Solving the Bellman Optimality Equations: Basic Methods

AI & Agents for IET
Lecturer: Liliana Mamani Sanchez

http://www.scss.tcd.ie/~mamanisl/teaching/cs7032/

November 30, 2015
The basic background & assumptions

▶ Environment is a finite MDP (i.e. $A$ and $S$ are finite).
▶ MDP’s dynamics defined by transition probabilities:

$$P_{a\text{ss}'} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

▶ and expected immediate rewards,

$$R_{a\text{ss}'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

▶ Goal: to search for good policies $\pi$
▶ Strategy: use value functions to structure search:
The basic background & assumptions

- Environment is a finite MDP (i.e. $A$ and $S$ are finite).
- MDP's dynamics defined by transition probabilities:
  $$P_{ss'}^a = P(s_{t+1} = s'|s_t = s, a_t = a)$$
- and expected immediate rewards,

- Goal: to search for good policies $\pi$
- Strategy: use value functions to structure search:
The basic background & assumptions

- Environment is a finite MDP (i.e. $A$ and $S$ are finite).
- MDP's dynamics defined by transition probabilities:
  \[ \mathcal{P}_{ss'}^a = P(s_{t+1} = s' | s_t = s, a_t = a) \]
- and expected immediate rewards,
  \[ \mathcal{R}_{ss'}^a = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \]
- Goal: to search for good policies $\pi$
- Strategy: use value functions to structure search:
The basic background & assumptions

▶ Environment is a finite MDP (i.e. $A$ and $S$ are finite).
▶ MDP’s dynamics defined by transition probabilities:

$$P_{ss'}^a = P(s_{t+1} = s'|s_t = s, a_t = a)$$

▶ and expected immediate rewards,

$$R_{ss'}^a = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}$$

▶ Goal: to search for good policies $\pi$
▶ Strategy: use value functions to structure search:

$$V^*(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^*(s')]$$ \quad \text{or} \quad

$$Q^*(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma \max_{a'} Q^*(s', a')]$$
Overview of methods for solving Bellman’s equations

- Dynamic programming:
  - well-understood mathematical properties...
  - but require a complete and accurate model of the environment

- Monte Carlo (simulation methods):
  - conceptually simple
  - no model required...
  - but unsuitable for incremental computation

- Temporal difference methods
  - also require no model;
  - suitable for incremental computation...
  - but mathematically complex to analyse
Overview of methods for solving Bellman’s equations

▶ Dynamic programming:
  ▶ well-understood mathematical properties...
  ▶ ...but require a complete and accurate model of the environment

▶ Monte Carlo (simulation methods):
  ▶ conceptually simple
  ▶ no model required...
  ▶ ...but unsuitable for incremental computation
Overview of methods for solving Bellman’s equations

- **Dynamic programming:**
  - well-understood mathematical properties...
  - ...but require a complete and accurate model of the environment

- **Monte Carlo (simulation methods):**
  - conceptually simple
  - no model required...
  - ...but unsuitable for incremental computation

- **Temporal difference methods**
  - also require no model;
  - suitable for incremental computation...
  - ... but mathematically complex to analyse
Dynamic programming

- Basic Idea: “sweep” through $S$ performing a full backup operation on each $s$.
- A few different methods exist. E.g.:
  - Policy Iteration and
  - Value Iteration.
- The building blocks:
  - Policy Evaluation: how to compute $V^\pi$ for an arbitrary $\pi$.
  - Policy Improvement: how to compute an improved $\pi$ given $V^\pi$. 
The task of computing $V^\pi$ for an arbitrary $\pi$ is known as the prediction problem.

As we have seen, a state-value function is given by

$$V^\pi(s) = E_\pi \{ R_t | s_t = s \} = E_\pi \{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \} = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [ R_{ss'}^a + \gamma V^\pi(s') ]$$

A system of $|S|$ linear equations in $|S|$ unknowns (the state values $V^\pi(s)$)
Iterative Policy evaluation

Consider the sequence of approximations $V_0, \ldots V^\pi$.

