Solving the Bellman Optimality Equations: Basic Methods

AI & Agents for IET
Lecturer: S Luz

http://www.scss.tcd.ie/~luzs/t/cs7032/

December 3, 2014

The basic background & assumptions

- Environment is a finite MDP (i.e. A and S are finite).
- MDP's dynamics defined by transition probabilities:
- and expected immediate rewards,
- Goal: to search for good policies π
- 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,

- Goal: to search for good policies π
- 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:

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

- and expected immediate rewards,

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

- Goal: to search for good policies π
- 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:

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

- and expected immediate rewards,

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

- Goal: to search for good policies π
- Strategy: use value functions to structure search:

$$\begin{aligned} V^*(s) &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V * (s')] \quad \text{or} \\ Q^*(s,a) &= \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')] \end{aligned}$$

Overview of methods for solving Bellman's equations

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

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^{π} for an arbitrary π.
  - Policy Improvement: how to compute an improved π given V^{π}.

Policy Evaluation

- The task of computing V^π for an arbitrary π is known as the prediction problem.
- As we have seen, a state-value function is given by

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} = E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

- a system of $|S|$ linear equations in $|S|$ unknowns (the state values $V^\pi(s)$)

Iterative Policy evaluation

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

$$\begin{aligned} V_{k+1}(s) &\leftarrow E_{\pi}\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\ &\leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')] \end{aligned} \tag{1}$$

- "Sweeps":

$$\begin{array}{ccccccccccccc} V_0 & \rightarrow & V_1 & \rightarrow & \dots & \rightarrow & V_k & \rightarrow & V_{k+1} & \dots & \rightarrow & V^{\pi} \\ & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow & \end{array}$$

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:

$$\begin{aligned} V_{k+1}(s) &\leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\ &\leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')] \end{aligned} \quad (1)$$

- "Sweeps":

$$\begin{array}{ccccccccccccc} V_0 & \rightarrow & V_1 & \rightarrow & \ldots & \rightarrow & V_k & \rightarrow & V_{k+1} & \ldots & \rightarrow & V^\pi \\ & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow & \end{array}$$

Iterative Policy evaluation

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

$$\begin{aligned} V_{k+1}(s) &\leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\ &\leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')] \end{aligned} \quad (1)$$

- "Sweeps":

$$\begin{array}{ccccccccccc} V_0 & \rightarrow & V_1 & \rightarrow & \dots & \rightarrow & V_k & \rightarrow & V_{k+1} & \dots & \rightarrow & V^\pi \\ & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow & \end{array}$$

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:

$$\begin{aligned} V_{k+1}(s) &\leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\ &\leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')] \end{aligned} \tag{1}$$

- "Sweeps":

$$\begin{array}{ccccccccccc} V_0 & \rightarrow & V_1 & \rightarrow & \ldots & \rightarrow & V_k & \rightarrow & V_{k+1} & \ldots & \rightarrow & V^\pi \\ & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow & \end{array}$$

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:

$$
\begin{aligned}
V_{k+1}(s) &\leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\
&\leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')] \qquad (1)
\end{aligned}
$$

- "Sweeps":

$$
\begin{array}{ccccccccccccc}
V_0 & \rightarrow & V_1 & \rightarrow & \ldots & \rightarrow & V_k & \rightarrow & V_{k+1} & \ldots & \rightarrow & V^\pi \\
 & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow &
\end{array}
$$

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:

$$\begin{aligned} V_{k+1}(s) &\leftarrow E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\} \\ &\leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')] \end{aligned} \quad (1)$$

- "Sweeps":

$$\begin{array}{ccccccccccc} V_0 & \rightarrow & V_1 & \rightarrow & \ldots & \rightarrow & V_k & \rightarrow & V_{k+1} & \ldots & \rightarrow & V^\pi \\ & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow & \end{array}$$

Iterative Policy Evaluation Algorithm

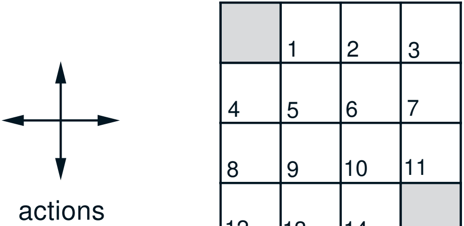
Initialisation:
  for (each $s \in S$)
    $V(s) \leftarrow 0$

