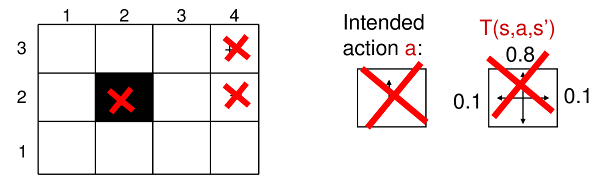
Reinforcement Learning

- R&N Chapter 21

- Demos and Data Contributions from
  Vivek Mehta (vivekm@cs.cmu.edu)
  Rohit Kelkar (ryk@cs.cmu.edu)

Reinforcement Learning



- Same (fully observable) MDP as before except:
  – We don't know the model of the environment
  – We don't know T(.,.,.)
  – We don't know R(.)
- Task is still the same:
  – Find an optimal policy

General Problem

- All we can do is try to execute actions and record the resulting rewards
  – World: You are in state 102, you have a choice of 4 actions
  – Robot: I'll take action 2
  – World: You get a reward of 1 and you are now in state 63, you have a choice of 3 actions
  – Robot: I'll take action 3
  – World: You get a reward of -10 and you are now in state 12, you have a choice of 4 actions
  – …………..

Learning from experience ….

Classes of Techniques

Reinforcement Learning

Model-Based
- Try to learn an explicit model of T(.,.,.) and R(.)

Model-Free
- Recover an optimal policy without ever estimating a model

Model-Based

- If we knew a good estimate $T^{est}(.,.,.)$ of $T(.,.,.)$ and $R(.)$, we could evaluate the optimal policy by solving the fundamental MDP relations:

$$U^{est}(s) = R(s) + \gamma \max_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

$$\pi^*(s) = \operatorname{argmax}_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

Model Estimation

	1	2	3	4
3				+1
2	s_1	■		-1
1	s	s_2		

I observed a trajectory during which, when I moved Up from $s = (1,1)$, I ended up in

$s_1 = (1,2)$ 10 times
$s_2 = (2,1)$ 2 times

$T(s, \text{Up}, s_1) \sim 10/(10+2) = 0.83$
$T(s, \text{Up}, s_2) \sim 2/(10+2) = 0.17$

Model-Based

- Move through the environment by executing a sequence of actions
- Evaluate T and R:
  - $R(s)$ = Reward received when visiting state s
  - $T^{est}(s, a, s')$ ~ (# times we moved from s to s' on action a)/(# times we applied action a from s)
- This gives us an estimated model of the Markov system

Model-Based

- Given T^{est} and R, we can now estimate the value at each state:

$$U^{est}(s)=R(s)+\gamma \max_a\left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

  - Value iteration
  - Policy iteration
- This can be expensive if we do that at each step
- May require matrix inversion (size = number of states) or
- Many iterations of value iteration

A Review: Solving MDPs

- No actions: Value iterations √

- With actions: Value iteration, Policy iteration

Stopping Value Iteration

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

$$J^2(S_i) = r_i + \gamma\left(\sum_k p_{i,k} J^1(s_k)\right)$$

$$J^{t+1}(S_i) = r_i + \gamma\left(\sum_k p_{i,k} J^t(s_k)\right)$$

Remember, we have a converging function

We can stop when $|J^{t-1}(s_i) - J^t(s_i)|_\infty < \varepsilon$

Infinity norm selects maximal element

The Alternative: Policy iteration

- We can also compute optimal policies by revising an existing policy.
- We initially select a policy at random (mapping from states to actions).
- We re-compute the expected long term reward at each state using the selected policy
- We select a new policy using the expected rewards and iterate until convergences

Policy iteration: algorithm

- Let $\pi_t(s_i)$ be the selected policy at time t

1. Randomly choose π_0 ; set t = 0
2. For each state s_i compute $J^*(s_i)$, the long term expected reward using policy π_t .
3. Set $\pi_t(s_i)$ = $\max_k r_i + \gamma\left(\sum_j p^k_{i,j} J^*(s_j)\right)$
4. Convergence? Yes: output policy. No: t = t + 1, go to 2.

Policy iteration: algorithm

- Let $\pi_t(s_i)$ be the selected policy at time t

1. Randomly chose π_0 ; set t = 0
2. For each state s_i compute $J^*(s_i)$, the long term expected reward using policy π_t .
3. Set $\pi_t(s_i)$ = $\max_k r_i + \gamma\left(\sum_j p_{i,j}^k J^*(s_j)\right)$
4. Convergence? Yes: output policy. No: t = t + 1, go to 2.

