CS440/ECE448 Lecture 20: Markov Decision Processes

Mark Hasegawa-Johnson, 3/2024
These slides are in the public domain.

Grid World
Invented and drawn by Peter Abbeel and Dan Klein, UC Berkeley CS 188
Outline

• Problem statement
• Utility
• The discount factor
• Value Iteration
• Policy Iteration
• Comparison of value iteration and policy iteration
How does an intelligent agent plan its actions?

- If there is no randomness: Use A* search to plan the best path
- What if our movements are affected by randomness?
Example: Grid World
Invented by Peter Abbeel and Dan Klein

- Maze-solving problem: state is $s = (i, j)$, where $0 \leq i \leq 2$ is the row and $0 \leq j \leq 3$ is the column.
- The robot is trying to find its way to the diamond.
- If it reaches the diamond, it gets a reward of $r((0,3)) = +1$ and the game ends.
- If it falls in the fire it gets a reward of $r((1,3)) = -1$ and the game ends.
Example: Grid World
Invented by Peter Abbeel and Dan Klein

Randomness: the robot has shaky actuators. If it tries to move forward,
• With probability 0.8, it succeeds
• With probability 0.1, it falls left
• With probability 0.1, it falls right

Source: P. Abbeel and D. Klein
Markov Decision Process

A Markov Decision Process (MDP) is defined by:

- A set of states, \( s \in S \)
- A set of actions, \( a \in A \)
- A transition model, \( P(S_{t+1} = s_{t+1} | S_t = s_t, a_t) \)
  - \( S_t \) is the state at time \( t \)
  - \( a_t \) is the action taken at time \( t \) (not random)
- A reward function, \( r(s) \)
Solving an MDP: The Policy

• The solution to a maze is a path: the shortest path from start to goal
• In MDP, finding 1 path is not enough: randomness might cause us to accidentally deviate from the optimal path.
Solving an MDP: The Policy

• Since $P(S_{t+1} = s_{t+1} | S_t = s_t, a_t)$ and $r(s)$ depend only on the state (the model is Markov), a complete solution can be expressed as follows:

• What is the best action to take in any given state?

• A policy, $a = \pi(s)$, is a function telling you, for any state $s$, what is the best action to take in that state.
Outline

• Problem statement
• Utility
• The discount factor
• Value Iteration
• Policy Iteration
• Comparison of value iteration and policy iteration
Utility

The utility of a state, $u(s)$, is defined to be:

- the sum of all current and future rewards that can be achieved if we start in state $s$,
- ...if we choose the best possible sequence of actions,
- ...and if we average over all possible results of those actions.
Example: Game show

• You’ve been offered a spot as a contestant in a game show.
• Reward: you receive successively larger prizes for each question you answer correctly, but if you answer any question incorrectly, you lose it all.
• Transition: the questions become harder and harder to answer.
• Actions: after each question, you can decide whether to take another question, or stop.
Example: Game show

Policy:

• If you’ve correctly answered N-1 questions, should you attempt question QN, or stop?
Example: Game show

Policy $\pi(Q4)$: If you’ve correctly answered 3 questions, should you attempt question Q4, or stop?

• If you stop: total reward is $11,100

• If you attempt Q4: expected total reward is $\frac{1}{10} \times 61100 + \frac{9}{10} \times 0 = 6110$

Policy: $\pi(Q4) = \text{stop}$.  
Utility: $u(Q4) = 11,100$
Example: Game show

Policy $\pi(Q3)$: If you’ve correctly answered 2 questions, should you attempt question Q3, or stop?

- If you stop: total reward is $1,100
- If you attempt Q3: expected total reward is $\frac{1}{2} \times 11,100 + \frac{1}{2} \times 0 = 5,550$

Policy: $\pi(Q3) = \text{continue}$. Utility: $u(Q3) = 5,550$
Example: Game show

Policy $\pi(Q2)$: If you’ve correctly answered 1 question, should you attempt question Q2, or stop?

- If you stop: total reward is $100$
- If you attempt Q2: expected total reward is
  $$\frac{3}{4} \times 5550 + \frac{1}{4} \times 0 = 4162.50$$

Policy: $\pi(Q2) = $ continue. 

Utility: $u(Q2) = 4162.50$
Example: Game show

Policy $\pi(Q1)$: If you’ve correctly answered no questions, then you have nothing to lose, so even though the chance of success is very small, you might as well try it!

Policy: $\pi(Q1) = \text{continue.}$

Utility: $u(Q1) = 41.63$
Utility

The utility of a state, \( u(s) \), is

- the maximum, over all possible sequences of actions, of
- the expected value, over all possible results of those actions, of
- the total of all future rewards.

\[
u(s) = r(s) + \max_a \sum_{s'} P(s'|s, a) \left( r(s') + \max_{a'} \sum_{s''} P(s''|s', s')(r(s'') + \cdots \cdots \cdots) \right)\]
Utility

The utility of a state, $u(s)$, is

- the maximum, over all possible sequences of actions, of
- the expected value, over all possible results of those actions, of
- the utility of the resulting state.

$$u(s) = r(s) + \max_a \sum_{s'} P(S_{t+1} = s'|S_t = s, a)u(s')$$
Outline

• Problem statement
• Utility
• The discount factor
• Value Iteration
• Policy Iteration
• Comparison of value iteration and policy iteration
Discount factor

You have just won a contest sponsored by the Galaxia Foundation. They offer you the choice of two options:

• $60,000 right now, or...
• $1000 per year, paid to you and your heirs annually forever.

