CSE 473  Markov Decision Processes

Dan Weld

Many slides from Chris Bishop, Mausam, Dan Klein, Stuart Russell, Andrew Moore & Luke Zettlemoyer

Logistics

- PS 2 due Thursday → Thursday 10/18
- PS 3 due Thursday 10/25

MDPs

Markov Decision Processes

- Planning Under Uncertainty
- Mathematical Framework
- Bellman Equations
- Value Iteration
- Real-Time Dynamic Programming
- Policy Iteration
- Reinforcement Learning

Andrey Markov (1856–1922)

Planning Agent

Environment

- Fully vs. Partially Observable
- Perfect vs. Noisy
- Deterministic vs. Stochastic
- Instantaneous vs. Durative

Objective of an MDP

- Find a policy \( \pi : V \rightarrow D \)
- which optimizes
  - minimizes \( \text{discounted} \) expected cost to reach a goal
  - maximizes \( \text{undiscounted} \) expected reward
  - maximizes expected (reward-cost)
- given a _____ horizon
  - finite
  - infinite
  - indefinite

Review: Expectimax

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In pacman, the ghosts act randomly
- Can do expectimax search
  - Max nodes as in minimax search
  - Chance nodes, like \( \min \) nodes, except the outcome is uncertain - take average (expectation) of children
  - Calculate expected utilities
- Today, we formalize as a Markov Decision Process
  - Handle \textit{intermediate rewards} & \textit{infinite plans}
  - More efficient processing
Grid World

- Walls block the agent’s path
- Agent’s actions may go astray:
  - 80% of the time, North action takes the agent North (assuming no wall)
  - 10% - actually go West
  - 10% - actually go East
- If there is a wall in the chosen direction, the agent stays put
- Small “living” reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards

Markov Decision Processes

- An MDP is defined by:
  - A set of states \( s \in S \)
  - A set of actions \( a \in A \)
  - A transition function \( T(s,a,s’) \)
    - Prob that \( a \) from \( s \) leads to \( s’ \)
    - i.e., \( P(s’|s,a) \)
  - Also called “the model”
  - A reward function \( R(s, a, s’) \)
  - Sometimes just \( R(s) \) or \( R(s’) \)
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs: non-deterministic search
  - Reinforcement learning: MDPs where we don’t know the transition or reward functions

What is Markov about MDPs?

- Andrey Markov (1856-1922)
- “Markov” generally means that
  - conditioned on the present state,
  - the future is independent of the past
- For Markov decision processes, “Markov” means:
  \[
P(S_{t+1} = s’|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)
  =
  P(S_{t+1} = s’|S_t = s_t, A_t = a_t)
\]

Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy \( \pi^*: S \rightarrow A \)
  - A policy \( \pi \) gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

Example Optimal Policies

- Optimal policy when \( R(s, a, s’) = -0.03 \)
  for all non-terminals \( s \)

Example Optimal Policies

- \( R(s) = -0.01 \)
- \( R(s) = -0.03 \)
Example: High-Low

- Three card types: 2, 3, 4
  - Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say "high" or "low"
- New card is flipped
  - If you're right, you win the points shown on the new card
  - Ties are no-ops (no reward) 0
  - If you're wrong, game ends

- Differences from expectimax problems:
  - #1: get rewards as you go
  - #2: you might play forever!

High-Low as an MDP

- States:
  - 2, 3, 4, done
- Actions:
  - High, Low
- Model: $T(s, a, s')$:
  - $P(s' = 4 \mid 4, Low) = \frac{1}{4}$
  - $P(s' = 3 \mid 4, Low) = \frac{1}{4}$
  - $P(s' = 2 \mid 4, Low) = \frac{1}{2}$
  - $P(s' = \text{done} \mid 4, Low) = 0$
  - $P(s' = 4 \mid 4, High) = \frac{1}{4}$
  - $P(s' = 3 \mid 4, High) = 0$
  - $P(s' = 2 \mid 4, High) = 0$
  - $P(s' = \text{done} \mid 4, High) = \frac{3}{4}$
- Rewards: $R(s, a, s')$:
  - Number shown on $s'$ if $s' < s \land a = \text{"high"}$ ...
  - 0 otherwise
- Start: 3

