# Reinforcement Learning and Markov Decision Processes

Ronald J. Williams
CSG220, Spring 2007

Contains a few slides adapted from two related Andrew Moore tutorials found at http://www.cs.cmu.edu/~awm/tutorials



## What is reinforcement learning?

Key Features:

- Agent interacts continually with its environment
- Agent has access to performance measure, not told how it should behave

  "That was a 3.5"

- Performance measure depends on sequence of actions chosen

  "Hmm, I wonder where I went wrong …"

  - Temporal credit assignment problem
- Not everything known to the agent in advance => learning required

## What is reinforcement learning?

- Tasks having these properties have come to be called *reinforcement learning* tasks
- A reinforcement learning agent is one that improves its performance over time in such tasks

## Historical background

- Original motivation: animal learning
- Early emphasis: neural net implementations and heuristic properties
- Now appreciated that it has close ties with
  - operations research
  - optimal control theory
  - dynamic programming
  - AI state-space search
- Best formalized as a set of techniques to handle *Markov Decision Processes (MDPs)* or *Partially Observable Markov Decision Processes (POMDPs)*

## Reinforcement learning task


$$s(0) \xrightarrow[r(0)]{a(0)} s(1) \xrightarrow[r(1)]{a(1)} s(2) \xrightarrow[r(2)]{a(2)} \ldots$$

Goal: Learn to choose actions that maximize the cumulative reward

$$r(0) + \gamma\, r(1) + \gamma^2\, r(2) + \ldots$$

where $0 \leq \gamma \leq 1$.

$\gamma$ = discount factor




## Markov Decision Process (MDP)

- Finite set of states $S$
- Finite set of actions $A$ *
- Immediate reward function

$$R : S \times A \rightarrow \text{Reals}$$

- Transition (next-state) function

$$T : S \times A \rightarrow S$$

- More generally, $R$ and $T$ are treated as stochastic
  - We'll stick to the above notation for simplicity
  - In general case, treat the immediate rewards and next states as random variables, take expectations, etc.

* The theory easily allows for the possibility that there are different sets of actions available at each state. For simplicity we use one set for all states.

# Markov Decision Process

- If no rewards and only one action, this is just a Markov chain
- Sometimes also called a *Controlled Markov Chain*
- Overall objective is to determine a *policy*

$$\pi : S \rightarrow A$$

such that some measure of cumulative reward is optimized




# What's a policy?

| If agent is in this state | Then a good action is |
|---|---|
| $s_1$ | $a_3$ |
| $s_2$ | $a_7$ |
| $s_3$ | $a_1$ |
| $s_4$ | $a_3$ |
| . . . | . . . |

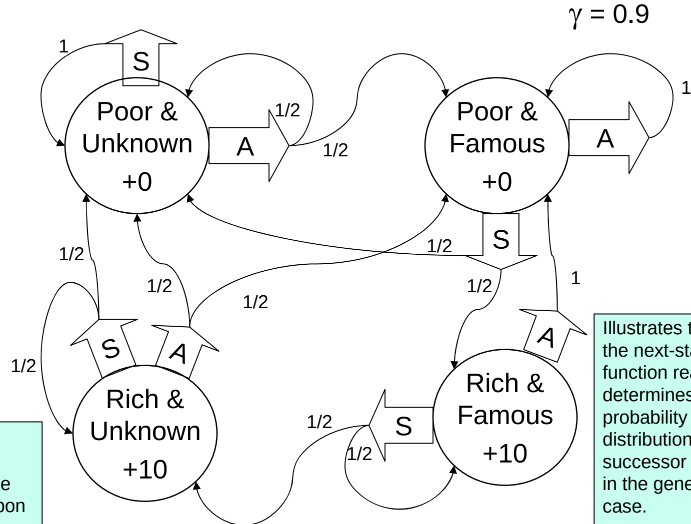
Note: To be more precise, this is called a *stationary* policy because it depends only on the state. The policy might depend, say, on the time step as well. Such policies are sometimes useful; they're called *nonstationary* policies.

# A Markov Decision Process

γ = 0.9

You run a startup company.

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



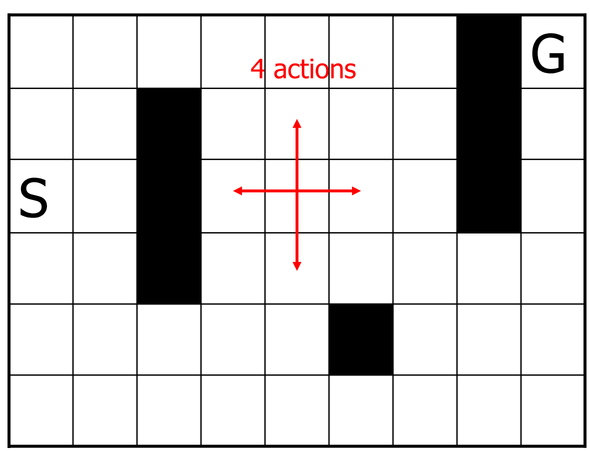

Here the reward shown inside any state represents the reward received upon entering that state.

Illustrates that the next-state function really determines a probability distribution over successor states in the general case.





# Another MDP




47 states

Reward = -1 at every step                γ = 1

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

# Applications of MDPs

Many important problems are MDPs….

… Robot path planning
… Travel route planning
… Elevator scheduling
… Bank customer retention
… Autonomous aircraft navigation
… Manufacturing processes
… Network switching & routing

And many of these have been successfully handled using RL methods




# From a situated agent's perspective

- At time step *t*
  - Observe that I'm in state *s(t)*
  - Select my action *a(t)*
  - Observe resulting immediate reward *r(t)*
- Now time step is *t+1*
  - Observe that I'm in state *s(t+1)*
  - etc.

## Value Functions

- It turns out that
  - RL theory
  - MDP theory
  - AI game-tree search

  all agree on the idea that evaluating states is a useful thing to do.
- A *(state) value function* *V* is any function mapping states to real numbers:

$$V : S \to \text{Reals}$$

## A special value function: the return

- For any policy $\pi$, define the *return* to be the function $V^{\pi} : S \to \text{Reals}$ assigning to each state the quantity

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r(t)$$

> Reminder: Use expected values in the stochastic case.

  where
  - *s(0) = s*
  - each action *a(t)* is chosen according to $\pi$
  - each subsequent *s(t+1)* arises from the transition function *T*
  - each immediate reward *r(t)* is determined by the immediate reward function *R*
  - $\gamma$ is a given discount factor in [0, 1]

## Technical remarks

- If the next state and/or immediate reward functions are stochastic, then the *r(t)* values are random variables and the return is defined as the expectation of this sum
- If the MDP has absorbing states, the sum may actually be finite
  - We stick with this infinite sum notation for the sake of generality
  - The discount factor can be taken to be 1 in absorbing-state MDPs
  - The formulation we use is called *infinite-horizon*




## Why the discount factor?

- Models idea that future rewards are not worth quite as much the longer into the future they're received
  - used in economic models
- Also models situations where there is a nonzero fixed probability 1-$\gamma$ of termination at any time
- Makes the math work out nicely
  - with bounded rewards, sum guaranteed to be finite even in infinite-horizon case

# What's a value function?

| If agent starts in this state | Return when following given policy should be |
|---|---|
| $s_1$ | 13 |
| $s_2$ | -1 |
| $s_3$ | 22.6 |
| $s_4$ | 6 |
| . . . | . . . |

Note: It is common to treat any value function as an *estimate* of the return from some policy since that's what's usually desired.




# Optimal Policies

- Objective: Find a policy $\pi^*$ such that

$$V^{\pi^*}(s) \geq V^{\pi}(s)$$

  for any policy $\pi$ and any state *s.*
- Such a policy is called an *optimal* policy.
- Define

$$V^* = V^{\pi^*}$$

optimal return or
optimal value function

## Interesting fact

For every MDP 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.

