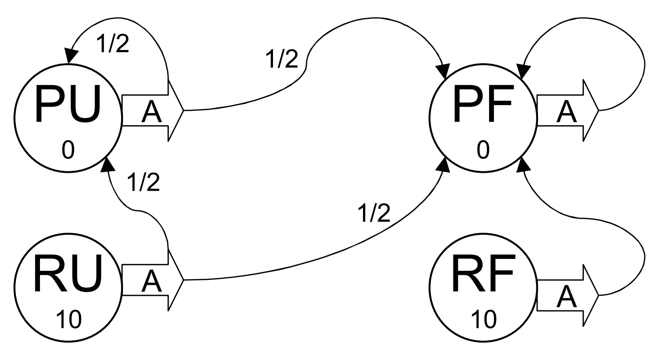
Policy Number 1:

STATE →	ACTION
PU	S
PF	A
RU	S
RF	A



Policy Number 2:

STATE →	ACTION
PU	A
PF	A
RU	A
RF	A



- How many possible policies in our example?
- Which of the above two policies is best?
- How do you compute the optimal policy?

Facts

For every M.D.P. there exists an optimal policy.

It's a policy such that for every possible start state there is no better option than to follow the policy.

(Not proved in this lecture)

Computing the best policy

Idea One:

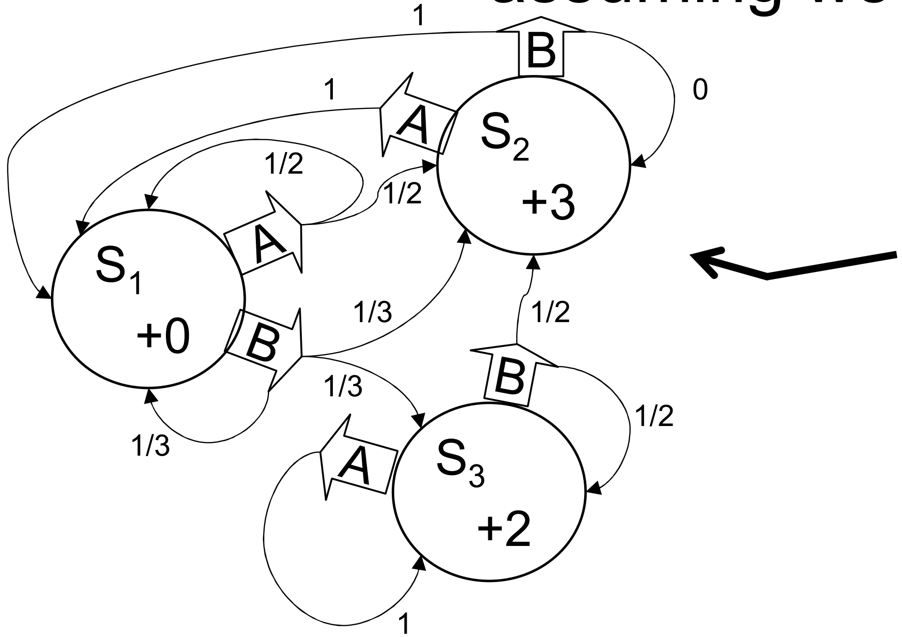
Run through all possible policies.

Select the best.

What's the problem ??

Optimal value function

Define $J^*(S_i)$ = Expected Discounted Future Rewards, starting from state S_i, assuming we use the optimal policy



Question

What (by inspection) is an optimal policy for that MDP?

(assume $\gamma = 0.9$)

What is $J^*(S_1)$?
What is $J^*(S_2)$?
What is $J^*(S_3)$?