Search Tree: High-Low

MDP Search Trees

- Each MDP state gives an expectimax-like search tree
Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:
  \[ [r, r_2, r_3, r_4, \ldots] \succ [r_0, r_1, r_2, r_3, \ldots] \]
  \[ [r_0, r_1, r_2, r_3, \ldots] \succ [r_0, r_1, r_2, r_3, \ldots] \]
- Theorem: only two ways to define stationary utilities
  - Additive utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots \]
  - Discounted utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \]

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards
- Solutions:
  - Finite horizon: terminate episodes after a fixed T steps (e.g., life)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)
  - Discounting: for \( 0 < \gamma < 1 \)
    \[ U([r_0, r_1, r_2, \ldots]) = \sum_{i=0}^{\infty} \gamma^i r_i \leq R_{\text{max}}/(1 - \gamma) \]
    - Smaller \( \gamma \) means smaller “horizon” = shorter term focus

Discounting

\[ U([r_0, \ldots, r_n]) = \sum_{i=0}^{n} \gamma^i r_i \leq R_{\text{max}}/(1 - \gamma) \]

- Typically discount rewards by \( \gamma < 1 \) each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge

Recap: Defining MDPs

- Markov decision processes:
  - States \( S \)
  - Start state \( s_0 \)
  - Actions \( A \)
  - Transitions \( P(s' | s, a) \) aka \( T(s, a, s') \)
  - Rewards \( R(s, a, s') \) (and discount \( \gamma \))
- MDP quantities so far:
  - Policy, \( \pi = \text{Function that chooses an action for each state} \)
  - Utility (aka “return”) = sum of discounted rewards

Optimal Utilities

- Define the value of a state \( s \):
  \[ V^*(s) = \text{expected utility starting in } s \text{ and acting optimally} \]
- Define the value of a \( q \)-state \( (s, a) \):
  \[ Q^*(s, a) = \text{expected utility starting in } s, \text{ taking action } a \text{ and thereafter acting optimally} \]
- Define the optimal policy:
  \[ \pi^*(s) = \text{optimal action from state } s \]

Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)
- Idea: Value iteration
  - Compute optimal values for all states all at once using successive approximations
  - Will be a bottom-up dynamic program similar in cost to memoization
  - Do all planning offline, no replanning needed!
The Bellman Equations

- Definition of "optimal utility" leads to a simple one-step look-ahead relationship between optimal utility values:

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

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

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

Bellman Equations for MDPs

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

Bellman Backup (MDP)

- Given an estimate of \( V^* \) function (say \( V_n \))
- Backup \( V_n \) function at state \( s \)
  - calculate a new estimate \( V_{n+1} \):

  \[ Q_{n+1}(s, a) = \sum_{s' \in S} P_r(s', s, a) \left[ R(s', s, a, s') + \gamma V_n(s') \right] \]

  \[ V_{n+1}(s) = \max_a \left[ Q_{n+1}(s, a) \right] \]

- \( Q_{n+1}(s, a) \) : value/cost of the strategy:
  - execute action \( a \) in \( s \), execute \( \pi_n \) subsequently
  - \( \pi_n = \arg\max_{a \in A(s)} Q_n(s, a) \)

Value iteration [Bellman’57]

- assign an arbitrary assignment of \( V_0 \) to each state.
- repeat
  - for all states \( s \)
    - compute \( V_{n+1}(s) \) by Bellman backup at \( s \)
  - until \( \max_s \left| V_{n+1}(s) - V_n(s) \right| < \varepsilon \)

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Bellman Backup

- Given \( V_i \), calculate the values for all states for depth \( i+1 \):

\[ Q_{i+1}(s, a) = \sum_{s' \in S} P_r(s', s, a) \left[ R(s', s, a, s') + \gamma V_i(s') \right] \]

\[ V_{i+1}(s) = \max_a \left[ Q_{i+1}(s, a) \right] \]

