Reinforcement Learning

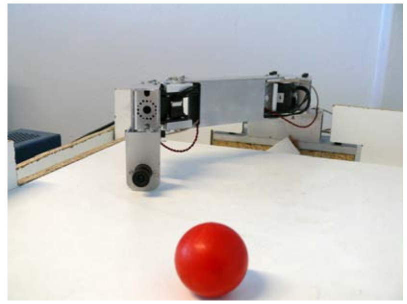
Part I: The Classical Setting

Lucian Buşoniu, Jelmer van Ast, Robert Babuška

Knowledge-Based Control Systems
2010-03-01

Demo: RL for a robot goalkeeper

Learn how to catch ball, using video camera image



Outline

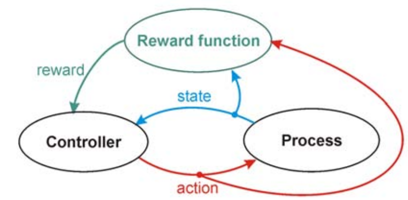
Why learning?

Learning can find solutions that:

1. cannot be found in advance
   - problem too complex (e.g., controlling highly nonlinear systems)
   - problem not fully known beforehand (e.g., robotic exploration of extraterrestrial planets)
2. steadily improve
3. adapt to time-varying environments

Essential for any **intelligent** system

Principle of RL

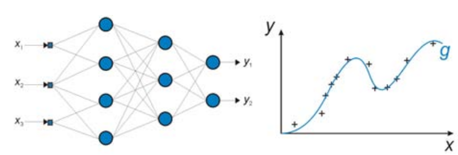


- Interact with a system through states and actions
- Receive rewards as performance feedback
- Inspired by human and animal learning

RL on the Machine Learning spectrum

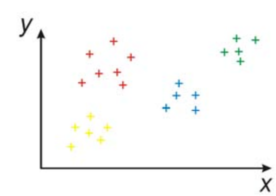
Spectrum: Supervised learning

- For each input sample x, **correct output** y is known
- Infer input-output relationship $y \approx g(x)$
- Example: neural networks

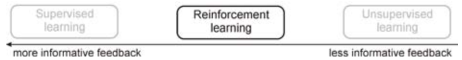


Spectrum: Unsupervised learning

- Only input samples x available – **no outputs**
- Find patterns in the data
- Example: clustering



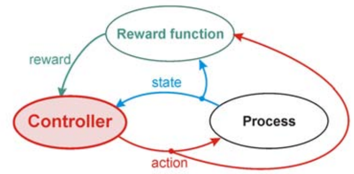
Spectrum: Reinforcement learning



- Correct outputs not available, **only rewards**
- Find optimal control behavior

Reinforcement learning = Control

Reinforcement learning is about **control**:
optimal, adaptive, and model-free



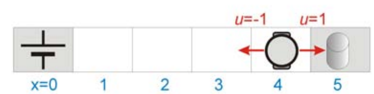
This presentation: **classical RL** – discrete states and actions

A simple cleaning robot example

- Cleaning robot in a 1-D world
- Either pick up trash (reward $+5$) or power pack (reward $+1$)
- After picking up item, episode terminates

Cleaning robot: State & action

- Robot in given state x (cell)
- and takes action u (e.g., move right)

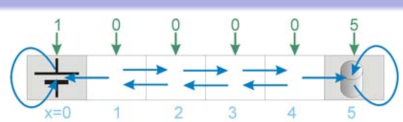


- State space $X = \{0, 1, 2, 3, 4, 5\}$
- Action space $U = \{-1, 1\} = \{\text{left, right}\}$

Cleaning robot: Transition & reward

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

Cleaning robot: Transition & reward functions



- Transition function (process behavior):
$$x' = f(x, u) = \begin{cases} x & \text{if } x \text{ is terminal (0 or 5)} \\ x + u & \text{otherwise} \end{cases}$$
- Reward function (immediate performance):
$$r = \rho(x, u) = \begin{cases} 1 & \text{if } x = 1 \text{ and } u = -1 \text{ (powerpack)} \\ 5 & \text{if } x = 4 \text{ and } u = 1 \text{ (trash)} \\ 0 & \text{otherwise} \end{cases}$$
- **Note:** Terminal states cannot be left & do not accumulate rewards!

