Markov Decision Processes II
The Bellman Equation

\[ U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \]

- The Bellman equation gives the utility of a state
- If there are \( n \) states, there are \( n \) Bellman Equations to solve
- This is a system of simultaneous equations
- But the equations are nonlinear because of the max operator
Iterative Solution

Define $U_1(s)$ to be the utility if the agent is at state $s$ and lives for 1 time step

$$U_1(s) = R(s)$$

Calculate this for all states $s$

Define $U_2(s)$ to be the utility if the agent is at state $s$ and lives for 2 time steps

$$U_2(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_1(s')$$

This has already been calculated above
The Bellman Update

More generally, we have:

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$$

- This is the maximum possible expected sum of discounted rewards (i.e., the utility) if the agent is at state $s$ and lives for $i+1$ time steps.
- This equation is called the Bellman Update.
The Bellman Update

• As the number of iterations goes to infinity, $U_{i+1}(s)$ converges to an equilibrium value $U^*(s)$.
• The final utility values $U^*(s)$ are solutions to the Bellman equations. Even better, they are the unique solutions and the corresponding policy is optimal.
• This algorithm is called **Value-Iteration**
• The optimal policy is given by:

$$
\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') U^*(s')
$$
The Value Iteration Algorithm

\[ U_1(s) = R(s) \]
\[ U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \]

1. Apply bellman update until it the utility function converges (to \( U^*(s) \)).
2. The optimal policy is given by:

\[ \pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') U^*(s') \]
Example

• We will use the following convention when drawing MDPs graphically:

- State
  \( R(s) \)

- Action

- Transition probability
Example

\[ \gamma = 0.9 \]

\[ \begin{align*}
A & \quad +12 \\
A1 & \quad -4 \\
A2 & \quad \text{circled}
\end{align*} \]

\[ \begin{align*}
B & \quad -4 \\
B1 & \quad \text{circled}
\end{align*} \]

\[ \begin{align*}
C & \quad +2 \\
C1 & \quad \text{circled}
\end{align*} \]

\[ \begin{align*}
0.5 & \quad 0.5 \\
0.25 & \quad 0.75 \\
1.0 & \quad 0.5 \\
\end{align*} \]
Example

\[ i = 1 \]
\[ U_1(A) = R(A) = 12 \]
\[ U_1(B) = R(B) = -4 \]
\[ U_1(C) = R(C) = 2 \]
Example

<table>
<thead>
<tr>
<th>$U_1(A)$</th>
<th>$U_1(B)$</th>
<th>$U_1(C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>-4</td>
<td>2</td>
</tr>
</tbody>
</table>

$i=2$

$U_2(A) = 12 + (0.9) \times \max\{(0.5)(12)+(0.5)(-4), (1.0)(2)\}$

$= 12 + (0.9) \times \max\{4.0, 2.0\} = 12 + 3.6 = 15.6$

$U_2(B) = -4 + (0.9) \times \{(0.25)(12)+(0.75)(-4)\} = -4 + (0.9)\times0 = -4$

$U_2(C) = 2 + (0.9) \times \{(0.5)(2)+(0.5)(-4)\} = 2 + (0.9)\times(-1)$

$= 2 - 0.9 = 1.1$
Example

<table>
<thead>
<tr>
<th>$U_2(A)$</th>
<th>$U_2(B)$</th>
<th>$U_2(C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.6</td>
<td>-4</td>
<td>1.1</td>
</tr>
</tbody>
</table>

i=3

$U_3(A) = 12 + (0.9) \times \max\{(0.5)(15.6)+(0.5)(-4),(1.0)(1.1)\} = 12 + (0.9) \times \max\{5.8,1.1\} = 12 + (0.9)(5.8) = 17.22$

$U_3(B) = -4 + (0.9) \times \{(0.25)(15.6)+(0.75)(-4)\} = -4 + (0.9)(3.9-3) = -4 + (0.9)(0.9) = -3.19$

$U_3(C) = 2 + (0.9) \times \{(0.5)(1.1)+(0.5)(-4)\} = 2 + (0.9)\{0.55-2.0\} = 2 + (0.9)(-1.45) = 0.695$
The Bellman Update

- What exactly is going on?
- Think of each Bellman update as an update of each local state
- If we do enough local updates, we end up propagating information throughout the state space
Value Iteration on the Maze

Notice that rewards are negative until a path to (4,3) is found, resulting in an increase in $U$. 

