Dynamic Programming

*Suggested reading:*

Chapter 4 in R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction
Dynamic Programming

Contents:

- Policy evaluation
- Policy improvement
- Policy iteration
- Value iteration
- Asynchronous dynamic programming
- Generalized policy iteration
- Efficiency of dynamic programming
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
Key idea of Dynamic Programming

- Environment finite MDP (not necessary but typical)
- the use of value functions to organize and structure the search for good policies
- Once we have the value function we can easily obtain optimal policies
- DP algorithms are obtained by turning Bellman equations such as these

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

into assignments, that is, into update rules for improving approximations of the desired value functions.
Policy Evaluation: for a given policy $\pi$, compute the state-value function $V^\pi$

Recall: State-value function for policy $\pi$:

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

Bellman equation for $V^\pi$:

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

—a system of $|S|$ simultaneous linear equations
Iterative Methods

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

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

A full policy-evaluation backup:

\[
V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a \left[ \mathcal{R}_{ss'}^a + \gamma V_k(s') \right]
\]
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 $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [R_{ss'}^a + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  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
  \[ P_{5,6}^{right} = 1, \quad P_{5,10}^{right} = 0 \quad \text{and} \quad P_{7,7}^{right} = 1 \]
- Reward is \(-1\) until the terminal state is reached

\( r = -1 \)

on all transitions
Iterative Policy Eval for the Small Gridworld

$$\pi = \text{random (uniform)}$$

action choices

$$V_k$$ for the random policy

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

For $$k = 0$$

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

For $$k = 1$$

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

For $$k = 2$$

Greedy policy w.r.t. $$V_k$$

Random policy
Iterative Policy Eval for the Small Gridworld

$k = 3$

$$
\begin{array}{cccc}
0.0 & -2.4 & -2.9 & -3.0 \\
-2.4 & -2.9 & -3.0 & -2.9 \\
-2.9 & -3.0 & -2.9 & -2.4 \\
-3.0 & -2.9 & -2.4 & 0.0
\end{array}
$$

$k = 10$

$$
\begin{array}{cccc}
0.0 & -6.1 & -8.4 & 9.0 \\
-6.1 & -7.7 & -8.4 & -8.4 \\
-8.4 & -8.4 & -7.7 & -6.1 \\
-9.0 & -8.4 & -6.1 & 0.0
\end{array}
$$

$k = \infty$

$$
\begin{array}{cccc}
0.0 & -14.0 & -20.0 & -22.0 \\
-14.0 & -18.0 & -20.0 & -20.0 \\
-20.0 & -20.0 & -18.0 & -14.0 \\
-22.0 & -20.0 & -14.0 & 0.0
\end{array}
$$

optimal policy
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:

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

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

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

$$Q^\pi(s, a) > V^\pi(s)$$
Policy improvement theorem

Let $\pi$ and $\pi'$ be any pair of deterministic policies such that, for all $s \in S$,

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s)$$

This means, the policy $\pi'$ must be as good as, or better than, $\pi$. That is, it must obtain greater or equal expected return from all states $s \in S$: 
Policy improvement theorem

\[ V^\pi(s) \leq Q^\pi(s, \pi'(s)) \]
\[ = E_{\pi'} \{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = t \} \]
\[ \leq E_{\pi'} \{ r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_t = t \} \]
\[ = E_{\pi'} \{ r_{t+1} + \gamma E_{\pi'} \{ r_{t+2} + \gamma V^\pi(s_{t+2}) \} \mid s_t = t \} \]
\[ = E_{\pi'} \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \mid s_t = t \} \]
\[ \leq E_{\pi'} \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) \mid s_t = t \} \]
\[ \vdots \]
\[ \leq E_{\pi'} \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \mid s_t = t \} \]
\[ = V^{\pi'}(s). \]
Policy Improvement Cont.

Do this for all states to get a new policy \( \pi' \) that is **greedy** with respect to \( V^\pi \):

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

Then \( V^{\pi'} \geq V^\pi \)
Policy Improvement Cont.

What if $V^{\pi'} = V^\pi$?

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

But this is the Bellman Optimality Equation.

So $V^{\pi'} = V^*$ and both $\pi$ and $\pi'$ are optimal policies.

Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.
Theorem

$$V^\pi'(s) = V^\pi(s) \quad \forall s \in S \quad \Rightarrow \quad \pi' = \pi = \pi^*$$

Proof:

$$V^\pi'(s) = \max_{a \in A(s)} Q^\pi(s, a) = \max_{a \in A(s)} \sum_{s' \in S} P_{ss'}^a \cdot [R_{ss'}^a + \gamma \cdot V^\pi(s')] = \max_{a \in A(s)} \sum_{s' \in S} P_{ss'}^a \cdot [R_{ss'}^a + \gamma \cdot V^\pi'(s')]$$

$$V^\pi'(s) = \max_{a \in A(s)} \sum_{s' \in S} P_{ss'}^a \cdot [R_{ss'}^a + \gamma \cdot V^\pi'(s')]$$

Bellmann Optimality Equation

$$V^\pi'(s) = V^* (s) \quad \Rightarrow \quad \pi' = \pi = \pi^*$$

optimal policy
Policy Improvement - nondeterministic policies

Let $\pi$ and $\pi'$ be any pair of arbitrary policies such that, for all $s \in S$

$$\sum_{a \in A(s)} \pi'(s, a) \cdot Q^\pi(s, a) \geq V^\pi(s)$$

$$\pi' \geq \pi \iff V^{\pi'}(s) \geq V^\pi(s) \quad \forall s \in S$$
Policy Iteration

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

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

3. Policy Improvement
   \( policy-stable \leftarrow true \)
   For each \( s \in \mathcal{S} \):
     \( b \leftarrow \pi(s) \)
     \( \pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')] \)
   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

\[ \pi_0 \]

\[ \pi_1 \]

\[ \pi_2 \]

\[ \pi_3 \]

\[ \pi_4 \]

\[ V_4 \]
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
Value Iteration

Recall the **full policy-evaluation backup**: 

\[ V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^{a} \left[ R_{ss'}^{a} + \gamma V_k(s') \right] \]

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

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[ R_{ss'}^{a} + \gamma V_k(s') \right] \]
Value Iteration Cont.

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

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

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

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

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

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

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

Value estimates

Final policy (stake)
Asynchronous Dynamic Programming

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

A geometric metaphor for convergence of GPI:
Efficiency of Dynamic Programming

• 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 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