CIS 471/571(Fall 2020): Introduction to Artificial Intelligence

Lectur art 2)

Thanh H. Nguyen

Source: http://ai.berkeley.edu/home.html

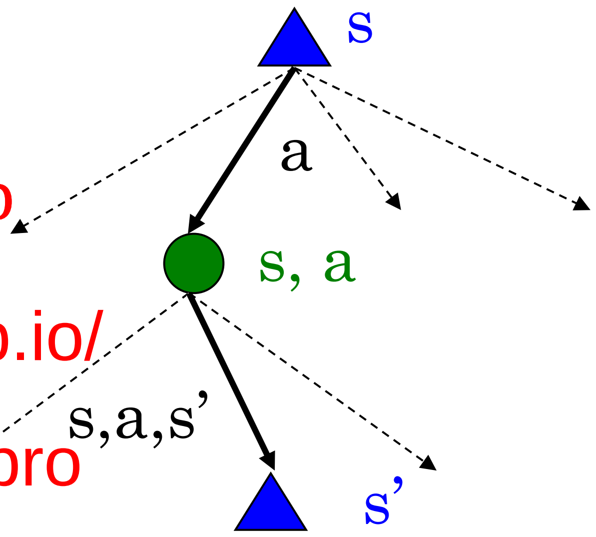
Announcement

- Project 3: Reinforcement Learning
  - Deadline: Nov 10th, 2020

- Homework 3: MDPs and Reinforcement Learning
  - Will be posted tomorrow
  - Deadline: Nov 09th, 2020

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 d
  - 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)

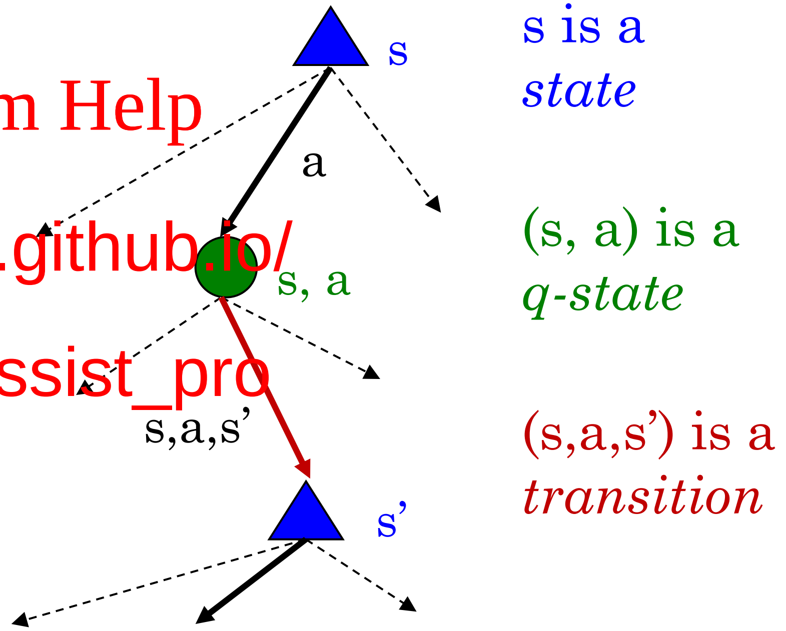
Optimal Quantities

- The value (utility) of a state s:

  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

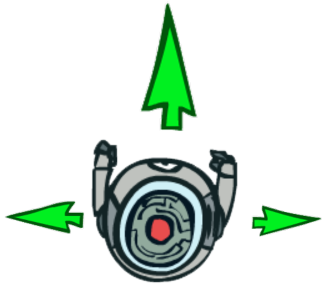


- The optimal policy:

  $\pi^*(s)$ = optimal action from state s

Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not
  - 80% of the time, the action Nor North
  - 10% of the time, North takes the agent West; 10%
  - If there is a wall in the direction the agent would been taken, the agent stays put
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of (discounted) rewards



The Bellman Equations

How to be optimal:

orrect first action

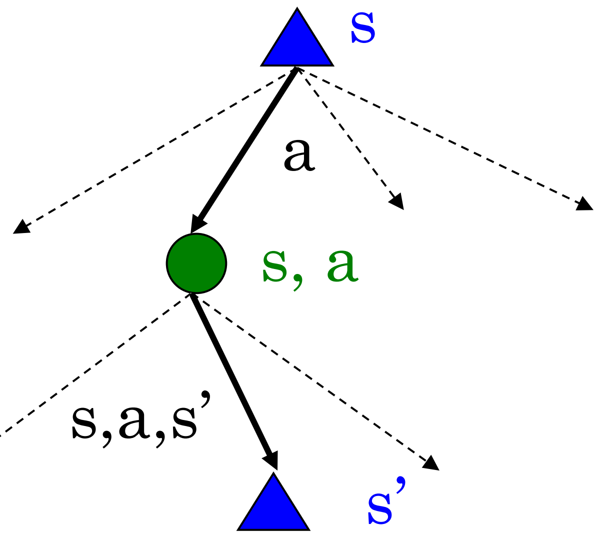
Step 2 ing optimal

The Bellman Equations

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

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

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



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

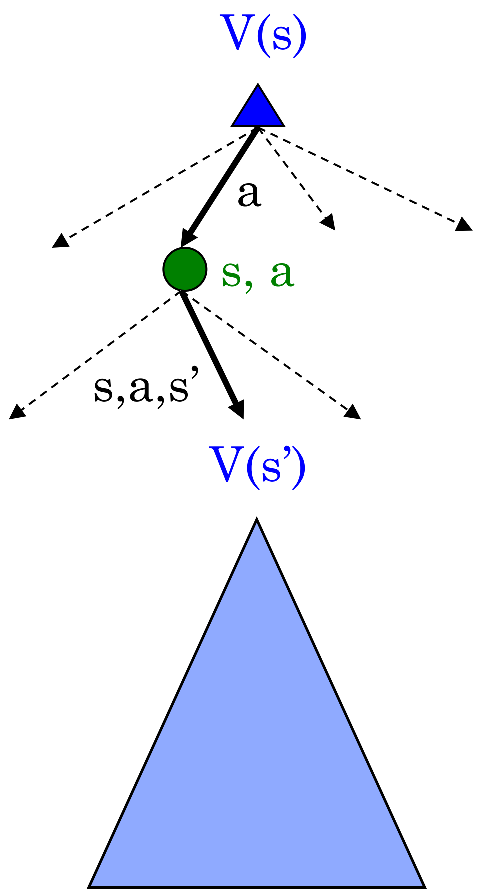
Value Iteration

- Bellman equations characterize the optimal values:

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

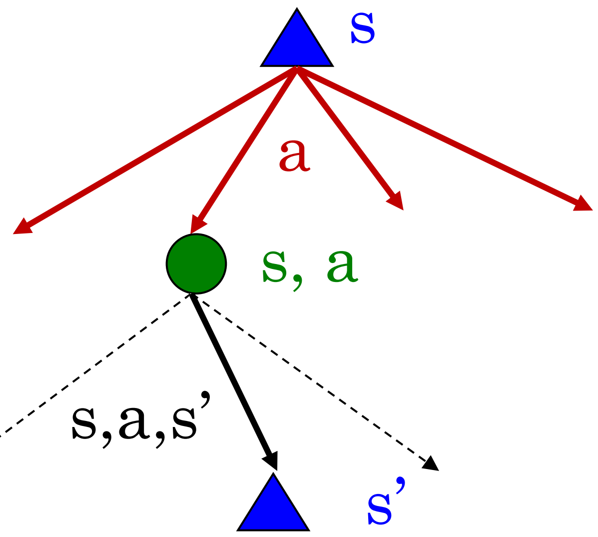
- Value iteration computes th

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do



Problems with Value Iteration

- Value iteration repeats the Bellman updates:



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

k=0

Noise = 0.2
Discount = 0.9
Living reward = 0

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

k=4

Noise = 0.2
Discount = 0.9
Living reward = 0

k=5

Noise = 0.2
Discount = 0.9
Living reward = 0

k=6

Noise = 0.2
Discount = 0.9
Living reward = 0

k=7

Noise = 0.2
Discount = 0.9
Living reward = 0

k=8

Noise = 0.2
Discount = 0.9
Living reward = 0

k=9

Noise = 0.2
Discount = 0.9
Living reward = 0

k=10

Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

Noise = 0.2
Discount = 0.9
Living reward = 0

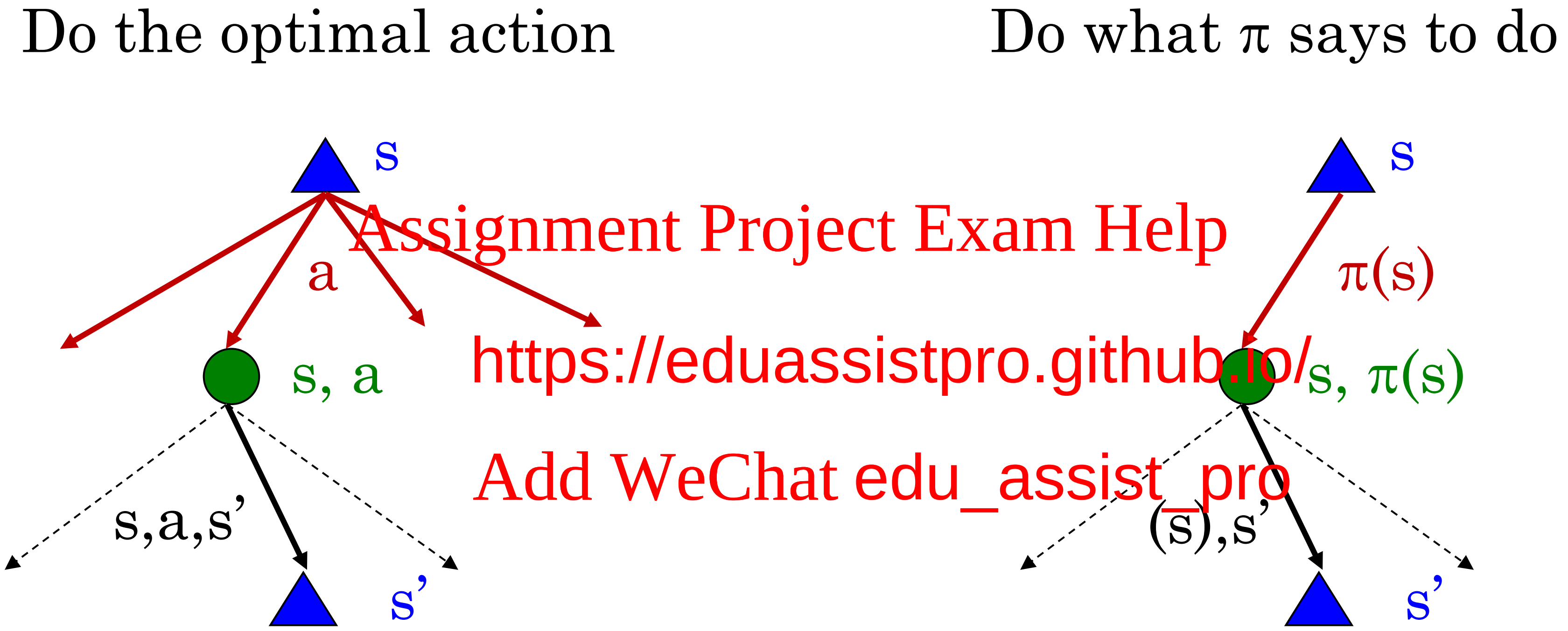
k=100

Noise = 0.2
Discount = 0.9
Living reward = 0

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

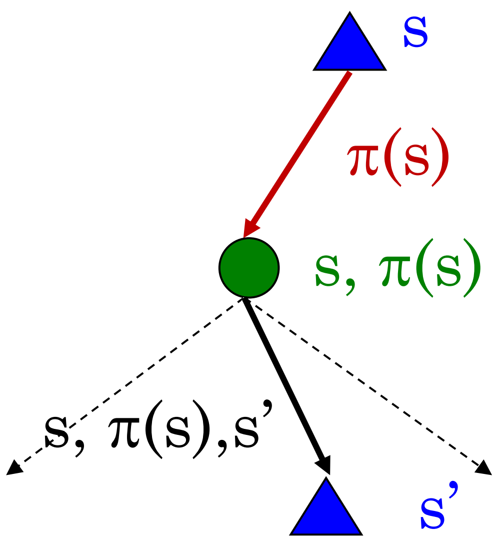
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a stat y π:

  $V^{\pi}(s)$ = expected total discount d following π

- Recursive relation (one-step look-ahead / Bellman equation):