IPE(π) /* π: policy to be evaluated */
  repeat
    $\Delta \leftarrow 0$
    $V_k \leftarrow V$
    for (each $s \in S / \{s_{terminal}\}$)
      $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')]$
      $\Delta \leftarrow \max(\Delta, |V_k(s) - V(s)|)$
  until $\Delta < \theta$ /* $\theta > 0$: a small constant */
  return V /* $V \approx V^\pi$ */

- NB: alternatively one could evaluate V^π *in place* (i.e usnig 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:



$r = -1$
on all transitions

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

$V_0 \rightarrow$

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

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

$V_0 \rightarrow$

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

$V_1 \rightarrow$

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

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

$V_0 \rightarrow$

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

$V_1 \rightarrow$

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

$V_2 \rightarrow$

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

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

$V_0 \rightarrow$

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

$V_1 \rightarrow$

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

$V_2 \rightarrow$

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

$V_3 \rightarrow$

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

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

$V_0 \rightarrow$

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

$V_1 \rightarrow$

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

$V_2 \rightarrow$

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

$V_3 \rightarrow$

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

$V_{10} \rightarrow$

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

Policy Evaluation for the GridWorld

- Iterative evaluation of V_k for equiprobable random policy π:

$V_0 \rightarrow$

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

$V_1 \rightarrow$

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

$V_2 \rightarrow$

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

$V_3 \rightarrow$

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

$V_{10} \rightarrow$

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

$V_\infty \rightarrow$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Improvement

- Consider the following: how would the expected return change for a policy π 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:

Policy Improvement

- Consider the following: how would the expected return change for a policy π 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:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a\} \\ &= \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s_{t+1})] \end{aligned}$$

- So, a should be preferred iff $Q^\pi(s,a) > V^\pi(s)$

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 π' obtained by choosing a every time s is encoutered (and following π otherwise) is at least as good as π (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

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

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 π' obtained by choosing a every time s is encoutered (and following π otherwise) is at least as good as π (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

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

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 π' obtained by choosing a every time s is encoutered (and following π otherwise) is at least as good as π (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

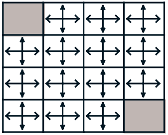
$$V^{\pi'}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

  which is... a form of the Bellman optimality equation.
- Therefore $V^\pi = V^{\pi'} = V^*$

Improving the GridWorld policy

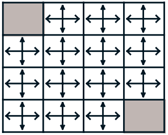
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$\longrightarrow$

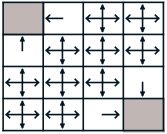


Improving the GridWorld policy

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



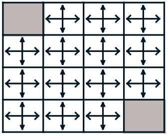
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



Improving the GridWorld policy

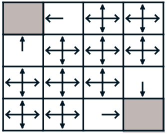
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$\longrightarrow$



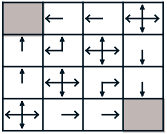
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$\longrightarrow$



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

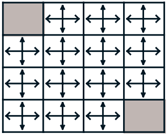
$\longrightarrow$



Improving the GridWorld policy

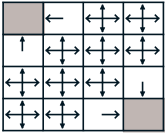
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$\longrightarrow$



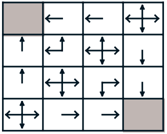
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$\longrightarrow$



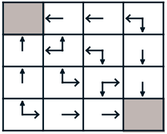
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$\longrightarrow$



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

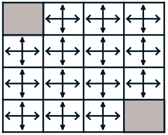
$\longrightarrow$



Improving the GridWorld policy

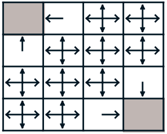
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

→



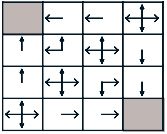
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

→



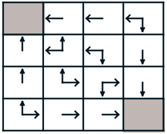
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

→



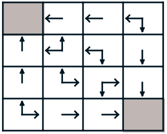
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

→



0.0	-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

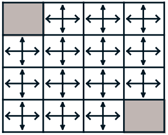
→



Improving the GridWorld policy

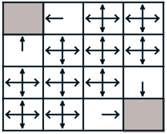
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$\rightarrow$



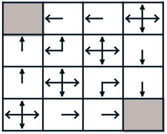
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$\rightarrow$



