Machine Learning for Systems and Control

5SC28

Lecture 6

dr. ir. Maarten Schoukens & dr. ir. Roland Tóth

Control Systems Group

Department of Electrical Engineering

Eindhoven University of Technology

Academic Year: 2020-2021 (version 1.0)

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Reinforcement Learning

Overview of RL

Approximate TD Learning for Prediction

Approximate TD Learning for Control

Deep Q-Networks

Reinforcement Learning

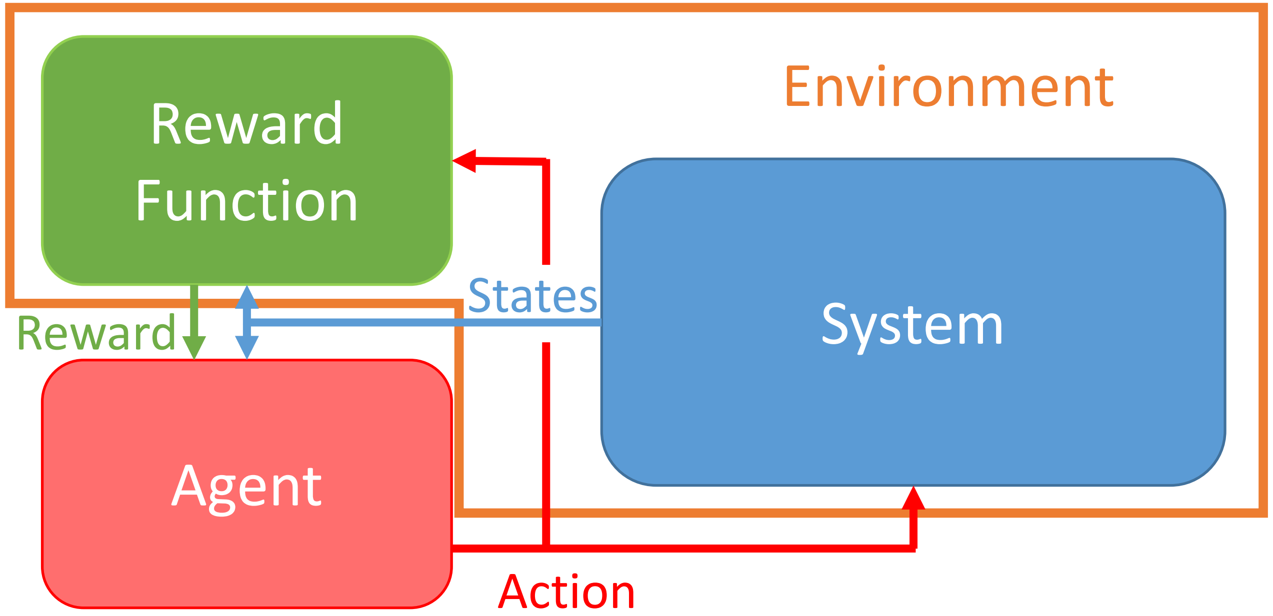
Overview of RL

Approximate TD Learning for Prediction

Approximate TD Learning for Control

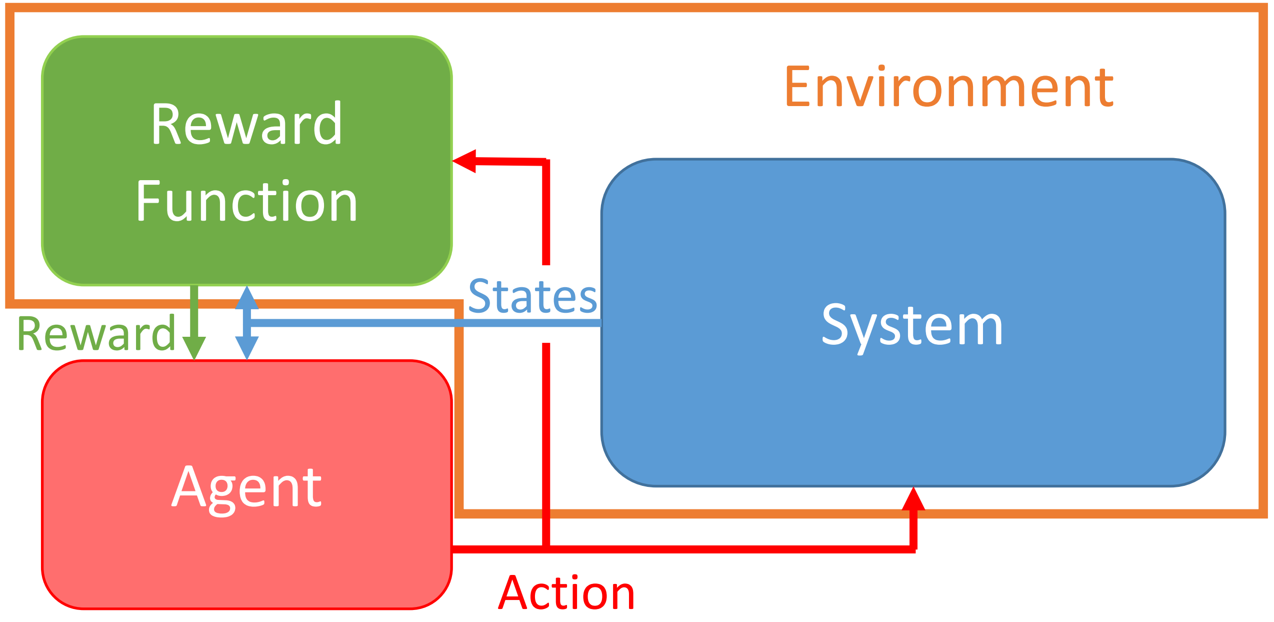
Deep Q-Networks

Principle of Reinforcement Learning (Recap)



- Agent interacts with the system through **States** and **Actions**
- Receive **Reward** as a performance feedback

Principle of Reinforcement Learning (Recap)



- Agent interacts with the system through **States** and **Actions**
- Receive **Reward** as a performance feedback
- This lecture: **approximate RL** – continuous states & actions

Reinforcement Learning

Overview of RL

Approximate TD Learning for Prediction

Approximate TD Learning for Control

Deep Q-Networks

Important ingredients

Discounted Return: $$G_k = \mathbb{E}_\pi \left\{ \sum_{\tau=0}^{\infty} \gamma^\tau r_{k+\tau+1} \right\}$$

Value Function: $$V_\pi(x_k) = \mathbb{E}_\pi \left\{ G_k | \, x_k \right\}$$

Q-Function: $$Q_\pi(x_t, u_t) = \mathbb{E}_\pi \left\{ G_k | \, x_k, u_k \right\}$$

Expected discounted return given the system states and the action taken at time k for policy π

The Prediction Problem

Estimate the Value Function $V_\pi(x)$ for a given policy π.

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size} \times [\text{Target - Old Estimate}]$$

$$V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \left[G_k - V_\pi(x_k)\right]$$

Incremental form of MC estimates

$$V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \underbrace{\left[r_{k+1} + \gamma V_\pi(x_{k+1}) - V_\pi(x_k)\right]}_{\text{Temporal Difference}}$$

Instead of waiting to see all obtain rewards G_k required for computing plug in the estimate $V_\pi(x_{k+1})$

The Prediction Problem

Estimate the Value Function $V_\pi(x)$ for a given policy π.

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size} \times [\text{Target - Old Estimate}]$$

$$V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \left[G_k - V_\pi(x_k)\right]$$

Incremental form of MC estimates

$$V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \left[r_{k+1} + \gamma V_\pi(x_{k+1}) - V_\pi(x_k)\right]$$

Temporal Difference

Fundamental update rule:

$$x_k \rightarrow \underbrace{\mu_{k+1}}_{\text{update traget}} = r_{k+1} + \gamma V_\pi(x_{k+1})$$

The Prediction Problem

What to do if

The state-space has many discrete elements $\mathrm{Card}(X) \approx 10^5$

- Tabular representations quickly run out of memory
- We can not wait till we explore all states
- Need for approximation / generalization capability

The state-space is continuous

- All points can not be visited (finite time) $X = \mathbb{R}^{n_\mathrm{x}}$
- How to handle functional representations of the value function, Q-function?