Choose $V_0$ arbitrarily and set each successive approximation accommodation to the Bellman equation:

$$V_{k+1}(s) \leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1})|s_t = s\}$$

$$\leftarrow \sum a \pi(s, a) \sum s' P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \quad (1)$$

“Sweeps”:

$V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \ldots \rightarrow V^\pi$

$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow$
Iterative Policy evaluation

- Consider the sequence of approximations $V_0, \ldots V^\pi$.
- Choose $V_0$ arbitrarily and set each successive approximation accommodation to the Bellman equation:

$$V_{k+1}(s) \leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1})|s_t = s\}$$
$$\leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$$  \hspace{1cm} (1)

- “Sweeps”:

$$V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \ldots \rightarrow V^\pi$$

\[\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \]
Iterative Policy evaluation

- Consider the sequence of approximations $V_0, \ldots, V^\pi$.
- Choose $V_0$ arbitrarily and set each successive approximation accommodation to the Bellman equation:

$$V_{k+1}(s) \leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\}$$
$$\leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$$

(1)

- “Sweeps”:
  $$V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \ldots \rightarrow V^\pi$$
  \[\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \]
Iterative Policy evaluation

Consider the sequence of approximations $V_0, \ldots V^\pi$.

Choose $V_0$ arbitrarily and set each successive approximation accommodation to the Bellman equation:

$$
V_{k+1}(s) \leftarrow E_\pi \{ r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s \} \\
\leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] 
$$

“Sweeps”:

$V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \ldots \rightarrow V^\pi$

↑ ↑ ↑ ↑ ↑ ↑
Iterative Policy evaluation

- Consider the sequence of approximations $V_0, \ldots V^\pi$.
- Choose $V_0$ arbitrarily and set each successive approximation accommodation to the Bellman equation:

$$V_{k+1}(s) \leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1})|s_t = s\}$$

$$\leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \quad (1)$$

- “Sweeps”:

$$V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \ldots \rightarrow V^\pi$$
Iterative Policy evaluation

- Consider the sequence of approximations $V_0, \ldots V^\pi$.
- Choose $V_0$ arbitrarily and set each successive approximation accommodation to the Bellman equation:

$$V_{k+1}(s) \leftarrow E_\pi \{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\}$$
$$\leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$$

(1)

- “Sweeps”:

\[
\begin{array}{cccccccc}
V_0 & \rightarrow & V_1 & \rightarrow & \ldots & \rightarrow & V_k & \rightarrow & V_{k+1} & \ldots & \rightarrow & V^\pi \\
\uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow &
\end{array}
\]
Iterative Policy Evaluation Algorithm

Initialisation:

for (each \( s \in S \))
    \( V(s) \leftarrow 0 \)

\[ \text{IPE}(\pi) \] /* \( \pi \): policy to be evaluated */

repeat
    \( \Delta \leftarrow 0 \)
    \( V_k \leftarrow V \)
    for (each \( s \in S/\{s_{terminal}\} \))
        \[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \]
        \( \Delta \leftarrow \max(\Delta, |V_k(s) - V(s)|) \)
    until \( \Delta < \theta \) /* \( \theta > 0 \): a small constant */
    return \( V \) /* \( V \approx V^\pi \) */

- NB: alternatively one could evaluate \( V^\pi \) in place (i.e using a single vector \( V \) to store all values and update it directly).
An example: an episodic GridWorld

- Rewards of $-1$ until terminal state (shown in grey) is reached
- Undiscounted episodic task:

```
1 2 3
4 5 6 7
8 9 10 11
12 13 14
```

$r = -1$
on all transitions
Policy Evaluation for the GridWorld

- Iterative evaluation of $V_k$ for equiprobable random policy $\pi$:
Policy Evaluation for the GridWorld

Iterative evaluation of $V_k$ for equiprobable random policy $\pi$:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

$V_0 \rightarrow$
Policy Evaluation for the GridWorld

Iterative evaluation of $V_k$ for equiprobable random policy $\pi$:
Policy Evaluation for the GridWorld

- Iterative evaluation of $V_k$ for equiprobable random policy $\pi$:

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_0$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>0.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td></td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td></td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td></td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>-1.7</th>
<th>-2.0</th>
<th>-2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_2$</td>
<td>0.0</td>
<td>-1.7</td>
<td>-2.0</td>
<td>-2.0</td>
</tr>
<tr>
<td></td>
<td>-2.0</td>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
<td>0.0</td>
</tr>
</tbody>
</table>

$V_0 \rightarrow V_1 \rightarrow V_2 \rightarrow V_{\infty}$
Policy Evaluation for the GridWorld

