Reinforcement Learning

Dynamic Programming

Stefano V. Albrecht
27 January 2023
Lecture Outline

- Policy iteration
- Iterative policy evaluation
- Policy improvement
- Value iteration
- Asynchronous and generalised DP
Recap: Markov Decision Process

Finite MDP consists of:

- Finite sets of states \( S \), actions \( A \), rewards \( R \)
- Environment dynamics \( p(s', r|s, a) \)
- Optimal policy \( \pi_* \) maximises expected return for all \( s \in S \):

\[
\max_{\pi} \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid S_t = s \right]
\]
Dynamic programming (DP) is a family of algorithms to compute optimal policy.

DP algorithms use Bellman equations as operators:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_\pi(s') \right]$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_\pi(s') \right]$$

⇒ Assumes knowledge of all components of MDP ($\mathcal{S}, \mathcal{A}, \mathcal{R}, p(s', a|s, a)$)
The basic DP algorithm is policy iteration which alternates between two phases:

- **Policy evaluation**: compute $v_\pi$ for current policy $\pi$
- **Policy improvement**: make policy $\pi$ greedy wrt $v_\pi$

\[
\begin{align*}
&\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*
\end{align*}
\]

Process converges to optimal policy $\pi_*$
Recall: Bellman equation for $v_\pi$ is system of linear equations

\[
v_\pi(s_1) = \sum_a \pi(a|s_1) \sum_{s',r} p(s', r|s_1, a) \left[ r + \gamma v_\pi(s') \right]
\]

\[
v_\pi(s_2) = \sum_a \pi(a|s_2) \sum_{s',r} p(s', r|s_2, a) \left[ r + \gamma v_\pi(s') \right]
\]

\[\vdots\]

\[
v_\pi(s_n) = \sum_a \pi(a|s_n) \sum_{s',r} p(s', r|s_n, a) \left[ r + \gamma v_\pi(s') \right]
\]

Could use this for policy evaluation step, but expensive

- Gauss elimination (de facto standard) has time complexity $O(n^3)$
Iterative Policy Evaluation

We can use Bellman equation as operator to \textit{iteratively} compute $v_\pi$:

- Initialise $v_0(s) = 0$
- Then repeatedly perform updates:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[ r + \gamma v_k(s') \right] \quad \text{for all } s \in S$$

- Sequence converges to fixed point $v_\pi$, so stop when no more changes to $v_k$

\textit{Updating estimates based on other estimates is called bootstrapping}
Iterative Policy Evaluation

Input $\pi$, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in S^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
Example: Gridworld

- States: cell location in grid; grey squares are terminal
- Actions: move north, south, east, west
- Rewards: -1 until terminal state reached (recall: absorbing state, reward 0)
- Undiscounted: $\gamma = 1$

$R_t = -1$
on all transitions
Example: Gridworld

Evaluating the uniform random policy: $\pi(a|s) = 0.25$ for all $s, a$

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</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>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-1.7</td>
<td>-2.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>-1.7</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>
</tr>
<tr>
<td>-2.0</td>
<td>-2.0</td>
<td>-1.7</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-6.1</td>
<td>-8.4</td>
<td>-9.0</td>
</tr>
<tr>
<td>-6.1</td>
<td>-7.7</td>
<td>-8.4</td>
<td>-8.4</td>
</tr>
<tr>
<td>-8.4</td>
<td>-8.4</td>
<td>-7.7</td>
<td>-6.1</td>
</tr>
<tr>
<td>-9.0</td>
<td>-8.4</td>
<td>-6.1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>-14.0</td>
<td>-20.0</td>
<td>-22.0</td>
</tr>
<tr>
<td>-14.0</td>
<td>-18.0</td>
<td>-20.0</td>
<td>-20.0</td>
</tr>
<tr>
<td>-20.0</td>
<td>-20.0</td>
<td>-18.0</td>
<td>-14.0</td>
</tr>
<tr>
<td>-22.0</td>
<td>-20.0</td>
<td>-14.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Why does the sequence $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots$ converge to $v_\pi$?

⇒ Because Bellman operator is a contraction mapping

**Contraction Mapping**

Operator $f$ on $|| \cdot ||$-normed vector space $\mathcal{X}$ is a $\gamma$-contraction, for $\gamma \in [0, 1)$, if for all $x, y \in \mathcal{X}$:

$$||f(x) - f(y)|| \leq \gamma ||x - y||$$

- **Banach fixed-point theorem**: repeated application of $f$ converges to a unique fixed point in $\mathcal{X}$ (if $\mathcal{X}$ complete)
Rewrite Bellman equation:

\[
v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_\pi(s') \right]
\]

\[
= \sum_{a,s',r} \pi(a|s) p(s',r|s,a) r + \sum_{a,s',r} \pi(a|s) p(s',r|s,a) \gamma v_\pi(s')
\]

As operator over vector \( v \):

\[
f^\pi(v) = r^\pi + \gamma T^\pi v
\]

where \( r^\pi_s = \sum_{a,s',r} \pi(a|s) p(s',r|s,a) r \) and \( T^\pi_{s,s'} = \sum_{a,r} \pi(a|s) p(s',r|s,a) \).
Consider the max-norm:

$$||x||_\infty = \max_i |x_i|$$

Bellman operator is a $\gamma$-contraction under max-norm:

$$||f^\pi(v) - f^\pi(u)||_\infty = ||(r^\pi + \gamma T^\pi v) - (r^\pi + \gamma T^\pi u)||_\infty$$

$$= \gamma ||T^\pi(v - u)||_\infty \quad (Why?)$$

$$\leq \gamma ||v - u||_\infty$$

- Thus, Bellman operator converges to a unique fixed point
- By definition, $v_\pi$ is fixed point of Bellman equation: $v_\pi = f^\pi(v_\pi)$
  $$\Rightarrow$$ Hence, Bellman operator converges to $v_\pi$
Once we have $v_\pi$, we improve $\pi$ by making it greedy wrt $v_k$:

$$\pi'(s) \doteq \arg \max_a q_\pi(s, a)$$

$$= \arg \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')]$$

For all $s \in S$.