Can you see why this is true?

## Finding an Optimal Policy

Idea One:

Run through all possible policies.

Select the best.

What's the problem ??

## Finding an Optimal Policy

- Dynamic Programming approach:
  - Determine the optimal return (optimal value function) for each state
  - Select actions "greedily" according to this optimal value function V*
- How do we compute V*?
  - Magic words: *Bellman equation(s)*




## Bellman equations

For any state *s* and policy $\pi$

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(T(s, \pi(s)))$$

For any state *s*,

$$V^{*}(s) = \max_{a}\{R(s,a) + \gamma V^{*}(T(s,a))\}$$

Extremely important and useful recurrence relations

Can be used to compute the return from a given policy or to compute the optimal return via *value iteration*

## Quick and dirty derivation of the Bellman equation

Given the state transition s $\longrightarrow$ s′,

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r(t)$$

$$= r(0) + \gamma \sum_{t=0}^{\infty} \gamma^t r(t+1)$$

$$= r(0) + \gamma V^{\pi}(s')$$




## Bellman equations: general form

For completeness, here are the Bellman equations for stochastic MDPs:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P_{ss'}(\pi(s)) V^{\pi}(s')$$

$$V^{*}(s) = \max_{a}\{R(s,a) + \gamma \sum_{s'} P_{ss'}(a) V^{*}(s')\}$$

where $R(s,a)$ now represents $E(r \mid s,a)$ and

$P_{ss'}(a) =$ probability that the next state is *s′* given that action *a* is taken in state *s*.

## From values to policies

- Given *any* function $V : S \to \mathrm{Reals}$, define a policy $\pi$ to be *greedy* for *V* if, for all *s*,

$$\pi(s) = \arg\max_a \{R(s,a) + \gamma V(T(s,a))\}$$

- The right-hand side can be viewed as a 1-step lookahead estimate of the return from $\pi$ based on the estimated return from successor states

Yet another reminder: In the general case, this is a shorthand for the appropriate expectations as spelled out in detail on the previous slide.




## Facts about greedy policies

- An optimal policy is greedy for $V^*$
  - Follows from Bellman equation
- If $\pi$ is not optimal then a greedy policy for $V^\pi$ will yield a larger return than $\pi$
  - Not hard to prove
  - Basis for another DP approach to finding optimal policies: *policy iteration*

## Finding an optimal policy

**Value Iteration Method**

Choose any initial state value function $V_0$

Repeat for all $n \geq 0$

For all $s$

$$V_{n+1}(s) \leftarrow \max_a \{R(s,a) + \gamma V_n(T(s,a))\}$$

Until convergence

This converges to $V^*$ and any greedy policy with respect to it will be an optimal policy

Just a technique for solving the Bellman equations for $V^*$ (system of $|S|$ nonlinear equations in $|S|$ unknowns)




## Finding an optimal policy

**Policy Iteration Method**

Choose any initial policy $\pi_0$

Repeat for all $n \geq 0$

Compute $V^{\pi_n}$

Choose $\pi_{n+1}$ greedy with respect to $V^{\pi_n}$

Until $V^{\pi_{n+1}} = V^{\pi_n}$

Can you prove that this terminates with an optimal policy?

# Finding an optimal policy

Policy Iteration Method

Choose any initial policy $\pi_0$

Repeat for all $n \geq 0$

Compute $V^{\pi_n}$ (Policy Evaluation Step)

Choose $\pi_{n+1}$ greedy with respect to $V^{\pi_n}$ (Policy Improvement Step)

Until $V^{\pi_{n+1}} = V^{\pi_n}$

Can you prove that this terminates with an optimal policy?

# Evaluating a given policy

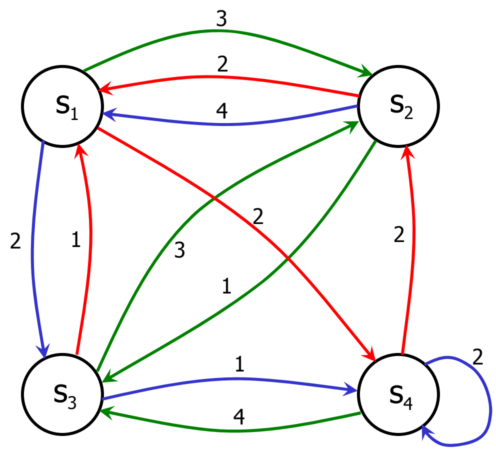
- There are at least 2 distinct ways of computing the return for a given policy $\pi$
  - Solve the corresponding system of linear equations (the Bellman equation for $V^{\pi}$)
  - Use an iterative method analogous to value iteration but with the update

$$V_{n+1}(s) \leftarrow R(s,\pi(s)) + \gamma V_n(T(s,\pi(s)))$$

- First way makes sense from an offline computational point of view
- Second way relates to online RL

# Deterministic MDP to Solve




3 actions at each state:

$a_1$, $a_2$, $a_3$

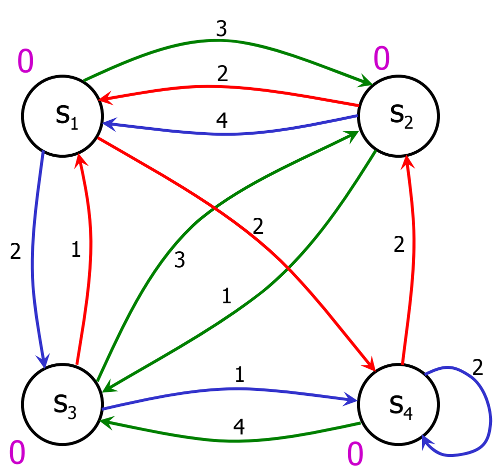
Numbers on arcs denote immediate reward received

Find optimal policy when γ = 0.9



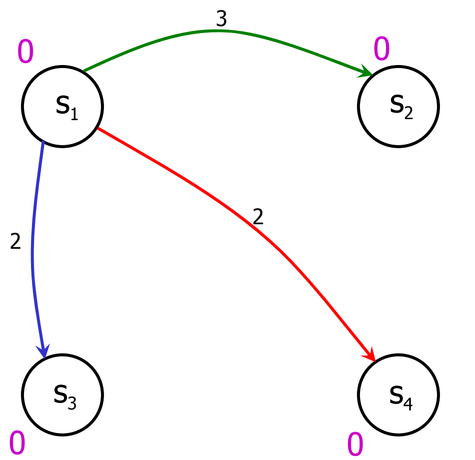


# Value Iteration




Arbitrary initial value function $V_0$

# Value Iteration




Arbitrary initial value function $V_0$

Computing a new value for $s_1$ using 1-step lookahead with previous values:

For action $a_1$ lookahead value is $2 + (.9)(0) = 2$

For action $a_2$ lookahead value is $3 + (.9)(0) = 3$

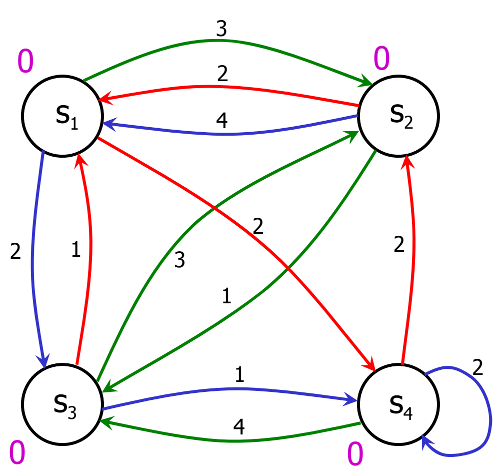
For action $a_3$ lookahead value is $2 + (.9)(0) = 2$

| $a_1$ | $a_2$ | $a_3$ |
|---|---|---|
| 2 | 3 | 2 |

$$V_1(s_1) = \max\{2,3,2\} = 3$$



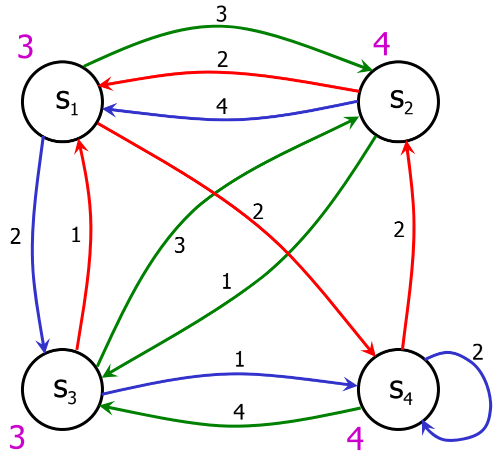

# Value Iteration




Arbitrary initial value function $V_0$

| Lookahead value along action | | | | |
|---|---|---|---|---|
| | $a_1$ | $a_2$ | $a_3$ | **max** |
| $s_1$ | 2 | 3 | 2 | **3** |
| $s_2$ | 2 | 1 | 4 | **4** |
| $s_3$ | 1 | 3 | 1 | **3** |
| $s_4$ | 2 | 4 | 2 | **4** |

# Value Iteration



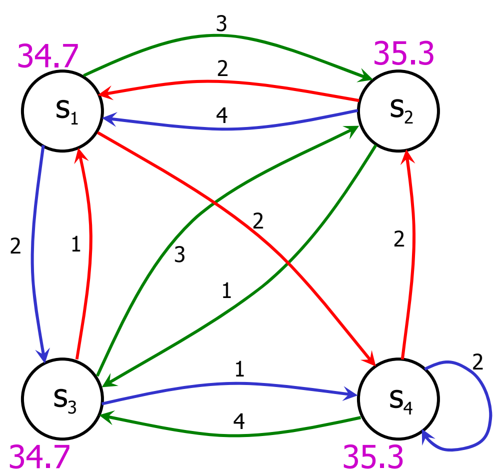

Updated approximation to V*:

$V_1(s_1)=3$
$V_1(s_2)=4$
$V_1(s_3)=3$
$V_1(s_4)=4$

New value function $V_1$ after one step of value iteration



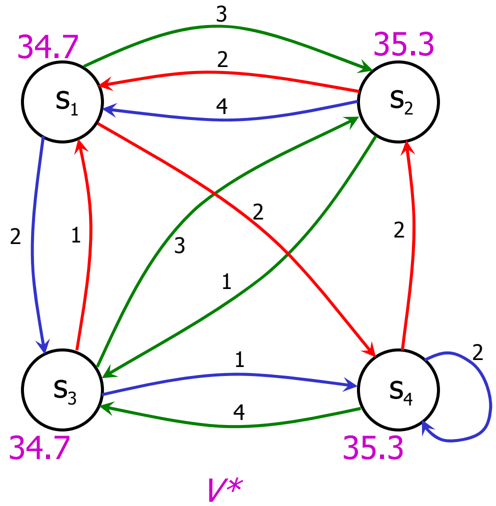

# Value Iteration




| | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| $V_0$ | 0 | 0 | 0 | 0 |
| $V_1$ | 3 | 4 | 3 | 4 |
| $V_2$ | 6.6 | 6.7 | 6.6 | 6.7 |
| $V_3$ | 9.0 | 9.9 | 9.0 | 9.9 |
| $V_4$ | 11.9 | 12.1 | 11.9 | 12.1 |
| $V_5$ | 13.9 | 14.8 | 13.9 | 14.8 |
| . . . | | | | |
| $V^*$ | 34.7 | 35.3 | 34.7 | 35.3 |

Keep doing this until it converges to $V^*$

# Value Iteration

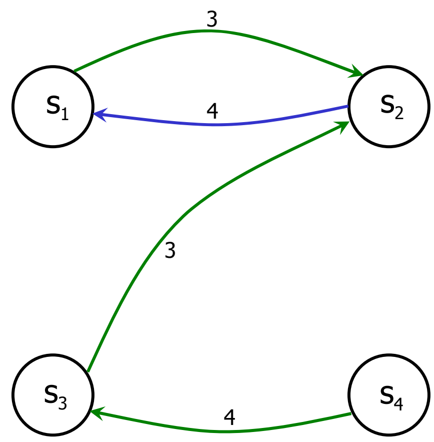



Determining a greedy policy for *V**

| Lookahead value along action | | | | |
|---|---|---|---|---|
| | $a_1$ | $a_2$ | $a_3$ | **best** |
| $s_1$ | 33.8 | 34.8 | 33.2 | $\mathbf{a_2}$ |
| $s_2$ | 33.2 | 32.2 | 35.2 | $\mathbf{a_3}$ |
| $s_3$ | 32.2 | 34.8 | 32.8 | $\mathbf{a_2}$ |
| $s_4$ | 33.8 | 35.2 | 33.8 | $\mathbf{a_2}$ |

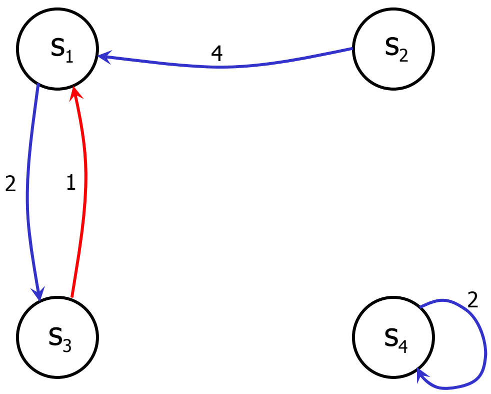



# Value Iteration




Optimal policy

# Policy Iteration



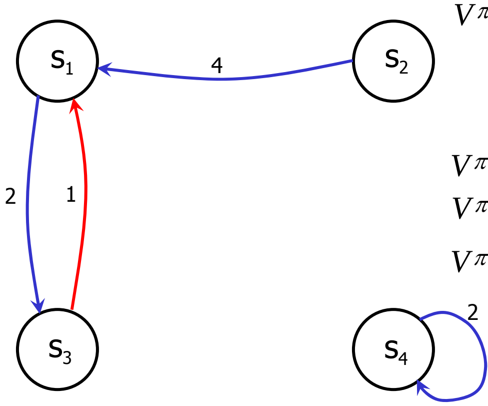

Start with this policy $\pi$




# Policy Iteration

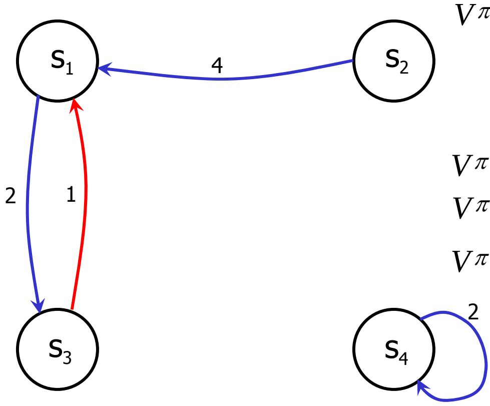
Compute its return:

$$V^{\pi}(s_1) = 2 + .9 \cdot 1 + (.9)^2 \cdot 2 + (.9)^3 + \cdots$$
$$= (2 + .9)[1 + (.9)^2 + (.9)^4 + \cdots]$$
$$= \frac{2.9}{1 - .81} = 15.3$$
$$V^{\pi}(s_2) = 4 + (.9) \cdot V^{\pi}(s_1) = 17.7$$
$$V^{\pi}(s_3) = 1 + (.9) \cdot V^{\pi}(s_1) = 14.7$$
$$V^{\pi}(s_4) = \frac{2}{1 - .9} = 20$$