Graph showing the progression of $U$ over iterations for different states $(4,3)$, $(3,3)$, $(1,1)$, and $(3,1)$. The diagram on the right represents the maze with rewards at the end states (4,3) and (3,1) and a starting point at (1,2).
Value-Iteration Termination

When do you stop?

In an iteration over all the states, keep track of the maximum change in utility of any state (call this $\delta$)

When $\delta$ is less than some pre-defined threshold, stop

This will give us an approximation to the true utilities, we can act greedily based on the approximated state utilities
Comments

Value iteration is designed around the idea of the utilities of the states.

The computational difficulty comes from the max operation in the bellman equation.

Instead of computing the general utility of a state (assuming acting optimally), a much easier quantity to compute is the utility of a state assuming a policy.
Utility of a policy at state $s$

- $U_\pi(s)$: the utility of policy $\pi$ at state $s$
- $U^*(s)$ can be considered as $U^*_\pi(s)$ where $\pi^*$ is an optimal policy
- Given a fixed policy, can compute its utility at state $s$ as follows:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') \cdot U_\pi(s')$$

Note the difference from:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

Notice that there is no max operator, so the equations are linear, which can be solved directly.
Evaluating a Policy

Once we compute the utilities of the states under policy $\pi$, we can easily compute an improved policy $\pi'$:

$$\pi'(s) = \arg \max_a \sum_{s'} T(s, a, s') U_\pi(s')$$

Iteratively apply this idea, we will eventually converge to the optimal policy
Policy Iteration

- Start with a randomly chosen initial policy $\pi_0$
- Iterate until no change in utilities:
  1. **Policy evaluation**: given a policy $\pi_i$, calculate the utility $U_{\pi_i}(s)$ of every state $s$ using policy $\pi_i$ by solving:

     $$U_{\pi_i}(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_{\pi_i}(s')$$

  2. **Policy improvement**: calculate the new policy $\pi_{i+1}$ based on $U_i(s)$ ie.

     $$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') U_{\pi_i}(s')$$
Policy iteration comments

• In each step of policy iteration:
  – **Policy evaluation** involves solving a set of linear equations
  – **Policy improvement**: straightforward

• Each step of policy iteration is guaranteed to strictly improve the policy at some state when improvement is possible

• Converge to optimal policy

• Gives exact value of optimal policy
Policy Iteration Example

Do one iteration of policy iteration on the MDP below. Assume an initial policy of $\pi_1(\text{Hungry}) = \text{Eat}$ and $\pi_1(\text{Full}) = \text{Sleep}$. Let $\gamma = 0.9$
Policy Iteration Example

Policy Evaluation Phase

Use initial policy for Hungry: $\pi_1(\text{Hungry}) = \text{Eat}$

$U_1(\text{Hungry}) = -10 + (0.9)[(0.1)U_1(\text{Hungry}) + (0.9)U_1(\text{Full})]$  
$\Rightarrow U_1(\text{Hungry}) = -10 + (0.09)U_1(\text{Hungry}) + (0.81)U_1(\text{Full})$  
$\Rightarrow (0.91)U_1(\text{Hungry}) - (0.81)U_1(\text{Full}) = -10$

Use initial policy for Full: $\pi_1(\text{Full}) = \text{Sleep}$.  
$U_1(\text{Full}) = 10 + (0.9)[(0.8)U_1(\text{Full}) + (0.2)U_1(\text{Hungry})]$  
$\Rightarrow U_1(\text{Full}) = 10 + (0.72)U_1(\text{Full}) + (0.18)U_1(\text{Hungry})$  
$\Rightarrow (0.28)U_1(\text{Full}) - (0.18)U_1(\text{Hungry}) = 10$
Policy Iteration Example

\[(0.91)U_1(\text{Hungry}) - (0.81)U_1(\text{Full}) = -10 \text{ \ldots (Equation 1)}\]
\[(0.28)U_1(\text{Full}) - (0.18)U_1(\text{Hungry}) = 10 \text{ \ldots (Equation 2)}\]

