

# Reinforcement Learning

Frank Dellaert

Most slides adapted from Michael Littman

# Reinforcement Learning

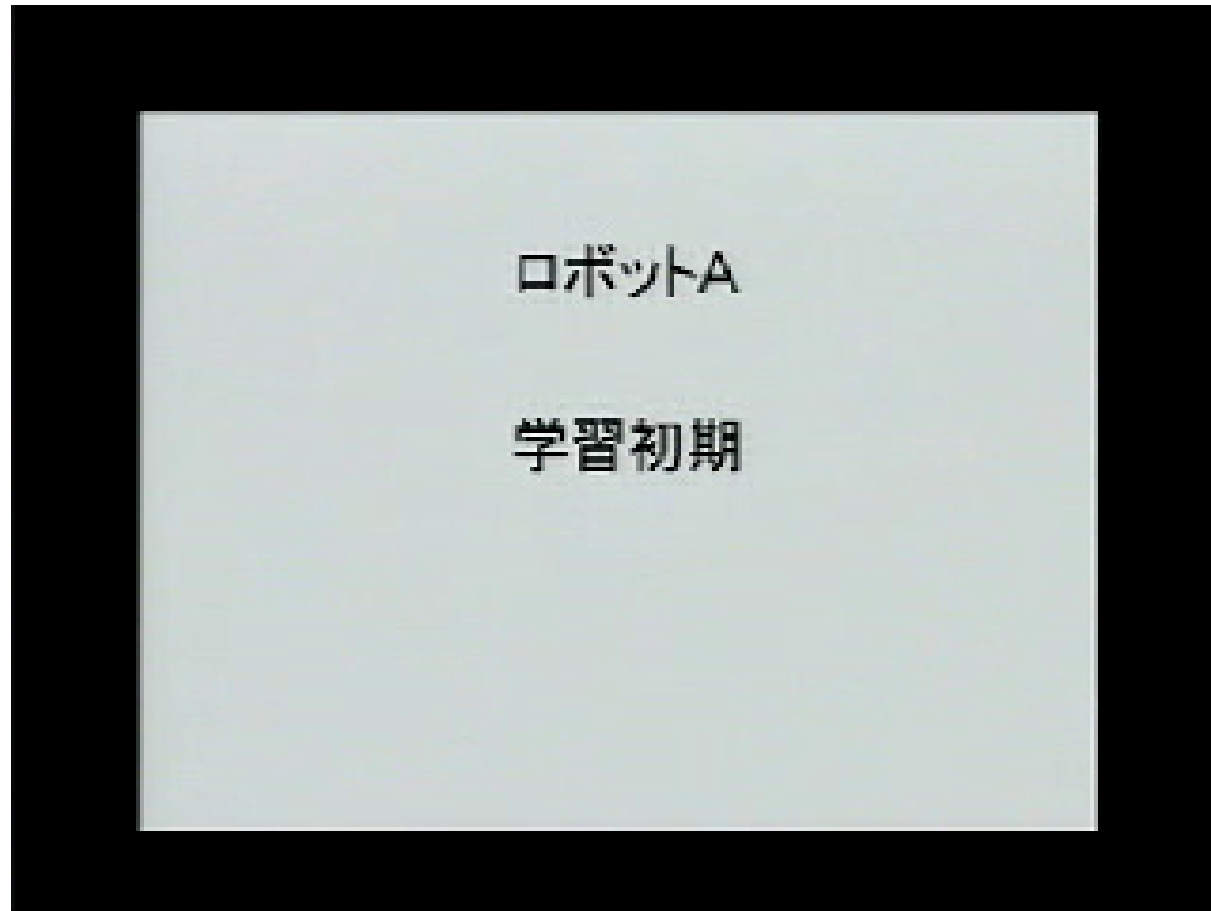
- Learning Control
- Policies that choose optimal actions
- Q learning
- Convergence

# Learning Control

Consider learning to choose actions, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

# Application: Robot Behavior



# Application: Learning to Walk



# Learning for Stable Vision



# Application: Helicopter Flight



# Problem Characteristics

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/actuators

# One Example: TD-Gammon

[Tesauro, 1995]