Iterative evaluation of $V_k$ for equiprobable random policy $\pi$:

- $V_0 \rightarrow$

- $V_1 \rightarrow$

- $V_2 \rightarrow$

- $V_3 \rightarrow$

- $V_\infty \rightarrow$
Policy Evaluation for the GridWorld

Iterative evaluation of $V_k$ for equiprobable random policy $\pi$:

- $V_0 \rightarrow$

- $V_1 \rightarrow$

- $V_2 \rightarrow$

- $V_3 \rightarrow$

- $V_{10} \rightarrow$

- $V_{\infty} \rightarrow$
## Policy Evaluation for the GridWorld

- **Iterative evaluation of** $V_k$ **for equiprobable random policy** $\pi$:

<table>
<thead>
<tr>
<th>$V_0$</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$V_3$</th>
<th>$V_{10}$</th>
<th>$V_{\infty}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>-1.0</td>
<td>-2.0</td>
<td>-1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-6.1</th>
<th>-8.4</th>
<th>-9.0</th>
<th>0.0</th>
<th>-14.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6.1</td>
<td>-7.7</td>
<td>-8.4</td>
<td>-8.4</td>
<td>-14.</td>
<td>-20.</td>
</tr>
<tr>
<td>-8.4</td>
<td>-8.4</td>
<td>-7.7</td>
<td>-6.1</td>
<td>-20.</td>
<td>-20.</td>
</tr>
<tr>
<td>-9.0</td>
<td>-8.4</td>
<td>-6.1</td>
<td>0.0</td>
<td>-22.</td>
<td>-14.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.7</th>
<th>-2.0</th>
<th>-2.0</th>
<th>-2.0</th>
<th>-14.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
<td>0.0</td>
<td>-22.</td>
<td>-14.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-6.1</th>
<th>-8.4</th>
<th>-9.0</th>
<th>0.0</th>
<th>-14.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6.1</td>
<td>-7.7</td>
<td>-8.4</td>
<td>-8.4</td>
<td>-14.</td>
<td>-20.</td>
</tr>
<tr>
<td>-8.4</td>
<td>-8.4</td>
<td>-7.7</td>
<td>-6.1</td>
<td>-20.</td>
<td>-20.</td>
</tr>
<tr>
<td>-9.0</td>
<td>-8.4</td>
<td>-6.1</td>
<td>0.0</td>
<td>-22.</td>
<td>-14.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.7</th>
<th>-2.0</th>
<th>-2.0</th>
<th>-2.0</th>
<th>-22.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
<td>0.0</td>
<td>-22.</td>
<td>-18.</td>
</tr>
</tbody>
</table>
Consider the following: how would the expected return change for a policy $\pi$ if instead of following $\pi(s)$ for a given state $s$ we choose an action $a \neq \pi(s)$?

For this setting, the value would be:
Consider the following: how would the expected return change for a policy \( \pi \) if instead of following \( \pi(s) \) for a given state \( s \) we choose an action \( a \neq \pi(s) \)?

For this setting, the value would be:

\[
Q^\pi(s, a) = E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a\} \\
= \sum_{s'} \mathcal{P}_a^{ss'} [R_a^{ss'} + \gamma V^\pi(s_{t+1})]
\]

So, \( a \) should be preferred iff \( Q^\pi(s, a) > V^\pi(s) \)
If choosing \( a \neq \pi(s) \) implies \( Q^\pi(s, a) \geq V^\pi(s) \) for a state \( s \), then the policy \( \pi' \) obtained by choosing \( a \) every time \( s \) is encountered (and following \( \pi \) otherwise) is at least as good as \( \pi \) (i.e. \( V^{\pi'}(s) \geq V^{\pi}(s) \)). If \( Q^\pi(s, a) > V^\pi(s) \) then \( V^{\pi'}(s) > V^{\pi}(s) \)

- If we apply this strategy to all states to get a new greedy policy \( \pi'(s) = \arg\max_a Q^\pi(s, a) \), then \( V^{\pi'} \geq V^{\pi} \)
- \( V^{\pi'} = V^{\pi} \) implies that

\[
V^{\pi'}(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')] \]

which is...
Policy improvement theorem

If choosing \( a \neq \pi(s) \) implies \( Q^\pi(s, a) \geq V^\pi(s) \) for a state \( s \), then the policy \( \pi' \) obtained by choosing \( a \) every time \( s \) is encoutered (and following \( \pi \) otherwise) is at least as good as \( \pi \) (i.e. \( V^\pi'(s) \geq V^\pi(s) \)).

