Lecture 03: Dynamic Programming

Davit Ghazaryan

February 11, 2025

American University of Armenia

Table of contents

1. Introduction
2. Policy evaluation (Prediction)
3. Policy improvement (Control)
4. Policy and value iteration
5. Further aspects

What is dynamic programming (DP)?

Basic DP definition

- Dynamic: sequential or temporal problem structure
- Programming: mathematical optimization, i.e., numerical solutions

What is dynamic programming (DP)?

Basic DP definition

- Dynamic: sequential or temporal problem structure
- Programming: mathematical optimization, i.e., numerical solutions

Further characteristics:

- DP is a collection of algorithms to solve MDPs and neighboring problems.
  - We will focus only on finite MDPs.
  - In case of continuous action/state space: apply quantization.

What is dynamic programming (DP)?

Basic DP definition

- Dynamic: sequential or temporal problem structure
- Programming: mathematical optimization, i.e., numerical solutions

Further characteristics:

- DP is a collection of algorithms to solve MDPs and neighboring problems.
  - We will focus only on finite MDPs.
  - In case of continuous action/state space: apply quantization.
- Use of value functions to organize and structure the search for an optimal policy.
- Breaks problems into subproblems and solves them.

Requirements for DP

DP can be applied to problems with the following characteristics.

- Optimal substructure:
  - Principle of optimality applies.
  - Optimal solution can be derived from subproblems.

Requirements for DP

DP can be applied to problems with the following characteristics.

- Optimal substructure:
  - Principle of optimality applies.
  - Optimal solution can be derived from subproblems.
- Overlapping subproblems:
  - Subproblems recur many times.
  - Hence, solutions can be cached and reused.

Requirements for DP

DP can be applied to problems with the following characteristics.

- Optimal substructure:
  - Principle of optimality applies.
  - Optimal solution can be derived from subproblems.
- Overlapping subproblems:
  - Subproblems recur many times.
  - Hence, solutions can be cached and reused.

How is that connected to MDPs?

- MDPs satisfy above's properties:
  - Bellman equation provides recursive decomposition.
  - Value function stores and reuses solutions.

Utility of DP in the RL context

DP is used for iterative model-based prediction and control in an MDP.

- Prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$ and policy π
  - Output: (estimated) value function $\hat{v}_\pi \approx v_\pi$

Utility of DP in the RL context

DP is used for iterative model-based prediction and control in an MDP.

- Prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$ and policy π
  - Output: (estimated) value function $\hat{v}_\pi \approx v_\pi$
- Control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$
  - Output: (estimated) optimal value function $\hat{v}_\pi^* \approx v_\pi^*$ or policy $\hat{\pi}^* \approx \pi^*$

Utility of DP in the RL context

DP is used for iterative model-based prediction and control in an MDP.

- Prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$ and policy π
  - Output: (estimated) value function $\hat{v}_\pi \approx v_\pi$
- Control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$
  - Output: (estimated) optimal value function $\hat{v}_\pi^* \approx v_\pi^*$ or policy $\hat{\pi}^* \approx \pi^*$

In both applications DP requires full knowledge of the MDP structure.

- Feasibility in real-world engineering applications (model vs. system) is therefore limited.

Utility of DP in the RL context

DP is used for iterative model-based prediction and control in an MDP.

- Prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$ and policy π
  - Output: (estimated) value function $\hat{v}_\pi \approx v_\pi$
- Control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$
  - Output: (estimated) optimal value function $\hat{v}_\pi^* \approx v_\pi^*$ or policy $\hat{\pi}^* \approx \pi^*$

In both applications DP requires full knowledge of the MDP structure.

- Feasibility in real-world engineering applications (model vs. system) is therefore limited.
- But: following DP concepts are largely used in modern data-driven RL algorithms.

Table of contents

1 Introduction

2 Policy evaluation (Prediction)

3 Policy improvement (Control)

4 Policy and value iteration

5 Further aspects

Policy evaluation background (1)

- Problem: evaluate a given policy π to predict v_π.

Policy evaluation background (1)

- Problem: evaluate a given policy π to predict v_π.
- Recap: Bellman expectation equation for $s \in \mathcal{S}$ is given as

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right], \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s \right], \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right]. \end{aligned}$$

Policy evaluation background (1)

- Problem: evaluate a given policy π to predict v_π.
- Recap: Bellman expectation equation for $s \in \mathcal{S}$ is given as

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right], \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right], \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right].
\end{aligned}
$$

- Or in matrix form:

$$
\boldsymbol{v}_{\mathcal{S}}^{\pi} = \boldsymbol{r}_{\mathcal{S}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi} \boldsymbol{v}_{\mathcal{S}}^{\pi},
$$

$$
\begin{bmatrix} v_1^\pi \\ \vdots \\ v_n^\pi \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1^\pi \\ \vdots \\ \mathcal{R}_n^\pi \end{bmatrix} + \gamma \begin{bmatrix} p_{11}^\pi & \cdots & p_{1n}^\pi \\ \vdots & & \vdots \\ p_{n1}^\pi & \cdots & p_{nn}^\pi \end{bmatrix} \begin{bmatrix} v_1^\pi \\ \vdots \\ v_n^\pi \end{bmatrix}.
$$

Policy evaluation background (1)

- Problem: evaluate a given policy π to predict v_π.
- Recap: Bellman expectation equation for $s \in \mathcal{S}$ is given as

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right], \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right], \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right]. \end{aligned}$$

- Or in matrix form:

$$\boldsymbol{v}_{\mathcal{S}}^{\pi} = \boldsymbol{r}_{\mathcal{S}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi} \boldsymbol{v}_{\mathcal{S}}^{\pi},$$

$$\begin{bmatrix} v_1^\pi \\ \vdots \\ v_n^\pi \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1^\pi \\ \vdots \\ \mathcal{R}_n^\pi \end{bmatrix} + \gamma \begin{bmatrix} p_{11}^\pi & \cdots & p_{1n}^\pi \\ \vdots & & \vdots \\ p_{n1}^\pi & \cdots & p_{nn}^\pi \end{bmatrix} \begin{bmatrix} v_1^\pi \\ \vdots \\ v_n^\pi \end{bmatrix}.$$

- Solving the Bellman expectation equation for v_π requires handling a linear equation system with n unknowns (i.e., number of states).

