Approximate dynamic programming and reinforcement learning for control

Lucian Buşoniu

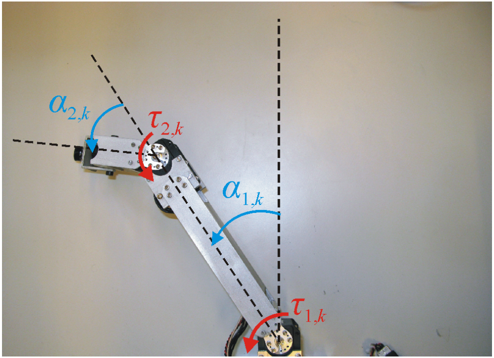
Universitat Politècnica de València, 21-23 June 2017

Part II

Continuous case

The need for approximation

- Classical algorithms – tabular representations, e.g. $Q(x, u)$ separately for all x, u values
- In control applications, x, u **continuous**! E.g. robot arm:



- **Tabular representation impossible**

The need for approximation (cont'd)

In real control applications,
the functions of interest must be **approximated**

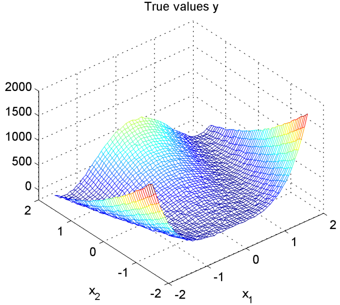
Part II in course structure

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

Approximation

Function approximation:
function with an infinite number of values
$\rightarrow$ represent using a small number of values

$f(x)$



$\widehat{f}(x)$

?

Parametric approximation

Parametric approximation: fixed form $\widehat{f}(x)$,
value determined by a **parameter vector** θ:

$$\widehat{f}(x;\theta)$$

1. Linear approximation: weighted sum of **basis functions** ϕ, with parameters as weights:

$$\widehat{f}(x;\theta) = \phi_1(x)\theta_1 + \phi_2(x)\theta_2 + \dots \phi_n(x)\theta_n$$
$$= \sum_{i=1}^{n} \phi_i(x)\theta_i = \phi^\top(x)\theta$$

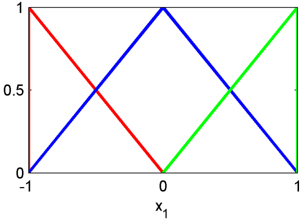
   Note: linear in the parameters, may be nonlinear in x!

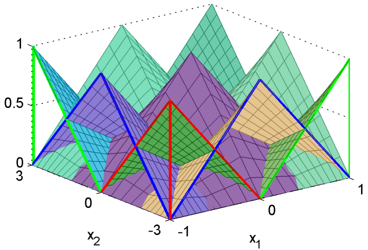
2. Nonlinear approximation: remains in the general form

Linear parametric approximation: Interpolation

Interpolation:

- D-dimensional grid of center points
- Multilinear interpolation between these points
- Equivalent to **pyramidal** basis functions





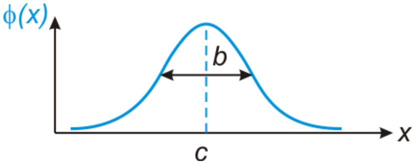
Linear parametric approximation: RBFs

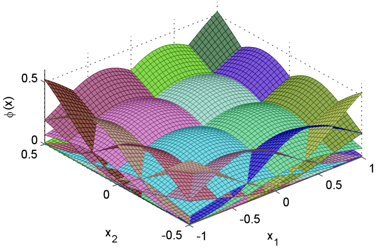
Radial basis functions (Gaussian):

$$\phi(x) = \exp\left[-\frac{(x-c)^2}{b^2}\right] \qquad \text{(1-dim)}$$

$$\phi(x) = \exp\left[-\sum_{d=1}^{D}\frac{(x_d-c_d)^2}{b_d^2}\right] \qquad \text{(D-dim)}$$

Possibly normalized: $\tilde{\phi}_i(x) = \frac{\phi_i(x)}{\sum_{i' \neq i} \phi_{i'}(x)}$





Training linear approximators: Least-squares

- n_s samples $(x_j, f(x_j))$, objective described by the system of equations:

$$\begin{aligned}
\widehat{f}(x_1;\theta) &= \phi_1(x_1)\theta_1 + \phi_2(x_1)\theta_2 + \dots \phi_n(x_1)\theta_n &&= f(x_1) \\
&\dots \\
\widehat{f}(x_{n_s};\theta) &= \phi_1(x_{n_s})\theta_1 + \phi_2(x_{n_s})\theta_2 + \dots \phi_n(x_{n_s})\theta_n &&= f(x_{n_s})
\end{aligned}$$

- Matrix form:

$$\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \dots & \dots & \dots & \dots \\ \phi_1(x_{n_s}) & \phi_2(x_1) & \dots & \phi_n(x_{n_s}) \end{bmatrix} \cdot \theta = \begin{bmatrix} f(x_1) \\ \dots \\ f(x_{n_s}) \end{bmatrix} \qquad A\theta = b$$

- **Linear regression**

Least-squares

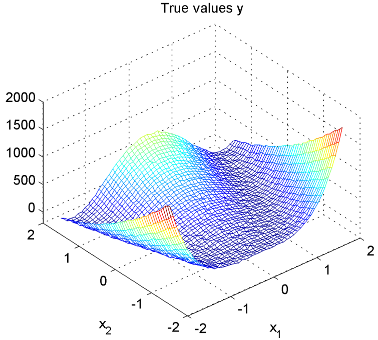
- System is overdetermined, ($n_s > n$), equations will not (all) hold with equality
  $\Rightarrow$ Solve in the least-squares sense:

$$\min_{\theta} \sum_{j=1}^{n_s} \left| f(x_j) - \widehat{f}(x_j; \theta) \right|^2$$

...linear algebra and calculus...

