dortmund university

Fakultät Bio- und Chemieingenieurwesen

Learning robust NMPC policies

Prof. Sergio Lucia
Chair of Process Automation Systems
Department of Biochemical and Chemical Engineering

Solving robust (N)MPC problems

- In general, difficult problem to solve online
- What can we do if very fast solutions are needed?

- A lot of this work is thanks to:

Benjamin Karg

Explicit model predictive control

Let's assume a simple MPC problem:

- Linear system dynamics
- Linear constraints
- Quadratic cost function

$$
\begin{aligned}
\underset{u_k, x_k}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) + V_f(x_N) \\
\text{subject to} \quad & x_{k+1} = f(x_k, u_k) \\
& g(x_k, u_k) \leq 0, \\
& x_0 = x_{\text{init}}, \\
& k \in [0, N-1]
\end{aligned}
$$

the optimal solution is a piecewise affine function of x_0

$$
u_0(x_0) = \begin{cases} K_1 x_0 + c_1 & \text{if } H_1 x_0 \leq h_1 \\ K_2 x_0 + c_2 & \text{if } H_2 x_0 \leq h_2 \\ \vdots & \vdots \\ K_M x_0 + c_M & \text{if } H_M x_0 \leq h_M \end{cases}
$$

[Bemporad et al., Automatica, 2002]

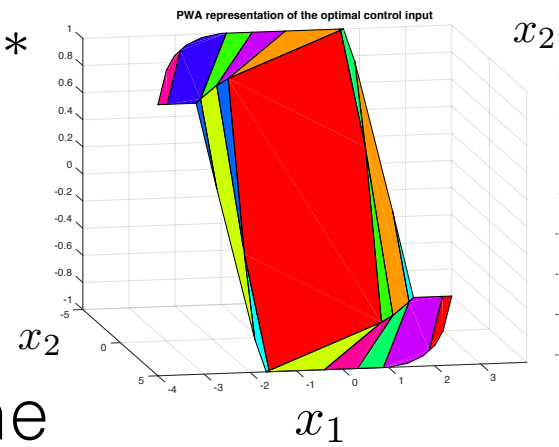
Example: explicit model predictive control

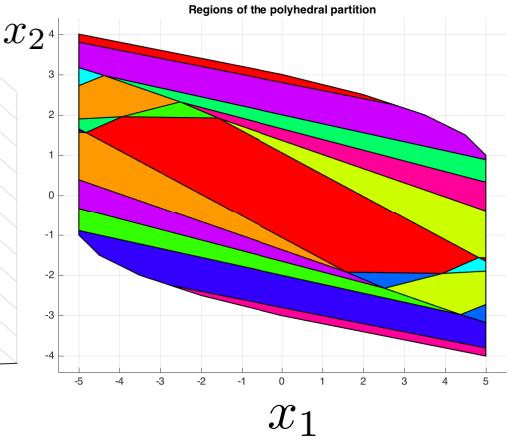
- Linear system with $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$
- Cost function and constraints $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad R = 1$

$$[-5, -5] \leq x_k \leq [5, 5]$$

$$-1 \leq u_k \leq 1$$

u^*





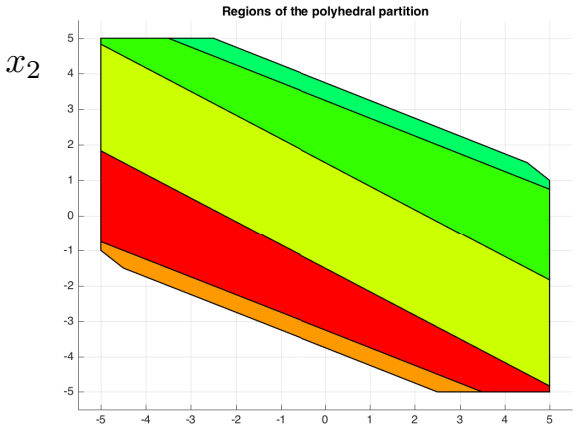
- **Compute and store all regions** offline
- Online cost reduces to search in which region the system

Explicit MPC in the linear case

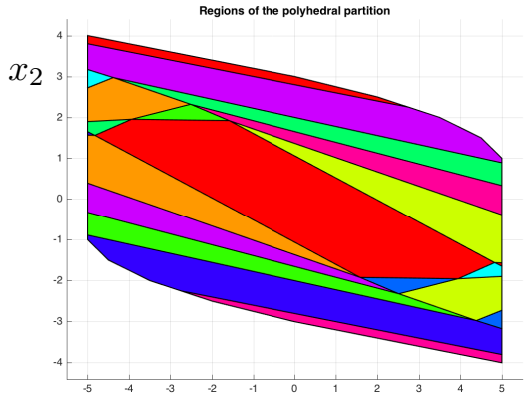
Can be extended to the robust case [Pistikopoulos et al., ADCHEM 2009]

- Advantages:
  - Applicable on embedded hardware
  - Low computational requirements
- Drawbacks:
  - Region evaluation
  - Memory footprint
  - Curse of dimensionality

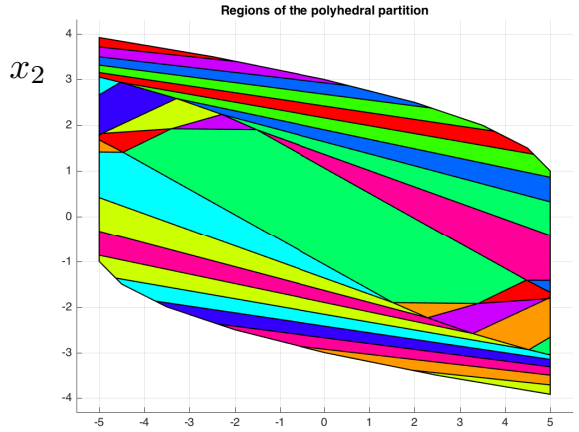
N = 2



N = 5



N = 10



Main idea

Most optimization-based decision-making problems are **parametric**

$$
\begin{aligned}
\underset{u_k, x_k}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) + V_f(x_N) \\
\text{subject to} \quad & x_{k+1} = f(x_k, u_k) \\
& g(x_k, u_k) \leq 0, \\
& x_0 = x_{\text{init}}, \qquad \text{system state} \\
& k \in [0, N-1]
\end{aligned}
$$

An implicit mapping $x_{\text{init}} \rightarrow u^*$ is defined by the optimization problem

- Is it possible to learn this mapping using machine learning techniques?