Which option is better?
Discount factor

- Inflation has averaged 3.8% annually from 1960 to 2024.
- Equivalently, $1000 received one year from now is worth approximately $962 today.
- A reward of $1000 annually forever (starting today, t=0) is equivalent to an immediate reward of

\[ r = \sum_{t=0}^{\infty} 1000(0.962)^t = \frac{1000}{1 - 0.962} = \$26,316 \]

We call the factor \( \gamma = 0.962 \) the discount factor.
Discount factor

Why is a dollar tomorrow worth less than a dollar today?

• A dollar will buy less tomorrow
• The person paying you might go out of business
• You might have to go into hiding and become unable to collect

The discount factor, $\gamma$, is our model of the unknowable uncertainty of promised future rewards.
The Bellman Equation

\[ u(s) = r(s) + \gamma \max_a \sum_{s'} P(S_{t+1} = s'|S_t = s, a)u(s') \]

• The Bellman equation specifies the utility of the current state.
• In solving the Bellman equation, we also find the optimum action, which is the policy.
• However...
The Bellman Equation

\[
\begin{bmatrix}
    u(1) \\
    \vdots \\
    u(n)
\end{bmatrix} = 
\begin{bmatrix}
    r(1) \\
    \vdots \\
    r(n)
\end{bmatrix} 
+ \gamma \max_a \begin{bmatrix}
    P(1|1, a) & \cdots & P(1|n, a) \\
    \vdots & \ddots & \vdots \\
    P(n|1, a) & \cdots & P(n|n, a)
\end{bmatrix} \begin{bmatrix}
    u(1) \\
    \vdots \\
    u(N)
\end{bmatrix}
\]

• If there are \(n\) states, then the Bellman equation is \(n\) nonlinear equations in \(n\) unknowns.
• There is no closed-form solution; we must use an iterative solution
Outline

• Problem statement
• Utility
• The discount factor
• Value Iteration
• Policy Iteration
• Comparison of value iteration and policy iteration
Value iteration

The Bellman Equation:

\[ u(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a) u(s') \]

Value iteration solves the Bellman equation iteratively. In iteration number \( i \), for \( i = 0, 1, ... \),

- For all states \( s \), \( u_i(s) \) is an estimate of \( u(s) \)
- Start out with \( u_0(s) = 0 \) for all states
- In the \( i^{th} \) iteration,

\[ u_i(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a) u_{i-1}(s') \]
Value iteration

\[ u_i(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)u_{i-1}(s') \]

Notice that:

- After \( i \) iterations, \( u_i(s) \) has information about the rewards earned in the first \( i \) steps after the agent starts the maze.
- A policy designed based on \( u_i(s) \) will act in order to maximize reward in the first \( i \) steps of the maze.
- In this sense, it’s kind of like BFS: each iteration explores farther and farther away from the starting state.
Example: Grid world

Assume a “loitering penalty” of $r(s) = -0.04$ for all non-terminal states.
Value Iteration: Iteration 1

$$u_1(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a)u_0(s')$$

<table>
<thead>
<tr>
<th>$u_1(s)$</th>
<th>$r(s)$</th>
<th>$u_0(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.04</td>
<td>-0.04</td>
<td>0</td>
</tr>
<tr>
<td>-0.04</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.04</td>
<td>-0.04</td>
<td>-0.04</td>
</tr>
<tr>
<td>-0.04</td>
<td>-0.04</td>
<td>-0.04</td>
</tr>
<tr>
<td>-0.04</td>
<td>-0.04</td>
<td>-0.04</td>
</tr>
<tr>
<td>-0.04</td>
<td>-0.04</td>
<td>-0.04</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Value Iteration: Iteration 2

\[ u_2(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a) u_1(s') \]

\[ u_2(s) \]

\[ \begin{array}{ccc}
-0.08 & -0.08 & +0.75 \\
-0.08 & -0.08 & -0.08 \\
-0.08 & -0.08 & -0.08 \\
\end{array} = \]

\[ r(s) \]

\[ \begin{array}{ccc}
-0.04 & -0.04 & -0.04 \\
-0.04 & -0.04 & -0.04 \\
-0.04 & -0.04 & -0.04 \\
\end{array} + \gamma \max_a \]

\[ \sum_{s'} P(s'|s, left) u_1(s') \]

\[ \begin{array}{ccc}
-0.04 & -0.04 & -0.04 \\
-0.04 & -0.04 & -0.04 \\
-0.04 & -0.04 & -0.04 \\
\end{array} \]

\[ \sum_{s'} P(s'|s, up) u_1(s') \]

\[ \begin{array}{ccc}
-0.04 & -0.04 & +0.06 \\
-0.04 & -0.04 & -0.14 \\
-0.04 & -0.04 & -0.04 \\
\end{array} \]

\[ \sum_{s'} P(s'|s, right) u_1(s') \]

\[ \begin{array}{ccc}
-0.04 & -0.04 & -0.04 \\
-0.04 & -0.04 & -0.14 \\
-0.04 & -0.04 & -0.81 \\
\end{array} \]
Quiz

