Chapter 4: Dynamic Programming

Objectives of this chapter:

- Overview of a collection of classical solution methods for MDPs known as dynamic programming (DP)
- Show how DP can be used to compute value functions, and hence, optimal policies
- Discuss efficiency and utility of DP
Policy Evaluation

**Policy Evaluation**: for a given policy \( \pi \), compute the state-value function \( V^\pi \)

Recall: **State-value function for policy \( \pi \):**

\[
V^\pi(s) = E_\pi \{ R_t \mid s_t = s \} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}
\]

**Bellman equation for \( V^\pi \):**

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

— a system of \(|S|\) simultaneous linear equations
Iterative Methods

\[ V_0 \rightarrow V_1 \rightarrow \cdots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \cdots \rightarrow V^\pi \]

A "sweep"

A sweep consists of applying a backup operation to each state.

A full policy-evaluation backup:

\[ V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \]
Iterative Policy Evaluation

Input $\pi$, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in S^+$
Repeat

\[ \Delta \leftarrow 0 \]

For each $s \in S$:

\[ v \leftarrow V(s) \]

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

\[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]

until $\Delta < \theta$ (a small positive number)

Output $V \approx V^\pi$
A Small Gridworld

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached
# Iterative Policy Eval for the Small Gridworld

<table>
<thead>
<tr>
<th>$V_k$ for the Random Policy</th>
<th>Greedy Policy w.r.t. $V_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>k = 0</td>
<td>![Policy Matrix]</td>
</tr>
<tr>
<td>k = 1</td>
<td>![Policy Matrix]</td>
</tr>
<tr>
<td>k = 2</td>
<td>![Policy Matrix]</td>
</tr>
<tr>
<td>k = 3</td>
<td>![Policy Matrix]</td>
</tr>
<tr>
<td>k = 10</td>
<td>![Policy Matrix]</td>
</tr>
<tr>
<td>k = $\infty$</td>
<td>![Policy Matrix]</td>
</tr>
</tbody>
</table>

$\pi = \text{equiprobable random action choices}$

---

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Policy Improvement

Suppose we have computed $V^\pi$ for a deterministic policy $\pi$.

For a given state $s$, would it be better to do an action $a \neq \pi(s)$?

The value of doing $a$ in state $s$ is:

$$Q^\pi(s,a) = E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \right\} | s_t = s, a_t = a$$

$$= \sum_{s'} P_{ss'}^{\alpha} \left[ R_{ss'}^{\alpha} + \gamma V^\pi(s') \right]$$

It is better to switch to action $a$ for state $s$ if and only if

$$Q^\pi(s,a) > V^\pi(s)$$
Policy Improvement Cont.

Do this for all states to get a new policy $\pi'$ that is \textbf{greedy} with respect to $V^\pi$:

$$\pi'(s) = \arg\max_a Q^\pi(s, a)$$

$$= \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]$$

Then $V^{\pi'} \geq V^\pi$
What if \( V' = V \)?

i.e., for all \( s \in S \), \( V'(s) = \max_a \sum_{s'} P_{ss'}^{a}[R_{ss'}^{a} + \gamma V(s')] \) ?

But this is the Bellman Optimality Equation.

So \( V' = V^* \) and both \( \pi \) and \( \pi' \) are optimal policies.
Policy Iteration

\[ \pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \cdots \pi^* \rightarrow V^* \rightarrow \pi^* \]

- **policy evaluation**
- **policy improvement**
- “greedification”
Policy Iteration

1. Initialization
   \( V(s) \in \mathbb{R} \) and \( \pi(s) \in \mathcal{A}(s) \) arbitrarily for all \( s \in \mathcal{S} \)

2. Policy Evaluation
   Repeat
   \[
   \Delta \leftarrow 0 \\
   \text{For each } s \in \mathcal{S}: \\
   v \leftarrow V(s) \\
   V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[ \mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right] \\
   \Delta \leftarrow \max(\Delta, |v - V(s)|) \\
   \]
   until \( \Delta < \theta \) (a small positive number)

3. Policy Improvement
   \( policy-stable \leftarrow true \)
   For each \( s \in \mathcal{S} \):
   \[
   b \leftarrow \pi(s) \\
   \pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[ \mathcal{R}_{ss'}^{a} + \gamma V(s') \right] \\
   \]
   If \( b \neq \pi(s) \), then \( policy-stable \leftarrow false \)
   If \( policy-stable \), then stop; else go to 2
Jack’s Car Rental

- $10 for each car rented (must be available when request rec’d)
- Two locations, maximum of 20 cars at each
- Cars returned and requested randomly
  - Poisson distribution, \( n \) returns/requests with prob \( \frac{\lambda^n}{n!} e^{-\lambda} \)
  - 1st location: average requests = 3, average returns = 3
  - 2nd location: average requests = 4, average returns = 2
- Can move up to 5 cars between locations overnight

- States, Actions, Rewards?
- Transition probabilities?
Jack’s Car Rental
Jack’s CR Exercise

- Suppose the first car moved is free
  - From 1st to 2nd location
  - Because an employee travels that way anyway (by bus)
- Suppose only 10 cars can be parked for free at each location
  - More than 10 cost $4 for using an extra parking lot
- Such arbitrary nonlinearities are common in real problems
Recall the **full policy-evaluation backup**:

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

Here is the **full value-iteration backup**:

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

Initialize $V$ arbitrarily, e.g., $V(s) = 0$, for all $s \in S^+$

Repeat

$\Delta \leftarrow 0$

For each $s \in S$:

$v \leftarrow V(s)$

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

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that

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

- Gambler can repeatedly bet $ on a coin flip
- Heads he wins his stake, tails he loses it
- Initial capital $ \in \{1, 2, \ldots, 99\}$
- Gambler wins if his capital becomes $100$
  loses if it becomes $0$
- Coin is unfair
  - Heads (gambler wins) with probability $p = .4$

- States, Actions, Rewards?
Gambler’s Problem Solution

![Graph showing the value estimates and final policy for the Gambler’s Problem solution.](image)
Herd Management

- You are a consultant to a farmer managing a herd of cows
- Herd consists of 5 kinds of cows:
  - Young
  - Milking
  - Breeding
  - Old
  - Sick
- Number of each kind is the State
- Number sold of each kind is the Action
- Cows transition from one kind to another
- Young cows can be born
Asynchronous DP

- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Pick a state at random and apply the appropriate backup
- Still need lots of computation, but does not get locked into hopelessly long sweeps
- Can you select states to backup intelligently? YES: an agent’s experience can act as a guide.
Generalized Policy Iteration

Generalized Policy Iteration (GPI):
any interaction of policy evaluation and policy improvement, independent of their granularity.

A geometric metaphor for convergence of GPI:
Efficiency of DP

- To find an optimal policy is polynomial in the number of states…
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- In practice, classical DP can be applied to problems with a few millions of states.
- Asynchronous DP can be applied to larger problems, and is appropriate for parallel computation.
- It is surprisingly easy to come up with MDPs for which DP methods are not practical.
Summary

- Policy evaluation: backups without a max
- Policy improvement: form a greedy policy, if only locally
- Policy iteration: alternate the above two processes
- Value iteration: backups with a max
- Full backups (to be contrasted later with sample backups)
- Generalized Policy Iteration (GPI)
- Asynchronous DP: a way to avoid exhaustive sweeps
- **Bootstrapping**: updating estimates based on other estimates