Value iteration

Define

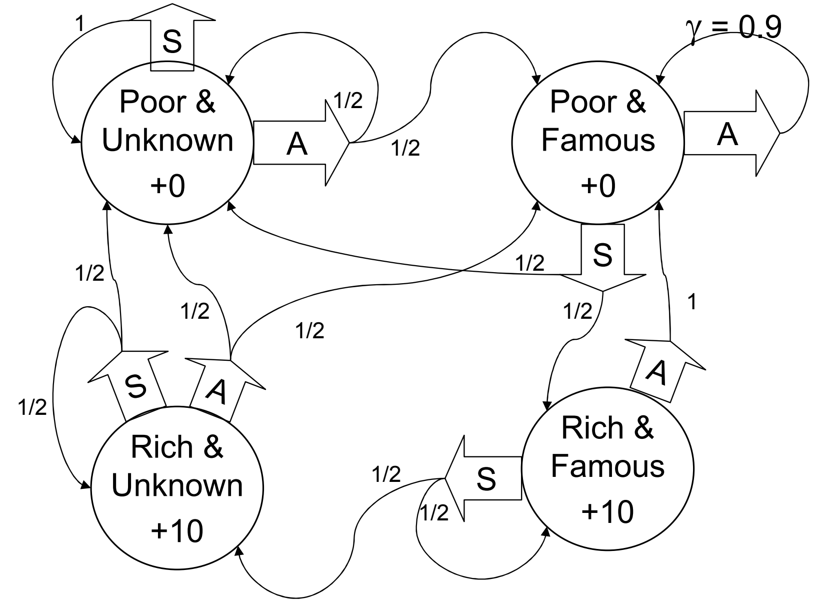
$J^k(S_i)$ = Maximum possible expected sum of discounted rewards I can get if I start at state S_i and I live for *k* time steps.

Note that $J^1(S_i) = r_i$

Example

You run a startup company.

In every state you must choose between Saving money or Advertising.



Let's compute $J^k(S_i)$ for the startup example

k	$J^k(PU)$	$J^k(PF)$	$J^k(RU)$	$J^k(RF)$
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	8.55	16.53	25.08
4	4.76	12.20	18.35	28.72

Bellman's equation

$$J^{n+1}(S_i) = \max_k \left[r_i + \gamma \sum_{j=1}^{N} P_{ij}^k J^n(S_j) \right]$$

Value Iteration for solving MDPs

- Compute $J^1(S_i)$ for all *i*
- Compute $J^2(S_i)$ for all *i*
- :
- Compute $J^n(S_i)$ for all *i*

…..until converged

$$\left[\text{converged when } \max_i \left| J^{n+1}(S_i) - J^n(S_i) \right| \langle \xi \right]$$

…Also known as

Dynamic Programming

Find the optimal policy

1. Compute $J^*(S_i)$ for all i using Value Iteration (a.k.a. Dynamic Programming)

2. Define the best action in state S_i as

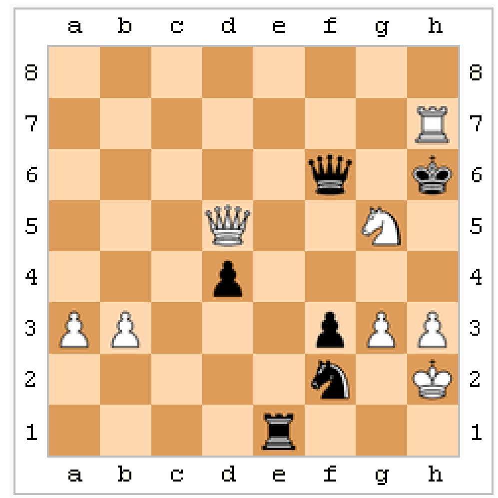
$$\arg\max_k \left[r_i + \gamma \sum_j P_{ij}^k J^*(S_j) \right]$$

(Why?)

