Reinforcement learning

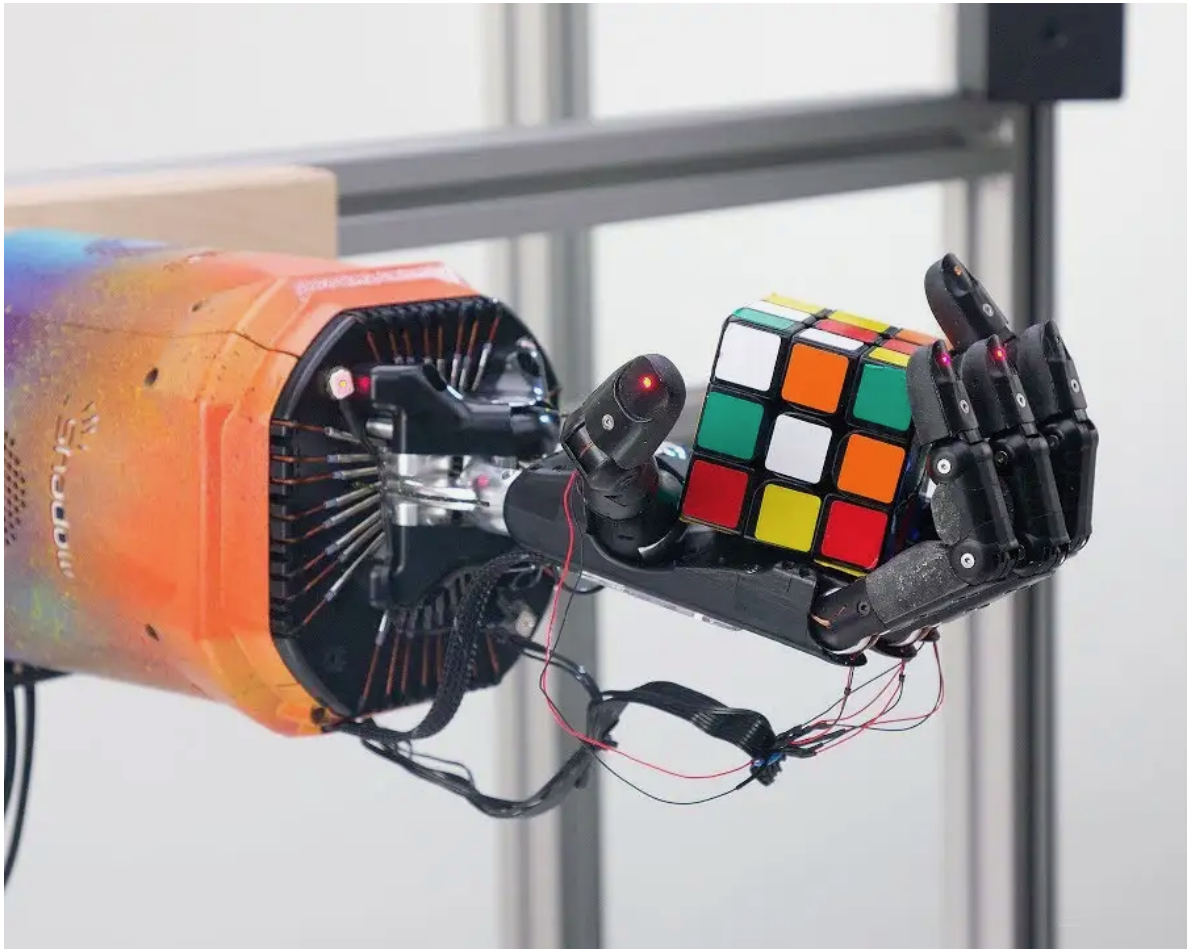
Master CPS, Year 2 Semester 1

Lucian Buşoniu, Florin Gogianu

Part IV

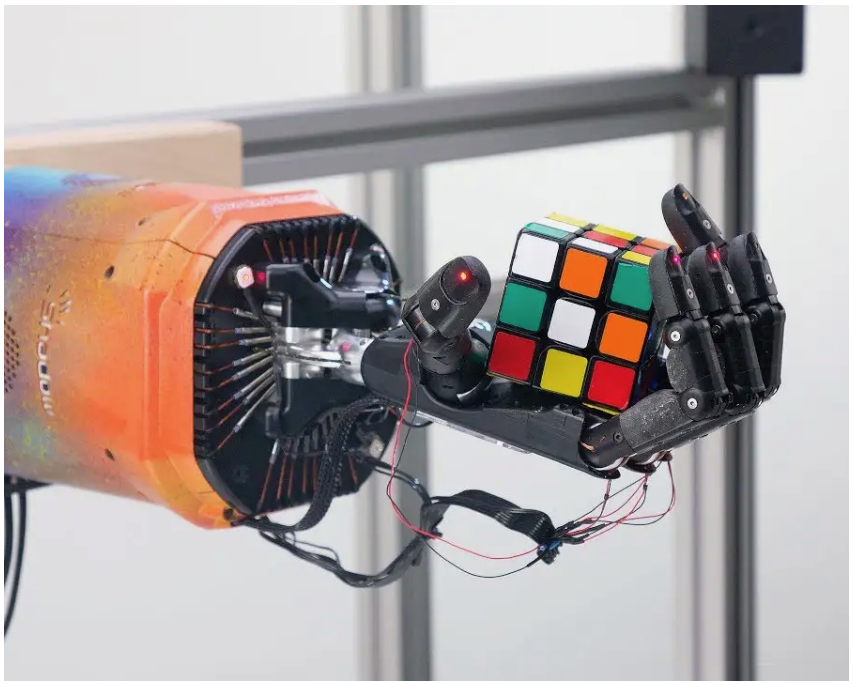
Approximate dynamic programming and offline approximate reinforcement learning

RL for manipulation of a Rubik's Cube (OpenAI)



The need for approximation

- Classical RL – representation in tabular form, e.g., $Q(x, u)$ separately for all values of x and u
- In real applications, x, u often **continuous** (or discrete with very many values)!



- **Tabular representation is impossible**

The need for approximation (continued)

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

Part IV in plan

- Reinforcement learning problem
- Optimal solution
- Exact dynamic programming
- Exact reinforcement learning
- **Approximation techniques**
- Deep neural networks
- Approximate dynamic programming
- Classical approximate reinforcement learning
- Deep reinforcement learning

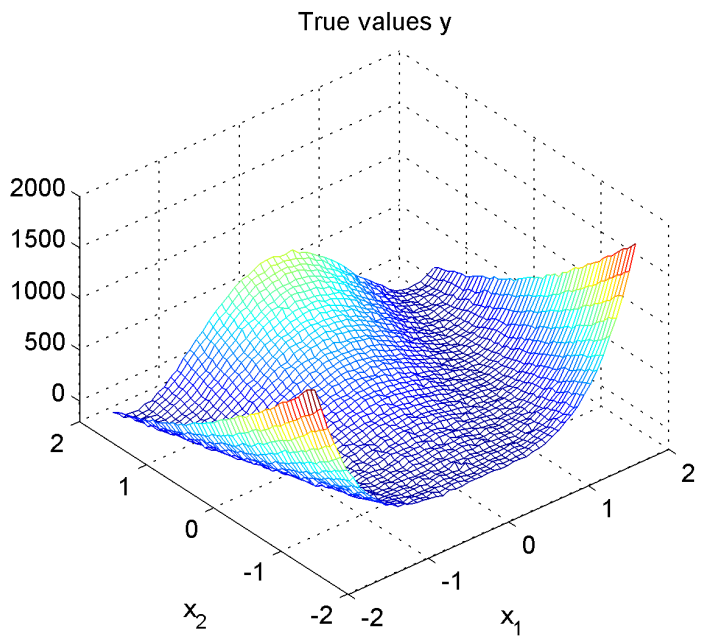
Contents of part IV

1. Approximation techniques
2. Approximation in DP&RL
3. Q-iteration with interpolation
4. Fitted Q-iteration

Approximation

Approximation:
a function with (uncountably) infinitely many values
must be represented using a small number of values

$f(x)$



$\widehat{f}(x)$

?