Policy evaluation background (2)

- Problem: directly calculating v_π is numerically costly for high-dimensional state spaces (e.g., by matrix inversion).

Policy evaluation background (2)

- Problem: directly calculating v_π is numerically costly for high-dimensional state spaces (e.g., by matrix inversion).
- General idea: apply iterative approximations $\hat{v}_i(s) = v_i(s)$ of $v_\pi(s)$ with decreasing errors:

$$\left\| v_i(s) - v_\pi(s) \right\|_\infty \to 0 \quad \text{for} \quad i = 1, 2, 3, \ldots \tag{3.1}$$

Policy evaluation background (2)

- Problem: directly calculating v_π is numerically costly for high-dimensional state spaces (e.g., by matrix inversion).
- General idea: apply iterative approximations $\hat{v}_i(s) = v_i(s)$ of $v_\pi(s)$ with decreasing errors:

$$\left\| v_i(s) - v_\pi(s) \right\|_\infty \to 0 \quad \text{for} \quad i = 1, 2, 3, \ldots \tag{3.1}$$

- The Bellman equation in matrix form can be rewritten as:

$$\underbrace{\left(\boldsymbol{I} - \gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi}\right)}_{\boldsymbol{A}} \underbrace{\boldsymbol{v}_{\mathcal{S}}^{\pi}}_{\boldsymbol{x}} = \underbrace{\boldsymbol{r}_{\mathcal{S}}^{\pi}}_{\boldsymbol{b}} . \tag{3.2}$$

Policy evaluation background (2)

- Problem: directly calculating v_π is numerically costly for high-dimensional state spaces (e.g., by matrix inversion).
- General idea: apply iterative approximations $\hat{v}_i(s) = v_i(s)$ of $v_\pi(s)$ with decreasing errors:

$$\left\| v_i(s) - v_\pi(s) \right\|_\infty \to 0 \quad \text{for} \quad i = 1, 2, 3, \dots \tag{3.1}$$

- The Bellman equation in matrix form can be rewritten as:

$$\underbrace{\left(\boldsymbol{I} - \gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi}\right)}_{\boldsymbol{A}} \underbrace{\boldsymbol{v}_{\mathcal{S}}^{\pi}}_{\boldsymbol{x}} = \underbrace{\boldsymbol{r}_{\mathcal{S}}^{\pi}}_{\boldsymbol{b}} . \tag{3.2}$$

- To iteratively solve this linear equation $\boldsymbol{Ax} = \boldsymbol{b}$, one can apply numerous methods such as
  - General gradient descent,
  - Richardson iteration,
  - Krylov subspace methods.

Richardson iteration (1)

In the MDP context, the Richardson iteration became the default solution approach to iteratively solve:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

The Richardson iteration is

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \omega(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i) \tag{3.3}$$

with ω being a scalar parameter that has to be chosen such that the sequence $\boldsymbol{x}_i$ converges.

Richardson iteration (1)

In the MDP context, the Richardson iteration became the default solution approach to iteratively solve:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

The Richardson iteration is

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \omega(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i) \tag{3.3}$$

with ω being a scalar parameter that has to be chosen such that the sequence $\boldsymbol{x}_i$ converges. To choose ω we inspect the series of approximation errors $\boldsymbol{e}_i = \boldsymbol{x}_i - \boldsymbol{x}$ and apply it to (3.3):

$$\boldsymbol{e}_{i+1} = \boldsymbol{e}_i - \omega\boldsymbol{A}\boldsymbol{e}_i = (\boldsymbol{I} - \omega\boldsymbol{A})\,\boldsymbol{e}_i. \tag{3.4}$$

Richardson iteration (1)

In the MDP context, the Richardson iteration became the default solution approach to iteratively solve:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

The Richardson iteration is

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \omega(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_i) \tag{3.3}$$

with ω being a scalar parameter that has to be chosen such that the sequence $\boldsymbol{x}_i$ converges. To choose ω we inspect the series of approximation errors $\boldsymbol{e}_i = \boldsymbol{x}_i - \boldsymbol{x}$ and apply it to (3.3):

$$\boldsymbol{e}_{i+1} = \boldsymbol{e}_i - \omega \boldsymbol{A}\boldsymbol{e}_i = (\boldsymbol{I} - \omega \boldsymbol{A})\, \boldsymbol{e}_i. \tag{3.4}$$

To evaluate convergence we inspect the following norm:

$$\|\boldsymbol{e}_{i+1}\|_\infty = \|(\boldsymbol{I} - \omega \boldsymbol{A})\, \boldsymbol{e}_i\|_\infty . \tag{3.5}$$

Richardson iteration (2)

Since any induced matrix norm is sub-multiplicative, we can approximate (3.5) by the inequality:

$$\|\boldsymbol{e}_{i+1}\|_\infty \leq \|(\boldsymbol{I} - \omega \boldsymbol{A})\|_\infty \|\boldsymbol{e}_i\|_\infty . \tag{3.6}$$

Richardson iteration (2)

Since any induced matrix norm is sub-multiplicative, we can approximate (3.5) by the inequality:

$$\|\boldsymbol{e}_{i+1}\|_\infty \leq \|(\boldsymbol{I} - \omega \boldsymbol{A})\|_\infty \|\boldsymbol{e}_i\|_\infty . \tag{3.6}$$

Hence, the series converges if

$$\|(\boldsymbol{I} - \omega \boldsymbol{A})\|_\infty < 1. \tag{3.7}$$

Richardson iteration (2)