Reinforcement learning

Game playing

- What are you doing when you're learning a game?
- Supervised learning:
  - teacher (ie desired output) giving detailed feedback of the best move in each situation
  - infeasible for complex games, e.g. which chess move is best from a given board position?
- Reinforcement learning:
  - no teacher; only feedback is whether you won or lost
  - computer could simulate many games to explore space

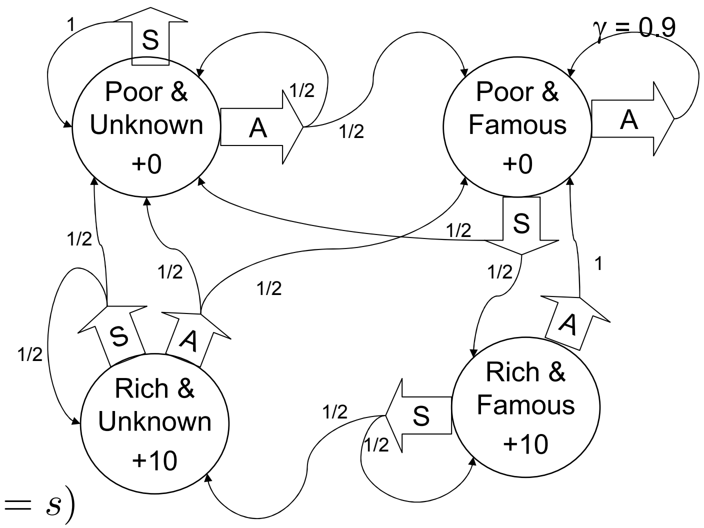


MDP

Recall Markov Decision Processes

- An MDP is defined by:

states:	$\{s_1, \ldots, s_N\}$
initial state:	S_0
actions:	$\{a_1, \ldots, a_M\}$
rewards:	$R(s) = \{r_1, \ldots, r_N\}$
transitions:	$T(s, a, s') = P(s_{t+1} = s' \mid a_t = a, s_t = s)$
discount:	γ



- Value iteration:

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V_i(s')$$

$$V^*(s) = \lim_{i \to \infty} V_i(s)$$

i	V_i(PU)	V_i(PF)	V_i(RU)	V_i(RF)
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	6.53	25.08	18.55
4	3.852	12.20	29.63	19.26
5	7.22	15.07	32.00	20.40
6	10.03	17.65	33.58	22.43

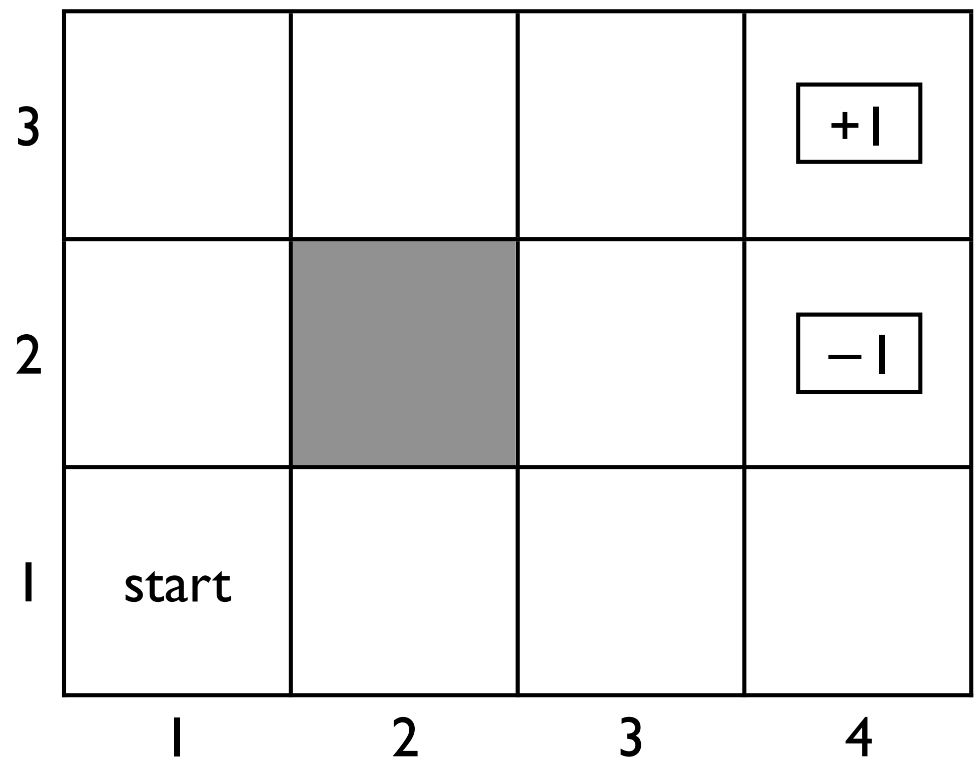
- Optimal policy:

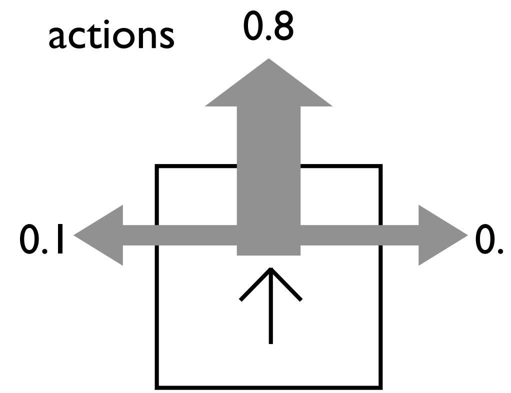
$$\pi^*(s) = \arg\max_a \left[R(s) + \gamma \sum_{s'} T(s, a, s') V^*(s') \right]$$

Example

The 4x3 grid world from R&N Chap. 17

R(s) = -0.04



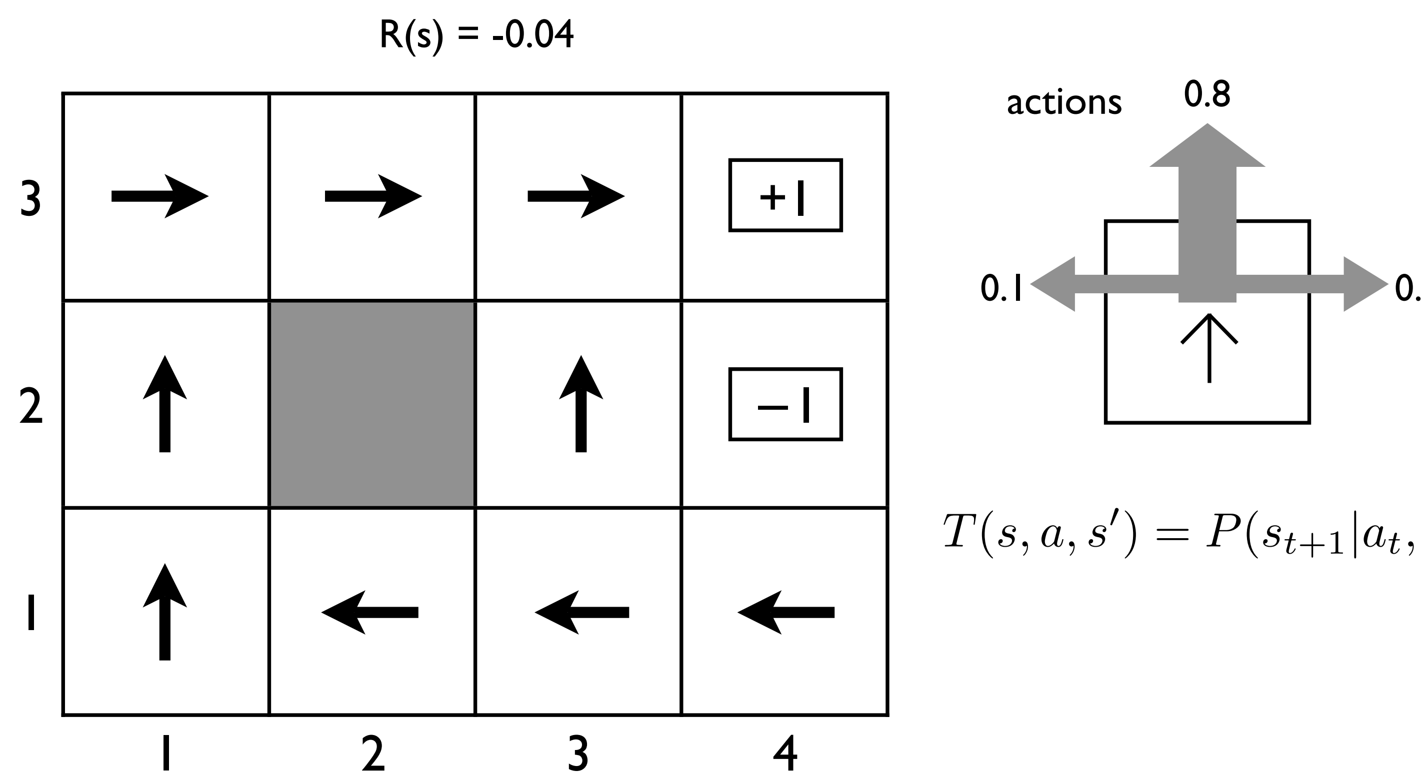


$$T(s, a, s') = P(s_{t+1}|a_t, s_t)$$

- Here, the agent has a complete model of the fully-observable world
- The reward function R(s) and the transition model T(s,a,s') are also known.