- $\theta = (A^\top A)^{-1} A^\top b$

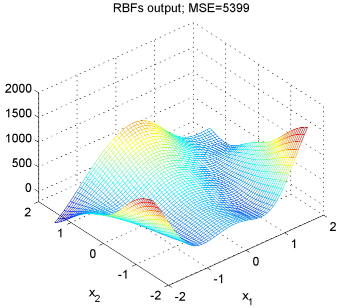
Example: Rosenbrock "banana" function



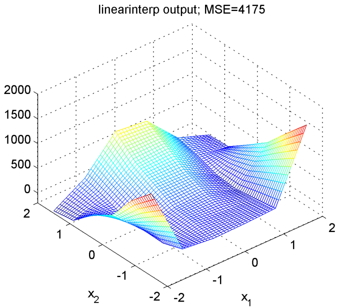
- $f(x) = (1 - x_1)^2 + 100[(x_2 + 1.5) - x_1^2]^2, \qquad x = [x_1, x_2]^\top$
- Training: 200 randomly distributed points
- Validation: grid of 31×31 points

Rosenbrock function: Linear approximator results

6x6 RBFs:



Interpolation on 6x6 grid:

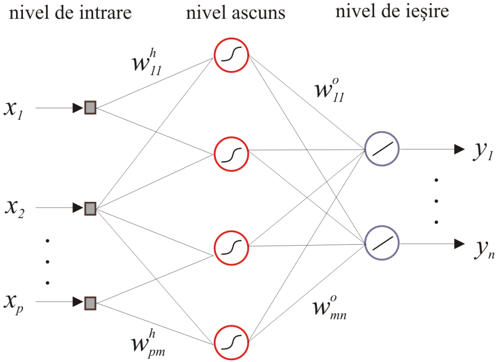


- RBF approximation smoother (wide RBFs)
- Interpolation = collection of multilinear surfaces

Nonlinear parametric approximation: Neural networks

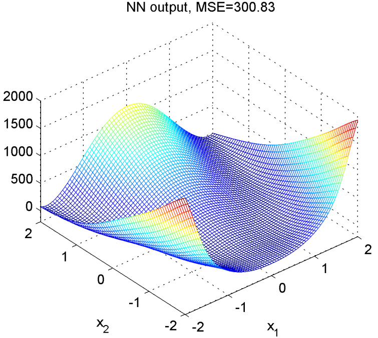
Neural network:

- **Neurons** with (non)linear activation functions
- **Interconnected** by weighted links
- On multiple layers



Rosenbrock function: Neural network result

One hidden layer with 10 neurons and tangent-sigmoidal activation functions; linear output layer.
500 training epochs.



Due to better flexibility of the neural network, results are better than with linear approximators.

Nonparametric approximation

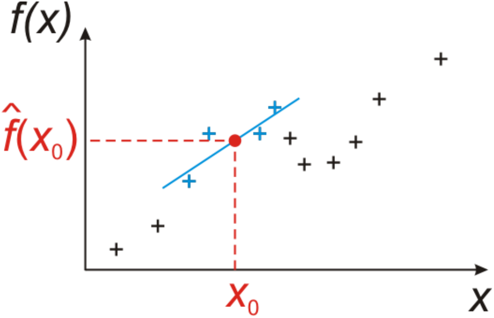
Recall parametric approximation:
fixed shape, fixed number of parameters

Nonparametric approximation:
shape, number of parameters **depend on the data**

Nonparametric approximation: LLR

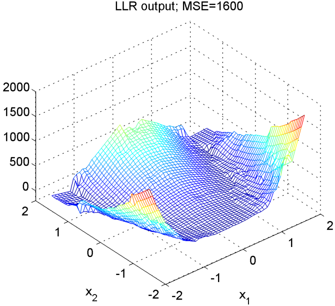
Local linear regression, LLR:

- Database of points $(x, f(x))$ (e.g. the training data)
- For given x_0, finds the ***k* nearest neighbors**
- Result computed with **linear regression** (LS) on these neighbors



Rosenbrock function: LLR result

Database = the 200 training points; $k = 5$
Validation: same grid of 31×31 points



- Performance in-between linear approximator and neural network

Comparison of approximators

In combination with DP and RL

- linear easier to analyze than nonlinear
- parametric easier to analyze than nonparametric

Flexibility

- nonlinear more flexible than linear
- nonparametric more flexible than parametric,
  shape of parametric approx. must be tuned manually
- nonparametric adapt to data:
  complexity as the number of data grows must be controlled

Approximation in DP and RL

Problems to address:

1. Representation: $Q(x, u)$, possibly $h(x)$
   Using the approximation methods discussed
2. Maximization: how to solve $\max_u Q(x, u)$