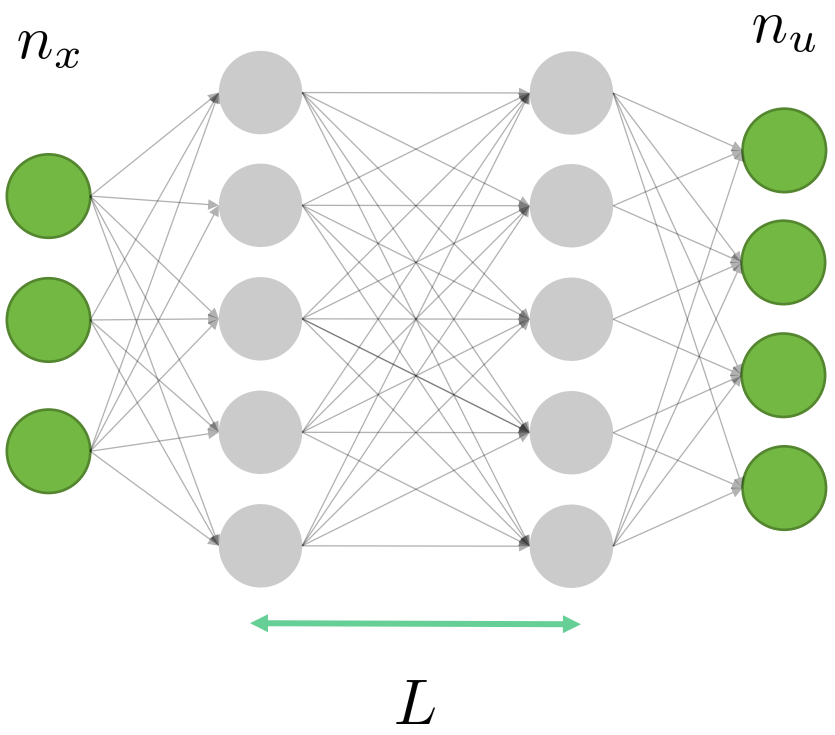
Using neural networks to approximate MPC

It is an old idea: already done in 1995 for nonlinear MPC

- What is new?

Common practice until recent successes in deep learning was:

- Because of **universal approximation theorem**: use only 1 layer



[T. Parisini and R. Zoppoli, 1995, Akesson and Toivonen, 2006]

Deep neural networks (DNN)

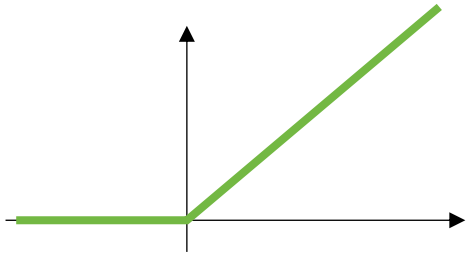
Neural network with L **hidden layers** and M neurons per layer

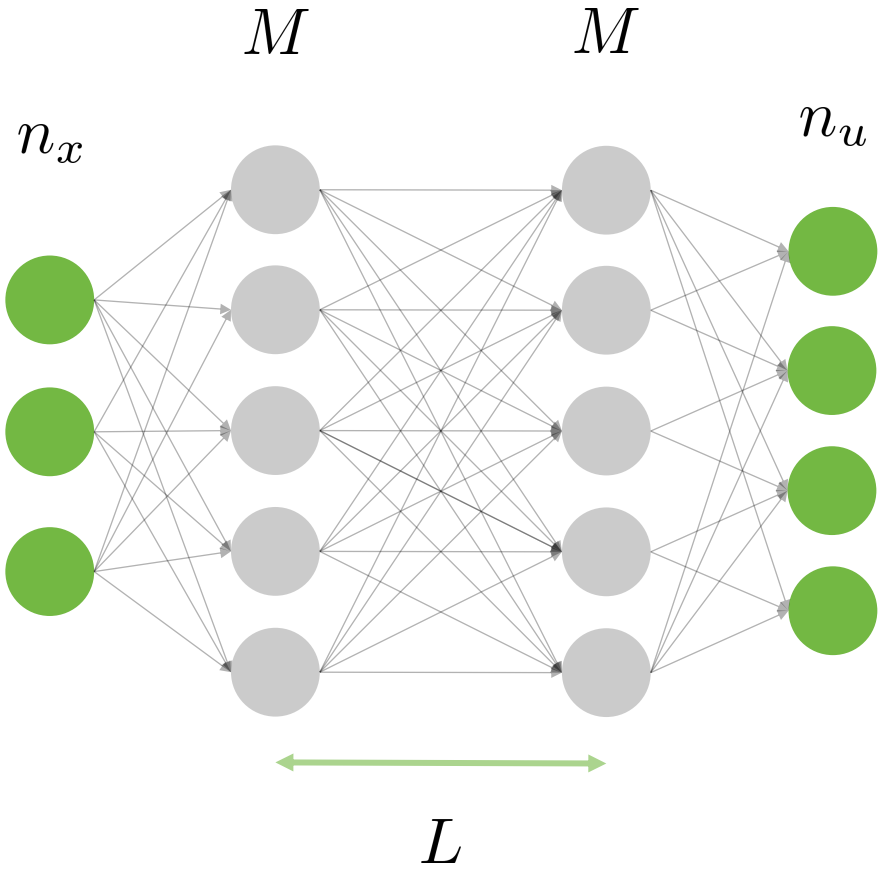
$$\mathcal{N}(x;\theta,M,L) = f_{L+1} \circ g_L \circ f_L \circ \cdots \circ g_1 \circ f_1(x)$$

