# Approximate dynamic programming and reinforcement learning for control

Lucian Buşoniu

Universitat Politècnica de València, 21-23 June 2017

# Part I

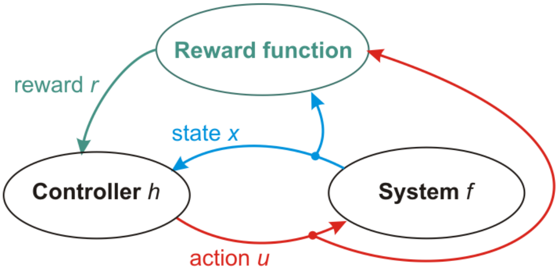
## Problem definition. Discrete case

# Main idea

Find a **control** law
to **optimize** cumulative performance
for a **general** system

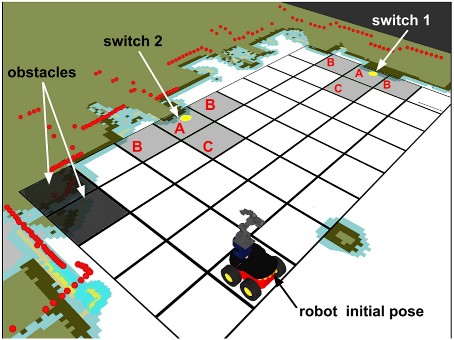
Reinforcement learning: system **unknown**, learn from data

# RL principle




- Interact with system: measure states, apply actions
- Performance feedback in the form of rewards
- Inspired by human and animal learning

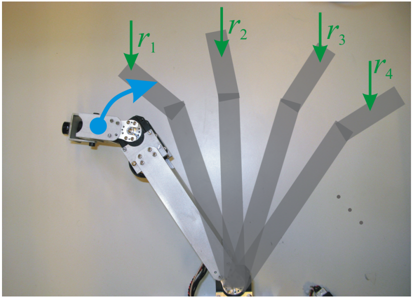
# Example: Domestic robot




A domestic robot ensures light switches are off
Abstractization to high-level control (physical actions implemented by low-level controllers)

- States: grid coordinates, switch states
- Actions: movements NSEW, toggling switch
- Rewards: when switches toggled on→off

# Example: Robot arm

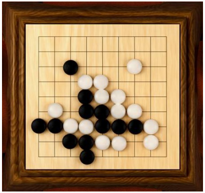


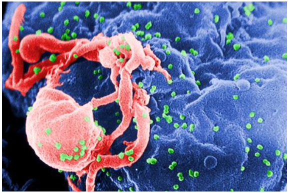

Low-level control

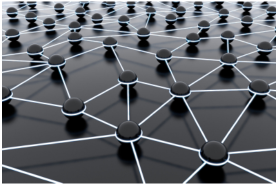
- States: link angles and angular velocities
- Actions: motor voltages
- Rewards: e.g. to reach a desired configuration, give larger rewards as robot gets closer to it

# Many other applications

Artificial intelligence, medicine, multiagent systems, economics etc.







# Why learning?

**Learning** finds solution that:

1. cannot be designed in advance
   - problem incompletely known
     (e.g. robotic space exploration)
   - problem too complex
     (e.g. controlling strongly nonlinear systems)
2. continually improve
3. adapt to time-varying environments

# Model-based methods

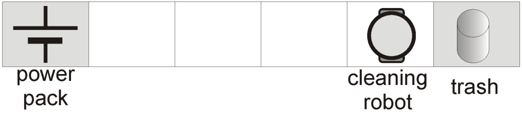
We will also focus on **model-based methods**, because they:

- form the basis of RL (e.g. dynamic programming)
- are inspired by RL (e.g. optimistic planning)
- are useful separately from RL, when a model is known, since they can address complex (nonlinear) problems

# High-level course structure

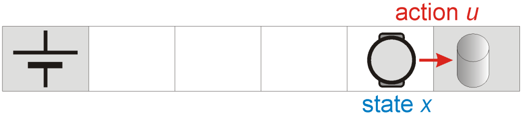
- Problem definition. Discrete-variable exact methods
- Continuous-variable, approximation-based methods
- Optimistic planning

# Simple example: Cleaning robot

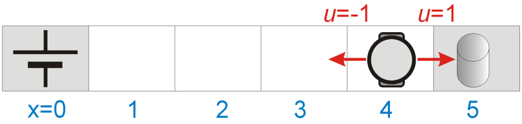



- Cleaning robot in a 1-D world
- Collects trash (reward $+5$) or power pack (reward $+1$)
- Once either trash of power pack collected, episode ends

# State & action

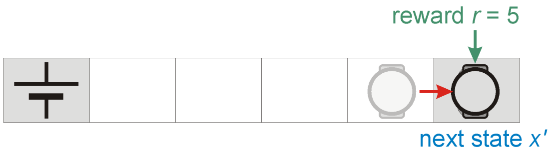



- Robot is in a certain state $x$ (cell)
- and applies an action $u$ (e.g. moves right)



