Markov Decision Processes

Joseph Le Roux

06/07/26

Notes

Outline

- Definitions
- Optimal Policy
- Iterative Algorithms for Optimal Policy
- Limitations – Bridge to RL

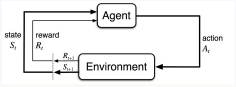
Notes

1 Definitions

Notes

1 The Agent-Environment Model

Fundamental model for autonomous agents (robots)



At each time t:

- the agent witnesses the environment ...
- ... which is in state s_t.
- the agent performs an action a_t...
- ... which transforms the environment to state s_{t+1} and gives reward r_{t+1}, and so on...

The agent will generate trajectories from initial state s_0:

- $s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots$
- a sequence of *state*, *action*, *reward* ... *state*

Notes

1 The Agent-Environment Model

Example: the *pick-and-place* robot (Sutton&Barto)

y

Use the Agent-Environment Model to represent an articulated robotic arm whose job is to pick up objects and place them on a table. The task consists in adjusting the motor speeds placed on articulation joints:

- States : the angular values the arm's joints and the motor speeds (*an observation of the system*)
- Actions : voltage to apply on each motor (*to change the state of the system*)
- Rewards : $+1$ for each object picked up and correctly placed, otherwise 0 ($+1$ *to promote solving the task*)

States are continuous, Actions continuous/discrete

Notes

1 The Agent-Environment Model

Example : GridWorld

- states: $\{1, 2, \ldots, 12\}$ from left to right, bottom up, initial state is 1.
- actions: move in 4 directions (might not succeed if walls, trees...)
- final state: the throne (12), you win!
- final state: lava (8) hurts, you lose!
- reward to reach throne: $+10$, reward to reach lava: -100
- reward to reach any other state: -1 (any idea why?)

How to model errors (wrong readings, wrong actions, unexpected problems...) ?

Use transition probabilities: in s when agent performs a several outcomes are possible!

Notes

1 Markov Decision Process

MDP $M = (\mathcal{S}, \mathcal{A}, \delta, \rho, H, \gamma)$:

- $\mathcal{S}$ set of states (one initial, several terminals); $\mathcal{A}$ set of actions
- $\delta : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow [0;1]$ transition distribution:

$$\delta(s, a, s') = p_\delta(S_{t+1} = s' | S_t = s, A_t = a)$$

(aka *Markovian* dynamics) models uncertainty of action results

- $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ reward function $\rho(s, a, s')$ when robot enters state s' from previous state s and action a.
- $H \in \mathbb{N} \cup \{\infty\}$ max numbers of actions in trajectories (horizon)
- $\gamma \in [0;1]$ discount gives less weight to farther actions (cf. gains later in this lecture)

Markovian

We have the markovian property for transitions $\delta(s, a, s') = p_\delta(S_{t+1} = s' | S_t = s, A_t = a)$: given the present, the future does not depend on the past.

Deterministic MDP

if $\forall (s, a)$ there exists one s' such that $\delta(s, a, s') = 1$ (or equivalently δ is a function $\mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S}$)

Notes

1 MDP Example: GridWorld (Deterministic)

$$\text{MDP} = \left(\{1, \ldots, 12\}, \{\uparrow, \downarrow, \leftarrow, \rightarrow\}, T, R, \infty, 0.9\right)$$

Transitions : deterministic

$$\delta(1, \uparrow, s) = \begin{cases} 1 & \text{if } s = 5, \\ 0 & \text{otherwise.} \end{cases} \qquad \delta(1, \leftarrow, s) = \begin{cases} 1 & \text{if } s = 1, \\ 0 & \text{otherwise.} \end{cases}$$

... continue for all states and actions (exercise) ...

Rewards

- when entering throne +10: $R(s, a, 12) = 10 \quad \forall s, a$
- when entering lava -100 : $R(s, a, 8) = -100 \quad \forall s, a$
- otherwise -1: $\rho(s, a, s') = -1 \quad \forall a, s$ and $s' \notin \{8, 12\}$

Remark

Final states are absorbing: trajectories stop when we reach them.

Notes

1 MDP Example: GridWorld (Probabilistic) (1)

$$\text{MDP} = \left(\{1, \dots, 12\}, \{\uparrow, \downarrow, \leftarrow, \rightarrow\}, T, R, \infty, 0.9\right)$$

Transitions : an example of probabilistic dynamics

Let us assume that we know that when the robot wants to go in one direction, it only ends in the target cell $1/3$ of the time. Otherwise it ends as if it had moved in a perpendicular direction (50/50 between the two perpendiculars)

$$\delta(7, \rightarrow, s') = \begin{cases} 1/3 & \text{if } s'=8, \\ 1/3 & \text{if } s'=11, \\ 1/3 & \text{if } s'=3, \\ 0 & \text{otherwise.} \end{cases} \qquad \delta(1, \leftarrow, s') = \begin{cases} 2/3 & \text{if } s'=1, \\ 1/3 & \text{if } s'=5, \\ 0 & \text{otherwise.} \end{cases}$$

... for all (non-terminal) states/directions ...

