Markov Decision Processes (MDPs)

Ron Parr
CSCI 2951-F
Brown University

With thanks to Kris Hauser for some slides

The MDP Framework

- State space: S
- Action space: A
- Transition function: P
- Reward function: R(s,a,s') or R(s,a) or R(s)
- Policy: $\pi(s) \to a$
- Discount factor: γ

Objective: *Maximize expected, discounted return (sum of rewards)*

Applications of MDPs

- AI/Computer Science
  - Robotic control
    (Koenig & Simmons, Thrun et al., Kaelbling et al.)
  - Air Campaign Planning (Meuleau et al.)
  - Elevator Control (Barto & Crites)
  - Computation Scheduling (Zilberstein et al.)
  - Control and Automation (Moore et al.)
  - Spoken dialogue management (Singh et al.)
  - Cellular channel allocation (Singh & Bertsekas)

Applications of MDPs

- Economics/Operations Research
  - Fleet maintenance (Howard, Rust)
  - Road maintenance (Golabi et al.)
  - Packet Retransmission (Feinberg et al.)
  - Nuclear plant management (Rothwell & Rust)
  - Debt collection strategies (Abe et al.)
  - Data center management (DeepMind)

Applications of MDPs

- EE/Control
  - Missile defense (Bertsekas et al.)
  - Inventory management (Van Roy et al.)
  - Football play selection (Patek & Bertsekas)
- Agriculture
  - Herd management (Kristensen, Toft)
- Other
  - Sports strategies
  - Board games
  - Video games

The Markov Assumption

- Let S_t be a random variable for the state at time t
- $P(S_t|A_{t-1}S_{t-1},\ldots,A_0S_0) = P(S_t|A_{t-1}S_{t-1})$
- Markov is special kind of *conditional independence*
- Future is independent of past given current state, ***action*** (similar to HMM assumptions, but adds actions)

About Rewards

- R(s,a,s') is most general – typically interpreted to associate rewards with transitions. Reward is accrued in state s.
- R(s,a) – Any R(s,a,s') model can be converted to this w/o changing the optimal policy (because of linearity of expectation)
- R(s) – Simplest to write and work with. In general, cannot convert from R(s,a) w/o changing the optimal policy.

- Can always convert from less complicated reward models to more complicated (upwards in this list) w/o consequences

Understanding Discounting: $0 \leq \gamma \leq 1$

- Mathematical motivation
  - Keeps values bounded
  - What if I promise you $0.01 every day you visit me?

- Economic motivation
  - Discount comes from inflation
  - Promise of $1.00 in future is worth $0.99 today

- Probability of dying (losing the game)
  - Suppose ε probability of dying at each decision interval
  - Transition w/prob ε to state with value 0
  - Equivalent to 1- ε discount factor

Discounting in Practice

- Often chosen unrealistically low
  - Faster convergence of the algorithms we'll see later
  - Leads to slightly myopic policies

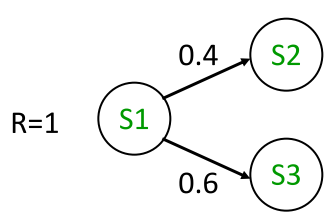
- Can reformulate most algs. for avg. reward
  - Mathematically uglier
  - Somewhat slower run time

Value Determination

Determine the value of each state under policy π

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' | s, \pi(s)) V^{\pi}(s')$$

Bellman Equation for a fixed policy π



$$V^{\pi}(s_1) = 1 + \gamma(0.4 V^{\pi}(s_2) + 0.6 V^{\pi}(s_3))$$

Matrix Form

$$\mathbf{P}^{\pi} = \begin{pmatrix} P(s_1 \mid s_1, \pi(s_1)) & P(s_2 \mid s_1, \pi(s_1)) & P(s_3 \mid s_1, \pi(s_1)) \\ P(s_1 \mid s_2, \pi(s_2)) & P(s_2 \mid s_2, \pi(s_2)) & P(s_3 \mid s_2, \pi(s_2)) \\ P(s_1 \mid s_3, \pi(s_3)) & P(s_2 \mid s_3, \pi(s_3)) & P(s_3 \mid s_3, \pi(s_3)) \end{pmatrix}$$

$$\mathbf{V}^{\pi} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi} + \mathbf{R}^{\pi}$$

Generalization of the game show example from earlier

How to solve this system efficiently? Does it even have a solution?