Approximation of The Value Function

Concept:

Parameters $\theta \in \mathbb{R}^d$

$$V_\pi(x_k) \approx \hat{V}_\pi(x_k, \theta)$$

Approximate value function

Approximation of The Value Function

Concept:

Parameters $\theta \in \mathbb{R}^d$

$$V_\pi(x_k) \approx \hat{V}_\pi(x_k, \theta)$$

How to optimize θ ?

Approximate value function

We need to mimic function behavior for observed input samples x_k and output samples μ_k

Lecture 2-4:
Function Estimation

Stochastic Gradient Method (SGD)

Cost function: $$\mathrm{VE}(\theta) = \sum_{x \in X} (V_\pi(x) - \hat{V}_\pi(x, \theta))^2$$

Observation: $$x_k \rightarrow V_\pi(x_{k+1})$$

Differentiable function

Gradient search: $$\theta_{k+1} = \theta_k - \frac{1}{2}\alpha \nabla_\theta \left[V_\pi(x_k) - \hat{V}_\pi(x_k, \theta_k) \right]^2$$

$$= \theta_k + \alpha \left[V_\pi(x_k) - \hat{V}_\pi(x_k, \theta_k) \right] \nabla_\theta \hat{V}_\pi(x_k, \theta_k)$$

Step size

Gradient

Stochastic Gradient Method for TD

Objective: $V_\pi(x_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k \right\}$

Observation: $x_k \to \underbrace{\mu_{k+1}}_{\text{update target}} = r_{k+1} + \gamma V_\pi(x_{k+1}) \approx G_k$

Gradient search: $\theta_{k+1} = \theta_k + \alpha \left[r_{k+1} + \gamma \hat{V}_\pi(x_{k+1}, \theta_k) - \hat{V}_\pi(x_k, \theta_k) \right] \nabla_\theta \hat{V}_\pi(x_k, \theta_k)$

Otherwise it can wander ⇦ Asymptotic convergence if $\mathbb{E}_\pi\{\mu_{k+1} \mid x_k = x\} = V_\pi(x)$ & $\alpha_k \to 0$ slow enough

Approximate Temporal Difference Learning

Input: the policy π to be evaluated
Parameters: step size $\alpha \in (0, 1]$
Initialize θ arbitrarily (e.g., $\theta = 0$)

for each episode **do**
  Initialize x_0
  Repeat for each time step of the episode
    Obtain u_k based on x_k using policy π

    Take action u_k, observe r_{k+1}, x_{k+1}

$$\theta \leftarrow \theta + \alpha \left[r_{k+1} + \gamma \hat{V}_\pi(x_{k+1}, \theta) - \hat{V}_\pi(x_k, \theta) \right] \nabla_\theta \hat{V}_\pi(x_k, \theta)$$

$$k = k + 1$$

  Until the states are terminal
end for

Similar algorithm for MC and DP (Bootstrapping) type of methods.

Choice of Parametrization

Linear parameterization

Features (basis functions)

$\phi_i : X \to \mathbb{R}$

$$\hat{V}_\pi(x, \theta) = \theta^\top \Phi(x) = \sum_{i=1}^{d} \theta_i \phi_i(x)$$

Choice of Parametrization

Linear parameterization

Features (basis functions)

$$\phi_i : X \to \mathbb{R}$$

$$\hat{V}_\pi(x, \theta) = \theta^\top \Phi(x) = \sum_{i=1}^{d} \theta_i \phi_i(x)$$

$$\nabla_\theta \hat{V}_\pi(x_k, \theta_k)$$

SGD search:

$$\theta_{k+1} = \theta_k + \alpha \left[r_{k+1} + \gamma \hat{V}_\pi(x_{k+1}, \theta_k) - \hat{V}_\pi(x_k, \theta_k) \right] \Phi(x_k)$$

$$= \theta_k + \alpha \left[r_{k+1} \Phi(x_k) - \Phi(x_k)(\Phi(x_k) - \gamma \Phi(x_{k+1}))^\top \theta_k \right]$$

Choice of Parametrization

Linear parameterization

Features (basis functions)

$$\phi_i : X \to \mathbb{R}$$

$$\hat{V}_\pi(x,\theta) = \theta^\top \Phi(x) = \sum_{i=1}^{d} \theta_i \phi_i(x)$$

$$\nabla_\theta \hat{V}_\pi(x_k, \theta_k)$$

SGD search:

$$\begin{aligned}\theta_{k+1} &= \theta_k + \alpha \left[r_{k+1} + \gamma \hat{V}_\pi(x_{k+1}, \theta_k) - \hat{V}_\pi(x_k, \theta_k) \right] \Phi(x_k) \\ &= \theta_k + \alpha \left[r_{k+1}\Phi(x_k) - \Phi(x_k)(\Phi(x_k) - \gamma \Phi(x_{k+1}))^\top \theta_k \right]\end{aligned}$$

Consequence:
Linear param. + quadratic cost = convex problem

Guaranteed convergence
global opt. = local opt.

Choice of Parametrization

Convergence

$$\theta_{k+1} = \theta_k + \alpha \left[\underbrace{r_{k+1}\Phi(x_k)} - \underbrace{\Phi(x_k)(\Phi(x_k) - \gamma\Phi(x_{k+1}))^\top}\theta_k\right]$$

$$b = \mathbb{E}\{r_{k+1}\Phi(x_k)\} \qquad A = \mathbb{E}\{\Phi(x_k)(\Phi(x_k) - \gamma\Phi(x_{k+1}))^\top\}$$

when converged: $\theta_k = \theta_k + \alpha \underbrace{\left[b - A\theta_k\right]}_{=0}$

Fixed point: $\theta_{\mathrm{TD}} = A^{-1} b$

must be positive definite

This can be also used as an update rule (LS-TD)

$$\mathrm{VE}(\theta_{\mathrm{TD}}) \leq \frac{1}{1-\gamma} \min_\theta \mathrm{VE}(\theta)$$

Choice of Parametrization

Convergence

$$\theta_k = \theta_k + \alpha \underbrace{[b - A\theta_k]}_{=0}$$

Fixed point: $\theta_{\mathrm{TD}} = A^{-1} b$

must be positive definite

$$\mathrm{VE}(\theta_{\mathrm{TD}}) \leq \frac{1}{1-\gamma} \min_{\theta} \mathrm{VE}(\theta)$$

Typically close to 1 ➔ Very loose upper bound

Feature Construction

Construction based on fixed selection

Polynomial features:

Integer in $\{0, \ldots, n\}$

$$\phi_i(x) = \prod_{j=1}^{n_\mathrm{x}} x_j^{c_{i,j}}$$

n th -order polynomial basis

$$x = \begin{pmatrix} x_1 & \cdots & x_{n_\mathrm{x}} \end{pmatrix}$$

Number of features: $(n+1)^{n_\mathrm{x}}$

Easily differentiable

Example:

$$\Phi(x) = \begin{pmatrix} 1 & x_1 & x_2 & x_1 x_2 & x_1^2 & x_2^2 & x_1 x_2^2 & x_1^2 x_2 & x_1^2 x_2^2 \end{pmatrix}$$