If \( Q^\pi(s, a) > V^\pi(s) \) then \( V^\pi'(s) > V^\pi(s) \)

- If we apply this strategy to all states to get a new greedy policy \( \pi'(s) = \arg \max_a Q^\pi(s, a) \), then \( V^\pi' \geq V^\pi \)

- \( V^\pi' = V^\pi \) implies that

\[
V^\pi'(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] 
\]

which is... a form of the Bellman optimality equation.
Policy improvement theorem

If choosing \( a \neq \pi(s) \) implies \( Q^\pi(s, a) \geq V^\pi(s) \) for a state \( s \), then the policy \( \pi' \) obtained by choosing \( a \) every time \( s \) is encountered (and following \( \pi \) otherwise) is at least as good as \( \pi \) (i.e. \( V^{\pi'}(s) \geq V^\pi(s) \)).

If \( Q^\pi(s, a) > V^\pi(s) \) then \( V^{\pi'}(s) > V^\pi(s) \)

- If we apply this strategy to all states to get a new greedy policy \( \pi'(s) = \text{arg max}_a Q^\pi(s, a) \), then \( V^{\pi'} \geq V^\pi \)
- \( V^{\pi'} = V^\pi \) implies that

\[
V^{\pi'}(s) = \max_a \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^\pi(s')] 
\]

which is... a form of the Bellman optimality equation.
- Therefore \( V^\pi = V^{\pi'} = V^* \)
Improving the GridWorld policy
Improving the GridWorld policy

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Improving the GridWorld policy
Improving the GridWorld policy

<table>
<thead>
<tr>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.7</th>
<th>-2.0</th>
<th>-2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.7</td>
<td>-2.0</td>
<td>-2.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
</tr>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.0</th>
<th>-2.4</th>
<th>-2.9</th>
<th>-3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.4</td>
<td>-2.9</td>
<td>-3.0</td>
<td>-2.9</td>
</tr>
<tr>
<td>-2.9</td>
<td>-3.0</td>
<td>-2.9</td>
<td>-2.4</td>
</tr>
<tr>
<td>-3.0</td>
<td>-2.9</td>
<td>-2.4</td>
<td>0.0</td>
</tr>
</tbody>
</table>
**Improving the GridWorld policy**

<table>
<thead>
<tr>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

→

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.7</th>
<th>-2.0</th>
<th>-2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.7</td>
<td>-2.0</td>
<td>-2.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
</tr>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
<td>0.0</td>
</tr>
</tbody>
</table>

→

<table>
<thead>
<tr>
<th>0.0</th>
<th>-6.1</th>
<th>-8.4</th>
<th>-9.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6.1</td>
<td>-7.7</td>
<td>-8.4</td>
<td>-8.4</td>
</tr>
<tr>
<td>-8.4</td>
<td>-8.4</td>
<td>-7.7</td>
<td>-6.1</td>
</tr>
<tr>
<td>-9.0</td>
<td>-8.4</td>
<td>-6.1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

→

<table>
<thead>
<tr>
<th>0.0</th>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

→

<table>
<thead>
<tr>
<th>0.0</th>
<th>-2.4</th>
<th>-2.9</th>
<th>-3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.4</td>
<td>-2.9</td>
<td>-3.0</td>
<td>-2.9</td>
</tr>
<tr>
<td>-2.9</td>
<td>-3.0</td>
<td>-2.9</td>
<td>-2.4</td>
</tr>
<tr>
<td>-3.0</td>
<td>-2.9</td>
<td>-2.4</td>
<td>0.0</td>
</tr>
</tbody>
</table>

→

<table>
<thead>
<tr>
<th>0.0</th>
<th>-14.</th>
<th>-20.</th>
<th>-22.</th>
</tr>
</thead>
</table>
Improving the GridWorld policy