Markov decision process

1. State space X
2. Action space U
3. Transition function $x' = f(x, u)$
4. Reward function $r = \rho(x, u)$

... form a **Markov decision process**

Note: stochastic formulation possible

Policy

- Policy h: mapping from x to u (state feedback)
- Determines controller behavior

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

1. Reinforcement learning basics
   - Introduction
   - Elements of RL
   - RL solution
2. Algorithms
3. Accelerating RL

Learning goal

Find h that maximizes discounted return:

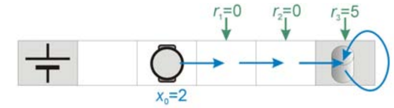
$$R^h(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, h(x_k))$$

from any x_0

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

- induces a "pseudo-horizon" for optimization
- bounds infinite sum
- encodes increasing uncertainty about the future
- helps convergence of algorithms

Cleaning robot: Return



Assume h always goes right

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

Because x_3 is terminal, all remaining rewards are 0

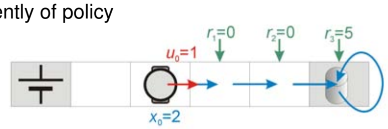
Q-function

- **Q-function** of policy h:

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

(return after taking u_0 in x_0 and then following h)

- Simply fix the first action in the sequence, independently of policy



- Why Q-function? Useful to choose actions (later)

Bellman equation

- Develop Q-function one step ahead:

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

Also, $x_1 = f(x_0, u_0)$

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

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

Optimal solution

- **Optimal Q-function**:

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

$\Rightarrow$ Greedy policy in Q^*:

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

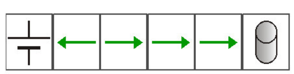
is **optimal** (achieves maximal returns)

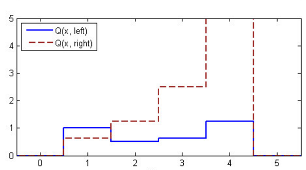
Bellman optimality equation (for Q^*)

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

Cleaning robot: Optimal solution

Discount factor $\gamma = 0.5$





1. Reinforcement learning basics
2. Algorithms
   - Taxonomy
   - Q-learning
   - SARSA
3. Accelerating RL

Types of algorithms

By model knowledge

1. Model-based – dynamic programming
   f, ρ known
2. Model-free – proper reinforcement learning
   f, ρ unknown, only transition data (x, u, x', r) available
3. Model-learning RL
   estimate f and ρ from transition data

Types of algorithms (cont'd)

By level of interaction

1. Offline
   data collected in advance
2. Online
   controller learns by interacting with the process

By path to optimal solution

1. Off-policy
   find Q^*, use it to compute h^*
2. On-policy
   find Q^h, improve h, repeat

Algorithms in this lecture

Online model-free reinforcement learning:

Off-policy	On-policy
Q-learning	SARSA

Off-policy online RL: Q-learning

Recall off-policy: **find Q^***, use it to compute h^*

1. Take Bellman optimality equation at some (x, u):
   $$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u),u')$$
2. Turn into **iterative update**:
   $$Q(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q(f(x,u),u')$$
3. Instead of model f, ρ, use **transition sample** $(x_k, u_k, x_{k+1}, r_{k+1})$ at each step k:
   $$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$
   Note: $x_{k+1} = f(x_k,u_k), r_{k+1} = \rho(x_k,u_k)$

Q-learning (cont'd)

3. Instead of model, use **transition sample** $(x_k, u_k, x_{k+1}, r_{k+1})$ at each step k:
   $$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$
   Note: $x_{k+1} = f(x_k,u_k), r_{k+1} = \rho(x_k,u_k)$
4. Finally, make update **incremental**:
   $$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u') - Q(x_k,u_k)]$$
   $\alpha_k \in (0,1]$ learning rate