Since any induced matrix norm is sub-multiplicative, we can approximate (3.5) by the inequality:

$$\left\|\boldsymbol{e}_{i+1}\right\|_\infty \leq \left\|(\boldsymbol{I} - \omega \boldsymbol{A})\right\|_\infty \left\|\boldsymbol{e}_i\right\|_\infty . \tag{3.6}$$

Hence, the series converges if

$$\left\|(\boldsymbol{I} - \omega \boldsymbol{A})\right\|_\infty < 1. \tag{3.7}$$

Inserting from (3.2) leads to:

$$\left\|(\boldsymbol{I}(1-\omega) + \omega\gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi})\right\|_\infty < 1. \tag{3.8}$$

Richardson iteration (2)

Since any induced matrix norm is sub-multiplicative, we can approximate (3.5) by the inequality:

$$\|\boldsymbol{e}_{i+1}\|_\infty \leq \|(\boldsymbol{I} - \omega\boldsymbol{A})\|_\infty \|\boldsymbol{e}_i\|_\infty . \tag{3.6}$$

Hence, the series converges if

$$\|(\boldsymbol{I} - \omega\boldsymbol{A})\|_\infty < 1. \tag{3.7}$$

Inserting from (3.2) leads to:

$$\|(\boldsymbol{I}(1-\omega) + \omega\gamma\boldsymbol{\mathcal{P}}_{ss'}^{\pi})\|_\infty < 1. \tag{3.8}$$

For $\omega = 1$ we receive:

$$\gamma \|(\boldsymbol{\mathcal{P}}_{ss'}^{\pi})\|_\infty < 1. \tag{3.9}$$

Richardson iteration (2)

Since any induced matrix norm is sub-multiplicative, we can approximate (3.5) by the inequality:

$$\|\boldsymbol{e}_{i+1}\|_\infty \leq \|(\boldsymbol{I} - \omega \boldsymbol{A})\|_\infty \|\boldsymbol{e}_i\|_\infty . \tag{3.6}$$

Hence, the series converges if

$$\|(\boldsymbol{I} - \omega \boldsymbol{A})\|_\infty < 1. \tag{3.7}$$

Inserting from (3.2) leads to:

$$\|(\boldsymbol{I}(1-\omega) + \omega\gamma \boldsymbol{\mathcal{P}}^{\pi}_{ss'})\|_\infty < 1. \tag{3.8}$$

For $\omega = 1$ we receive:

$$\gamma \|(\boldsymbol{\mathcal{P}}^{\pi}_{ss'})\|_\infty < 1. \tag{3.9}$$

Since the row elements of $\boldsymbol{\mathcal{P}}^{\pi}_{ss'}$ always sum up to 1,

$$\gamma < 1 \tag{3.10}$$

follows. Hence, when discounting the Richardson iteration always converges for MDPs even if we assume $\omega = 1$.

Iterative policy evaluation by Richardson iteration (1)

Applying the Richardson iteration (3.3) with $w = 1$ to the Bellman equation for any $s \in \mathcal{S}$ at iteration i results in:

$$v_{i+1}(s) = \sum_{a \in \mathcal{A}} \boldsymbol{\pi}(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_i(s') \right) . \tag{3.11}$$

Iterative policy evaluation by Richardson iteration (1)

Applying the Richardson iteration (3.3) with $w = 1$ to the Bellman equation for any $s \in \mathcal{S}$ at iteration i results in:

$$v_{i+1}(s) = \sum_{a \in \mathcal{A}} \boldsymbol{\pi}(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_i(s') \right) . \tag{3.11}$$

Matrix form then is:

$$\boldsymbol{v}_{\mathcal{S},i+1}^{\pi} = \boldsymbol{r}_{\mathcal{S}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi} \boldsymbol{v}_{\mathcal{S},i}^{\pi} . \tag{3.12}$$

Iterative policy evaluation by Richardson iteration (1)

Applying the Richardson iteration (3.3) with $w = 1$ to the Bellman equation for any $s \in \mathcal{S}$ at iteration i results in:

$$v_{i+1}(s) = \sum_{a \in \mathcal{A}} \boldsymbol{\pi}(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_i(s') \right) . \tag{3.11}$$

Matrix form then is:

$$\boldsymbol{v}_{\mathcal{S},i+1}^{\pi} = \boldsymbol{r}_{\mathcal{S}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}_{ss'}^{\pi} \boldsymbol{v}_{\mathcal{S},i}^{\pi} . \tag{3.12}$$

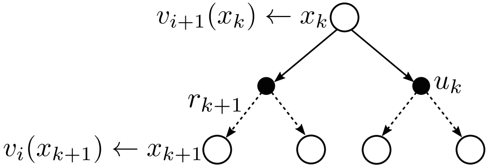


Fig. 3.1: Backup diagram for iterative policy evaluation

Iterative policy evaluation by Richardson iteration (2)

- During one Richardson iteration the 'old' value of s is replaced with a 'new' value from the 'old' values of the successor state s'.
  - Update $v_{i+1}(s)$ from $v_i(s')$, see Fig. 3.1.
  - Updating estimates (v_{i+1}) on the basis of other estimates (v_i) is often called bootstrapping.

Iterative policy evaluation by Richardson iteration (2)

- During one Richardson iteration the 'old' value of s is replaced with a 'new' value from the 'old' values of the successor state s'.
  - Update $v_{i+1}(s)$ from $v_i(s')$, see Fig. 3.1.
  - Updating estimates (v_{i+1}) on the basis of other estimates (v_i) is often called bootstrapping.
- The Richardson iteration can be interpreted as a gradient descent algorithm for solving (3.2).