- Affine transformation $\theta_l = \{W_l, b_l\}$

  $f_l(x_{l-1}) = W_l x_{l-1} + b_l$

- Activation function
  - tanh: $g_l(f_l) = \tanh(f_l) = \dfrac{e^{f_l} - e^{-f_l}}{e^{f_l} - e^{-f_l}}$
  - ReLU: $g_l(f_l) = \max(0, f_l)$

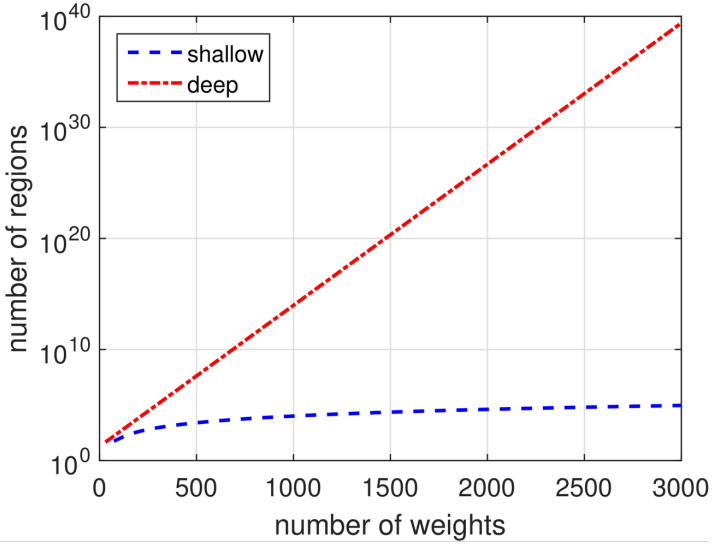




Why deep neural networks to approximate MPC?

- One of the main questions in machine learning
- An intuitition why more hidden layers can help:
  - The number of linear regions represented by a network grows exponentially with the number of layers [Montufar 2014], [Chen et al, 2018; Karg and Lucia, 2018]

$$n_{\mathrm{r}} = \left(\prod_{l=1}^{L-1} \left\lfloor \frac{M}{n_x} \right\rfloor^{n_x} \right) \sum_{j=0}^{n_x} \binom{L}{j}$$



- It is also possible to compute the number of neurons and layers necessary to exactly represent the solution of an MPC problem

[Karg and Lucia, IEEE Transactions on Cybernetics, 2020]

Theorem 1. *There always exist parameters $\theta_{\gamma,i}$ and $\theta_{\eta,i}$ for $2n_u$ deep ReLU neural networks with depth $r_{\gamma,i}$ and $r_{\eta,i}$ for $i = 1, \ldots, n_u$ and width $M = n_x + 1$, such that the vector of neural networks defined by*

$$\begin{bmatrix} \mathcal{N}(x;\theta_{\gamma,1},M,r_{\gamma,1}) - \mathcal{N}(x;\theta_{\eta,1},M,r_{\eta,1}) \\ \vdots \\ \mathcal{N}(x;\theta_{\gamma,n_u},M,r_{\gamma,n_u}) - \mathcal{N}(x;\theta_{\eta,n_u},M,r_{\eta,n_u}) \end{bmatrix} \quad (15)$$

can exactly represent an explicit MPC law $\mathcal{K}(x) : [0,1]^{n_x} \to \mathbb{R}^{+n_u}$.

Exact representation

It is possible to exactly represent a linear MPC law with a ReLU neural network

- Divide the PWA function in a difference of two convex PWA functions with r_γ, r_η regions [Hempel, Goulart, Lygeros, ECC, 2013]

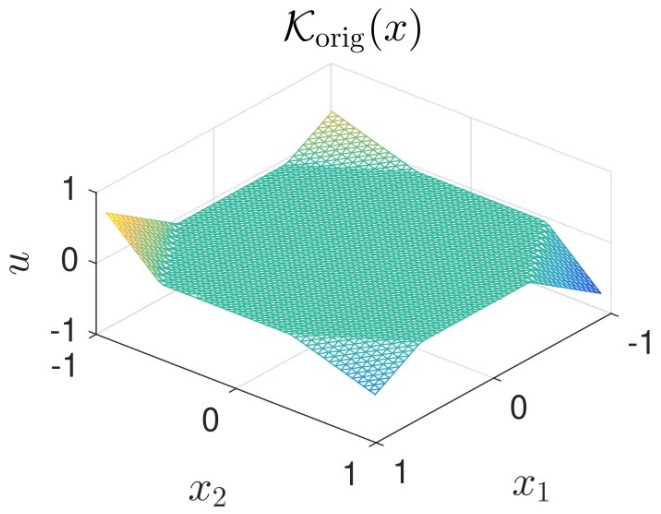
$$f(x) = \gamma(x) - \eta(x)$$

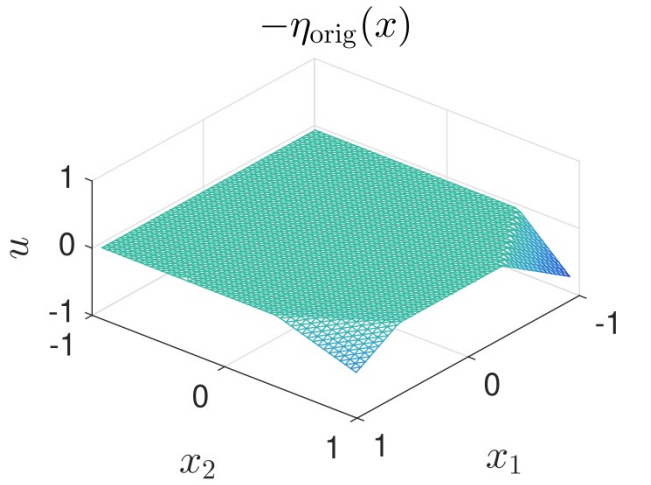
- Each convex PWA function can be represented by the difference of two ReLU network of width $M = n_x + 1$ and depth r_γ, r_η