Feature Construction

Construction based on fixed selection

Fourier features:

$$c_i = \begin{pmatrix} c_{i,1} & \cdots & c_{i,n_\mathrm{x}} \end{pmatrix}$$

$$c_{i,j} \in \{0, \ldots, n\}$$

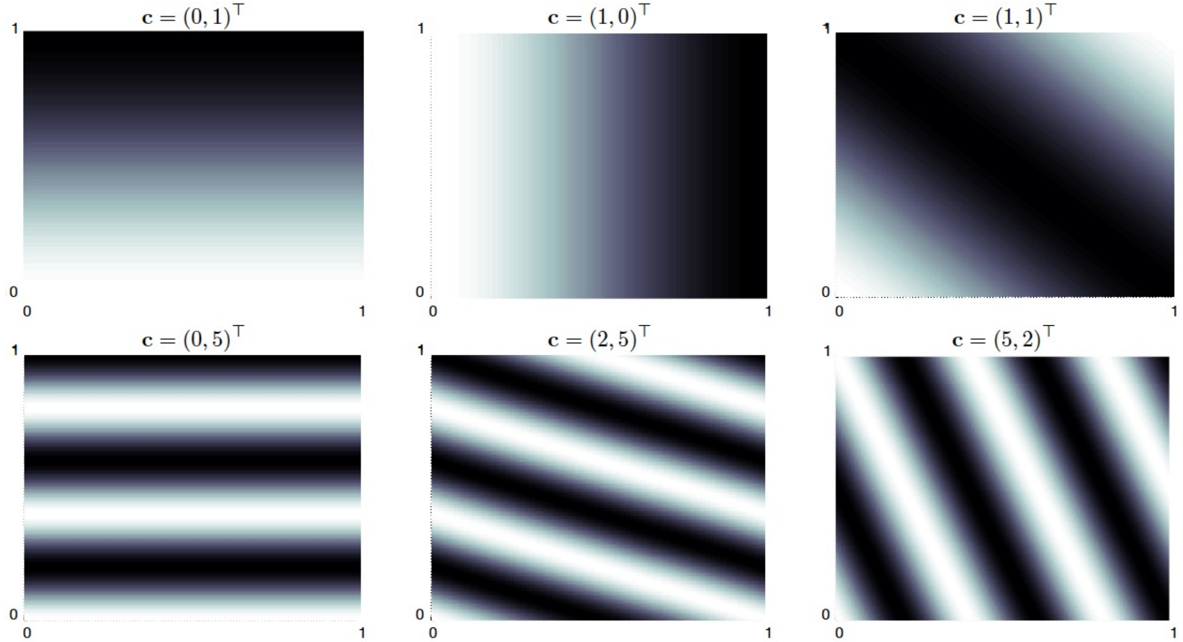
$$\phi_i(x) = \cos(\pi x^\top c_i)$$

n^{th} -order Fourier basis

Number of features:

$$(n+1)^{n_\mathrm{x}}$$

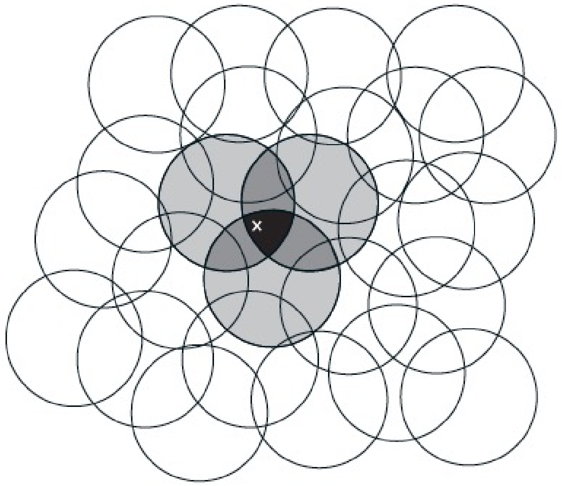
Easily differentiable



Feature Construction

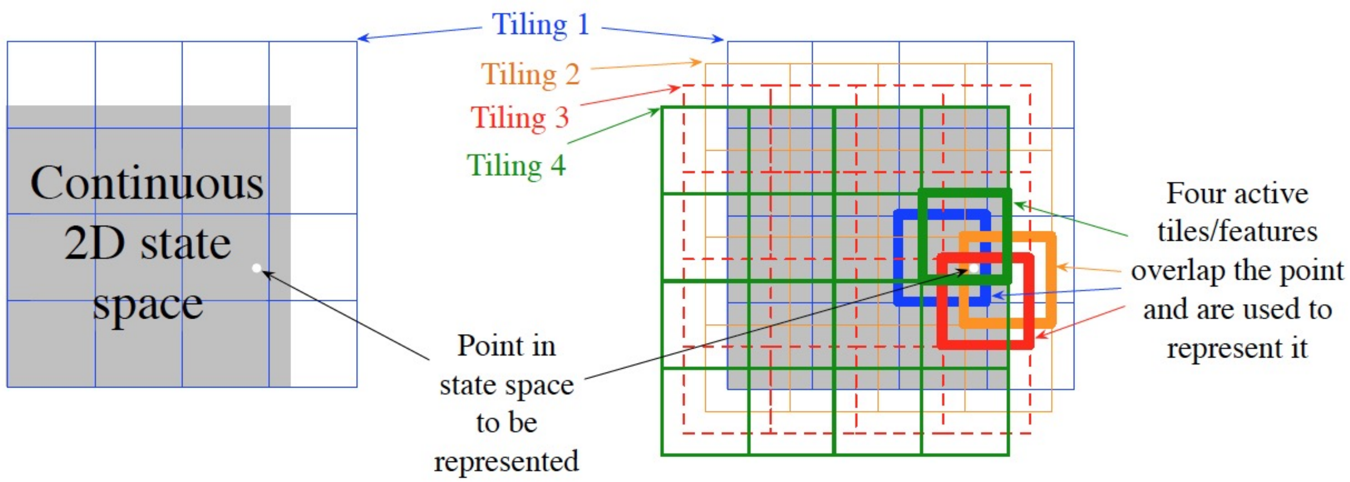
Construction based on fixed selection

Coarse coding



$$\phi_i(x) = \begin{cases} 1 & x \in \text{region } i \\ 0 & \text{else} \end{cases}$$

Tile coding



Similar to convolutional layer in ANN

Feature Construction

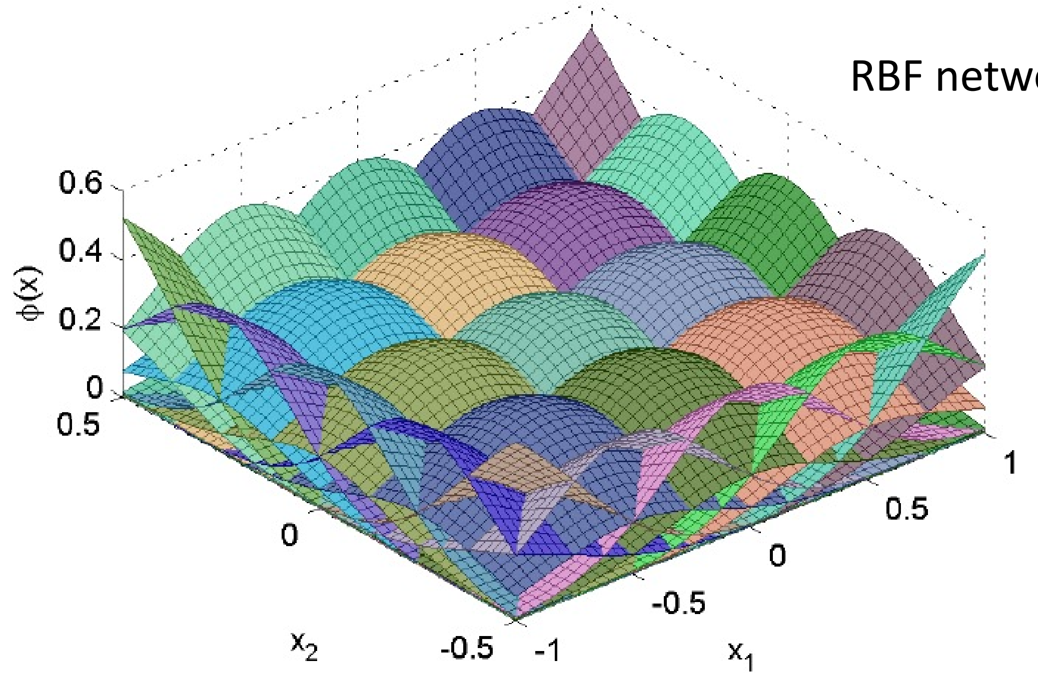
Construction based on fixed selection

Kernel-based

$$\phi_i(x) = K(x, x_i)$$

Kernel function
(differentiable)