Learn to play Backgammon

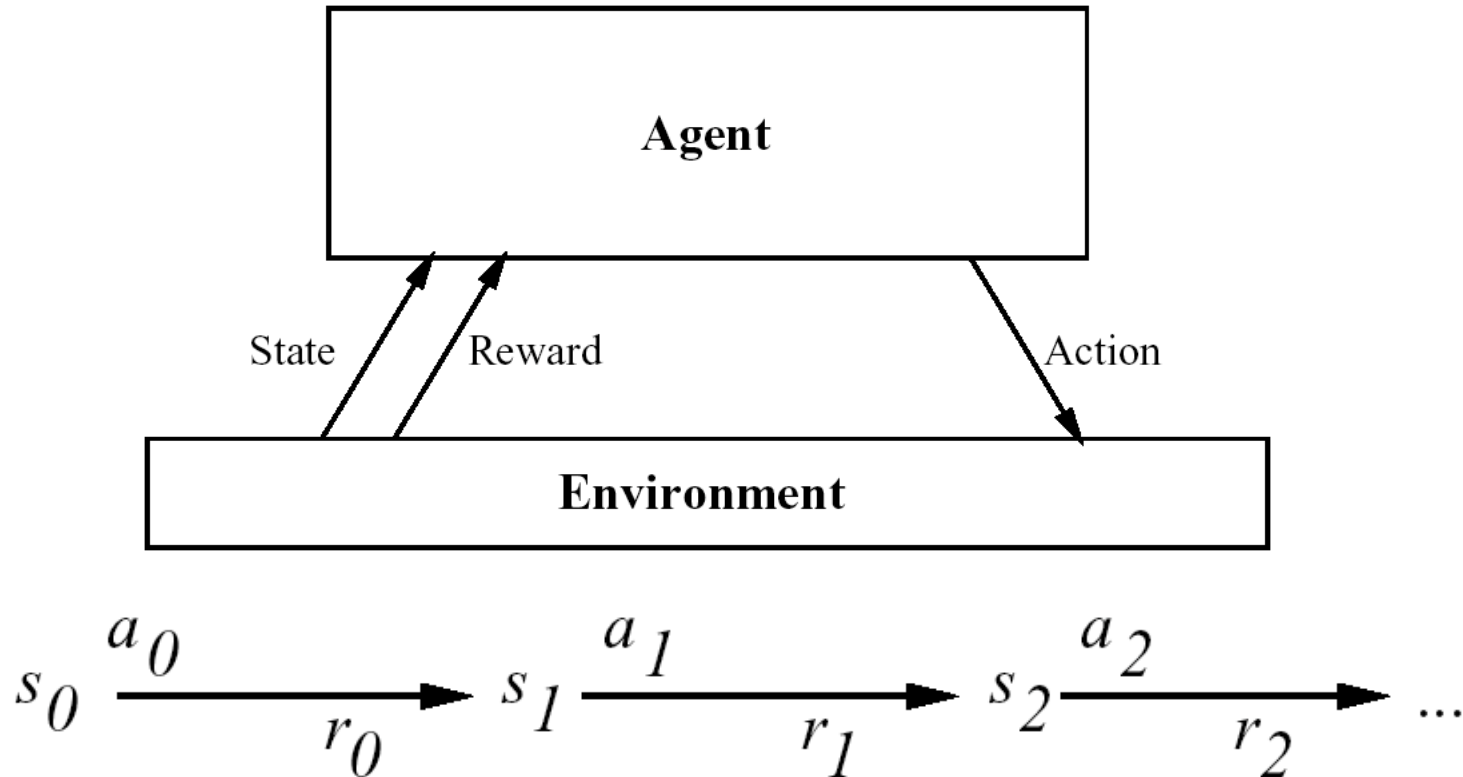
Immediate reward

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games against itself

Now approximately equal to best human player

# The RL Problem



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$

# Markov Decision Processes

Assume

- finite set of states  $S$ ; set of actions  $A$
- at each discrete time agent observes state  $s_t$  in  $S$  and chooses action  $a_t$  in  $A$
- then receives immediate reward  $r_t$  & state changes to  $s_{t+1}$
- Markov assumption:
  - $r_t = r(s_t, a_t)$  and  $s_{t+1} = \delta(s_t, a_t)$  depend only on *current* state and action
  - $\delta$  and  $r$  may be nondeterministic
  - $\delta$  and  $r$  not necessarily known to agent

# A Dialogue View

**Environment:** You are in state 65. You have 4 possible actions.  
**Agent:** I'll take action 2.  
**Environment:** You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions.  
**Agent:** I'll take action 1.  
**Environment:** You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.  
**Agent:** I'll take action 2.  
**Environment:** You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.  
⋮ ⋮

# Agent's Learning Task

Execute actions in environment, observe results, and

- learn action policy  $\pi: S \rightarrow A$  that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

from any starting state in  $S$

- here  $0 \leq \gamma < 1$  is the discount factor for future rewards

# Learning Problem

Note:

- Target function is  $\pi: S \rightarrow A$
- but we have no training examples of form  $\langle s, a \rangle$
- training examples are of form  $\langle \langle s, a \rangle, r \rangle$

# Value Function

To begin, consider deterministic worlds...