0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0

0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0

0.0 -6.1 -8.4 -9.0
-6.1 -7.7 -8.4 -8.4
-8.4 -8.4 -7.7 -6.1
-9.0 -8.4 -6.1 0.0

0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
-1.0 -1.0 -1.0 0.0

0.0 -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 0.0

0.0 -14.0 -20.0 -22.0
-14.0 -18.0 -20.0 -20.0
-20.0 -20.0 -18.0 -14.0
-22.0 -20.0 -14.0 0.0
Putting them together: Policy Iteration

\[ \pi_0 \]
Putting them together: Policy Iteration

\[ \pi_0 \overset{eval}{\rightarrow} \]

1. **Initialization:**
   - For all \( s \in S \),
     
     \[ V(s) \leftarrow \text{arbitrary } v \in \mathbb{R} \]

2. **Policy Improvement** (\( \pi \)):
   - do
     - Stable (\( \pi \)) \( \leftarrow \) true
     - \( V \leftarrow \text{IPE}(\pi) \)
     - for each \( s \in S \),
       - \( b \leftarrow \pi(s) \)
       - \( \pi(s) \leftarrow \text{arg max}_a \sum_{s'} \mathbb{P}(ss') \left[ R_{ss'} + \gamma V(s') \right] \)
       - if (\( b \neq \pi(s) \))
         - Stable (\( \pi \)) \( \leftarrow \) false
   - while (not Stable (\( \pi \)))
   - return \( \pi \)
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \]

1. Initialization:
   \[ \text{for all } s \in S \]
   \[ V(s) \leftarrow \text{an arbitrary } v \in \mathbb{R} \]

2. Policy Improvement (\( \pi \)):
   \[ \text{do} \]
   \[ \text{stable}(\pi) \leftarrow \text{true} \]
   \[ V \leftarrow \text{IPE}(\pi) \]
   \[ \text{for each } s \in S \]
   \[ b \leftarrow \pi(s) \]
   \[ \pi(s) \leftarrow \text{arg max}_a \sum_{s' \in S} P_{ss'} [R_{ss'} + \gamma V(s')] \]
   \[ \text{if } (b \neq \pi(s)) \]
   \[ \text{stable}(\pi) \leftarrow \text{false} \]
   \[ \text{while } (\neg \text{stable}(\pi)) \]
   \[ \text{return } \pi \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e} \ldots \]

**Initialization:**
\[ \forall s \in S \]
\[ V(s) \leftarrow \text{arbitrary } v \in \mathbb{R} \]

**Policy Improvement (\( \pi \)):**
\[ \text{do} \]
\[ \text{stable}(\pi) \leftarrow \text{true} \]
\[ V \leftarrow \text{IPE}(\pi) \]
\[ \text{for each } s \in S \]
\[ b \leftarrow \pi(s) \]
\[ \pi(s) \leftarrow \text{arg max}_a \sum_{s'} P_{as} \left[ R_{as} + \gamma V(s') \right] \]
\[ \text{if } (b \neq \pi(s)) \]
\[ \text{stable}(\pi) \leftarrow \text{false} \]
\[ \text{while } (\text{not stable}(\pi)) \]
\[ \text{return } \pi \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e} V_{\pi_1} \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \cdots \xrightarrow{i} \]

Initialisation:

For all \( s \in S \):

\[ V(s) \leftarrow \text{arbitrary } v \in \mathbb{R} \]

Policy Improvement (\( \pi \)):

Do

- stable (\( \pi \)) \leftarrow true
- \( V \leftarrow \text{IPE (\( \pi \))} \)
- for each \( s \in S \):
  - \( b \leftarrow \pi(s) \)
  - \( \pi(s) \leftarrow \text{arg max}_a \sum_{s'} P_{ss'} [R_{sa} + \gamma V(s')] \)
- if \( b \neq \pi(s) \)
  - stable (\( \pi \)) \leftarrow false

While (not stable (\( \pi \)))

Return \( \pi \)

13
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{\text{eval}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \cdots \xrightarrow{i} \pi^* \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \ldots \xrightarrow{i} \pi^* \xrightarrow{e} \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{\text{eval}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \cdots \xrightarrow{i} \pi^* \xrightarrow{e} V^* \]
Putting them together: Policy Iteration

\[ \pi_0 \xrightarrow{\text{eval}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \cdots \xrightarrow{i} \pi^* \xrightarrow{e} V^* \]

---

**Initialisation:**

for all \( s \in S \)

\[ V(s) \leftarrow \text{an arbitrary } v \in \mathbb{R} \]

