Reinforcement Learning

Dynamic Programming

Stefano V. Albrecht, Michael Herrmann

26 January 2024

THE UNIVERSITY of EDINBURGH
informatics

Lecture Outline

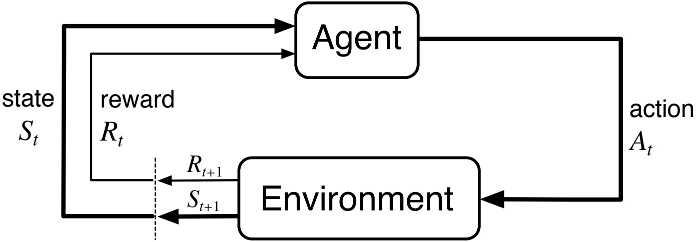
- Policy iteration
- Iterative policy evaluation
- Policy improvement
- Value iteration
- Asynchronous and generalised DP

Recap: Markov Decision Process

Finite MDP consists of:

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

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,|\, S_t = s\right]$$



Dynamic Programming

Dynamic programming (DP) is a family of algorithms to compute optimal policy

DP algorithms use Bellman equations as operators:

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

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

$\Rightarrow$ Assumes knowledge of all components of MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p(s', r|s, a))$

Policy Iteration

The basic DP algorithm is policy iteration which alternates between two phases:

- **Policy evaluation:** compute v_π for current policy π
- **Policy improvement:** make policy π *greedy* wrt v_π

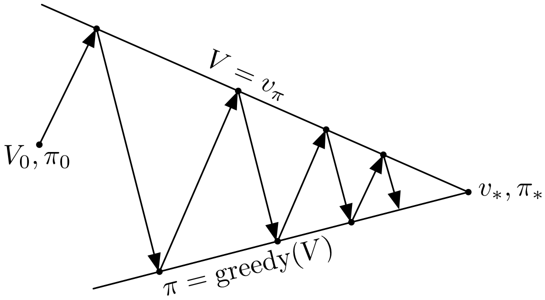
$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

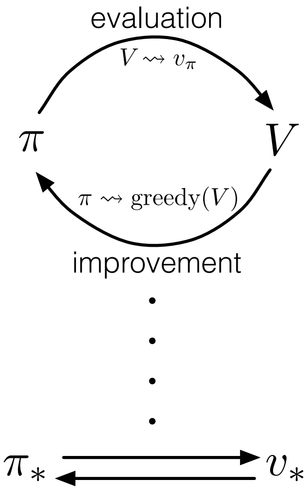
Process converges to optimal policy π_*

Generalised Policy Iteration

DP methods can perform policy evaluation and improvement at different granularity:

- full sweeps $>$ single sweep $>$ single states





Policy Evaluation

Recall: Bellman equation for v_π 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 *iteratively* compute v_π:

- Initialise $v_0(s) = 0$
- Then perform updates $v_k \to v_{k+1}$ for $k = 0, 1, 2, \ldots$:

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

- Sequence converges to fixed point v_π, so stop when no more changes to v_k

Updating estimates based on other estimates is called bootstrapping

Iterative Policy Evaluation

Input π, the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\quad \Delta \leftarrow 0$

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

$\qquad v \leftarrow V(s)$

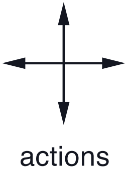
$\qquad V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

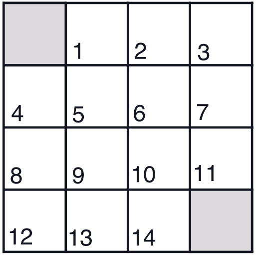
$\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$

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

Output $V \approx v_\pi$

Example: Gridworld





$$R_t = -1$$

on all transitions

- 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: $\gamma = 1$

Example: Gridworld

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

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

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

$k = 10$

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

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

[Extra] Iterative Policy Evaluation – Convergence Proof (1/3)

Why does the sequence $v_0 \to v_1 \to v_2 \to \cdots$ converge to v_π?

$\Rightarrow$ Because Bellman operator is a contraction mapping

Contraction Mapping

Operator f on $||\cdot||$–normed vector space $\mathcal{X}$ is a γ-*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)

[Extra] Iterative Policy Evaluation – Convergence Proof (2/3)

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 $r_s^\pi = \sum_{a,s',r} \pi(a|s)\, p(s',r|s,a)\, r$ and $T_{s,s'}^\pi = \sum_{a,r} \pi(a|s)\, p(s',r|s,a)$

[Extra] Iterative Policy Evaluation – Convergence Proof (3/3)

Consider the max-norm:

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

Bellman operator is a γ-contraction under max-norm:

$$\begin{aligned} ||f^\pi(v) - f^\pi(u)||_\infty &= ||(r^\pi + \gamma T^\pi v) - (r^\pi + \gamma T^\pi u)||_\infty \\ &= \gamma ||T^\pi (v - u)||_\infty \\ &\leq \gamma ||v - u||_\infty \end{aligned}$$

(Why?)