Iterative policy evaluation by Richardson iteration (2)

- During one Richardson iteration the 'old' value of s is replaced with a 'new' value from the 'old' values of the successor state s'.
  - Update $v_{i+1}(s)$ from $v_i(s')$, see Fig. 3.1.
  - Updating estimates (v_{i+1}) on the basis of other estimates (v_i) is often called bootstrapping.
- The Richardson iteration can be interpreted as a gradient descent algorithm for solving (3.2).
- This leads to synchronous, full backups of the entire state space $\mathcal{S}$.

Iterative policy evaluation by Richardson iteration (2)

- During one Richardson iteration the 'old' value of s is replaced with a 'new' value from the 'old' values of the successor state s'.
  - Update $v_{i+1}(s)$ from $v_i(s')$, see Fig. 3.1.
  - Updating estimates (v_{i+1}) on the basis of other estimates (v_i) is often called bootstrapping.
- The Richardson iteration can be interpreted as a gradient descent algorithm for solving (3.2).
- This leads to synchronous, full backups of the entire state space $\mathcal{S}$.
- Also called expected update because it is based on the expectation over all possible next states (utilizing full model knowledge).

Iterative policy evaluation by Richardson iteration (2)

- During one Richardson iteration the 'old' value of s is replaced with a 'new' value from the 'old' values of the successor state s'.
  - Update $v_{i+1}(s)$ from $v_i(s')$, see Fig. 3.1.
  - Updating estimates (v_{i+1}) on the basis of other estimates (v_i) is often called bootstrapping.
- The Richardson iteration can be interpreted as a gradient descent algorithm for solving (3.2).
- This leads to synchronous, full backups of the entire state space $\mathcal{S}$.
- Also called expected update because it is based on the expectation over all possible next states (utilizing full model knowledge).
- In subsequent lectures, the expected update will be supplemented by data-driven samples from the environment.

Iterative policy evaluation example: forest tree MDP

Let's reuse the forest tree MDP example with *fifty-fifty policy*:

$$\boldsymbol{\mathcal{P}}_{ss'}^{\pi} = \begin{bmatrix} 0 & \frac{1-\alpha}{2} & 0 & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{r}_{\mathcal{S}}^{\pi} = \begin{bmatrix} 0.5 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

Iterative policy evaluation example: forest tree MDP

Let's reuse the forest tree MDP example with *fifty-fifty policy*:

$$\mathcal{P}^{\pi}_{ss'} = \begin{bmatrix} 0 & \frac{1-\alpha}{2} & 0 & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{r}^{\pi}_{\mathcal{S}} = \begin{bmatrix} 0.5 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0

Iterative policy evaluation example: forest tree MDP

Let's reuse the forest tree MDP example with *fifty-fifty policy*:

$$\mathcal{P}^{\pi}_{ss'} = \begin{bmatrix} 0 & \frac{1-\alpha}{2} & 0 & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{r}^{\pi}_{\mathcal{S}} = \begin{bmatrix} 0.5 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0
1	0.5	1	2	0

Iterative policy evaluation example: forest tree MDP

Let's reuse the forest tree MDP example with *fifty-fifty policy*:

$$\mathcal{P}_{ss'}^{\pi} = \begin{bmatrix} 0 & \frac{1-\alpha}{2} & 0 & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{r}_{\mathcal{S}}^{\pi} = \begin{bmatrix} 0.5 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0
1	0.5	1	2	0
2	0.82	1.64	2.64	0

Iterative policy evaluation example: forest tree MDP

Let's reuse the forest tree MDP example with *fifty-fifty policy*:

$$\boldsymbol{\mathcal{P}}_{ss'}^{\pi} = \begin{bmatrix} 0 & \frac{1-\alpha}{2} & 0 & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{r}_{\mathcal{S}}^{\pi} = \begin{bmatrix} 0.5 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0
1	0.5	1	2	0
2	0.82	1.64	2.64	0
3	1.03	1.85	2.85	0

Iterative policy evaluation example: forest tree MDP

Let's reuse the forest tree MDP example with *fifty-fifty policy*:

$$\mathcal{P}_{ss'}^{\pi} = \begin{bmatrix} 0 & \frac{1-\alpha}{2} & 0 & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & \frac{1-\alpha}{2} & \frac{1+\alpha}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{r}_{\mathcal{S}}^{\pi} = \begin{bmatrix} 0.5 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0
1	0.5	1	2	0
2	0.82	1.64	2.64	0
3	1.03	1.85	2.85	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
∞	1.12	1.94	2.94	0

Tab. 3.1: Policy evaluation by Richardson iteration (3.12) for forest tree MDP with $\gamma = 0.8$ and $\alpha = 0.2$

Variant: in-place updates

Instead of applying (3.12) to the entire vector $\boldsymbol{v}^{\pi}_{\mathcal{S},i+1}$ in 'one shot' (synchronous backup), an elementwise in-place version of the policy evaluation can be carried out:

input: full model of the MDP, i.e., $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$ including policy π
parameter: $\delta > 0$ as accuracy termination threshold
init: $v_0(s)\,\forall\, s \in \mathcal{S}$ arbitrary except $v_0(s) = 0$ if s is terminal
repeat
  $\Delta \leftarrow 0$;
  for $\forall\, s \in \mathcal{S}$ **do**
    $\tilde{v} \leftarrow \hat{v}(s)$;
    $\hat{v}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} p^a_{ss'} \hat{v}(s')\right)$;
    $\Delta \leftarrow \max\left(\Delta, |\tilde{v} - \hat{v}(s)|\right)$;
until $\Delta < \delta$;

Algo. 3.1: Iterative policy evaluation using in-place updates (output: estimate of $\boldsymbol{v}^{\pi}_{\mathcal{S}}$)

In-place policy evaluation updates for forest tree MDP

- In-place algorithms allow to update states in a beneficial order.

In-place policy evaluation updates for forest tree MDP

- In-place algorithms allow to update states in a beneficial order.
- May converge faster than regular Richardson iteration if state update order is chosen wisely (sweep through state space).