**Policy Improvement (\( \pi \)):**

\[
\begin{align*}
&\text{do} \\
&\quad \text{stable}(\pi) \leftarrow \text{true} \\
&\quad V \leftarrow \text{IPE}(\pi) \\
&\quad \text{for each } s \in S \\
&\quad \quad b \leftarrow \pi(s) \\
&\quad \quad \pi(s) \leftarrow \arg \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \\
&\quad \quad \text{if } (b \neq \pi(s)) \\
&\quad \quad \quad \text{stable}(\pi) \leftarrow \text{false} \\
&\quad \text{while } (\text{not stable}(\pi)) \\
&\quad \text{return } \pi
\end{align*}
\]
Other DP methods

- **Value Iteration**: evaluation is stopped after a single sweep (one backup of each state). The backup rule is then:

  \[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \]

- **Asynchronous DP**: back up the values of states in any order, using whatever values of other states happen to be available.
  - On problems with large state spaces, asynchronous DP methods are often preferred.
Other DP methods

- **Value Iteration**: evaluation is stopped after a single sweep (one backup of each state). The backup rule is then:

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \]

- **Asynchronous DP**: back up the values of states in any order, using whatever values of other states happen to be available.
  - On problems with large state spaces, asynchronous DP methods are often preferred

---

**Generalised policy iteration**

\[ \pi \rightarrow \text{greedy}(V) \]

\[ V \rightarrow V^\pi \]

\[ \pi^* \leftrightarrow V^* \]
Value Iteration

- Potential computational savings over Policy Iteration in terms of policy evaluation

```
1  \textbf{Initialisation:}
2  \textbf{for} all \( s \in S \)
3  \hspace{1em} V(s) \leftarrow \text{an arbitrary } v \in \mathbb{R}

4  \textbf{Value Iteration (}\pi\text{):}
5  \textbf{repeat}
6  \hspace{1em} \Delta \leftarrow 0
7  \textbf{for each } s \in S
8  \hspace{2em} v \leftarrow V(s)
9  \hspace{2em} V(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$
10 \hspace{2em} \Delta \leftarrow \max(\Delta, |v - V(s)|)
11 \textbf{until } \Delta < \theta$
12 \textbf{return deterministic } \pi \text{ s.t.}
13 \hspace{1em} \pi(s) = \arg \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$
```
Summary of methods

- Full backups
- Sample backups
- Shallow backups
- Deep backups
- Bootstrapping, $\lambda$

Exhaustive search
Dynamic programming
Summary of methods

- Exhaustive search
- Dynamic programming

Methods:
- Bootstrapping, $\lambda$
- Full backups
- Sample backups
- Shallow backups
- Deep backups
Summary of methods

Dynamic programming

Exhaustive search

Full backups

Sample backups

Shallow backups

Bootstrapping, $\lambda$

Deep backups

Dynamic programming

Exhaustive search

Full backups

Sample backups

Shallow backups

Bootstrapping, $\lambda$

Deep backups
Monte Carlo Methods

- Complete knowledge of environment is not necessary
- Only experience is required
- Learning can be on-line (no model needed) or through simulated experience (model only needs to generate sample transitions.
  - In both cases, learning is based on averaged sample returns.
- As in DP, one can use an evaluation-improvement strategy.
- Evaluation can be done by keeping averages of:
  - Every Visit to a state in an episode, or
  - of the First Visit to a state in an episode.
Estimating value-state functions in MC

The first visit policy evaluation method:

```
1 FirstVisitMC(π)
2 Initialisation:
3    V ← arbitrary state values
4    Returns(s) ← empty list of size |S|
5
6 Repeat
7    Generate an episode E using π
8    For each s in E
9        R ← return following the first occurrence of s
10       Append R to Returns(s)
11       V(s) ← mean(Returns(s))
```
Example

Evaluate the policy described below for blackjack
[Sutton and Barto, 1998, section 5.1]

- Actions: stick (stop receiving cards), hit (receive another card)
- Play against dealer, who has a fixed strategy (‘hit’ if sum < 17; ‘stick’ otherwise).
- You win if your card sum is greater than the dealer’s without exceeding 21.
- States:
  - current sum (12-21)
  - dealers showing card (ace-10)
  - do I have a useable ace (can be 11 without making sum exceed 21)?
- Reward: +1 for winning, 0 for a draw, -1 for losing
- Policy: Stick if my sum is 20 or 21, else hit
Question: compare MC to DP in estimating the value function. Do we know the environment? The transition probabilities? The expected returns given each state and action? What does the MC backup diagram look like?
Monte Carlo control