Start with this policy $\pi$

# Policy Iteration




Compute its return:

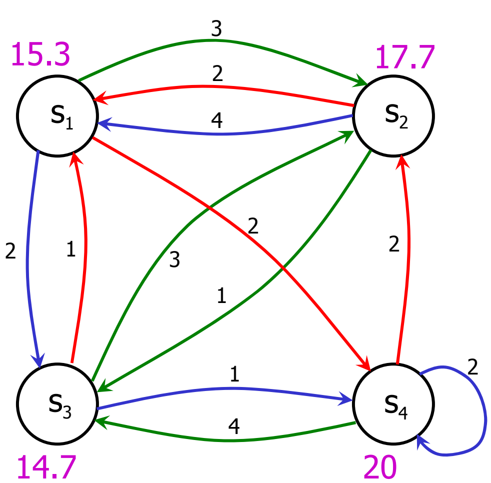
$$V^{\pi}(s_1) = 2 + .9 \cdot 1 + (.9)^2 \cdot 2 + (.9)^3 + \cdots$$
$$= (2 + .9)[1 + (.9)^2 + (.9)^4 + \cdots]$$
$$= \frac{2.9}{1 - .81} = 15.3$$
$$V^{\pi}(s_2) = 4 + (.9) \cdot V^{\pi}(s_1) = 17.7$$
$$V^{\pi}(s_3) = 1 + (.9) \cdot V^{\pi}(s_1) = 14.7$$
$$V^{\pi}(s_4) = \frac{2}{1 - .9} = 20$$

Start with this policy $\pi$

Really just solving a system of linear equations



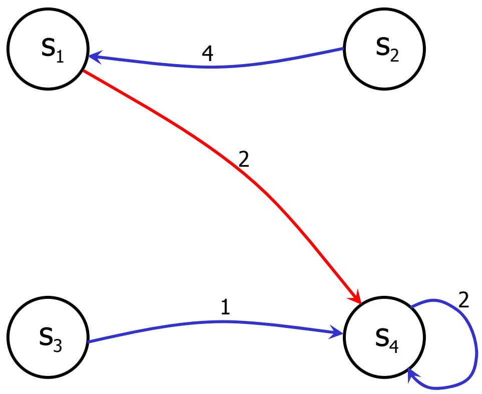

# Policy Iteration




Determining a greedy policy for $V^{\pi}$

| Lookahead value along action | | | | |
|---|---|---|---|---|
| | $a_1$ | $a_2$ | $a_3$ | **best** |
| $s_1$ | 20.0 | 18.9 | 15.2 | $\mathbf{a_1}$ |
| $s_2$ | 15.8 | 14.2 | 17.8 | $\mathbf{a_3}$ |
| $s_3$ | 14.8 | 18.9 | 19.0 | $\mathbf{a_3}$ |
| $s_4$ | 17.9 | 17.2 | 20.0 | $\mathbf{a_3}$ |

# Policy Iteration




New policy after one step of policy iteration

# Policy Iteration vs. Value Iteration: Which is better?

It depends.

- Lots of actions? Policy Iteration
- Already got a fair policy? Policy Iteration
- Few actions, acyclic? Value Iteration

Best of Both Worlds:

Modified Policy Iteration [Puterman]

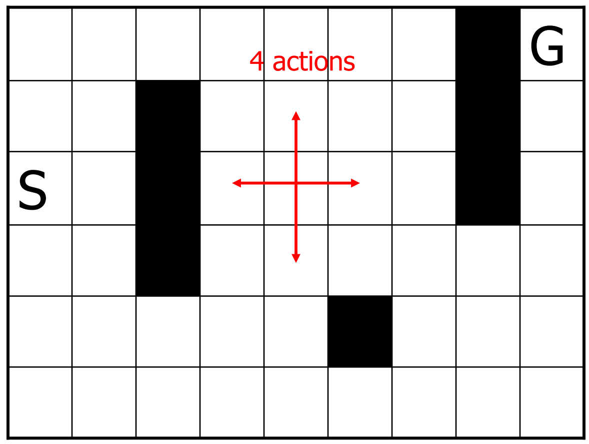
…a simple mix of value iteration and policy iteration

3rd Approach

Linear Programming

# Maze Task




Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic




# Maze Task

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 86 | 87 | 88 | 89 | 90 | 91 | 92 | | 100 | G |
| | 85 | 86 | | 90 | 91 | 92 | 93 | | 99 | |
| S | 86 | 87 | | 91 | 92 | 93 | 94 | | 98 | |
| | 87 | 88 | | 92 | 93 | 94 | 95 | 96 | 97 | |
| | 88 | 89 | 90 | 91 | 92 | | 94 | 95 | 96 | |
| | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | |

$V^*$

What's an optimal path from S to G?

# Maze Task

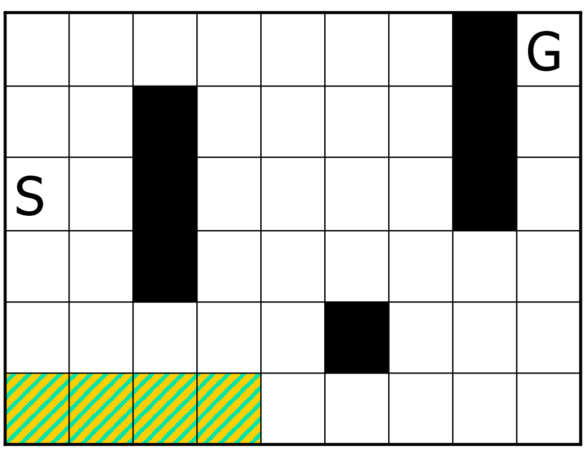
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 86 | 87 | 88 | 89 | 90 | 91 | 92 | | 100 | G |
| | 85 | 86 | | 90 | 91 | 92 | 93 | | 99 | |
| S | 86 | 87 | | 91 | 92 | 93 | 94 | | 98 | |
| | 87 | 88 | | 92 | 93 | 94 | 95 | 96 | 97 | |
| | 88 | 89 | 90 | 91 | 92 | | 94 | 95 | 96 | |
| | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | |

$V^*$





# Another Maze Task



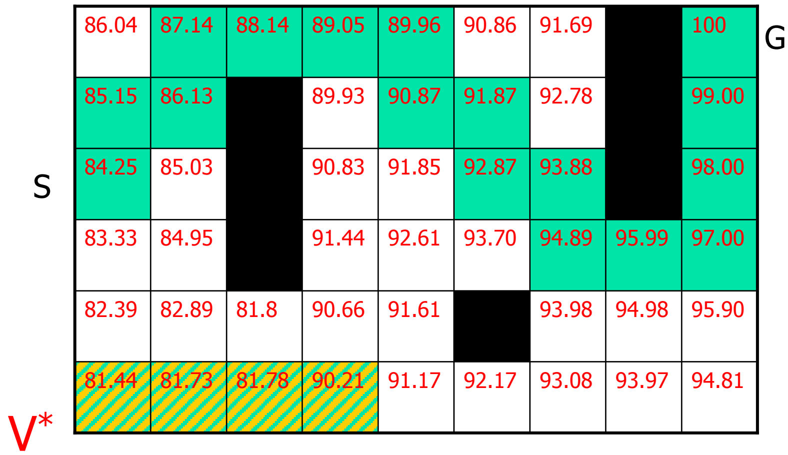

Now what's an optimal path from S to G?