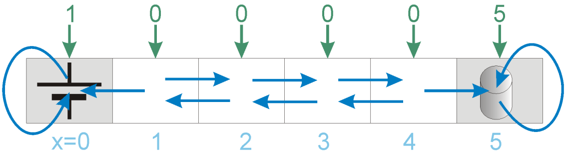

- State space $X = \{0, 1, 2, 3, 4, 5\}$
- Action space $U = \{-1, 1\} = \{\text{left, right}\}$

# Transitions and rewards




- Robot reaches a new state $x'$
- and receives a reward $r$ = quality of transition (here, $+5$ for collecting the trash)

# Transition and reward functions




- Transition function (system behavior):

$$x' = f(x, u) = \begin{cases} x & \text{if } x \text{ terminal (0 sau 5)} \\ x + u & \text{otherwise} \end{cases}$$

- Reward function (immediate performance):

$$r = \rho(x, u) = \begin{cases} 1 & \text{if } x = 1 \text{ and } u = -1 \text{ (power pack)} \\ 5 & \text{if } x = 4 \text{ and } u = 1 \text{ (trash)} \\ 0 & \text{otherwise} \end{cases}$$

- **Note:** Terminal states cannot be exited & are not rewarded!

# Markov decision process

## Markov decision process (MDP)

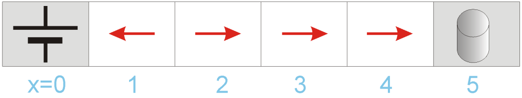
Consists of:

1. State space $X$
2. Action space $U$
3. Transition function $x' = f(x, u), \quad f : X \times U \to X$
4. Reward function $r = \rho(x, u), \quad \rho : X \times U \to \mathbb{R}$

1. Introduction
2. Problem definition
   - Markov decision process
   - Control policy and objective
   - Optimal solution
3. Dynamic programming, DP
4. Monte Carlo, MC
5. Temporal differences, TD

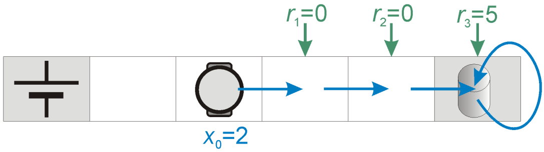
# Control policy

- Control policy $h$: maps $x$ to $u$ (state feedback)
- Encodes the behavior of the controller




Example: $h(0) = *$ (terminal state, action is irrelevant), $h(1) = -1$, $h(2) = 1$, $h(3) = 1$, $h(4) = 1$, $h(5) = *$

# Return




Take policy *h* that always moves right

$$R^h(2) = \gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + \gamma^3 0 + \gamma^4 0 + \dots$$
$$= \gamma^2 \cdot 5$$

Since $x_3$ is terminal, all later rewards are 0

# Control objective

Find $h$ that maximizes the return:

$$R^h(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, h(x_k))$$

from any $x_0$

Discount factor $\gamma \in [0, 1)$:

- represents an increasing uncertainty about the future
- bounds the infinite sum (if rewards bounded)
- induces a "pseudo-horizon" for the optimal control
- helps the convergence of algorithms

Note: There are also other types of return!

# Choosing the discount factor

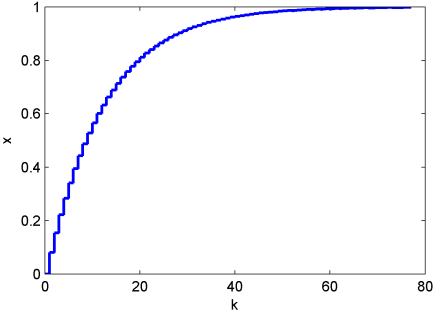
To choose $\gamma$, **trade-off** between:

1. Long-term quality of the solution (large $\gamma$)
2. "Simplicity" of the problem (small $\gamma$)

In practice, $\gamma$ should be sufficiently large so as not to ignore important rewards along the system trajectories

# Example: Choosing $\gamma$ for a simple system

Step response of a first-order linear system:



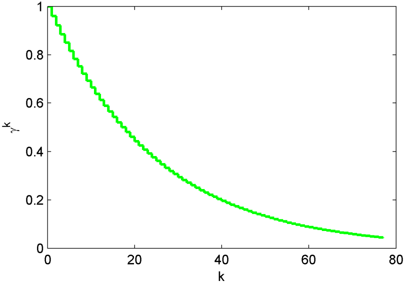

What should $\gamma$ be so that the rewards upon entering steady state are visible from the initial state?

# Solution: Choosing $\gamma$ for a simple system

For $k \approx 60$, $\gamma^k$ should not be too small, e.g.

$$\gamma^{60} \geq 0.05$$
$$\gamma \geq 0.05^{1/60} \approx 0.9513$$

$\gamma^k$ for $\gamma = 0.96$:



# Stochastic case outline

In response to $u$ in $x$, system no longer reacts deterministically – it can reach one of several states with different probabilities

## Stochastic MDP

1. State and action spaces $X$, $U$ have the same meaning
2. Transition function gives probabilities $\tilde{f}(x, u, x')$, $\tilde{f} : X \times U \times X \rightarrow [0, 1]$
3. Reward function of the whole transition $\tilde{\rho}(x, u, x')$, $\tilde{\rho} : X \times U \times X \rightarrow \mathbb{R}$

