Reinforcement Learning
Part I: The Classical Setting

Lucian Buşoniu, Jelmer van Ast, Robert Babuška

Knowledge-Based Control Systems
2010-03-01

Reinforcement learning basics

Why learning?

Learning can find solutions that:

- 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
- steadily improve
- 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

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
### Reinforcement learning basics

#### Introduction

- Reinforcement learning basics
  - Elements of RL
    - RL solution
  - Algorithms
  - Accelerating RL

### Elements of RL

#### 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)

### Cleaning robot: Transition & reward

- Robot reaches next state \( x' \)
- and receives reward \( r = \text{quality of transition} \)
  (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):
  \[
  r = p(x, u) = \begin{cases} 
  1 & \text{if } x = 1 \text{ and } u = -1 \text{ (powerpack)} \\
  5 & \text{if } x = 4 \text{ and } u = 1 \text{ (trash)} \\
  0 & \text{otherwise}
  \end{cases}
  \]

### Markov decision process

- State space \( X = \{0, 1, 2, 3, 4, 5\} \)
- Action space \( U = \{-1, 1\} = \{\text{left, right}\} \)
- Policy \( h: \text{mapping from } x \text{ to } u \) (state feedback)
- Determines controller behavior

### Policy

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

---

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

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

This presentation: classical RL – discrete states and actions

---

- Reinforcement learning basics
  - Introduction
  - Elements of RL
    - RL solution
  - Algorithms
  - Accelerating RL

---

- Reinforcement learning = Control

---

- Spectrum: Reinforcement learning
- Reinforcement learning basics
  - Algorithms
  - Accelerating RL

---

- Elements of RL
  - A simple cleaning robot example
  - Cleaning robot: State & action
  - Cleaning robot: Transition & reward
  - Cleaning robot: Transition & reward functions
  - Markov decision process
  - Policy

---

- TUDelft
Reinforcement learning basics

- Introduction
- Elements of RL
- RL solution

Algorithms

- Taxonomy
- Q-learning
- SARSA

Cleaning robot: Return

Assume $h$ always goes right

$$R^h(x_0) = 0 + 1r_1 + 2r_2 + 30 + 40 + \cdots$$

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

Bellman equation

- Develop Q-function one step ahead:
  $$Q^h(x_0, u_0) = \rho(x_0, u_0) + \gamma R^h(x_1) = \rho(x_0, u_0) + \gamma [\rho(x_1, h(x_1)) + \gamma R^h(x_2)] = \rho(x_0, u_0) + \gamma Q^h(x_1, h(x_1))$$
  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 $Q^*$)
  $$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

Types of algorithms

- Reinforcement learning basics
- Algorithms
  - Taxonomy
  - Q-learning
  - SARSA
- Accelerating RL

By model knowledge

- Model-based – dynamic programming $f, \rho$ known
- Model-free – proper reinforcement learning $f, \rho$ unknown, only transition data $(x, u, x', r)$ available
- Model-learning RL estimate $f$ and $\rho$ from transition data
Off-policy online RL: Q-learning

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

- Take Bellman optimality equation at some \((x, u)\):
  \[
  Q^*(x, u) = r(x, u) + \gamma \max_u Q^*(f(x, u), u')
  \]

- Turn into iterative update:
  \[
  Q(x, u) \leftarrow r(x, u) + \gamma \max_u Q(f(x, u), u')
  \]

- Instead of model \( f, \rho \), use transition sample
  \((x_k, u_k, x_{k+1}, f_{k+1})\) at each step \( k \):
  
  \[
  Q(x_k, u_k) \leftarrow f_{k+1} + \gamma \max_u Q(x_{k+1}, u')
  \]

- Note: \( x_{k+1} = f(x_k, u_k), f_{k+1} = \rho(x_k, u_k) \)

Q-learning (cont'd)

- Instead of model, use transition sample
  \((x, u, x_{k+1}, f_{k+1})\) at each step \( k \):
  
  \[
  Q(x, u) \leftarrow f_{k+1} + \gamma \max_u Q(x_{k+1}, u')
  \]