- In this example, there is no discounting
- How does the agent maximize reward? Value (or policy) iteration.

Example

The optimal policy



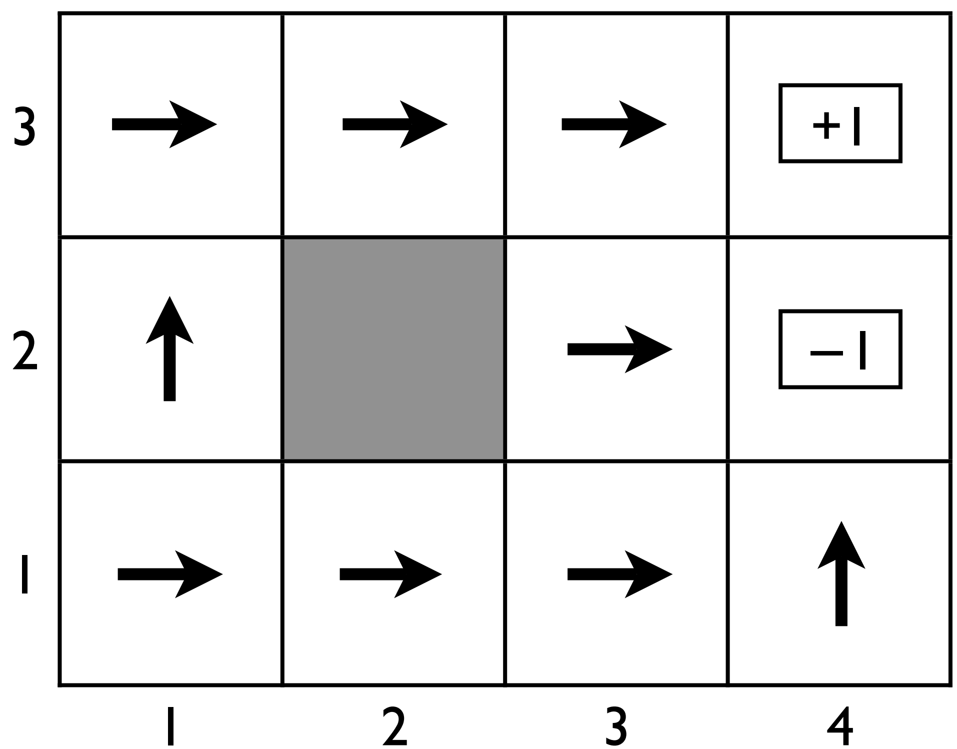
$$T(s, a, s') = P(s_{t+1} | a_t, s_t)$$

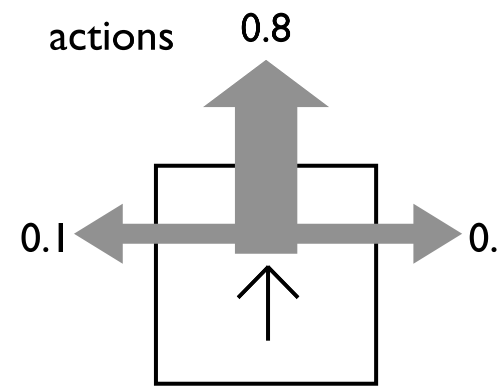
- With weak negative reward, best policy is to avoid -1 when possible and seek +1