Solving for Values

$$\mathbf{V}^{\pi} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi} + \mathbf{R}^{\pi}$$

For moderate numbers of states we can solve this system exacty:

$$\mathbf{V}^{\pi} = \underbrace{(\mathbf{I} - \gamma \mathbf{P}^{\pi})}^{-1} \mathbf{R}^{\pi}$$

Guaranteed invertible because $\gamma \mathbf{P}^{\pi}$ has spectral radius <1 when γ<1

Iteratively Solving for Values

$$\mathbf{V}^{\pi} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi} + \mathbf{R}^{\pi}$$

For larger numbers of states we can solve this system *indirectly*:

$$\mathbf{V}^{\pi}{}_{i+1} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi}{}_{i} + \mathbf{R}^{\pi}$$

Guaranteed convergent because $\gamma \mathbf{P}_{\pi}$ has spectral radius <1

Converges to V^{π}, which we call a fixed point because updates
Don't change the value any more

When to stop an iterative solver?

- Just pick some big number of iterations
- Use convergence rates to bound minimum number of iterations required to get within some range of true value function
- Dynamically decide when to stop when change in value function is small from one iteration to the next (also can be guided by theory)

Interpreting the Iterations

- Suppose $V^{\pi}_0 = 0$, and R is defined on (s,a)
- Then $V^{\pi}_1 = R^{\pi}$ (value of executing 1 step of π)
- $V^{\pi}_2 = R^{\pi}+\gamma P^{\pi}V^{\pi}_1 = R^{\pi}+\gamma P^{\pi}R^{\pi}$
  (expected value of executing 2 steps of π)
- $V^{\pi}_3 = R^{\pi}+\gamma P^{\pi}V^{\pi}_2 = R^{\pi}+\gamma P^{\pi}R^{\pi} + \gamma^2(P^{\pi})^2R^{\pi}$
  (expected value of executing 3 steps of π)
- Can interpret these as the value of a finite horizon problem, where everything stops after i steps

Interpretation Continued

- $V^{\pi}_{\infty} = (I-\gamma P^{\pi})^{-1}R = V^{\pi}$=infinite horizon values
- Infinite horizon = value of running π forever
- Nota bene: This interpretation applies when $V^{\pi}_0 = 0$, but iteration converges to V^{π} for any choice of V^{π}_0

Notation Alert

- Policy (π) is sometimes used as a subscript rather than superscript by some authors
- π may be dropped if there (should be) no confusion about which policy is under evaluation
- Some authors (e.g., textbook) use T(s,a,s') for P(s'|s,a)
- Most CS authors use V for the value function
  - textbook uses U
  - Some from operations research use J or other letters
  - Some formulate in terms of minimizing cost (replace max w/min)

Establishing Convergence

- Eigenvalue analysis
- Monotonicity
  - Assume all values start pessimistic
  - One value must always increase
  - Can never overestimate
  - Easy to prove
- Contraction analysis…

Contraction Analysis

- Define maximum norm

$$\|V\|_\infty = \max_i |V[i]|$$

- Consider two value functions V^a and V^b each at iteration 1:

$$\left\|V_1^{\,a} - V_1^{\,b}\right\|_\infty = \varepsilon$$

- WLOG say

$$V_1^{\,a} \le V_1^{\,b} + \vec{\varepsilon}$$ (Vector of all ε's)

Contraction Analysis Contd.