Complete Q-learning algorithm

Q-learning

for every trial **do**
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    $Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u') - Q(x_k,u_k)]$
  until terminal state
end for

Exploration-exploitation tradeoff

- Essential condition for convergence to Q^*:
  all (x,u) pairs must be visited infinitely often

⇒ **Exploration** necessary:
  sometimes, choose actions randomly

- **Exploitation** of current knowledge is also necessary:
  sometimes, choose actions greedily:
  $$u_k = \arg\max_{\bar{u}} Q(x_k, \bar{u})$$

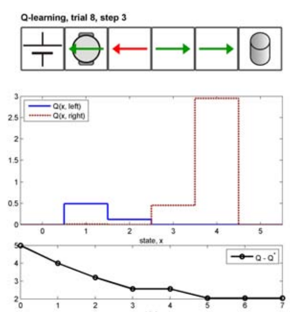
Exploration-exploitation tradeoff crucial for performance of online RL

Exploration-exploitation: ε-greedy strategy

- Simple solution: **ε-greedy**
  $$u_k = \begin{cases} \arg\max_{\bar{u}} Q(x_k,\bar{u}) & \text{with probability } (1-\varepsilon_k) \\ \text{a random action} & \text{with probability } \varepsilon_k \end{cases}$$
- Exploration probability $\varepsilon_k \in (0,1)$ is usually decreased over time

Cleaning robot: Q-learning demo

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



1. Reinforcement learning basics
2. Algorithms
   - Taxonomy
   - Q-learning
   - SARSA
3. Accelerating RL

On-policy online RL: SARSA

Recall on-policy: **find Q^h**, improve h, repeat

Similar to Q-learning:

1. Take Bellman equation for Q^h, at some (x, u):
$$Q^h(x,u) = \rho(x,u) + \gamma Q^h(f(x,u), h(f(x,u)))$$
2. Turn into iterative update:
$$Q(x,u) \leftarrow \rho(x,u) + \gamma Q(f(x,u), h(f(x,u)))$$
3. Use sample $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note: $u_{k+1} = h(f(x_k, u_k))$, **h = policy being followed**

SARSA (cont'd)

3. Use sample $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note: $u_{k+1} = h(f(x_k, u_k))$, **h = policy being followed**
4. Make update incremental:
$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

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

Complete SARSA algorithm

SARSA

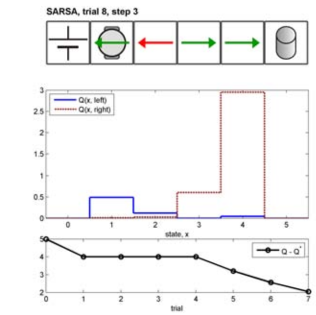
for every trial **do**
  initialize x_0, choose initial action u_0
  repeat for each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose **next** action u_{k+1}
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$
  until terminal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

Exploration-exploitation in SARSA

- For convergence—besides infinite exploration—SARSA requires **policy to eventually become greedy**
- E.g., ε-greedy
$$u_k = \begin{cases} \arg\max_{\bar{u}} Q(x_k, \bar{u}) & \text{with probability } (1-\varepsilon_k) \\ \text{a random action} & \text{with probability } \varepsilon_k \end{cases}$$
with $\lim_{k\to\infty} \varepsilon_k = 0$
- Greedy actions $\Rightarrow$ policy implicitly improved!
(Recall on-policy: find Q^h, **improve** h, repeat)

Cleaning robot: SARSA demo

Parameters like Q-learning: $\alpha = 0.2$, $\varepsilon = 0.3$ (constant)
$x_0 = 2$ or 3 (randomly)



1. Reinforcement learning basics
2. Algorithms
3. Accelerating RL
   - Eligibility traces
   - Experience replay

Accelerating RL

In practice, transition data costs:

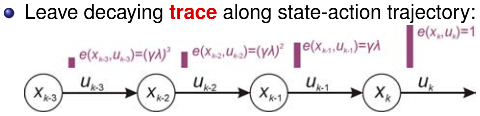
- time
- profits (suboptimal performance due to exploration)
- process wear & tear

Fast RL = **use data efficiently**

(computational demands are secondary)

Eligibility traces

- Leave decaying **trace** along state-action trajectory:



- $\lambda \in [0, 1]$ decay rate, γ discount factor
- Implementation:

$e(x, u) \leftarrow 0$ for all x, u
for each step k **do**
  $e(x, u) \leftarrow \lambda\gamma e(x, u)$ for all x, u
  $e(x_k, u_k) \leftarrow 1$
end for

Q(λ)-learning

- Recall basic Q-learning only updates $Q(x_k, u_k)$:
$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$
- Q(λ)-learning updates **all eligible pairs**:
$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)] \textbf{ for all } x, u$$
- Note: exploratory actions break causality
$\Rightarrow$ reset eligibility trace to 0

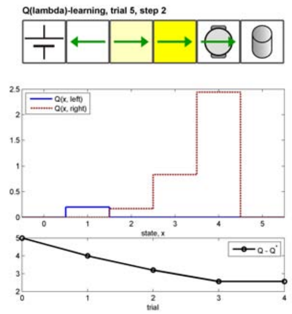
Complete Q(λ)-learning algorithm

Q(λ)-learning

for every trial **do**
  $e(x, u) \leftarrow 0$ for all x, u
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    if u_k exploratory **then** $e(x, u) \leftarrow 0$ for all x, u
    else $e(x, u) \leftarrow \lambda\gamma e(x, u)$ for all x, u
    end if
    $e(x_k, u_k) \leftarrow 1$
    $Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot$
      $[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$ for all x, u
  until terminal state
end for

Cleaning robot: Q(λ)-learning demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$ (like basic Q-learning), $\lambda = 0.5$
$x_0 = 2$ or 3 (randomly)



SARSA(λ)

Similar to Q-learning:

- Basic SARSA:
$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$
- SARSA(λ)-learning:
$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)] \textbf{ for all } x, u$$
- SARSA on-policy, including exploration
$\Rightarrow$ exploratory actions not a problem

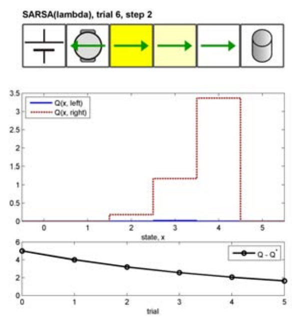
Complete SARSA(λ) algorithm

SARSA(λ)

for every trial **do**
  $e(x, u) \leftarrow 0$ for all x, u
  initialize x_0, choose initial action u_0
  repeat for each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose next action u_{k+1}
    $e(x, u) \leftarrow \lambda\gamma e(x, u)$ for all x, u
    $e(x_k, u_k) \leftarrow 1$
    $Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot$
      $[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$ for all x, u
  until terminal state
end for

Cleaning robot: SARSA(λ) demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$ (like basic SARSA), $\lambda = 0.5$
$x_0 = 2$ or 3 (randomly)



Effects of eligibility trace

- Accelerates learning: fewer trials to convergence
- However: too large λ can make algorithm settle on suboptimal solution!

Experience replay (ER)

- Store each transition sample $(x_k, u_k, x_{k+1}, r_{k+1})$ into a database
- At every step, **replay** N transitions from the database
- Improvement: replay most informative samples first: **prioritized sweeping**

ER Q-learning

Q-learning with experience replay

for every trial **do**
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$
      $[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$
    add $(x_k, u_k, x_{k+1}, r_{k+1})$ to database
    ReplayExperience
  until terminal state
end for

ER Q-learning (cont'd)

ReplayExperience

loop N times
  retrieve a sample (x, u, x', r) from database
  $Q(x, u) \leftarrow Q(x, u) + \alpha \cdot$
    $[r + \gamma \max_{u'} Q(x', u') - Q(x, u)]$
end loop

