Reinforcement Learning and Markov Decision Processes

Ronald J. Williams
CSG120, Fall 2003

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

What is reinforcement learning?

- A reinforcement learning agent
  - interacts with its environment
  - is goal-seeking
- The term *reinforcement learning* is used to characterize tasks having these properties
- A reinforcement learning algorithm is any algorithm for addressing such tasks

Historical background

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

Reinforcement learning task

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

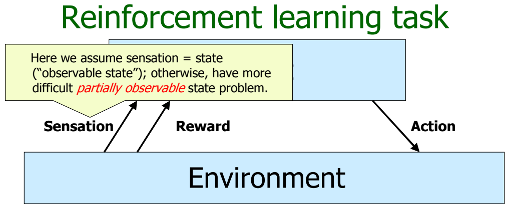
Goal: Learn to choose actions that maximize the cumulative reward

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

γ = discount factor

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

Reinforcement learning task



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

Goal: Learn to choose actions that maximize the cumulative reward

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

γ = discount factor

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

Markov Decision Process

- 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
. . .	. . .

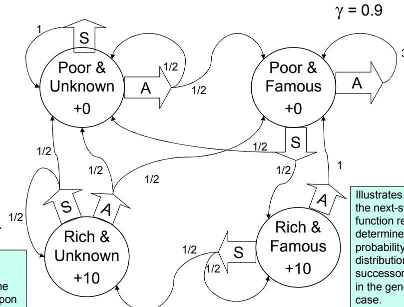
Note: To be more precise, this is called a *stationary* policy because it depends only on the state. The policy might depend, say, on the time step as well. Such policies are sometimes useful; they're called *nonstationary* policies.

A Markov Decision Process

$\gamma = 0.9$

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.

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.

Cumulative Reward

- Objective: Find a policy $\pi^* : S \rightarrow A$ maximizing, for every state *s*, the *return*

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

where
  - *s(0) = s*
  - each action *a(t)* is chosen according to π^*
  - each subsequent *s(t+1)* arises from the transition function *T*
  - each immediate reward *r(t)* is determined by the immediate reward function *R*
  - γ is a 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 of termination at any time
- Makes the math work out nicely
  - with bounded rewards, sum guaranteed to be finite even in infinite-horizon case

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?

Computing an Optimal Policy

Idea One:

Run through all possible policies.

Select the best.

What's the problem ??

Where's the learning?

- Standard MDP theory starts with knowledge of *R* and *T* and tries to solve for an optimal policy
  - can be viewed as planning using a known model
  - however, can be intractable for various reasons
  - even with *R* and *T* known, there may be reasons to use techniques developed in RL research to compute good policies
- What if *R* and/or *T* are not known?
  - this is basis of most RL research
  - look at this a lot more later

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)
- 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 non-MDP situations (e.g., POMDPs)
  - Downside: the space of policies may be way too big

Back to MDP theory ...

- 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:

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

State Value Functions

- For any policy π, define the state value function V^π by

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

Reminder: Use expected values in the stochastic case.

where the initial state is s and all subsequent states, actions, and rewards arise from the transition, policy, and reward functions, respectively.
- Define $V^* = V^{\pi^*}$, where π^* is an optimal policy.

Return from a policy

- V^π is the return (cumulative reward) obtained by following policy π (as a function of the start state)
- V^* is the optimal return (i.e., the return obtained by following an optimal policy)
- Recall that the return is the quantity we want to maximize

It can be shown that an optimal policy maximizes the return from all starting states. I.e., there is no policy that gives a higher return than the optimal policy when starting from some states but not when starting from others.

Bellman equations

For any state s and policy π

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

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 (Dynamic Programming)

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 state value function V, define a policy π 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 π 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 π is not optimal then a greedy policy for V^π will yield a larger return than π
  - Not hard to prove
  - Basis for policy iteration method

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 π_0

Repeat for all $n \geq 0$

Compute V^{π_n}

Choose π_{n+1} greedy with respect to V^{π_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 π_0

Repeat for all $n \geq 0$ Policy Evaluation Step

Compute V^{π_n} Policy Improvement Step

Choose π_{n+1} greedy with respect to V^{π_n}

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

Can you prove that this terminates with an optimal policy?