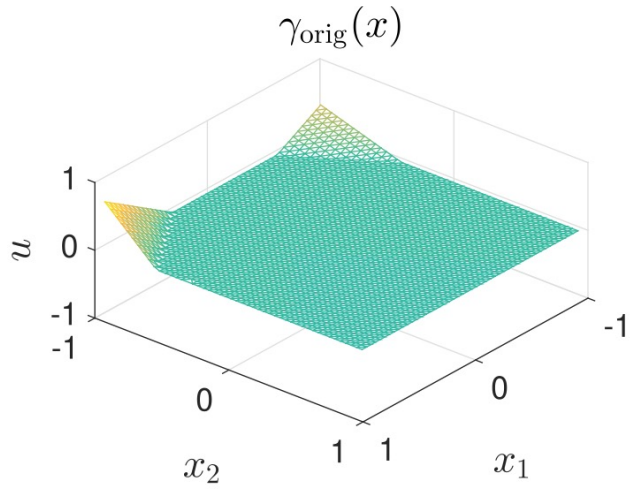
Main problem: in general, still exponential growth of r_γ, r_η

[Karg and Lucia, IEEE Transactions on Cybernetics, 2020]

Exact representation: Example

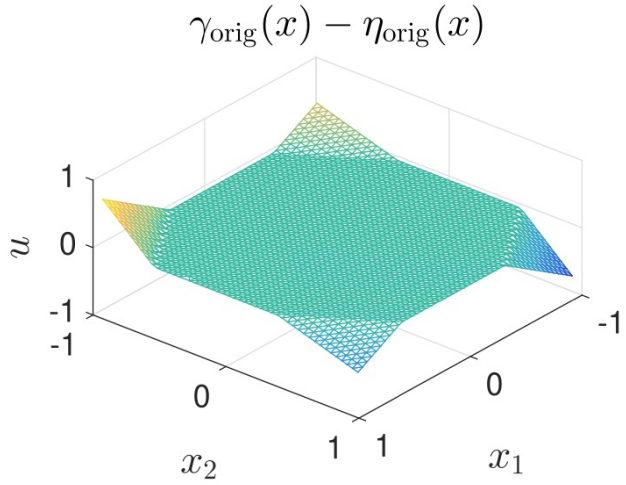






[Karg and Lucia, IEEE Transactions on Cybernetics, 2020]

Exact representation via NNs



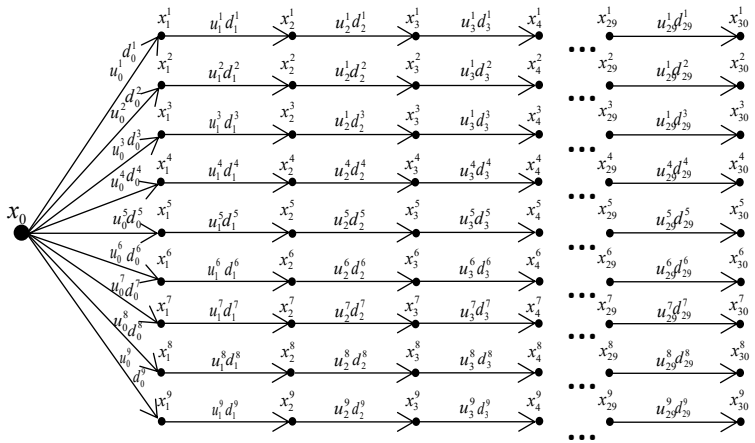
3 layers each,
3 neurons per layer

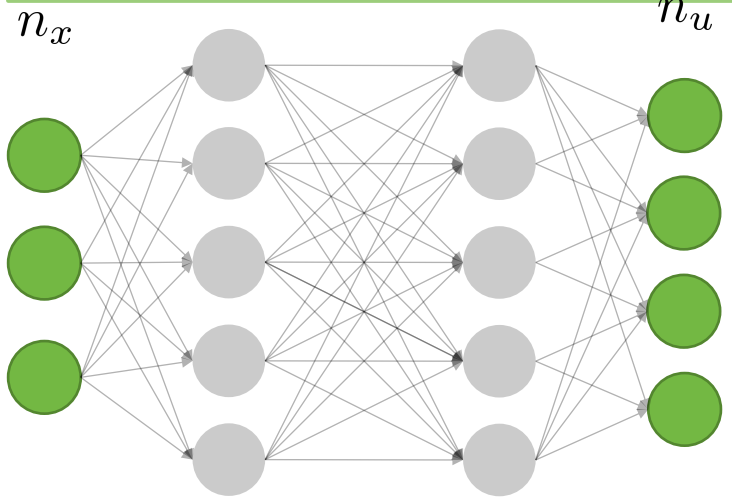
Approximating a complex controller

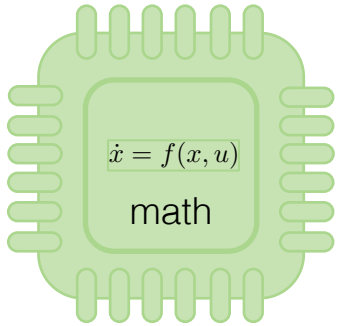
1: Generate training samples by solving many MPC problems

(x_{init}, u_0^*)