Can be computed using $J^*(s_i)$ for all states

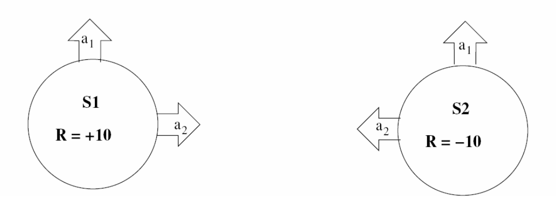
Can be computed using value iteration

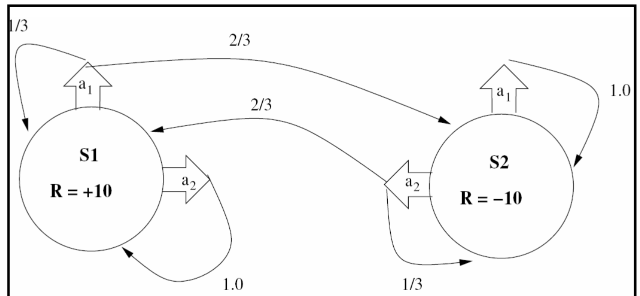
Value iteration vs. policy iteration

- Depending on the model and the information at hand:
  - If you have a good guess regarding the optimal policy then policy iteration would converge much faster
  - similarly, if there are many possible actions, policy iteration might be faster
  - otherwise value iteration is a safer way

Best Policy?

$$
\begin{aligned}
&(\text{Start} = S_1, \text{Action} = a_1, \text{Reward} = 10, \text{End} = S_2)\\
&(\text{Start} = S_2, \text{Action} = a_2, \text{Reward} = \text{-}10, \text{End} = S_1)\\
&(\text{Start} = S_1, \text{Action} = a_2, \text{Reward} = 10, \text{End} = S_1)\\
&(\text{Start} = S_1, \text{Action} = a_1, \text{Reward} = 10, \text{End} = S_1)\\
&(\text{Start} = S_1, \text{Action} = a_2, \text{Reward} = 10, \text{End} = S_1)\\
&(\text{Start} = S_1, \text{Action} = a_1, \text{Reward} = 10, \text{End} = S_2)\\
&(\text{Start} = S_2, \text{Action} = a_1, \text{Reward} = \text{-}10, \text{End} = S_2)\\
&(\text{Start} = S_2, \text{Action} = a_2, \text{Reward} = \text{-}10, \text{End} = S_2)\\
&(\text{Start} = S_2, \text{Action} = a_2, \text{Reward} = \text{-}10, \text{End} = S_1)
\end{aligned}
$$





$\pi^*(S_1) = a_2$
$\pi^*(S_2) = a_2$

Problems

- Separates learning the model from using the model (not on-line learning)
- Expensive because entire set of equations is solved to find U^{est}
- How should the environment be explored? → No guidance until model is built
- Cannot handle changing environments

Solution

- Update U^{est} for some state *s only* instead of solving for all the states

$$U^{est}(s) \leftarrow R(s) + \gamma \max_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

- Similar to one step of value iteration
- Terminology → "Backup step"
- Advantage:
  - Computation interleaved with exploration
  - Less computation at each step

Example: Model-Based Learning

- Update the current estimate of $U(s)$ = expected sum of future discounted reward using estimated $T(.,.,.)$

$$U^{est}(s) \leftarrow R(s) + \gamma \max_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

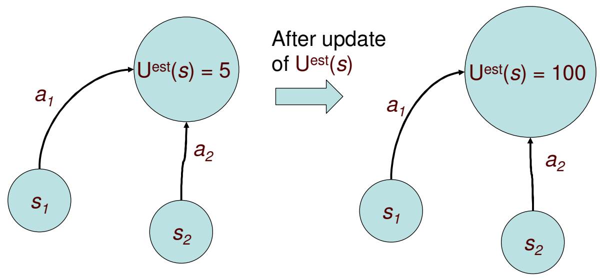
$$\pi(s) = \operatorname{argmax}_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

Two Problems

- Which states to update? U^{est} may have already converged for some states, so that the update does not make any difference
- How to explore the environment? We have not said how we generate the actions *a*