- Monte Carlo version of DP policy iteration:

Policy improvement theorem applies:

\[ Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, \arg \max_a Q^{\pi_k}(s, a)) \]
\[ = \max_a Q^{\pi_k}(s, a) \]
\[ \geq Q^{\pi_k}(s, a) \]
\[ = V^{\pi_k}(s) \]
MC policy iteration (exploring starts)

- As with DP, we have evaluation-improvement cycles.
- Learn $Q^*$ (if no model is available)
- One must make sure that each state-action pair can be a starting pair (with probability $> 0$).

---

```plaintext
MonteCarloES()

Initialisation, $\forall s \in S, a \in A$:
- $Q(s, a) \leftarrow \text{arbitrary}$; $\pi(s) \leftarrow \text{arbitrary}$
- $Returns(s, a) \leftarrow \text{empty list of size } |S|$

Repeat until stop criterion met
- Generate an episode $E$ using $\pi$ and exploiting starts
- For each $(s, a)$ in $E$
  - $R \leftarrow \text{return following the first occurrence of } s, a$
  - Append $R$ to $Returns(s, a)$
  - $Q(s, a) \leftarrow \text{mean}(Returns(s, a))$
- For each $s$ in $E$
  - $\pi(s) \leftarrow \text{arg max}_a Q(s, a)$
```
Optimal policy for blackjack example

- Optimal policy found by MonteCarloES for the blackjack example, and its state-value function:
On- and off-policy MC control

- MonteCarloES assumes that all states are observed an infinite number of times and episodes are generated with exploring starts
  - For an analysis of convergence properties, see [Tsitsiklis, 2003]
- On-policy and off-policy methods relax these assumptions to produce practical algorithms
- On-policy methods use a given policy and $\epsilon$-greedy strategy (see lecture on Evaluative Feeback) to generate episodes.
- Off-policy methods evaluate a policy while generating an episode through a different policy
On-policy control

Initialize, for all \( s \in S, a \in A(s) \):
\[
Q(s, a) \leftarrow \text{arbitrary} \\
\text{Returns}(s, a) \leftarrow \text{empty list} \\
\pi \leftarrow \text{an arbitrary } \varepsilon\text{-soft policy}
\]

Repeat forever:
(a) Generate an episode using \( \pi \)
(b) For each pair \( s, a \) appearing in the episode:
   \( R \leftarrow \text{return following the first occurrence of } s, a \)
   Append \( R \) to \( \text{Returns}(s, a) \)
   \( Q(s, a) \leftarrow \text{average}(\text{Returns}(s, a)) \)
(c) For each \( s \) in the episode:
   \( a^* \leftarrow \text{arg max}_a Q(s, a) \)
   For all \( a \in A(s) \):
   \[
   \pi(s, a) \leftarrow \begin{cases} 
   1 - \varepsilon + \varepsilon/|A(s)| & \text{if } a = a^* \\
   \varepsilon/|A(s)| & \text{if } a \neq a^*
   \end{cases}
   \]
Summary of methods

- Bootstrapping, $\lambda$
- Full backups
- Sample backups
- Shallow backups
- Deep backups

- Exhaustive search
- Dynamic programming
- Monte Carlo
Summary of methods

Exhaustive search

Dynamic programming

Monte Carlo
Summary of methods

- Dynamic programming
- Exhaustive search
- Monte Carlo
Summary of methods

- **Dynamic programming**
- **Exhaustive search**
- **Monte Carlo**

**Methods with full backups:**
- Bootstrapping, $\lambda$
- Full backups

**Methods with sample backups:**
- Bootstrapping, $\lambda$
- Sample backups

**Methods with shallow backups:**
- Shallow backups

**Methods with deep backups:**
- Deep backups
Notes based on [Sutton and Barto, 1998, ch 4-6]. Convergence results for several MC algorithms are given by [Tsitsiklis, 2003].

- **Sutton, R. S. and Barto, A. G. (1998).**
  *Reinforcement Learning: An Introduction.*
  MIT Press, Cambridge, MA.

- **Tsitsiklis, J. N. (2003).**
  On the convergence of optimistic policy iteration.