Parametric approximation

Parametric approximation: $\widehat{f}$ has a fixed form,
its output is determined by a vector of **parameters** θ:

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

1. Linear approximation – weighted combination of **features** (basis functions) ϕ_i:

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

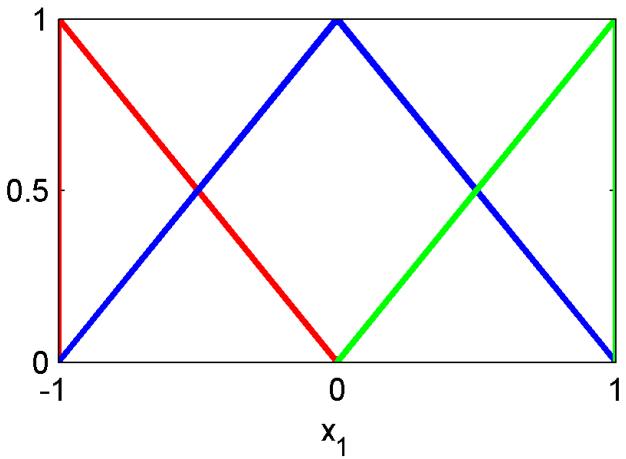
   Note: linear in parameters, can be nonlinear in x

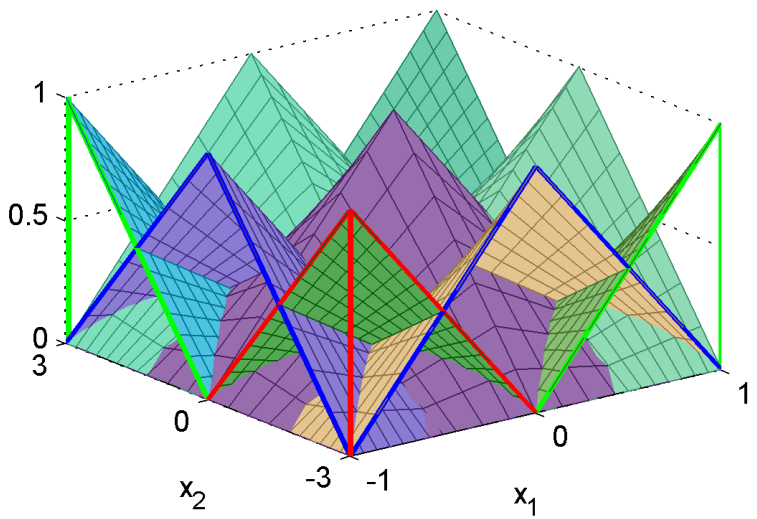
2. Nonlinear approximation: stays in the general form

Linear parametric approximation: Interpolation

Interpolation:

- D-dimensional grid of points
- Multilinear interpolation between points
- Equivalent to **pyramidal** features





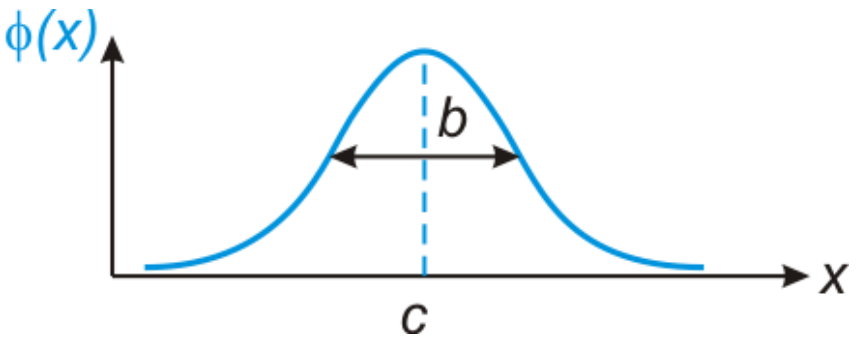
Linear parametric approximation: RBF

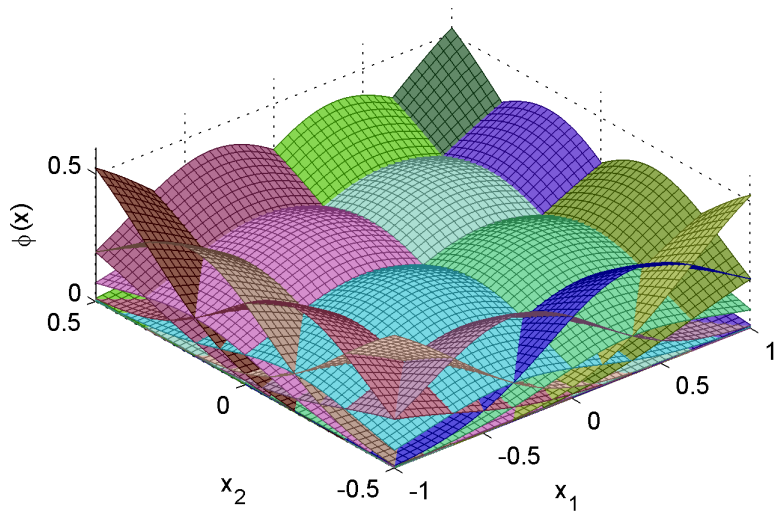
Radial basis function (Gaussian):

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

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

Optionally, normalization: $\tilde{\phi}_i(x) = \frac{\phi_i(x)}{\sum_{i'}\phi_{i'}(x)}$





Training linear approximators: least squares

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

$$\widehat{f}(x_1;\theta) = \phi_1(x_1)\theta_1 + \phi_2(x_1)\theta_2 + \dots \phi_n(x_1)\theta_n = f(x_1)$$

$$\dots$$

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

- Matrix form:

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

Least squares (continued)

- Overdetermined system ($n_s > n$), equations cannot all be satisfied with equality.
- Define a least-squares **loss function**:

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

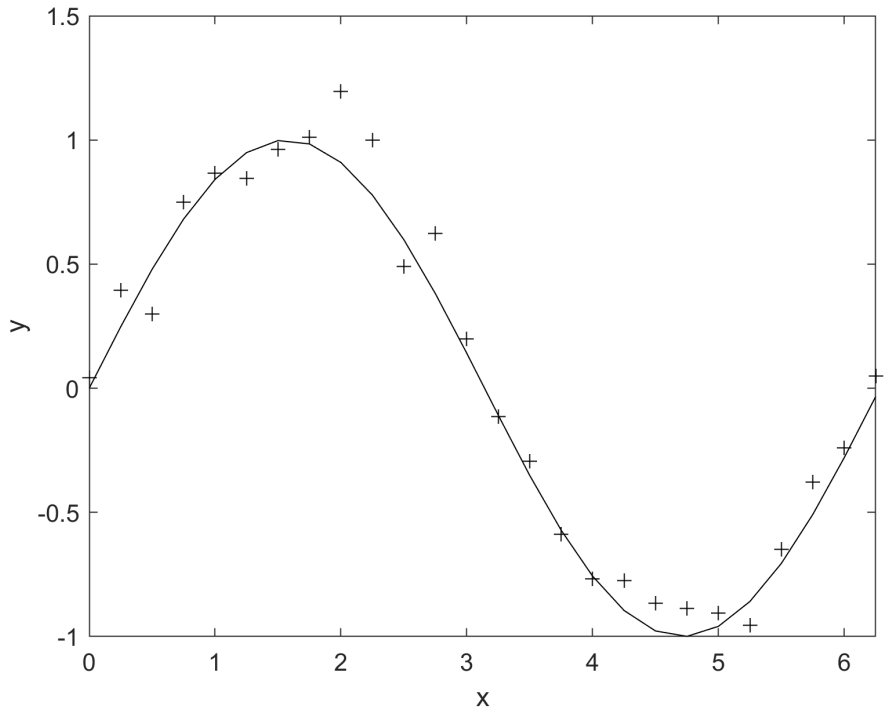
- Solve in the least-squares sense:

$$\min_{\theta} \mathcal{L}(\theta)$$

...linear algebra and analysis...

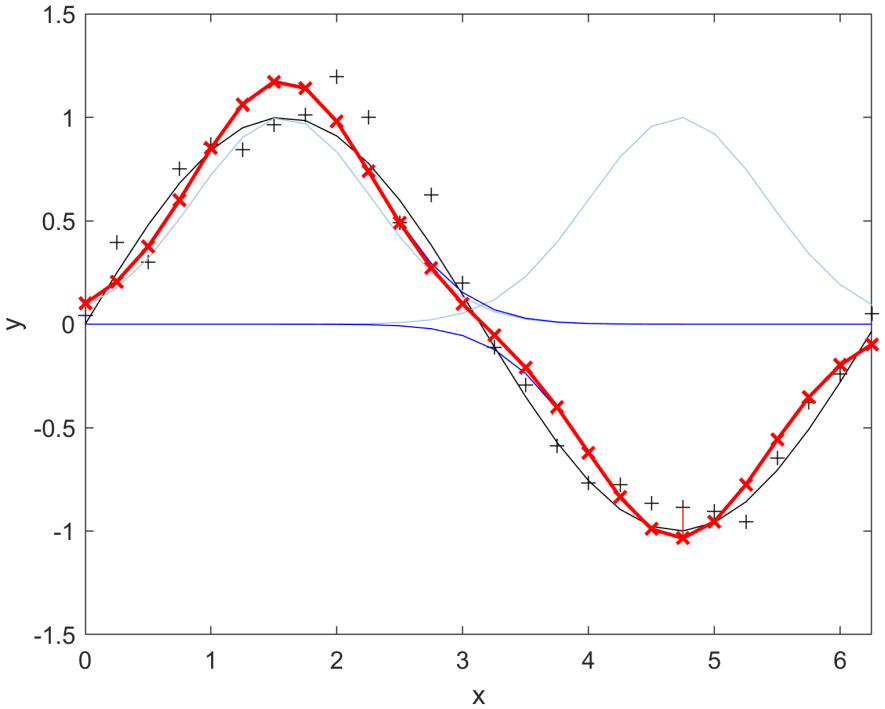
- $\theta = (A^\top A)^{-1} A^\top b \qquad (\Leftarrow (A^\top A)\theta = A^\top b)$

