Universidad Zaragoza 1542

Planning in Markov decision processes

Machine Learning (69152)

Rubén Martínez Cantín
Dpto. Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Markov decision processes

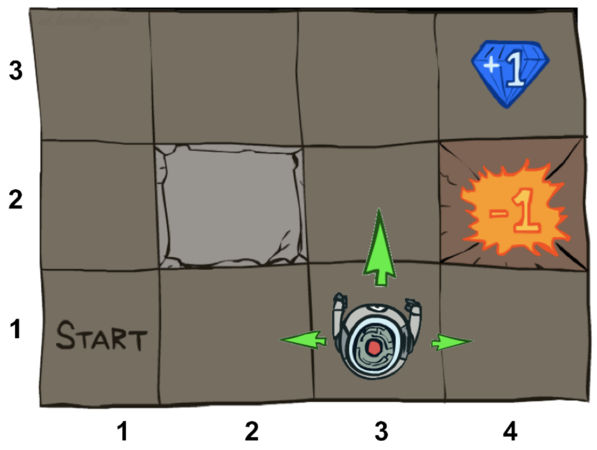
- Markov decision processes are formal descriptions of the environment for reinforcement learning.
- The system is Markov $\Rightarrow$ The state fully characterizes the process.
- The state is *fully observable*.
- Almost all RL problems can be formulated as MDPs:
  - Control problems can be represented as continuous MDP.
  - Partially observable processes can be modeled as *belief-MDPs*.
  - Multi-armed bandits[1] as one state MDPs.

[1] Defined in the next lecture.

Markov Decission Process

- A Markov Decision Process is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$
  - A set of states $x \in \mathcal{X}$.
  - A set of actions $a \in \mathcal{A}$.
  - A transition function $T(x, a, x') = p(x'|x, a)$
  - A reward function $R(x, a, x')$
  - Maybe a start state and a terminal state.
- The dynamic model (states and transitions) is a Markov chain.
  - *"The future is independent of the past given the present"*

$$p(x_{t+1}|x_t) = p(x_{t+1}|x_1, \ldots, x_t)$$

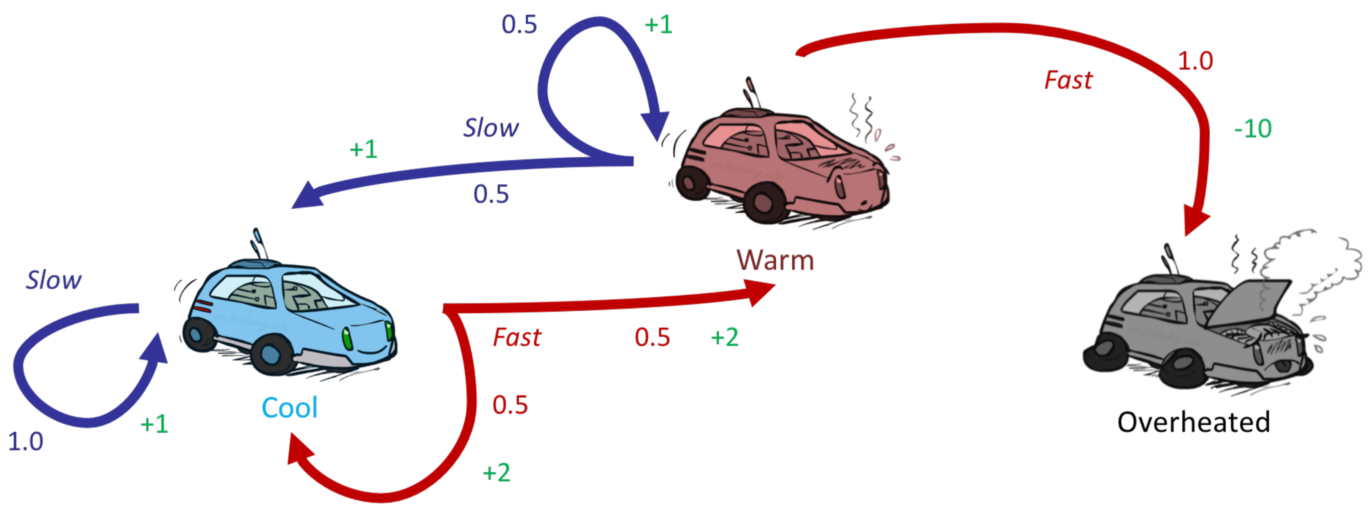


- In this lecture, we assume a discrete MDP, that is, we can iterate over all elements (states, actions, values...).

