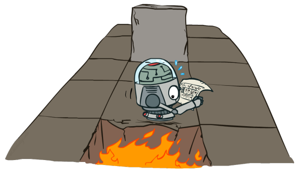
CS 188: Artificial Intelligence

Markov Decision Processes III



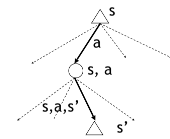
Instructor: Marco Alvarez

University of Rhode Island

1

Recap: MDPs

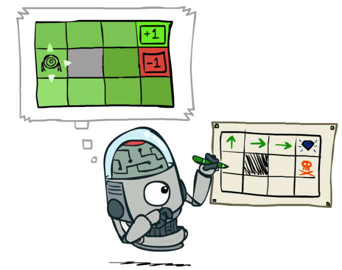
- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0



- 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)

2

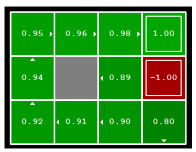
Policy Extraction



3

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)



- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

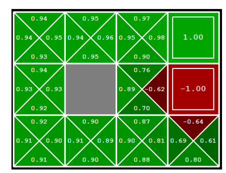
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- This is called policy extraction, since it gets the policy implied by the values

4

Computing Actions from Q-Values

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



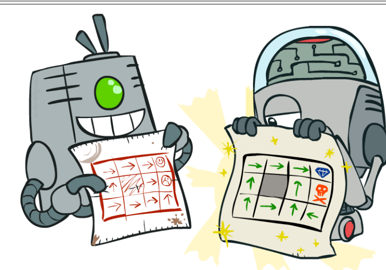
- How should we act?
  - Completely trivial to decide!

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

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

5

Policy Methods

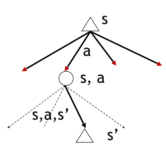


6

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

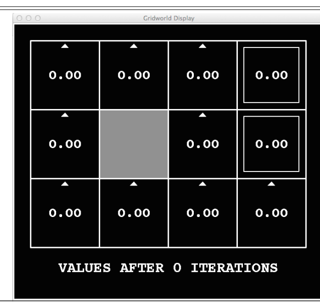


- Problem 1: It's slow – O(S^2A) per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

[Demo: value iteration (L9D2)]

7

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

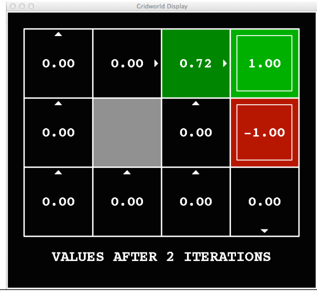
8

k=1

Noise = 0.2
Discount = 0.9
Living reward = 0

9

k=2



Noise = 0.2
Discount = 0.9
Living reward = 0

10

k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

11

k=4

Noise = 0.2
Discount = 0.9
Living reward = 0

12

k=5

Noise = 0.2
Discount = 0.9
Living reward = 0

13

k=6

Noise = 0.2
Discount = 0.9
Living reward = 0

14

k=7

Noise = 0.2
Discount = 0.9
Living reward = 0

15

k=8

Noise = 0.2
Discount = 0.9
Living reward = 0

16

k=9

Noise = 0.2
Discount = 0.9
Living reward = 0

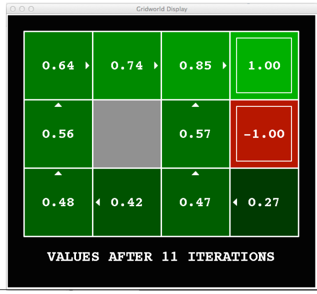
17

k=10

Noise = 0.2
Discount = 0.9
Living reward = 0

18

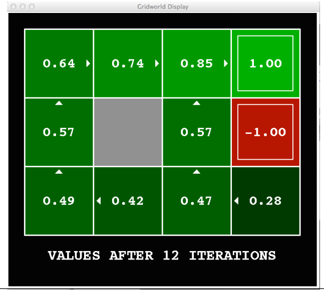
k=11



Noise = 0.2
Discount = 0.9
Living reward = 0

19

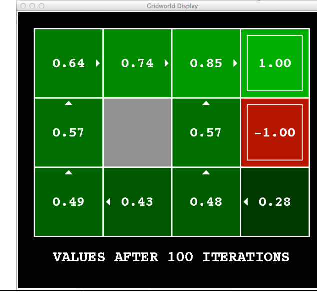
k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

20

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

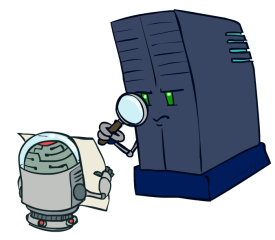
21

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead (**policy extraction**) with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

22

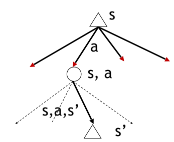
Policy Evaluation



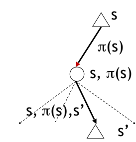
23

Fixed Policies

Do the optimal action



Do what π says to do



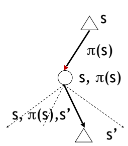
- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler - only one action per state
  - … though the tree's value would depend on which policy we fixed

24

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):

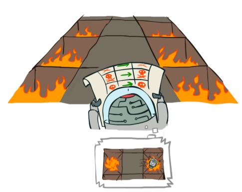
$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



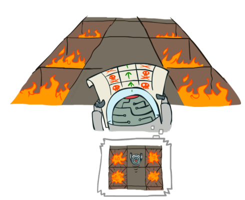
25

Example: Policy Evaluation

Always Go Right



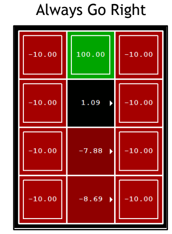
Always Go Forward



26

Example: Policy Evaluation

Always Go Right

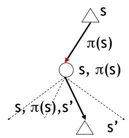


Always Go Forward

27

Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)



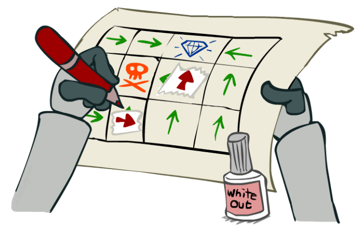
$$V_0^{\pi}(s) = 0$$

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- Efficiency: $O(S^2)$ per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

28

Policy Iteration



29

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

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

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

30

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

31

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

32