## Revised objective

Find $h$ to maximize the expected return:

$$R^h(x_0) = \mathrm{E}\left\{\sum_{k=0}^{\infty} \gamma^k \tilde{\rho}(x_k, h(x_k), x_{k+1})\right\}$$

from any $x_0$

1. Introduction
2. Problem definition
   - Markov decision process
   - Control policy and objective
   - Optimal solution
3. Dynamic programming, DP
4. Monte Carlo, MC
5. Temporal differences, TD

# Back to deterministic objective

Find optimal policy $h^*$ that maximizes return

$$R^h(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, h(x_k))$$

from any $x_0$

- We will **characterize** the optimal solution
- Before that, characterize **any** policy

# Q-value function

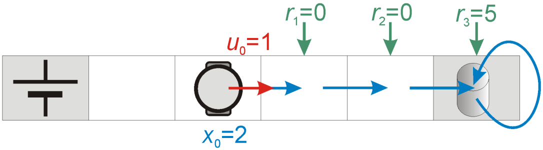
**Q-function** of a policy $h$
measures the quality of state-action pairs:

$$Q^h(x_0, u_0) = \rho(x_0, u_0) + \gamma R^h(x_1)$$

(return achieved by executing $u_0$ in $x_0$ and then following $h$)

# Q-function details

- First action $u_0$ free; remaining actions chosen with $h$




- Explicit formula using return:

$$Q^h(x_0, u_0) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) = \rho(x_0, u_0) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, h(x_k))$$

$$= \rho(x_0, u_0) + \gamma \sum_{k=0}^{\infty} \gamma^k \rho(x_{k+1}, h(x_{k+1}))$$

$$= \rho(x_0, u_0) + \gamma R^h(x_1)$$

# Bellman equation

- Go one step further in the equation:

$$\begin{aligned} Q^h(x_0, u_0) &= \rho(x_0, u_0) + \gamma R^h(x_1) \\ &= \rho(x_0, u_0) + \gamma[\rho(x_1, h(x_1)) + \gamma R^h(x_2)] \\ &= \rho(x_0, u_0) + \gamma Q^h(x_1, h(x_1)) \end{aligned}$$

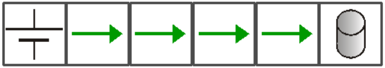
Recall that $x_1 = f(x_0, u_0)$

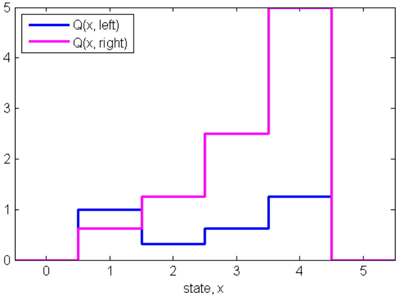
$\Rightarrow$ **Bellman equation for $Q^h$**

$$Q^h(x, u) = \rho(x, u) + \gamma Q^h(f(x, u), h(f(x, u)))$$

# Cleaning robot: Q-function example

Discount factor $\gamma = 0.5$
Policy $h(x) = 1$, always move right





# Optimal solution

- **Optimal Q-function**:

$$Q^* = \max_h Q^h$$

$\Rightarrow$ "Greedy" policy in $Q^*$:

$$h^*(x) = \arg\max_u Q^*(x, u)$$

is **optimal** (achieves maximal returns)
(if multiple actions maximize, break ties arbitrarily)

# Bellman optimality equation

$$
\begin{aligned}
Q^*(x_0, u_0) &= \max_h Q^h(x_0, u_0) \\
&= \max_{u_1, u_2, \dots} \left[\rho(x_0, u_0) + \gamma \rho(x_1, u_1) + \gamma^2 \rho(x_2, u_2) + \dots\right] \\
&= \rho(x_0, u_0) + \gamma \max_{u_1, u_2, \dots} \left[\rho(x_1, u_1) + \gamma \rho(x_2, u_2) + \dots\right] \\
&= \rho(x_0, u_0) + \gamma \max_{u_1} \left\{\rho(x_1, u_1) + \gamma \max_{u_2, \dots} [\rho(x_2, u_2) + \dots]\right\} \\
&= \rho(x_0, u_0) + \gamma \max_{u_1} Q^*(x_1, u_1)
\end{aligned}
$$

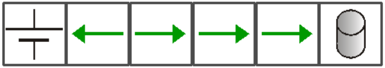
Recall $x_1 = f(x_0, u_0)$

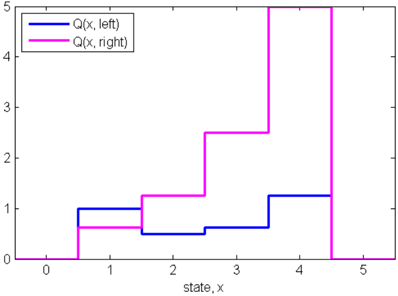
**Bellman optimality equation** (for $Q^*$)

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

# Cleaning robot: Optimal Q-function

Discount factor $\gamma = 0.5$





# A small detour: Familiar linear case

$$
\begin{aligned}
x_{k+1} &= Ax_k + Bu_k =: f(x_k, u_k) \\
\min J(x_0) &= \min \sum_{k=0}^{\infty} \gamma^k (x_k^\top Q x_k + u_k^\top R u_k) \\
&= \max \sum_{k=0}^{\infty} \gamma^k (-x_k^\top Q x_k - u_k^\top R u_k) \\
&=: \max \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k)
\end{aligned}
$$