Credit: Dan Klein, Pieter Abbeel

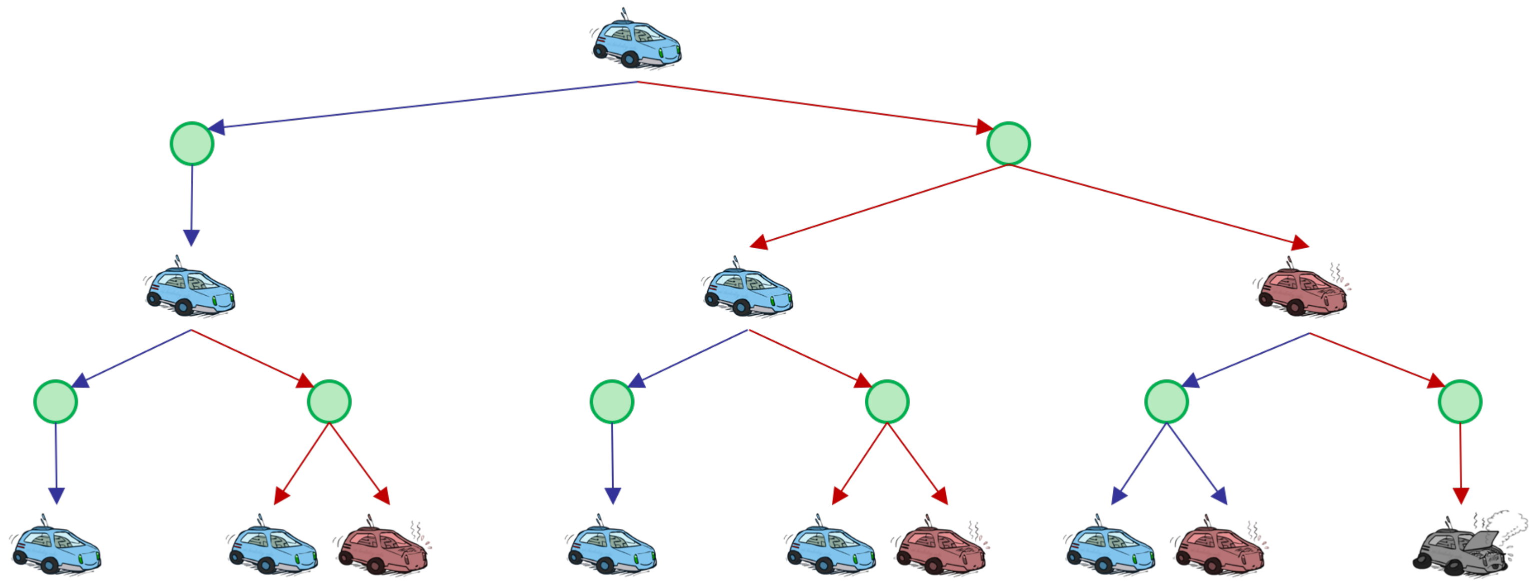
Example: Racing

Represent the racing problem in tabular form (states, actions, transitions, reward and policy)



Credit: Dan Klein, Pieter Abbeel

Search tree solution



Credit: Dan Klein, Pieter Abbeel

Policies

Definition

A policy π is a distribution over actions given states,

$$\pi(a|x) = p(a_t = a|x_t = x)$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
  - Policies are stationary (time-independent), except for finite-horizon problems, $a_t \sim \pi(\cdot|x_t), \forall t > 0$

Utilities

- What preferences should an agent have over reward sequences?
  - More or less? $[1, 2, 3]$ or $[2, 3, 4]$
  - Now or later? $[0, 0, 1]$ or $[1, 0, 0]$

Theorem: Stationary preferences

If we assume stationary preferences:

$$[a_1, a_2, \dots] \succ [b_1, b_2, \dots] \Leftrightarrow [r, a_1, a_2, \dots] \succ [r, b_1, b_2, \dots]$$

There is only one way to define utilities (returns).

$$\mathcal{U}([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

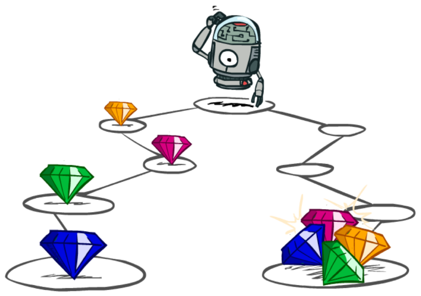
Discounted utility

- The utility defined as:

$$\mathcal{U}([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$

  is called a discounted utility or discounted return, for $\gamma \in [0, 1)$.
- The special case where $\gamma = 1$ is called additive utility.