Value Iteration

- Idea:
  - Start with \( V_0(s) = 0 \), which we know is right (why?)
  - Given \( V_i \), calculate the values for all states for depth \( i+1 \):

\[ V_{i+1}(s) = \max_a \sum_{s' \in S} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right] \]

- This is called a value update or Bellman update
- Repeat until convergence

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do
Value Estimates

- Calculate estimates $V_k(s)$
  - The optimal value considering only next $k$ time steps (k rewards)
  - As $k \to \infty$, $V_k$ approaches the optimal value

- Why:
  - If discounting, distant rewards become negligible
  - If terminal states reachable from everywhere, fraction of episodes not ending becomes negligible
  - Otherwise, can get infinite expected utility and then this approach actually won’t work

Example: Bellman Updates

Example: Value Iteration

Example: Value Iteration

Practice: Computing Actions

- Which action should we chose from state $s$:
  - Given optimal values $Q$?
    $$\arg \max_a Q^*(s, a)$$
  - Given optimal values $V$?
    $$\arg \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$
  - Lesson: actions are easier to select from $Q$’s!

Comments

- Decision-theoretic Algorithm
- Dynamic Programming
- Fixed Point Computation
- Probabilistic version of Bellman-Ford Algorithm
  - for shortest path computation
    - $\text{MDP}_1$: Stochastic Shortest Path Problem
  - Time Complexity
    - one iteration: $O(|V|^2|D|)$
    - number of iterations: $\text{poly}(|V|, |D|, 1/(1-\gamma))$
  - Space Complexity: $O(|V|)$
  - Factored MDPs = Planning under uncertainty
    - exponential space, exponential time
Convergence Properties

- $V_n \rightarrow V^*$ in the limit as $n \rightarrow \infty$
- $\varepsilon$-convergence: $V_n$ function is within $\varepsilon$ of $V^*$
- Optimality: current policy is within $2\gamma/(1-\gamma)$ of optimal

- Monotonicity
  - $V_n \leq V \Rightarrow V_n \leq V^*$ ($V_n$ monotonic from below)
  - $V_n \geq V \Rightarrow V_n \geq V^*$ ($V_n$ monotonic from above)
  - otherwise $V_n$ non-monotonic

Convergence

- Define the max-norm: $||U|| = \max_x |U(x)|$

- Theorem: For any two approximations $U^t$ and $V^t$
  - $||U^{t+1} - V^{t+1}|| \leq \gamma ||U^t - V^t||$
    - i.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true $V^*$ (aka $U$) and value iteration converges to a unique, stable, optimal solution
  - Theorem:
    - $||U^{t+1} - U^t|| < \epsilon \Rightarrow ||U^{t+1} - U|| < 2\epsilon/(1-\gamma)$
    - i.e. once the change in our approximation is small, it must also be close to correct

Value Iteration Complexity

- Problem size:
  - $|A|$ actions and $|S|$ states

- Each Iteration
  - Computation: $O(|A| \cdot |S|^2)$
  - Space: $O(|S|)$

- Num of iterations
  - Can be exponential in the discount factor $\gamma$

MDPs

Markov Decision Processes
- Planning Under Uncertainty
- Mathematical Framework
- Bellman Equations
- Value Iteration
- Real-Time Dynamic Programming
- Policy Iteration
- Reinforcement Learning

Asynchronous Value Iteration

- States may be backed up in any order
  - Instead of systematically, iteration by iteration

- Theorem:
  - As long as every state is backed up infinitely often...
  - Asynchronous value iteration converges to optimal

Asynchronous Value Iteration

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
Asynchronous Value Iteration
Real Time Dynamic Programming
[Barto, Bradtke, Singh'95]

- Trial: simulate greedy policy starting from start state; perform Bellman backup on visited states

- RTDP:
  - Repeat Trials until value function converges

Why?
- Why is next slide saying min

Comments
- Properties
  - if all states are visited infinitely often then $V_n \to V^*$
- Advantages
  - Anytime: more probable states explored quickly
- Disadvantages
  - complete convergence can be slow!

