Markov Decision Processes (MDPs)

Ron Parr
CPS 590.2

The Winding Path to RL

- Decision Theory
- Markov Decision Processes
- Reinforcement Learning
- Descriptive theory of optimal behavior
- Mathematical/Algorithmic realization of Decision Theory
- Application of learning techniques to challenges of MDPs with numerous or unknown parameters
Covered Today

- Decision Theory Review

- MDPs

- Algorithms for MDPs
  - Value Determination
  - Optimal Policy Selection
    - Value Iteration
    - Policy Iteration
    - Linear Programming

Decision Theory

What does it mean to make an optimal decision?

- Asked by economists to study consumer behavior
- Asked by MBAs to maximize profit
- Asked by leaders to allocate resources
- Asked in OR to maximize efficiency of operations
- Asked in AI to model intelligence

- Asked (sort of) by any intelligent person every day
Utility Functions

• A utility function is a mapping from world states to real numbers
• Also called a value function
• Rational or optimal behavior is typically viewed as maximizing expected utility:

$$\max_a \sum_s P(s \mid a) U(s)$$

a = actions, s = states

Swept under the rug today

• Utility of money (assumed 1:1)
• How to determine costs/utilities
• How to determine probabilities
Playing a Game Show

- Assume series of questions
  - Increasing difficulty
  - Increasing payoff
- Choice:
  - Accept accumulated earnings and quit
  - Continue and risk losing everything
- “Who wants to be a millionaire?”

State Representation

Dollar amounts indicate the payoff for getting the question right.

Probabilistic Transitions on Attempt to Answer

- Start $100
- $0
- 1 correct $1,000
- $0
- 2 correct $10K
- $0
- 3 correct $50K
- $0
- $61,100

Downward green arrows indicate the choice to exit the game.

N.B.: These exit transitions should actually correspond to states.

Green indicates profit at exit from game.
Making Optimal Decisions

- Work backwards from future to present

- Consider $50,000 question
  - Suppose $P(\text{correct}) = 1/10$
  - $V(\text{stop}) = $11,100
  - $V(\text{continue}) = 0.9 * $0 + 0.1 * $61.1K = $6.11K$

- Optimal decision stops

Working Backwards

\[ V=\$3,749 \quad V=\$4,166 \quad V=\$5,555 \quad V=\$11.1K \]

\[ \$100 \quad \rightarrow \quad \$1K \quad \rightarrow \quad \$10K \quad \rightarrow \quad \$50K \]

Red X indicates bad choice
Decision Theory Review

- Provides theory of optimal decisions
- Principle of maximizing utility
- Easy for small, tree structured spaces with
  - Known utilities
  - Known probabilities

Covered in Today

- Decision Theory
- MDPs
- Algorithms for MDPs
  - Value Determination
  - Optimal Policy Selection
    - Value Iteration
    - Policy Iteration
    - Linear Programming
Dealing with Loops

Suppose you can pay $1000 (from any losing state) to play again.

\[ V(s_0) = 0.10(-1000 + V(s_0)) + 0.90V(s_1) \]
\[ V(s_1) = 0.25(-1000 + V(s_0)) + 0.75V(s_2) \]
\[ V(s_2) = 0.50(-1000 + V(s_0)) + 0.50V(s_3) \]
\[ V(s_3) = 0.90(-1000 + V(s_0)) + 0.10(61100) \]

From Policies to Linear Systems

- Suppose we always pay until we win.
- What is value of following this policy?

\[ V(s_0) = 0.10(-1000 + V(s_0)) + 0.90V(s_1) \]
\[ V(s_1) = 0.25(-1000 + V(s_0)) + 0.75V(s_2) \]
\[ V(s_2) = 0.50(-1000 + V(s_0)) + 0.50V(s_3) \]
\[ V(s_3) = 0.90(-1000 + V(s_0)) + 0.10(61100) \]
And the solution is...

\[ V = \frac{3,749}{10} \]
\[ V = \frac{4,166}{2} \]
\[ V = \frac{5,555}{4} \]
\[ V = \frac{11,11}{10} \]

V = $3,749  
V = $4,166  
V = $5,555  
V = $11,111  

w/o cheat

9/10  3/4  1/2  1/10

$-1000

Is this optimal?  
How do we find the optimal policy?