- Why discount?
  - The agent should prefer sooner rewards. Do not procrastinate!
  - It helps convergence, specially for infinite horizon problems.
  - Biological inspiration (human/animal).
  - Uncertainty about future events.
  - $\gamma \to 0$ represents a *myopic* agent.
  - $\gamma \to 1$ leads to a *far-sighted* agent.



Credit: Dan Klein, Pieter Abbeel

Quiz: discounting

a	b	c	d	e
10				1

- Given the previous states $[a, b, \ldots]$ and rewards $(10, 1)$, where a and e are terminal states.
- Actions are *left* and *right*.
- Transitions are deterministic (e.g.: *left* always move left except on terminal state)

- For $\gamma = 1$, what is the optimal policy?
- For $\gamma = 0.1$, what is the optimal policy?
- For which γ is *left* and *right* equally good in state d?

Infinite rewards

The utility or return of a problem must always be finite.

Finite horizon: episodes terminate after a fixed number of steps T.

- Nonstationary policies: the action might depend on the state and time step.

Discounting: use $0 < \gamma < 1$

- Smaller γ means short term focus.

Absorbing state: a terminal state that is reached for every policy.

The sequence of states from the starting state to the end (finite horizon or terminal state) is called episode.

episodic (terminating) task vs continuous (running forever) task

Value function

Definition

The *state-value* function $V^{\pi}(x)$ of an MDP is the expected utility starting from state x, and then following policy π

$$V^{\pi}(x) = \mathbb{E}_{\pi}[U_t | x_t = x]$$

Definition

The *action-value* function $Q^{\pi}(x, a)$ is the expected utility starting from state s, taking action a, and then following policy π

$$Q^{\pi}(x, a) = \mathbb{E}_{\pi}[U_t | x_t = x, a_t = a]$$

Bellman equation for value functions

The value function can be computed recursively. The value function at any time $V^\pi(x_t)$ can be decomposed as:

- the inmediate reward R_{t+1}
- the (discounted) future value function $\gamma V^\pi(x_{t+1})$

$$\begin{aligned} V^\pi(x) &= \mathbb{E}_\pi[U_t|x_t = x] = \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |x_t = x] = \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots)|x_t = x] = \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma U_{t+1}|x_t = x] = \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V^\pi(x_{t+1})|x_t = x] \end{aligned}$$

Bellman Expectation Equation

- Similarly, for action-value funcions, we can follow the same procedure to obtain:

$$Q^\pi(x,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q^\pi(x_{t+1}, a_{t+1})|x_t = x, a_t = a]$$

- Remember that for discrete distributions:

$$\mathbb{E}_\mathcal{Z}[f(z)] = \sum_{z \in Z} p(z) f(z)$$

- Then we can see how the value functions are related:

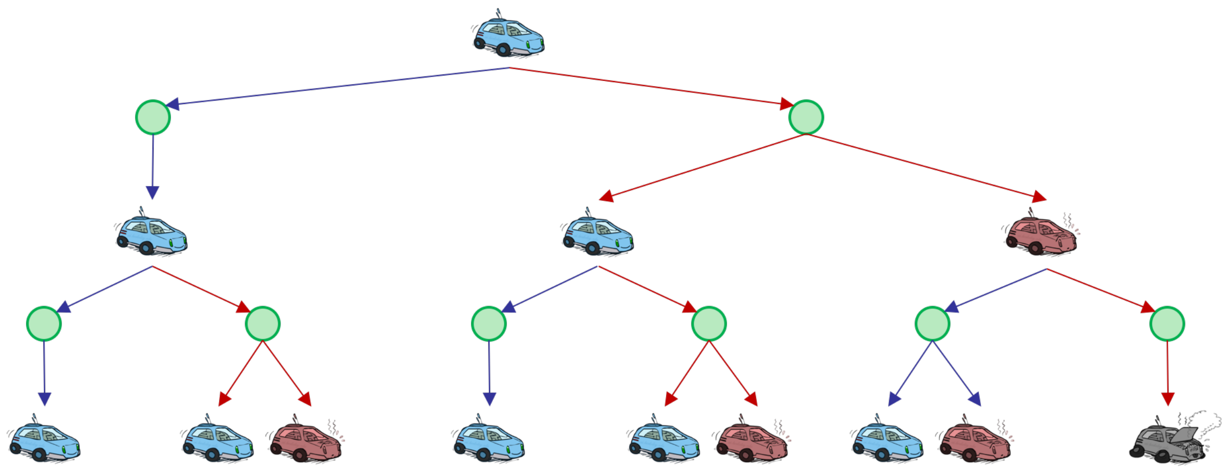
$$V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x) Q^\pi(x,a)$$