- Usually, $\gamma = 1$ taken in control, whereas we need $\gamma < 1$
- Note $x$ and $u$ are continuous during this detour

# Linear case solution

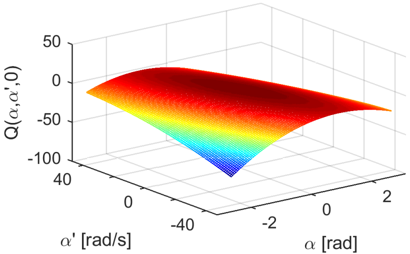
- Bellman optimality equation turns into the Riccati equation:

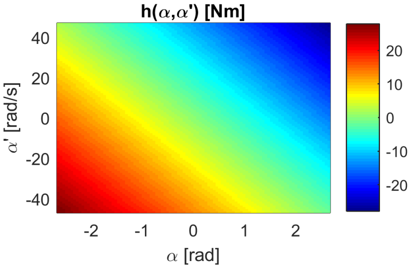
$$Y = A^\top(\gamma Y - \gamma^2 YB(\gamma B^\top YB + R)^{-1} B^\top Y)A + Q$$

with optimal Q-function:

$$Q^*(x, u) = -x^\top Qx - u^\top Ru - \gamma(Ax + Bu)^\top Y(Ax + Bu)$$

- Intuition: optimal cost $J(x) = x^\top Yx$
- Optimal control policy $h^*(x) = -\gamma(\gamma B^\top YB + R)^{-1} B^\top YAx$



Up next:

Algorithms to find the optimal solution

# Algorithm landscape

By model usage:

- Model-based: $f$, $\rho$ known
- Model-free: $f$, $\rho$ unknown (reinforcement learning)

By interaction level:

- Offline: algorithm runs in advance
- Online: algorithm runs with the system

Exact vs. approximate:

- Exact: $x, u$ small number of discrete values
- Approximate: $x, u$ continuous (or many discrete values)

First: Dynamic programming in the discrete case

# Value iteration idea

We use Q-functions ⇒ specific algorithm "Q-iteration" (there are others)

1: find optimal Q-function $Q^*$
2: compute $h^*$, greedy in $Q^*$

# Q-iteration

- Transforms Bellman optimality equation:

$$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u),u')$$

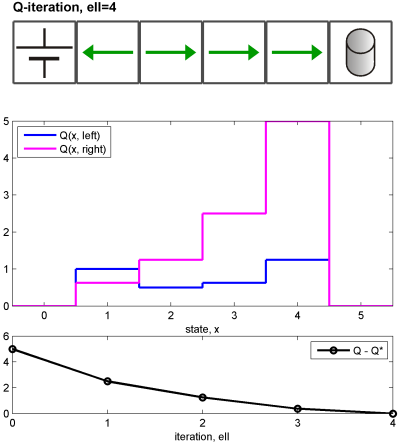
into an **iterative procedure**:

## Q-iteration

**repeat** at each iteration $\ell$
  **for all** $x, u$ **do**
    $Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$
  **end for**
**until** convergence to $Q^*$
Once $Q^*$ available: $h^*(x) = \arg\max_u Q^*(x,u)$

# Cleaning robot: Q-iteration demo

Discount factor: $\gamma = 0.5$



# Cleaning robot: Q-iteration

$$Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$$

| | $x=0$ | $x=1$ | $x=2$ | $x=3$ | $x=4$ | $x=5$ |
|---|---|---|---|---|---|---|
| $Q_0$ | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 |
| $Q_1$ | 0 ; 0 | 1 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 5 | 0 ; 0 |
| $Q_2$ | 0 ; 0 | 1 ; 0 | 0.5 ; 0 | 0 ; 2.5 | 0 ; 5 | 0 ; 0 |
| $Q_3$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 1.25 | 0.25 ; 2.5 | 1.25 ; 5 | 0 ; 0 |
| $Q_4$ | 0 ; 0 | 1 ; 0.625 | 0.5 ; 1.25 | 0.625 ; 2.5 | 1.25 ; 5 | 0 ; 0 |
| $Q_5$ | 0 ; 0 | 1 ; 0.625 | 0.5 ; 1.25 | 0.625 ; 2.5 | 1.25 ; 5 | 0 ; 0 |
| $h^*$ | $*$ | $-1$ | 1 | 1 | 1 | $*$ |

$$h^*(x) = \arg\max_u Q^*(x,u)$$

# Policy iteration

## Policy iteration

initialize policy $h_0$
**repeat** at each iteration $\ell$
1: policy evaluation: find $Q^{h_\ell}$
2: policy improvement:
$$h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x, u)$$
**until** convergence to $h^*$

# Policy evaluation

Similarly to Q-iteration:

- Transforms Bellman equation for $Q^h$:

$$Q^h(x, u) = \rho(x, u) + \gamma Q^h(f(x, u), h(f(x, u)))$$

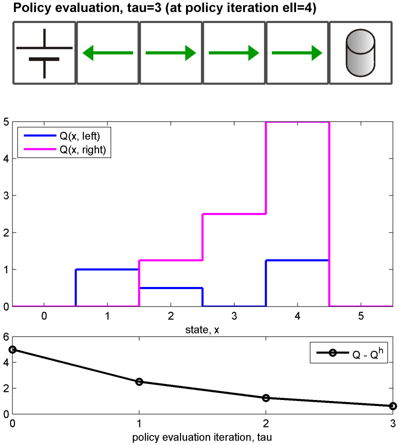
into an iterative procedure:

**Policy evaluation**

**repeat** at each iteration $\tau$
  **for all** $x, u$ **do**
    $Q_{\tau+1}(x, u) \leftarrow \rho(x, u) + \gamma Q_\tau(f(x, u), h(f(x, u)))$
  **end for**
**until** convergence to $Q^h$

# Cleaning robot: Policy iteration demo

Initial policy: always move left



# Cleaning robot: Policy iteration

$$Q_{\tau+1}(x,u) \leftarrow \rho(x,u) + \gamma Q_\tau(f(x,u), h(f(x,u)))$$

$$h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x,u)$$

| | $x=0$ | $x=1$ | $x=2$ | $x=3$ | $x=4$ | $x=5$ |
|---|---|---|---|---|---|---|
| $h_0$ | * | $-1$ | $-1$ | $-1$ | $-1$ | * |
| $Q_0$ | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 |
| $Q_1$ | 0 ; 0 | 1 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 5 | 0 ; 0 |
| $Q_2$ | 0 ; 0 | 1 ; 0 | 0.5 ; 0 | 0 ; 0 | 0 ; 5 | 0 ; 0 |
| $Q_3$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 0 | 0.25 ; 0 | 0 ; 5 | 0 ; 0 |
| $Q_4$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 0.125 | 0.25 ; 0 | 0.125 ; 5 | 0 ; 0 |
| $Q_5$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 0.125 | 0.25 ; 0.0625 | 0.125 ; 5 | 0 ; 0 |
| $Q_6$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 0.125 | 0.25 ; 0.0625 | 0.125 ; 5 | 0 ; 0 |
| $h_1$ | * | $-1$ | $-1$ | $-1$ | 1 | * |

...algorithm continues...

# Cleaning robot: Policy iteration (cont'd)

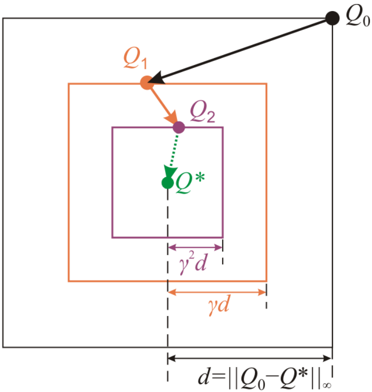
| | $x=0$ | $x=1$ | $x=2$ | $x=3$ | $x=4$ | $x=5$ |
|---|---|---|---|---|---|---|
| $h_1$ | * | $-1$ | $-1$ | $-1$ | 1 | * |
| $Q_0$ | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 |
| ... | ... | ... | ... | ... | ... | ... |
| $Q_5$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 0.125 | 0.25 ; 2.5 | 0.125 ; 5 | 0 ; 0 |
| $h_2$ | * | $-1$ | $-1$ | 1 | 1 | * |
| $Q_0$ | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 |
| ... | ... | ... | ... | ... | ... | ... |
| $Q_4$ | 0 ; 0 | 1 ; 0.25 | 0.5 ; 1.25 | 0.25 ; 2.5 | 1.25 ; 5 | 0 ; 0 |
| $h_3$ | * | $-1$ | 1 | 1 | 1 | * |
| $Q_0$ | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 | 0 ; 0 |
| ... | ... | ... | ... | ... | ... | ... |
| $Q_5$ | 0 ; 0 | 1 ; 0.625 | 0.5 ; 1.25 | 0.625 ; 2.5 | 1.25 ; 5 | 0 ; 0 |
| $h_4$ | * | $-1$ | 1 | 1 | 1 | * |

# Convergence of Q-iteration

- Each iteration is a contraction with factor $\gamma$:

$$\|Q_{\ell+1} - Q^*\|_\infty \leq \gamma \|Q_\ell - Q^*\|_\infty$$

$\Rightarrow$ Q-iteration **monotonically converges** to $Q^*$, **with convergence rate $\gamma$** $\Rightarrow \gamma$ helps convergence



# Stopping condition

- Convergence to $Q^*$ only guaranteed asymptotically, as $\ell \to \infty$
- In practice, algorithm can be stopped when:

$$\|Q_{\ell+1} - Q_\ell\|_\infty \le \varepsilon_{\text{qiter}}$$

# Convergence of policy iteration

Policy evaluation component – like Q-iteration:

- Policy evaluation is a contraction with factor $\gamma$

$\Rightarrow$ **monotonic convergence** to $Q^h$, with rate $\gamma$

Complete policy iteration algorithm:

- Policy is either improved or already optimal
- But the maximum number of improvements is finite! ($|U|^{|X|}$)

$\Rightarrow$ **convergence** to $h^*$ in a finite number of iterations

# Stopping conditions

In practice:

- Policy evaluation can be stopped when:

$$\|Q_{\tau+1} - Q_\tau\| \leq \varepsilon_{\text{peval}}$$

- Policy iteration can be stopped when:

$$\|h_{\ell+1} - h_\ell\| \leq \varepsilon_{\text{piter}}$$

- Note: $\varepsilon_{\text{piter}}$ can be taken 0!