Summary

- **Reinforcement learning** = optimal, adaptive, model-free control
- Principle: reward signal as performance feedback
- Inspired from human and animal learning, but solid mathematical foundation
- Classical RL: small, discrete X and U (this presentation)

A final look at the algorithms

	Off-policy	On-policy
Basic RL	Q-learning Param: $\gamma, \alpha_k, \varepsilon_k$	SARSA Param: $\gamma, \alpha_k, \varepsilon_k$
RL with eligibility traces	Q(λ)-learning Param: $\gamma, \alpha_k, \varepsilon_k, \lambda$	SARSA(λ) Param: $\gamma, \alpha_k, \varepsilon_k, \lambda$

Typical parameter values:

γ 0.9 or larger
α_k under 0.5 or diminishing schedule
ε_k around 0.1 or diminishing schedule
λ between 0.5 and 0.9

Outlook

- Other algorithms: actor-critic, model-learning, policy search, etc.
- Continuous X, U: Part II – RL using function approximation
- State not fully measurable: "partially observable Markov decision process"
- RL for distributed (multi-agent) control

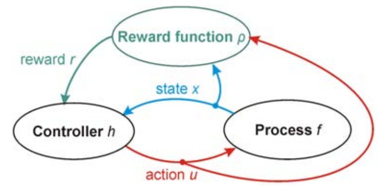
Reinforcement Learning

Part II: Approximate RL for Continuous-Space Control

Lucian Buşoniu, Jelmer van Ast, Robert Babuška

Knowledge-Based Control Systems
2010-03-03

Principle of RL



- Interact with a system through states and actions
- Receive rewards as performance feedback

This presentation: **approximate RL**
– continuous states & actions

Outline

1. Introduction
2. Classical offline algorithms
3. Approximate algorithms for continuous spaces

Recall: Solution of the RL problem

- Q-function Q^h of policy h
- Optimal Q-function $Q^* = \max_h Q^h$
  Satisfies Bellman optimality equation:
  $$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u),u')$$
- Optimal policy h^* – greedy in Q^*:
  $$h^*(x) = \arg\max_u Q^*(x,u)$$

Why approximation?

- Classical RL – tabular representation of Q-functions: separate Q-value for each x and u
- But in real-life control, X, U **continuous**!
  Tabular representation impossible

$\Rightarrow$ need to **approximate the Q-function**

Recall: Types of algorithms

By model knowledge
1. Model-based – f and ρ known (dynamic programming)
2. Model-free – no f and ρ, only transition data (RL)
3. Model-learning – estimate f and ρ from transition data

By level of interaction
1. Offline – data collected in advance
2. Online – learn by interacting with the process

By path to optimal solution
1. Off-policy – find Q^*, use it to compute h^*
2. On-policy – find Q^h, improve h, repeat

Algorithms considered

	Off-policy	On-policy
(previous lecture) Classical online RL	Q-learning	SARSA
(this lecture) Classical offline DP	Q-iteration	Policy iteration
Approximate offline	**fuzzy** Q-iteration	**approximate** policy iteration
Approximate online	**approximate** Q-learning	**online approximate** policy iteration

Technical focus: Q-iteration & fuzzy Q-iteration

---

1. Introduction
2. Classical offline algorithms
3. Approximate algorithms for continuous spaces

Classical offline algorithms

	Off-policy	On-policy
Classical offline DP	Q-iteration	Policy iteration
Approximate offline	fuzzy Q-iteration	approximate policy iteration
Approximate online	approximate Q-learning	online approximate policy iteration

Offline, off-policy: Q-iteration

- Turn Bellman optimality equation:

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

into an **iterative update**:

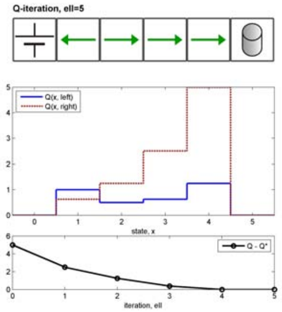
Q-iteration

repeat at each iteration ℓ
  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: $\gamma = 0.5$