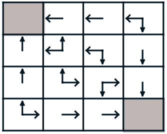
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$\rightarrow$



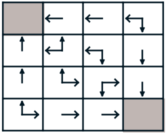
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$\rightarrow$



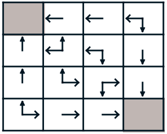
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$\rightarrow$



0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\rightarrow$



Putting them together: Policy Iteration

π_0

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval}$$

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval} V_{\pi_0}$$

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve}$$

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1$$

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e}$$

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

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

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \cdots \xrightarrow{i} \pi^*$$

Putting them together: Policy Iteration

$$\pi_0 \xrightarrow{eval} V_{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{i} \cdots \xrightarrow{i} \pi^* \xrightarrow{e}$$

Putting them together: Policy Iteration

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

Putting them together: Policy Iteration

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

```
Initialisation:
  for all s ∈ S
    V(s) ← an arbitrary v ∈ ℝ

Policy_Improvement(π):
  do
    stable(π) ← true
    V ← IPE(π)
    for each s ∈ S
      b ← π(s)
      π(s) ← arg max_a Σ_{s'} P^a_{ss'} [R^a_{ss'} + γV(s')]
      if ( b ≠ π(s))
        stable(π) ← false
  while (not stable(π))
  return π
```

Other DP methods

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

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- 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

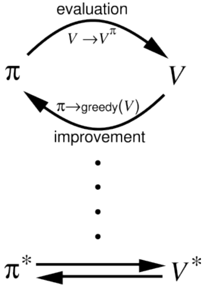
Other DP methods

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

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- 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

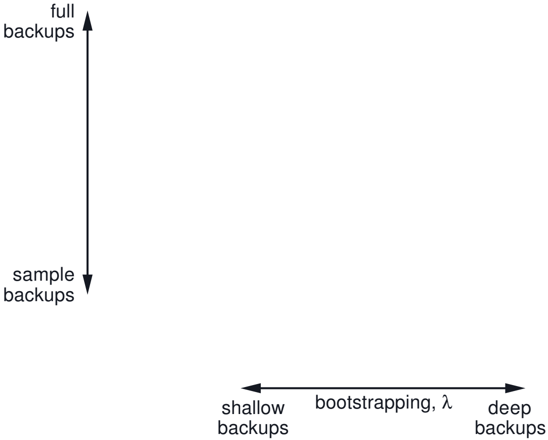


Value Iteration

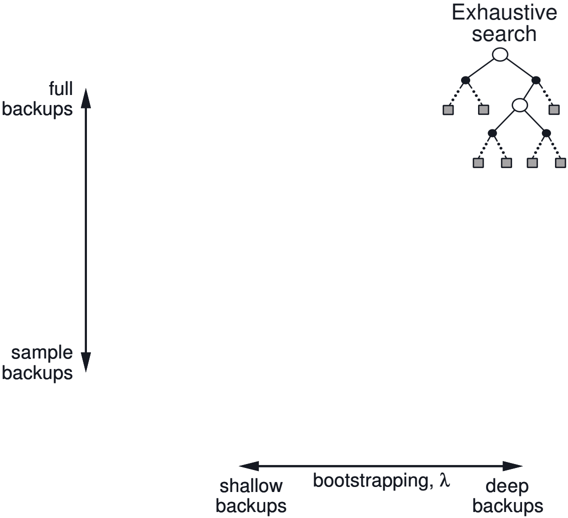
- Potential computational savings over Policy Iteration in terms of policy evaluation

```
Initialisation:
  for all s ∈ S
    V(s) ← an arbitrary v ∈ ℝ

Value Iteration(π):
  repeat
   Δ ← 0
   for each s ∈ S
     v ← V(s)
     V(s) ← max_a Σ_{s'} P^a_{ss'} [R^a_{ss'} + γ V(s')]
     Δ ← max(Δ, |v − V(s)|)
  until Δ < θ
  return deterministic π s.t.
     π(s) = arg max_a Σ_{s'} P^a_{ss'} [R^a_{ss'} + γ V(s')]
```