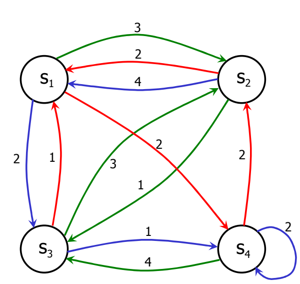
Evaluating a given policy

- There are at least 2 distinct ways of computing the return for a given policy π
  - Solve the corresponding system of linear equations (the Bellman equation for V^π)
  - 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



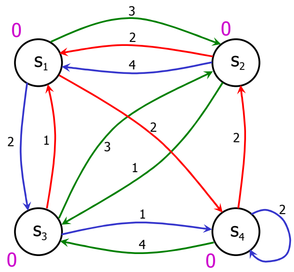
3 actions at each state:

a_1, a_2, a_3

Numbers on arcs denote immediate reward received

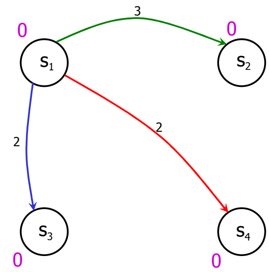
Find optimal policy when $\gamma = 0.9$

Value Iteration



Arbitrary initial value function V_0

Value Iteration



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 $3 + (.9)(0) = 3$

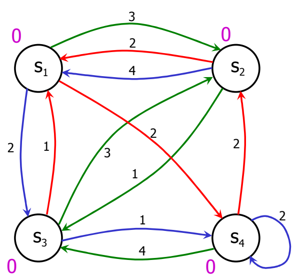
For action a_3 lookahead value is $2 + (.9)(0) = 2$

a_1	a_2	a_3
2	3	2

Arbitrary initial value function V_0

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

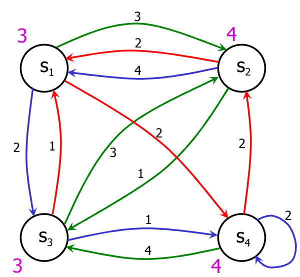
Value Iteration



	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**

Arbitrary initial value function V_0

Value Iteration

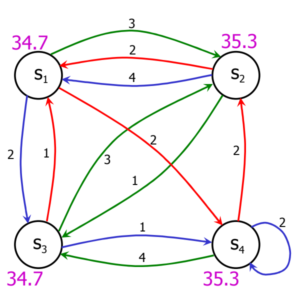


Updated approximation to V*:

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

New value function V_1 after one step of value iteration

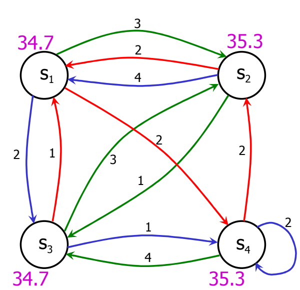
Value Iteration



	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

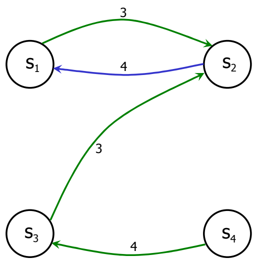


Determining a greedy policy for V^*

	Lookahead value along action			
	a_1	a_2	a_3	**best**
s_1	33.8	34.8	33.2	a_2
s_2	33.2	32.2	35.2	a_3
s_3	32.2	34.8	32.8	a_2
s_4	33.8	35.2	33.8	a_2

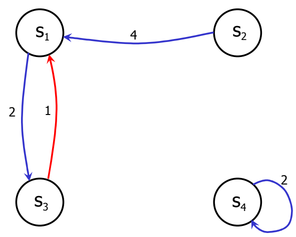
V^*

Value Iteration



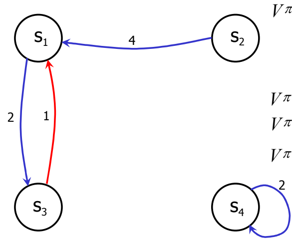
Optimal policy

Policy Iteration



Start with this policy π

Policy Iteration

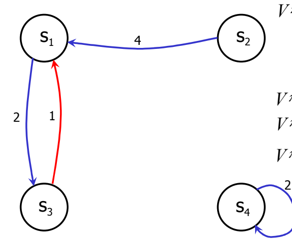


Compute its return:

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

Start with this policy π

Policy Iteration



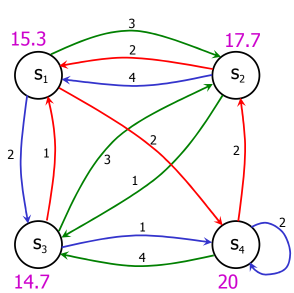
Compute its return:

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

Really just solving a system of linear equations

Start with this policy π

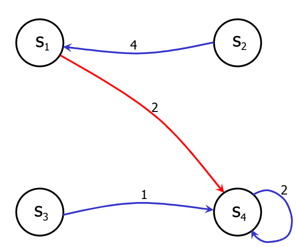
Policy Iteration



Determining a greedy policy for V^π

Lookahead value along action	a_1	a_2	a_3	best
s_1	20.0	18.9	15.2	**a_1**
s_2	15.8	14.2	17.8	**a_3**
s_3	14.8	18.9	19.0	**a_3**
s_4	17.9	17.2	20.0	**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? Choose 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

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

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

Sometimes call this a *max-backup*

- 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 implemented in practice can be guaranteed to work

Learning – Finally!