Solution 1 for maximization: Implicit policy

- Policy never represented explicitly
- Greedy actions computed on-demand from $\widehat{Q}$:

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

- Approximator must ensure **efficient solution for** $\arg\max$
- Problem then boils down to **approximating the Q-function**

Solution 2 for maximization: Explicit policy

- Policy explicitly approximated, $\widehat{h}(x)$

Advantages:

- **Continuous actions** easier to use
- Easier to incorporate **a priori knowledge** in the policy representation

Action discretization

- For now, we use solution 1 (implicit h)
- Approximator must ensure efficient solution for $\arg\max$

$\Rightarrow$ Typically: **action discretization**

- Choose M discrete actions $u_1, \dots, u_M \in U$
  compute "$\arg\max$" by direct enumeration

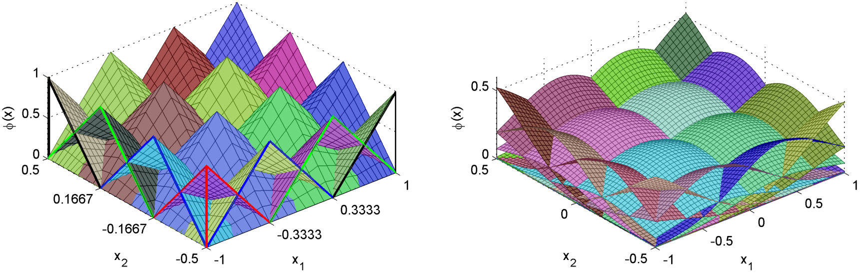
- Example: discretization on a grid

State-space approximation

- Typically: basis functions

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

- E.g. pyramidal, RBFs



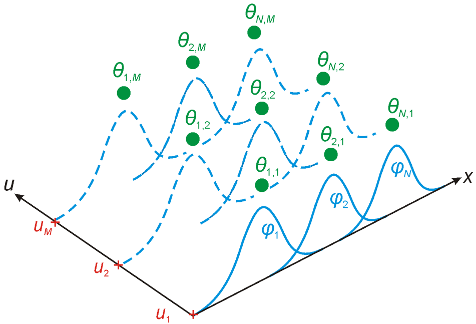
Discrete-action Q-function approximator

Given:

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

Store:

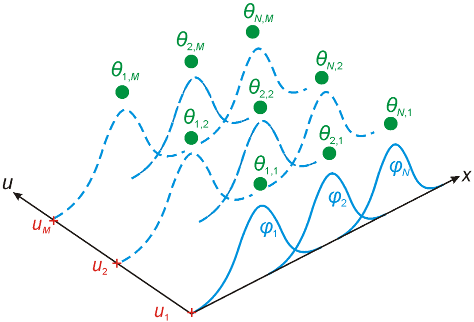
3. $N \cdot M$ parameters θ
   (one for each basis function – discrete action pair)



Discrete-action Q-function approximator (cont'd)

Approximate Q-function:

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



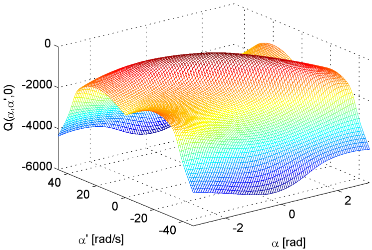
Example: Inverted pendulum

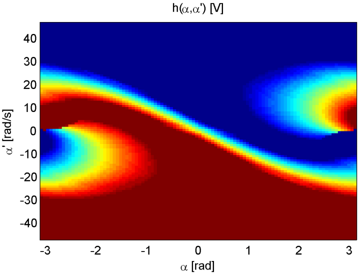
- $x = [\text{angle } \alpha, \ \text{velocity } \dot{\alpha}]^\top$
- u = voltage
- $\rho(x, u) = -x^\top \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix} x - u^\top 1 u$
- Discount factor $\gamma = 0.98$

- Objective: stabilize pointing up
- Insufficient torque $\Rightarrow$ swing-up required

Inverted pendulum: Optimal solution

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





Replay

Additional questions raised by approximation

1. **Convergence**: does the algorithm remain convergent?
2. **Solution quality**: is the solution found at a controlled distance from the optimum?
3. **Consistency**: for an ideal, infinite-precision approximator, would the optimal solution be recovered?

Algorithm landscape

By model usage:

- Model-based: f, ρ known
- Model-free: f, ρ unknown (reinforcement learning)

By interaction level:

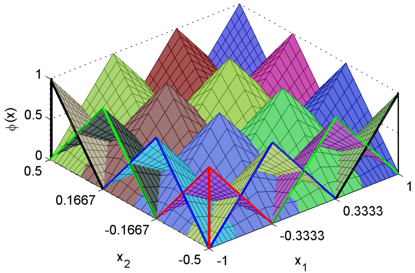
- Offline: algorithm runs in advance
- Online: algorithm runs with the system

Exact vs. approximate:

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

Interpolation-based approximator ("fuzzy")

- Interpolation = pyramidal BFs =
  = cross-product of triangular MFs



- Each BF i has center x_i
- $\theta_{i,j}$ has meaning of Q-value for the pair (x_i, u_j), since: $\phi_i(x_i) = 1, \phi_{i'}(x_i) = 0$ for $i' \neq i$