In-place policy evaluation updates for forest tree MDP

- In-place algorithms allow to update states in a beneficial order.
- May converge faster than regular Richardson iteration if state update order is chosen wisely (sweep through state space).
- For forest tree MDP: reverse order, i.e., start with $s = 4$.

In-place policy evaluation updates for forest tree MDP

- In-place algorithms allow to update states in a beneficial order.
- May converge faster than regular Richardson iteration if state update order is chosen wisely (sweep through state space).
- For forest tree MDP: reverse order, i.e., start with $s = 4$.
- As can be seen in Tab. 3.2 the in-place updates especially converge faster for the 'early states'.

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0
1	1.03	1.64	2	0
2	1.09	1.85	2.64	0
3	1.11	1.91	2.85	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
∞	1.12	1.94	2.94	0

Tab. 3.2: In-place updates for forest tree MDP

Table of contents

1 Introduction

2 Policy evaluation (Prediction)

3 Policy improvement (Control)

4 Policy and value iteration

5 Further aspects

General idea on policy improvement

- If we know v_π of a given MDP, how to improve the policy?

General idea on policy improvement

- If we know v_π of a given MDP, how to improve the policy?
- The simple idea of policy improvement is:
  - Consider a new (non-policy conform) action $a \neq \pi(s)$.
  - Follow thereafter the current policy π.
  - Check the action value of this 'new move'. If it is better than the 'old' value, take it:

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a\right] . \tag{3.13}$$

General idea on policy improvement

- If we know v_π of a given MDP, how to improve the policy?
- The simple idea of policy improvement is:
  - Consider a new (non-policy conform) action $a \neq \pi(s)$.
  - Follow thereafter the current policy π.
  - Check the action value of this 'new move'. If it is better than the 'old' value, take it:

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a\right] . \tag{3.13}$$

Theorem 3.1: Policy improvement

If for any deterministic policy pair π and π'

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \quad \forall s \in \mathcal{S} \tag{3.14}$$

applies, then the policy π' must be as good as or better than π. Hence, it obtains greater or equal expected return

$$v_{\pi'}(s) \geq v_\pi(s) \quad \forall s \in \mathcal{S}. \tag{3.15}$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$v_\pi(s) \leq q_\pi(s, \pi'(s)),$$

(3.16)

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right],
\end{aligned}
\tag{3.16}
$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right],
\end{aligned}
\tag{3.16}
$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right],
\end{aligned}
\tag{3.16}
$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \\
&\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, \pi'(S_{t+1})\right] | S_t = s\right],
\end{aligned}
\tag{3.16}
$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, \pi'(S_{t+1})\right] | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s\right],
\end{aligned}
\tag{3.16}
$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \\
&\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, \pi'(S_{t+1})\right] | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s\right], \\
&\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s\right],
\end{aligned}
\tag{3.16}
$$

Proof of policy improvement theorem

Start with (3.14) and recursively reapply (3.13):

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)), \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, \pi'(S_{t+1})\right] | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s\right], \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s\right], \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots | S_t = s\right], \\
&= v_{\pi'}(s).
\end{aligned}
\tag{3.16}
$$

Greedy policy improvement (1)

- So far, policy improvement addressed only changing the policy at a single state.

Greedy policy improvement (1)

- So far, policy improvement addressed only changing the policy at a single state.
- Now, extend this scheme to all states by selecting the best action according to $q_\pi(s, a)$ in every state (greedy policy improvement):

Greedy policy improvement (1)

- So far, policy improvement addressed only changing the policy at a single state.
- Now, extend this scheme to all states by selecting the best action according to $q_\pi(s, a)$ in every state (greedy policy improvement):

$$
\begin{aligned}
\pi'(s) &= \arg\max_{a\in\mathcal{A}} q_\pi(s, a), \\
&= \arg\max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a\right], \\
&= \arg\max_{a\in\mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} p_{ss'}^a v_\pi(s') \, .
\end{aligned}
\tag{3.17}
$$

Greedy policy improvement (2)

- Each greedy policy improvement takes the best action in a one-step look-ahead search and, therefore, satisfies Theo. 3.1.

Greedy policy improvement (2)

- Each greedy policy improvement takes the best action in a one-step look-ahead search and, therefore, satisfies Theo. 3.1.
- If after a policy improvement step $v_\pi(s) = v_{\pi'}(s)$ applies, it follows:

$$\begin{aligned} v_{\pi'}(s) &= \max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a\right], \\ &= \max_{a\in\mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} p_{ss'}^a v_{\pi'}(s') . \end{aligned} \tag{3.18}$$

Greedy policy improvement (2)

- Each greedy policy improvement takes the best action in a one-step look-ahead search and, therefore, satisfies Theo. 3.1.
- If after a policy improvement step $v_\pi(s) = v_{\pi'}(s)$ applies, it follows:

$$\begin{aligned} v_{\pi'}(s) &= \max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a\right], \\ &= \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_{\pi'}(s'). \end{aligned} \tag{3.18}$$

- This is the Bellman optimality equation, which guarantees that $\pi' = \pi$ must be optimal policies.

Greedy policy improvement (2)

- Each greedy policy improvement takes the best action in a one-step look-ahead search and, therefore, satisfies Theo. 3.1.
- If after a policy improvement step $v_\pi(s) = v_{\pi'}(s)$ applies, it follows:

$$\begin{aligned} v_{\pi'}(s) &= \max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a\right], \\ &= \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_{\pi'}(s'). \end{aligned} \tag{3.18}$$

- This is the Bellman optimality equation, which guarantees that $\pi' = \pi$ must be optimal policies.
- Although the proof for policy improvement theorem was presented for deterministic policies, transfer to stochastic policies $\pi(a|s)$ is possible.