Bellman Expectation Equation

Further expanding the expectations que can see that:

$$V^{\pi}(x) = \sum_{a \in \mathcal{A}} \pi(a|x) Q^{\pi}(x, a)$$

$$Q^{\pi}(x, a) = \sum_{x' \in \mathcal{X}} p(x'|x, a) \left(R(x, a, x') + \gamma V^{\pi}(x') \right)$$



Credit: Dan Klein, Pieter Abbeel

Bellman Expectation Equation

Combining the previous equations:

$$V^{\pi}(x) = \sum_{a \in \mathcal{A}} \pi(a|x) \left(\sum_{x' \in \mathcal{X}} p(x'|x,a) \left(R(x,a,x') + \gamma V^{\pi}(x') \right) \right)$$

Finally:

$$Q^{\pi}(x,a) = \sum_{x' \in \mathcal{X}} p(x'|x,a) \left(R(x,a,x') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|x') Q^{\pi}(x',a') \right)$$

For small MDPs, these equations can be solved directly, as a system of equations.

Optimal Value Functions

Definition

The optimal *state-value function* $V^*(x)$ is the maximum value function over all policies:

$$V^*(x) = \max_{\pi} V^{\pi}(x)$$

Definition

The optimal *action-value function* $Q^*(x, a)$ is the maximum action-value function over all policies:

$$Q^*(x, a) = \max_{\pi} Q^{\pi}(x, a)$$

The optimal value function is the best an agent can do in the MDP. It is the solution.

Optimal policy

Theorem

For any Markov Decision Process:

- There is at least one optimal policy π^*.
- All optimal policies achive the optimal value functions:

$$V^*(x) = V^{\pi^*}(x), \qquad Q^*(x,a) = Q^{\pi^*}(x,a)$$

- Important: The optimal policy is greedy with respect to the optimal value function.
  - Deterministic policy:

$$a = \pi^*(x) = \arg\max_{a \in \mathcal{A}} Q^*(x,a)$$

  - Stochastic policy (without ties):

$$\pi^*(a|x) = \begin{cases} 1 & \text{if } \; a = \arg\max_{a \in \mathcal{A}} Q^*(x,a) \\ 0 & \text{otherwise} \end{cases}$$

Bellman Optimality Equation

- Similarly to what we did for the *Bellman expectation equations* for $V^\pi(x)$ and $Q^\pi(x)$, we can use *Bellman optimality equations* for the optimal values $V^*(x)$ and $Q^*(x)$.

$$V^*(x) = \max_a Q^*(x, a)$$
$$Q^*(x, a) = \sum_{x' \in \mathcal{X}} p(x'|x, a) \left(R(x, a, x') + \gamma V^*(x')\right)$$

- Compare the result with the equations in slide 14

Bellman Optimality Equation

Combining the previous equations:

$$V^*(x) = \max_a \left(\sum_{x' \in \mathcal{X}} p(x'|x,a) \left(R(x,a,x') + \gamma V^*(x') \right) \right)$$

Finally:

$$Q^*(x,a) = \sum_{x' \in \mathcal{X}} p(x'|x,a) \left(R(x,a,x') + \gamma \max_{a'} Q^*(x',a') \right)$$

Compare the result with the equations in slide 15

Solving the Bellman Equations

- MDPs have all the ingredients for *dynamic programming*.
  - Bellman equations give recursive decomposition.
  - Value function stores and reuses solutions.
- We are going to see three algorithms:

  Policy evaluation: Use *Bellman expectation equation*.

