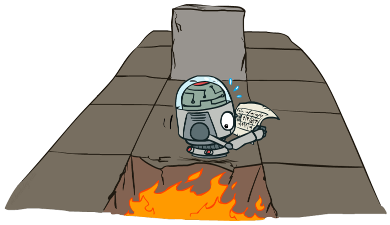
CS 188: Artificial Intelligence

Markov Decision Processes K+1

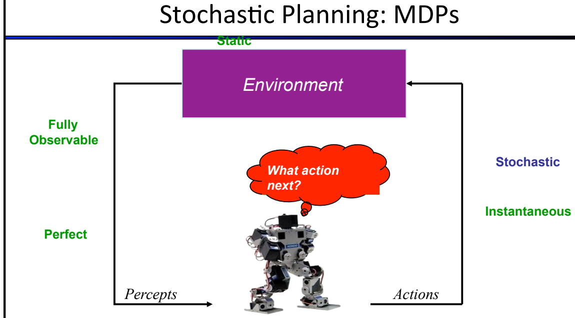


Instructors: Dan Klein and Pieter Abbeel --- University of California, Berkeley

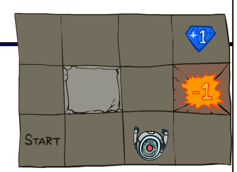
Midterm Review

- Agency: types of agents, types of environments
- Search
  - Formulating a problem in terms of search
  - Algorithms: DFS, BFS, IDS, best-first, uniform-cost, A*, local
  - Heuristics: admissibility, consistency, creation, pattern DBs
  - Constraints: formulation, search, forward checking, arc-consistency, structure
  - Adversarial: min/max, alpha-beta, expectimax
- MDPs
  - Formulation, Bellman eqns, V*, Q*, backups, value iteration, policy iteration

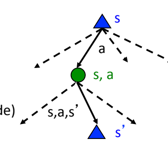
Stochastic Planning: MDPs



Recap: MDPs

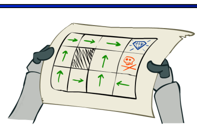


- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0



- Quantities:
  - Policy = map of states to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state (max node)
  - Q-Values = expected future utility from a q-state (chance node)

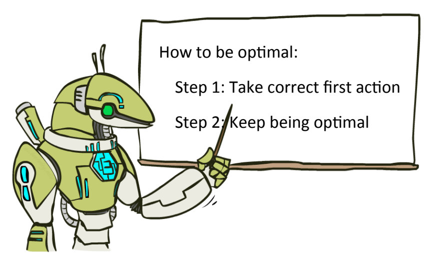
Solving MDPs



- Value Iteration
- Policy Iteration

- Reinforcement Learning

The Bellman Equations



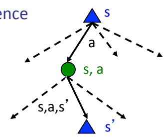
The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

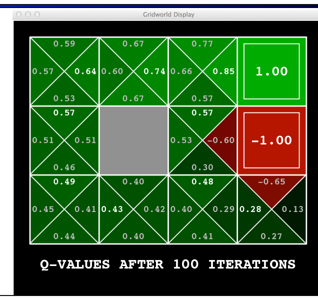
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

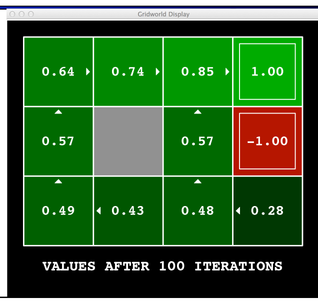


- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

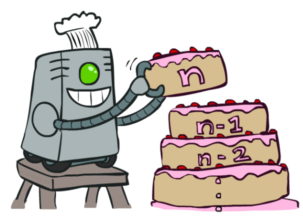
Gridworld: Q*



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



Value Iteration



Value Iteration

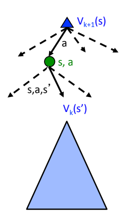
- **Forall s, Initialize $V_0(s) = 0$** *no time steps left means an expected reward of zero*
- **Repeat** *do Bellman backups*