2: Offline training of the deep neural network

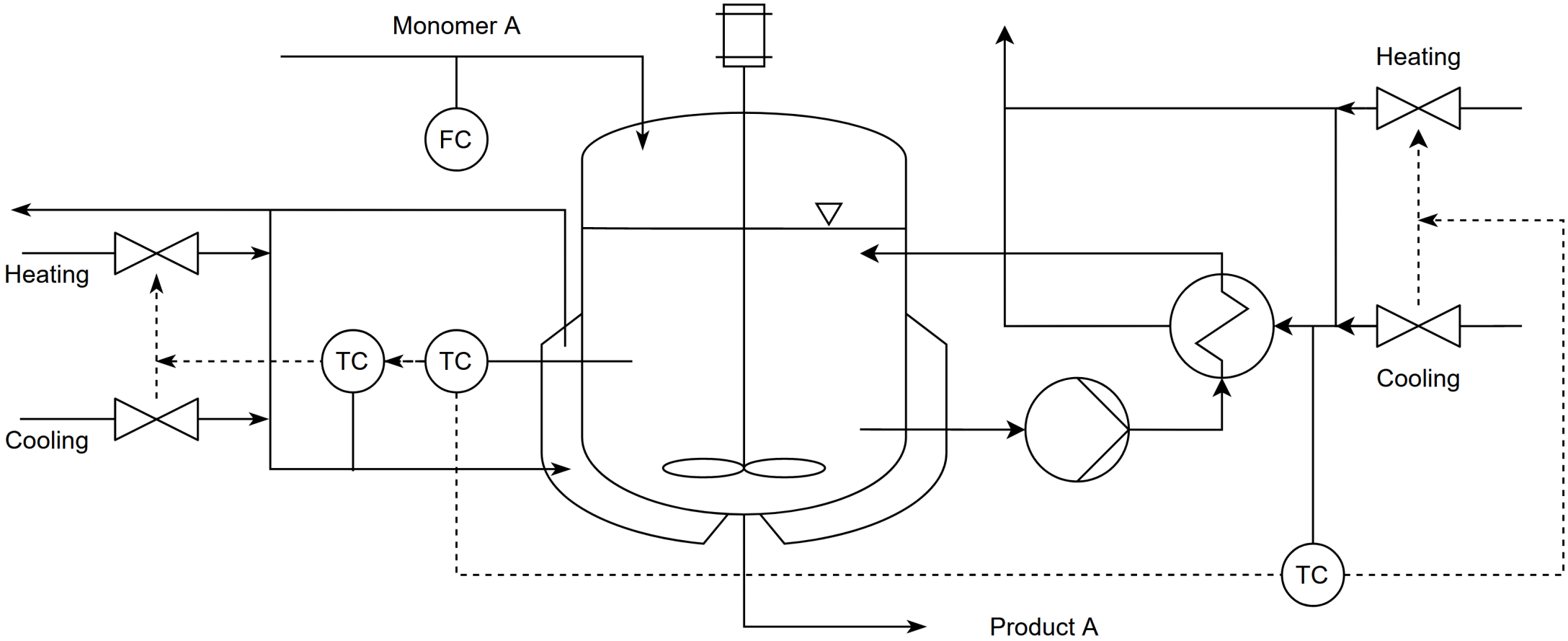






3: High performance control even on simple hardware

Case-study: Industrial polymerization reactor



[Lucia, Andersson, Brandt, Diehl and Engell, *Journal of Process Control, 2014*]

An industrial polymerization reactor

8 differential states
3 control inputs
2 uncertain parameters

$$\dot{m}_W = \dot{m}_{W,F}$$

$$\dot{m}_A = \dot{m}_{A,F} - k_{R1} m_{A,R} - \frac{p_1 k_{R2} m_{AWT} m_A}{m_{ges}}$$

$$\dot{m}_P = k_{R1} m_{A,R} + \frac{p_1 k_{R2} m_{AWT} m_A}{m_{ges}}$$

$$\dot{T}_R = \frac{1}{c_{p,R} m_{ges}} \left[\dot{m}_F c_{p,F}(T_F - T_R) + \Delta H_R k_{R1} m_{A,R} - k_K A(T_R - T_S) - \dot{m}_{AWT} c_{p,R}(T_R - T_{EK})\right]$$

$$\dot{T}_S = 1/(c_{p,S} m_S)[k_K A(T_R - T_S) - k_K A(T_S - T_M)]$$

$$\dot{T}_M = \frac{1}{c_{p,W} m_{M,KW}} \left[\dot{m}_{M,KW} c_{pW}(T_M^{IN} - T_M) + k_K A(T_S - T_M)\right]$$

$$\dot{T}_{EK} = \frac{1}{c_{p,R} m_{AWT}} \left[\dot{m}_{AWT} c_{p,W}(T_R - T_{EK}) - \alpha(T_{EK} - T_{AWT}) + \frac{p_1 k_{R2} m_A m_{AWT} \Delta \mathrm{H}_R}{m_{ges}}\right]$$

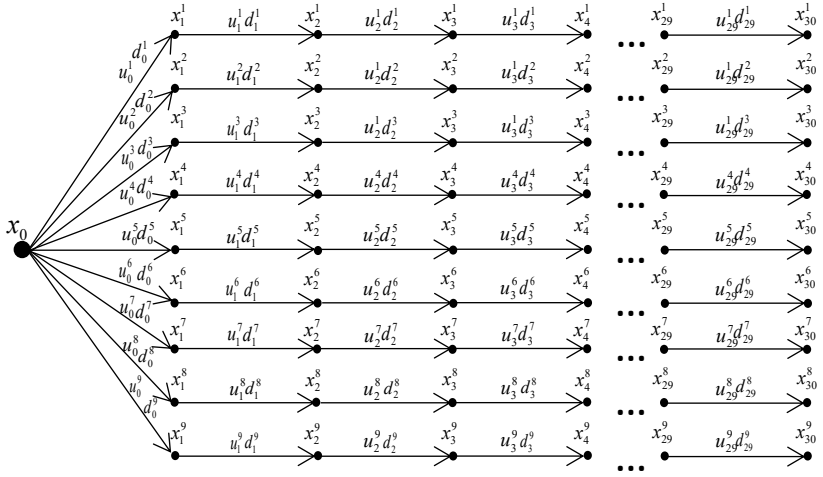
$$\dot{T}_{AWT} = \frac{1}{c_{p,W} m_{AWT,KW}} \left[\dot{m}_{AWT,KW} c_{p,W}(T_{AWT}^{IN} - T_{AWT}) - \alpha(T_{AWT} - T_{EK})\right]$$

