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 $V^\pi$:

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

— 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 \textbf{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 $s \in S$:
  $v \leftarrow V(s)$
  $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
  $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
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

$r = -1$ on all transitions
Iterative Policy Eval for the Small Gridworld

\( \pi = \) random (uniform) action choices

<table>
<thead>
<tr>
<th>( k )</th>
<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></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0 0.0 0.0 0.0</td>
<td><img src="image" alt="Random Policy" /></td>
</tr>
<tr>
<td>( k = 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0 -1.0 -1.0 -1.0</td>
<td><img src="image" alt="Greedy Policy w.r.t. ( V_k )" /></td>
</tr>
<tr>
<td>( k = 2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0 -1.7 -2.0 -2.0</td>
<td><img src="image" alt="Greedy Policy w.r.t. ( V_k )" /></td>
</tr>
<tr>
<td>( k = 3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0 -2.4 -2.9 -3.0</td>
<td><img src="image" alt="Greedy Policy w.r.t. ( V_k )" /></td>
</tr>
<tr>
<td>( k = 10 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0 -6.1 -8.4 -9.0</td>
<td><img src="image" alt="Greedy Policy w.r.t. ( V_k )" /></td>
</tr>
<tr>
<td>( k = \infty )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0 -14. -20. -22.</td>
<td><img src="image" alt="Greedy Policy w.r.t. ( V_k )" /></td>
</tr>
</tbody>
</table>
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:

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

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

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

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

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

Then $V^{\pi'} \geq V^\pi$
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 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
   \[ V(s) \in \mathcal{R} \text{ and } \pi(s) \in \mathcal{A}(s) \text{ arbitrarily for all } s \in S \]

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

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

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

\[ \pi(s) = \arg \max_a \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')] \]
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 (GPI):
any interaction of policy evaluation and policy improvement, independent of their granularity.

A geometric metaphor for convergence of GPI:

\[ V \rightarrow V^\pi \]
\[ \pi \rightarrow \text{greedy}(V) \]
\[ V^\pi \rightarrow V \]
\[ \pi^* \rightarrow \text{greedy}(V) \]
\[ \pi \rightarrow V^\pi \]
\[ V^* \]

\[ \pi = \text{greedy}(V) \]
\[ V \approx V^\pi \]
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 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