- At next iteration for V^b:

$$V_2^b = R + \gamma P V_1^b$$

- For V^a

$$V_2^a = R + \gamma P(V_1^a) \le R + \gamma \underbrace{P(V_1^b + \vec{\varepsilon})}_{\text{Distribute}} = R + \gamma P V_1^b + \gamma P\vec{\varepsilon} = R + \gamma P V_1^b + \gamma\vec{\varepsilon}$$

- Conclude:

$$\left\|V_2^a - V_2^b\right\|_\infty \le \gamma\varepsilon$$

Importance of Contraction

- Any two value functions get closer
- True value function V* is a fixed point (value doesn't change with iteration)
- Max norm distance from V* decreases *dramatically* quickly with iterations

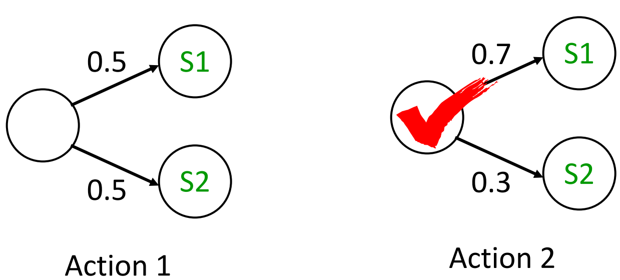
$$\left\| V_0 - V^* \right\|_\infty = \varepsilon \rightarrow \left\| V_n - V^* \right\|_\infty \le \gamma^n \varepsilon$$

Finding Good Policies

Suppose an expert told you the "true value" of each state:

V(S1) = 10

V(S2) = 5



Improving Policies

- How do we get the optimal policy?
- If we knew the values under the optimal policy, then just take the optimal action in every state
- How do we define these values?
- ***Fixed point*** equation with choices (Bellman equation):

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

Decision theoretic optimal choice given V*
If we know V*, picking the optimal action is easy
If we know the optimal actions, computing V* is easy
How do we compute both at the same time?

Value Iteration

We can't solve the system directly with a max in the equation
Can we solve it by iteration?

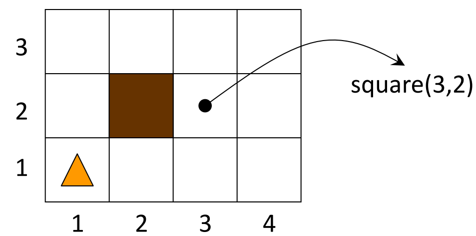
$$V_{i+1}(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_i(s')$$

- Called *value iteration* or simply *successive approximation*
- Same as value determination, but we can *change* actions
- Converges to V*

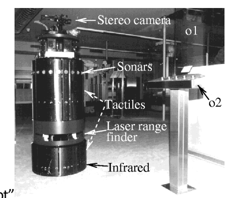
- Convergence:
  - Can't do eigenvalue analysis (not linear)
  - Still monotonic
  - Still a contraction in max norm (fun exercise)
  - Converges quickly

Robot Navigation Example

(from Russell & Norvig AIMA text)

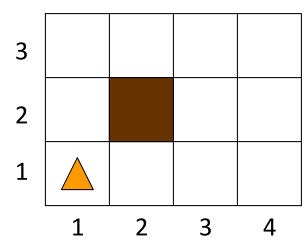


- The robot (shown) lives in a world described by a 4x3 grid of squares with square (2,2) occupied by an obstacle
- A state is defined by the square in which the robot is located: (1,1) in the above figure
  $\rightarrow$ 11 states



From Burgard et al.,
"Experiences with an interactive museum tour-guide robot"

Action (Transition) Model

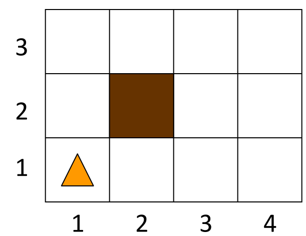


U brings the robot to:
- (1,2) with probability 0.8
- (2,1) with probability 0.1
- (1,1) with probability 0.1

- In each state, the robot's possible actions are {U, D, R, L}
- For each action:
  - With probability 0.8 the robot does the right thing (moves up, down, right, or left by one square)
  - With probability 0.1 it moves in a direction perpendicular to the intended one
  - If the robot can't move, it stays in the same square