Everything else same as before, except:

With some nonzero probability, a small wind gust might displace the agent one cell to the right or left of its intended direction of travel on any step

Entering any of the 4 patterned cells at the southwest corner yields a reward of -100

# Another Maze Task




With probability 0.2, a small wind gust might displace the agent one cell to the right or left of its intended direction of travel on any step

Entering any of the 4 patterned cells at the southwest corner yields a reward of -100




# State-action values (Q-values)

- Note that in this example it's misleading to consider optimal *path* – especially since randomness may knock the agent off it at any time
- To use these state values to choose actions, need to consult transition function *T* for each action at the current state, then choose the one giving the best expected cumulative reward
- Alternative approach: For this example, at each state keep track of 4 numbers, not just 1, corresponding to each possible action – best action is the one with the highest such state-action value

## Q-Values

- For any policy $\pi$, define $Q^\pi : S \times A \rightarrow \text{Reals}$

  by $Q^\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t r(t)$

  Once again, the correct expression for a general MDP should use expected values here

  where the initial state *s(0)* = *s*, the initial action *a(0)* = *a*, and all subsequent states, actions, and rewards arise from the transition, policy, and reward functions, respectively.
- Just like $V^\pi$ except that action *a* is taken as the very first step and only after this is policy $\pi$ followed
- Bellman equations can be rewritten in terms of Q-values

© 2004, Ronald J. Williams Reinforcement Learning: Slide 51

## Q-Values (cont.)

- Define $Q^* = Q^{\pi^*}$, where $\pi^*$ is an optimal policy.
- There is a corresponding Bellman equation for $Q^*$ since

  $$V^*(s) = \max_a Q^*(s,a)$$
- Given any state-action value function *Q*, define a policy $\pi$ to be greedy for *Q* if

  $$\pi(s) = \arg\max_a Q(s,a)$$

  for all *s*.
- An optimal policy is greedy for $Q^*$
- Ultimately just a convenient reformulation of the Bellman equation

Why it's convenient will become apparent once we start discussing learning

© 2004, Ronald J. Williams Reinforcement Learning: Slide 52

# What are Q-values?

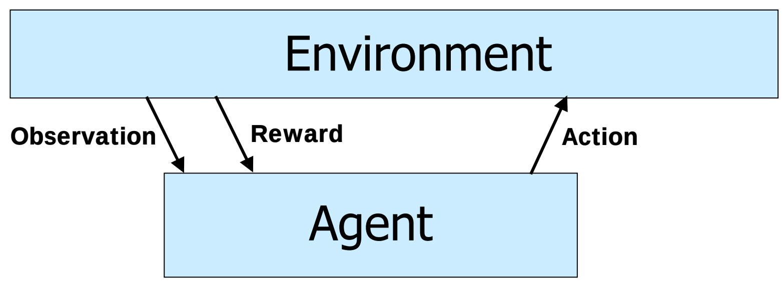
| If agent is in this state | And starts with this action and then follows the policy | Return should be |
|---|---|---|
| $s_1$ | $a_1$ | -5 |
| $s_1$ | $a_2$ | 3 |
| $s_2$ | $a_1$ | 17.1 |
| $s_2$ | $a_2$ | 10 |
| . . . | . . . | . . . |




# Where's the learning?

- So far, just looking at how to solve MDPs and how such solutions lead to optimal choices of action
- Before getting to learning, let's take a peek beyond MDPs: POMDPs
- More realistic but much harder to solve

# More General RL Task




$$o(0) \xrightarrow[r(0)]{a(0)} o(1) \xrightarrow[r(1)]{a(1)} o(2) \xrightarrow[r(2)]{a(2)} \dots$$

Goal: Learn to choose actions that maximize the cumulative reward

$$r(0) + \gamma\, r(1) + \gamma^2\, r(2) + \dots$$

$\gamma$ = discount factor

where $0 \le \gamma \le 1$.




# Partially Observable Markov Decision Process

- Set of states $S$
- Set of observations $O$
- Set of actions $A$
- Immediate reward function

$$R : S \times A \to \text{Reals}$$

- Transition (next-state) function

$$T : S \times A \to S$$

- Observation function

$$B : S \to O$$

- More generally, $R$, $T$, and $B$ are stochastic

## POMDP (cont.)

- Ideally, want a policy mapping all possible histories to a choice of actions that optimizes the cumulative reward measure
- In practice, settle for policies that choose actions based on some amount of memory of past actions and observations
- Special case: *reactive policies*
  - Map most recent observation to a choice of action
  - Also called *memoryless policies*




## What's a reactive policy?

| If agent observes this | Then a good action is |
| --- | --- |
| $o_1$ | $a_3$ |
| $o_2$ | $a_7$ |
| $o_3$ | $a_1$ |
| $o_4$ | $a_3$ |
| . . . | . . . |

# Maze Task with Perceptual Aliasing

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1100 | 0100 | 0110 | 0100 | 0100 | 0100 | 0101 | | 1101 | G |
| | 1000 | 0001 | | 1000 | 0000 | 0000 | 0001 | | 1001 | |
| S | 1000 | 0001 | | 1000 | 0000 | 0000 | 0001 | | 1001 | |
| | 1000 | 0001 | | 1000 | 0000 | 0010 | 0000 | 0010 | 0001 | |
| | 1000 | 0000 | 0100 | 0000 | 0001 | | 1000 | 0000 | 0001 | |
| | 1010 | 0010 | 0010 | 0010 | 0010 | 0110 | 0010 | 0010 | 0011 | |

Can sense if there is a wall immediately to east, north, south, or west

Represented as a corresponding 4-bit string

Only 12 distinct possible observations

Turns this maze task into a POMDP



# POMDP Theory

- In principle, can convert any POMDP into an MDP with states = belief states
- Belief state is a function: *S -> Reals* assigning to any *s* the probability that actual state is *s*
- Drawback: Even if underlying state space is finite (say, n states), space of belief states is an (n-1)-dimensional simplex. Solving this continuous-state MDP is much too hard.

# Practical approaches to POMDPs

- Use certain MDP methods, treating observations like states, and hope for the best
- Try to determine how much past history to store to represent actual states, then treat as an MDP (involves inference of hidden state, as in *hidden Markov models*)
  - history window
  - finite-state memory
  - recurrent neural nets
- Do direct policy search in a restricted set of policies (e.g., reactive policies)

Revisit this briefly later




- Now back to the observable state case ...

## AI state space planning

- Traditionally, true world model available *a priori*
- Consider all possible sequences of actions starting from current state up to some horizon – forms a tree
- Evaluate the states reached at the leaves
- Find the best, and choose the first action in that sequence
- How should non-terminal states be evaluated?
  - $V^*$ would be ideal
  - But then only 1 step of lookahead would be necessary
- Usual perspective: use depth of search to make up for imperfections in state evaluation
- In control engineering, called *receding horizon* controller




## Once again, where's the learning?

- Patience – we're almost there

## Backups

- Term used in the RL literature for any updating of *V(s)* by replacing it by

$$R(s,a)+\gamma V(T(s,a))$$

where *a* is some action, which also includes the possibility of replacing it by

$$\max_a \{R(s,a)+\gamma V(T(s,a))\}$$

- Closely related to notion of backing up values in a game tree




## Backups

- Term used in the RL literature for any updating of *V(s)* by replacing it by

$$R(s,a)+\gamma V(T(s,a))$$

Sometimes call this a *backup along action a*

where *a* is some action, which also includes the possibility of replacing it by

Sometimes call this a *max-backup*

$$\max_a \{R(s,a)+\gamma V(T(s,a))\}$$

- Closely related to notion of backing up values in a game tree

## Backups

- The operation of backing up values is one of the primary links between MDP theory and RL methods
- Some key facts making these classical MDP algorithms relevant to online learning
  - value iteration consists solely of (max-)backup operations
  - policy evaluation step in policy iteration can be performed solely with backup operations (along the policy)
  - backups modify the value at a state solely based on the values at successor states




## Synchronous vs. asynchronous

- The value iteration and policy iteration algorithms demonstrated here use *synchronous* backups, but asynchronous backups (implementable by "updating in place") can also be shown to work
- Value iteration and policy iteration can be seen as two ends of a spectrum
- Many ways of interleaving backup steps and policy improvement steps can be shown to work, but not all (Williams & Baird, 1993)