This works because of...
Policy Improvement Theorem

Let $\pi$ and $\pi'$ be policies such that for all $s$:

$$\sum_a \pi'(a|s) q_\pi(s,a) \geq \sum_a \pi(a|s) q_\pi(s,a)$$

$$= v_\pi(s)$$

Then $\pi'$ must be as good as or better than $\pi$:

$$\forall s : v_{\pi'}(s) \geq v_\pi(s)$$
Policy Improvement Theorem – Proof Sketch

\[ v_\pi(s) \leq q_\pi(s, \pi'(s)) \]  
\[ = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \]  
\[ = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \]  
\[ \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \quad \text{(by premise)} \]  
\[ = \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \]  
\[ = \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \]  
\[ \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s] \]  
\[ \cdots \]  
\[ \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \]  
\[ = v_{\pi'}(s) \]
What if greedy policy $\pi'$ has not changed from $\pi$ after policy improvement?

Then $v_{\pi'} = v_{\pi}$ (why?) and it follows for all $s \in S$:

\[
v_{\pi'}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = a]
\]

(by greedy construction)

\[
= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a]
\]

($v_{\pi'} = v_{\pi}$)

\[
= \max_a \sum_{s', r} p(s', r | s, a) \left[ r + \gamma v_{\pi'}(s') \right]
\]

\[
= v_*(s)
\]
What if greedy policy $\pi'$ has not changed from $\pi$ after policy improvement?

Then $v_{\pi'} = v_\pi$ (why?) and it follows for all $s \in S$:

$$v_{\pi'}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \text{ (by greedy construction)}$$

$$= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \text{ (} v_{\pi'} = v_\pi \text{)}$$

$$= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi'}(s')]$$

$$= v_*(s) \quad \Rightarrow \pi' \text{ (and } \pi \text{) is optimal and policy iteration is complete!}$$
Policy Iteration

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

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

3. Policy Improvement
   \( policy-stable \leftarrow true \)
   For each \( s \in S \):
   \[ a \leftarrow \pi(s) \]
   \[ \pi(s) \leftarrow \arg \max_a \sum_{s', r} p(s', r | s, a) \left[ r + \gamma V(s') \right] \]
   If \( a \neq \pi(s) \), then \( policy-stable \leftarrow false \)
   If \( policy-stable \), then stop and return \( V \) and \( \pi \); else go to 2
Example: Jack’s Car Rental

- Two car rental locations
- Cars are requested and returned randomly based on a distribution (see book)
- States: \((n_1, n_2)\) — \(n_i\) is number of cars at location \(i\) (max 20 each)
- Actions: number of cars moved from one location to other (max 5) (positive is from location 1 to 2, negative is from 2 to 1)
- Rewards:
  + $10 per rented car in time step
  − $2 per moved car in time step
- \(\gamma = 0.9\)
Example: Jack’s Car Rental
Value Iteration

Iterative policy evaluation may take many sweeps $v_k \to v_{k+1}$ to converge

Do we have to wait until convergence before policy improvement?
Value Iteration

Iterative policy evaluation uses Bellman equation as operator:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_k(s') \right] \quad \text{for all } s \in \mathcal{S}$$

Value iteration uses *Bellman optimality equation* as operator:

$$v_{k+1}(s) = \max_a \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_k(s') \right] \quad \text{for all } s \in \mathcal{S}$$

- Combines one sweep of iterative policy evaluation and policy improvement
- Sequence converges to optimal policy
  *(can show that Bellman optimality operator is $\gamma$-contraction)*
Value Iteration

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

Repeat
\[ \Delta \leftarrow 0 \]
For each $s \in S$:
\[ v \leftarrow V(s) \]
\[ V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[ r + \gamma V(s') \right] \]
\[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
\[ \pi(s) = \arg \max_a \sum_{s',r} p(s', r|s, a) \left[ r + \gamma V(s') \right] \]
Asynchronous Dynamic Programming

DP methods so far perform exhaustive sweeps:
Policy evaluation and improvement for all \( s \in \mathcal{S} \) → prohibitive if state space large!

Asynchronous DP methods evaluate and improve policy on subset of states:

- Gives flexibility to choose best states to update
  → e.g. random states, recently visited states (real-time DP)

- Can perform updates *in parallel* on multiple processors

- Still guaranteed to converge to optimal policy if all states in \( \mathcal{S} \) are updated infinitely many times in the limit
Generalised Policy Iteration

DP methods can perform policy evaluation and improvement at different granularity:
- full sweeps > single sweep > single states
Required:

- RL book, Chapter 4 (4.1–4.7)
  (Iterative Policy Evaluation proof from slides not examined)

Optional:

- *Dynamic Programming and Optimal Control*
  by Dimitri P. Bertsekas
  
  http://www.athenasc.com/dpbook.html
  
  Search on Google ...