Example of how linear approximators work



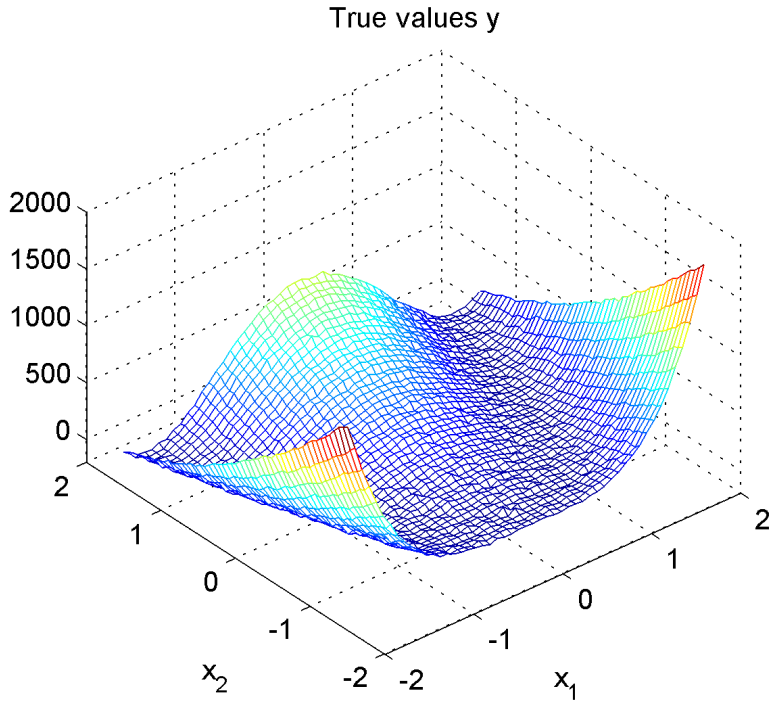
We approximate $\sin(x)$ on the interval $[0, 2\pi]$. Note: the method does not have access to the function, only to noisy samples!

Example of how linear approximators work



- Since the data has a “hill” and a “valley”, two RBFs should be enough
- The parameters θ_1, θ_2 scale & reflect the RBFs such that the approximate solution minimizes the sum of squared errors between the real and the approximated data.

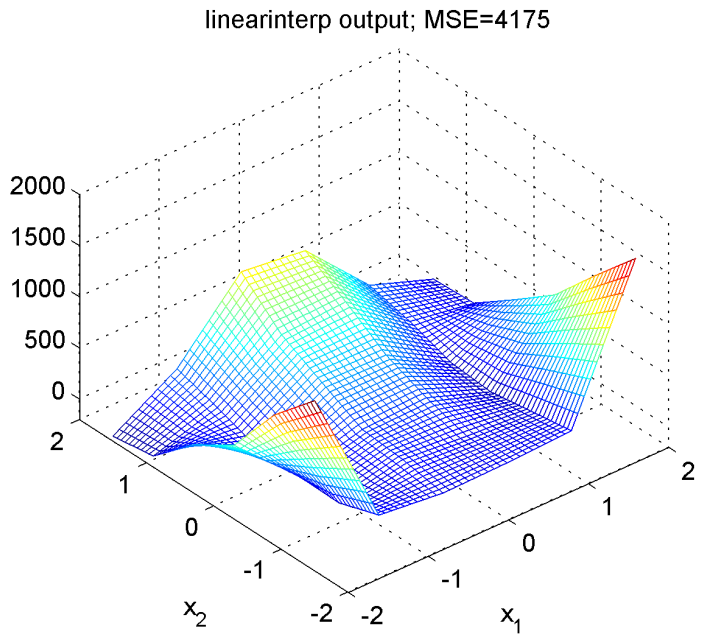
2D example: Rosenbrock's "banana" function



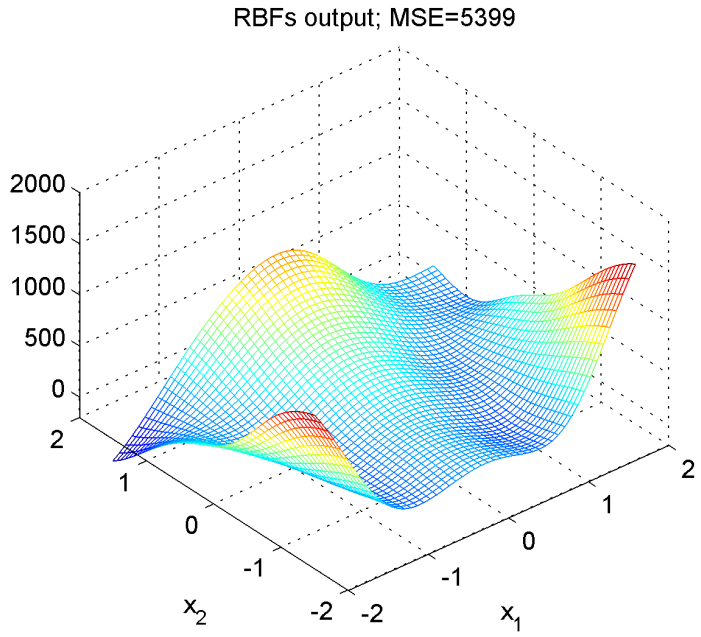
- $f(x) = (1 - x_1)^2 + 100[(x_2 + 1.5) - x_1^2]^2, \qquad x = [x_1, x_2]^\top$
- Training: 200 points, uniformly randomly distributed
- Validation: 31×31 point grid

Rosenbrock function: Linear approximator results

Interpolation on a 6x6 grid:



6x6 RBFs:

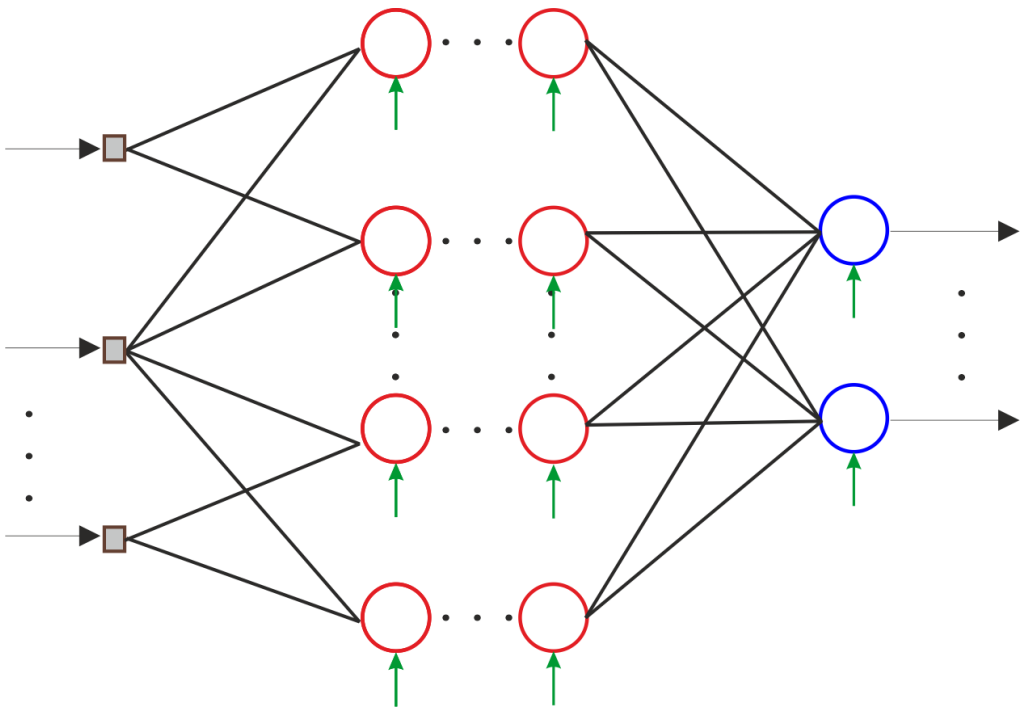


- Interpolation = collection of multilinear surfaces
- RBF approximation is smoother (wide RBFs)

Nonlinear parametric approximation: neural network

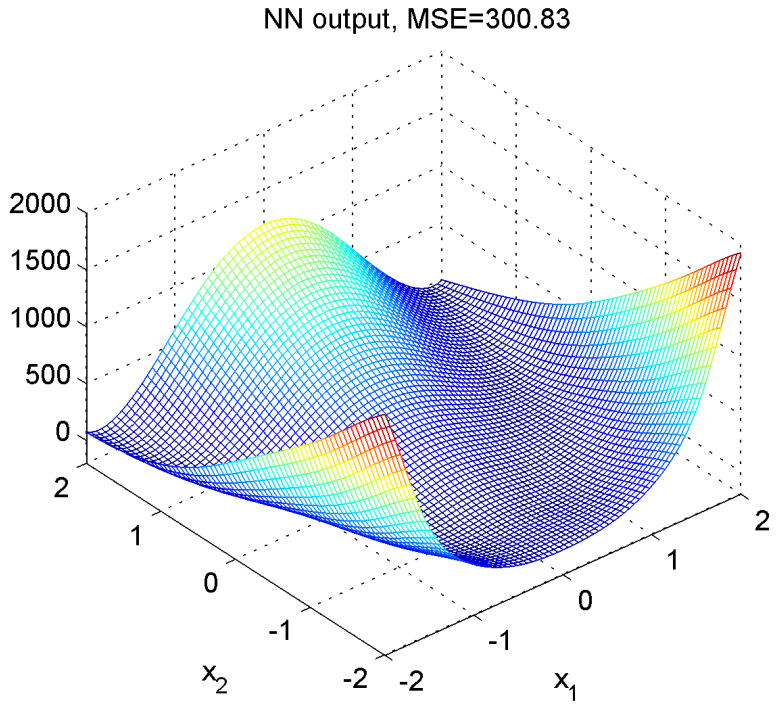
Neural network:

- Neurons with (non)linear activation functions
- Interconnected on multiple layers
  + 1 output layer
- Through **weighted connections** + biases



Rosenbrock function: Neural network result

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



Thanks to the greater flexibility of the neural network, the results are better than those with linear approximators.