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)
- Discount factor: \( \gamma \)
- Policy: \( \pi(s) \rightarrow a \)

Objective: **Maximize expected, discounted return**  
(decision theoretic optimal behavior)
Applications of MDPs

• AI/Computer Science
  – Robotic control (Koenig & Simmons, Thrun et al., Kaelbling et al.)
  – Air Campaign Planning (Meleual 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
  – 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
Understanding Discounting

- **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**
  - Suppose $\varepsilon$ probability of dying at each decision interval
  - Transition w/prob $\varepsilon$ to state with value 0
  - Equivalent to $1 - \varepsilon$ 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
Covered Today

- Decision Theory
- MDPs

- Algorithms for MDPs
  - Value Determination
  - Optimal Policy Selection
    - Value Iteration
    - Policy Iteration
    - Linear Programming

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

Bellman Equation for a fixed policy \(\pi\)

Determine the value of each state under policy \(\pi\)

\[ V(s_1) = 1 + \gamma(0.4V(s_2) + 0.6V(s_3)) \]
Matrix Form

\[ P = \begin{pmatrix}
    P(s_1 | s_1, \pi(s_1)) & P(s_2 | s_1, \pi(s_1)) & P(s_3 | s_1, \pi(s_1)) \\
    P(s_1 | s_2, \pi(s_2)) & P(s_2 | s_2, \pi(s_2)) & P(s_3 | s_2, \pi(s_2)) \\
    P(s_1 | s_3, \pi(s_3)) & P(s_2 | s_3, \pi(s_3)) & P(s_3 | s_3, \pi(s_3))
\end{pmatrix} \]

\[ V = \gamma P \pi V + R \]

This is a generalization of the game show example from earlier

How do we solve this system efficient? Does it even have a solution?

Solving for Values

\[ V = \gamma P \pi V + R \]

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

\[ V = (I - \gamma P \pi)^{-1} R \]

Guaranteed invertible because \( \gamma P \pi \)
has spectral radius < 1
Iteratively Solving for Values

\[ V = \gamma P \pi V + R \]

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

\[ V^{i+1} = \gamma P \pi V^i + R \]

Guaranteed convergent because \( \gamma P \pi \) has spectral radius \(<1 \)

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
  \[ \| \mathbf{V} \|_\infty = \max_i V^i \]

• Consider V1 and V2
  \[ \| V^a_i - V^b_i \|_\infty = \varepsilon \]

• WLOG say
  \[ V^a_i \leq V^b_i + \vec{\varepsilon} \]  (Vector of all \( \varepsilon \)'s)

Contraction Analysis Contd.

• At next iteration for \( V^b \):
  \[ V^b_2 = R + \gamma PV^1_1 \]

• For \( V^a \)
  \[ V^a_2 = R + \gamma P \left( V^a_1 \right) \leq R + \gamma P \left( V^b_1 + \vec{\varepsilon} \right) = R + \gamma PV^b_1 + \gamma P \vec{\varepsilon} = R + \gamma PV^b_1 + \gamma \vec{\varepsilon} \]

• Conclude:
  \[ \| V^a_2 - V^b_2 \|_\infty \leq \gamma \varepsilon \]
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

\[ \| V_0 - V^* \|_\infty = \varepsilon \rightarrow \| V_n - V^* \|_\infty \leq \gamma^n \varepsilon \]

Covered Today

- Decision Theory
- MDPs
- Algorithms for MDPs
  - Value Determination
  - Optimal Policy Selection
    - Value Iteration
    - Policy Iteration
    - Linear Programming
Finding Good Policies

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

\[ V(S1) = 10 \quad V(S2) = 5 \]

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

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

Properties of Value Iteration

• VI converges to the optimal policy
  (implicit in the maximizing action at each state)

• Why? (Because we figure out \( V^* \))

• Optimal policy is stationary (i.e. Markovian – depends only on current state)

• Why? (Because we are summing utilities. Thought experiment: Suppose you think it’s better to change actions the second time you visit a state. Why didn’t you just take the best action the first time?)
Covered Today

- Decision Theory
- MDPs
- Algorithms for MDPs
  - Value Determination
  - Optimal Policy Selection
    - Value Iteration
    - Policy Iteration
    - Linear Programming

Greedy Policy Construction

Let’s name the action that looks best WRT $V$:

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

Expectation over next-state values

$$
\pi_v = \text{greedy}(V)
$$
Consider our first policy

V=$3.7K  V=$4.1K  V=$5.6K  V=$11.1K  w/o cheat

Recall: We played until last state, then quit
Is this greedy with cheat option?

Value of paying to cheat in the first state is:
0.1(-1000 + 3.7K) + 0.9*(4.1K)=$3960
(much better than just giving up, which has value 0)

Bank

Bootstrapping: Policy Iteration

Idea: Greedy selection is useful even with suboptimal V

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

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 at every step
  - Many 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 in practice

Computational Complexity

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

- Decision Theory
- MDPs
- Algorithms for MDPs
  - Value Determination
  - Optimal Policy Selection
    - Value Iteration
    - Policy Iteration
    - Linear Programming

Linear Programming Review

- Minimize: \( c^T x \)
- Subject to: \( Ax \geq b \)
- 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

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

**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)