Labeled RTDP
[Bonet&Gelfner ICAPS03]
- Stochastic Shortest Path Problems
  - Policy w/ min expected cost to reach goal
- Initialize $V^\beta(s)$ with admissible heuristic
  - Underestimates remaining cost
- Theorem:
  - if residual of $V^\beta(s) < \varepsilon$ and $V^\beta(s') < \varepsilon$ for all succ(s), s', in greedy graph
  - Then $V^\beta$ is $\varepsilon$-consistent and will remain so
- Labeling algorithm detects convergence

MDPs
Markov Decision Processes
- Planning Under Uncertainty
- Mathematical Framework
- Bellman Equations
- Value Iteration
- Real-Time Dynamic Programming
- Policy Iteration
- Reinforcement Learning

Andrey Markov (1856-1922)
Changing the Search Space

• Value Iteration
  • Search in value space
  • Compute the resulting policy

• Policy Iteration
  • Search in policy space
  • Compute the resulting value

Utilities for Fixed Policies

• Another basic operation: compute the utility of a state \( s \) under a fixed (general non-optimal) policy
• Define the utility of a state \( s \), under a fixed policy \( \pi \):
  \[ V^\pi(s) = \text{expected total discounted rewards (return) starting in } s \text{ and following } \pi \]
• Recursive relation (one-step look-ahead / Bellman equation):
  \[ V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')] \]

Policy Evaluation

• How do we calculate the \( V \)'s for a fixed policy?
• Idea one: modify Bellman updates
  \[
  V^\pi_0(s) = 0 \\
  V^\pi_{n+1}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi_n(s')] 
  \]
• Idea two: it's just a linear system, solve with Matlab (or whatever)

Policy Iteration

• Problem with value iteration:
  • Considering all actions each iteration is slow: takes \(|A|\) times longer than policy evaluation
  • But policy doesn’t change each iteration, time wasted
• Alternative to value iteration:
  • Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence (fast)
  • Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  • Repeat steps until policy converges

Policy Iteration

• Policy evaluation: with fixed current policy \( \pi \), find values with simplified Bellman updates:
  • Iterate until values converge
  \[
  V^\pi_{n+1}(s) = \sum_{s'} T(s, \pi_k(s), s') [R(s, \pi_k(s), s') + \gamma V^\pi_n(s')] 
  \]
• Policy improvement: with fixed utilities, find the best action according to one-step look-ahead
  \[
  \pi_{k+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi_k(s')] 
  \]

• Policy Iteration: assign an arbitrary assignment of \( \pi_0 \) to each state.
  • repeat
    • Policy Evaluation: compute \( V_{n+1} \): the evaluation of
    • Policy Improvement: for all states \( s \)
      • compute \( \pi_{n+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi_n(s')] \)
      • until \( \pi_{n+1} = \pi_n \)

Advantage

• searching in a finite (policy) space as opposed to uncountably infinite (value) space ⇒ convergence faster.
• all other properties follow!
### Modified Policy Iteration
- assign an arbitrary assignment of $\pi_0$ to each state.

- repeat
  - Policy Evaluation: compute $V_{n+1}$ the approx. evaluation of $\pi_n$
  - Policy Improvement: for all states $s$
    - compute $\pi_{n+1}(s)$: $\arg\max_a \{Q_{n+1}(s,a)\}$
- until $\pi_{n+1} = \pi_n$

**Advantage**
- probably the most competitive synchronous dynamic programming algorithm.

### Policy Iteration Complexity
- **Problem size:**
  - $|A|$ actions and $|S|$ states
- **Each Iteration**
  - Computation: $O(|S|^3 + |A| \cdot |S|^2)$
  - Space: $O(|S|)$
- **Num of iterations**
  - Unknown, but can be faster in practice
  - Convergence is guaranteed

### Comparison
- **In value iteration:**
  - Every pass (or “backup”) updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)
- **In policy iteration:**
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies
- **Hybrid approaches (asynchronous policy iteration):**
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often

### 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 $\gamma$)
  - Start state $s_0$
- **Quantities:**
  - Returns = sum of discounted rewards
  - Values = expected future returns from a state (optimal, or for a fixed policy)
  - Q-Values = expected future returns from a q-state (optimal, or for a fixed policy)