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2403836
Outline

• Problem statement
• Utility
• The discount factor
• Value Iteration
• Policy Iteration
• Comparison of value iteration and policy iteration
Method 2: Policy Iteration

• **Policy Evaluation:** \( u_i(s) = r(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) u_i(s') \)
  - Given a **fixed** policy \( \pi_i(s) \),
  - Calculate the resulting utility \( u_i(s) \).

• **Policy Improvement:** \( \pi_{i+1}(s) = \text{argmax} \sum_{a} P(s'|s, a) u_{i}(s') \)
  - Given a **fixed** utility \( u_i(s) \),
  - Find an improved \( \pi_{i+1}(s) \).

• Unlike Value Iteration, this is guaranteed to converge in a finite number of steps (less than or equal to the number of distinct policies)
Step 1: Policy Evaluation

Bellman equation: n nonlinear equations in n unknowns:

\[
\begin{bmatrix}
u(1)
\vdots
v(n)
\end{bmatrix} =
\begin{bmatrix}
r(1)
\vdots
r(n)
\end{bmatrix} + \gamma \max_a \begin{bmatrix}
P(1|1, a) & \cdots & P(1|n, a)
\vdots & \ddots & \vdots
P(n|1, a) & \cdots & P(n|n, a)
\end{bmatrix}
\begin{bmatrix}
u(1)
\vdots
v(N)
\end{bmatrix}
\]

Policy Evaluation: n linear equations in n unknowns:

\[
\begin{bmatrix}
u_i(1)
\vdots
v_i(n)
\end{bmatrix} =
\begin{bmatrix}
r(1)
\vdots
r(n)
\end{bmatrix} + \gamma \begin{bmatrix}
P(1|1, \pi_i(1)) & \cdots & P(1|n, \pi_i(n))
\vdots & \ddots & \vdots
P(n|1, \pi_i(1)) & \cdots & P(n|n, \pi_i(n))
\end{bmatrix}
\begin{bmatrix}
u_i(1)
\vdots
v_i(N)
\end{bmatrix}
\]

The difference is that policy evaluation is linear, so it can be solved by inverting a matrix: \(u_i = (I - \gamma P_i)^{-1} r\).
Example: Grid World

**Policy Evaluation:** $u_i(s) = r(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))u_i(s')$

- Assume the initial policy is $\pi_1(s) = “Go Right”$ for all states
- Solve the linear equations to find $u_1(s)$
Policy Improvement

**Policy Evaluation:** $u_i(s) = r(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))u_i(s')$

**Policy Improvement:** $\pi_{i+1}(s) = \arg\max_a \sum_{s'} P(s'|s, a)u_i(s')$
Outline

• Problem statement
• Utility
• The discount factor
• Value Iteration
• Policy Iteration
• Comparison of value iteration and policy iteration
Value iteration

Optimal utilities with discount factor 1
(Result of value iteration)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.812</td>
<td>0.868</td>
<td>0.918</td>
<td>+1</td>
</tr>
<tr>
<td>2</td>
<td>0.762</td>
<td>0.660</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
</tr>
</tbody>
</table>

Utility estimates

Number of iterations

Final policy
Comparison of value iteration and policy iteration

- Bellman equation is \( n \) equations in \( n \) unknowns; cannot be solved in closed form, needs an iterative solution
- Value iteration
  - Behaves like BFS: each iteration looks one step farther from the start node
  - Usually converges exponentially fast to the correct policy
  - However, if there are loops possible in the maze, may never converge exactly
- Policy iteration
  - Kind of like gradient descent: evaluate a policy, then improve it
  - Guaranteed to converge in a finite number of steps
  - Harder to implement, and might take a while before it starts to converge
Summary

• Bellman equation:
  \[ u(s) = r(s) + \gamma \max_a \sum_{s'} P(S_{t+1} = s'|S_t = s, a) u(s') \]

• Value iteration:
  \[ u_i(s) = r(s) + \gamma \max_a \sum_{s'} P(S_{t+1} = s'|S_t = s, a) u_{i-1}(s') \]

• Policy iteration:
  \[ u_i(s) = r(s) + \gamma \sum_{s'} P(S_{t+1} = s'|S_t = s, \pi_i(s)) u_i(s') \]
  \[ \pi_{i+1}(s) = \arg\max_a \sum_{s'} P(S_{t+1} = s'|S_t = s, a) u_i(s') \]