RBF network

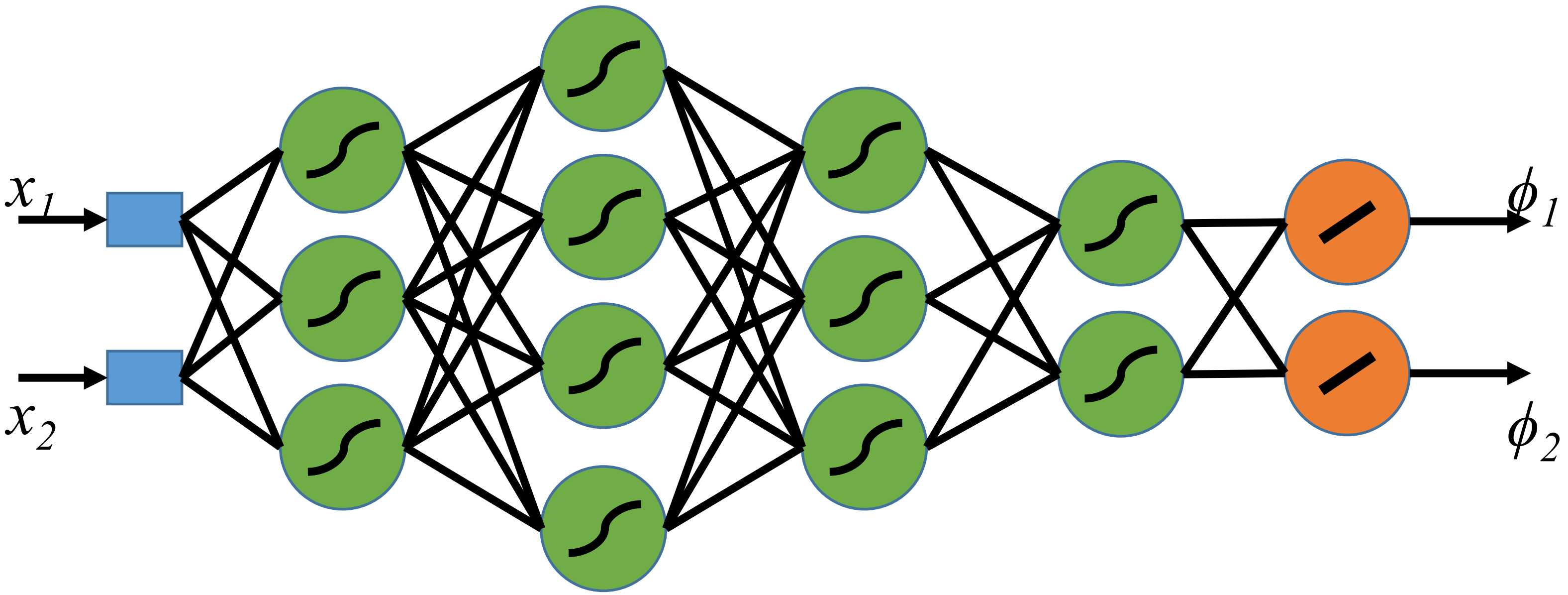


GP-based generalization (non-fixed selection):

- Store the samples (or only the relevant ones: support features)
- Directly apply GP (with marginalized likelihood optimization for the hyper-parameters)

Feature Construction

Construction based on ANN



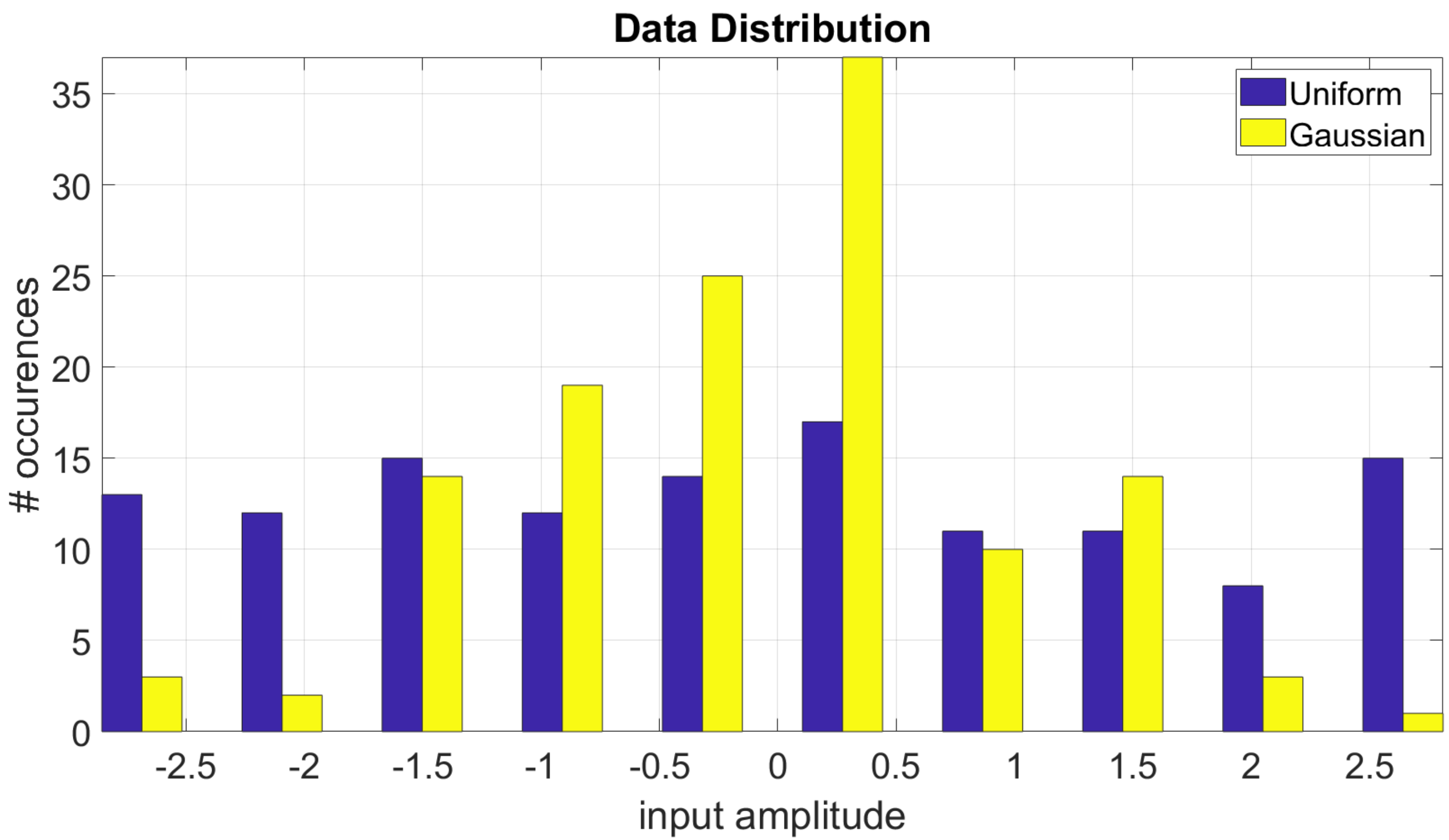
Same concepts apply as in identification

Use backpropagation

$$\theta_{k+1} = \theta_k + \alpha \left[r_{k+1} + \gamma \hat{V}_\pi(x_{k+1}, \theta_k) - \hat{V}_\pi(x_k, \theta_k) \right] \nabla_\theta \hat{V}_\pi(x_k, \theta_k)$$