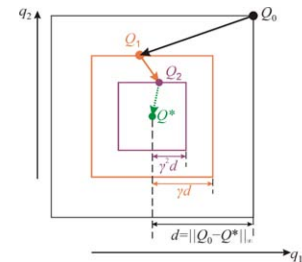


Q-iteration convergence

- Each update is a contraction with factor γ:

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

$\Rightarrow$ Q-iteration **monotonically converges** to Q^*



Offline, on-policy: Policy iteration

- Recall on-policy: find Q^h, improve h, repeat

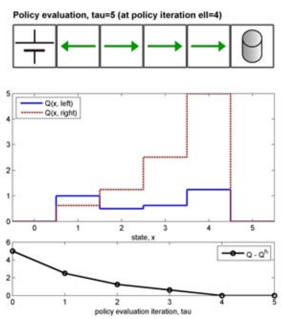
Policy iteration

starting from an initial policy
repeat at each iteration ℓ
  policy evaluation: find Q^{h_ℓ}
  policy improvement: $h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x,u)$
until convergence to h^*

- Policy evaluation: iterative, from Bellman equation for Q^h (like Q-iteration)

Cleaning robot: Policy iteration demo

Initial policy: always go left



Approximate algorithms for continuous spaces

	Off-policy	On-policy
Classical offline DP	Q-iteration	Policy iteration
Approximate offline	**fuzzy** Q-iteration	**approximate** policy iteration
Approximate online	**approximate** Q-learning	**online approximate** policy iteration

Q-function approximation

- In real-life control, X, U continuous

$\Rightarrow$ **approximate Q-function** $\widehat{Q}$ must be used

- Usually, policy not approximated
  Greedy in $\widehat{Q}$, computed on demand for given x:

$$h(x) = \arg\max_u \widehat{Q}(x,u)$$

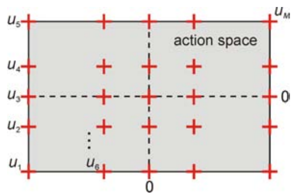
- Approximator must ensure **efficient** $\arg\max$ **solution**

Approximating over the action space

- Approximator must ensure efficient "arg max" solution

⇒ Typically: **action discretization**

- Choose M discrete actions $u_1, \ldots, u_M \in U$
  Solve "arg max" by explicit enumeration
- Example: grid discretization

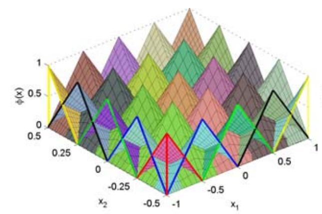


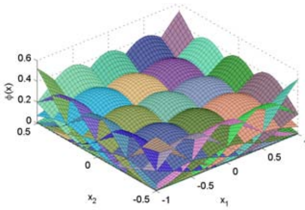
Approximating over the state space

- Typically: **basis functions**

$$\phi_1, \ldots, \phi_N : X \to [0, \infty)$$

- Usually normalized: $\sum_i \phi_i(x) = 1$
- E.g., fuzzy approximation, RBF network approximation





Linear Q-function approximation

Given:

1. N basis functions $\phi_1, \ldots, \phi_N$
2. M discrete actions $u_1, \ldots, u_M$

Store:

3. $N \times M$ matrix of **parameters** θ
   (one for each pair basis function–discrete action)

Approximate Q-function

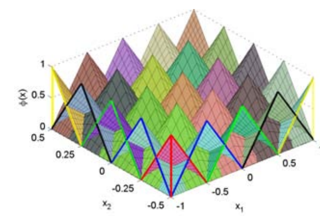
$$\widehat{Q}^{\theta}(x, u_j) = \sum_{i=1}^{N} \phi_i(x)\theta_{i,j} = [\phi_1(x) \ldots \phi_N(x)] \begin{bmatrix} \cdots & \begin{bmatrix} \theta_{1,j} \\ \vdots \\ \theta_{N,j} \end{bmatrix} & \cdots \end{bmatrix}$$