Interpolated Q-iteration (fuzzy Q-iteration)

Recall classical 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

Fuzzy Q-iteration

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

Policy

- Recall optimal policy:

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

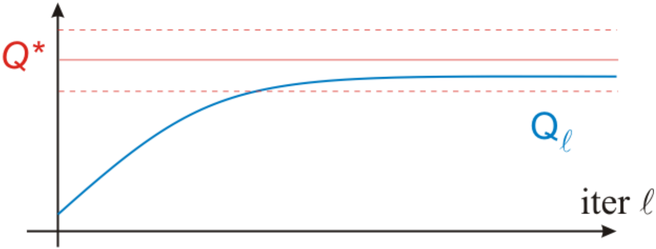
- In fuzzy Q-iteration:

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

θ^* = parameters at convergence

Convergence

Monotonic convergence to a **near-optimal** solution



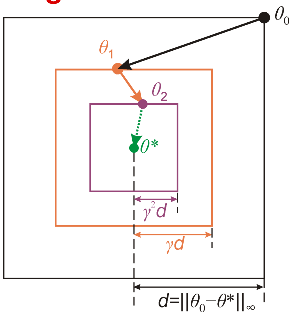
Convergence proof line

Similarly to classical Q-iteration:

- Each iteration is a contraction with factor γ:

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

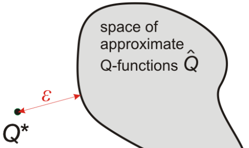
$\Rightarrow$ **Monotonic convergence** to θ^*



Solution quality

Approximator characterized by minimum distance to Q^*:

$$\varepsilon = \min_{\theta} \left\| Q^*(x,u) - \widehat{Q}(x,u;\theta) \right\|_\infty$$



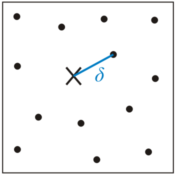
1. Sub-optimality of Q-function $\widehat{Q}(x,u;\theta^*)$ bounded:

$$\left\| Q^*(x,u) - \widehat{Q}(x,u;\theta^*) \right\|_\infty \le \frac{2\varepsilon}{1-\gamma}$$

2. Sub-optimality of resulting policy $\widehat{h}^*$ bounded by $\frac{4\varepsilon}{(1-\gamma)^2}$

Consistency

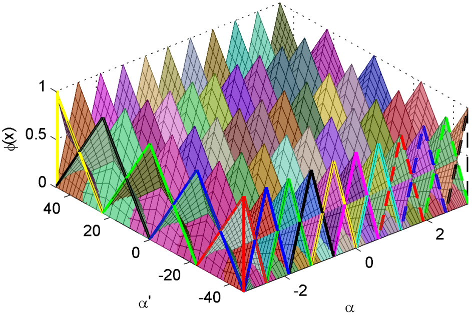
- Consistency: $\widehat{Q}^{\theta^*} \to Q^*$ as precision increases
- Precision: $\begin{cases} \delta_x = \max_x \min_i \|x - x_i\|_2 \\ \delta_u = \max_u \min_j \|u - u_j\|_2 \end{cases}$

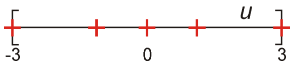


- Under appropriate technical conditions,
  $\Rightarrow \lim_{\delta_x \to 0, \delta_u \to 0} \widehat{Q}^{\theta^*} = Q^*$ — consistency

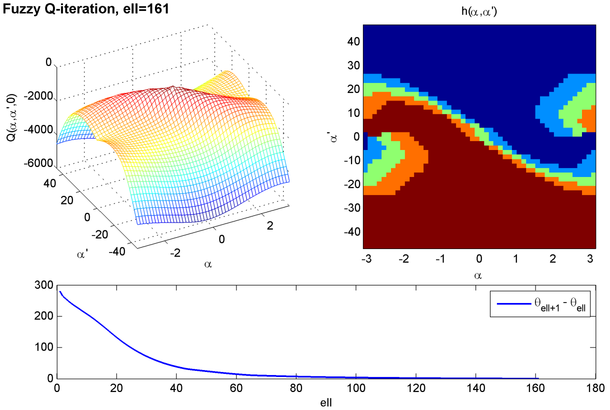
Inverted pendulum: Fuzzy Q-iteration

BFs: equidistant grid 41 $\times$ 21
Discretization: 5 actions, distributed around 0





Inverted pendulum: Fuzzy Q-iteration demo



Algorithm landscape

By model usage:

- Model-based: f, ρ known
- Model-free: f, ρ 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)

Note: All remaining algorithms in this part work directly in stochastic problems (although we introduce them in the deterministic case)

Fitted Q-iteration

Start from fuzzy Q-iteration and extend it to:

- other approximators than fuzzy/interpolation
- **model-free** context – RL

Note: For offline RL methods, exploration boils down to having a "sufficiently informative" set of transitions

Intermediate model-based algorithm

Recall fuzzy Q-iteration:

for all x_i, u_j $\quad \theta_{\ell+1,i,j} \leftarrow \rho(x_i, u_j) + \gamma \max_{j'} \widehat{Q}(f(x_i, u_j), u_{j'}; \theta_\ell)$ **end for**