  [This model satisfies the Markov condition]

Action (Transition) Model



L brings the robot to:
- (1,1) with probability 0.8 + 0.1 = 0.9
- (1,2) with probability 0.1

- In each state, the robot's possible actions are {U, D, R, L}
- For each action:
  - With probability 0.8 the robot does the right thing (moves up, down, right, or left by one square)
  - With probability 0.1 it moves in a direction perpendicular to the intended one
  - If the robot can't move, it stays in the same square

  [This model satisfies the Markov condition]

Terminal States, Rewards, and Costs

3	-.04	-.04	-.04	+1
2	-.04		-.04	-1
1	-.04	-.04	-.04	-.04
	1	2	3	4

"terminal" states
Not part of formal MDP specification. Usually handled by forcing state to have a fixed value, e.g. +1

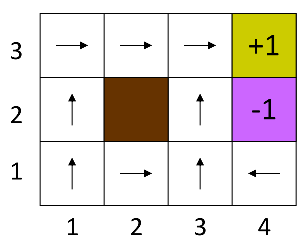
- Two terminal states: (4,2) and (4,3)
- Rewards:
  - R(4,3) = +1 [The robot finds gold]
  - R(4,2) = -1 [The robot gets trapped in quicksand]
  - R(s) = -0.04 in all other states

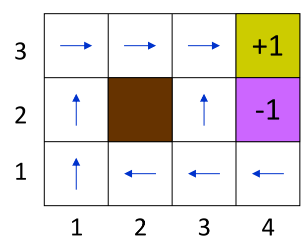
- This example (from the Russell & Norvig text) assumes no discounting (γ=1)
- Discussion: Is this a good modeling decision?

How to Implement Terminal States

- Modify your algorithm
  - For states s that are "terminal"
    - For an iterative solver, just set V(s)=R(s) at each iteration
    - If using matrix inversion, hack your matrix
- Modify your MDP
  - Create a state T with R(T)=0, P(T|T,a)=1 for all a
  - For all states s that are "terminal"
    - Set P(T|s,a) = 1 for all a
    - This forces V(s)=R(s)

The Optimal Policy is *Stationary*

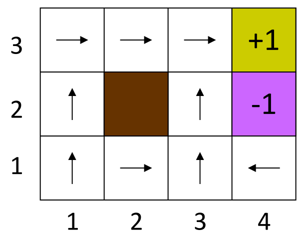


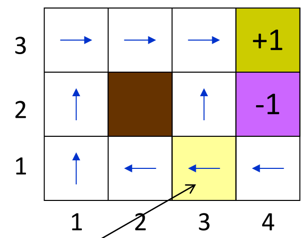


- A stationary policy is a complete map π: state $\rightarrow$ action
- For each non-terminal state it recommends an action, independent of when and how the state is reached
- Under the Markov and infinite horizon assumptions, the optimal policy π^* is necessarily a stationary policy
  [The best action in a state does not depend on the past]

Is it obvious which policy is optimal for this problem?

(Stationary) Policy

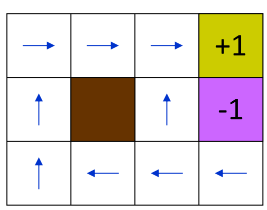




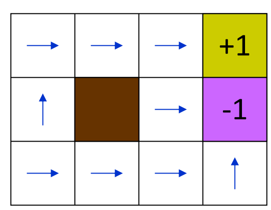
- A stationary policy is a complete map π: state $\rightarrow$ action
- For each non-terminal state it recommends an action, independent of when and how the
- Under the M mal policy π^* is necessarily a stationary policy
  [The best action in a state does not depend on the past]

The optimal policy tries to avoid "dangerous" state (3,2)

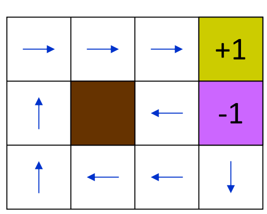
Optimal Policies for Various R(s)