Offline, off-policy: Fuzzy Q-iteration

	Off-policy	On-policy
Classical offline DP	Q-iteration	Policy iteration
Approximate offline	**fuzzy** Q-iteration (DP)	approximate policy iteration
Approximate online	approximate Q-learning	online approximate policy iteration

Fuzzy approximator

- Basis functions: **pyramidal membership functions** (MFs)
  = cross-product of triangular MFs



- Each MF i has core (center) x_i
- $\theta_{i,j}$ can be seen as $\widehat{Q}(x_i, u_j)$

Fuzzy Q-iteration

Recall classical Q-iteration:

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x, u) = \rho(x, u) + \gamma \max_{u'} Q_\ell(f(x, u), u')$
  end for
until convergence

Fuzzy Q-iteration

repeat at each iteration ℓ
  for all cores x_i, discrete actions u_j **do**
    $\theta_{\ell+1,i,j} = \rho(x_i, u_j) + \gamma \max_{j'} \widehat{Q}^{\theta_\ell}(f(x_i, u_j), u_{j'})$
  end for
until convergence

Fuzzy Q-iteration policy

- Recall optimal policy:

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

- Fuzzy Q-iteration policy:

$$\widehat{h}^*(x) = \arg\max_{u_j,\, j=1,\ldots,M} \widehat{Q}^{\theta^*}(x, u_j)$$

(θ^* = converged parameter matrix)

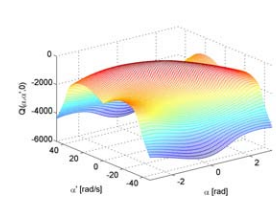
Example: Inverted pendulum swing-up

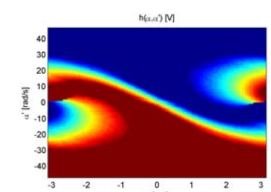
- $x = [\text{angle } \alpha, \text{ velocity } \dot{\alpha}]^{\mathrm{T}}$
- u = voltage
- $\rho(x, u) = -x^{\mathrm{T}} \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix} x - u^{\mathrm{T}} 1 u$
- Discount factor $\gamma = 0.98$

- Goal: stabilize pointing up
- Insufficient actuation ⇒ need to swing back & forth

Inverted pendulum: Near-optimal solution

Left: Q-function for $u = 0$ | Right: policy

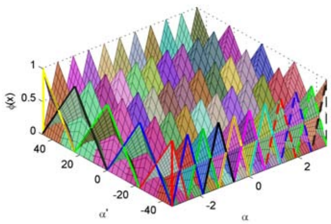
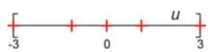




Replay

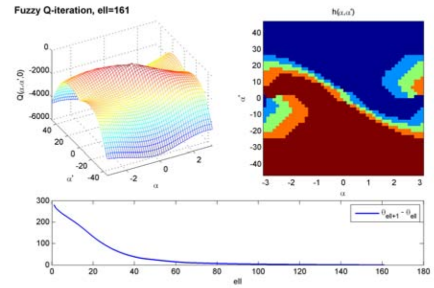
Inverted pendulum: Fuzzy Q-iteration demo

MFs: 41×21 equidistant grid
Discretization: 5 actions, logarithmically spaced around 0

Inverted pendulum: Fuzzy Q-iteration demo

Demo



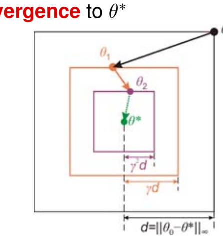
Fuzzy Q-iteration convergence

Like classical Q-iteration:

- Each update is a contraction with factor γ:

$$\|\theta_{\ell+1} - \theta^*\|_\infty \leq \gamma \|\theta_\ell - \theta^*\|_\infty$$