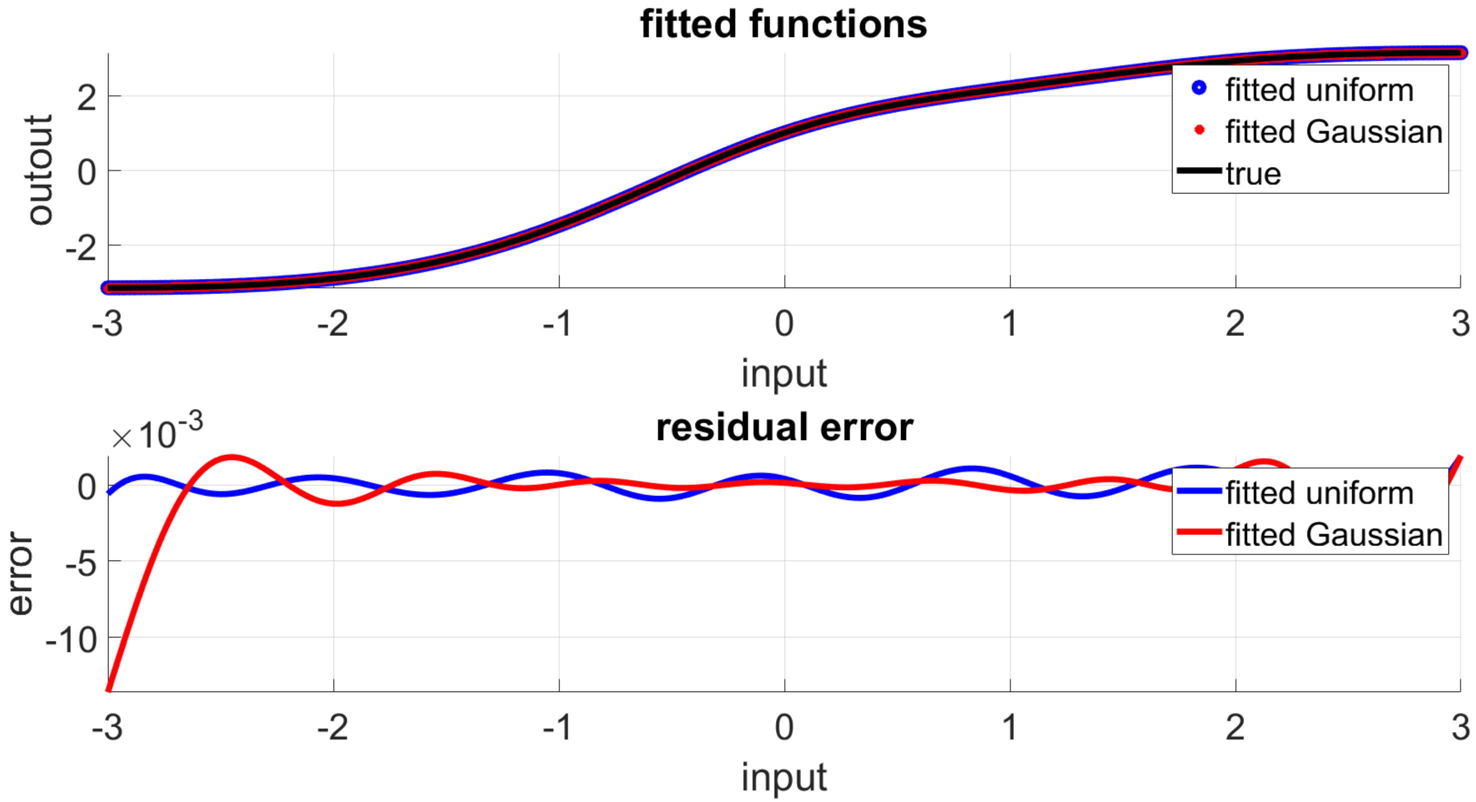
Function Approximation

Data distribution becomes important in function approximation algorithms



Function Approximation

Data distribution becomes important in function approximation algorithms



Function Approximation

Data distribution becomes important in function approximation algorithms

→ it influences how the approximation errors are spread

Can be problematic for high-dimensional state spaces

Compensate for data-distribution by using weighted cost functions

$$\mathrm{VE}(\theta) = \sum_{x \in X} w_x \left(V_\pi(x) - \hat{V}_\pi(x, \theta) \right)^2$$

Reinforcement Learning

Overview of RL

Approximate TD Learning for Prediction

Approximate TD Learning for Control

Deep Q-Networks

The control problem

For a control objective, we want to estimate

Q-Function: $$Q_\pi(x_k, u_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k, u_k \right\}$$

For a given policy π

OR

Optimal Q-Function: $$Q_*(x_k, u_k) = \sum_{x_{k+1}, r} p(x_{k+1}, r \mid x_k, u_k) \left(r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)$$

The control problem

For a control objective, we want to estimate

Q-Function: $$Q_\pi(x_k, u_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k, u_k \right\}$$

For a given policy π

OR

Optimal Q-Function: $$Q_*(x_k, u_k) = \sum_{x_{k+1}, r} p(x_{k+1}, r \mid x_k, u_k) \left(r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)$$

TO FIND

Optimal policy: $$\pi_*(x_k) = \arg\max_{u_k} Q_*(x_k, u_k)$$

Greedy in $Q_*(x_k, u_k)$

Types of RL Control

By path to optimal solution

- Off-policy – find Q_*, use it to compute π_*
- On-policy – find Q_π, improve π, repeat

By level of interaction with the process

- Online – learn by interacting with the process
- Offline – data collected in advance (Monte-Carlo methods)

By model knowledge

- Model-free – no p, only transition data (standard RL)
- Model-based – p is known (Dynamic Programming)
- Model-learning – estimate p from transition data

Q-Learning (off policy, online, model-free)

Take **Bellman optimality equation** at some state and action

$$Q_*(x_k, u_k) = \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left(r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)$$

Turn into **iterative update**

$$Q(x_k, u_k) \leftarrow \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left(r + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1}) \right)$$

Instead of a transition model, use the **transition sample** at each step

$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1})$$

Q-Learning (off policy, online, model-free)

Implemented with incremental update

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left(\underbrace{r_{k+1} + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)}_{\text{Temporal Difference}} \right)$$

Learning rate

$\alpha \in (0, 1]$

How to use this over continuous spaces?

Approximation of The Q Function

Concept:

Parameters $\theta \in \mathbb{R}^d$

$$Q_\pi(x_k, u_k) \approx \hat{Q}_\pi(x_k, u_k, \theta)$$

Approximate Q function

Policy computation:

$$\pi(x_k) = \arg\max_{u_k} \hat{Q}(x_k, u_k, \theta)$$ (greedy policy)

Approximation of The Q Function

Parametrization of $\hat{Q}(x, u, \theta)$:

- Must be differentiable with simple computation of the gradient, like

$$\hat{Q}(x, u, \theta) = \sum_{i=1}^{d} \theta_i \phi_i(x, u)$$

(linear parametrization)

- Policy optimization should be simple

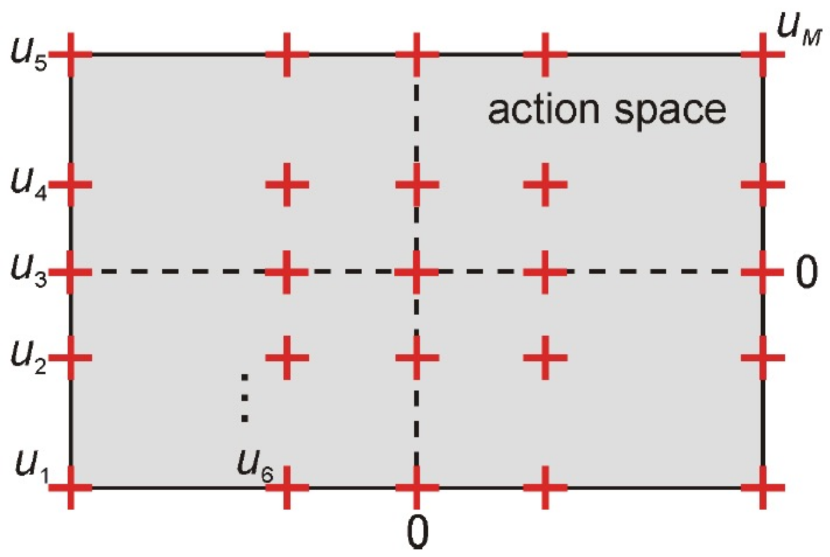
$$\pi(x) = \arg\max_u \sum_{i=1}^{d} \theta_i \phi_i(x, u)$$

(cont. optimization)

Approximation of The Q Function

Parametrization of $\hat{Q}(x, u, \theta)$:

- Feature construction: same as for prediction
- Policy optimization: typically, action discretization is used:



Action space is gridded:

$$u_1, \dots, u_M \in U$$

Policy computation:

$$\pi(x_k) = \arg \max_{u \in \{u_j\}_{j=1}^M} \hat{Q}(x_k, u, \theta)$$

(simple max operation)

Approximate Q-Learning

Based on SGD:

$$\theta_{k+1} = \theta_k + \alpha \underbrace{\left[r_{k+1} + \gamma \max_{u_{k+1}} \hat{Q}(x_{k+1}, u_{k+1}, \theta) - \hat{Q}(x_k, u_k, \theta) \right]}_{\text{Temporal Difference}} \nabla_\theta \hat{Q}(x_k, u_k, \theta)$$