Neural network to find features

Usually, the last layer of a neural network is linear, leading again to the feature-based approximator:

$$\widehat{f}(x;\theta) = \phi^\top(x;\theta_{\text{feat}})\theta_{\text{lin}}$$

where $\theta = [\theta_{\text{feat}}^T, \theta_{\text{lin}}^T]^T$.

Key difference from linear approximation: features ϕ are now themselves parametrized – by θ_{feat} – and **automatically found by the neural network**. In linear approximation, the features are manually designed.

Note that overall the approximator is nonlinear!

Neural network features

Example: Convolutional neural net features:



Details to follow

Comparison between approximators

- Linear approximators are easier to handle theoretically than nonlinear ones
- Nonlinear approximators are more flexible than linear ones

1 Approximation techniques

2 Approximation in DP&RL

3 Q-iteration with interpolation

4 Fitted Q-iteration

Approximation in RL

Problems to be solved:

1. Representation: $Q(x, u)$, $V(x)$, $h(x)$
   Using the approximation techniques discussed earlier

2. Maximization: e.g., $\max_u Q(x, u)$

Option 1: h implicit

- The policy is implicitly represented...
- ...by computing greedy actions on demand from $\widehat{Q}$:

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

$\Rightarrow$ Main problem: **approximating the Q-function**

- Approximator must ensure **efficient solution for** arg max
- Will be the focus in this lecture

Option 2: h explicit

- The policy is explicitly approximated: $\widehat{h}(x)$

Advantages:

- **Continuous actions** are easier to handle
- The representation can more easily incorporate **prior knowledge**

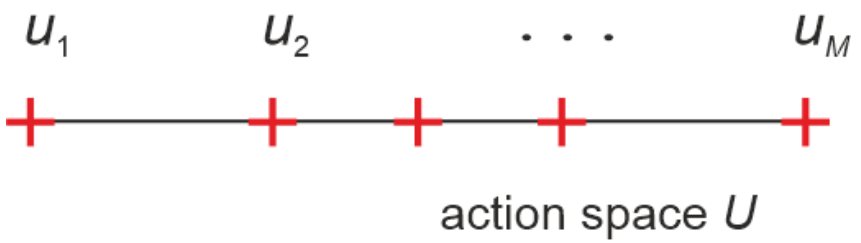
Action discretization

- Approximator must ensure efficient solution for arg max

$\Rightarrow$ Typically: **discretize actions**

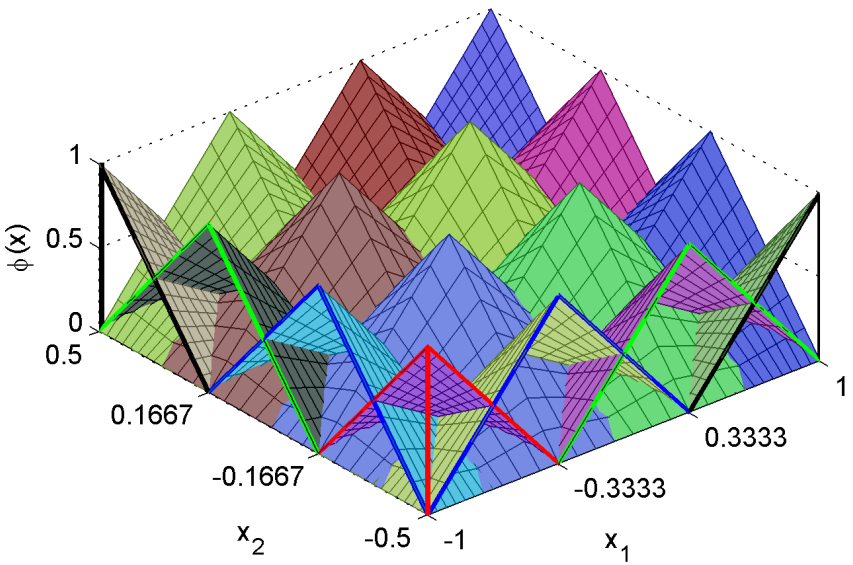
- Select M discrete actions $u_1, \ldots, u_j, \ldots, u_M \in U$
  Compute "arg max" using explicit enumeration

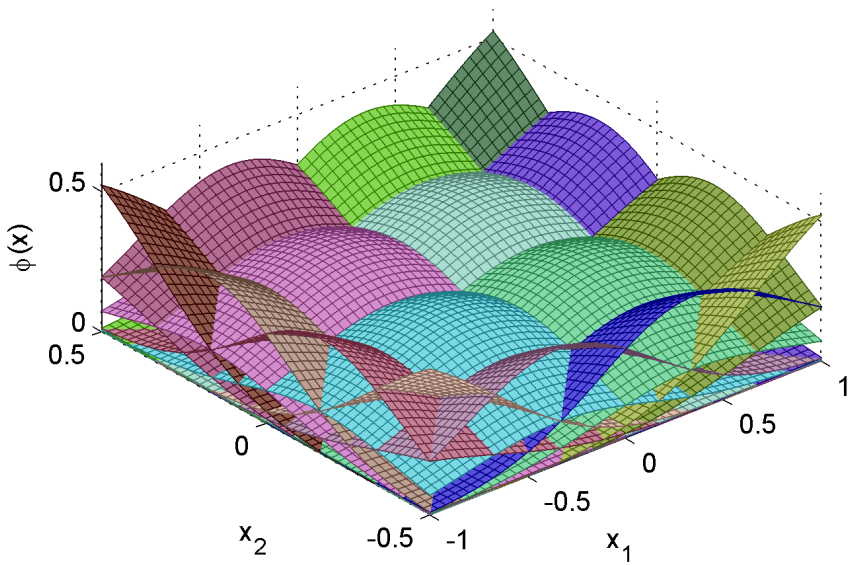
- Example: discretization on a grid



State-space approximation

- Often features $\phi_1, \ldots, \phi_N : X \to [0, \infty)$
- Ex. pyramidal, RBF





or perhaps found by neural network

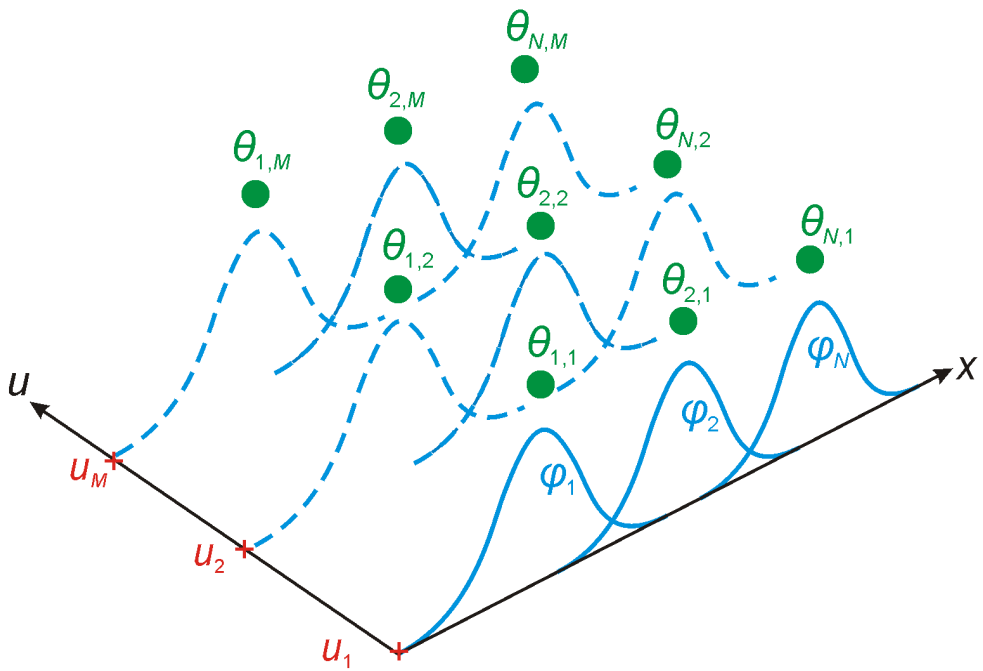
Approximating Q with discrete actions

Given:

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

Store:

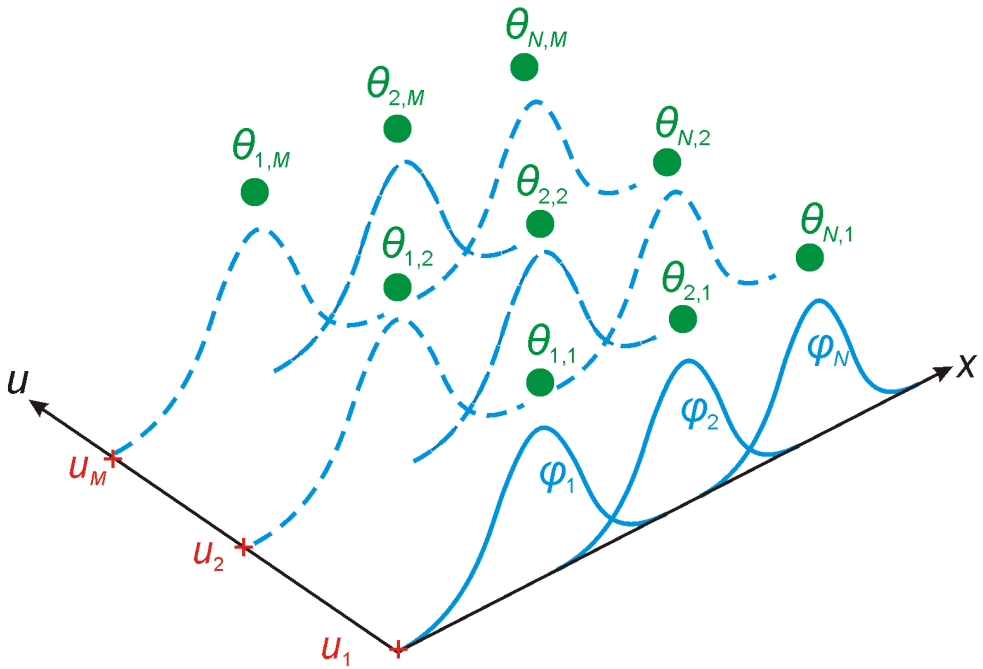
3. $N \cdot M$ parameters θ
   (for each feature–discrete action pair)



Approximating Q with discrete actions (continued)

Approximate Q-function:

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



Benefit of approximation in RL

Approximation allows applying RL to realistic problems

Simple control example: Inverted pendulum



- State $x = [\alpha, \dot{\alpha}]^\top$ with
  angle $\alpha \in [-\pi, \pi)$ rad,
  velocity $\dot{\alpha} \in [-15\pi, 15\pi]$ rad/s
- Action $u \in [-3, 3]$ V
- Dynamics:

$$\ddot{\alpha} = 1/J \cdot \left[mgl \sin(\alpha) - (b + \frac{K^2}{R})\dot{\alpha} + \frac{K}{R} u \right]$$

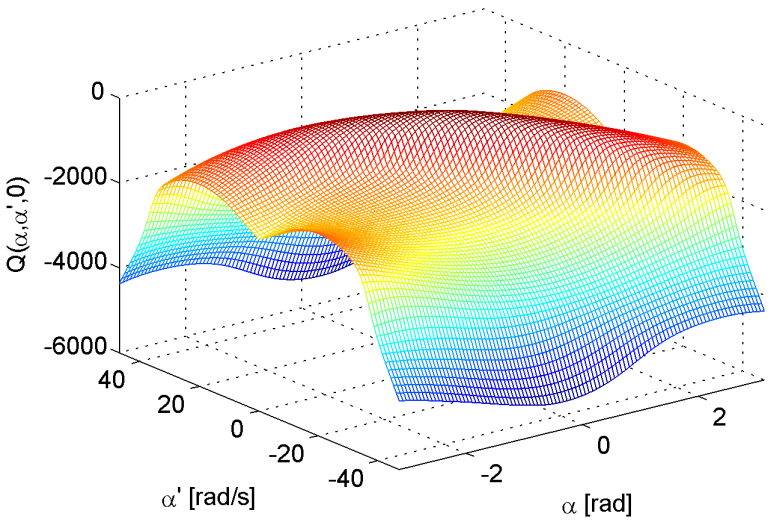
- Objective: stabilize pointing up, encoded by reward:

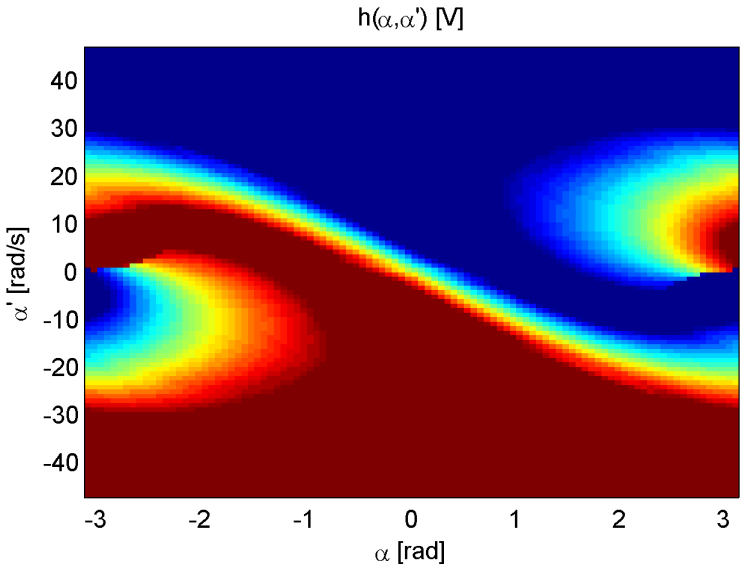
$$\rho(x, u) = -5\alpha^2 - 0.1\dot{\alpha}^2 - u^2$$

  normalized to $[0, 1]$
- Discount factor $\gamma = 0.98$
- Insufficient power $\Rightarrow$ swing back & forth before stabilizing

Inverted pendulum: Optimal solution

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





▸ Replay

New questions raised by approximation

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

Algorithm landscape

By model usage:

- Model-based: f, ρ known a priori
- Model-free: f, ρ unknown (only data; RL)

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)

1 Approximation techniques

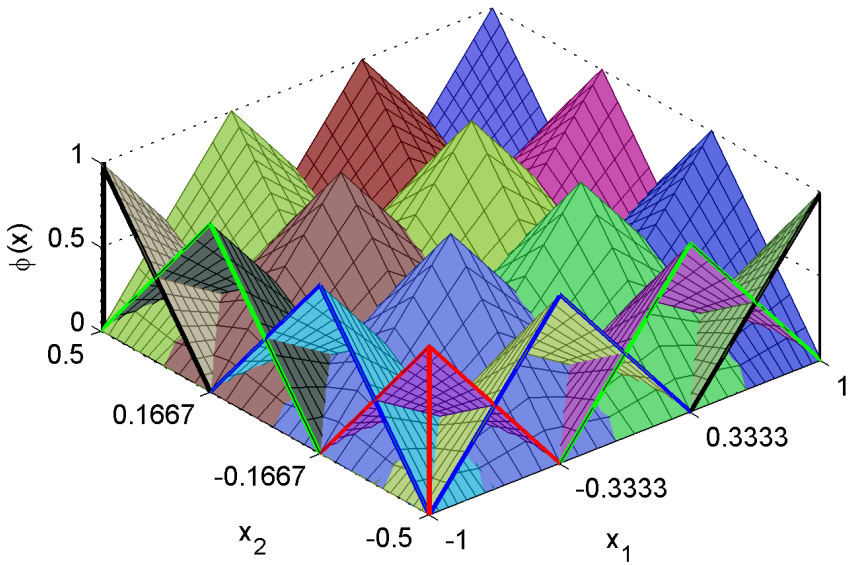
2 Approximation in DP&RL

3 Q-iteration with interpolation

4 Fitted Q-iteration

Q-function approximator

- Interpolation = pyramidal features



- Each feature i has center x_i
- $\theta_{i,j}$ can be viewed as $\widehat{Q}(x_i, u_j)$, since: $\phi_i(x_i) = 1, \phi_{i'}(x_i) = 0$ for $i' \neq i$

Q-iteration with interpolation

Recall classical Q-iteration:

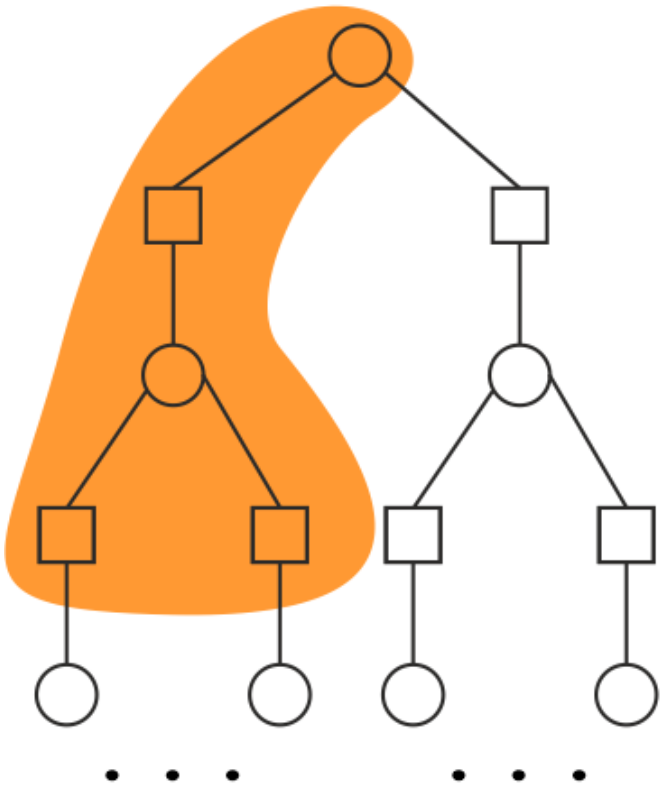
repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$
  end for
until convergence

Q-iteration with interpolation

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

Stochastic version exists, but here we only consider the deterministic case

Illustration



Policy

- Recall the optimal policy:

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

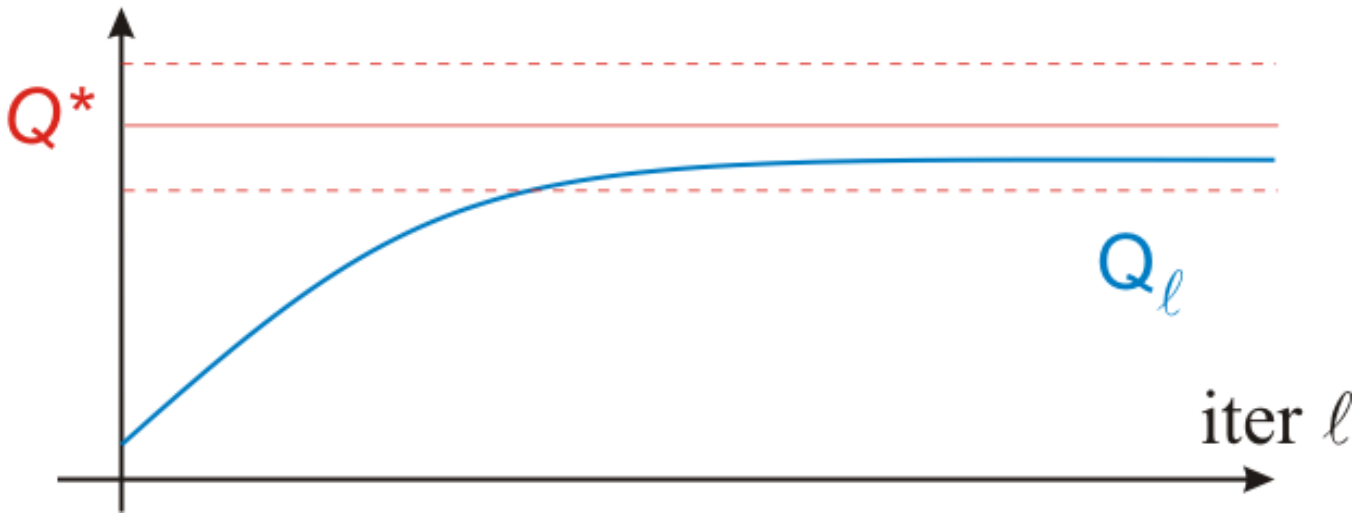
- When Q is approximated using action discretization (e.g. the interpolating approximator above):

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

θ^* = parameters at convergence

Q-iteration w/ interpolation: Illustration of properties

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



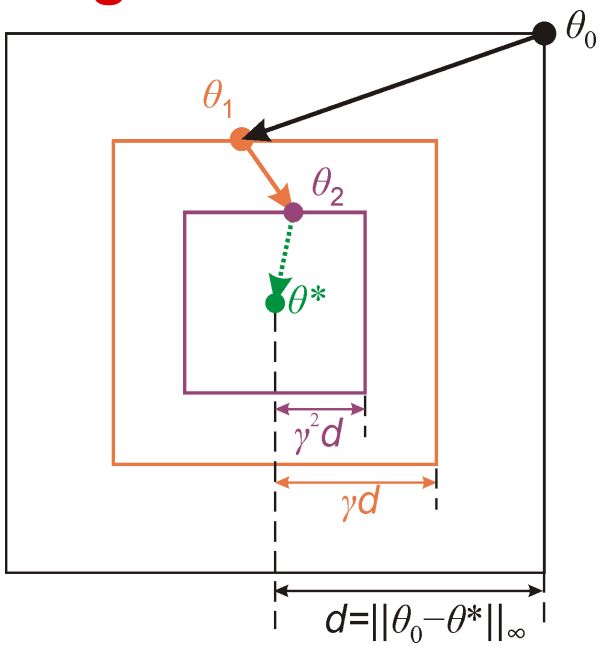
Convergence

Similar to classical Q-iteration:

- Each iteration is a contraction with factor γ:

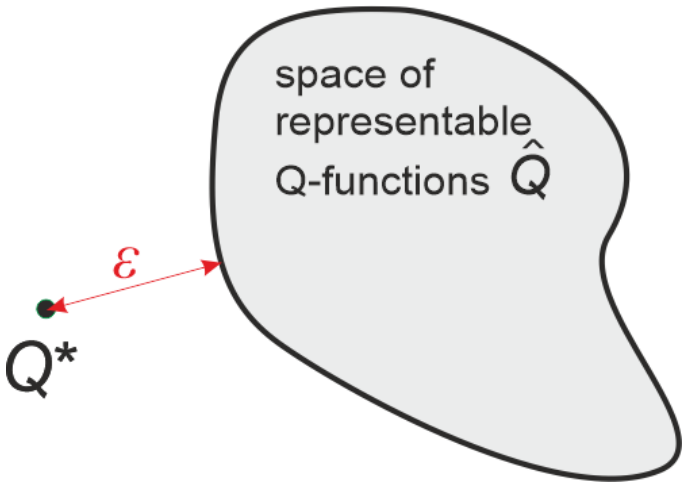
$$\|\theta_{\ell+1} - \theta^*\|_\infty \leq \gamma \|\theta_\ell - \theta^*\|_\infty$$

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



Near-optimality

Characterize approximator by the minimum distance to Q^*:



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

Then:

1. Suboptimality of resulting $\hat{Q}(x,u;\theta^*)$ is bounded:

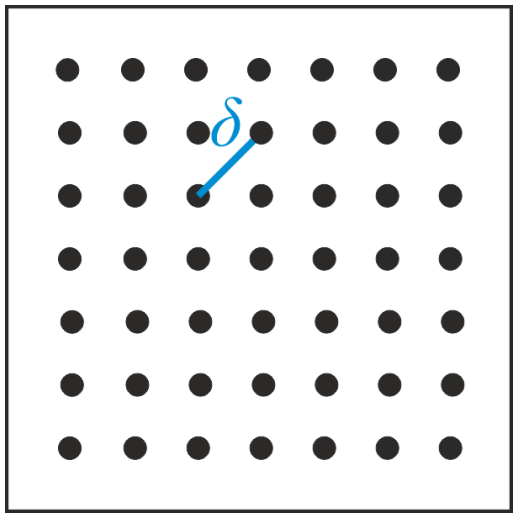
$$\left\| Q^*(x,u) - \hat{Q}(x,u;\theta^*) \right\|_\infty \leq \frac{2\varepsilon}{1-\gamma}$$

2. Suboptimality of policy $\hat{h}^*$ is bounded:

$$\left\| Q^*(x,u) - Q^{\hat{h}^*}(x,u) \right\|_\infty \leq \frac{4\varepsilon}{(1-\gamma)^2}$$

Consistency

- Consistency: $\widehat{Q}(x, u; \theta^*) \rightarrow Q^*(x, u)$ as resolution increases
- Precision measured by the resolution δ_x of the interpolation grid in x, and δ_u of the discretization grid in u