1. Use arbitrary state-action samples
2. Extend to generic approximation
3. Find parameters using least-squares

given (x_s, u_s), $s = 1, \dots, n_s$
repeat at each iteration ℓ
  for $s = 1, \dots, n_s$ **do**
    $q_s \leftarrow \rho(x_s, u_s) + \gamma \max_{u'} \widehat{Q}(f(x_s, u_s), u'; \theta_\ell)$
  end for
  $\theta_{\ell+1} \leftarrow \arg\min \sum_{s=1}^{n_s} \left| q_s - \widehat{Q}(x_s, u_s; \theta) \right|^2$
until finished

Note: Fuzzy Q-iteration equivalent to generalized algo if interpolation is used and the samples are all the combinations x_i, u_j

Fitted Q-iteration: Final algorithm

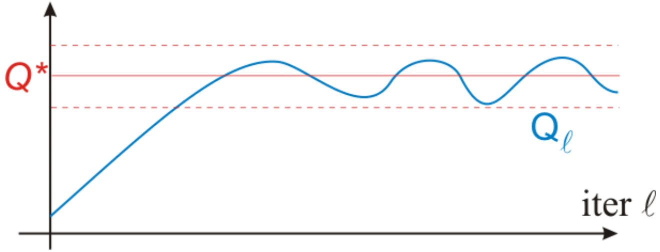
4. Use transitions instead of model

Fitted Q-iteration

given (x_s, u_s, r_s, x'_s), $s = 1, \dots, n_\mathrm{s}$
repeat at each iteration ℓ
  for $s = 1, \dots, n_\mathrm{s}$ **do**
    $q_s \leftarrow r_s + \gamma \max_{u'} \widehat{Q}(x'_s, u'; \theta_\ell)$
  end for
  $\theta_{\ell+1} \leftarrow \arg\min \sum_{s=1}^{n_\mathrm{s}} \left| q_s - \widehat{Q}(x_s, u_s; \theta) \right|^2$
until finished

Fitted Q-iteration: Convergence

Convergence to a **sequence** of solutions,
all of them **near-optimal**



1. Introduction
2. Approximation
3. Model-based approximate dynamic programming
4. Model-free approximate dynamic programming
   - Fitted Q-iteration
   - Least-squares policy iteration
5. Approximate temporal difference methods
6. Policy gradient

Approximate policy iteration

Recall: classical policy iteration

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

repeat at each iteration ℓ
approximate policy evaluation: find $\widehat{Q}^{h_\ell}$
policy improvement: $h_{\ell+1}(x) \leftarrow \arg\max_u \widehat{Q}^{h_\ell}(x, u)$
until finished

Policy still **implicitly represented** (solution 1)

Approximate policy evaluation

Main problem: Approximate policy evaluation:

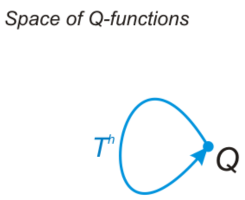
find $\widehat{Q}^{h_\ell}$

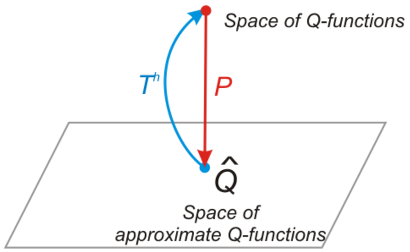
Projected Bellman equation

- Recall: Bellman equation for Q^h, discrete case:

$$Q^h(x,u) = \rho(x,u) + \gamma Q^h(f(x,u), h(f(x,u)))$$
$$Q^h = T^h(Q^h) \text{ (Bellman mapping)}$$

- Approximation: $\widehat{Q} = PT^h(\widehat{Q})$





Solution (sketch)

- Projected Bellman equation:

$$\widehat{Q} = PT^h(\widehat{Q}), \qquad \widehat{Q}(x, u; \theta) = \phi^\top(x, u)\theta$$

- Matrix form:

$$A\theta = \gamma B\theta + b, \qquad A, B \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$$

(equivalent to $(A - \gamma B)\theta = b$)

- Estimate from data (x_s, u_s, r_s, x'_s):

$$\begin{aligned} A &\leftarrow A + \phi(x_s, u_s)\phi^\top(x_s, u_s) \\ B &\leftarrow B + \phi(x_s, u_s)\phi^\top(x'_s, h(x'_s)) \\ b &\leftarrow b + \phi(x_s, u_s)r_s \end{aligned}$$

Least-squares policy iteration

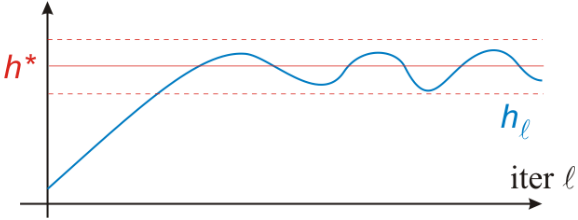
Evaluates h using projected Bellman equation

