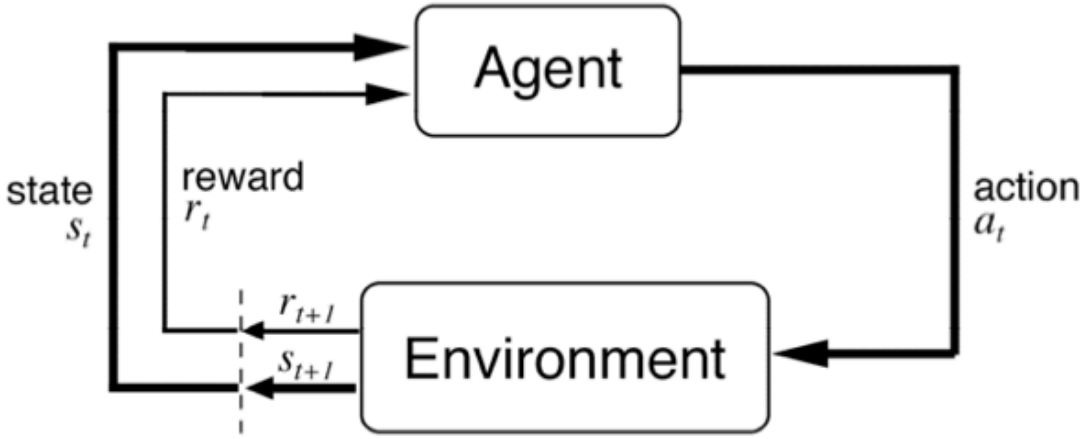
Reinforcement Learning

91.450 Robotics I
Spring 2014
Prof. Yanco

Many of the slides in this presentation are from
R. Sutton and A. Barto, as noted at the bottom of the slides

The Agent-Environment Interface



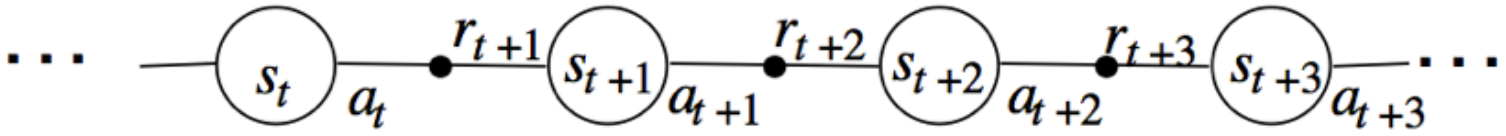
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



The Agent Learns a Policy

Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s,a) =$ probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

Getting the Degree of Abstraction Right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), "mental" (e.g., shift in focus of attention), etc.
- States can low-level "sensations", or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being "surprised" or "lost").
- An RL agent is not like a whole animal or robot.
- Reward computation is in the agent's environment because the agent cannot change it arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.

Goals and Rewards

- Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.
- A goal should specify what we want to achieve, not how we want to achieve it.
- A goal must be outside the agent's direct control—thus outside the agent.
- The agent must be able to measure success:
  - explicitly;
  - frequently during its lifespan.

The reward hypothesis

- That all of what we mean by goals and purposes can be well thought of as maximizing a received scalar signal

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \cdots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T ,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

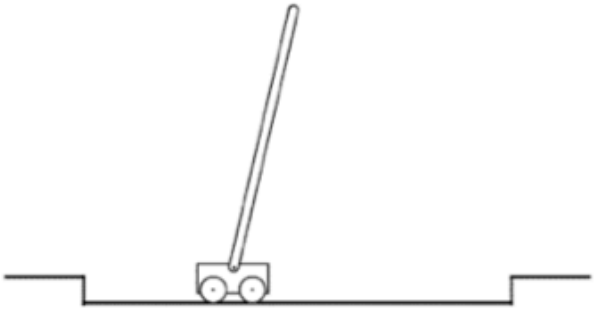
Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ, $0 \leq \gamma \leq 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