# Q-iteration vs. policy iteration

## Number of iterations to convergence

- Q-iteration $>$ policy iteration

## Complexity

- one iteration of Q-iteration
  $>$ one iteration of policy evaluation
- complete Q-iteration **???** complete policy iteration

1. Introduction
2. Problem definition
3. Dynamic programming, DP
4. **Monte Carlo, MC**
5. Temporal differences, TD

# Algorithm landscape

By model usage:

- Model-based: $f$, $\rho$ known
- Model-free: $f$, $\rho$ unknown (reinforcement learning)

By interaction level:

- Offline: algorithm runs in advance
- Online: algorithm runs with the system

Exact vs. approximate:

- Exact: $x, u$ small number of discrete values
- Approximate: $x, u$ continuous (or many discrete values)

Next: Online RL, still in the discrete case

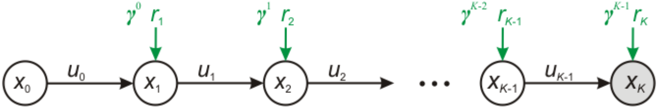
# Policy evaluation change

To find $Q^h$:

- So far: model-based policy evaluation
- Reinforcement learning: **model not available!**
- Learn $Q^h$ from data or by **online interaction with the system**

# Monte Carlo policy evaluation

Recall: $Q^h(x_0, u_0) = \rho(x_0, u_0) + \gamma R^h(x_1)$




- Trial (trajectory) from $(x_0, u_0)$ to **terminal** $x_K$ using $u_1 = h(x_1)$, $u_2 = h(x_2)$ etc.

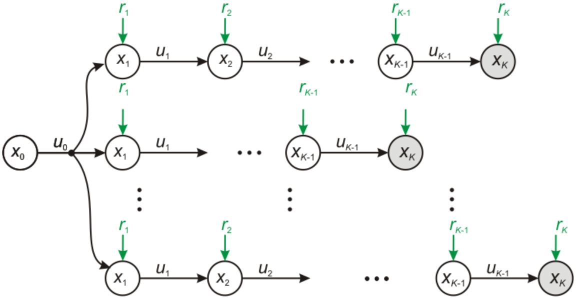
$\Rightarrow$ $Q^h(x_0, u_0) =$ return along trajectory:

$$Q^h(x_0, u_0) = \sum_{j=0}^{K-1} \gamma^j r_{j+1}$$

- Furthermore, at each step:

$$Q^h(x_k, u_k) = \sum_{j=k}^{K-1} \gamma^{j-k} r_{j+1}$$

# Stochastic case idea




Average return samples over multiple trajectories

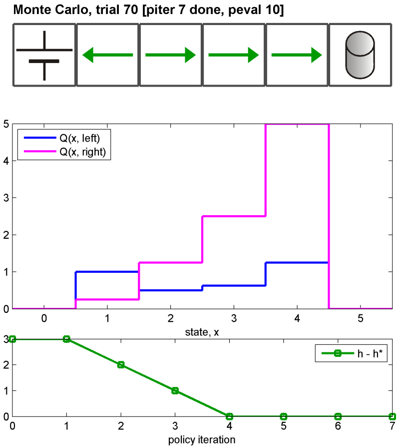
# Monte Carlo policy iteration

**Monte Carlo policy iteration**

**for** each iteration $\ell$ **do**
  run $N$ trials applying $h_\ell$
  reset accumulator $A(x, u)$, counter $C(x, u)$ to 0
  **for** each step $k$ of each trial $i$ **do**
    $A(x_k, u_k) \leftarrow A(x_k, u_k) + \sum_{j=k}^{K_i-1} \gamma^{j-k} r_{i,j+1}$ (return)
    $C(x_k, u_k) \leftarrow C(x_k, u_k) + 1$
  **end for**
  $Q^{h_\ell}(x, u) \leftarrow A(x, u)/C(x, u)$
  $h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x, u)$
**end for**

Note: must ensure **a terminal state is always reached**!

# Cleaning robot: Monte Carlo demo



# Need for exploration

$$Q^h(x,u) \leftarrow A(x,u)/\mathbf{C(x,u)}$$

How to ensure $C(x,u) > 0$ – **information** about each $(x,u)$?

