Introduction to reinforcement learning

Urtzi Ayesta and Matthieu Jonckheere

CNRS-IRIT & CNRS-LAAS

Outline

- Principle of optimality
- Dynamic Programming
- Bellman Expectation as an Operator: Iterative Policy Evaluation
- Bellman Optimality Equation as an Operator: Value Iteration
- Policy Iteration

Principle of optimality

The tail of an optimal policy is optimal for the "tail" problem

Principle of optimality

The tail of an optimal policy is optimal for the "tail" problem

$$V_0^{\pi}(s) \;\; = \;\; \mathbb{E}_{\pi}(\sum_{t=1}^{T} R_t | S_0 = s)$$

Let $\pi^* = (\pi_1, \ldots, \pi_T)$ be the optimal policy. Then, the tail policy $(\pi_k, \ldots, \pi_T)$ is optimal for the tail cost

$$V_k^{\pi}(s) \;\; = \;\; \mathbb{E}_{\pi}(\sum_{t=k+1}^{T} R_t | S_k = s)$$

Principle of optimality (cont.)

Imagine action a, which yields reward $r(s, a)$. The best we can do from now on is

$$r(s, a) + \sum_{s'} p(s'|s, a) V^{T-1}(s')$$

Principle of optimality (cont.)

Imagine action a, which yields reward $r(s,a)$. The best we can do from now on is

$$r(s,a) + \sum_{s'} p(s'|s,a) V^{T-1}(s')$$

Then, the best action is

$$V^T(s) = \max_a \left(r(s,a) + \sum_{s'} p(s'|s,a) V^{T-1}(s') \right)$$

What is Dynamic Programming

Dynamic sequential or temporal component to the problem

Programming : optimising a ”program”, i.e., a policy

- A method to solve complex problems by breaking them down into subproblems

Planning by Dynamic Programming

Dynamic programming assumes full knowledge of MDP

It is used for planning in an MDP

For prediction:

- Input $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
- output: value function V_π

Or for control:

- Input $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
- output: optimal value function π_*

Bellman Expectation Equation for V_π

$$V_\pi(s) = \sum_a \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right)$$

in case $\pi(a|s) = 1, \forall s$:

$$V_\pi(s) = \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right)$$

Bellman Optimality Equation for V_*

$$V_*(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_*(s') \right)$$

Other Applications of Dynamic Programming

- Scheduling algorithms
- String algorithms (e.g. sequence alignment)
- Graph algorithms (e.g. shortest path algorithms)
- Graphical models (e.g. Viterbi algorithm)
- Bioinformatics (e.g. lattice models)

Contraction Mapping Theorem

Definition: γ-contraction

We say that an operator T is a γ-contraction if

$$||T(u) - T(v)||_\infty \leq \gamma ||u - v||_\infty$$

Contraction Mapping Theorem

Definition: γ-contraction

We say that an operator T is a γ-contraction if

$$||T(u) - T(v)||_\infty \leq \gamma ||u - v||_\infty$$

Theorem: Contraction Mapping Theorem

Let $\mathcal{V}$ be a complete metric space, en let $T(\cdot)$ be a γ-contraction. Then:

- $T(\cdot)$ converges to a unique fixed point
- Convergence rate is exponential in γ

Bellman Expectation is a γ-contraction

Define the *Bellman* expectation operator T^π

$$T^\pi(V) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi V$$

$$T^\pi(V)(s) = \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right)$$

Bellman Expectation is a γ-contraction

Define the *Bellman* expectation operator T^π

$$T^\pi(V) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi V$$

$$T^\pi(V)(s) = \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right)$$

$$\begin{aligned} ||T(U) - T(V)||_\infty &= ||(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi U) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi V)||_\infty \\ &= ||\gamma \mathcal{P}^\pi (U - V)||_\infty \\ &\leq ||\gamma \mathcal{P}^\pi|| ||(U - V)||_\infty \\ &\leq \gamma ||U - V||_\infty \end{aligned}$$