$$k_{R1} = k_0 e^{-\frac{E_a}{R T_R}} (k_{U1}(1 - U) + k_{U2} U)$$

$$k_{R2} = k_0 e^{-\frac{E_a}{R T_{EK}}} (k_{U1}(1 - U) + k_{U2} U)$$

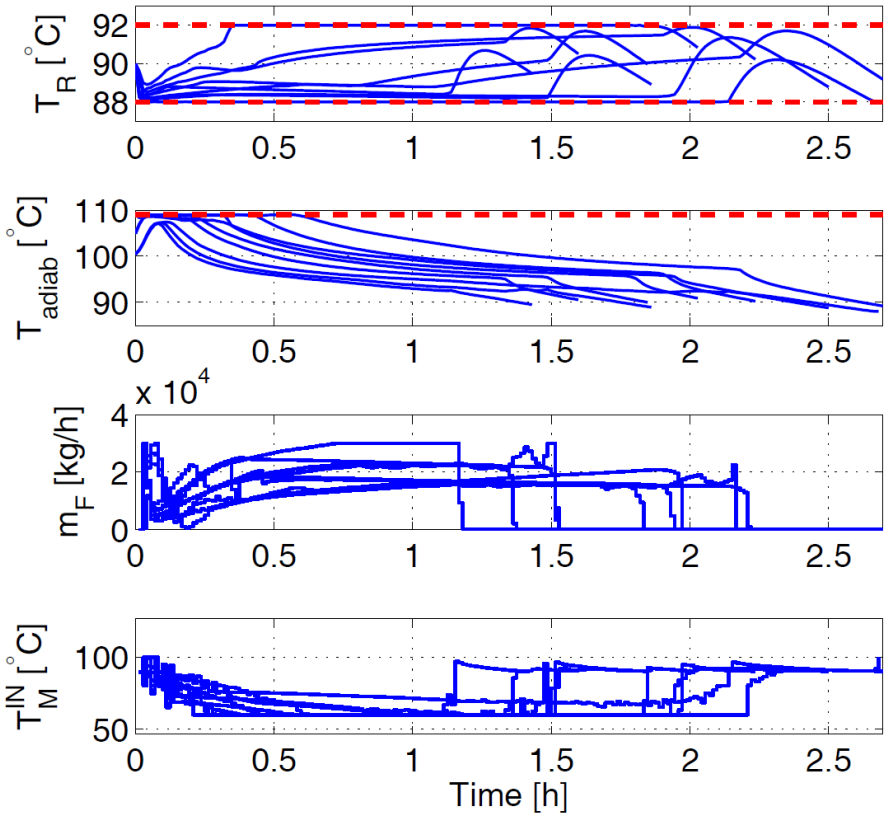
Simulation results for multi-stage NMPC

- A robust MPC is possible by considering different values of critical uncertainty parameters in a simple scenario tree



- At the cost of increased computational cost

Multi-stage NMPC



Can we learn such controller?

Training

Samples generated solving multi-stage NMPC (CasADi + IPOPT)

100 batches of data (with random initial cond. and uncertain param.)

- 50 s sampling time
- Total of 21050 samples

- Simple training with pytorch

$$\min_{W_l, b_l} \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} ||\hat{u}(x_{\text{tr},i}) - u^*(x_{\text{tr},i})||^2$$