An Example



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

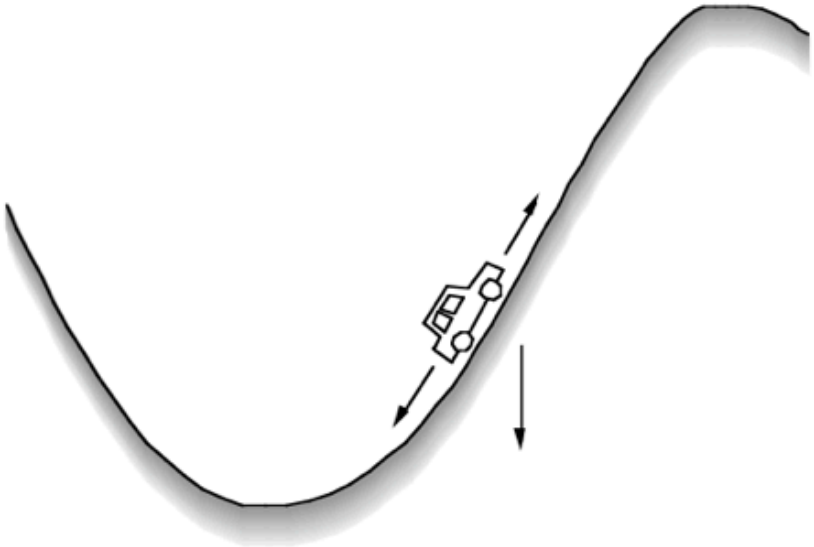
As a **continuing task** with discounted return:

reward = –1 upon failure; 0 otherwise

$\Rightarrow$ return = $-\gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill
as quickly as possible.

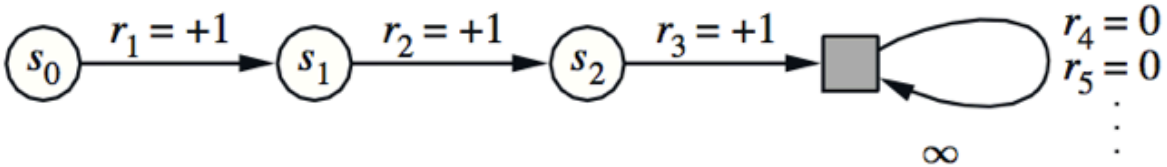
reward = –1 for each step where **not** at top of hill

$\Rightarrow$ return = – number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

A Unified Notation

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have to distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step t of episode j.
- Think of each episode as ending in an **absorbing state** that always produces a reward of zero:



- We can cover all cases by writing $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$,

where γ can be 1 only if a zero reward absorbing state is always reached.

The Markov Property

- By "the state" at step t, the book means whatever information is available to the agent at step t about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should **summarize past sensations** so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\} =$$

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1}=s' \mid s_t=s, a_t=a\} \text{ for all } s,s' \in S, a \in A(s).$$

  - **expected rewards**:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t=s, a_t=a, s_{t+1}=s'\} \text{ for all } s,s' \in S, a \in A(s).$$

An Example Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if it runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