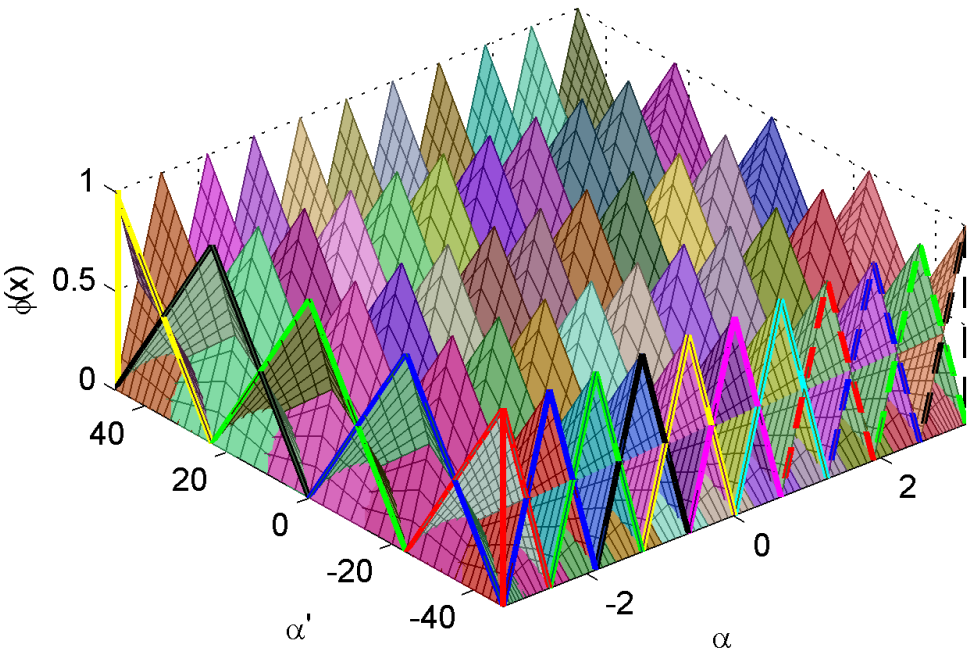
- Given certain technical conditions,

  $\Rightarrow \lim_{\delta_x \rightarrow 0, \delta_u \rightarrow 0} \widehat{Q}(x, u; \theta^*) = Q^*(x, u)$ — consistency

Inverted pendulum: Q-iteration w/ interpolation, demo

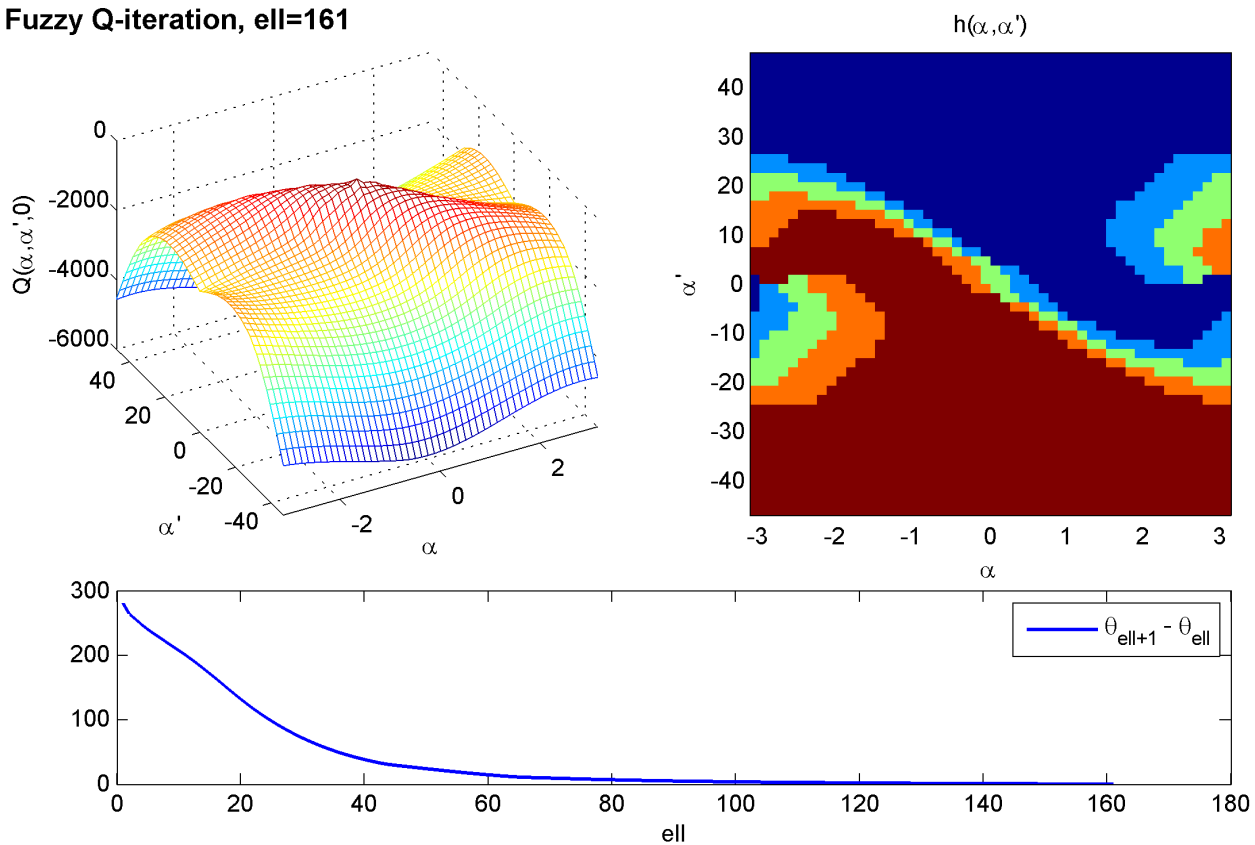
Features: equidistant grid 41×21

Discretization: 5 actions, distributed around 0





Inverted pendulum: Q-iteration w/ interpolation, demo



1 Approximation techniques

2 Approximation in DP&RL

3 Q-iteration with interpolation

4 Fitted Q-iteration

Fitted Q-iteration

Extend Q-iteration with interpolation so that it works:

- with other approximators
- **model-free** – RL

Intermediate model-based algorithm

Recall Q-iteration with interpolation:

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 approximator
3. Find parameters using least-squares

get state-action samples (x_s, u_s), $s = 1, \ldots, n_s$
repeat at each iteration ℓ
  for $s = 1, \ldots, n_s$ **do**
    compute bootstrapped target for $\widehat{Q}(x_s, u_s; \theta)$:
    $\hat{R}_s \leftarrow \rho(x_s, u_s) + \gamma \max_{u'} \widehat{Q}(f(x_s, u_s), u'; \theta_\ell)$
  end for
  $\theta_{\ell+1} \leftarrow \arg\min \frac{1}{n_s} \sum_{s=1}^{n_s} \left[\hat{R}_s - \widehat{Q}(x_s, u_s; \theta)\right]^2$
until termination

Q-iteration with interpolation is equivalent to this generalized algorithm if samples = all combinations x_i, u_j

Fitted Q-iteration: Algorithm

4. Use transitions instead of model

Fitted Q-iteration

get or collect dataset $\mathcal{D} = \{(x_s, u_s, r_s, x'_s), s = 1, \ldots, n_s\}$
repeat at each iteration ℓ
  for $s = 1, \ldots, n_s$ **do**
    compute bootstrapped target for $\widehat{Q}(x_s, u_s; \theta)$:
    $\hat{R}_s \leftarrow r_s + \gamma \max_{u'} \widehat{Q}(x'_s, u'; \theta_\ell)$
  end for
  $\theta_{\ell+1} \leftarrow \arg\min \frac{1}{n_s} \sum_{s=1}^{n_s} \left[\hat{R}_s - \widehat{Q}(x_s, u_s; \theta)\right]^2 =: \mathcal{L}(\theta)$
until termination

$\mathcal{L}$ is the **loss function**

Deterministic versus stochastic

- In the deterministic case, $x'_s = f(x_s, u_s)$, $r_s = \rho(x_s, u_s)$
  – substitutions are exact
- In the stochastic case, $x'_s \sim \tilde{f}(x_s, u_s, \cdot)$, $r_s = \tilde{\rho}(x_s, u_s, x'_s)$

$\Rightarrow$ Algorithm **remains valid**; intuition:

- Ideally, $Q(x,u) \leftarrow \mathrm{E}_{x'}\left\{r + \gamma \max_{u'} \widehat{Q}(x', u'; \theta_\ell)\right\}$
- Assuming for the moment all samples are at $(x_s, u_s) = (x, u)$,