Greedy policy improvement (2)

- Each greedy policy improvement takes the best action in a one-step look-ahead search and, therefore, satisfies Theo. 3.1.
- If after a policy improvement step $v_\pi(s) = v_{\pi'}(s)$ applies, it follows:

$$\begin{aligned} v_{\pi'}(s) &= \max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a\right], \\ &= \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_{\pi'}(s'). \end{aligned} \tag{3.18}$$

- This is the Bellman optimality equation, which guarantees that $\pi' = \pi$ must be optimal policies.
- Although the proof for policy improvement theorem was presented for deterministic policies, transfer to stochastic policies $\pi(a|s)$ is possible.
- Takeaway message: policy improvement theorem guarantees finding optimal policies in finite MDPs (e.g., by DP).

Table of contents

1. Introduction
2. Policy evaluation (Prediction)
3. Policy improvement (Control)
4. Policy and value iteration
5. Further aspects

Concept of policy iteration

- Policy iteration combines the previous policy evaluation and policy improvement in an iterative sequence:

$$\pi_0 \rightarrow v_{\pi_0} \rightarrow \pi_1 \rightarrow v_{\pi_1} \rightarrow \cdots \pi^* \rightarrow v_{\pi^*} \tag{3.19}$$

- Evaluate $\rightarrow$ improve $\rightarrow$ evaluate $\rightarrow$ improve ...

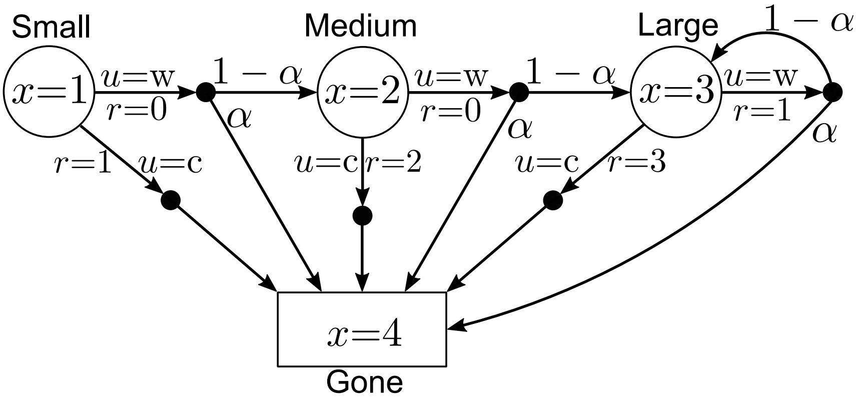
Concept of policy iteration

- Policy iteration combines the previous policy evaluation and policy improvement in an iterative sequence:

$$\pi_0 \rightarrow v_{\pi_0} \rightarrow \pi_1 \rightarrow v_{\pi_1} \rightarrow \cdots \pi^* \rightarrow v_{\pi^*} \tag{3.19}$$

- Evaluate $\rightarrow$ improve $\rightarrow$ evaluate $\rightarrow$ improve ...
- In the 'classic' policy iteration, each policy evaluation step in (3.19) is fully executed, i.e., for each policy π_i an exact estimate of v_{π_i} is provided either by iterative policy evaluation with a sufficiently high number of steps or by any other method that fully solves (3.2).

Policy iteration example: forest tree MDP (1)



- Two actions possible in each state:
  - Wait $a = w$: let the tree grow.
  - Cut $a = c$: gather the wood.

Policy iteration example: forest tree MDP (2)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree hater' initial policy:

Policy iteration example: forest tree MDP (2)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree hater' initial policy:

1. $\pi_0 = \pi(a = \mathsf{c}|s) \quad \forall s \in \mathcal{S}$.

Policy iteration example: forest tree MDP (2)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree hater' initial policy:

1. $\pi_0 = \pi(a = \mathsf{c}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix}1 & 2 & 3 & 0\end{bmatrix}^\mathsf{T}$

Policy iteration example: forest tree MDP (2)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree hater' initial policy:

1. $\pi_0 = \pi(a = \mathsf{c}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix}1 & 2 & 3 & 0\end{bmatrix}^\mathsf{T}$
3. Greedy policy improvement:

$$\begin{aligned}\pi_1(s) &= \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_0}(S_{t+1}) | S_t = s, A_t = a\right], \\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\}\end{aligned}$$

Policy iteration example: forest tree MDP (2)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree hater' initial policy:

1. $\pi_0 = \pi(a = \mathsf{c}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix}1 & 2 & 3 & 0\end{bmatrix}^\mathsf{T}$
3. Greedy policy improvement:

$$\begin{aligned}\pi_1(s) &= \underset{a\in\mathcal{A}}{\arg\max}\, \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_0}(S_{t+1})|S_t = s, A_t = a\right], \\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\}\end{aligned}$$

4. Policy evaluation: $v_{\mathcal{S}}^{\pi_1} = \begin{bmatrix}1.28 & 2 & 3 & 0\end{bmatrix}^\mathsf{T}$

Policy iteration example: forest tree MDP (2)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree hater' initial policy:

1. $\pi_0 = \pi(a = \mathsf{c}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix}1 & 2 & 3 & 0\end{bmatrix}^{\mathsf{T}}$
3. Greedy policy improvement:

$$\begin{aligned}\pi_1(s) &= \arg\max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_0}(S_{t+1})|S_t = s, A_t = a\right],\\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\}\end{aligned}$$

4. Policy evaluation: $v_{\mathcal{S}}^{\pi_1} = \begin{bmatrix}1.28 & 2 & 3 & 0\end{bmatrix}^{\mathsf{T}}$
5. Greedy policy improvement:

$$\begin{aligned}\pi_2(s) &= \arg\max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_1}(S_{t+1})|S_t = s, A_t = a\right],\\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\},\\ &= \pi_1(s)\\ &= \pi^*\end{aligned}$$

Policy iteration example: forest tree MDP (3)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree lover' initial policy:

Policy iteration example: forest tree MDP (3)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree lover' initial policy:

1. $\pi_0 = \pi(a = \mathsf{w}|s) \quad \forall s \in \mathcal{S}$.

