CNRS - Toulouse INP - UT3 - UT Capitole - UT2

Institut de Recherche en Informatique de Toulouse

Lecture 3: Dynamic Programming

N8EN18B - Contrôle et Apprentissage

Guilherme IECKER RICARDO

IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3

Contents

1 Introduction

2 Policy Evaluation
- Iterative Policy Evaluation
- Example: Evaluating a Random Policy

3 Policy Iteration

4 Value Iteration

5 Questions?

Recap – MRPs and MDPs

Recap – Bellman Equations

- Bellman Expectation Equation

$$
\begin{aligned}
v_\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|\, s) \cdot \left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \cdot v_\pi(s') \right) \\
q_\pi(s,a) &= \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} \pi(a'|\, s') \cdot q_\pi(s',a')
\end{aligned} \tag{1}
$$

- Bellman Optimality Equation

$$
\begin{aligned}
v_*(s) &= \max_{a\in\mathcal{A}} \left\{ \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \cdot v_*(s') \right\} \\
q_*(s,a) &= \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \cdot \max_{a'\in\mathcal{A}} \left\{ q_*(s',a') \right\}
\end{aligned} \tag{2}
$$

Introduction

What is Dynamic Programming?

- **Dynamic**: sequential or temporal component to the problem
- **Programming**: optimizing a “problem”, i.e., a policy
  - c.f. linear programming
- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
  - Principle of optimality
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused

MDPs satisfy both properties, i.e.,

- Bellman equation gives recursive decomposition
- Value function stores and reuses solutions

Planning by Dynamic Programming

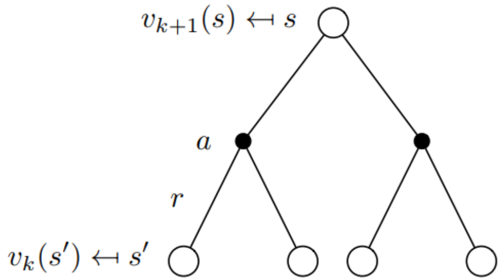
- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP
- Either for **prediction**:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π (or MRP)
  - Output: Value function v_π
- Or for **control**:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: Optimal value function v_* and optimal policy π_*

Policy Evaluation

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \to v_2 \to v_3 \to \cdots \to v_\pi$
- Using *synchronous* backups,
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s
- (Maybe) We will discuss *asynchronous* backups later
- (Maybe) Convergence to v_π will be proved at the end of the lecture

Iterative Policy Evaluation



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \cdot v_k(s') \right) \tag{3}$$

$$\mathbf{v}_{k+1} = \boldsymbol{\mathcal{R}}^\pi + \gamma \cdot \boldsymbol{\mathcal{P}}^\pi \cdot \mathbf{v}_k$$

Example: Small *Gridworld*

Rules:

- Undiscounted episodic MDP, i.e., $\gamma = 1$
- Non-terminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is $r = -1$ until the terminal state is reached
- Agent follows uniform random policy

$$\pi(\uparrow | \cdot) = \pi(\rightarrow | \cdot) = \pi(\downarrow | \cdot) = \pi(\leftarrow | \cdot) = 0.25$$

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

Hands-on Example: Small Gridworld

https://guilhermeir.github.io/teaching/rl/dp.ipynb
Notebook's exercises (1) – (4)

- Describe the MDP
- Define the policy evaluation function
- Evaluate a random policy
- Evaluate a better (?) policy

Policy Iteration

How to Improve a Policy

- Given a policy π
  - **Evaluate** the policy π

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+s} + \dots \,|S_t = s] \tag{4}$$

  - **Improve** the policy by acting greedily with respect to v_π

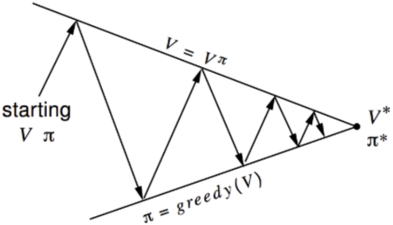
$$\pi' = \text{greedy}(v_\pi) \tag{5}$$

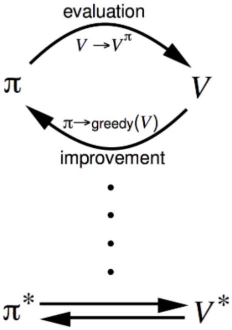
- In general, it needs more iterations of improvement / evaluation
- But this process of **policy iteration** always converges to π^*

Policy Iteration - Example

Gridworld Example – Textbook §4.2 - Figure 4.1

Policy Iteration - Intuition





Policy Evaluation: Estimate v_π Iterative policy evaluation
Policy Improvement: Generate $\pi' \geq \pi$ Greedy policy improvement

Policy Iteration - Algorithm

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\mathit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r \,|\, s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \,|\, s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration - Proof of Optimality 1/3

- Consider a deterministic policy, $a = \pi(s)$
- We can improve the policy by acting greedily, i.e.,

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s, a) \tag{6}$$

- This improves the value from any state s over one step, because

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s) \tag{7}$$

Policy Iteration - Proof of Optimality 2/3

Policy iteration improves value function, i.e., $v_{\pi'}(s) \geq v_\pi(s)$, because

$$
\begin{aligned}
v_\pi &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma \cdot q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots | S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}
\tag{8}
$$

Policy Iteration - Proof of Optimality 3/3

- If improvements stop, i.e.,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s) \tag{9}$$

- Then the Bellman optimality equation has been satisfied, i.e.,

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a) \tag{10}$$

- Therefore $v_\pi(s) = v_*(s), \forall s \in \mathcal{S}$
- so π is an optimal policy

Modified Policy Iteration

- In order to obtain the optimal policy π_*, do we need to converge to v_π?
- Include stop criterion:
  - ϵ-convergence of value function
  - after k iterations of PE-PI
- Gridworld Example – Textbook §4.2 - Figure 4.1
- What if we update the policy at every iteration of the PE (i.e., stop with $k = 1$)?

Hands-on Example: Small Gridworld

Notebook's exercises (5) – (6)

- Observe the application of Policy Improvement
- Define the policy iteration function

Hands-on Example: Gridworld

- Implement the Policy Iteration algorithm
- Observe the convergence in practice

Value Iteration

Principle of Optimality

Any optimal policy can be subdivided into two components:

- An optimal first action a_*
- Followed by an optimal policy from successor state s'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, iff

- *For any state s' reachable from s*
- *π achieves the optimal value from state s', $v_\pi(s) = v_*(s)$*

Deterministic Value Iteration

- If we know the solution to subproblems $v_*(s')$, then
- solution $v_*(s)$ can be found by Bellman Optimality Equations, i.e.,

$$v_*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \cdot v_*(s') \right], \tag{11}$$

- which is called one-step *lookahead*
- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \to v_2 \to \cdots \to v_*$
- Using synchronous backups:
  - At each iteration $k+1$,
  - for all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later (maybe)
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Value Iteration – Algorithm

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Hands-on Example: Small Gridworld

Notebook's exercises (7) – (8)

- Define the Value Iteration function
- Compare Policy Iteration and Value Iteration

Questions?