Notes

1 MDP and Expectations: Expected Reward (1)

One question that we will ask ourselves a lot: How much immediate reward can a robot get on average when it is in state s and performing action a?

Expected Reward (given state and action)

average reward at time $t+1$ given current state and action at time t:

$$\mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \mathbb{E}_{s' \sim p_\delta(\cdot|s,a)}[R_{t+1} = \rho(s, a, s')] \quad (1)$$

$$= \sum_{s'} p_\delta(s'|s, a)\rho(s, a, s') \quad (2)$$

$$= \sum_{s'} \delta(s, a, s')\rho(s, a, s') \quad (3)$$

In other words: fix s, a and sum all possible rewards weighted by transition probabilities

For instance, on probabilistic grid world:

- $\mathbb{E}[R|1, \uparrow] = \frac{1}{3}(-1) + \frac{1}{3}(-1) + \frac{1}{3}(-1) = -1$
- $\mathbb{E}[R|7, \uparrow] = \frac{1}{3}(-1) + \frac{1}{3}(-1) + \frac{1}{3}(-100) = -34$
- $\mathbb{E}[R|7, \leftarrow] = \frac{1}{3}(-1) + \frac{1}{3}(-1) + \frac{1}{3}(-1) = -1$

Notes

1 MDP Example: recycling robot (Sutton&Barto) (1)

READ AT HOME

This robot must collect cans in bins placed in an office building. It must move, search cans and pick them up.

States

The robot functions on a rechargeable battery which has 3 distinct states $\mathcal{S} = \{\texttt{full}, \texttt{low}, \texttt{empty}\}$. Initial state is `full`, and state `empty` is terminal.

Actions

At each time robot can perform one of the following actions (set $\mathcal{A}$):

1. `search` actively for can to recycle
2. `recharge` its battery
3. `power` to search (but less actively) in order to save power

Notes

1 MDP Example: recycling robot (Sutton&Barto) (2)

Dynamics

- `search` with `full` battery will leave it `full` with probability α, or leave it `low` with probability $1 - \alpha$.
- `search` with `low` battery will leave it `low` with prob. β or `empty` with prob $1 - \beta$.
- `power` leaves the robot in the same state with prob 1.
- `recharge` makes the robot go from `low` to `full` with prob 1.

Rewards

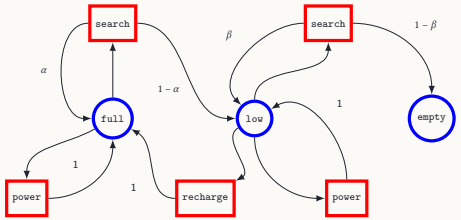
- When battery gets `empty`, robot must be rescued: a negative reward of -1000 is given.
- When robot does `search`, reward is r_s.
- When robot does `power`, reward is r_p (with $r_p \leq r_s$)
- When robot does `recharge`, reward is 0.

Notes

1 MDP Example: recycling robot (Sutton&Barto) (3)

Zero and unreachable $p_\delta(s' \mid s, a)$ are ignored:

s	a	s'	$p\delta(s' \mid s, a)$	$\rho(s, a, s')$
full	search	full	α	r_s
full	search	low	$1 - \alpha$	r_s
low	search	low	β	r_s
low	search	empty	$1 - \beta$	-1000
full	power	full	1	r_p
low	power	low	1	r_p
low	recharge	full	1	0



Notes

1 MDP: Scoring Trajectories and Gains

Trajectories : What has been done by our agent?

From s_0, sequence of action, reward, state until final state: $\tau = s_0, a_0, r_1, s_1, a_1, r_2, \ldots, r_F, s_F$ with $F \leq H$

Evaluate trajectories?

- *good* trajectory accumulates more rewards
- *bad* trajectory accumulates less rewards

Problem : *not useful*, what the robot should do when something goes wrong?

$\Rightarrow$ Evaluate states instead!

A state is *good* if after visiting it, we get more rewards. Define reward to-go for s_t in τ:

$$g_t = r_{t+1} + r_{t+2} + \cdots + r_F = \sum_{k=t+1}^{F} r_k$$

We will use a version with *discount*, called the gain of a state:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{F-t-1} r_F = \sum_{k=t+1}^{F} \gamma^{k-t-1} r_k$$

👉 The gain for the final state (s_F above) is always zero!

😇 *useful*, we'd like the robot to perform action that leads to next state with high gainS!