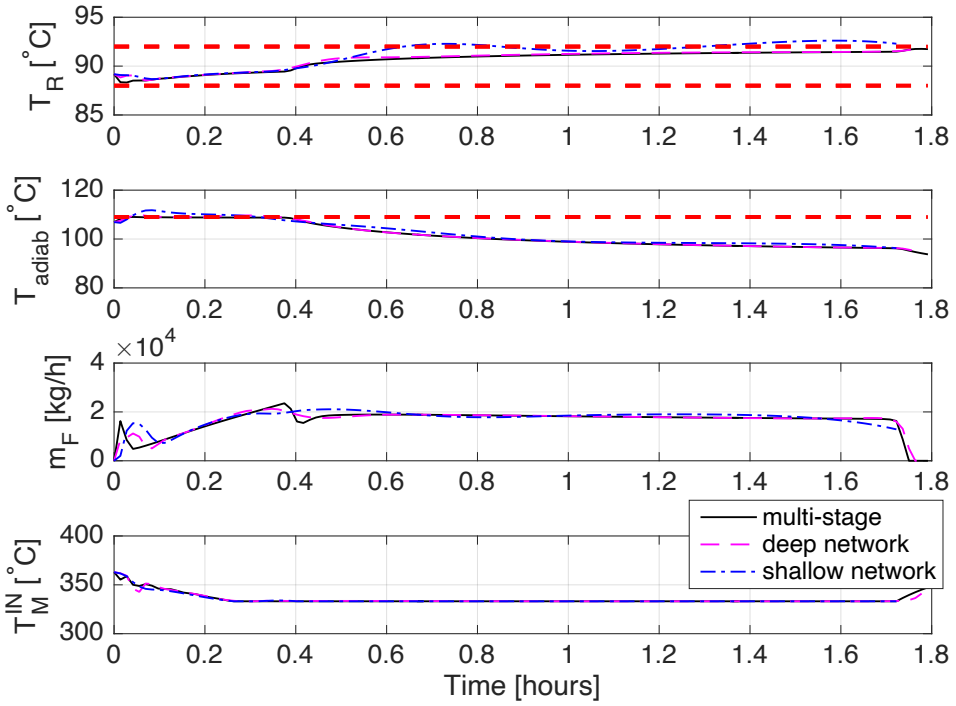
Output neural network

Output multi-stage NMPC

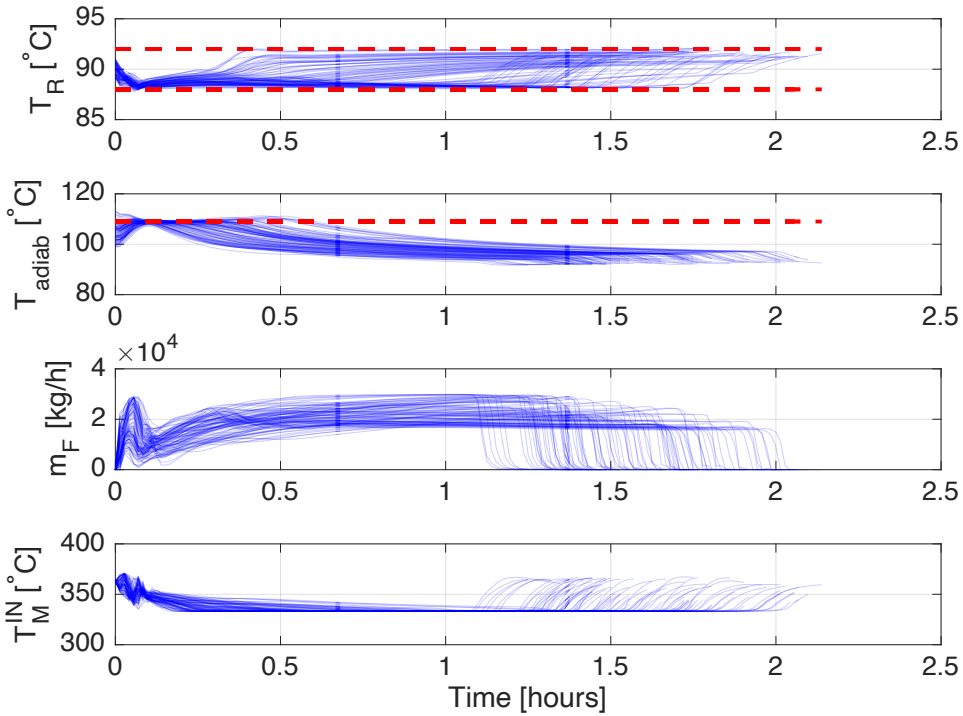
n_{l}	n_{n}	n_{tot}	n_{w}	MSE_{train}	MSE_{test}
1	90	90	1263	0.0043	0.0046
2	45	90	2703	0.0024	0.0024
6	15	90	1413	0.0015	0.0014
9	10	90	1023	0.0014	0.0014

Performance of deep-learning based ms-NMPC

Exact vs. deep vs. shallow multi-stage NMPC



Deep-learning based multi-stage NMPC



Average performance over random 100 batches

Algorithm	Batch time [h]	Cons. Viol. [°C/h]
Exact	1.6459	0.0058
Shallow	1.7328	0.3087
Deep	1.6297	0.0549

[Lucia and Karg, NMPC 2018]

A batch polymerization reactor

Enable low-cost emb. implementation

- 32 bit ARM Cortex M0+
- 48 MHz with 32 kB RAM

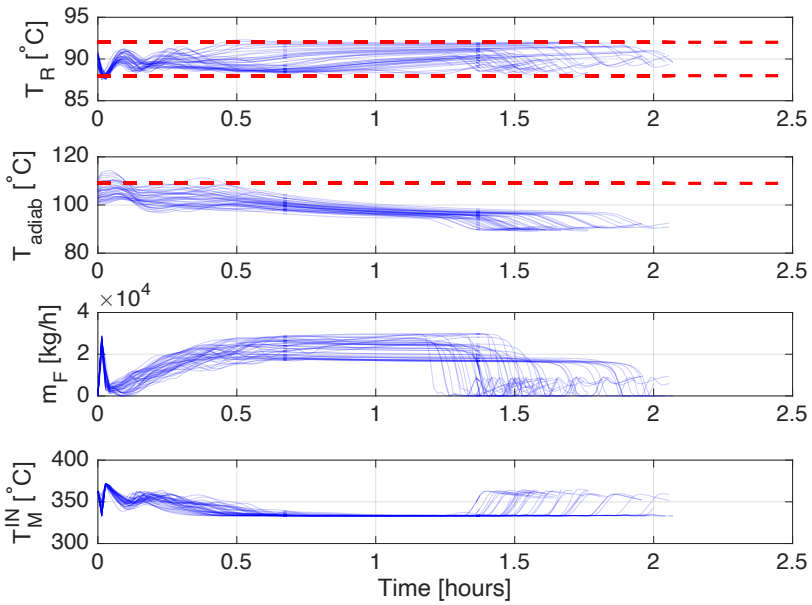
Approx. robust NMPC:

- Memory footprint: 27 kB
- Evaluation time on a uC: 37 ms
- Trivial code-generation (PLC, uC)