1. Select representative initial states $x_0$
2. Actions:
   $u_0$ representative, sometimes different from $h(x_0)$
   and in addition, perhaps:
   $u_k$ representative, sometime different from $h(x_k)$

# Exploration-exploitation

- **Exploration** needed:
  actions different from the current policy
- **Exploitation** of current knowledge also needed:
  current policy must be applied

> Exploration-exploitation dilemma
> – essential in all RL algorithms

(not just in MC)

# Exploration-exploitation: $\varepsilon$-greedy strategy

- Simple solution: **$\varepsilon$-greedy**

$$u_k = \begin{cases} h(x_k) = \arg\max_u Q(x_k, u) & \text{with probability } (1 - \varepsilon_k) \\ \text{a random action} & \text{w.p. } \varepsilon_k \end{cases}$$

- Exploration probability $\varepsilon_k \in (0, 1)$ usually decreased over time

# Optimistic policy improvement

- Policy unchanged for $N$ trials

$\Rightarrow$ Algorithm learns slowly

- Policy improvement after each trial = **optimistic**

# Optimistic Monte Carlo

## Optimistic Monte Carlo method

init accumulator $A(x, u)$, counter $C(x, u)$ to 0

**for** each trial **do**

  execute trial, e.g. applying $\varepsilon$-greedy:

$$u_k = \begin{cases} \arg\max_u Q(x_k, u) & \text{w.p. } (1 - \varepsilon_k) \\ \text{random} & \text{w.p. } \varepsilon_k \end{cases}$$

  **for** each step $k$ **do**

    $A(x_k, u_k) \leftarrow A(x_k, u_k) + \sum_{j=k}^{K-1} \gamma^{j-k} r_{j+1}$

    $C(x_k, u_k) \leftarrow C(x_k, u_k) + 1$

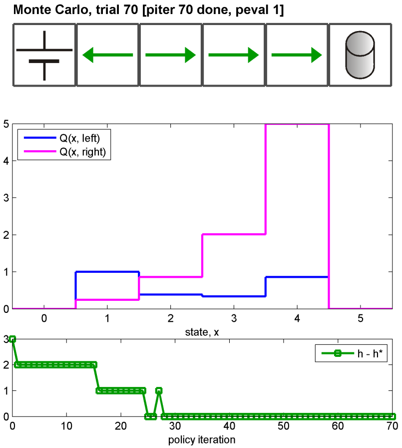
  **end for**

  $Q(x, u) \leftarrow A(x, u)/C(x, u)$

**end for**

- $h$ implicit, greedy in $Q$
- $Q$ updated $\Rightarrow$ implicit improvement of policy $h$

# Cleaning robot: Optimistic Monte Carlo demo



# DP perspective

1. Start from policy evaluation:
$$Q_{\tau+1}(x, u) \leftarrow \rho(x, u) + \gamma Q_\tau(f(x, u), h(f(x, u)))$$

2. Instead of model, use the **transition** at each step $k$ $(x_k, u_k, x_{k+1}, r_{k+1}, u_{k+1})$:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note: $x_{k+1} = f(x_k, u_k), r_{k+1} = \rho(x_k, u_k), u_{k+1} \sim h(x_{k+1})$

3. Turn into **incremental update**:
$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$
$\alpha_k \in (0, 1]$ learning rate

# Intermediate algorithm

**Temporal differences for policy $h$ evaluation**

**for** each trial **do**
  init $x_0$, choose initial action $u_0$
  **repeat** at each step $k$
    apply $u_k$, measure $x_{k+1}$, receive $r_{k+1}$
    choose **next** action $u_{k+1} \sim h(x_{k+1})$
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$
      $[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$
  **until** trial finished
**end for**

# MC perspective

**Temporal differences for policy $h$ evaluation**

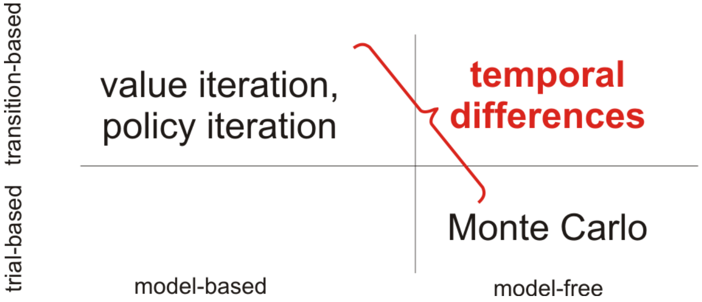
**for** each trial **do**
  ...
  **repeat** each step $k$
    apply $u_k$, measure $x_{k+1}$, receive $r_{k+1}$
    $Q(x_k, u_k) \leftarrow ...Q...$
  **until** trial finished
**end for**

Monte Carlo

**for** each trial **do**
  execute trial
  ...
  $Q(x, u) \leftarrow A(x, u)/C(x, u)$
**end for**

# MC and DP perspectives

- Learn from online interaction: like MC, unlike DP
- Update after each transition, using previous Q-values: like DP, unlike MC



# Exploration-exploitation

choose next action $u_{k+1} \sim h(x_{k+1})$

- Information about $(x, u) \neq (x, h(x))$ needed
  $\Rightarrow$ **exploration**
- $h$ must be followed
  $\Rightarrow$ **exploitation**

- E.g. $\varepsilon$-greedy:

$$u_{k+1} = \begin{cases} h(x_{k+1}) & \text{w.p. } (1 - \varepsilon_{k+1}) \\ \text{random} & \text{w.p. } \varepsilon_{k+1} \end{cases}$$

# Policy improvement

- Previous algorithm: $h$ fixed
- Improving $h$: simplest, after each transition

$\Rightarrow$ interpretation: policy iteration
**optimistic** at the transition level