Solve for \(U_1(\text{Hungry})\) and \(U_1(\text{Full})\)

From Equation 1:
\[(0.91)U_1(\text{Hungry}) = -10 + (0.81)U_1(\text{Full})\]
\[\Rightarrow U_1(\text{Hungry}) = (-10/0.91) + (0.81/0.91)U_1(\text{Full})\]
\[\Rightarrow U_1(\text{Hungry}) = -10.9 + (0.89)U_1(\text{Full})\]
Policy Iteration Example

\[(0.91)U_1(\text{Hungry})-(0.81)U_1(\text{Full}) = -10 \quad \text{...(Equation 1)}\]
\[(0.28)U_1(\text{Full}) - (0.18)U_1(\text{Hungry})=10 \quad \text{...(Equation 2)}\]

\{ Solve for 
\[U_1(\text{Hungry}) \text{ and } U_1(\text{Full}) \]

Substitute \[U_1(\text{Hungry}) = -10.9 + (0.89)U_1(\text{Full}) \] into Equation 2

\[(0.28)U_1(\text{Full}) - (0.18)[-10.9+(0.89)U_1(\text{Full})]=10\]
\[=>(0.28)U_1(\text{Full}) + 1.96-(0.16)U_1(\text{Full})=10\]
\[=>(0.12)U_1(\text{Full})=8.04\]
\[=>U_1(\text{Full})=67\]

\[=>U_1(\text{Hungry})=-10.9+(0.89)(67)=-10.9+59.63=48.7\]
Policy Iteration Example

$$\pi_2(\text{Hungry})$$

$$= \arg\max_{\{\text{Eat}, \text{Watch TV}\}} \begin{cases} T(\text{Hungry, Eat, Full})U_1(\text{Full}) + \\ T(\text{Hungry, Eat, Hungry})U_1(\text{Hungry}) \\ T(\text{Hungry, Watch TV, Hungry})U_1(\text{Hungry}) \end{cases}$$

$$= \arg\max_{\{\text{Eat}, \text{Watch TV}\}} \begin{cases} (0.9)U_1(\text{Full}) + (0.1)U_1(\text{Hungry}) \\ (1.0)U_1(\text{Hungry}) \end{cases}$$

$$= \arg\max_{\{\text{Eat}, \text{Watch TV}\}} \begin{cases} (0.9)(67) + (0.1)(48.7) \\ (1.0)(48.7) \end{cases}$$

$$= \arg\max_{\{\text{Eat}, \text{Watch TV}\}} \begin{cases} 65.2 \\ 48.7 \end{cases}$$

$$= \text{Eat}$$
Policy Iteration Example

\[ \pi_2(\text{Full}) \]

\[
= \arg\max_{\{\text{Exercise, Sleep}\}} \left\{ T(\text{Full, Exercise, Hungry}) U_1(\text{Hungry}) + T(\text{Full, Sleep, Full}) U_1(\text{Full}) + T(\text{Full, Sleep, Hungry}) U_1(\text{Hungry}) \right\}
\]

\[
= \arg\max_{\{\text{Exercise, Sleep}\}} \left\{ (1.0) U_1(\text{Hungry}) + (0.8) U_1(\text{Full}) + (0.2) U_1(\text{Hungry}) \right\}
\]

\[
= \arg\max_{\{\text{Exercise, Sleep}\}} \left\{ (1.0)(48.7) + (0.8)(67) + (0.2)(48.7) \right\}
\]

\[
= \arg\max_{\{\text{Exercise, Sleep}\}} \left\{ 48.7 + 53.6 + 9.74 \right\}
\]

\[= \text{Sleep} \]
Policy Iteration Example

- $\pi_2(\text{Hungry}) = \text{Eat}$
- $\pi_2(\text{Full}) = \text{Sleep}$
Comparison

• Which would you prefer, policy or value iteration?

• Depends...
  – If you have lots of actions in each state: policy iteration
  – If you have few actions in each state: value iteration
  – If you have a pretty good policy to start with: policy iteration
Limitations

• Need to represent the utility (and policy) for every state
• In real problems, the number of states may be very large
• Leads to intractably large tables
• Need to find compact ways to represent the states eg
  – Function approximation
  – Hierarchical representations
  – Memory-based representations