Notes

1 MDP Trajectories and gains Example

Deterministic MDP $= \left(\{1, \dots, 12\}, \{\uparrow, \downarrow, \leftarrow, \rightarrow\}, \delta, \rho, \infty, 0.9\right)$

Best Trajectory: highest gain from s_0 (more precisely, one of the 2 best)

$\tau = 1, \uparrow, -1, 5, \uparrow, -1, 9, \rightarrow, -1, 10, \rightarrow, -1, 11, \rightarrow, 10, 12$
$g_{s_0=1} = (-1) + 0.9 \times (-1) + 0.9^2 \times (-1) + 0.9^3 \times (-1) + 0.9^4 \times 10 \approx 3.12$

🤔 What is the trajectory with the same gain for $s_0 = 1$?

Second-Best Trajectory (one of them...)

$\tau = 1, \uparrow, -1, 5, \uparrow, -1, 9, \uparrow, -1, 9, \rightarrow, -1, 10, \rightarrow, -1, 11, \rightarrow, 10, 12$
$g_0 = (-1) + 0.9 \times (-1) + 0.9^2 \times (-1) + 0.9^3 \times (-1) + 0.9^4 \times (-1) + 0.9^5 \times 10 \approx 1.81$

Notes

1 MDP: Policy

Robot must make decisions. In state s, it will choose action a by applying *policy* π

Policy π : Which action should the robot choose?

2 types of policy:

1. *Deterministic* policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a function. In state s always choose same action a. (ex. grid world: $\pi(1) = \uparrow$);
2. *Stochastic* policy is a conditional distribution $\pi(a|s) = p(A_t = a|S_t = s)$: in s pick/sample a given probabilities defined by π. (ex. grid world $\pi(\uparrow|1) = 0.75, \pi(\rightarrow|1) = 0.25$)

Remarks

1. Given deterministic π_d, we can define an equivalent stochastic π: $\pi(a|s) = 1$ if $\pi_d(s) = a$, and 0 otherwise: all distribution mass is on $\pi_d(s)$
2. Given stochastic π_s, we can define an approximate deterministic (called greedy deterministic) $\pi_g(s) = \text{argmax}_a\left(\pi_s(a|s)\right)$: always choose most probable action

🤔 Transform the examples given above

Notes

1 MDP and Expectations: Expected Reward (2)

One question that we will ask ourselves a lot: How much immediate reward can a robot get on average when it is in state s but we haven't made a policy decision (no action) yet?
$\Rightarrow$ Need to average over actions and how *frequent* they are:

Expected Reward (given state only)

$$\begin{aligned}\mathbb{E}[R_{t+1}|S_t = s] &= \mathbb{E}_{a\sim\pi(\cdot|s)}\big[\mathbb{E}[R_{t+1}|S_t = s, A_t = a]\big] \\ &= \mathbb{E}_{a\sim\pi(\cdot|s)}\big[\mathbb{E}_{s'\sim p_\delta(\cdot|s,a)}[R_{t+1} = \rho(s,a,s')|S_t = s, A_t = a]\big] \\ &= \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a)\rho(s,a,s')\end{aligned}$$

expectation over π and p_δ. (Detailed proof as exercise in tutorials)

Notes

1 MDP: State Values (1)

What is a good policy?

Use policy to make decisions during trajectory: choose actions that give more rewards $\longrightarrow$ visit states with high gains on average.

Define the expected gain: the state value

Average of all possible gains from state s for all possible trajectories where s appears, weighted by π. By definition:

$$v^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=t+1}^{H} \gamma^{k-t-1} R_k|S_t = s]$$

Average gain after we drop the robot in s and let it choose a sequence of actions.

Remark

- R_k: random variable whose value is the reward at time k,
- r_k: actual reward obtained at time k
- same with $G_k/g_k, S_k/s_k$ ...

Notes

1 MDP: State Values (2)

Define the expected gain: the *state value*

How can we compute this discounted sum? Separate immediate reward from the rest (future rewards).

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=t+1} \gamma^{k-t-1} R_k | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{k=t+2} \gamma^{k-t-2} R_k | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s] \end{aligned}$$

Remember: if s is terminal, gain is always zero

- 🤔 How did we go from 1st to 2nd line??
- 🤔 How did we go from 3rd to 4th line??