For each possible policy  $\pi$  the agent might adopt, we can define an evaluation function over states

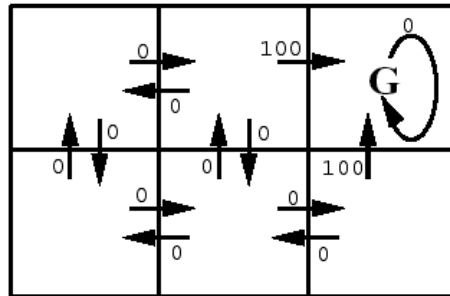
$$\begin{aligned} V^\pi(s) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

where  $r_t, r_{t+1}, \dots$  are generated by following policy  $\pi$  starting at state  $s$

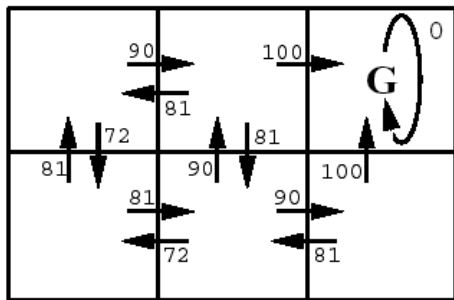
Restated, the task is to learn the optimal policy  $\pi^*$ :

$$\pi^* \equiv \operatorname{argmax}_{\pi} V^\pi(s), (\forall s).$$

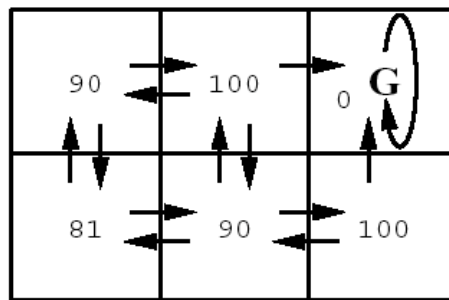
# Example MDP



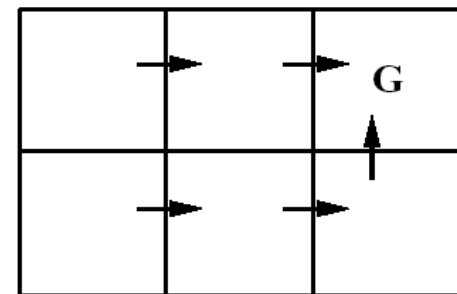
$r(s, a)$  (immediate reward) values



$Q(s, a)$  values



$V^*(s)$  values



One optimal policy

# What to Learn

We might try to have agent learn the evaluation function  $V^{\pi^*}$  (we write as  $V^*$ )

It could then do a lookahead search to choose best action from any state  $s$  because

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

A problem:

- This works well if agent knows  $\delta: S \times A \rightarrow S$ , and  $r: S \times A \rightarrow \mathfrak{R}$
- But when it doesn't, it can't choose actions this way

# Value Iteration

```
initialize  $V(s)$  arbitrarily
loop until policy good enough
  loop for  $s \in \mathcal{S}$ 
    loop for  $a \in \mathcal{A}$ 
       $Q(s, a) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V(s')$ 
       $V(s) := \max_a Q(s, a)$ 
    end loop
  end loop
end loop
```

From Kaelbling et al 96

# Policy Iteration

```
choose an arbitrary policy  $\pi'$ 
loop
   $\pi := \pi'$ 
  compute the value function of policy  $\pi$ :
    solve the linear equations
      
$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s')$$

    improve the policy at each state:
      
$$\pi'(s) := \arg \max_a (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_\pi(s'))$$

until  $\pi = \pi'$ 
```

# Q Function

Define new function very similar to  $V^*$

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

If agent learns  $Q$ , it can choose optimal action even without knowing  $\delta$ !

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

$Q$  is the evaluation function the agent will learn

# Training Rule to Learn Q

Note  $Q$  and  $V^*$  closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write  $Q$  recursively as

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Nice! Let  $Q$  denote learner's current approximation to  $Q$ .