- Suppose a situated agent doesn't know the reward function *R* and/or the transition function *T* but only interacts with its environment
- What then?
  - One possibility: Learn the MDP through exploration, then solve it using offline methods
  - Another intriguing way: Never represent anything about the MDP itself, just try to learn the values directly – model free
  - These are 2 extremes in an interesting spectrum of possibilities

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

$s \xrightarrow{r} s'$

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

what should you do?

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:

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

α 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)

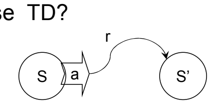
- Updating the value at a state based on just the succeeding state is actually the special case TD(0) of a parameterized family of TD methods
- TD(1) updates the value at a state based on *all* succeeding states
- For 0 < λ < 1, TD(λ) updates a state's value base on all succeeding states, but to a lesser 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?

Observe



update

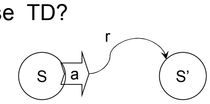
$$V(s) \leftarrow \alpha(r + \gamma V(s')) + (1-\alpha)V(s')$$

What's wrong with this?

Model-free RL

Why not use TD?

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: won't help us act optimally

State-Action Value Functions

- For any policy π, 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^{π} except that action *a* is taken as the very first step and only after this is policy π followed

State-Action Value Functions

- Define $Q^* = Q^{\pi^*}$, where π^* 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 π 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: 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]

  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

Two-component RL systems

- One of the earliest RL systems (pole balancer of Barto, Sutton & Anderson, 1983) had 2 components:
  - Adaptive Search Element (ASE)
  - Adaptive Critic Element (ACE)
- ASE essentially represents the policy
- ACE essentially represents the state value estimates – updated using TD(λ)
- Both components adapted on-line simultaneously
- Overall approach is a prime example of Generalized Policy Iteration
- No good mathematical analysis yet available for such 2-component systems

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

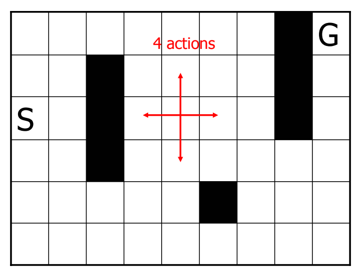
Integrating learning & planning

- Sutton's 1990 Dyna system introduced a seamless integration of RL and planning
- Stores a collection of transitions experienced
- Backups applied to
  - current on-line transition
  - plus a fixed number of other randomly chosen stored transitions
- Improvement on this idea
  - add a priority queue to prioritize backups along transitions in parts of state space most likely to improve performance fastest (Moore & Atkeson, 1993; Williams & Peng, 1993)

A toy problem

- The RL literature contains numerous examples of toy problems designed to shed light on the use of various techniques
- Here's one ...

Maze Learning Task



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

G is an absorbing state, terminating any single trial

Effect of actions is deterministic

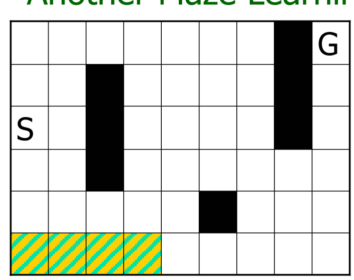
Maze Learning Task

-14	-13	-12	-11	-10	-9	-8		G 0
-15	-14		-10	-9	-8	-7		-1
-14	-13		-9	-8	-7	-6		-2
-13	-12		-8	-7	-6	-5	-4	-3
-12	-11	-10	-9	-8		-6	-5	-4
-13	-12	-11	-10	-9	-8	-7	-6	-5

V^*

What's the optimal path from S to G?

Another Maze Learning Task



Now what's the 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

Challenges

- How do we apply these techniques to infinite (e.g., continuous), or even just very large, state spaces?
  - Pole-balancer
  - Mountain car
  - Acrobot
  - Multi-jointed snake
  - Bioreactor

Together with 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.)

Dealing with large numbers of states

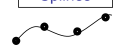
Don't use a Table…

STATE	VALUE
s_1	
s_2	
:	
$s_{15122189}$	

use…

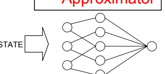
(Generalizers)

Splines



A Function Approximator

STATE



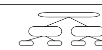
VALUE

(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"

Many 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.)

If we had time…

- Value function approximation
  - Use a Neural Net to represent *V* [e.g. Tesauro]
  - Use a Neural Net to represent *Q* [e.g. Crites]
  - Use a decision tree

    …with Q-learning [Chapman & Kaelbling '91]

    …How to split up continuous space?
    - Significance test on Q values [Chapman & Kaelbling '91]
    - Execution accuracy monitoring [Moore '91]
    - Game Theory [Moore & Atkeson '95] [Al-Ansari & Williams '98]