- Thus, Bellman operator converges to a unique fixed point
- By definition, v_π is fixed point of Bellman equation: $v_\pi = f^\pi(v_\pi)$

  $\Rightarrow$ Hence, Bellman operator converges to v_π

Policy Improvement

Once we have v_π, we *improve* π by making it greedy wrt v_k:

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

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

For all $s \in \mathcal{S}$.

This works because of...

Policy Improvement Theorem

Let π and π' be policies such that for all s:

$$\begin{aligned} \sum_a \pi'(a|s)\, q_\pi(s,a) &\geq \sum_a \pi(a|s)\, q_\pi(s,a) \\ &= v_\pi(s) \end{aligned}$$

Then π' must be as good as or better than π:

$$\forall s : v_{\pi'}(s) \geq v_\pi(s)$$

Policy Improvement Theorem – Proof Sketch

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) && \text{(here for deterministic policies)} \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s, A_t = \pi'(s)\right] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \,|\, S_t = s\right] && \text{(by premise)} \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi(S_{t+2}) \,|\, S_{t+1}, A_{t+1} = \pi'(S_{t+1})\right] \,|\, S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \,|\, S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \,|\, S_t = s\right] \\
&\dots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \,|\, S_t = s\right] \\
&= v_{\pi'}(s)
\end{aligned}
$$

Policy Improvement

What if greedy policy π' has not changed from π after policy improvement?

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

$$\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s, A_t = a] && \text{(by greedy construction)} \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \,|\, S_t = s, A_t = a] && (v_{\pi'} = v_\pi) \\
&= \max_a \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma v_{\pi'}(s') \right] \\
&= v_*(s)
\end{aligned}$$

Policy Improvement

What if greedy policy π' has not changed from π after policy improvement?

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

$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] && \text{(by greedy construction)} \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] && (v_{\pi'} = v_\pi) \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_{\pi'}(s')\right] \\
&= v_*(s) && \Rightarrow \pi' \text{ (and } \pi\text{) is optimal and policy iteration is complete!}
\end{aligned}
$$

Policy Iteration

See Tutorial 3 & 5

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 $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

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

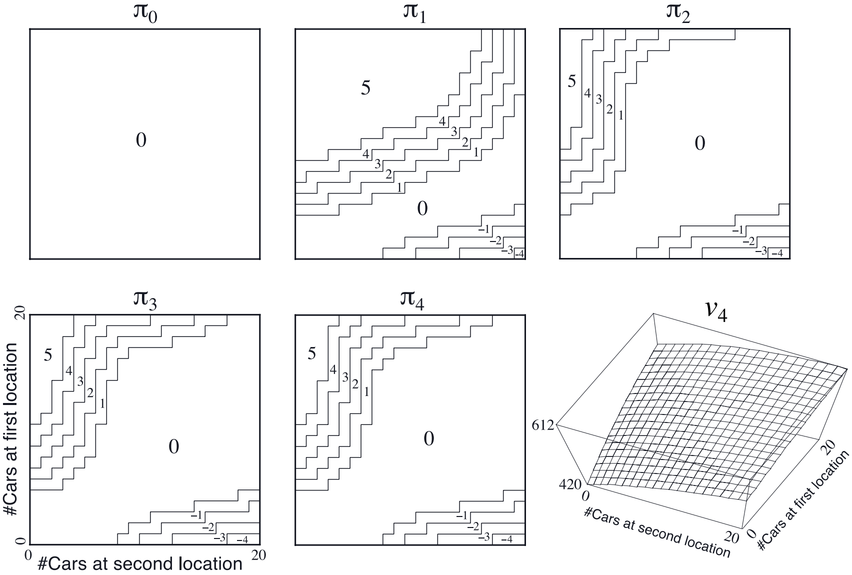
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - $a \leftarrow \pi(s)$
   - $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   - If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ $false$

   If *policy-stable*, then stop and return V and π; 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
- $\gamma = 0.9$

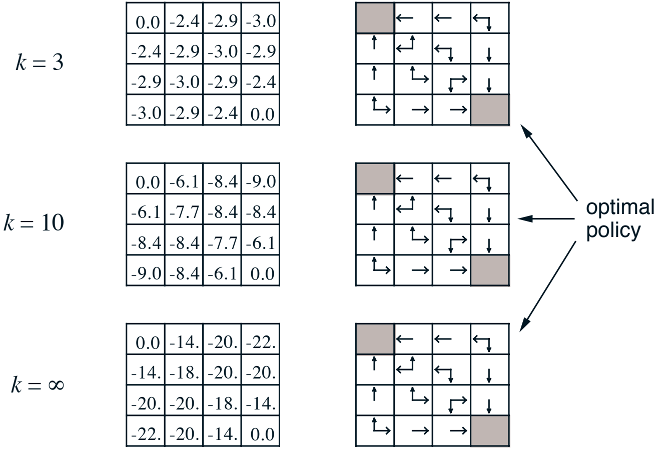
Example: Jack's Car Rental



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 γ-contraction)

Value Iteration

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

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

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

Asynchronous Dynamic Programming

DP methods so far perform exhaustive *sweeps*:

Policy evaluation and improvement for all $s \in \mathcal{S}$ $\Rightarrow$ prohibitive if state space large!

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

- Gives flexibility to choose best states to update

  $\Rightarrow$ 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

Reading

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 ...