$\Rightarrow$ **Monotonic convergence** to θ^*



- θ^* leads to **near-optimal** $\widehat{Q}^*$, $\widehat{h}^*$

Offline, on-policy: Approximate policy iteration

	Off-policy	On-policy	
Classical offline DP	Q-iteration	Policy iteration	
Approximate offline	fuzzy Q-iteration	**approximate** policy iteration	(RL)
Approximate online	approximate Q-learning	online approximate policy iteration	

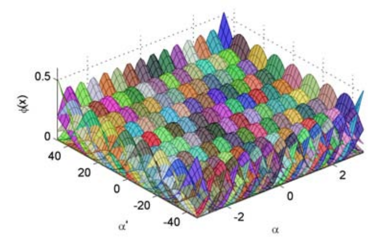
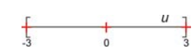
Approximate policy iteration

Recall classical policy iteration:
starting from an initial policy
repeat at each iteration ℓ
policy evaluation: find Q^{h_ℓ}
policy improvement: $h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x,u)$
until convergence

Approximate policy iteration (API)
starting from an initial policy
repeat at each iteration ℓ
approximate policy evaluation: find θ so that $\widehat{Q}^\theta \approx Q^{h_\ell}$
policy improvement: $h_{\ell+1}(x) \leftarrow \arg\max_u \widehat{Q}^\theta(x,u)$
until convergence

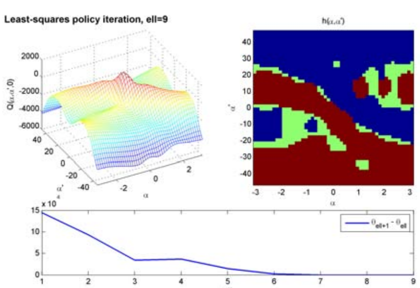
Inverted pendulum: API demo

Basis functions: 15×9 grid of RBFs
Discretization: 3 equidistant actions

Inverted pendulum: API demo

Demo



Online, on-policy: Online API

	Off-policy	On-policy	
Classical offline DP	Q-iteration	Policy iteration	
Approximate offline	fuzzy Q-iteration	approximate policy iteration	
Approximate online	approximate Q-learning	**online approximate** policy iteration	(RL)

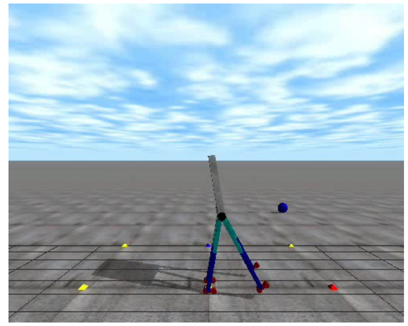
Inverted pendulum: Online API demo

Real-time learning control
Approximator: similar to offline API (except 11×11 RBFs)

Online, off-policy: Approximate Q-learning

	Off-policy	On-policy
Classical offline DP	Q-iteration	Policy iteration
Approximate offline	**fuzzy** Q-iteration	**approximate** policy iteration
Approximate online	**approximate** Q-learning (RL)	**online approximate** policy iteration

Demo: Q-learning for walking robot (Erik Schuitema)

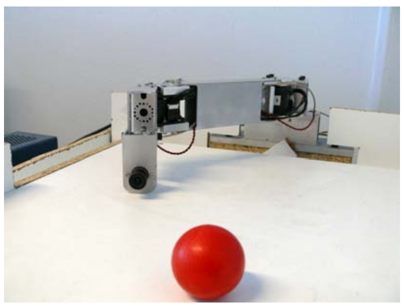


Recall: Experience replay

- Store each transition sample $(x_k, u_k, x_{k+1}, r_{k+1})$ into a database
- At every step, **replay** several transitions from the database

Demo: Q-learning for goalkeeper robot (Sander Adam)

Real-time learning control
Employs **experience replay**



Take-home message

Approximate reinforcement learning =
Learn how to **optimally** control
complex systems from scratch