Iterative Policy Evaluation

Problem : Evaluate a given policy π

- Solution: Iterations over Bellman equation
- At each iteration $k + 1$, update $V_{k+1}(s)$ from state $V_k(s')$

Iterative Policy Evaluation

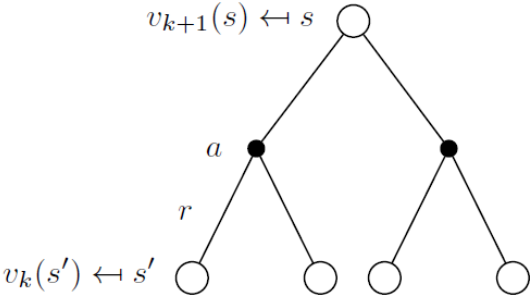
Problem : Evaluate a given policy π

- Solution: Iterations over Bellman equation
- At each iteration $k+1$, update $V_{k+1}(s)$ from state $V_k(s')$

Convergence of Iterative Policy Evaluation

- The Bellman expectation operator T^π has a unique fixed point
- V_π is a fixed point of T^π
- Iterative policy evaluation converges on V_π

Iterative Policy Evaluation



$$V_{k+1}(s) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_k(s')$$

Bellman Optimality Equation

Proposition:

V^* is the unique solution of V^*:

$$V_*(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_*(s') \right)$$

Any stationary policy π^* that satisfies

$$\pi^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_*(s') \right)$$

is an optimal policy

Bellman Optimality Equation (proof sketch)

Consider the operator T_*

$$T_*(V) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right)$$

and show that is a contraction mapping.

Value Iteration

Starting with arbitrary V_0, define recursively $\forall s$:

$$V_{n+1}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_n(s') \right)$$

Then $\lim_{n \to \infty} V_n = V^*$, where the rate of convergence is exponential

Value Iteration

Starting with arbitrary V_0, define recursively $\forall s$:

$$V_{n+1}(s) = \max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V_n(s') \right)$$

Then $\lim_{n \to \infty} V_n = V^*$, where the rate of convergence is exponential

Retrieve the optimal policy π^* by:

$$\pi^*(s) \in \text{argmax}_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V_*(s') \right)$$

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

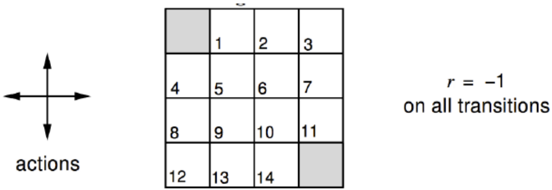
0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

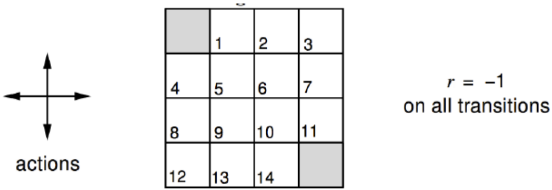
0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Evaluation a random policy in a small gridworld



Evaluation a random policy in a small gridworld



- Undiscounted $\gamma = 1$
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

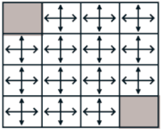
$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 1/4$$

Iterative Policy Evaluation in Small Gridworld

v_k for the Random Policy | Greedy Policy w.r.t. v_k

$k = 0$

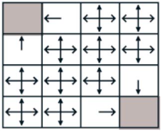
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

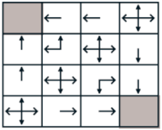
$k = 1$

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



Iterative Policy Evaluation in Small Gridworld (2)

$k = 3$

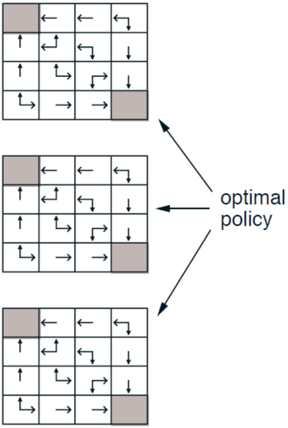
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