Which State to Update: Prioritized Sweeping

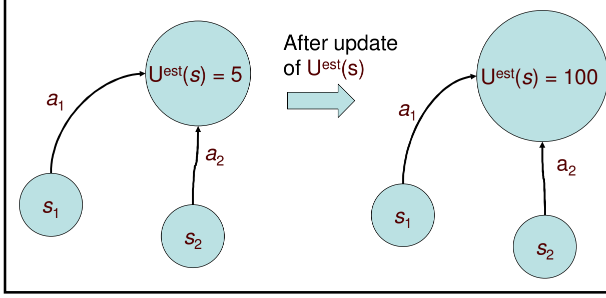
- Idea: Update the predecessors of the states that yield the largest change in U^{est}



$U^{est}(s)$ has changed a lot and

$U^{est}(s_1) = R(s_1) + \ldots. + T(s_1,a_1,s)U^{est}(s)$

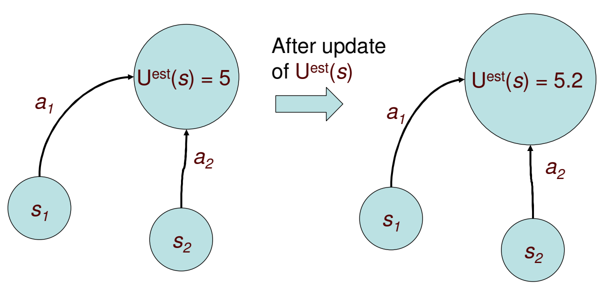
So $U^{est}(s_1)$ is probably going to change a lot too so we should update it right away



$U^{est}(s)$ has not changed a lot and

$U^{est}(s_1) = R(s_1) + \ldots. + T(s_1,a_1,s)U^{est}(s)$

So $U^{est}(s_1)$ is probably not going to change a lot so it's not useful to waste time updating it



Prioritized Sweeping

- For each state: Remember Pred(s) = {visited states s' and action a such that a moves from s' to s}
- Store P = priority queue with "most promising" state first

1. s = top of the queue; U^{old} = current value of $U^{est}(s)$
2. Update the state value

$$U^{est}(s) \leftarrow R(s) + \gamma \max_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

3. $\Delta = |U^{old} - U^{est}(s)|$

5. For every predecessor (s_p,a_p) in Pred(s)
   - Add s_p to P with priority: $T^{est}(s_p,a_p,s)\Delta$

Prioritized Sweeping

Δ small = boring state, no change in the value
Δ large = interesting state, new information is discovered

- … ember Pred(s) = {visited states s' … at a moves from s' to s}
- … eue with "most promising" state …

The value for s_p is likely to change if:
1. The value of its successor s has changed a lot (Δ is large) and
2. Some action is likely to move from s_p to s

1. … op of the queue; …
2. … date the state value …

$U^{est}(s) \leftarrow R(s) + \gamma \max_a$ …

3. $\Delta = |U^{old} - U^{est}(s)|$
5. For every predecess… (s_p, a_p) in Pred(s)
   – Add s_p to P with priority: $T^{est}(s_p, a_p, s)\Delta$

Exploration Strategy

- In principle, we can compute a current estimate of the best policy:

$$\pi^*(s) = \text{argmax}_a \left(\sum_{s'} T^{est}(s,a,s')\, U^{est}(s')\right)$$

- What is then the best strategy for exploration?
  – Greedy: Always use $\pi^*(s)$ when in state s?
  – Random
  – Mixed: Sometimes use the best and sometimes use random

Why Not Obvious?

- N-armed bandit problem:
- We have N slot machines, each can yield \$1 with some probability (different for each machine)
- In what order should we try the machines?
  – Stay with the machine with highest probability so far?
  – Random?
  – Something else?

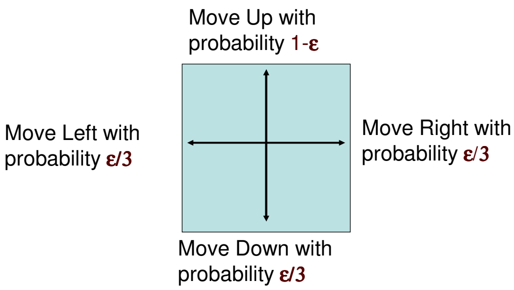
Possible Solutions

- ε-greedy
  – Choose the (current) best one with probability $1-\varepsilon$
  – Choose another one randomly with probability ε/(number of machines – 1)
- Boltzmann exploration
  – Choose machine k with Prob ~ $e^{-Pk/T}$
  – Decrease T as time passes

What does this remind you of?

Maze Example

Current optimal action for this state is Up



Other Solutions?

- ε-greedy
- “Cooling off”
- Ideas: welcome to research! What else should we think through trying?