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



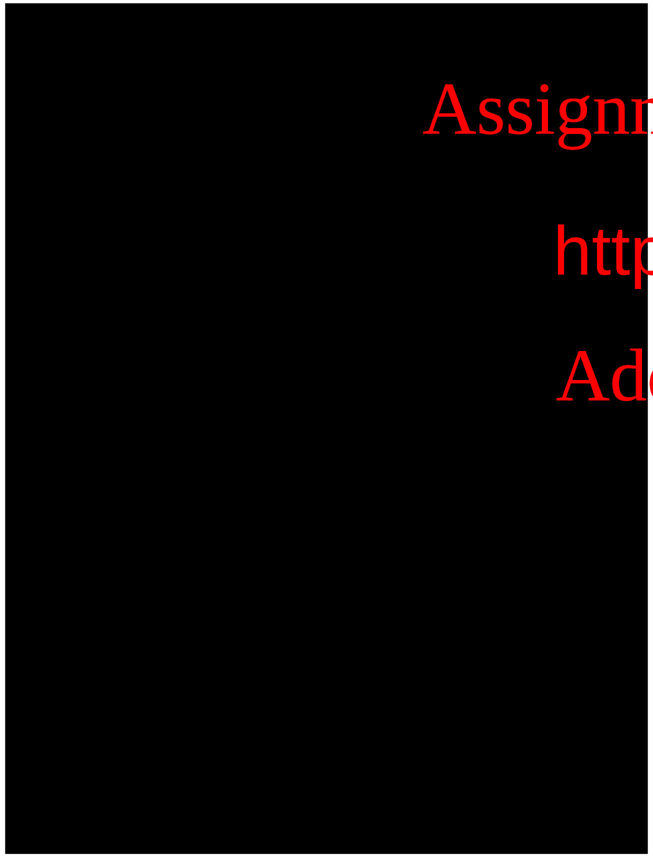
Example: Policy Evaluation

Always Go Right

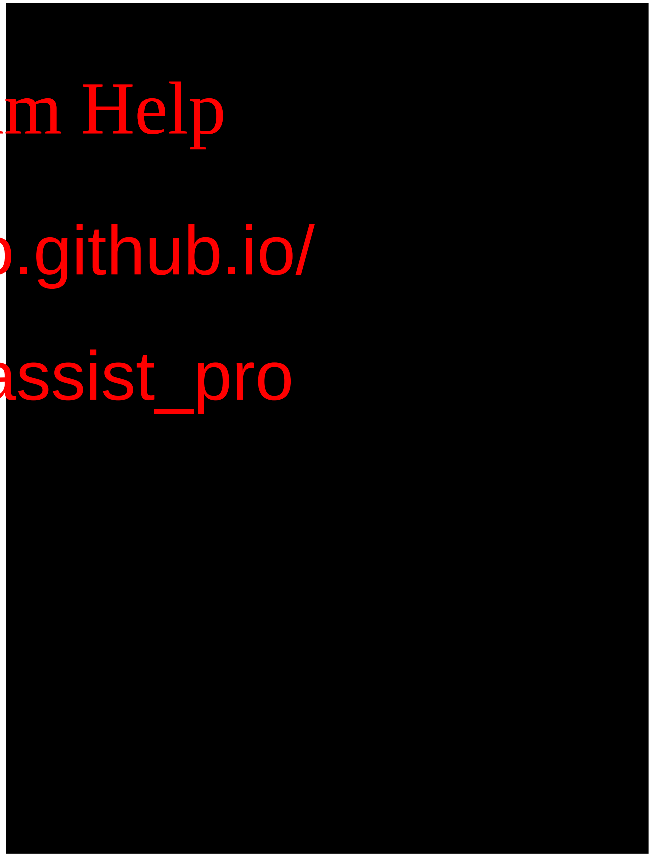
Always Go Forward

Example: Policy Evaluation

Always Go Right



Always Go Forward

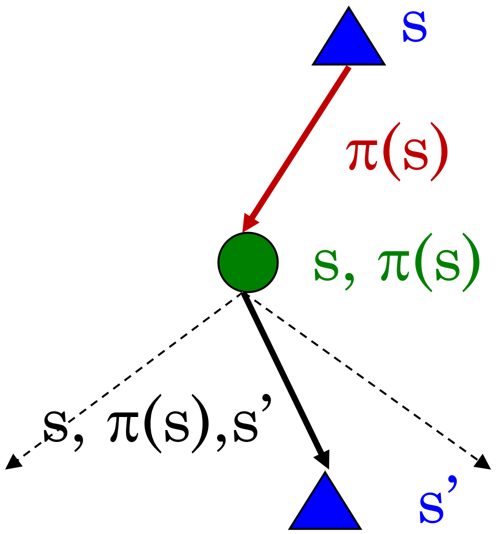


Policy Evaluation

- How do we calculate the V's for a fixed policy π?

- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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



- 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)

Policy Extraction

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 (

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

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

Computing Actions from Q-Values

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

- How should we act?

  - Completely trivial to deci

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

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

Policy Iteration

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 improvem g one-step look-ahead with resulting converged (but

 as future values
  - Repeat steps until polic

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

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

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

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

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

- In value iteration:

  - Every iteration updates bo citly) the policy
  - We don't track the policy, 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 ction (one-step lookahead)

- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step look-ahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Example: Racing

- Discount: $\gamma = 0.1$
- Initial policy
  - $\pi_0(Cool) = Slow$
  - $\pi_0(Warm) = Slow$
  - $\pi_0(Overheated) = \emptyset$