Policy iteration example: forest tree MDP (3)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree lover' initial policy:

1. $\pi_0 = \pi(a = \mathsf{w}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix}1.14 & 1.78 & 2.78 & 0\end{bmatrix}^{\mathsf{T}}$

Policy iteration example: forest tree MDP (3)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree lover' initial policy:

1. $\pi_0 = \pi(a = \mathsf{w}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix} 1.14 & 1.78 & 2.78 & 0 \end{bmatrix}^\mathsf{T}$
3. Greedy policy improvement:

$$\begin{aligned}\pi_1(s) &= \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_0}(S_{t+1}) | S_t = s, A_t = a\right],\\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\}\end{aligned}$$

Policy iteration example: forest tree MDP (3)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree lover' initial policy:

1. $\pi_0 = \pi(a = \mathsf{w}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix}1.14 & 1.78 & 2.78 & 0\end{bmatrix}^\mathsf{T}$
3. Greedy policy improvement:

$$\begin{aligned}\pi_1(s) &= \arg\max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_0}(S_{t+1})|S_t = s, A_t = a\right],\\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\}\end{aligned}$$

4. Policy evaluation: $v_{\mathcal{S}}^{\pi_1} = \begin{bmatrix}1.28 & 2 & 3 & 0\end{bmatrix}^\mathsf{T}$

Policy iteration example: forest tree MDP (3)

Assume $\alpha = 0.2$ and $\gamma = 0.8$ and start with 'tree lover' initial policy:

1. $\pi_0 = \pi(a = \mathsf{w}|s) \quad \forall s \in \mathcal{S}$.
2. Policy evaluation: $v_{\mathcal{S}}^{\pi_0} = \begin{bmatrix} 1.14 & 1.78 & 2.78 & 0 \end{bmatrix}^{\mathsf{T}}$
3. Greedy policy improvement:

$$\begin{aligned}\pi_1(s) &= \arg\max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_0}(S_{t+1})|S_t = s, A_t = a\right],\\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\}\end{aligned}$$

4. Policy evaluation: $v_{\mathcal{S}}^{\pi_1} = \begin{bmatrix} 1.28 & 2 & 3 & 0 \end{bmatrix}^{\mathsf{T}}$
5. Greedy policy improvement:

$$\begin{aligned}\pi_2(s) &= \arg\max_{a\in\mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_1}(S_{t+1})|S_t = s, A_t = a\right],\\ &= \{\pi(a = \mathsf{w}|s = 1), \pi(a = \mathsf{c}|s = 2), \pi(a = \mathsf{c}|s = 3)\},\\ &= \pi_1(s)\\ &= \pi^*\end{aligned}$$

Value iteration (1)

- Policy iteration involves full policy evaluation steps between policy improvements.
- In large state-space MDPs the full policy evaluation may be numerically very costly.

Value iteration (1)

- Policy iteration involves full policy evaluation steps between policy improvements.
- In large state-space MDPs the full policy evaluation may be numerically very costly.

- Value iteration: One step iterative policy evaluation followed by policy improvement.

Value iteration (1)

- Policy iteration involves full policy evaluation steps between policy improvements.
- In large state-space MDPs the full policy evaluation may be numerically very costly.

- Value iteration: One step iterative policy evaluation followed by policy improvement.
- Allows simple update rule which combines policy improvement with truncated policy evaluation in a single step:

$$\begin{aligned} v_{i+1}(s) &= \max_{s \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma v_i(S_{t+1}) | S_t = s, A_t = a\right], \\ &= \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_i(s'). \end{aligned} \tag{3.20}$$

Value iteration (2)

input: full model of the MDP, i.e., $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{P}}, \mathcal{R}, \gamma \rangle$
parameter: $\delta > 0$ as accuracy termination threshold
init: $v_0(s) \, \forall \, s \in \mathcal{S}$ arbitrary except $v_0(s) = 0$ if s is terminal
repeat
    $\Delta \leftarrow 0$;
    for $\forall \, s \in \mathcal{S}$ **do**
        $\tilde{v} \leftarrow \hat{v}(s)$;
        $\hat{v}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a \hat{v}(s') \right)$;
        $\Delta \leftarrow \max \left(\Delta, |\tilde{v} - \hat{v}(x_k)| \right)$;
until $\Delta < \delta$;
output: deterministic policy $\pi \approx \pi^*$, such that
$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a \hat{v}(s') \right)$;

Algo. 3.2: Value iteration (note: compared to policy iteration, value iteration does not require an initial policy but only a state-value guess)

Value iteration example: forest tree MDP

- Assume again $\alpha = 0.2$ and $\gamma = 0.8$.

Value iteration example: forest tree MDP

- Assume again $\alpha = 0.2$ and $\gamma = 0.8$.
- Similar to in-place update policy evaluation, reverse order and start value iteration with $s = 4$.

Value iteration example: forest tree MDP

- Assume again $\alpha = 0.2$ and $\gamma = 0.8$.
- Similar to in-place update policy evaluation, reverse order and start value iteration with $s = 4$.
- As shown in Tab. 3.3 value iteration converges in one step (for the given problem) to the optimal state value.

i	$v_i(s=1)$	$v_i(s=2)$	$v_i(s=3)$	$v_i(s=4)$
0	0	0	0	0
1	1.28	2	3	0
*	1.28	2	3	0

Tab. 3.3: Value iteration for forest tree MDP

Table of contents

1. Introduction
2. Policy evaluation (Prediction)
3. Policy improvement (Control)
4. Policy and value iteration
5. Further aspects

Summarizing DP algorithms

- All DP algorithms are based on the state value $v(s)$.
  - Complexity is $\mathcal{O}(m \cdot n^2)$ for m actions and n states.
  - Evaluate all n^2 state transitions while considering up to m actions per state.