Notes

1 MDP: State Values (3)

Compute v^π in practice: find a recurrence

Starting from $v^\pi(s) = \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s]$

Left-Hand Part (expected reward, only state):

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a) \rho(s,a,s')$$

Right-Hand Part:

$$\begin{aligned} \mathbb{E}_\pi[G_{t+1}|S_t = s] &= \mathbb{E}_{a\sim\pi(\cdot|s)}\big[\mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a]\big] \\ &= \mathbb{E}_{a\sim\pi(\cdot|s)}\big[\mathbb{E}_{s'\sim p(\cdot|s,a)}\mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a, S_{t+1} = s']\big] \\ &= \sum_a \pi(a|s) \mathbb{E}_{s'\sim p(\cdot|s,a)} \mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] \\ &= \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a) \mathbb{E}_\pi[G_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] \\ &= \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a) \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] \quad \text{thanks Markov 😇} \\ &= \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a) v^\pi(s') \end{aligned}$$

Notes

1 MDP: State Values (4)

Summing all terms

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s] \\ &= (\sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a)\rho(s,a,s')) + \gamma(\sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a)v^\pi(s')) \\ &= \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma v^\pi(s')) \end{aligned}$$

Bellman equation

If s is terminal (either $t = H$, or s final) : $v^\pi(s) = 0$ (no more gain), otherwise:

$$v^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma v^\pi(s'))$$

Remark

looks a little like Bellman from "cours de graphes INFO1" but:

- *average* instead of max/min.
- *discounted* recurrence call, with discount γ positive, ≤ 1

Notes

1 MDP (State-Action Values and Advantage Values)

Other quantities:

State-Action Value

Expected Gain in s after action a is chosen:

$$q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s; A_t = a]$$

We can derive:

- if s is terminal $\forall a, q(s,a) = 0$
- otherwise (by Bellman):

$$q^\pi(s,a) = \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma v^\pi(s'))$$

NB: In practice (lab sessions) we will use q more often than v (we'll see why)

Advantage

What gain returns action a compared to other actions, in state s:

$$A^\pi(s,a) = q^\pi(s,a) - v^\pi(s)$$

$\Rightarrow$ If $A^\pi(s,a) > 0$, a is a better-than-average decision to make when in s.

Notes

1 MDP: Policy Evaluation (1)

Policy Evaluation

Given a MDP and a policy π, compute $v^\pi(s)$ for all $s, in \mathcal{S}$ (solve all Bellman equations)

- This amounts to solving a linear system

$$v = r_\pi + \gamma P_\pi v$$

with

  - $v \in \mathbb{R}^{|\mathcal{S}|}$ is the vector of state values
  - $r_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is the vector of average immediate rewards
  - $P_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ matrix s.t. $P_\pi[s,s'] = p(s'|s) = \sum_a \pi(a|s) p_\delta(s'|s,a)$

Solution:

$$v = (I - \gamma P_\pi)^{-1} r_\pi$$

1. good news: we compute state values given π
2. bad news: we need to inverse a (big) matrix

Notes

1 MDP: Policy Evaluation (2)

Iterative solving of Policy Evaluation

Given a MDP and a policy π, compute $v^\pi(s)$ for all $s, in \mathcal{S}$ (solve all Bellman equations)

- init: $v^0 = 0 \in \mathbb{R}^{\mathcal{S}}$
- iterate: \$\forall k \geq 0, $v^{k+1} = r_\pi + \gamma P_\pi v^k$
- return: v^K

Proof of convergence

Let $\delta_k = ||v^k - v^\pi||$ be the approximation error at iteration k. Show that $\lim_k \delta_k = 0$

$$\begin{aligned}
v_{k+1} &= r_\pi + \gamma P_\pi v_k \\
\delta_{k+1} + v^\pi &= r_\pi + \gamma P_\pi (\delta_k + v^\pi) \\
\delta_{k+1} &= r_\pi - v^\pi + \gamma P_\pi (\delta_k + v^\pi) \\
&= \gamma P_\pi \delta_k - v^\pi + r_\pi + \gamma P_\pi v^\pi \\
&= \gamma P_\pi \delta_k \\
&= \gamma^k P_\pi^k \delta_0 \\
&\Longrightarrow \lim_k \delta_k = 0
\end{aligned}$$

Notes

2 Optimal Policy

Notes

2 Optimal Policy: Definition

Idea: policy that returns *the most rewards on average*

Partial order over policies

state values define a partial order: $\pi \geq \pi'$ iff $\forall s \in S, v^{\pi}(s) \geq v^{\pi'}(s)$

Optimal Policy

With finite MDP (S and A finite) there exists a policy greater than any other one, written π_*, called optimal policy.
We note $v^{\pi_*}(s)$ as $v_*(s)$, and $q^{\pi_*}(s,a)$ as $q_*(s,a)$
We can *build* the optimal policy by picking the best policy at each state:

- policy value: $\pi_*(s) = \operatorname{argmax}_{\pi} v^{\pi}(s)$
- State values are: $v_*(s) = \max_{\pi} v^{\pi}(s)$
- State-action values are: $q_*(s,a) = \max_{\pi} q^{\pi}(s,a)$
- Also (Bellman optimality condition): $v_*(s) = \max_{a} q_*(s,a)$
  - why ? In general, only $v^{\pi}(s) \leq \max_{a} q^{\pi}(s,a)$

Remark

The optimal policy is always deterministic. Why? (cf. excursion)

Notes

2 Example: Optimal Policy by Enumeration

Mini-Grid word

A	B	C

- MDP $= \{\{A, B, C\}, \{\leftarrow, \rightarrow, \emptyset\}, T, R, \infty, 0.8\}$
- B is initial, A and C are final
- 3 actions: left,right, idle
- rewards for entering A:1,B:0,C:2
- Transitions: when choosing a, 80% to perform a, and 10% other action.