  K += 1

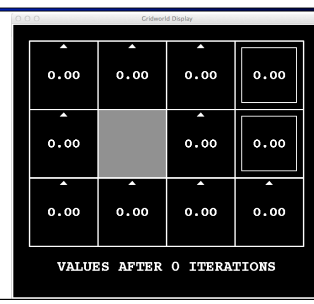
  $Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$

  $V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

- **Repeat until $|V_{k+1}(s) - V_k(s)| < \varepsilon$, forall s** *"convergence"*



k=0

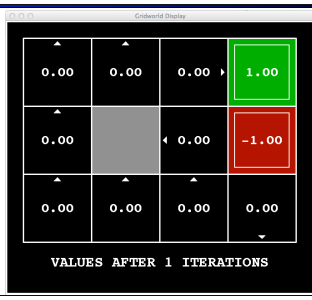


Noise = 0.2
Discount = 0.9
Living reward = 0

k=1

If agent is in 4,3, it only has one legal action: get jewel. It gets a reward and the game is over.
If agent is in the pit, it has only one legal action, die. It gets a penalty and the game is over.

Agent does NOT get a reward for moving INTO 4,3.



Noise = 0.2
Discount = 0.9
Living reward = 0

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

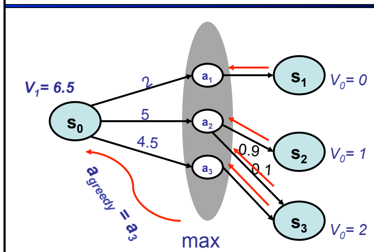
k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

Noise = 0.2
Discount = 0.9
Living reward = 0

Example: Bellman Backup

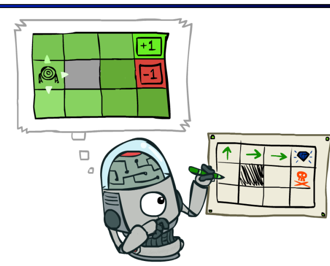


$Q_1(s,a_1) = 2 + \gamma\, 0$
~ 2

$Q_1(s,a_2) = 5 + \gamma\, 0.9 \sim 1$
$+ \gamma\, 0.1 \sim 2$
~ 6.1

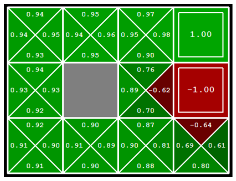
$Q_1(s,a_3) = 4.5 + \gamma\, 2$
~ 6.5

Policy Extraction



Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

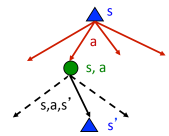
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Important lesson: actions are easier to select from q-values than values!

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

[Demo: value iteration (L9D2)]

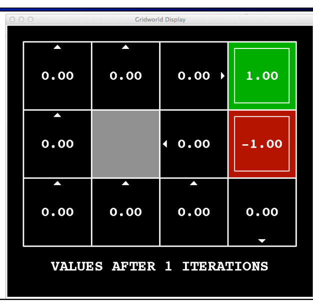
VI → Asynchronous VI

- Is it essential to back up ***all*** states in each iteration?
  - No!
- States may be backed up
  - many times or not at all
  - in any order
- As long as no state gets starved…
  - convergence properties still hold!!

Prioritization of Bellman Backups

- Are all backups equally important?
- Can we avoid some backups?
- Can we schedule the backups more appropriately?

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

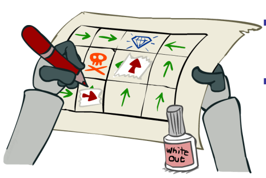
k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

Asynch VI: Prioritized Sweeping

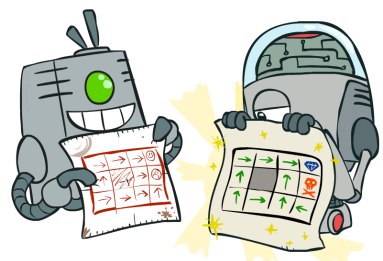
- Why backup a state if values of successors same?
- Prefer backing a state
  - whose successors had most change
- Priority Queue of (state, expected change in value)
- Backup in the order of priority
- After backing a state update priority queue
  - for all predecessors