Least-squares policy iteration (LSPI)

date fiind (x_s, u_s, r_s, x'_s), $s = 1, \dots, n_\mathrm{s}$
repeat at each iteration
$A \leftarrow 0,\ B \leftarrow 0,\ b \leftarrow 0$
for $s = 1, \dots, n_\mathrm{s}$ **do**
$A \leftarrow A + \phi(x_s, u_s)\phi^\top(x_s, u_s)$
$B \leftarrow B + \phi(x_s, u_s)\phi^\top(x'_s, h(x'_s))$
$b \leftarrow b + \phi(x_s, u_s) r_s$
end for
solve $A\theta = \gamma B\theta + b$ to find θ
implicit policy improvement: $h(x) \leftarrow \arg\max_u \widehat{Q}(x, u; \theta)$
until finished

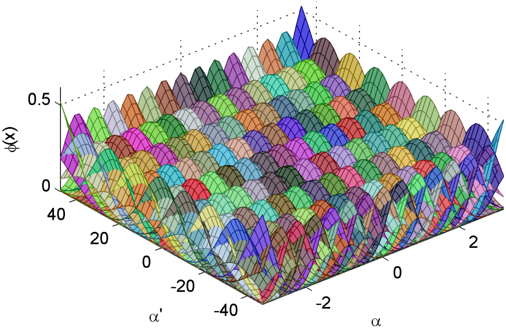
LSPI: Convergence

Under appropriate conditions, LSPI converges to
a **sequence** of policies,
all within a bounded distance from h^*

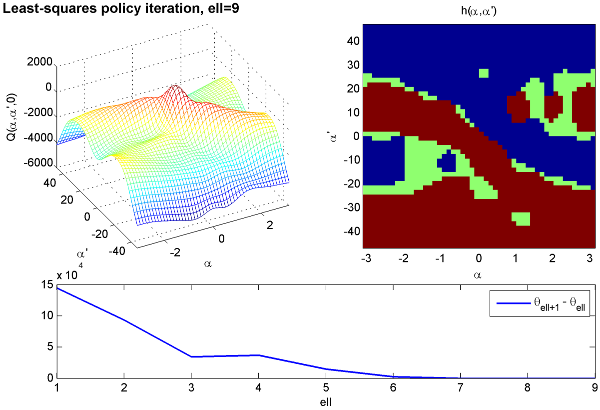


Inverted pendulum: LSPI

Basis functions: 15×9 grid of RBFs
Discretization: 3 equidistant actions
Data: 7500 transitions from uniformly random (x, u)



Inverted pendulum: LSPI demo



AVI vs. API comparison

Number of iterations to convergence

- Usually, approximate value iteration $>$
  approximate policy iteration

Complexity

- Depends on the particular algorithms
- E.g. one fuzzy Q iteration $<$ one LSPI iteration

Convergence

- approximate value and policy iteration
  both converge to a sequence of solutions,
  each of them near-optimal
- in interesting cases (e.g. interpolation), approximate value iteration converges to a unique solution

Algorithm landscape

By model usage:

- Model-based: f, ρ known
- Model-free: f, ρ 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)

Recall: Classical Q-learning

Q-learning with ε-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

Temporal difference: $[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$

Approximate Q-learning

- Q-learning **decreases the temporal difference**:

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

- $r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u')$ replaces **ideal** target $Q^*(x_k, u_k)$

  [See Bellman: $Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(x', u')$]

$\Rightarrow$ Ideally, decrease error $[Q^*(x_k, u_k) - Q(x_k, u_k)]$

Approximate Q-learning (cont'd)

Approximation: use $\widehat{Q}(x, u; \theta)$, update **parameters**

- **Gradient descent** on the error $[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta)]$:

$$\begin{aligned}\theta_{k+1} &= \theta_k - \frac{1}{2}\alpha_k \frac{\partial}{\partial\theta}\left[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k)\right]^2 \\ &= \theta_k + \alpha_k \frac{\partial}{\partial\theta}\widehat{Q}(x_k, u_k; \theta_k) \cdot \left[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k)\right]\end{aligned}$$

- Use available **estimate** of $Q^*(x_k, u_k)$:

$$\begin{aligned}\theta_{k+1} = \theta_k + \alpha_k \frac{\partial}{\partial\theta}\widehat{Q}(x_k, u_k; \theta_k) \cdot \\ \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}(x_{k+1}, u'; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k)\right]\end{aligned}$$

(approximate temporal difference)

Approximate Q-learning: Algorithm

Approximate Q-learning with ε-greedy exploration

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

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

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

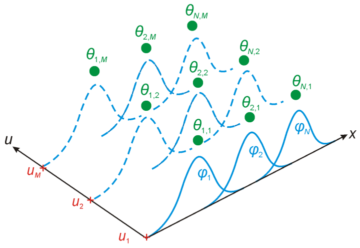
$$\theta_{k+1} = \theta_k + \alpha_k \frac{\partial}{\partial \theta} \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}(x_{k+1}, u'; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

  until trial finished
end for

Of course, **exploration** needed also in approximate case

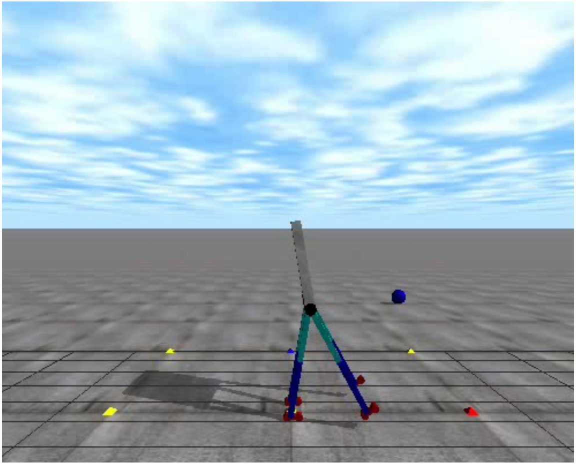
Maximization in approximate Q-learning

- Greedy actions computed on-demand, greedy policy represented implicitly (type 1)
- Approximator must ensure efficient max solution
- E.g. discrete actions & basis functions in x