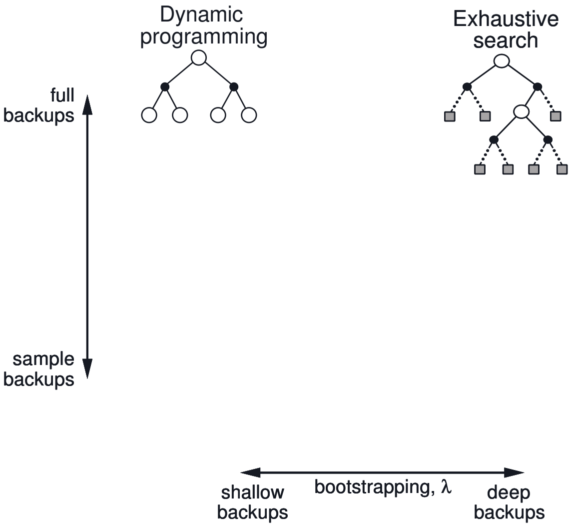
Summary of methods



Summary of methods



Summary of methods



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:

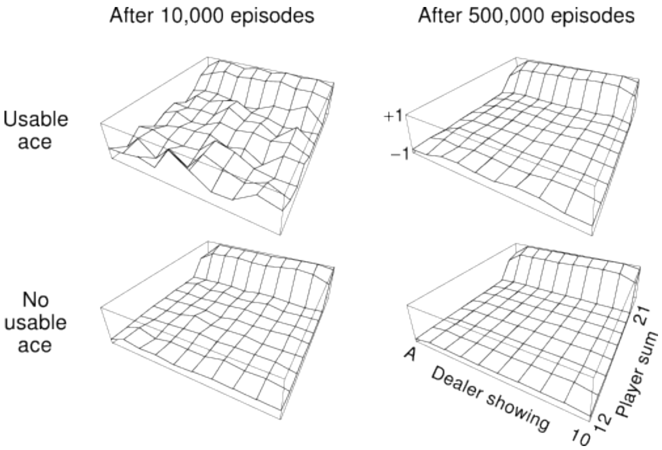
```
FirstVisitMC(π)
Initialisation:
  V ← arbitrary state values
  Returns(s) ← empty list of size |S|

Repeat
  Generate an episode E using π
  For each s in E
     R ← return following the fisrt occurrence of s
     Append R to Returns(s)
     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

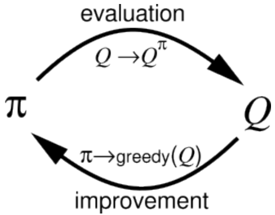
MC value function for blackjack example



- Question: compare MC to DP in estimating the value function. Do we know the 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:

$$
\begin{aligned}
Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s,a)) \\
&= \max_a Q^{\pi_k}(s,a) \\
&\geq Q^{\pi_k}(s,a) \\
&= V^{\pi_k}(s)
\end{aligned}
$$

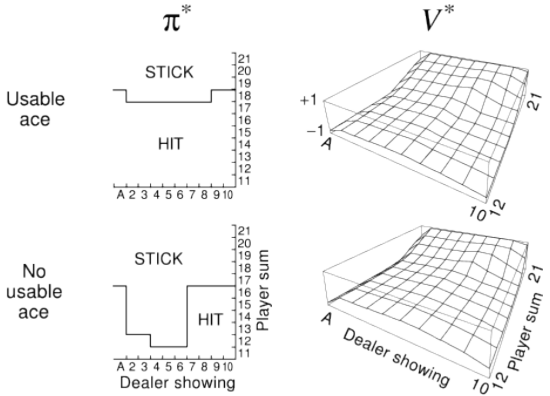
MC policy iteration (exploring starts)

- 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).

```
MonteCarloES()
Initialisation, ∀s ∈ S, a ∈ A:
  Q(s,a) ← arbitrary; π(s) ← arbitrary
  Returns(s,a) ← empty list of size |S|

Repeat until stop−criterion met
  Generate an episode E using π and exploting starts
  For each (s,a) in E
      R ← return following the fisrt occurrence of s,a
      Append R to Returns(s,a)
      Q(s,a) ← mean(Returns(s,a))
  For each s in E
      π(s) ← 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 ϵ-greedy strategy (see lecture on Evaluative Feeback) to generate episodes.
- Off-policy methods evaluate a policy while generating an episode through a different policy