Solving MDPs

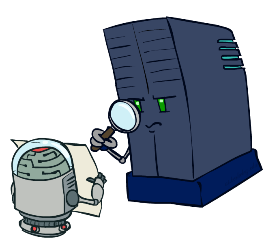


- Value Iteration
- Policy Iteration
- Reinforcement Learning

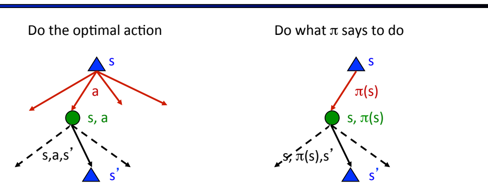
Policy Methods



Policy Evaluation



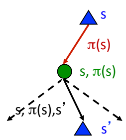
Fixed Policies



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

Utilities for a Fixed Policy

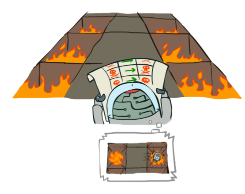
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy
- Define the utility of a state s, under a fixed policy π:
  $V^\pi(s)$ = expected total discounted rewards starting in s and following π
- Recursive relation (one-step look-ahead / Bellman equation):



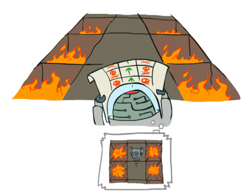
$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

Example: Policy Evaluation

Always Go Right

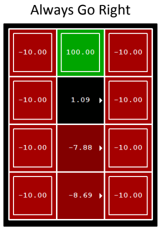


Always Go Forward

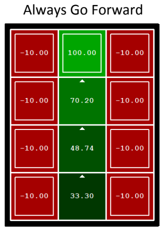


Example: Policy Evaluation

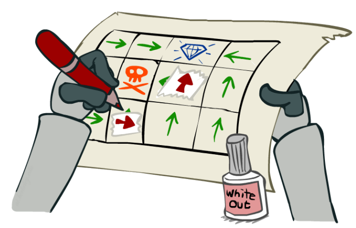
Always Go Right



Always Go Forward



Policy Iteration



Policy Iteration

- Initialize $\pi(s)$ to random actions
- Repeat
  - Step 1: Policy evaluation: calculate utilities of π at each s using a nested loop
  - Step 2: Policy improvement: update policy using one-step look-ahead
    "For each s, what's the best action I could execute, assuming I then follow π?
    Let $\pi'(s)$ = this best action.
    $\pi = \pi'$
- Until policy doesn't change

Policy Iteration Details

- Let i =0
- Initialize $\pi_i(s)$ to random actions
- Repeat
  - Step 1: Policy evaluation:
    - Initialize k=0; Forall s, $V_0^\pi(s) = 0$
    - Repeat until V^π converges
      - For each state s, $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s') \left[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')\right]$
      - Let k += 1
  - Step 2: Policy improvement:
    - For each state, s, $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$
    - If $\pi_i == \pi_{i+1}$ then it's optimal; return it.
    - Else let i += 1

Example

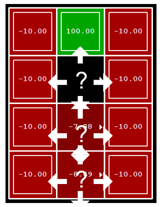
Initialize π_0 to "always go right"

Perform policy evaluation

Perform policy improvement
Iterate through states

Has policy changed?

Yes! i += 1



Example

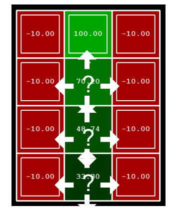
π_1 says "always go up"

Perform policy evaluation

Perform policy improvement
Iterate through states

Has policy changed?

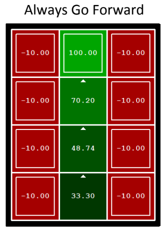
No! We have the optimal policy



Example: Policy Evaluation

Always Go Right

Always Go Forward



Policy Iteration Properties

- Policy iteration finds the optimal policy, guarenteed!
- Often converges (much) faster

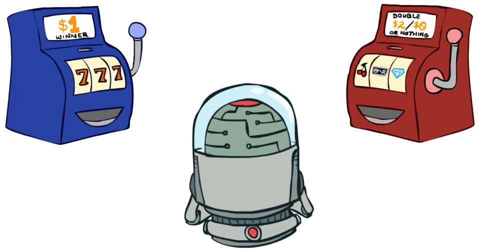
Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Double Bandits



Next Time: Reinforcement Learning!