How to improve a policy: Policy Iteration

Given a policy π, let V^π be its value function. We define the $\bar{\pi} = \text{greedy}(V^\pi)$ to be the greedy policy with respect to V^π

$$\bar{\pi}(s) \in \text{argmax}_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right) \qquad \forall s.$$

How to improve a policy: Policy Iteration

Given a policy π, let V^π be its value function. We define the $\bar{\pi} = \text{greedy}(V^\pi)$ to be the greedy policy with respect to V^π

$$\bar{\pi}(s) \in \text{argmax}_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right) \qquad \forall s.$$

Proposition: Policy Improvement

$V^{\bar{\pi}(s)} \geq V^{\pi(s)}$ and equality holds if and only if π is optimal

Sketch of proof

Recall the operator T_* as

$$T_*(V) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right)$$

$$V^\pi = T_\pi V^\pi \leq T_*(V^\pi) = T_{\bar{\pi}}(V^\pi)$$

Sketch of proof

Recall the operator T_* as

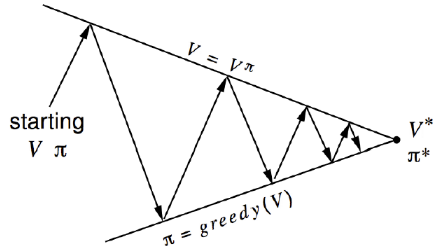
$$T_*(V) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right)$$

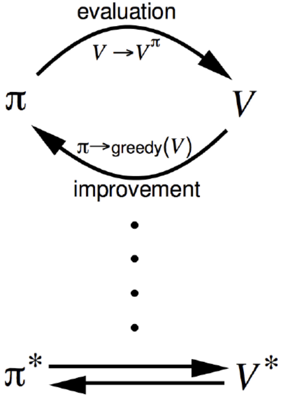
$$V^\pi = T_\pi V^\pi \leq T_*(V^\pi) = T_{\bar{\pi}}(V^\pi)$$

It can be shown that $T_{\bar{\pi}}$ is a monotone operator, i.e., $V_1 \leq V_2$ implies $T_{\bar{\pi}}(V_1) \leq T_{\bar{\pi}}(V_2)$. Repeatedly applying $T_{\bar{\pi}}$ to the above inequality

$$V^\pi \leq T_{\bar{\pi}} V^\pi \leq (T_{\bar{\pi}})^2 V^\pi \leq \cdots \leq \lim_{n \to \infty} (T_{\bar{\pi}})^n V^\pi = V^{\bar{\pi}}$$

Policy Iteration Algorithm





Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

Policy Iteration Algorithm (cont.)

Pseudocode

- 0. Initialization: choose a policy π_0

For $k = 0, 1, \ldots$:

- Policy Evaluation: Compute V_{π_k} for example

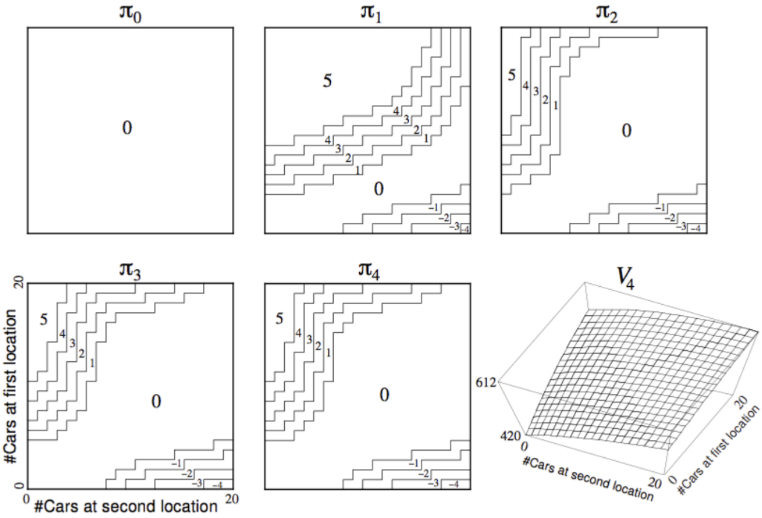
$$V_{\pi_k} = (I - \gamma \mathcal{P}^{\pi_k})^{-1} \mathcal{R}^{\pi_k}$$