- Note: \( x_{k+1} = f(x, u), f_{k+1} = \rho(x, u) \)

- Finally, make update incremental:
  
  \[
  Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \]

  \[
  [f_{k+1} + \gamma \max_u Q(x_{k+1}, u') - Q(x_k, u_k)]
  \]

  \( \alpha_k \in (0, 1) \) 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 \]
        \[
        [f_{k+1} + \gamma \max_u Q(x_{k+1}, u') - Q(x_k, u_k)]
        \]
  - 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

- Exploration of current knowledge is also necessary:
  - sometimes, choose actions greedily:
    \( u_k = \arg \max_x Q(x_k, u) \)

- Exploration-exploitation tradeoff crucial for performance of online RL

Exploration-exploitation: \( \varepsilon \)-greedy strategy

- Simple solution: \( \varepsilon \)-greedy

  \[
  u_k = \begin{cases} 
  \arg \max_u Q(x_k, u) & \text{with probability } (1 - \varepsilon_k) \\
  \text{a random action} & \text{with probability } \varepsilon_k 
  \end{cases}
  \]

- 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)
Accelerating RL

**On-policy online RL: SARSA**

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

Similar to Q-learning:

- Take Bellman equation for $Q^*$, at some $(x, u)$:
  \[
  Q^*(x, u) = \rho(x, u) + \gamma Q^*(f(x, u), h(f(x, u)))
  \]

- Turn into iterative update:
  \[
  Q(x, u) \leftarrow \rho(x, u) + \gamma Q(f(x, u), h(f(x, u)))
  \]

- 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

**Exploration-exploitation in SARSA**

- For convergence—besides infinite exploration—SARSA requires policy to eventually become greedy

  - E.g., $\epsilon$-greedy
    \[
    u_k = \begin{cases} 
      \arg \max_u Q(x_k, u) & \text{with probability } (1 - \epsilon_k) \\
      \text{a random action} & \text{with probability } \epsilon_k
    \end{cases}
    \]

    with $\lim_{k \to \infty} \epsilon_k = 0$

- Greedy actions $\Rightarrow$ policy implicitly improved!
  (Recall on-policy: find $Q^*$, improve $h$, repeat)

**SARSA (cont’d)**

- Use sample $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ at each step $k$:
  \[
  Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})
  \]

- Make update incremental:
  \[
  Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \left[ r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k) \right]
  \]

(Cleaning robot: SARSA demo)

Parameters like Q-learning: $\alpha = 0.2$, $\epsilon = 0.3$ (constant)

$x_0 = 2$ or $3$ (randomly)

**Eligibility traces**

- Leave decaying trace along state-action trajectory:
  \[
  e(x_k, u_k) \leftarrow 0 \text{ for all } x, u \\
  \text{for each step } k \text{ do } \\
  e(x_k, u_k) \leftarrow \gamma e(x_k, u_k) \text{ for all } x, u \\
  e(x_k, u_k) \leftarrow 1
  \]

- $\lambda \in [0, 1]$ decay rate, $\gamma$ discount factor

- Implementation:

**Complete SARSA algorithm**

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 $u_{k+1}$
        \[
        Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \left[ r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k) \right]
        \]
        until terminal state
    end for

- In practice, transition data costs:
  - time
  - profits (suboptimal performance due to exploration)
  - process wear & tear

- Fast RL = use data efficiently

  (computational demands are secondary)
**Q(\(\lambda\))-learning**

- Recall basic Q-learning only updates \(Q(x_k, u_k)\):
  \[
  Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]
  \]

- \(Q(\lambda)\)-learning updates all eligible pairs:
  \[
  Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]
  \]
  for all \(x, u\)

- Note: exploratory actions break causality
  \[ \Rightarrow \text{reset eligibility trace to 0} \]

**SARSA(\(\lambda\))**

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

**Complete Q(\(\lambda\))-learning algorithm**

- for every trial
  - \(e(x, u) \leftarrow 0\) 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 \(e(x, u) \leftarrow \lambda \cdot e(x, u)\) for all \(x, u\)
      - \(e(x, u_k) \leftarrow 1\)
      - \(Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]\)
    - for all \(x, u\)
  - until terminal state

