Markov Decision Processes
Value Iteration

Andrea Danyluk
February 24, 2017

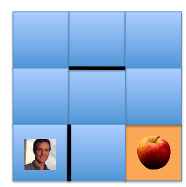
Announcements

- Programming Assignment 2 in progress
- How to find coding partners

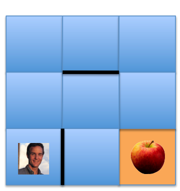
Today's Lecture

- Markov Decision Processes
- Value Iteration

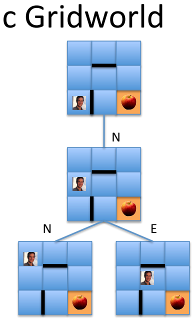
Deterministic Gridworld



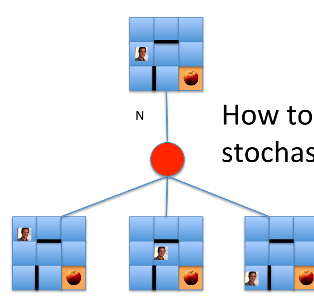
Deterministic Gridworld



N, E, E, S
Or
N, E, S, E

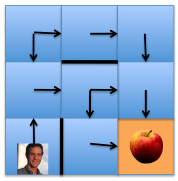


Stochastic Gridworld



How to plan in a stochastic world?

Policies, not Plans



Markov Decision Processes

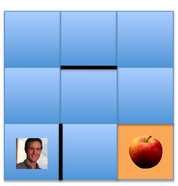
An MDP consists of:

- S: a set of states
- A: a set of actions
- P(s' | s, a): the probability of ending up in sate s', given that the agent is in state s and takes action a
- R(s): the immediate reward at state s
- A designated start state
- [Sometimes] a designated terminal state

"Markov" = given the present state, the future and the past are independent

Gridworld State Rewards

R(s) = +1, if s is "apple state"
-.05 otherwise



If our goal is to maximize the sum of the rewards (or something like that),
negative reward will help us reach our goal as efficiently as possible.

Value of a State

- Value (Utility) of being in a state is not the same as the reward
- First consider the utility of a state history. Can be
  - Additive: $V([s_1,s_2,\cdots s_n])=R(s_1)+R(s_2)+\cdots R(s_n)$
  - Discounted: $V([s_1,s_2,\cdots s_n])=R(s_1)+\gamma R(s_2)+\gamma^2 R(s_3)+\cdots\gamma^n R_{n+1}$
  - Where
    - γ is a discount factor between 0 and 1

Value of a State (cont'd)

- Don't want to restrict ourselves to a finite horizon.
- For an infinite horizon:
  - Additive: $V([s_1,s_2,\cdots])=R(s_1)+R(s_2)+\cdots$
  - Discounted: $V([s_1,s_2,\cdots])=R(s_1)+\gamma R(s_2)+\gamma^2 R(s_3)+\cdots$
  - γ is a discount factor between 0 and 1
- If environment has no terminal state or if agent never reaches one, undiscounted rewards will generally lead to infinite value
  - Discounted rewards result in finite state values

Why infinite horizon?

- Optimal policy for a finite horizon is non-stationary
  - Optimal action from a state can change
- Optimal policy for an infinite horizon is stationary
  - No reason to behave differently in the same state at different times

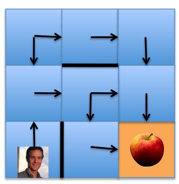
Utility is directly linked to policy

Action Policy

- Deterministic policy: π: S -> A
  - π(*s*) gives the action to take in state *s*
- Probabilistic policy: π: S x A -> [0, 1].
  - π(*s*, *a*) specifies a probability for choosing action *a* in state *s*
- We'll focus on the former for now

Optimal Policies

- Want optimal policy
  - π*: S -> A
- If followed, optimal policy maximizes expected utility (i.e., expected value)



- Find the **expected value** (expected utility) of each state
- Choose the action that maximizes expected value
- Optimal values define optimal policies

Optimal Values (Utilities)

Note slight (but not significant) differences in S&B and R&N formulations

- Define V*(S) to be the expected utility of acting optimally from S.
- Define Q*(S, a) to be the expected utility of taking action a from state S and from there acting optimally.

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

$Q^*(s, a) = \Sigma\, P(s' \mid s,a) \cdot [R(s') + \gamma \cdot V^*(s')]$,
where the sum is over all s'

Bellman Equations

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

$Q^*(s, a) = \Sigma\, P(s' \mid s,a) \cdot [R(s') + \gamma \cdot V^*(s')]$,
where the sum is over all s'

Definition of value (utility) leads to a simple one-step lookahead relationship among optimal utilities

Total optimal reward = optimize over choice of (first action + optimal future)

[Adapted from CS 188 Berkeley]

Computing Optimal Values

- Calculating V*(s) just once won't give you the optimal value
  - Like doing a 1-step lookahead in expectimax
- If we look ahead ∞ steps, then we approach the true optimum, V*(s)
  - But we won't do an expectimax search

Value Iteration

- Will calculate successive estimates V_k^* of V^*
- Start with $V_0^*(s) = 0$ for all s
- Given V_i^*, calculate the values for all states for depth i+1

  $V_{i+1}^*(s) = \max_a \Sigma P(s' | s,a) \cdot [R(s') + \gamma \cdot V_i^*(s')]$
- Throw out old vector V_i^*
- Repeat until convergence
- Called **value update** or **Bellman update**

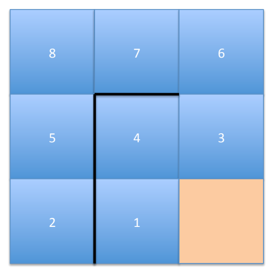
[Adapted from CS 188 Berkeley]

Value Iteration Demos

- All rewards are 1
- The value of a state is either the value itself or the *value + the penalty* if you got there by running into a wall (so in this case we aim to minimize expected "reward")
- PJOG = how badly you go off course
  - 0 means your action does what you intended
  - 0.3 means 70% of the time your action does what's intended; splits the 30% evenly among the remaining options
- Discount rate (gamma) is always 1

Value Iteration: Exercise 1

- Smallest maze
- PJOG = 0
- Demo



Value Iteration: Exercise 2

- Smallest maze
- PJOG = 0.75
  - For any action, have .25 probability of taking any of the four possible actions
- Notice what happens with the policy!
- Demo

Value Iteration: Exercise 3

- Smallest maze
- PJOG = 0.3
  - For any action, have .7 probability of taking that action; .1 probability of taking each of the others
- Demo

Things to notice in the demos

- Value approximations get refined toward optimal values
- Information propagates outward from the terminal states until all states have correct information
- The policy may converge long before the values do