## Generalized Policy Iteration

- GPI coined to apply to the wide range of RL algorithms that combine simultaneous updating of values and policies in intuitively reasonable ways
- It is known that not every possible GPI algorithm converges to an optimal policy
- However, only known counterexamples are contrived
- Remains an open question whether some of the ones found successful in practice are mathematically guaranteed to work




## Generalized Policy Iteration

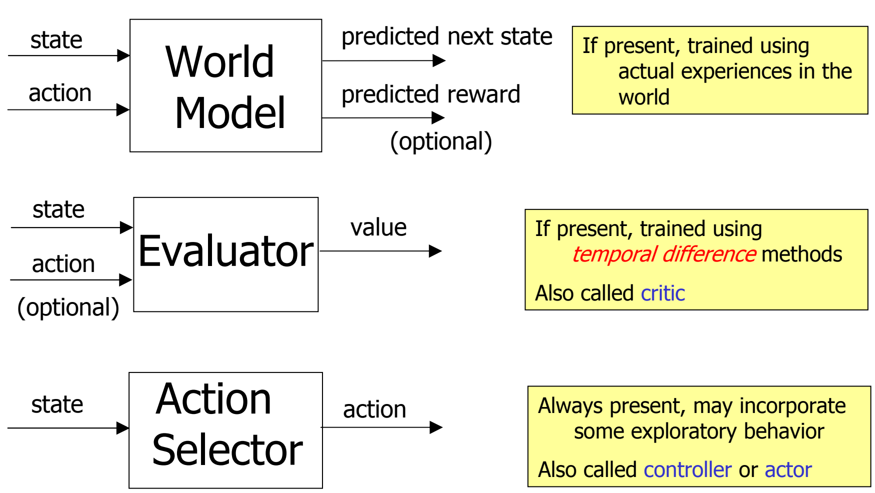
| If agent is in this state | Estimated best action | Estimated optimal return |
|---|---|---|
| $s_1$ | $a_7$ | -5 |
| $s_2$ | $a_3$ | 3 |
| $s_3$ | $a_4$ | 17.1 |
| $s_4$ | $a_1$ | 10 |
| . . . | . . . | . . . |

# Learning – Finally!

- Almost everything we've discussed so far is "classical" MDP (or POMDP) theory
  - Transition, reward functions known *a priori*
  - Issue is purely one of (off-line) *planning*
- Four ways RL theory goes beyond this
  - Assume transition and/or reward functions not known *a priori* – must be discovered through environmental interactions
  - Try to address tasks for which classical approach is intractable
  - Take seriously the idea that policy and/or values not represented simply using table lookup
  - Even when $T$ and $R$ are known, only do a kind of *online planning* in parts of state space actually experienced




# Internal components of a RL agent



## Unknown transition and/or reward functions

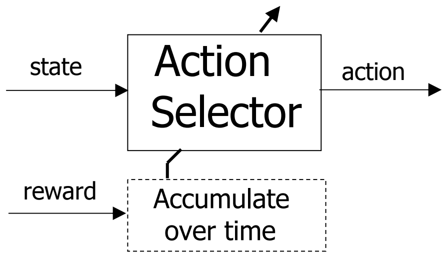
- One possibility: Learn the MDP through exploration, then solve it (*plan*) using offline methods: *learn-then-plan* approach
- Another way: Never represent anything about the MDP itself, just try to learn the values directly: *model-free* approach
- Yet another possibility: Interleave learning of the MDP with planning – every time the model changes, re-plan as if current model is correct: *certainty-equivalence planning*
- Many approaches to RL can be viewed as trying to blend learning and planning more seamlessly




## What about directly learning a policy?

- One possibility: Use supervised learning
  - Where do training examples come from?
  - Need prior expertise
  - What if set of actions is different in different states? (e.g. games) *may be difficult to represent the policy*
- Another possibility: generate and test
  - Search the space of policies, evaluating many candidates
    - Genetic algorithms, genetic programming, e.g.
    - Policy-gradient techniques
  - Upside:
    - can work even in POMDPs
  - Downside:
    - the space of policies may be way too big
    - evaluating each one individually may be too time-consuming

# Direct policy search




- Model-free *and* value-free
- Can be used for POMDPs as well
- Requires that action selector have a way to explore policy space

- Many possible approaches
  - Genetic algorithms
  - Policy gradient




- For the rest of this lecture, we focus solely on RL approaches using value functions:
  - Temporal difference methods
  - Q-learning
  - Actor/critic systems
  - RL as a blend of learning and planning

## Temporal Difference Learning

[Sutton 1988]

Only maintain a $V$ array… nothing else

So you've got

$V(s_1), V(s_2), \cdots V(s_n)$

and you observe

$s \xrightarrow{r} s'$

what should you do?

A transition from s that receives an immediate reward of r and jumps to s'

*Can You Guess ?*




## TD Learning

After making a transition from $s$ to $s'$ and receiving reward $r$, we nudge $V(s)$ to be closer to the estimated return based on the observed successor, as follows:

$$V(s) \leftarrow \alpha(r + \gamma V(s')) + (1-\alpha)V(s)$$

$\alpha$ is called a "learning rate" parameter.

For $\alpha < 1$ this represents a *partial backup*.

Furthermore, if the rewards and/or transitions are stochastic, as in a general MDP, this is a *sample backup*.

The reward and next-state values are only noisy estimates of the corresponding expectations, which is what offline DP would use in the appropriate computations (*full backup*).

Nevertheless, this converges to the return for a fixed policy (under the right technical assumptions, including decreasing learning rate)

# TD(λ)

- Updating the value at a state based on just the succeeding state is actually the special case TD(0) of a parameterized family of TD methods
- TD(1) updates the value at a state based on *all* succeeding states
- For 0 < λ < 1, TD(λ) updates a state's value base on all succeeding states, but to a lesser extent the further into the future
- Implemented by maintaining decaying *eligibility traces* at each state visited (decay rate = λ)
- Helps distribute credit for future rewards over all earlier actions

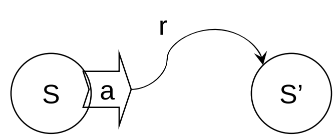
Can help mitigate effects of violation of Markov property




# Model-free RL

Why not use TD on state values?

Observe




update

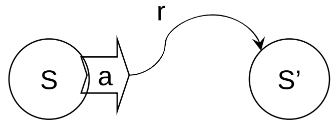
$$V(s) \leftarrow \alpha\left(r + \gamma V(s')\right) + (1-\alpha)V(s')$$

What's wrong with this?

# Model-free RL

Why not use TD on state values?

Observe




update

$$V(s) \leftarrow \alpha(r + \gamma V(s')) + (1-\alpha)V(s')$$

What's wrong with this?