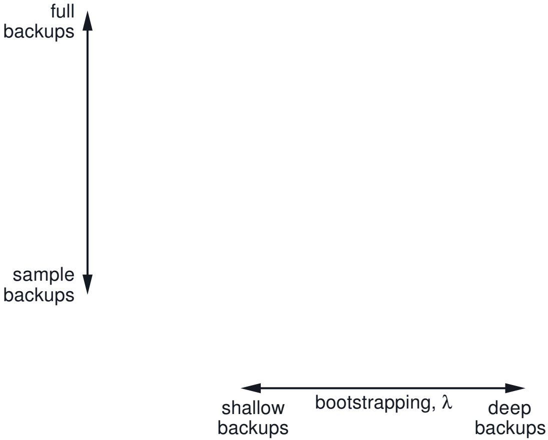
On-policy control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
  $\pi \leftarrow$ an arbitrary ε-soft policy

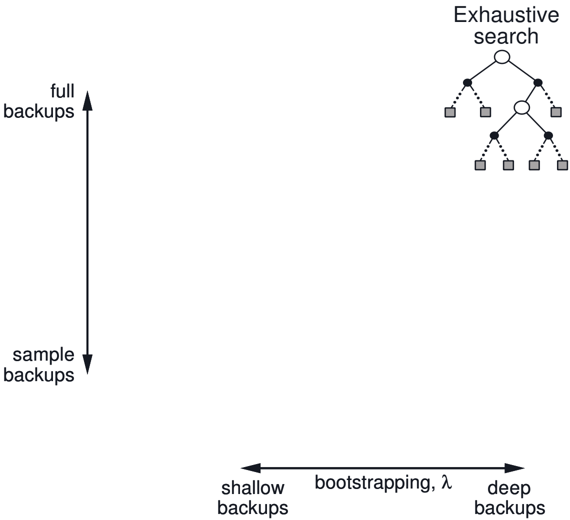
Repeat forever:
  (a) Generate an episode using π
  (b) For each pair s, a appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s, a
    Append R to $Returns(s,a)$
    $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
  (c) For each s in the episode:
    $a^* \leftarrow \arg\max_a Q(s,a)$
    For all $a \in \mathcal{A}(s)$:

$$\pi(s,a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

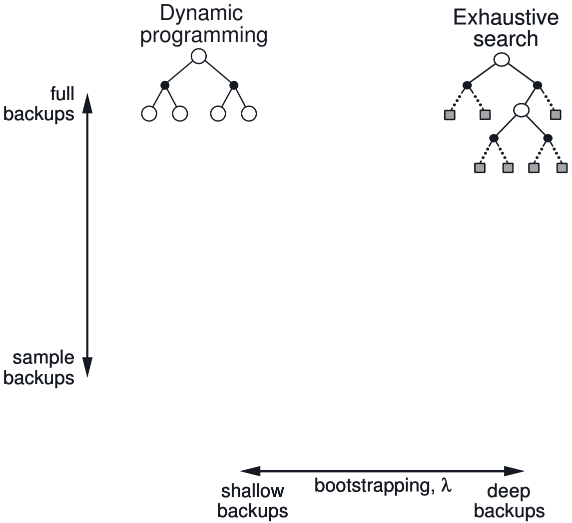
Summary of methods



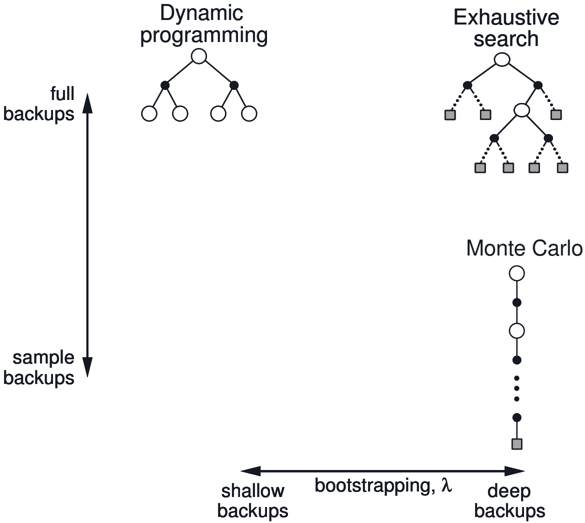
Summary of methods



Summary of methods



Summary of methods



References

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.
The Journal of Machine Learning Research, 3:59–72.