🤔 There are only 3 possible policies: which ones?
🤔 In general in MDP with n non-final states and p actions?

$\pi_1(B) = \leftarrow$

Bellman gives:

- $v^{\pi_1}(B) = 1 + 0.08 \times v^{\pi_1}(B)$
- $v^{\pi_1}(B) = 1/0.92 = 1.09$

$\pi_2(B) = \rightarrow$

Bellman gives:

- $v^{\pi_2}(B) = 1.7 + 0.08 \times v^{\pi_2}(B)$
- $v^{\pi_2}(B) = 1.7/0.92 = 1.85$

$\pi_3(B) = \emptyset$

Bellman gives:

- $v^{\pi_3}(B) = 0.3 + 0.64 \times v^{\pi_3}(B)$
- $v^{\pi_3}(B) = 0.3/0.36 = 0.83$

Conclusion $\pi_*(B) = \rightarrow$ and $v_*(B) = 1.85$

Notes

2 Excursion: Always a Deterministic Optimal Policy (1)

We will show that from a policy π not deterministic, we can build a policy π' such that $\pi' \geq \pi$ and π' is *more deterministic*

1. First we say that for a state s, a policy π is s-deterministic if for s there exists an action a for which $\pi(a|s) = 1$, and for all other actions $a' \neq a, \pi(a'|s) = 0$. A deterministic policy is s-deterministic for all states s.
2. If π is not deterministic, there exists a state s for which it is not s-deterministic. For this state we have:

$$v^{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p_S(s'|s,a)(\rho(s,a,s') + \gamma v^{\pi}(s')) = \sum_a \pi(a|s) q^{\pi}(s,a)$$

   We note $Q = max_a q^{\pi}(s,a)$ and $\hat{a} = \operatorname{argmax}_a q^{\pi}(s,a)$

3. We will build π' exactly like π for all states $s' \neq s$, but a. $v^{\pi'}(s) \geq v^{\pi}(s) \Rightarrow \pi' \geq \pi$ b. π' will be s-deterministic

Notes

2 Excursion: Always a Deterministic Optimal Policy (2)

4. We set $\pi'(\hat{a}|s) = 1$, so $\pi'(a \neq \hat{a}|s) = 0$. We thus have for new policy π':

$$\begin{aligned} v^{\pi'}(s) &= \sum_a \pi'(a|s) \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma v^{\pi'}(s')) \\ &= \sum_{s'} p_\delta(s'|s,\hat{a})(\rho(s,\hat{a},s') + \gamma v^{\pi}(s')) \\ &= q^{\pi}(s,\hat{a}) = Q \end{aligned}$$

5. But we have for old policy π:

$$\begin{aligned} v^{\pi}(s) &= \sum_a \pi(a|s) q^{\pi}(s,a) \\ &\leq \sum_a \pi(a|s) Q \\ &= Q \sum_a \pi(a|s) = Q \times 1 = Q = v^{\pi'}(s) \end{aligned}$$

QED: By repeating this procedure from π on all states, we obtain a deterministic policy π' better than π. Thus we conclude that the optimal policy is deterministic.

😴 There is an slight oversimplification in 4. Where? (hint, look at the recurrence...) How can we fix it? (we'll see that in the next tutorial)

Notes

2 Bellman Optimality Equation (1)

If we apply Bellman Equation to the optimal policy, we get a new relation:

Recurrence on v_*

Either s is terminal and $v_*(s) = 0$, or:

$$\begin{aligned} v_*(s) &= \max_a q_*(s,a) = \max_a \mathbb{E}_{\pi_*}[G_t|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1}|S_t = s, A_t = a] + \mathbb{E}_{\pi_*}[\gamma G_{t+1}|S_t = s, A_t = a] \\ &= \max_a \Big(\sum_{s'} p_\delta(s'|s,a)\rho(s,a,s') + \gamma \mathbb{E}_{\pi_*}[G_{t+1}|S_t = s, A_t = a] \Big) \\ &= \max_a \Big(\sum_{s'} p_\delta(s'|s,a)\rho(s,a,s') + \gamma \sum_{s'} p_\delta(s'|s,a) \mathbb{E}_{\pi_*}[G_{t+1}|S_{t+1} = s'] \Big) \\ &= \max_a \Big(\sum_{s'} p_\delta(s'|s,a)\rho(s,a,s') + \gamma \Big(\sum_{s'} p_\delta(s'|s,a) v_*(s') \Big) \Big) \\ v_*(s) &= \max_a \sum_{s'} p_\delta(s'|s,a) \big(\rho(s,a,s') + \gamma v_*(s') \big) \end{aligned}$$

Notes

2 Bellman Optimality Equation (2)

Remark

Eventually no policy π in formula: at optimality, take the action which maximises gain.