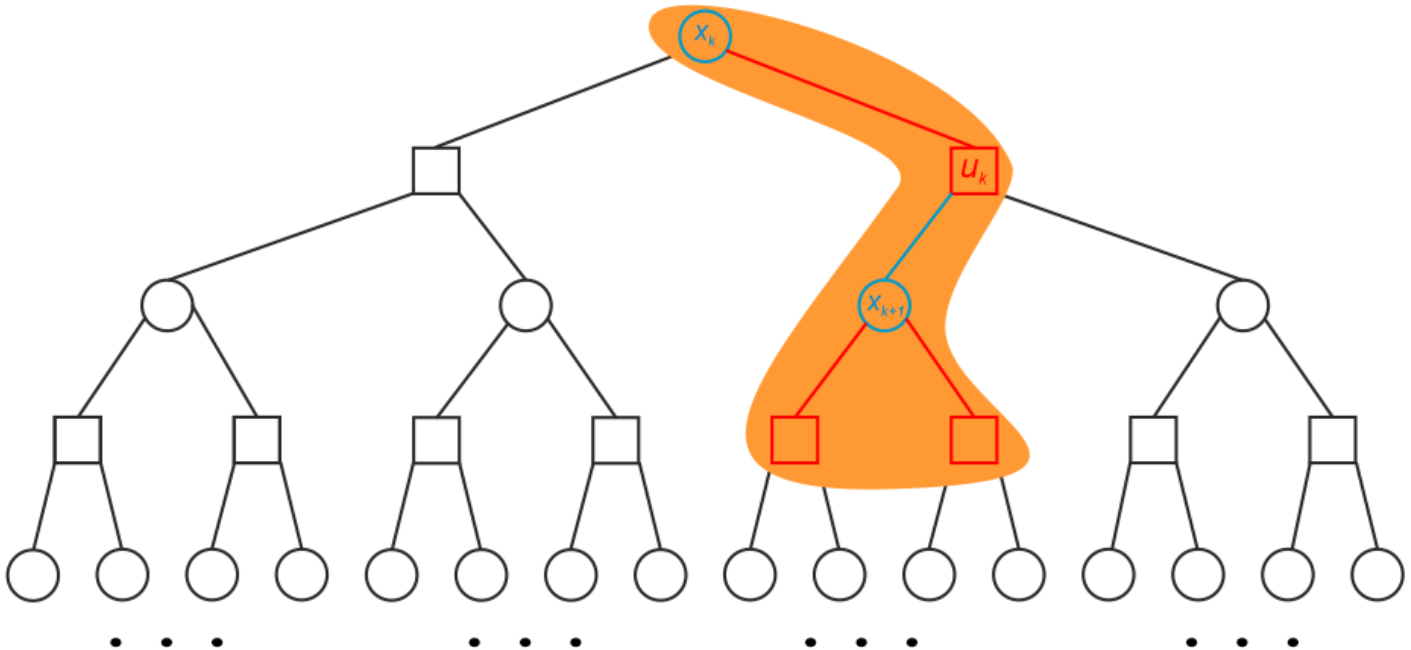
$$\min_\theta \frac{1}{n_s} \sum_{s=1}^{n_s} \left| r_s + \gamma \max_{u'} \widehat{Q}(x'_s, u'; \theta_\ell) - \widehat{Q}(x, u; \theta) \right|^2$$

  leads to $\widehat{Q}(x, u; \theta) \approx \mathrm{E}\{\ldots\}$ (like in Monte-Carlo)
- Even if (x_s, u_s) do not repeat, least-squares still approximate the expected value

- Similar to the fact that the incremental updates $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k[R_k - Q(x_k, u_k)]$ converge to the correct Q-values

Illustration

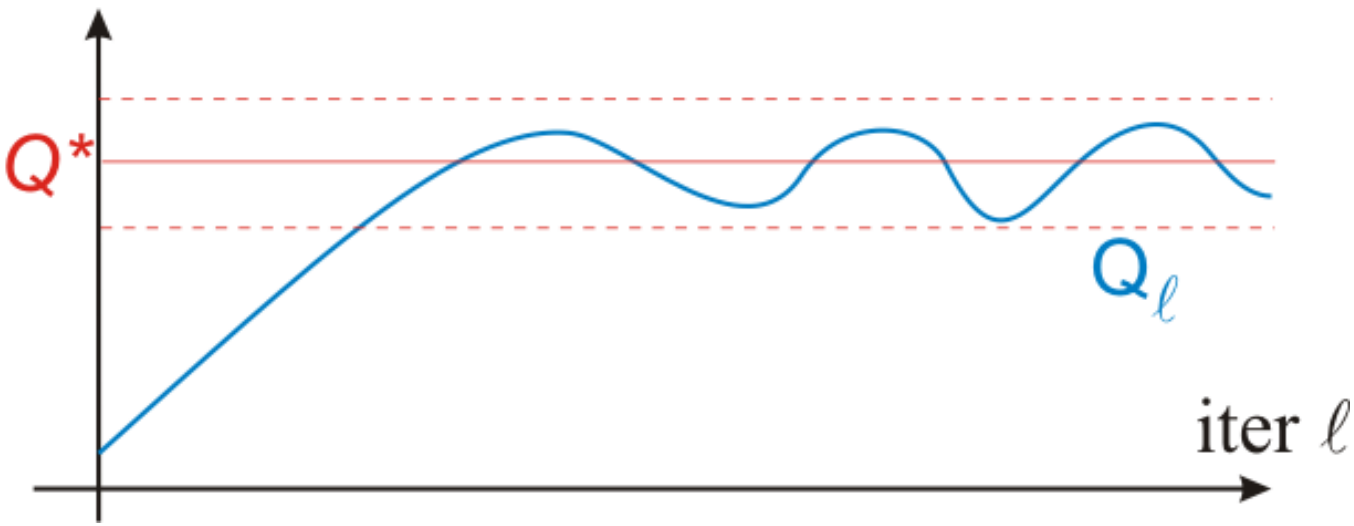
- Targets are bootstrapping estimates for Q^*, so the errors in the loss functions are temporal differences – like in TD methods.



- Like DP, fitted Q-iteration updates synchronously the complete Q-function across the entire state-action space.

Fitted Q-iteration: Illustration of properties

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

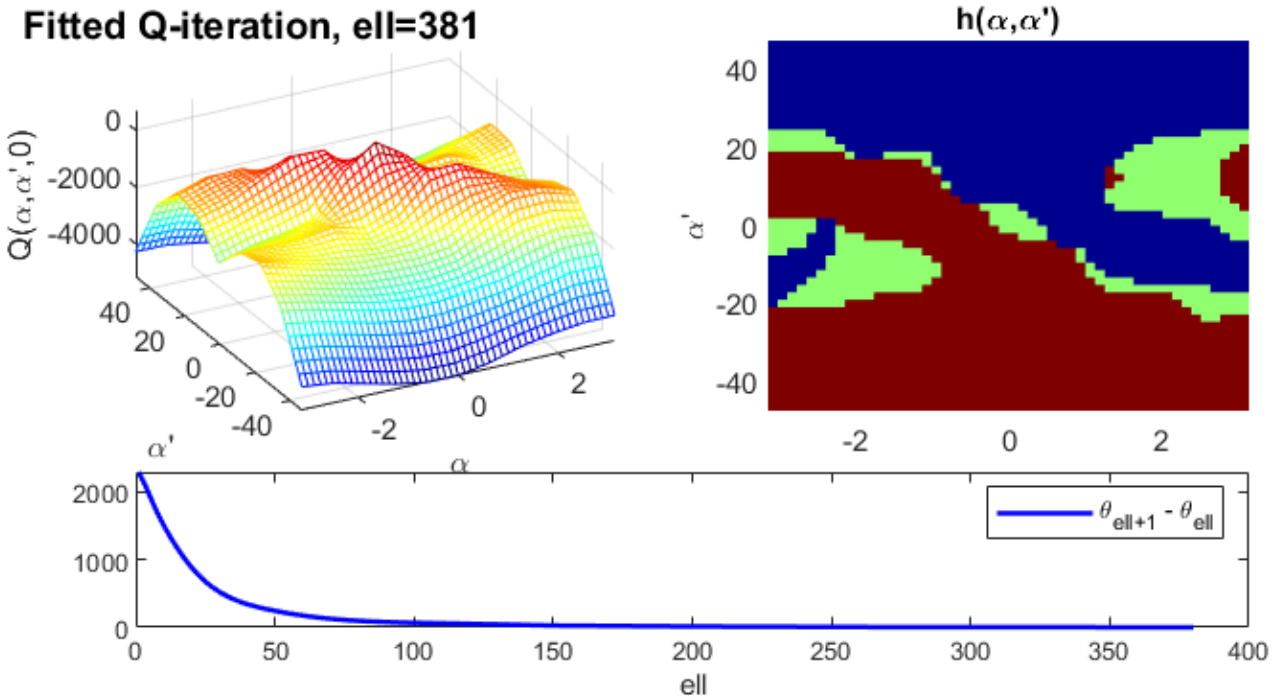


Inv. pend.: Fitted Q-iteration w/ interpolation, demo

Features: equidistant grid 11×9

Action discretization: 3 values, +/- max voltage and 0

Dataset: 10000 samples, uniformly distributed in the continuous x - discretized u space



Key terms in this part

- function approximation
- features / basis functions and parameters
- linear and nonlinear approximation
- interpolation, radial basis functions, neural network
- action discretization
- approximate dynamic programming
- fitted Q-iteration
- convergence, near-optimality, consistency

Exercises

1. Prove that the updates of Q-iteration with interpolation:

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

   are contractive with factor γ. Hint: it is essential that the features sum up to 1, and therefore the approximate Q-value is an average of the parameters!
2. What is the limit of the distance ε between Q^* and the space of representable Q-functions when $\delta_x \to 0, \delta_u \to 0$? Explain why.
3. Write an approximate V-iteration method with interpolation and prove that its updates are contractive.
4. Write a fitted V-iteration method. Is this method model-free?