- $h$ implicit, greedy in $Q$
  (update $Q \Rightarrow$ implicitly improve $h$)

# SARSA

## SARSA with $\varepsilon$-greedy exploration

**for** each trial **do**
  init $x_0$

$$u_0 = \begin{cases} \arg\max_u Q(x_0, u) & \text{w.p. } (1 - \varepsilon_0) \\ \text{random} & \text{w.p. } \varepsilon_0 \end{cases}$$

  **repeat** at each step $k$
    apply $u_k$, measure $x_{k+1}$, receive $r_{k+1}$

$$u_{k+1} = \begin{cases} \arg\max_u Q(x_{k+1}, u) & \text{w.p. } (1 - \varepsilon_{k+1}) \\ \text{random} & \text{w.p. } \varepsilon_{k+1} \end{cases}$$

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

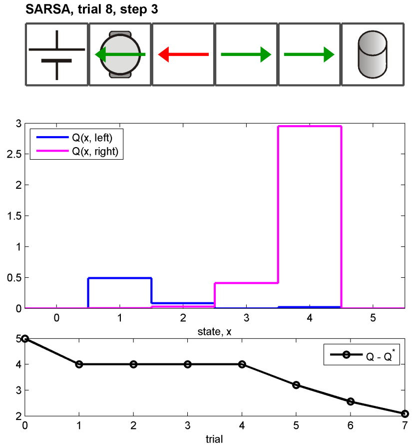
  **until** trial finished
**end for**

# Origin of the name SARSA

$(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1}) =$
(**S**tate, **A**ction, **R**eward, **S**tate, **A**ction) = **SARSA**

# Cleaning robot: SARSA demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$ (constant)
$x_0 = 2$ or 3 (random)



# Q-learning

1. Similarly to SARSA, start from Q-iteration:
$$Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$$

2. Instead of model, use at each step $k$ the **transition** $(x_k, u_k, x_{k+1}, r_{k+1})$:
$$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$
Note: $x_{k+1} = f(x_k,u_k)$, $r_{k+1} = \rho(x_k,u_k)$

3. Turn into **incremental** update:
$$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u') - Q(x_k,u_k)\right]$$

# Q-learning

### Q-learning with $\varepsilon$-greedy exploration

**for** each trial **do**
  init $x_0$
  **repeat** at each step $k$

$$u_k = \begin{cases} \arg\max_u Q(x_k, u) & \text{w.p. } (1 - \varepsilon_k) \\ \text{random} & \text{w.p. } \varepsilon_k \end{cases}$$

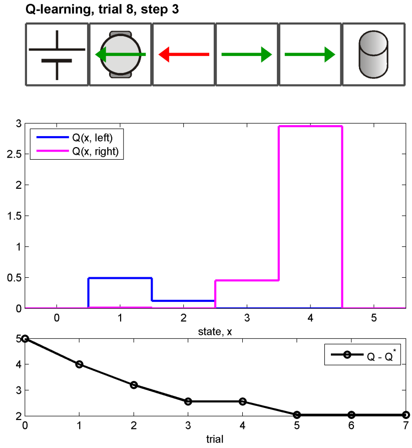
  apply $u_k$, measure $x_{k+1}$, receive $r_{k+1}$

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right]$$

  **until** trial finished
**end for**

# Cleaning robot: Q-learning demo

Parameters – like in SARSA: $\alpha = 0.2$, $\varepsilon = 0.3$ (constant)
$x_0 = 2$ or 3 (random)



# Convergence

Conditions for convergence to $Q^*$
in both SARSA and Q-learning:

1. All pairs $(x, u)$ continue to be updated:
   requires **exploration**, e.g. $\varepsilon$-greedy
2. Technical conditions on $\alpha_k$ (goes to 0, $\sum_{k=0}^{\infty} \alpha_k^2 =$ finite,
   but not too fast, $\sum_{k=0}^{\infty} \alpha_k \to \infty$)

In addition, for SARSA:

3. Policy must become greedy asymptotically
   e.g. for $\varepsilon$-greedy, $\lim_{k\to\infty} \varepsilon_k = 0$

# Discussion

SARSA **on-policy**

- Always updates towards the Q-function of the current policy

Q-learning **off-policy**

- No matter what the current policy, always updates towards optimal Q-function

Both algorithms remain valid for stochastic problems

# Discussion (cont'd)

Advantages of temporal differences

- Easy to understand and implement
- Low complexity $\Rightarrow$ fast execution

SARSA vs. Q-learning

- SARSA complexity smaller than Q-learning (no max in the update)
- Performance: better algo depends on the problem

$\alpha_k$, $\varepsilon_k$ sequences **greatly influence** performance

Main disadvantage: TD require large number of data

Two possible solutions:

- Eligibility traces
- Experience replay

# References for Part I


- Bertsekas, *Dynamic Programming and Optimal Control*, vol. 2, 4th ed., 2012.
- Sutton & Barto, *Reinforcement Learning: An Introduction*, 1998.
- Szepesvári, *Algorithms for Reinforcement Learning*, 2010.
- Buşoniu, Babuška, De Schutter, & Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 2010.