**Complete SARSA(\(\lambda\)) algorithm**

- for every trial
  - \(e(x, u) \leftarrow 0\) 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 \(u_{k+1}\)
    - \(e(x, u) \leftarrow \lambda \cdot e(x, u)\) for all \(x, u\)
      - \(e(x, u_k) \leftarrow 1\)
      - \(Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]\)
    - for all \(x, u\)
  - until terminal state

**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
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 \( r_{k+1} \)
    \[
    Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \left[ r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k) \right]
    \]
    add \( (x_k, u_k, x_{k+1}, r_{k+1}) \) to database ReplayExperience
  until terminal state
end for

Reinforcement learning basics

**Algorithms**

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

**Summary and outlook**

- **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)

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 Bușoniu, Jelmer van Ast, Robert Babuška
Knowledge-Based Control Systems
2010-03-03

Introduction
Classical offline algorithms

Approximate algorithms for continuous spaces

Principle of RL

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

This presentation: approximate RL
– continuous states & actions

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 $h^* \text{ greedy in } Q^*$:
  $h^*(x) = \arg \max_u Q^*(x,u)$

Recall: Types of algorithms

By model knowledge
- Model-based – $f$ and $\rho$ known (dynamic programming)
- Model-free – no $f$ and $\rho$, only transition data (RL)
- Model-learning – estimate $f$ and $\rho$ from transition data

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

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

Algorithms considered

<table>
<thead>
<tr>
<th>Off-policy</th>
<th>On-policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>(previous lecture) Q-learning</td>
<td>SARSA</td>
</tr>
<tr>
<td>(this lecture)  Q-iteration</td>
<td>Policy iteration</td>
</tr>
<tr>
<td>Approximate offline fuzzy Q-iteration</td>
<td>approximate policy iteration</td>
</tr>
<tr>
<td>Approximate online approximate Q-learning</td>
<td>online approximate policy iteration</td>
</tr>
</tbody>
</table>

Technical focus: Q-iteration & fuzzy Q-iteration
**Introduction**

Classical offline algorithms

Approximate algorithms for continuous spaces

---

### Offline, off-policy: Q-iteration

- Turn Bellman optimality equation:
  \[ Q^*(x, u) = r(x, u) + \gamma \max_{u'} Q^*(f(x, u), u') \]

  into an iterative update:

  **Q-iteration**

  \[
  \begin{align*}
  \text{repeat at each iteration } & \ell \\
  \text{for all } & x, u \text{ do} \\
  Q_{\ell+1}(x, u) & \leftarrow r(x, u) + \gamma \max_{u'} Q_\ell(f(x, u), u') \\
  \text{end for} \\
  \text{until convergence to } & Q^* \\
  \end{align*}
  \]

- Once \( Q^* \) available: \( h^*(x) = \arg \max_u Q^*(x, u) \)

---

### Offline, on-policy: Policy iteration

- Recall on-policy: find \( Q^\pi \), improve \( \pi \), repeat

  **Policy iteration**

  starting from an initial policy

  \[
  \begin{align*}
  \text{repeat at each iteration } & \ell \\
  \text{policy evaluation: } & Q^\pi \\
  \text{policy improvement: } & h_{\ell+1}(x) \leftarrow \arg \max_u Q^\pi(x, u) \\
  \text{until convergence to } & h^\pi \\
  \end{align*}
  \]

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

---

### Approximate algorithms for continuous spaces

<table>
<thead>
<tr>
<th>Off-policy</th>
<th>On-policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical offline DP:</td>
<td>Q-iteration</td>
</tr>
<tr>
<td>Approximate offline:</td>
<td>fuzzy</td>
</tr>
<tr>
<td>Approximate online:</td>
<td>approximate Q-learning</td>
</tr>
</tbody>
</table>

---

### Discount robot: Q-iteration demo

Discount factor: \( \gamma = 0.5 \)