Learning rate

$\alpha \in (0, 1]$

Approximate Q-Learning

Parameters: step size $\alpha \in (0, 1]$ and $0 < \epsilon < 1$
Initialize $\hat{Q}(x, u, \theta)$ arbitrarily (e.g., $\theta = 0$)

for each episode **do**
  Initialize x_0
  Repeat for each time step of the episode
    Obtain u_k based on x_k using policy π derived from $\hat{Q}$
    $\boxed{\text{(e.g., } \epsilon\text{-greedy)}}$ common choice

    Take action u_k, observe r_{k+1}, x_{k+1}

$$\theta_{k+1} = \theta_k + \alpha \left[r_{k+1} + \gamma \max_u \hat{Q}(x_{k+1}, u, \theta) - \hat{Q}(x_k, u_k, \theta) \right] \nabla_\theta \hat{Q}(x_k, u_k, \theta)$$

    $k = k + 1$
  Until the states are terminal
end for

Approximate Q-Learning

Properties

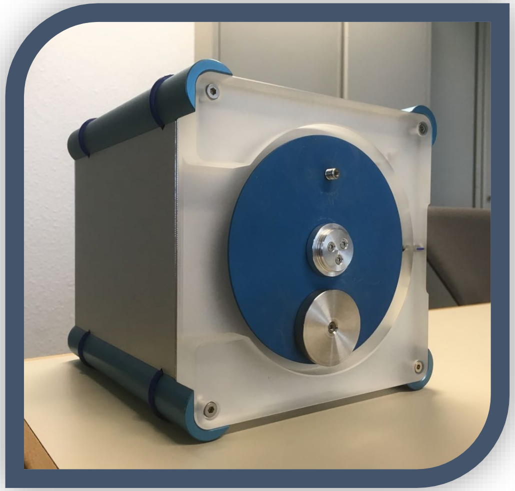
On-policy approximate methods: convergence can be achieved

- The promise of future reward must be kept, and the corresponding action is needed to be taken for convergence.

Off-policy approximate methods: no convergence guarantees

- Promise of future reward is made, but then another action might follow and the promise, including its error, is forgotten.
- Approximate Q-learning with ε–greedy policy can diverge, but usually works well in practice!

Example



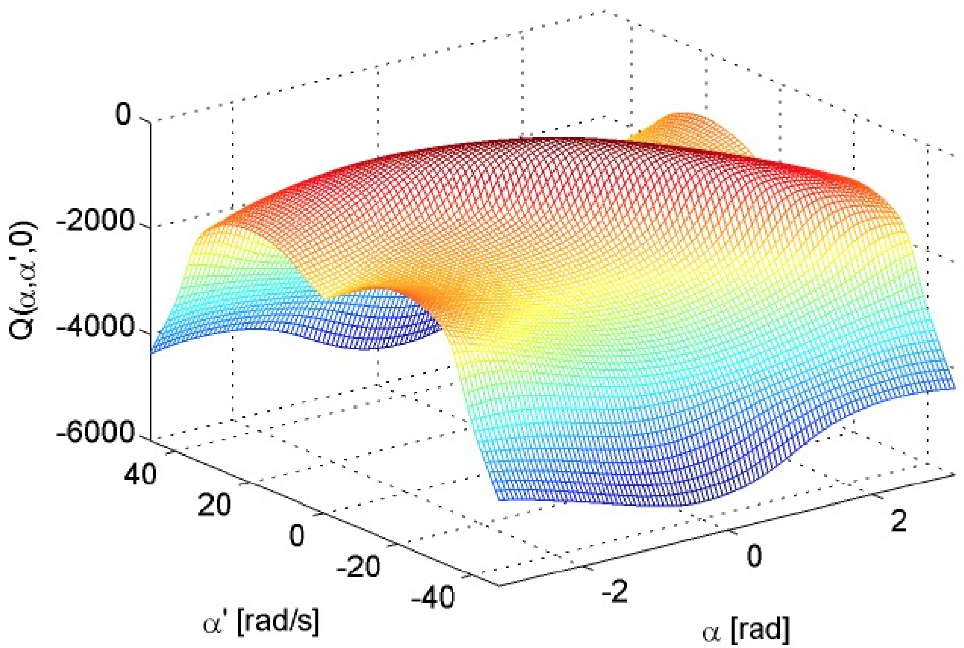
- State: $x = \begin{pmatrix} \alpha & \dot{\alpha} \end{pmatrix}$ (angle and velocity)
- Input: $u = \text{voltage}$
- Reward: $r_{k+1} = \rho(x_k, u_k) = -x_k^\top \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix} x_k - u_k^\top u_k$
- Discount: $\gamma = 0.98$

- Objective: stabilize top position (swing up)
- Insufficient actuation (needs to swing back & forth)