1. Still can't choose actions without knowing what next state (or distribution over next states) results: requires an internal model of $T$
2. The values learned will represent the return for the policy we've followed, including any suboptimal exploratory actions we've taken: not clear this will t help us act optimally




# But ...

- Recall our earlier definition of Q-values:

## Q-values

- For any policy $\pi$, define $Q^{\pi} : S \times A \rightarrow \text{Reals}$

  by $Q^{\pi}(s,a) = \sum_{t=0}^{\infty} \gamma^t r(t)$

  Once again, the correct expression for a general MDP should use expected values here

  where the initial state *s(0)* = *s*, the initial action *a(0)* = *a*, and all subsequent states, actions, and rewards arise from the transition, policy, and reward functions, respectively.
- Just like $V^{\pi}$ except that action *a* is taken as the very first step and only after this is policy $\pi$ followed




## Q-values

- Define $Q^* = Q^{\pi^*}$, where $\pi^*$ is an optimal policy.
- There is a corresponding Bellman equation for $Q^*$ since

$$V^*(s) = \max_a Q^*(s,a)$$

- Given any state-action value function *Q*, define a policy $\pi$ to be greedy for *Q* if

$$\pi(s) = \arg\max_a Q(s,a)$$

  for all *s*.
- An optimal policy is greedy for $Q^*$

## Q-learning

(Watkins, 1988)

- Assume no knowledge of *R* or *T*.
- Maintain a table-lookup data structure Q (estimates of Q*) for all state-action pairs
- When a transition $s \xrightarrow{r} s'$ occurs, do

$$Q(s,a) \leftarrow \alpha\left(r + \gamma \max_{a'} Q(s',a')\right) + (1-\alpha)Q(s,a)$$

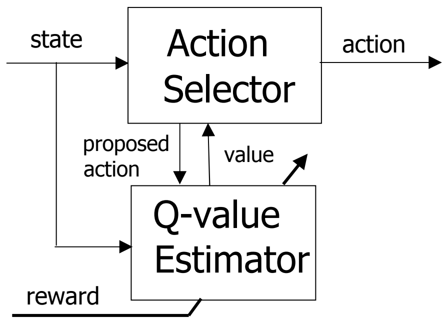
- Essentially implements a kind of asynchronous Monte Carlo value iteration, using sample backups
- Guaranteed to eventually converge to Q* as long as every state-action pair sampled infinitely often




## Q-learning

- This approach is even cleverer than it looks: the Q values are not biased by any particular exploration policy. It avoids the credit assignment problem.
- The convergence proof extends to any variant in which every Q(s,a) is updated infinitely often, whether on-line or not.

# Q-learning Agent




- Action selector trivial: queries Q-values to find action for current state with highest value
- Occasionally also takes exploratory actions

- Model-free: Does not need to know the effects of actions




# Using Estimated Optimal Q-values

| If agent is in this state | And starts with this action and then follows the optimal policy thereafter | Return should be |
|---|---|---|
| $s_1$ | $a_1$ | -5 |
| $s_1$ | $a_2$ | 3 |
| $s_2$ | $a_1$ | 17.1 |
| $s_2$ | $a_2$ | 10 |
| . . . | . . . | . . . |

# Q-Learning: Choosing Actions

- Don't always be greedy
- Don't always be random (otherwise it will take a long time to reach somewhere exciting)
- Boltzmann exploration [Watkins]

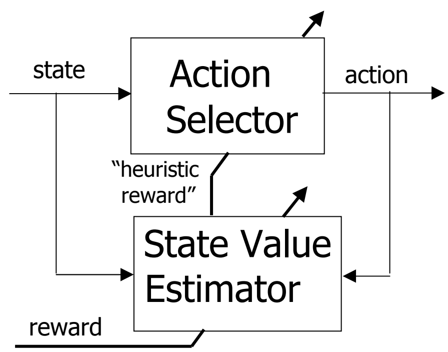
  $$\text{Prob(choose action a)} \propto \exp\left(-\frac{Q(s,a)}{K_t}\right)$$

- With some small probability, pick random action; else pick greedy action (called *ε-greedy* policy)
- Optimism in the face of uncertainty [Sutton '90, Kaelbling '90]
  - Initialize Q-values optimistically high to encourage exploration
  - Or take into account how often each (s,a) pair has been tried




# Another Model-free RL Approach

"Actor/Critic" (Barto, Sutton & Anderson, 1983)




- Action selector implements a *randomized* policy
- Its parameters are adjusted based on a reward/penalty scheme
- No definitive theoretical analysis yet available, but has been found to work in practice
- Represents a specific instance of generalized policy iteration (extended to randomized policies)

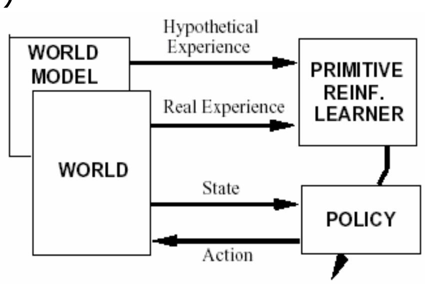
## Learning or planning?

- Classical DP emphasis for optimal control
  - Dynamics and reward structure known
  - Off-line computation
- Traditional RL emphasis
  - Dynamics and/or reward structure initially unknown
  - On-line learning
- Computation of an optimal policy off-line with known dynamics and reward structure can be regarded as planning

## Primitive use of a learned model: DYNA

(Sutton, 1990)

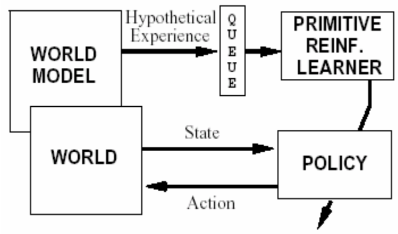



- In this diagram, *primitive* just means model-free
- Seamlessly integrates learning and planning
- World model can just be stored past transitions
- Main purpose is to improve efficiency over a model-free RL agent without incorporating a sophisticated model-learning component

# Priority DYNA

(Williams & Peng, 1993; Moore & Atkeson, 1993)




- Original DYNA used randomly selected transitions
- Efficiency improved significantly by prioritizing value updating along transitions in parts of state space most likely to improve performance fastest
- In goal-state tasks updating may occur in breadth-first fashion backwards from goal, or like A* working backwards, depending on how priority is defined



# Beyond table lookup

- Why not table lookup?
  - Too many states (even if finitely many)
  - Continuous state space
  - Want to be able to generalize – no hope of visiting every state, or computing something at every state
- Alternatives
  - State aggregation (e.g., quantization of continuous state spaces)
  - Generalizing function approximators
    - Neural networks (including variants like radial basis functions, tile codings)
    - Nearest neighbor methods
    - Decision trees

Bad news: very little theory to predict how well or poorly such techniques will perform

# Challenges

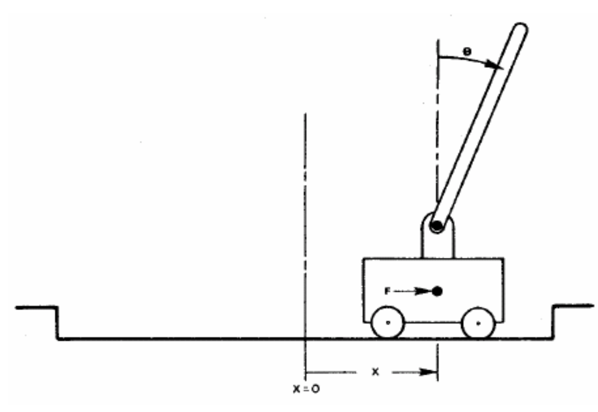
- How do we apply these techniques to infinite (e.g., continuous), or even just very large, state spaces?
  - Pole-balancer
  - Truck backer-upper
  - Mountain car (or puck-on-a-hill)
  - Bioreactor
  - Acrobot
  - Multi-jointed snake
  - Continuous mazes

Together with finite-state mazes of various kinds, these tasks have become benchmark test problems for RL techniques

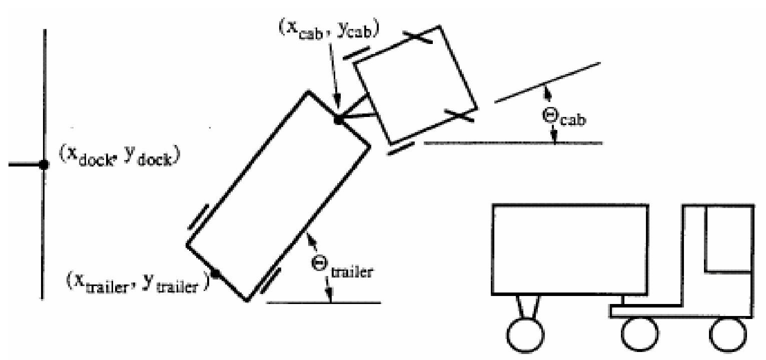
- Two basic approaches for continuous state spaces
  - Quantize (to obtain a finite-state approximation)
    - One promising approach: adaptive partitioning
  - Use function approximators (nearest-neighbor, neural networks, radial basis functions, tile codings, etc.)