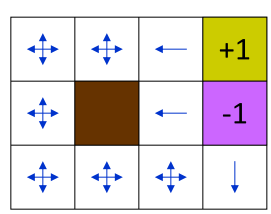
R(s) = -0.04



R(s) = -2



R(s) = -0.01



R(s) > 0

Bellman Equation

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

The utility of s depends on the utility of other states s' (possibly, including s), and vice versa

Appl(s) used if not all actions are defined in all states

- If s is terminal:

$$V(s) = R(s)$$

- If s is non-terminal:

$$V(s) = R(s) + \max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|s,a)V(s')$$

[Bellman equation]

- $\pi^*(s) = \arg\max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|s,a)V(s')$

Value Iteration Applied

3	0	0	0	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

→

3	0.81	0.87	0.92	+1
2	0.76		0.66	-1
1	0.71	0.66	0.61	0.39
	1	2	3	4

1. Initialize the utility of each non-terminal states to $V_0(s) = 0$
2. For t = 0, 1, 2, ... do

$$V_{t+1}(s) = R(s) + \max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|s,a)V_t(s')$$

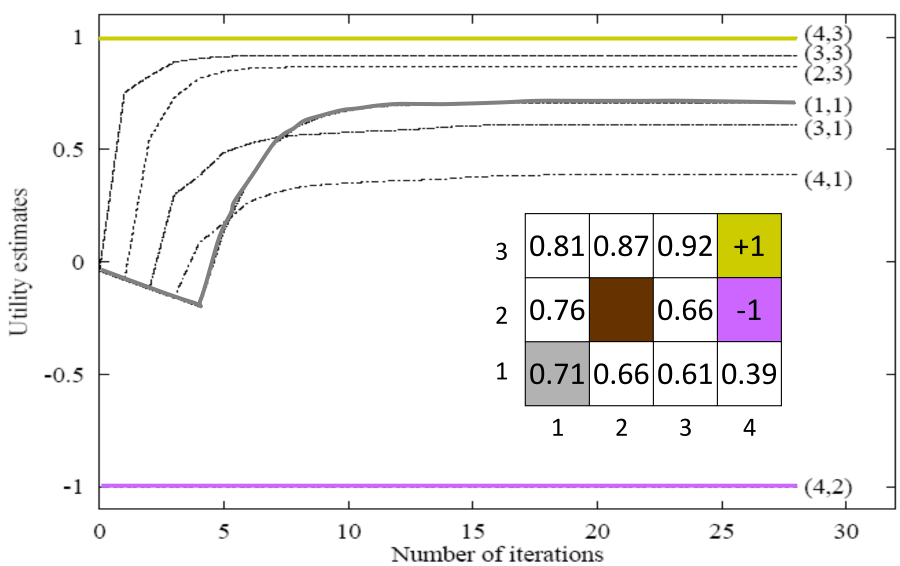
for each non-terminal state s

State Utilities/Values

3	0.81	0.87	0.92	+1
2	0.76		0.66	-1
1	0.71	0.66	0.61	0.39
	1	2	3	4

- The utility of a state s is the maximal expected amount of reward that the robot will collect from s and future states by executing some action in each encountered state, until it reaches a terminal state (infinite horizon)
- Under the Markov and infinite horizon assumptions, the utility of s is independent of when and how s is reached
  [It only depends on the possible sequences of states after s, not on the possible sequences before s]

Convergence of Value Iteration



Properties of Value Iteration

- VI converges to V* ($||.||_\infty$ from V* shrinks by γ factor each iteration)
- Converges to optimal policy
- Why? (Because we figure out V*, optimal policy is argmax)
- Optimal policy is stationary
  (i.e. Markovian – depends only on current state)