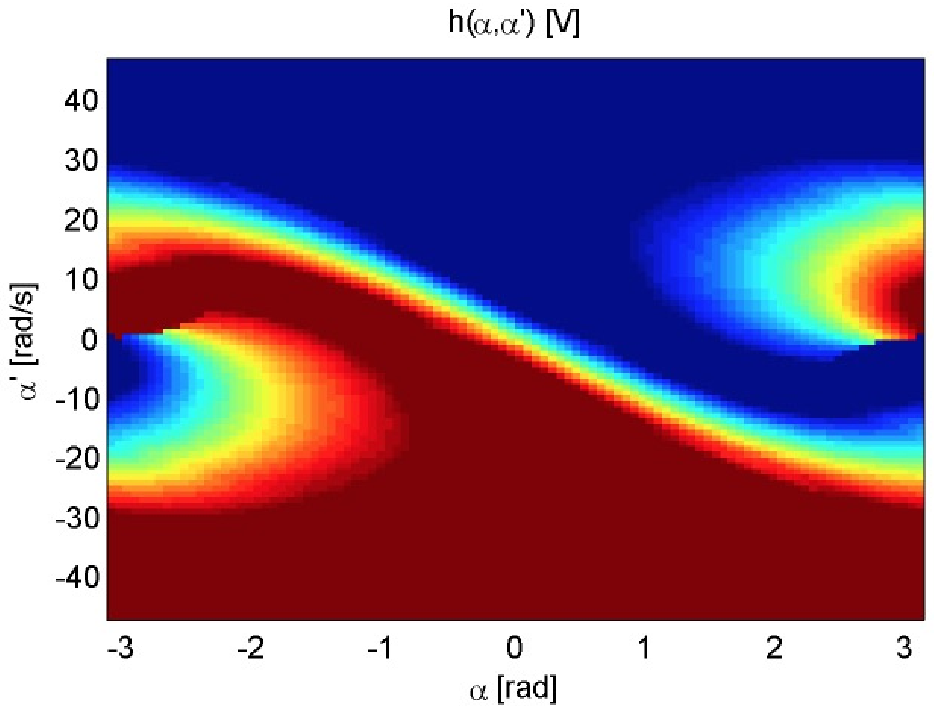
Example from SC42050 - TUDelft

Example

Q-function for $u = 0$



Policy $\pi(\alpha, \dot{\alpha})$

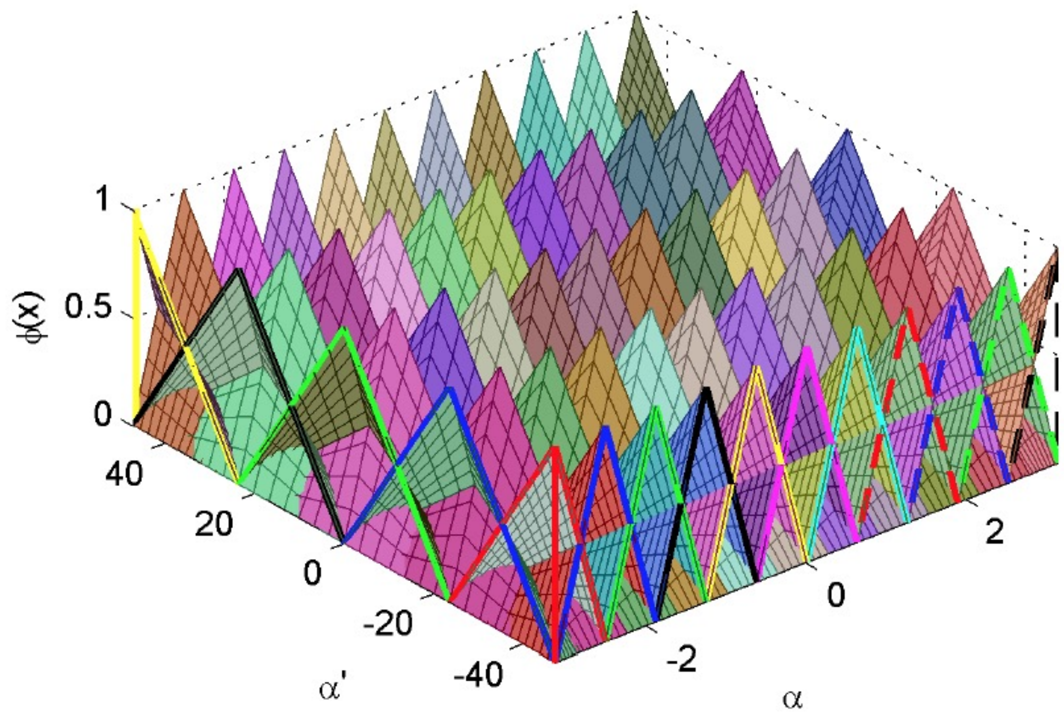


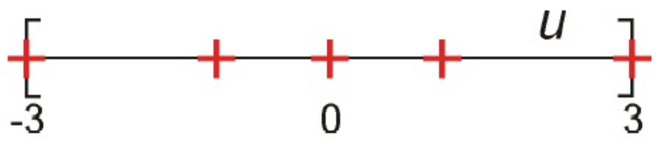
Near optimal solution

Example

Features: Triangular membership functions: 41x21 equidistant grid

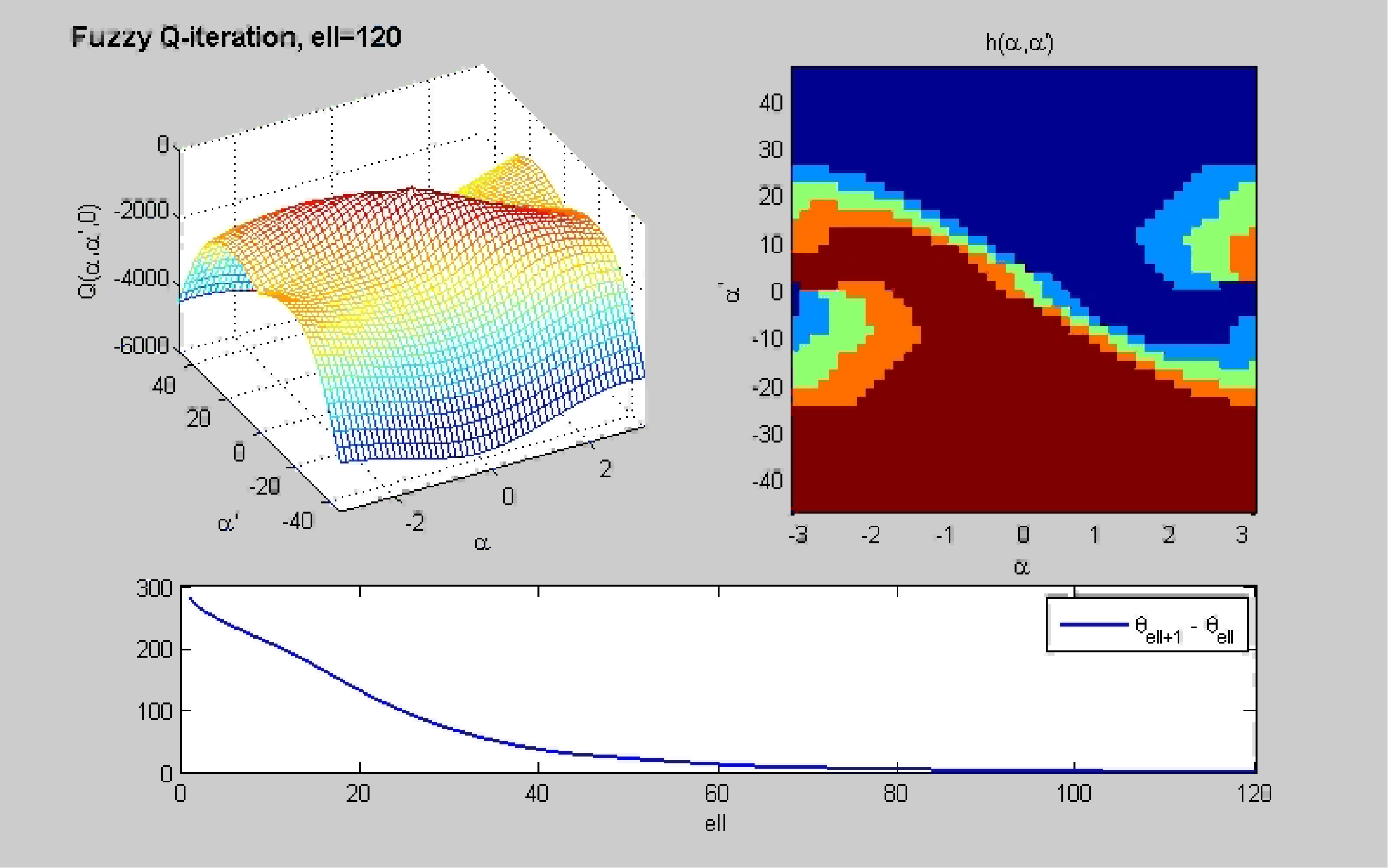
Input space: 5 actions, log-placed around 0





Example

Results:



Reinforcement Learning

Overview of RL

Approximate TD Learning for Prediction

Approximate TD Learning for Control

Deep Q-Networks

Deep Q Networks (DQN)

Parameters

$$Q_\pi(x_k, u_k) \approx \hat{Q}_\pi(x_k, u_k, \theta)$$

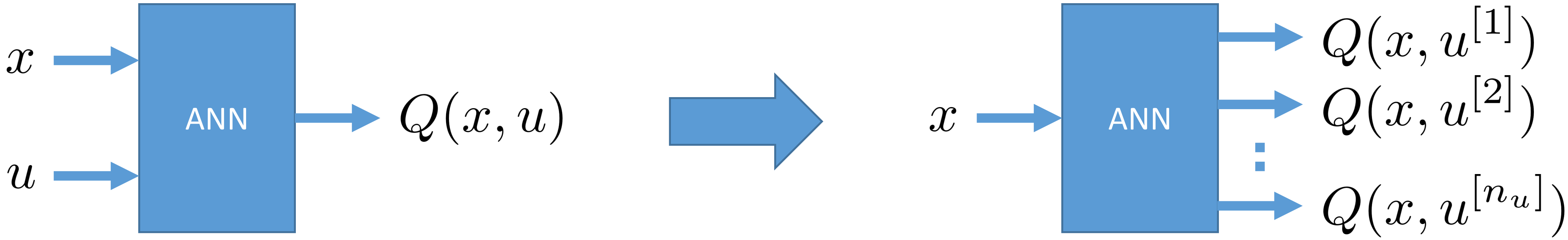
Approximate Q function
with deep neural net

Deep Q Networks (DQN)

Parameters

$$Q_\pi(x_k, u_k) \approx \hat{Q}_\pi(x_k, u_k, \theta)$$

Approximate Q function
with deep neural net



Requires network evaluation for each possible action

Requires only one network evaluation

Deep Q Networks (DQN)

$$\theta_{k+1} = \theta_k + \alpha \left[r_{k+1} + \gamma \max_{u_{k+1}} \hat{Q}(x_{k+1}, u_{k+1}, \theta) - \hat{Q}(x_k, u_k, \theta) \right] \nabla_\theta \hat{Q}(x_k, u_k, \theta)$$

Learning rate

$\alpha \in (0, 1]$

Temporal Difference

Backpropagation

Deep Q Networks (DQN)

$$\theta_{k+1} = \theta_k + \alpha \left[\underbrace{r_{k+1} + \gamma \max_{u_{k+1}} \hat{Q}(x_{k+1}, u_{k+1}, \theta)}_{\text{Non-stationary target}} - \hat{Q}(x_k, u_k, \theta) \right] \nabla_\theta \hat{Q}(x_k, u_k, \theta)$$

Challenge: **Non-stationary target**

The target is continuously changing with each iteration. In deep learning, the target variable does not change and hence the training is stable, which is just not true for RL.