Approx. Q-learning: robot walking demo (E. Schuitema)

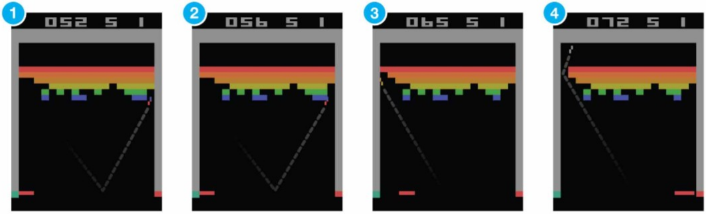
Approximator: tile coding



Approximate Q-learning with deep neural networks

- Q-function represented by neural networks $\widehat{Q}(x_{k+1}, \cdot; \theta_k)$
- Deep neural networks, i.e. many layers
  with specific structures and activation functions
- Network trained to minimize temporal difference,
  like standard approximate Q-learning
- Training on mini-batches of samples, so in fact algorithm is
  in-between fitted Q-iteration and Q-learning

(DeepMind, *Human-level control through deep reinforcement learning*, Nature 2015)



Approximate SARSA

Recall classical SARSA:

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

Approximation: similar to Q-learning

- update parameters
- based on the gradient of the Q-function
- and the approximate temporal difference

$$\theta_{k+1} = \theta_k + \alpha_k \frac{\partial}{\partial \theta} \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \widehat{Q}(x_{k+1}, u_{k+1}; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

Approximate SARSA: Algorithm

Approximate SARSA

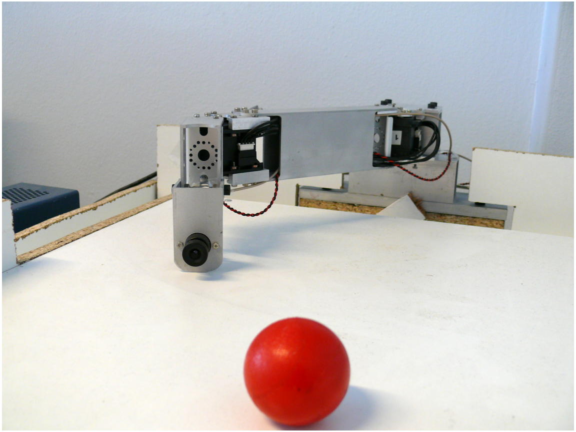
for each trial **do**
  init x_0
  choose u_0 (e.g. ε-greedy in $Q(x_0, \cdot; \theta_0)$)
  repeat at each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose u_{k+1} (e.g. ε-greedy in $Q(x_{k+1}, \cdot; \theta_k)$)

$$\theta_{k+1} = \theta_k + \alpha_k \frac{\partial}{\partial \theta} \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \widehat{Q}(x_{k+1}, u_{k+1}; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

  until trial finished
end for

Goalkeeper robot: SARSA demo (S. Adam)

Learn how to catch ball, using video camera image
Employs experience replay



Algorithm landscape

By model usage:

- Model-based: f, ρ known
- Model-free: f, ρ 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)

Same classification as approximate TD

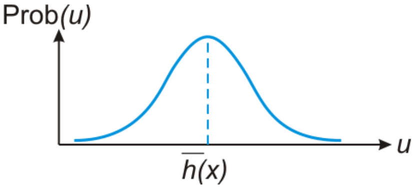
Policy representation

- Type 2: Policy **explicitly approximated**
- Recall advantages: easier to handle continuous actions, prior knowledge
- For example, BF representation:

$$\bar{h}(x; \vartheta) = \sum_{i=1}^{n} \phi_i(x)\vartheta_i$$

Policy with exploration

- Online RL $\Rightarrow$ policy gradient must explore

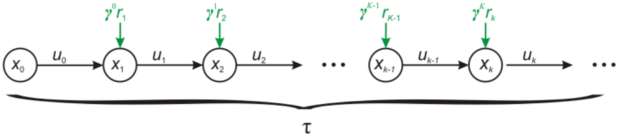


- Zero-mean Gaussian exploration:

$$P(u|x) = \mathcal{N}(\overline{h}(x;\vartheta), \Sigma) =: \widehat{h}(x, u; \theta)$$

  with θ containing ϑ as well as the covariances in Σ
- So policy in fact represented as probabilities, including random exploration

Trajectory



- Trajectory $\tau := (x_0, u_0, \ldots, x_k, u_k, \ldots)$ generated with $\widehat{h}$; and resulting rewards $r_1, \ldots, r_{k-1}, \ldots$
- Return along the trajectory:

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

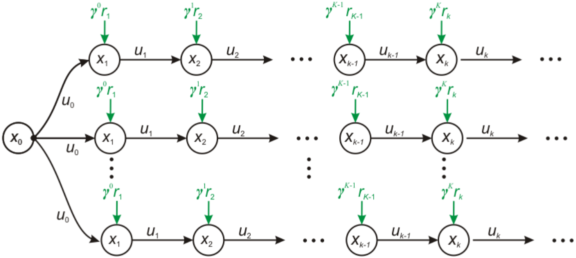
- Probability of the trajectory under policy parameters θ:

$$P_\theta(\tau) = \prod_{k=0}^{\infty} \widehat{h}(x_k, u_k; \theta)$$

where $x_{k+1} = f(x_k, u_k)$

Performance objective

Take x_0 fixed, for simplicity



Objective

Maximize expected return from x_0 of policy $\widehat{h}(\cdot, \cdot; \theta)$, given by parameter θ:

$$R(x_0) = \mathrm{E}_\theta \left\{ R(\tau) \right\} = \int R(\tau) P_\theta(\tau) d\tau =: J_\theta$$

Main idea

Gradient ascent on $J(\theta)$:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J_\theta$$

Gradient derivation

$$
\begin{aligned}
\nabla_\theta J_\theta &= \int R(\tau)\nabla_\theta P_\theta(\tau)d\tau \\
&= \int R(\tau)P_\theta(\tau)\nabla_\theta \log P_\theta(\tau)d\tau \\
&= \mathrm{E}_\theta\left\{R(\tau)\nabla_\theta \log\left[\prod_{k=0}^{\infty}\widehat{h}(x_k, u_k; \theta)\right]\right\} \\
&= \mathrm{E}_\theta\left\{R(\tau)\sum_{k=0}^{\infty}\nabla_\theta \log\widehat{h}(x_k, u_k; \theta)\right\}
\end{aligned}
$$

Where we:

- used "likelihood ratio trick" $\nabla_\theta P_\theta(\tau) = P_\theta(\tau)\nabla_\theta \log P_\theta(\tau)$
- replaced integral by expectation, and substituted $P_\theta(\tau)$
- replaced log of product by sum of logs

Gradient implementation

- Many methods exist to estimate gradient, based on Monte-Carlo
- E.g. REINFORCE uses current policy to execute n_s sample trajectories, each of finite length K, and estimates:

$$\widehat{\nabla}_\theta J_\theta = \frac{1}{n_s} \sum_{j=1}^{n_s} \left[\sum_{k=0}^{K-1} \gamma^k r_{s,k} \right] \left[\sum_{k=0}^{K-1} \nabla_\theta \log \widehat{h}(x_{s,k}, u_{s,k}; \theta) \right]$$

(with possible addition of a baseline to reduce variance)

- Compare with exact formula:

$$\nabla_\theta J_\theta = \mathrm{E}_\theta \left\{ R(\tau) \sum_{k=0}^{\infty} \nabla_\theta \log \widehat{h}(x_k, u_k; \theta) \right\}$$

- Gradient $\nabla_\theta \log \widehat{h}$ preferably computable in closed-form

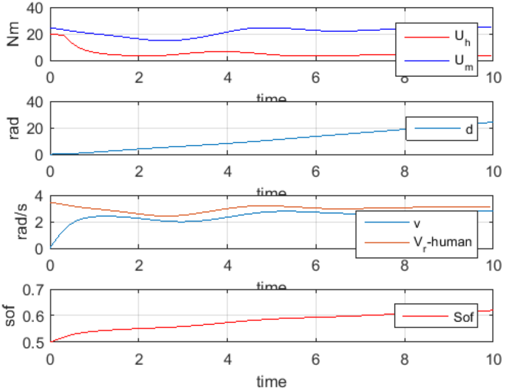
Power-assisted wheelchair (Autonomad, G. Feng)

- Hybrid power source: human and battery
- Objective: drive a given distance, optimizing assistance to:
  - (i) attain desired user fatigue level at task completion
  - (ii) minimize battery usage
- Challenge: user has **unknown torque dynamics**, based on fatigue, motivation, velocity etc.

PAW: Policy gradient

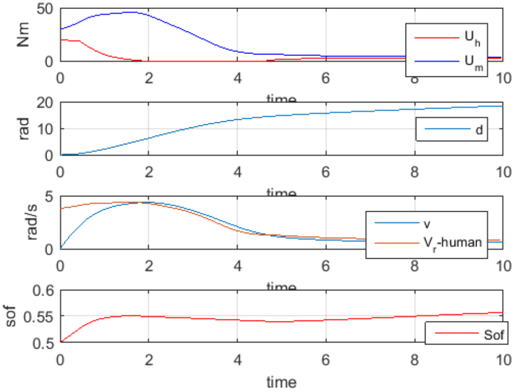
- Policy parameterized using RBFs
- Literature model for user, **unknown to the algorithm**
- Rewards on distance, fatigue, and electrical power components

PAW: Early results



Target distance inaccurately reached

PAW: Final learning results



Large assistance at start, to motivate user;
tapering down so desired location and fatigue reached

References for Part II

- Bertsekas & Tsitsiklis, *Neuro-Dynamic Programming*, 1996.
- 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.
- Deisenroth, Neumann, & Peters, *A Survey of Policy Search for Robotics*, Foundations and Trends in Robotics 2, 2011.