Summarizing DP algorithms

- All DP algorithms are based on the state value $v(s)$.
  - Complexity is $\mathcal{O}(m \cdot n^2)$ for m actions and n states.
  - Evaluate all n^2 state transitions while considering up to m actions per state.
- Could be also applied to action values $q(s, a)$.
  - Complexity is inferior with $\mathcal{O}(m^2 \cdot n^2)$.
  - There are up to m^2 action values which require n^2 state transition evaluations each.

Summarizing DP algorithms

- All DP algorithms are based on the state value $v(s)$.
  - Complexity is $\mathcal{O}(m \cdot n^2)$ for m actions and n states.
  - Evaluate all n^2 state transitions while considering up to m actions per state.
- Could be also applied to action values $q(s, a)$.
  - Complexity is inferior with $\mathcal{O}(m^2 \cdot n^2)$.
  - There are up to m^2 action values which require n^2 state transition evaluations each.

Problem	Relevant Equations	Algorithm
prediction	Bellman expectation eq.	policy evaluation
control	Bellman expectation eq. & greedy policy improvement	policy iteration
control	Bellman optimality eq.	value iteration

Tab. 3.4: Short overview addressing the treated DP algorithms

Curse of dimensionality

- DP is much more efficient than an exhaustive search over all n states and m actions in finite MDPs in order to find an optimal policy.
  - Exhaustive search for deterministic policies: m^n evaluations.
  - DP results in polynomial complexity regarding m and n.

Curse of dimensionality

- DP is much more efficient than an exhaustive search over all n states and m actions in finite MDPs in order to find an optimal policy.
  - Exhaustive search for deterministic policies: m^n evaluations.
  - DP results in polynomial complexity regarding m and n.
- Nevertheless, DP uses full-width backups:
  - For each state update, every successor state and action is considered.
  - While utilizing full knowledge of the MDP structure.

Curse of dimensionality

- DP is much more efficient than an exhaustive search over all n states and m actions in finite MDPs in order to find an optimal policy.
  - Exhaustive search for deterministic policies: m^n evaluations.
  - DP results in polynomial complexity regarding m and n.
- Nevertheless, DP uses full-width backups:
  - For each state update, every successor state and action is considered.
  - While utilizing full knowledge of the MDP structure.
- Hence, DP is can be effective up to medium-sized MDPs (i.e., million finite states)

Curse of dimensionality

- DP is much more efficient than an exhaustive search over all n states and m actions in finite MDPs in order to find an optimal policy.
  - Exhaustive search for deterministic policies: m^n evaluations.
  - DP results in polynomial complexity regarding m and n.
- Nevertheless, DP uses full-width backups:
  - For each state update, every successor state and action is considered.
  - While utilizing full knowledge of the MDP structure.
- Hence, DP is can be effective up to medium-sized MDPs (i.e., million finite states)
- For large problems DP suffers from the curse of dimensionality:
  - Single update step may become computational infeasible.
  - Also: if continuous states need quantization, number of finite states n grows exponentially with the number of state variables (assuming fixed number of discretization levels).

Generalized policy iteration (GPI)

- Almost all RL methods are well-described as GPI.
- Push-pull: Improving the policy will deteriorate value estimation.
- Well balanced trade-off between evaluating and improving is required.

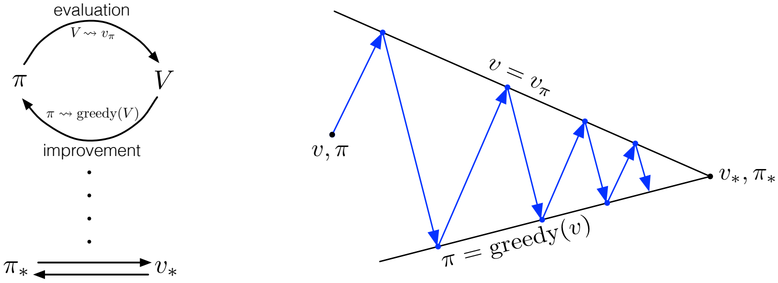


Fig. 3.2: Interpreting generalized policy iteration to switch back and forth between (arbitrary) evaluations and improvement steps

Summary

- DP is applicable for prediction and control problems in MDPs.

Summary

- DP is applicable for prediction and control problems in MDPs.
- But requires always full knowledge about the environment (i.e., it is a model-based solution).

Summary

- DP is applicable for prediction and control problems in MDPs.
- But requires always full knowledge about the environment (i.e., it is a model-based solution).
- DP is more efficient than the exhaustive search.

Summary

- DP is applicable for prediction and control problems in MDPs.
- But requires always full knowledge about the environment (i.e., it is a model-based solution).
- DP is more efficient than the exhaustive search.
- But suffers from the curse of dimensionality for large MDPs.

Summary

- DP is applicable for prediction and control problems in MDPs.
- But requires always full knowledge about the environment (i.e., it is a model-based solution).
- DP is more efficient than the exhaustive search.
- But suffers from the curse of dimensionality for large MDPs.
- (Iterative) policy evaluations and (greedy) improvements solve MDPs.

Summary

- DP is applicable for prediction and control problems in MDPs.
- But requires always full knowledge about the environment (i.e., it is a model-based solution).
- DP is more efficient than the exhaustive search.
- But suffers from the curse of dimensionality for large MDPs.
- (Iterative) policy evaluations and (greedy) improvements solve MDPs.
- Both steps can be combined via value iteration.

Summary

- DP is applicable for prediction and control problems in MDPs.
- But requires always full knowledge about the environment (i.e., it is a model-based solution).
- DP is more efficient than the exhaustive search.
- But suffers from the curse of dimensionality for large MDPs.
- (Iterative) policy evaluations and (greedy) improvements solve MDPs.
- Both steps can be combined via value iteration.
- The idea of (generalized) policy iteration is a basic scheme of RL.