CS 188: Artificial Intelligence

Markov Decision Processes II

Instructor: Angela Liu and Yanlai Yang

University of California, Berkeley

[These slides adapted from Dan Klein and Pieter Abbeel]
Values of States

- Recursive definition of value:

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

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

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

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero.

- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:
  
  $$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$$

- Repeat until convergence, which yields $V^*$.

- Complexity of each iteration: $O(S^2A)$.

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do.
$k=0$

Noise $= 0.2$
Discount $= 0.9$
Living reward $= 0$
k=1

VALUES AFTER 1 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=2$

VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
k=3

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=4$

VALUES AFTER 4 ITERATIONS

- Top row: 0.37, 0.66, 0.83, 1.00
- Middle row: 0.00, 0.51, -1.00
- Bottom row: 0.00, 0.00, 0.31

Noise = 0.2
Discount = 0.9
Living reward = 0
k=5

VALUES AFTER 5 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
k=6

VALUES AFTER 6 ITERATIONS

0.59  0.73  0.85  1.00

0.41  0.57  -1.00

0.21  0.31  0.43  0.19

Noise = 0.2
Discount = 0.9
Living reward = 0
<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.62</td>
<td>0.74</td>
<td>0.85</td>
<td>1.00</td>
</tr>
<tr>
<td>0.50</td>
<td>0.57</td>
<td>-1.00</td>
<td></td>
</tr>
<tr>
<td>0.34</td>
<td>0.36</td>
<td>0.45</td>
<td>0.24</td>
</tr>
</tbody>
</table>

Values after 7 iterations

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=8$

VALUES AFTER 8 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
k=10

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
k=11

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k = 12$

VALUES AFTER 12 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
k=100

VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
Policy Extraction
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 (one step)

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

This is called policy extraction, since it gets the policy implied by the values.
Let’s imagine we have the optimal q-values:

How should we act?
  - Completely trivial to decide!

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

Important lesson: actions are easier to select from q-values than values!
Problems with Value Iteration

- Value iteration repeats the Bellman updates:

\[ V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right] \]

- Problem 1: It’s slow – \(O(S^2A)\) per iteration

- Problem 2: The “max” at each state rarely changes

- Problem 3: The policy often converges long before the values
Noise = 0.2
Discount = 0.9
Living reward = 0

VALUES AFTER 12 ITERATIONS

k=12
k=100

VALUES AFTER 100 ITERATIONS

<table>
<thead>
<tr>
<th>0.64</th>
<th>0.74</th>
<th>0.85</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.57</td>
<td></td>
<td>0.57</td>
<td>-1.00</td>
</tr>
<tr>
<td>0.49</td>
<td>0.43</td>
<td>0.48</td>
<td>0.28</td>
</tr>
</tbody>
</table>

Noise = 0.2
Discount = 0.9
Living reward = 0
Policy Methods
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 Improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- **This is Policy Iteration**
  - It’s still optimal!
  - Can converge (much) faster under some conditions
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.
Another basic operation: compute the utility of a state $s$ under a fixed (generally non-optimal) policy

Define the utility of a state $s$, under a fixed policy $\pi$:

$V^\pi(s) = \text{expected total discounted rewards starting in } s \text{ and following } \pi$

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

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

\[
V_0^\pi(s) = 0
\]

\[
V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]
\]

- 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)
Example: Policy Evaluation

Always Go Right

Always Go Forward
Example: Policy Evaluation

Always Go Right

Always Go Forward
Policy Iteration
Policy Iteration

- **Evaluation:** For fixed current policy $\pi$, find values with policy evaluation:
  - Iterate until values converge:
    \[
    V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[ R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]
    \]

- **Improvement:** For fixed values, get a better policy using policy extraction:
  - One-step look-ahead:
    \[
    \pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^{\pi_i}(s') \right]
    \]
Example: Policy Iteration

\[
\pi_0 = \begin{bmatrix} S & F & \emptyset \end{bmatrix}
\]