# Pole balancer



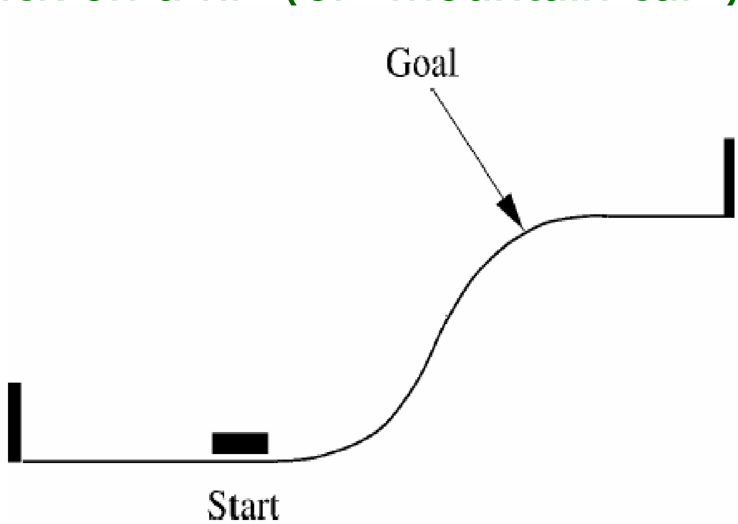
# Truck backer-upper



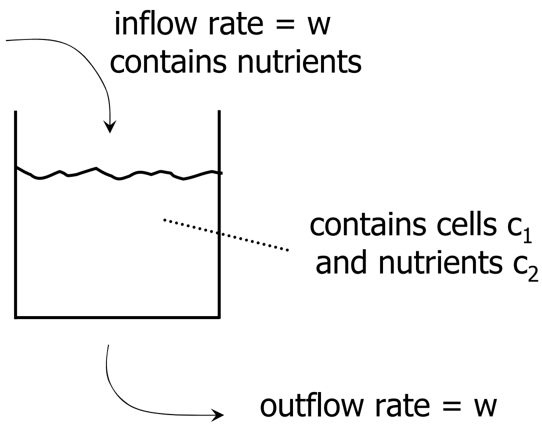




# Puck on a hill (or "mountain car")

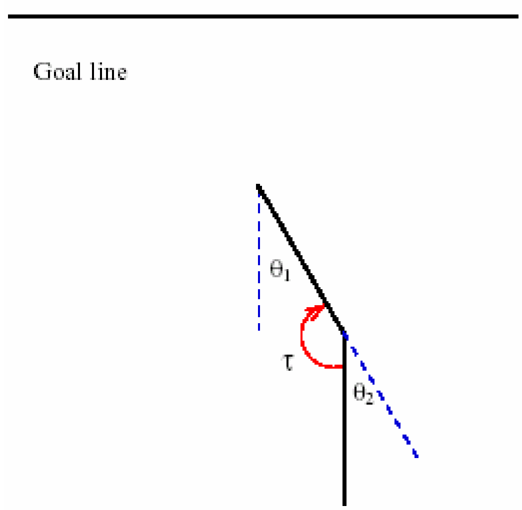


# Bioreactor



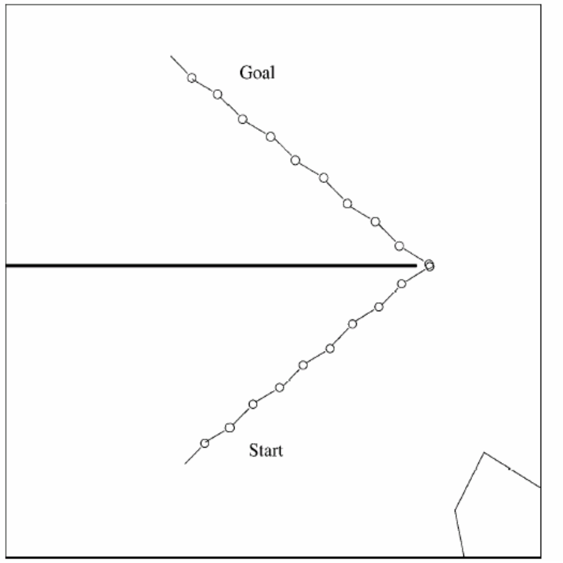

 

# Acrobot



# Multi-jointed "snake"



# Dealing with large numbers of states

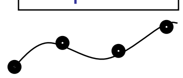
Don't use a Table…

| STATE | VALUE |
|---|---|
| $s_1$ | |
| $S_2$ | |
| : | |
| $S_{15122189}$ | |

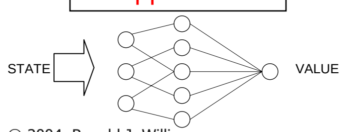
use…

(Generalizers)

Splines

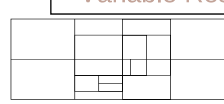


A Function Approximator



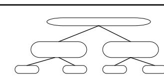

(Hierarchies)

Variable Resolution

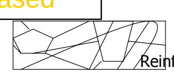


[Munos 1999]

Multi Resolution



Memory Based



## Function approximation for value functions

Polynomials → [Samuel, Boyan, Much O.R. Literature]

Neural Nets → [Barto & Sutton, Tesauro, Crites, Singh, Tsitsiklis]

Backgammon, Pole Balancing, Elevators, Tetris, Cell phones

Checkers, Channel Routing, Radio Therapy

Splines → Economists, Controls

Downside: All convergence guarantees disappear.




## Memory-based Value Functions

*V(s)* = *V* (most similar state in memory to *s* )

or

Average of *V* (20 most similar states)

or

Weighted Average of *V* (20 most similar states)

[Jeff Peng, Atkenson & Schaal,
Geoff Gordon, ← proved stuff
Scheider, Boyan & Moore 98]

"Planet Mars Scheduler"

# Hierarchical Methods

Continuous State Space: "Split a state when statistically significant that a split would improve performance"

Discrete Space:

Chapman & Kaelbling 92, McCallum 95 (includes hidden state)

A kind of Decision Tree Value Function

Continuous Space

e.g. Simmons et al 83, Chapman & Kaelbling 92, Mark Ring 94 …, Munos 96

with interpolation!

"Prove needs a higher resolution"

Moore 93, Moore & Atkeson 95

Multiresolution

A hierarchy with high level "managers" abstracting low level "servants"

<u>Many</u> O.R. Papers, Dayan & Sejnowski's Feudal learning, Dietterich 1998 (MAX-Q hierarchy) Moore, Baird & Kaelbling 2000 (airports Hierarchy)




# Open Issues

- Better ways to deal with very large state and/or action spaces
- Theoretical understanding of various practical GPI schemes
- Theoretical understanding of behavior when value function approximators used
- More efficient ways to integrate learning of dynamics and GPI
- Computationally tractable approaches when Markov property violated
- Better ways to learn and take advantage of hierarchical structure and modularity

## Valuable References

- Books
  - Bertsekas, D. P. & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific
  - Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press
- Survey paper
  - Kaelbling, L. P., Littman, M. & Moore, A. (1996). "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237-285. (Available as a link off the main Andrew Moore tutorials web page.)




## What You Should Know

- Definition of an MDP (and a POMDP)
- How to solve an MDP
  - using value iteration
  - using policy iteration
- Model-free learning (TD) for predicting delayed rewards
- How to formulate RL tasks as MDPs (or POMDPs)
- Q-learning (including being able to work through small simulated examples of RL)