Similar recurrence for q:

1. version q/v

$$q_*(s,a) = \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma v_*(s'))$$

2. version q/q

$$q_*(s,a) = \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma \max_{a'} q_*(s',a'))$$

Notes

2 Finding Optimal Policy (1)

How can we use the Bellman optimality condition to find the optimal policy?
A 2-step process:

1/ Find v_*

For Finite MDPs, we can use Bellman optimality condition and compute v_* (or q_*)

- for final states v_* is zero
- an equations system (non-linear because of max!):

$$v = \max_\pi (r_\pi + \gamma P_\pi v)$$

Remarks:

- lots of computations when size of the MDP grows
- needs to know dynamic and reward function (transition distribution T and R), which is not the case in general

Notes

2 Finding Optimal Policy (2)

A 2-step process:

2/ Deduce π_* from v_*:

In state s, we take the action that maximizes the probability to arrive at the best successor s' (deterministic policy) with maximum reward:

- $\pi(s) = \text{argmax}_a\, q_*(s,a) = \text{argmax}_a \sum_{s'} p_\delta(s'|s,a)(\rho(s,a,s') + \gamma v_*(s'))$ (if multiple solutions a, take one randomly)

Notes

3 Iterative Algorithms for Optimal Policy

Notes

3 Why Iterative Algorithms?

- Difficult to solve the Bellman equations system for big MDPs
- We can reason directly on the implicit graph of the MDP
- We will see 2 algorithms:
  - using *dynamic programming* (create and update a table)
  - with approximation
  - compute v_* and π_*

Notes

3 Find v_π incrementally (I)

Idea: iterate the Bellman-Condition to $v(s)$ until we find a fixed point.

Algorithm: policy evaluation

- Input: policy π , precision threshold θ (e.g. 10^{-3}),
- Init : table V, s.t. $\forall s$ MDP state, $V(s) = 0$, $\Delta = +\infty$
- While $\Delta > \theta$:
  - $\Delta \leftarrow 0$; $V_{previous} \leftarrow V$
  - For each non terminal state s:
    - $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p_S(s'|s,a)[r(s,a,s') + \gamma V_{previous}(s')]$
    - $\Delta \leftarrow \max(\Delta, |V_{previous}(s) - V(s)|)$
- Return V

Convergence: V(s) tends towards $v_\pi(s)$ (Sketch of Proof, cf. lectures by Pieter Abbeel or this link)

- At loop iteration k, paths to terminal nodes of length k at most
- when $k \nearrow$: because of γ^k, discounted rewards far away tend toward zero
- $\Longrightarrow$ update Δ get smaller and values V get closer to optimum v_π.

Notes

3 Policy Iteration

1. Find v_π incrementally (use previous algorithm)
2. Compute new policy $\pi' \geq \pi$: if $\pi \neq \pi'$, go to 1
3. Return new policy

Policy Iteration Algorithm

1. Init: $\pi(a|s)$ initialised arbitrarily
2. $V \leftarrow$ policy evaluation
3. Policy Improvement for π
   - `stable` $\leftarrow$ true
   - For each state s
     - pa $\leftarrow \pi(s)$
     - $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} p_\delta(s'|s,a)[\rho(s,a,s') + \gamma V(s')]$ (*i.e.* $= \operatorname{argmax}_a q(s,a)$)
     - if $\pi(s) \neq$ pa, then `stable` $\leftarrow$ false
   - if `stable`, return (V, π) otherwise go to 2.

Convergence

- Successive π are $\geq$ than preceeding, and converge to q_* (cf. proof in Sutton and Barto, Chapter 4.2)
- remark: step 3 looks like the determinization/improvement excursion 🤯

Notes

3 Value Iteration: Idea

Compare the two algorithms we have:

Policy Iteration (reminder)

1. Fix a policy π; fully evaluate it (inner loop until convergence $\rightarrow v^\pi$)
2. Improve π greedily $\rightarrow \pi' \geq \pi$; if $\pi' \neq \pi$, repeat

Cost: full policy evaluation at each outer step is expensive.

Key Observation

We do *not* need v^π to converge before improving. Apply the Bellman optimality update directly, *without fixing a policy*:

$$V(s) \leftarrow \max_a \sum_{s'} p_\delta(s'|s,a)\big(\rho(s,a,s') + \gamma V(s')\big)$$

This is one application of the Bellman optimality operator $\mathcal{T}$:

$$(\mathcal{T}V)(s) = \max_a \sum_{s'} p_\delta(s'|s,a)\big(\rho(s,a,s') + \gamma V(s')\big)$$

$\Rightarrow$ iterate $\mathcal{T}$ until V converges to v_*, *then* extract the policy once.