**Policy Evaluation:**

\[
V_{\pi_0}(B) = 1 + 0.9 \cdot V_{\pi_0}(B) \quad \rightarrow \quad V_{\pi_0}(B) = 10
\]

\[
V_{\pi_0}(W) = -10 + 0.9 \cdot V_{\pi_0}(O) \quad \rightarrow \quad V_{\pi_0}(W) = -10
\]

\[
V_{\pi_0}(O) = 0
\]

**Policy Improvement:**

\[
\pi_1 = \begin{bmatrix} S: 1 + 0.9 \cdot 10 = 10 \\ F: 0.5(2 + 0.9 \cdot 10) + 0.5(2 + 0.9 \cdot -10) = 0 \end{bmatrix}
\]

Assume discount = 0.9

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

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

```
\[ \pi_0 \]

\[
\begin{array}{ccc}
S & F & \emptyset \\
\end{array}
\]

Policy Evaluation:

\[ V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \quad \rightarrow \quad V^{\pi_0}(B) = 10 \]
\[ V^{\pi_0}(W) = -10 + 0.9 \cdot V^{\pi_0}(O) \quad \rightarrow \quad V^{\pi_0}(W) = -10 \]
\[ V^{\pi_0}(O) = 0 \]

Policy Improvement:

\[
\pi_1 \quad S: 0.5(1 + 0.9 \cdot 10) + 0.5(1 + 0.9 \cdot -10) = 0 \\
F: -10 + 0.9 \cdot 0 = -10
\]

Assume discount = 0.9

\[ V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^{\pi}(s')] \]
\[ \pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')] \]
Example: Policy Iteration

\[ \pi_1 \]

\[ \begin{array}{c}
S & S & \emptyset
\end{array} \]

Policy Evaluation:

\[ V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \rightarrow V^{\pi_0}(B) = 10 \]

\[ V^{\pi_0}(W) = 0.5 \left(1 + 0.9 \cdot V^{\pi_0}(B)\right) + 0.5 \left(1 + 0.9 \cdot V^{\pi_0}(W)\right) \rightarrow V^{\pi_0}(W) = 10 \]

\[ V^{\pi_0}(\emptyset) = 0 \]

Policy Improvement:

\[ \pi_2 \]

\[ \begin{array}{c}
S: 1 + 0.9 \cdot 10 = 10 \\
F: 0.5(2 + 0.9 \cdot 10) + 0.5(2 + 0.9 \cdot 10) = 11
\end{array} \]

Assume discount = 0.9

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

\[ \pi_{i+1}(s) = \text{arg max}_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^{\pi_i}(s') \right] \]
Example: Policy Iteration

Policy Evaluation:

\[ V^{\pi_0}(B) = 1 + 0.9 \cdot V^{\pi_0}(B) \quad \rightarrow \quad V^{\pi_0}(B) = 10 \]
\[ V^{\pi_0}(W) = 0.5(1 + 0.9 \cdot V^{\pi_0}(B)) + 0.5(1 + 0.9 \cdot V^{\pi_0}(W)) \quad \rightarrow \quad V^{\pi_0}(W) = 10 \]
\[ V^{\pi_0}(O) = 0 \]

Policy Improvement:

\[ \pi_2 \]
\[ F: 0.5(1 + 0.9 \cdot 10) + 0.5(1 + 0.9 \cdot 0) = 10 \]
\[ F: -10 + 0.9 \cdot 0 = -10 \]
Example: Policy Iteration

Policy Evaluation:

\[ V^\pi_0(B) = 0.5(2 + 0.9 \cdot V^\pi_0(B)) + 0.5(2 + 0.9 \cdot V^\pi_0(W)) \]
\[ V^\pi_0(W) = 0.5(1 + 0.9 \cdot V^\pi_0(B)) + 0.5(1 + 0.9 \cdot V^\pi_0(W)) \rightarrow V^\pi_0(W) = 14.5 \]
\[ V^\pi_0(\emptyset) = 0 \]

Policy Improvement:

\[ \pi_2 = \begin{cases} F, & 0.5(2 + 0.9 \cdot 15.5) + 0.5(2 + 0.9 \cdot 14.5) = 15.5 \\ S, & 1 + 0.9 \cdot 15.5 = 14.95 \end{cases} \]
Assume discount = 0.9
Example: Policy Iteration

Policy Evaluation:

$V^{\pi_0}(B) = 0.5 (2 + 0.9 \cdot V^{\pi_0}(B)) + 0.5 (2 + 0.9 \cdot V^{\pi_0}(W))$