Example

Another optimal policy

R(s) = < 1.6285





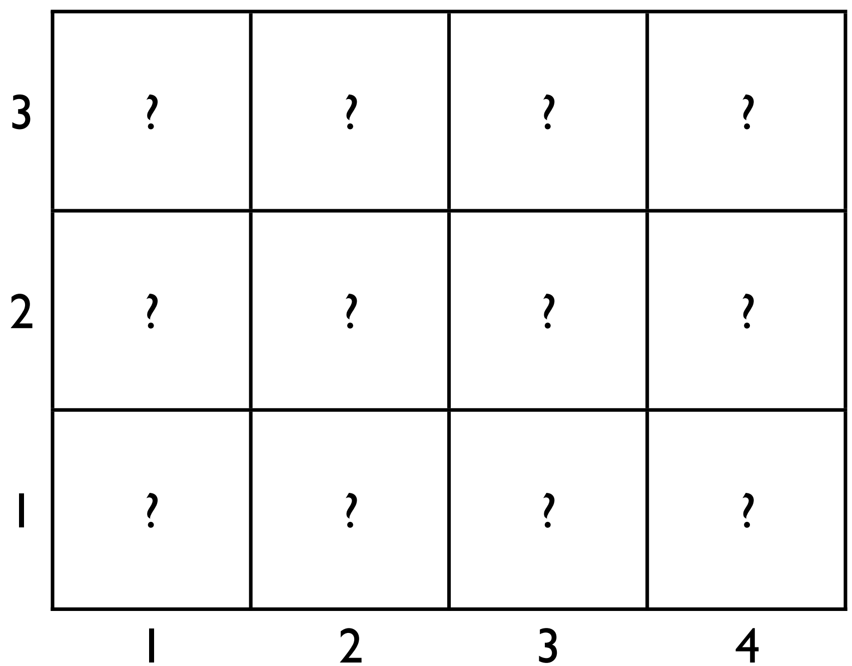
$$T(s, a, s') = P(s_{t+1}|a_t, s_t)$$

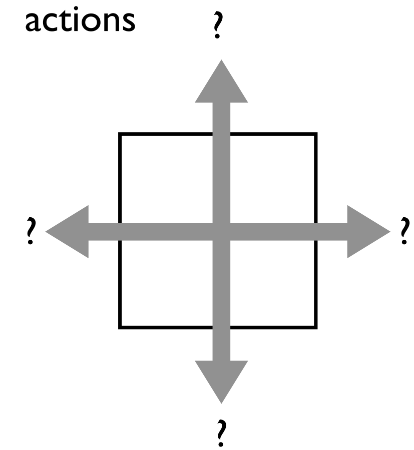
- With strong negative reward, best policy is to always seek end states

Example

Reinforcement learning: *learning from experience*

R(s) = ?