Notes

3 Value Iteration: Algorithm

Algorithm

1. **Input**: precision threshold θ (e.g.\ 10^{-3})
2. **Init**: table V s.t. $\forall s,\ V(s) = 0$; $\Delta = +\infty$
3. **While** $\Delta > \theta$:
   - $\Delta \leftarrow 0$; $V_{\text{prev}} \leftarrow V$
   - For each non-terminal state s:
     - $V(s) \leftarrow \max_a \sum_{s'} p_\delta(s'|s,a)\left[\rho(s,a,s') + \gamma V_{\text{prev}}(s')\right]$
     - $\Delta \leftarrow \max(\Delta,\ |V_{\text{prev}}(s) - V(s)|)$
4. **Extract policy**: $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} p_\delta(s'|s,a)\left[\rho(s,a,s') + \gamma V(s')\right]$
5. **Return** (V, π)

Difference from Policy Iteration

- Replace $\sum_a \pi(a|s)[\cdots]$ with $\max_a[\cdots]$: no policy to maintain
- Policy extracted *once* at the end, not updated at each iteration

Notes

3 Value Iteration: Convergence and Comparison

Convergence: V converges to v_* (sketch)

The Bellman optimality operator $\mathcal{T}$ is a contraction in the ℓ^∞ norm:

$$\|\mathcal{T}V - \mathcal{T}V'\|_\infty \ \leq\ \gamma\, \|V - V'\|_\infty$$

Since $\gamma < 1$, repeated application of $\mathcal{T}$ converges to its unique fixed point v_* (Banach fixed-point theorem). Error decreases geometrically:

$$\|V_k - v_*\|_\infty \ \leq\ \gamma^k \|V_0 - v_*\|_\infty$$

Policy Iteration vs. Value Iteration

	Policy Iteration	Value Iteration
Inner loop	Full policy evaluation	None
Cost per outer step	High	Low (one sweep)
Outer iterations needed	Few	More
Policy	Maintained throughout	Extracted at end
Convergence	Exact	v^π at each step & Asymptotic to v_*

🤔 For large state spaces, which would you prefer?

Notes

3 Value Iteration: GridWorld Example

Same probabilistic MDP as before: $\gamma = 0.9$, rewards $+1$ (12), -1 (8), 0 elsewhere.
No policy fixed upfront: $V_0(s) = 0\ \forall s$.

First iterations of $\mathcal{T}$ (selected states)

V	$k=0$	$k=1$	$k=2$	$k=3$	best action
11	0	0.800	0.872	0.921	$\rightarrow$
10	0	0	0.576	0.732	$\rightarrow$
9	0	0	0	0.415	$\rightarrow$
7	0	0	0.476	0.571	$\uparrow$
other	...	...	...	...	(exercise)

Observations

- At $k = 1$: only state 11 (one step from goal) gets a non-zero value.
- Values propagate backward from terminal states, one step per iteration.
- State 7 learns to go $\uparrow$ (toward 11) rather than $\rightarrow$ (into lava 8) 😈
- π is extracted *once* from the converged V: no policy tracking needed.

Notes

3 Policy Iteration: Simple GridWorld Example (1)

$\rightarrow$	$\rightarrow$	$\rightarrow$	Win
$\uparrow$	X	$\rightarrow$	Fail
$\uparrow$	$\leftarrow$	$\downarrow$	$\rightarrow$

- the reward is $+1$ for 12, -1 for 8, and 0 for all other states
- dynamic is : when going direction d, 80% chance to go d but 10% chance to go on each perpendicular direction
- if movement is impossible, stay on the same cell
- π is deterministic and denoted by arrows
- 1 is initial while 8 and 12 are final.
- discount γ is 0.9

Notes

3 Policy Iteration: Simple GridWorld Example (2)

State Values evaluation with Bellman:

- Let's have a look at state 1
- Since R only depends on s', we will simplify notations: $\rho(s,a,s') = \rho(s')$
- Bear in mind that π is deterministic $\longrightarrow$ less computations!