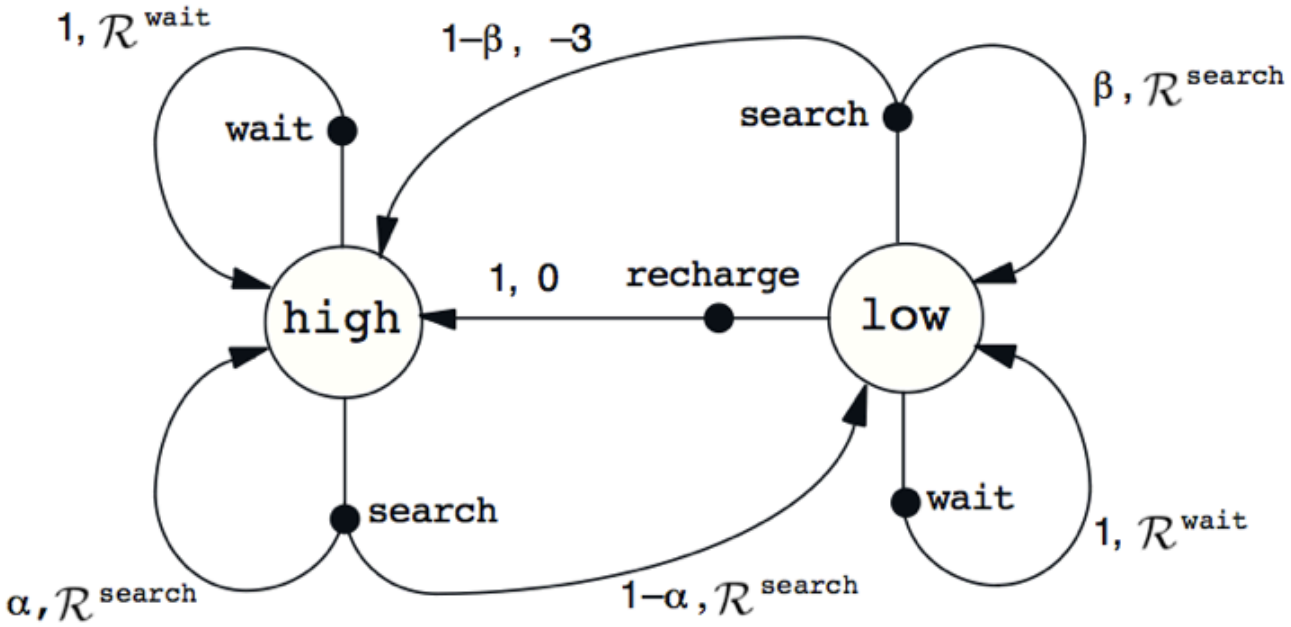
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$\mathcal{R}^{\texttt{search}}$ = expected no. of cans while searching

$\mathcal{R}^{\texttt{wait}}$ = expected no. of cans while waiting

$\mathcal{R}^{\texttt{search}} > \mathcal{R}^{\texttt{wait}}$



The value-function hypothesis

- All efficient methods for solving sequential decision problems determine (learn or compute) value functions as an intermediate step

Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State-value function for policy π :

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s\right\}$$

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π :

Action-value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a\right\}$$

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

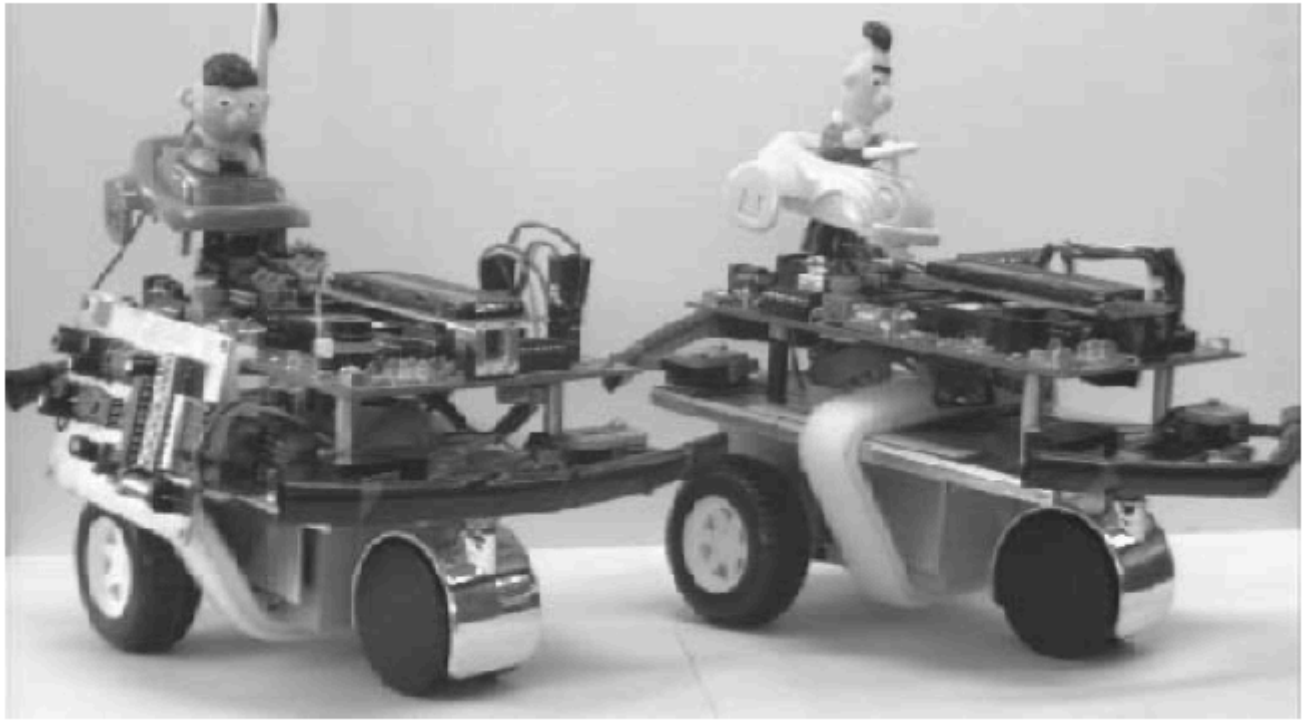
$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π^*.
- Optimal policies share the same **optimal state-value function**: $V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all} \quad s \in S$