$T(s, a, s') = ?$

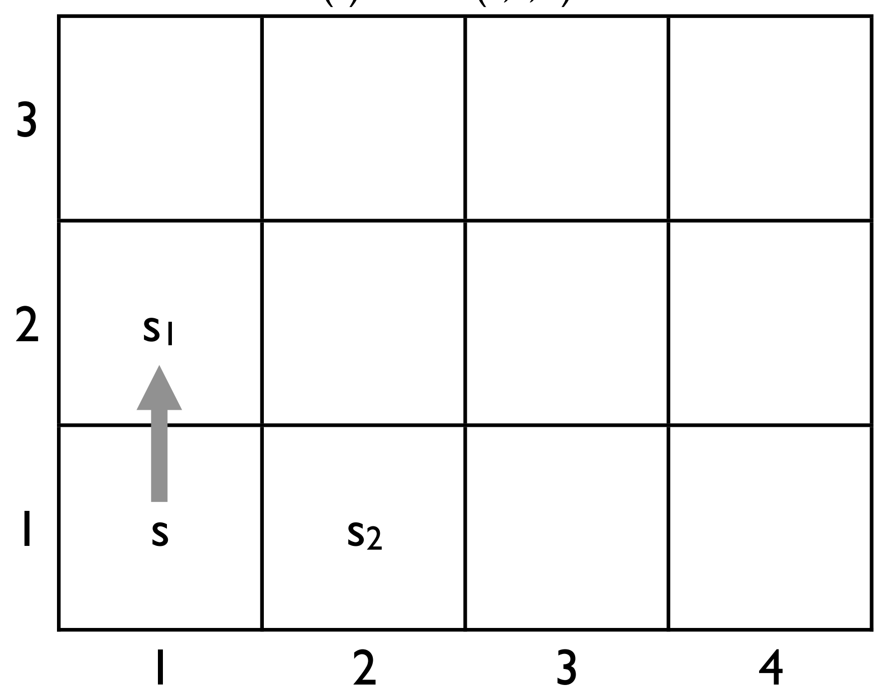
- Now: don't know environment, transition model, or reward function
- Need to explore the world. At each state the agent:
    1. selects one of the available actions
    2. receives reward
    3. observes resulting state
    4. repeat until a terminal state

In *reinforcement learning*, the objective is to find an optimal policy from these observations.

Example

Model estimation

R(s) = ? T(s,a,s') = ?



$$R(s) = \text{reward for state s}$$

$$\hat{T}(s, a, s') \approx \frac{\#\text{ transitions } s \rightarrow s' \text{ for action } a}{\#\text{ times } a \text{ selected at state } s}$$

- Observe statistics for actions from each state.
- Eg, for 10 "up" actions from state (1,1) we observe next state is
  - s_1 = (1,2) 8 times & R= -0.04
  - s_2 = (2,1) 2 times & R= -0.04
- ⇒T(s, Up, s_1) = 8/10 = 0.8

  T(s, Up, s_2) = 2/10 = 0.2

  R(s_1) = R(s_2) = 0.04
- Could use Bayesian estimates here
- Continued exploration of the grid world will give increasingly accurate estimates of T(s,a,s') and R(s).

Model-based reinforcement learning

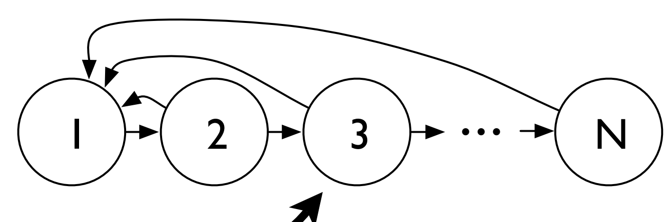
- With estimates of T(s,a,s') and R(s), we can just treat it like an MDP.
- Compute the expected value of each state as:

$$V_{i+1}(s) = \hat{R}(s) + \gamma \max_a \sum_{s'} \hat{T}(s,a,s') V_i(s')$$
$$V^*(s) = \lim_{i \to \infty} V_i(s)$$

- And the optimal policy (with value iteration) is:

$$\pi^*(s) = \arg\max_a \left[\hat{R}(s) + \gamma \sum_{s'} \hat{T}(s,a,s') V^*(s') \right]$$

- This is called *Certainty Equivalent* learning.
- Issues and caveats:
  - Could be a very inefficient way to explore the world.
  - When do we stop and estimate the policy?
  - Solving for V*(s) every time could be expensive.



Model-free reinforcement learning

And there are more methods that do not estimate the MDP model.

R. Sutton. Introduction to Reinforcement Learning.