  1. Given a policy $\pi \rightarrow$ compute $V^{\pi}(x)$ iteratively.

  Value iteration: Use *Bellman optimality equation*.

  1. Compute $V^{*}(x)$ recursively.

  Policy iteration: Evaluate and improve a policy iteratively.

  1. Given $\pi \rightarrow$ compute $V^{\pi}(x)$
  2. Given $V^{\pi}(x)$, choose π' greedy $\rightarrow$ compute $V^{\pi'}(x)$
  3. ...

  n. Converges to π^{*} and $V^{*}(x)$.

Iterative policy evaluation

- Prediction problem: Given a policy π, compute $V^{\pi}(x)$, $\forall x$
- Idea: apply *Bellman expectation backups* iteratively

$$V_{k+1}(x) = \sum_{a \in \mathcal{A}} \pi(a|x) \left(\sum_{x' \in \mathcal{X}} p(x'|x,a) \left(R(x,a.x') + \gamma V_k(x') \right) \right)$$

- Start with a default value $V_0(x) = 0$, $\forall x$
- Compute $V_1(x)$ for all states *synchronously* using Bellman equation.
- $V_0 \to V_1 \to V_2 \to \ldots \to V^{\pi}$
- Improvements:
  - Synchronous version in matrix form: $\mathbf{V}_{k+1}(x) = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} \mathbf{V}_k(x')$
  - *Asynchronous* version where we backup only one state per iteration also converges.

Policy evaluation algorithm

Algorithm 1 Policy evaluation algorithm

```
Input: $\pi$, the policy to be evaluated. Accuracy threshold $\theta > 0$
  Initialize $V(x)$, for all $x \in \mathcal{X}$, arbitrarily except that $V(terminal) = 0$
  repeat
    $v \leftarrow V$                                        ▷ Keep backup value
    for all $x \in \mathcal{X}$ do
      $V(x) \leftarrow \sum_{a\in\mathcal{A}} \pi(a|x) \sum_{x'\in\mathcal{X}} p(x'|x,a)\left(R(x,a,x') + \gamma v(x')\right)$
  until $\max(|v - V|) < \theta$
```

This algorithm can also be done in-place without backup.

Value iteration

- Can we use Bellman optimality equation directly in a recursive way?

$$V^*(x) = \max_a \sum_{x' \in \mathcal{X}} p(x'|x, a) \left(R(x, a, x') + \gamma V^*(x')\right)$$

- Principle of optimality: An optimal policy is...
  - An optimal first action a.
  - An optimal policy in the successor state x'.
- Idea: Work backwards iteratively

$$V_{k+1}(x) = \max_a \sum_{x' \in \mathcal{X}} p(x'|x, a) \left(R(x, a, x') + \gamma V_k(x')\right)$$

Value iteration algorithm

Algorithm 2 Batch value iteration algorithm

```
Input: Accuracy threshold $\theta > 0$
    Initialize $V_k(x)$, for all $x \in \mathcal{X}$, arbitrarily except that $V_k(\mathit{terminal}) = 0$
    repeat
        $v_k \leftarrow V_k$
        for all $x \in \mathcal{X}$ do
            $V_{k+1}(x) \leftarrow \max_a \left(\sum_{x' \in \mathcal{X}} p(x'|x,a) \left(R(x,a,x') + \gamma v_k(x')\right)\right)$
    until $\max(|v_k - V_k|) < \theta$

    Output: $\pi(x) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{x' \in \mathcal{X}} p(x'|x,a)(R(x,a,x') + \gamma V_k(x'))$
```

This algorithm can also be computed asynchronously. For example, we can do inline updates or use prioritized sweeping. (lab sessions)

Demo of dynamic programming

Javascript demo of gridworld using dynamic programming:

`https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html`

Credit: Andrej Karpathy

Bibliography

- Richard S. Sutton, Andrew G. Barto. Reinforcement learning: An Introduction (second edition), 2018
  `http://incompleteideas.net/book/the-book.html`
- David Silver. Advanced Topics in Machine Learning, 2015
  `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`
- Dan Klein, Pieter Abbeel. Artificial Intelligence CS188
  `http://ai.berkeley.edu`