Use training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where  $s'$  is the state resulting from applying action  $a$  in state  $s$

# Q Learning in Deterministic Case

For each  $s, a$  initialize table entry  $Q(s, a) \leftarrow 0$

Observe current state  $s$

Do forever:

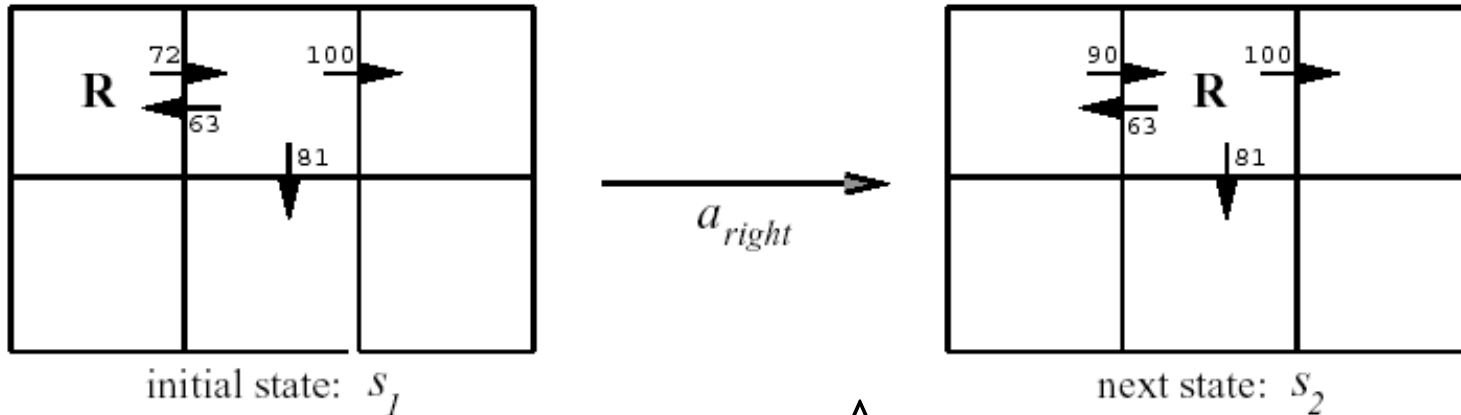
- Select an action  $a$  and execute it
- Receive immediate reward  $r$
- Observe the new state  $s'$
- Update the table entry for  $Q(s, a)$  via:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

- $s \leftarrow s'$

^

# Updating Q



$$Q(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} Q(s_2, a')$$

$$\leftarrow 0 + 0.9 \max \{63, 81, 100\} = 90$$

notice if rewards non-negative, then

$$(\forall s, a, n) Q_{n+1}(s, a) \geq Q_n(s, a)$$

and

$$(\forall s, a, n) 0 \leq Q_n(s, a) \leq Q(s, a)$$

# Convergence Proof

$\hat{Q}$  converges to  $Q$ . Consider case of deterministic world where see each  $\langle s, a \rangle$  visited infinitely often.

*Proof:* Define a full interval to be an interval during which each  $\langle s, a \rangle$  is visited. During each full interval the largest error in  $Q$  table is reduced by factor of  $\gamma$

Let  $Q_n$  be table after  $n$  updates, and  $\Delta_n$  be the maximum error in  $Q_n$ ; that is

$$\Delta_n = \max_{s,a} |Q_n(s, a) - Q(s, a)|$$

# Proof Continued

For table entry  $Q_n(s, a)$  updated on iteration  $n + 1$ , the error in the revised estimate  $Q_{n+1}(s, a)$  is

$$\begin{aligned} & |Q_{n+1}(s, a) - Q(s, a)| \\ &= |(r + \gamma \max_{a'} Q_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\ &= \gamma |\max_{a'} Q_n(s', a') - \max_{a'} Q(s', a')| \\ &\leq \gamma \max_{a'} |Q_n(s', a') - Q(s', a')| \\ &\leq \gamma \max_{a', s''} |Q_n(s'', a') - Q(s'', a')| \\ &|Q_{n+1}(s, a) - Q(s, a)| \leq \gamma \Delta_n \end{aligned}$$

Note that we used the fact that

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

# Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine  $V, Q$  by taking expected values

$$\begin{aligned} V^\pi(s) &\equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots] \\ &\equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \end{aligned}$$

$$Q(s, a) \equiv E[r(s, a) + \gamma V^*(\delta(s, a))]$$

# Nondeterministic Case

Q learning generalizes to nondeterministic worlds

Alter training rule to  $\hat{Q}$

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')]$$

where

$$\alpha_n = 1 / (1 + \text{visits}_n(s, a)).$$

Can still prove convergence of  $\hat{Q}$  to  $\hat{Q}$   
[Watkins and Dayan, 1992].