Use Bellman and simplify to remove null terms:

$$\begin{aligned} V(1) &= \sum_a \pi(a|1) \sum_{s'} p_\delta(s'|1,a)(\rho(1,a,s') + \gamma V(s')) \\ &= \sum_{s'} p_\delta(s'|1,\uparrow)(\rho(s') + 0.9V(s')) \\ &= 0.8(\rho(5) + 0.9V(5)) + 0.1(\rho(1) + 0.9V(1)) \\ &\quad + 0.1(\rho(2) + 0.9V(2)) \\ &= 0.72V(5) + 0.09V(1) + 0.09V(2) \end{aligned}$$

Notes

3 Policy Iteration: Simple GridWorld Example (3)

State Values evaluation with Bellman
Let's have a look at 11

$$\begin{aligned} V(11) &= \sum_a \pi(a|11) \sum_{s'} p_\delta(s'|11,a)(\rho(11),a,s') + \gamma V(s')) \\ &= \sum_{s'} p_\delta(s'|11,\rightarrow)(\rho(s') + 0.9V(s')) \\ &= 0.8(\rho(12) + 0.9V(12)) + 0.1(\rho(11) + 0.9V(11)) \\ &\quad + 0.1(\rho(7) + 0.9V(7) \\ &= 0.8 + 0.72V(12) + 0.09V(11) + 0.09V(7) \\ &= 0.8 + 0.09V(11) + 0.09V(7) \end{aligned}$$

Notes

3 Policy Iteration: Simple GridWorld Example (4)

V	expression
1	$0.72V(5) + 0.09V(1) + 0.09V(2)$
2	$0.72V(1) + 0.18V(2)$
3	$0.72V(3) + 0.09V(2) + 0.09V(4)$
4	$0.81V(4) - 0.1 + 0.09V(8)$
5	$0.72V(2) + 0.18V(5)$
7	$-0.8 + 0.72V(8) + 0.09V(11) + 0.09V(3)$
9	$0.72V(10) + 0.09V(9) + 0.09V(5)$
10	$0.72V(11) + 0.18V(10)$
11	$0.8 + 0.72V(12) + 0.09V(11) + 0.09V(7)$

Could be simplified further: $V12 = V8 = 0$

Notes

3 Policy Iteration: Simple GridWorld Example (5)

V	k=0	k=1	k=2	k=10	k=100	k=1000
1	0	0	0.0	0.461	0.467	0.467
2	0	0	0.0	0.393	0.410	0.410
3	0	0	-0.009	-0.046	-0.0374	-0.0374
4	0	-0.1	-0.181	-0.462	-0.526	-0.526
5	0	0	0.0	0.538	0.539	0.539
7	0	-0.8	-0.728	-0.732	-0.730	-0.730
9	0	0	0.0	0.613	0.614	0.614
10	0	0	0.576	0.708	0.708	0.708
11	0	0.8	0.8	0.807	0.807	0.807

Notes

3 Policy Iteration: Simple GridWorld Example (6)

We can now re-evaluate our policy, and improve it:

- $\forall s, \pi'(s) = \arg\max_a \sum_{s'} p_\delta(s'|s,a)[\rho(s,a,s') + \gamma V(s')]$

$$\pi'(7) = \underset{a}{\operatorname{argmax}} \sum_{s'} p_\delta(s'|7,a)[R(7,a,s') + \gamma V(s')]$$

- for $a = \uparrow$

$$\begin{aligned} & 0.8 * [R(11) + \gamma V(11)] && +0.1 * [R(7) + \gamma V(7)] \\ & && +0.1 * [R(8) + \gamma V(8)] \\ = \; & 0.8 * [\gamma V(11)] + 0.1 * [\gamma V(7)] && +0.1 * [-1 + \gamma V(8)] \\ = \; & 0.8 * [0.9 * 0.807] + 0.1 * [0.9 * (-0.730)] && +0.1 * [-1 + 0.9 * 0] \\ = \; & 0.58104 - 0.0657 - 0.1 \\ = \; & 0.41534 \end{aligned}$$

Notes

3 Policy Iteration: Simple GridWorld Example (7)

- for $a = \rightarrow$

$$= -0.796103$$

- for $a = \leftarrow$

$$= -0.456336$$

- for $a = \downarrow$

$$= -0.00017691696$$

Conclusion

$\pi'(7) = \uparrow$

Notes

3 Policy Iteration: Simple GridWorld Example (8)

This is not finished!

We only have performed one improvement iteration for some states!

- compute π' for every state
- since $\pi' \neq \pi$, loop again: evaluation + improvement

Notes

4 Limitations – Bridge to RL

Notes

4 What next?

So far, we assumed full knowledge of δ and ρ. This is called planning. We were solving the MDP, not learning it.

When does that assumption fail?

Most real problems don't come with a known model. The model is either too complex to write down, unknown, or non-stationary:

- A robot on unknown terrain: physics not fully known
- An agent playing a video game: transition function has millions of states
- A financial trading agent: market dynamics change over time

What will Reinforcement Learning change?

In RL, agent must find optimal policy by interacting with environment, observing (s, a, r, s') without access to δ or ρ.

A software engineering analogy

- So far we had a known API: we could inspect δ and ρ directly.
- RL is when you only have a black-box API
  - you can call it (take actions), observe the return value (reward, next state),
  - but you cannot read the source code.

Notes

Notes