- Optimal policies also share the same **optimal action-value function**: $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all} \quad s \in S \text{ and } a \in A(s)$

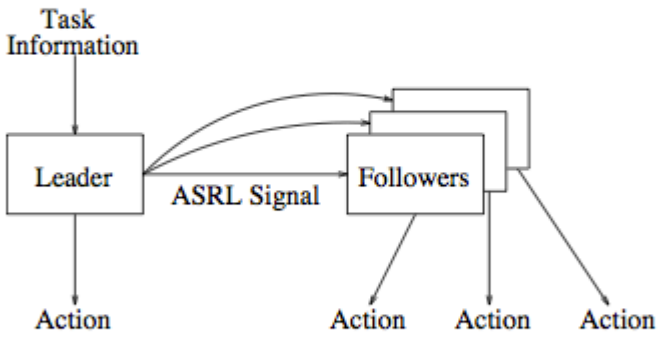
  This is the expected return for taking action a in state s and thereafter following an optimal policy.

Reinforcement Learning for Robot Language Learning



Holly Yanco and Lynn Andrea Stein. "An Adaptive Communication Protocol for Cooperating Mobile Robots." In *From Animals to Animats 2: Proceedings of the Second International Conference on the Simulation of Adaptive Behavior*, edited by J.-A. Meyer, H.L. Roitblat and S.W. Wilson. The MIT Press/Bradford Books, 1993, pp. 478-485.

Robot Pseudocode




```
Leader:

   loop:  on valid input signal from environment
             choose an action to perform
             choose a signal to send to follower via
                the radio boards
          wait for reinforcement signal
          on reinforcement signal
             increment variables for action and signal
          goto loop

Follower:

   loop:  on valid input signal from leader
             choose an action to perform
          wait for reinforcement signal
          on reinforcement signal
             increment variables for action
          goto loop
```

Example run with 2 robots

	Appropriate action	Leader's action	Leader's signal	Follower's action	Reinforcement
1.	↑↑	*spin*	low	*spin*	−
2.	○○	*spin*	low	*straight*	−
3.	↑↑	*straight*	high	*spin*	−
4.	○○	*straight*	high	*straight*	−
5.	○○	*spin*	low	*spin*	+
6.	↑↑	*straight*	high	*spin*	−
7.	○○	*spin*	low	*spin*	+
8.	○○	*spin*	low	*spin*	+
9.	○○	*spin*	low	*spin*	+
10.	↑↑	*spin*	low	*spin*	−
11.	↑↑	*straight*	high	*straight*	+
12.	↑↑	*straight*	high	*straight*	+
13.	○○	*spin*	low	*spin*	+

Curse of dimensionality

Size of Language	Number of Iterations to Convergence		
	Average	Minimum	Maximum
2	15.34	10	24
3	110.30	33	501
4	340.38	53	990
5	906.62	255	2472
10	15011.61	2868	51031
20	232267.82	44196	1241767

Table 2: Learning times for a two member troupe. Experiments for each language size were run 100 times.

Size of Language	Number of Iterations to Convergence		
	Average	Minimum	Maximum
2	27.21	10	80
3	327.71	35	1211
4	1530.12	340	6666
5	4415.60	652	17533
10	163530.62	37130	705029

Table 3: Data above is for a three member troupe and was collected over 100 runs for each language size.