$V^{\pi_0}(W) = 0.5 (1 + 0.9 \cdot V^{\pi_0}(B)) + 0.5 (1 + 0.9 \cdot V^{\pi_0}(W)) \rightarrow V^{\pi_0}(W) = 14.5$

$V^{\pi_0}(O) = 0$

Policy Improvement:

Assume discount = 0.9

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

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

$\pi_2$

$\begin{array}{ccc}
F & S & \emptyset \\
\end{array}$

Policy Evaluation: $\rightarrow V^{\pi_0}(B) = 15.5$

$V^{\pi_0}(B) = 0.5 (2 + 0.9 \cdot V^{\pi_0}(B)) + 0.5 (2 + 0.9 \cdot V^{\pi_0}(W))$

$V^{\pi_0}(W) = 0.5 (1 + 0.9 \cdot V^{\pi_0}(B)) + 0.5 (1 + 0.9 \cdot V^{\pi_0}(W)) \rightarrow V^{\pi_0}(W) = 14.5$

$V^{\pi_0}(O) = 0$

$\pi_3$

$\begin{array}{ccc}
F & S & \emptyset \\
\end{array}$

Policy Improvement:

$F: 0.5 (1 + 0.9 \cdot 15.5) + 0.5 (1 + 0.9 \cdot 14.5) = 14.5$

$F: -10 + 0.9 \cdot 0 = -10$
**Example: Value Iteration**

<table>
<thead>
<tr>
<th>$V_0$</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$V_2$</td>
<td>3.35</td>
<td>2.35</td>
<td>0</td>
</tr>
<tr>
<td>$V_3$</td>
<td>4.565</td>
<td>3.565</td>
<td>0</td>
</tr>
<tr>
<td>$V_4$</td>
<td>5.6585</td>
<td>4.6585</td>
<td>0</td>
</tr>
</tbody>
</table>

Assume discount $= 0.9$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$$

... $V_{91}$ | 15.499 | 14.499 | 0 |
Convergence*

**Proof Sketch**

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

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

- **Monotonic improvement:** \( \forall s \ V^\pi_{i+1}(s) \geq V^\pi_i(s) \)
- **Termination:** \( \pi_i \) is optimal if \( \forall s \ \pi_i(s) = \pi_{i+1}(s) \)
  - \( \pi_{i+1}(s) \) chooses the best action to take under \( V^\pi_i(s) \)
  - If \( \forall s \ \pi_i(s) = \pi_{i+1}(s) \), then \( \pi_i(s) \) was already the best action for all states

**Guaranteed termination:** only finite number of policies
Comparison

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

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
  - Runtime per iteration: $O(|S|^2|A|)$

- 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)
    - Runtime per value iteration update: $O(|S|^3)$ → total runtime to get fixed policy values: $O(|S|^3)$
  - After policy is evaluated, a new policy is chosen (slow like a value iteration pass → $O(|S|^2|A|)$)
  - The new policy will be better (or we’re done)
  - Runtime per iteration: $O(|S|^3) + O(|S|^2|A|)$ → slower but can take much fewer iterations

- Both are dynamic programs for solving MDPs
Summary: MDP Algorithms

- So you want to....
  - Compute optimal values: use \textit{value iteration} or \textit{policy iteration}
  - Compute values for a particular policy: use \textit{policy evaluation}
  - Turn your values into a policy: use \textit{policy extraction} (one-step lookahead)

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