- Each update is a contraction with factor \( \gamma \):
  \[ \| Q_{\ell+1} - Q^* \|_\infty \leq \gamma \| Q_\ell - Q^* \|_\infty \]

  \( \Rightarrow \) Q-iteration monotonically converges to \( Q^* \)

---

### Cleaning robot: Policy iteration demo

**Initial policy:** always go left

**Policy evaluation:** find \( Q^\pi \)

**Policy improvement:**

\[
 Q_{\ell+1}(x, u) \leftarrow r(x, u) + \gamma \max_{u'} Q_\ell(f(x, u), u')
\]

---

### Approximate Q-function approximation

- In real-life control, \( X, U \) continuous
  \( \Rightarrow \) approximate Q-function \( \tilde{Q} \) must be used

- Usually, policy not approximated
  Greedy in \( \tilde{Q} \), computed on demand for given \( x \):
  \[
  h(x) = \arg \max_u \tilde{Q}(x, u)
  \]

- Approximator must ensure efficient \( \arg \max_u \) 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

Typically: basis functions

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

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

Approximate algorithms for continuous spaces

Action discretization

Typically: approximate algorithms for continuous spaces

Approximate Q-iteration

Store:

- $N \times M$ matrix of parameters $\theta$
- (one for each pair basis function–discrete action)

$$\hat{Q}^f(x, u_j) = \sum_{i=1}^{N} \phi_i(x) \theta_{ij}$$

Offline, off-policy: Fuzzy Q-iteration

Fuzzy Q-iteration policy

- Recall optimal policy:
  $$h^*(x) = \arg \max_u Q^*(x, u)$$
- Fuzzy Q-iteration policy:
  $$\hat{h}^*(x) = \arg \max_{u_j} Q^f(x, u_j)$$
  ($\hat{\theta}^*$ = converged parameter matrix)

Example: Inverted pendulum swing-up

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

- Goal: stabilize pointing up
- Insufficient actuation $\Rightarrow$ need to swing back & forth

Inverted pendulum: Near-optimal solution

Left: Q-function for $u = 0$

Right: policy

Fuzzy Q-iteration

Recall classical Q-iteration:

repeat at each iteration $t$
  for all $x$, $u$ do
    $$Q_{t+1}(x, u) = \rho(x, u) + \gamma \max_{u'} Q_{t}(f(x, u), u')$$
  end for
  until convergence

Fuzzy Q-iteration

repeat at each iteration $t$
  for all cores $x_i$, discrete actions $u_j$ do
    $$\hat{\theta}_{t+1, ij} = \rho(x, u_j) + \gamma \max_{u'_{t+1}} \hat{Q}^f(f(x, u_j), u_{t+1})$$
  end for
  until convergence
**Fuzzy Q-iteration convergence**

Like classical Q-iteration:
- Each update is a contraction with factor $\gamma$:
  $$\|\theta_{t+1} - \theta^*\|_\infty \leq \gamma \|\theta_t - \theta^*\|_\infty$$
- Monotonic convergence to $\theta^*$

$\theta^*$ leads to near-optimal $Q^*$, $h^*$

---

**Approximate policy iteration**

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

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

---

**Online approximate policy iteration**

Real-time learning control
- Approximator: similar to offline API (except $11 \times 11$ RBFs)
## Classical offline algorithms

- Q-iteration
- Policy iteration

## Approximate offline algorithms

- Fuzzy Q-iteration
- Online approximate policy iteration

## Approximate online algorithms

- Approximate Q-learning (RL)

### Off-policy

<table>
<thead>
<tr>
<th>Classical offline</th>
<th>Approximate offline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-iteration</td>
<td>Policy iteration</td>
</tr>
<tr>
<td>Fuzzy Q-iteration</td>
<td>Approximate policy iteration</td>
</tr>
</tbody>
</table>

### On-policy

<table>
<thead>
<tr>
<th>Classical offline</th>
<th>Approximate offline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-iteration</td>
<td>Policy iteration</td>
</tr>
</tbody>
</table>

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

Real-time learning control
Employs experience replay

### 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

### Conclusion

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

- Take-home message