Challenge: **Correlated trajectories**

Samples are received from a trajectory of a dynamical system, hence the data is strongly correlated over (short) timeframes.

Break the Correlation: Experience Replay

Idea: Introduce a memory that stores pervious [action - state transitions - reward] values

Usage: Sample mini-batches from the memory to compute the gradient at each optimization iteration

Result:

When replay *memory is large* ➔ experience replay is *close to sampling independent transitions* from an explorative policy.

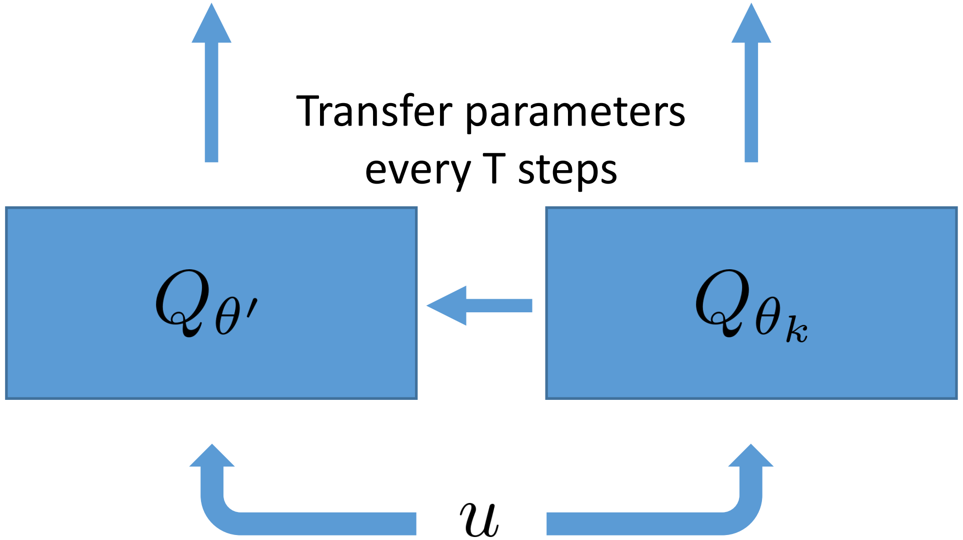
This *reduces the variance of the gradient*, which is used to update θ.

Experience replay *stabilizes the training of DQN*, which benefits the algorithm in terms of computation.

Solve the Non-Stationarity: Target Network

target prediction

$$\left(r_{k+1} + \gamma \max_u Q(x_{k+1}, u, \theta') - Q(x_k, u_k, \theta_k)\right)^2$$



Create a separate **Target Network**:

- Same architecture as training network
- Frozen parameters
- Transfer parameters every T step

Stabilizes Q-Network training

Bias-Variance Trade-Off of the Bellman Error

$$Y_k = r_{k+1} + \gamma \mathrm{max}_u Q_{\theta^\star}(x_{k+1}, u)$$

Targets

$$\mathrm{VE}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - Q_\theta(x_i, u_i)\right)^2$$

Cost

Bias-Variance Trade-Off of the Bellman Error

$$Y_k = r_{k+1} + \gamma \mathrm{max}_u Q_{\theta^\star}(x_{k+1}, u)$$

Targets

$$\mathrm{VE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Q_\theta(x_i, u_i))^2$$

Cost

Bias-Variance Decomposition

$$\boxed{\mathbb{E}\{\mathrm{VE}(\theta)\} = \|Q_\theta - TQ_\theta\|^2 + \mathbb{E}\left\{[Y_1 - (TQ_\theta)(x_1, u_1)]^2\right\}}$$

$$(TQ)(x_k, u_k) = r_{k+1} + \gamma \mathbb{E}\left\{\max_u Q(x_{k+1}, u) \middle| x_k, u_k\right\}$$

Bellman operator
(see previous lecture)

$$(TQ^*) = Q^*$$

Bellman operator applied on optimal Q function = optimal Q function

Bias-Variance Trade-Off of the Bellman Error

Without Target Network

Bias-Variance Decomposition

$$\boxed{\mathbb{E}\{\mathrm{VE}(\theta)\} = \|Q_\theta - TQ_\theta\|^2 + \mathbb{E}\left\{[Y_1 - (TQ_\theta)(x_1, u_1)]^2\right\}}$$

Mean Squared Bellman Error (MSBE)

Variance of Y_1

Both depend on θ!

⇒ Minimizing the cost function is different than minimizing the MSBE

Bias-Variance Trade-Off of the Bellman Error

Without Target Network

Bias-Variance Decomposition

$$\boxed{\mathbb{E}\{\mathrm{VE}(\theta)\} = \|Q_\theta - TQ_\theta\|^2 + \mathbb{E}\left\{[Y_1 - (TQ_\theta)(x_1, u_1)]^2\right\}}$$

What we actually want to minimize since for the optimal Q function it holds that:

$$(TQ^*) = Q^*$$

Mean Squared Bellman Error (MSBE)

Variance of Y_1

Both depend on θ!

Minimizing the cost function is different than minimizing the MSBE

Bias-Variance Trade-Off of the Bellman Error

With Target Network

Bias-Variance Decomposition

$$\boxed{\mathbb{E}\{\mathrm{VE}(\theta)\} = \|Q_\theta - TQ_{\theta'}\|^2 + \mathbb{E}\left\{[Y_1 - (TQ_{\theta'})(x_1, u_1)]^2\right\}}$$

Mean Squared Bellman Error (MSBE)

Variance of Y_1

dependent on θ

independent from θ

Minimizing the cost function is close to solving

$$\mathrm{minimize}_\theta \|Q_\theta - TQ_{\theta'}\|^2$$

Bias-Variance Trade-Off of the Bellman Error

With Target Network

Bias-Variance Decomposition

$$\boxed{\mathbb{E}\{\mathrm{VE}(\theta)\} = \|Q_\theta - TQ_{\theta'}\|^2 + \mathbb{E}\left\{[Y_1 - (TQ_{\theta'})(x_1, u_1)]^2\right\}}$$

Mean Squared Bellman Error (MSBE)

Variance of Y_1

dependent on θ

independent from θ

Minimizing the cost function is close to solving

$$\mathrm{minimize}_\theta \, \|Q_\theta - TQ_{\theta'}\|^2$$

By iteratively (but slowly) updating θ' we hope to converge to:

$$(TQ^*) = Q^*$$

DQN Algorithm

Input: a family of deep Q-networks Q_θ

Parameters: stepsize α, exploration probability $\epsilon \in (0,1)$
Parameters: update freq. T_{target}, minibatch size n, replay memory $\mathcal{M}$

Initialize θ arbitrarily (e.g., $\theta = 0$)
Initialize the replay memory $\mathcal{M}$ to be empty

for each episode **do**
    Initialize x_0
    Repeat for each time step of the episode
        Obtain u_k based on x_k using policy π derived from $\hat{Q}$ (e.g., ϵ-greedy)

        Take action u_k, observe r_{k+1}, x_{k+1}

replay

        Store transition $(x_k, a_k, r_{k+1}, x_{k+1})$ in $\mathcal{M}$

        Sample random minibatch $(x_i, u_i, r_{i+1}, x_{i+1})_{i\in[n]}$ from $\mathcal{M}$

minibatch SGD

$$\theta_{k+1} = \theta_k + \alpha \frac{1}{n} \sum_{i=1}^{n} \left[r_{i+1} + \gamma \max_u \hat{Q}(x_{i+1}, u, \theta') - \hat{Q}(x_i, u_i, \theta_k) \right] \nabla_\theta \hat{Q}(x_i, u_i, \theta_k)$$

        every T_{target} steps: $\theta' \leftarrow \theta_{k+1}$

target network update

        $k = k + 1$
    Until the states are terminal
end for

Reinforcement Learning

Overview of RL

Approximate TD Learning for Prediction

Approximate TD Learning for Control

Deep Q-Networks

Perspectives

There are many alternative methods for approximate learning

- On-policy methods
- Policy gradient methods
- Actor-critic methods

Fundamental dilemma:

- Efficiency of DP: Models make it possible to plan and synthetize policy, otherwise we only rely on experience. Experiments are costly and risky.
- Efficiency of RL: Experiments allow to explore and improve exploitation on the long run. Models are inherently uncertain.
- How to have a working marriage of DP and RL?