Policy Iteration

Greedy Policy Construction

Let's *name* the action that looks best WRT V:

$$\pi_v(s) = \arg\max_a R(s,a) + \gamma \underbrace{\sum_{s'} P(s' \mid s,a) V(s')}_{\text{Expectation over next-state values}}$$

$$\pi_v = \text{greedy}(V)$$

Bootstrapping: Policy Iteration

Idea: Greedy selection is useful even with suboptimal V

Guess $\pi_V = \pi_0$
$V_{\pi V}$ = value of acting on π_V
(solve linear system)
$\pi_V \leftarrow \text{greedy}(V_\pi)$

Repeat until policy doesn't change

Guaranteed to find optimal policy
Usually takes very small number of iterations
Computing the value functions is the expensive part

Comparing VI and PI

- VI
  - Value changes at every step
  - Policy *may* change before exact value of policy is computed
  - Many relatively cheap iterations
- PI
  - Alternates policy/value updates
  - Solves for value of each policy *exactly*
  - Fewer, slower iterations (need to invert matrix)
- Convergence
  - Both are contractions in max norm
  - PI is *shockingly* fast (small number of iterations) in practice

Computational Complexity

- VI and PI are both contraction mappings w/rate γ
  (we didn't prove this for PI in slides)
- VI costs less per iteration
- For n states, a actions PI tends to take O(n) iterations in practice
  - Recent(ish) results indicate ~$O(n^2a/1-\gamma)$ worst case
  - Interesting aside: Biggest insight into PI came ~50 years after the algorithm was introduced

MDP Limitations → Reinforcement Learning

- MDP operate at the level of *states*
  - States = atomic events
  - We usually have exponentially (or infinitely) many of these
- We assume P and R are known

- Machine learning to the rescue!
  - Infer P and R (implicitly or explicitly from data)
  - Generalize from small number of states/policies

A Unified View of Value Iteration and Policy Iteration

Notation

- Update for for a fixed policy – definition of T^{π} operator (matrix-vector form):

$$T^{\pi}V \equiv R^{\pi} + \gamma P^{\pi}V$$

- Update with policy improvement – definition of the T operator:

$$TV(s) \equiv \max_{a} r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')$$

Value Determination

- For 0 steps $V_0 = R^{\pi}$

- For i steps $V_i = T^{\pi}V_{i-1} = T^{\pi}T^{\pi}V_{i-2} = \cdots = (T^{\pi})^i R^{\pi}$

- Infinite horizon $\lim_{i\to\infty} V_i = (T^{\pi})^{\infty}R^{\pi} = (1-\gamma P^{\pi})^{-1}R^{\pi} = V^{\pi}$

Value Iteration (includes MAX)

- For 0 steps $V_0 = R$ (If R depends on a, pick a with the highest immediate reward)
- For i steps $V_i = TV_{i-1} = T^i R$
- Infinite horizon $\lim_{i \to \infty} V_i = T^\infty R = TV^* = V^*$

Modified Policy Iteration

- Guess V_0 (usually just R), and π - or set π = greedy(V_0)
- i=1
- Repeat until convergence*
  - For j=1 to n
    - $V_i = T^\pi V_{i-1}$
    - i = i+1

    n steps of iterative policy evaluation
  - π =greedy(V_{i-1})
- Special cases: n=1 (VI), n⟶∞ (PI)

Q-Values

- "Shift" or "split" Bellman equation
  - $V(s) = \max_a Q(s,a)$
  - $Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$

- What's the advantage of this representation?
- Trade offs in storing different things:
  - V(S) – store values, potentially expensive to compute policy
  - π(s) – stores policy, but forgets values
  - Q(s,a) – stores values, easy to compute policy is #actions is small

Bonus Material

Linear Programming Review

- Minimize: $\mathbf{c}^T\mathbf{x}$
- Subject to: $\mathbf{Ax} \geq \mathbf{b}$
- Can be solved in weakly polynomial time
- Arguably most common and important optimization technique in history

Linear Programming

$$V(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')$$

Issue: Turn the non-linear max into a collection of linear constraints

$$\forall s,a : V(s) \geq \underbrace{R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')}$$

MINIMIZE: $\sum_s V(s)$

Optimal action has tight constraints

Weakly polynomial; slower than PI in practice
(though can be modified to behave like PI)