- Policy improvement: Compute $\pi_{k+1} = \text{greedy}(V_{\pi_k})$
- Stop if $V_{\pi_{k+1}} = V_{\pi_k}$, else repeat

Car's rental

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars between locations overnight
- Reward: 10 for each car rented (must be available)
- Transitions: Cars returned and requested randomly
  - Poisson distribution, n returns/requests with prob $p(n)$
  - $1st$ location: average requests $= 3$, average returns $= 3$
  - $2nd$ location: average requests $= 4$, average returns $= 2$

Cars's rental (cont.)



Synchronous Dynamic Programming Algorithm

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectaion Equation	Iter. PI
Control	Bellman Expectaion Equation + Greedy PI	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $V_\pi(s)$ or $V_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$, complexity $O(m^2n^2)$ per iteration

Asynchronous Dynamic Programming

- Synchronous backups $\implies$ all states are backed up in parallel
- Asynchronous DP backs up states individually, in any order
- For each selected state, apply the appropriate backup
- Can significantly reduce computation
- Guaranteed to converge if all states continue to be selected

Asynchronous Dynamic Programming

Three simple approaches

- In-place dynamic programming
- Prioritised sweeping
- Real-time DP

In-place dynamic programming

In-place value iteration only stores one copy of value function.

$$V(S) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right)$$

Prioritized Sweeping

Use magnitude of Bellman error to guide state selection, e.g

$$\left| \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right) - V(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

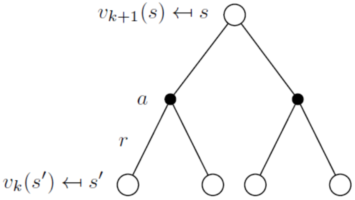
Real-Time Dynamic Programming

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step S_t, A_t, R_{t+1}
- Backup the state S_t

$$V(s_t) \longleftarrow \max_{a \in \mathcal{A}} \left(r(s_t, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s_t, a) V(s') \right)$$

Full-width backups

- DP uses full-width backups
- For each backup (sync or async)
  - Every successor state and action is considered
  - Using knowledge of the MDP transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's curse of dimensionality
  - Number of states $n = |\mathcal{S}|$ grows exponentially with number of state variables
- Even one backup can be too expensive



Average Reward MDP

An ergodic MDP has an average reward per time-step g^π that is independent of start state

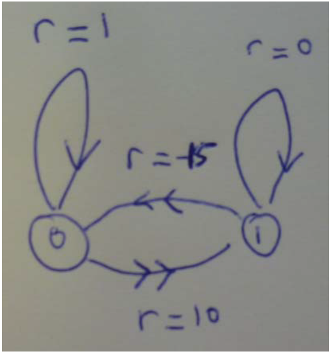
$$g^\pi = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} R_t\right]$$

We define the value function as

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\left[\sum_{k=1}^{\infty}(R_{t+k} - g^\pi)|S_t = s\right] \\
&= \mathbb{E}_\pi\left[(R_{t+1} - g^\pi) + V_\pi(S_{t+1})|S_t = s\right] \\
&= r(s,a) - g^\pi + \sum_{s'} p(s'|s,a)V_\pi(s')
\end{aligned}$$

Exercise: Infinite Horizon MDP

Note: The reward from state 1 to 0 under action $<<$ is -15

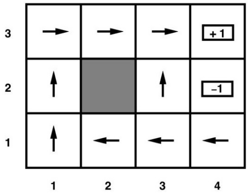


What is the optimal policy (for total discounted reward) for various values of γ?

- Solve the optimality equation it by value iteration
- Characterize the optimal policy using policy iteration

Help: You might use Value Iteration (slide $15/35$), or policy iteration (slide $23/35$)

Exercise: Grid World



Optimal policy when R(s) = -0.04 for every non-terminal state

Find the optimal policy using the Policy Iteration Algorithm (slide 32/45).
Deadline: 14 Mai