Enable large(r)-scale problems

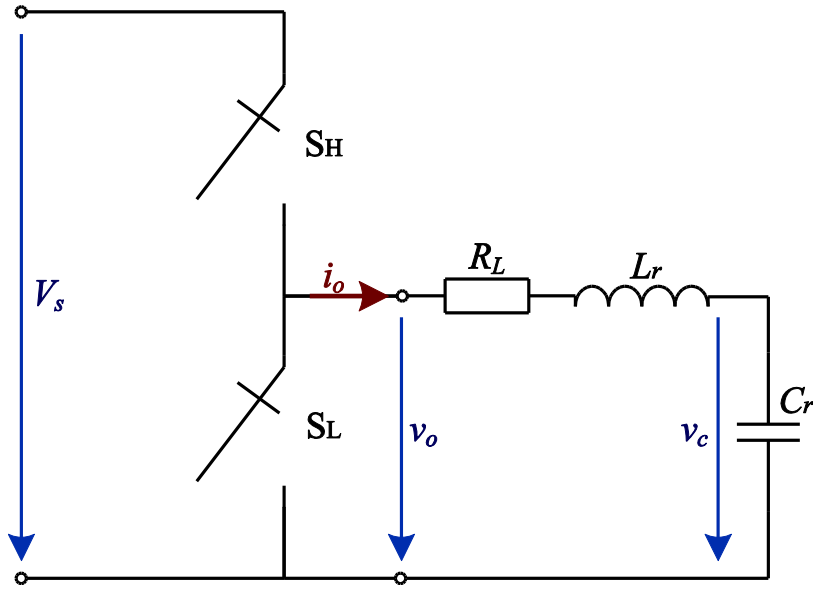
Problem with 5 uncertainties

- 243 scenarios
- ~115,000 variables and constraints



Robust nonlinear optimal control in microseconds

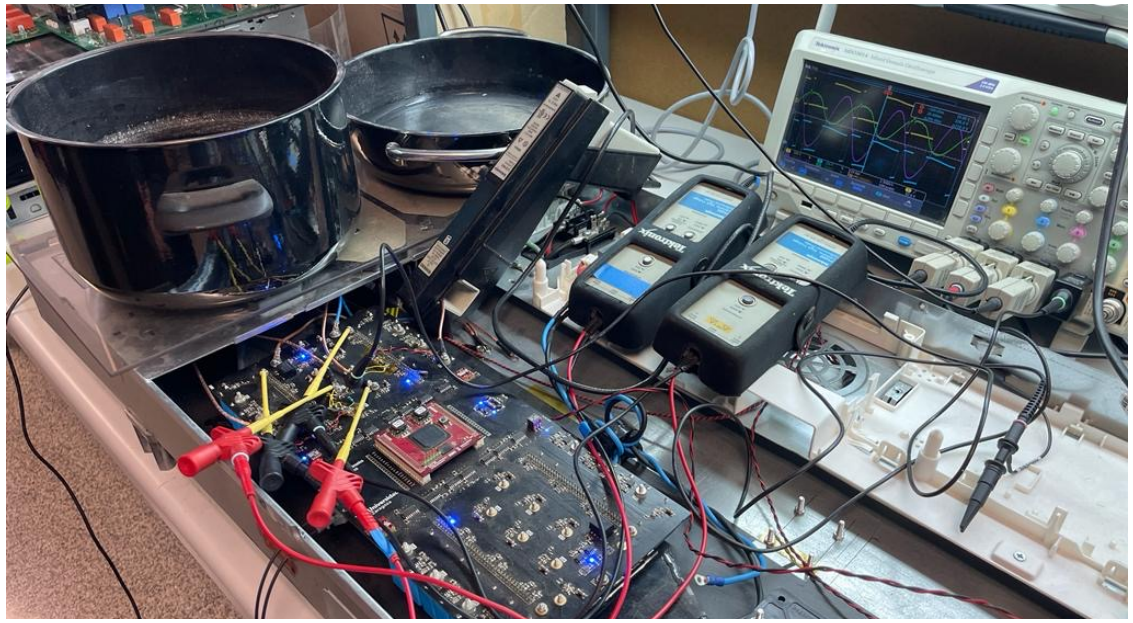
Induction heating is currently used in many industrial and domestic applications



Control switching frequency and duty cycle. Satisfy constraints under uncertainty

Real implementation

Approximate robust NMPC in 1 μs (on an FPGA)



[S. Lucia et al., IEEE Transactions on Industrial Informatics, 2020]

[P. Guillen et al., IEEE Access, 2022]

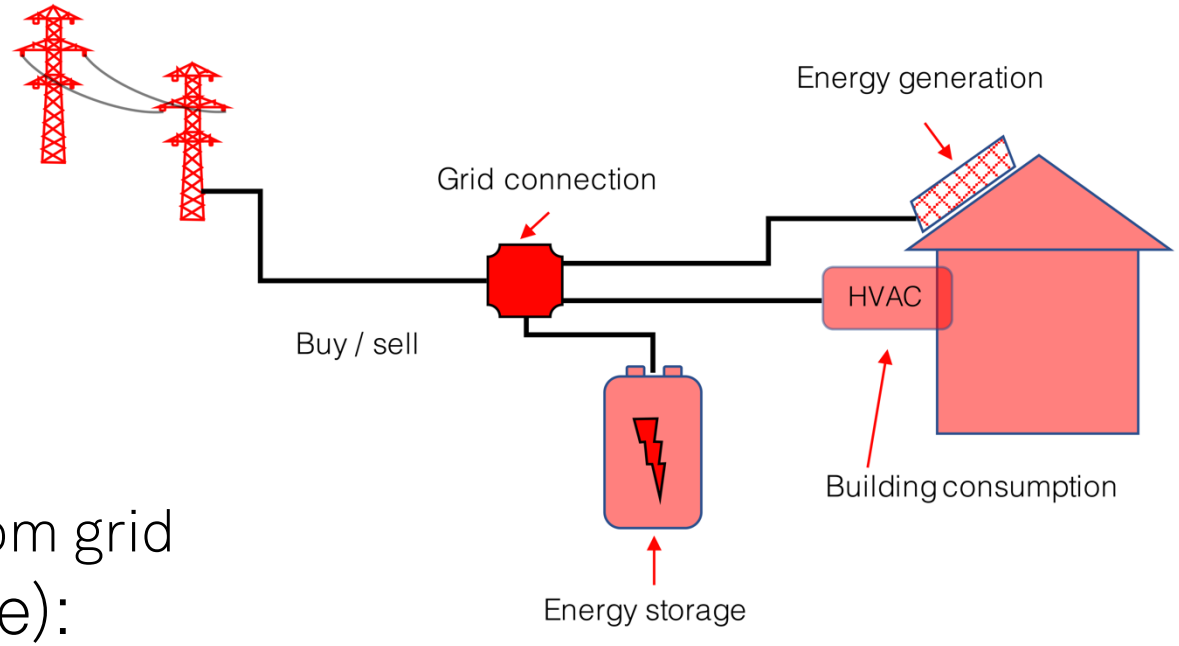
Universidad Zaragoza

1542

Energy management system

The **goal** is to manage the room temperature with **minimum cost**

- States:
  - Room temperature
  - Wall temperatures
  - State of charge of the battery
- Inputs:
  - Grid power
  - Battery power
  - HVAC power
  - **Integer variable** to model buy/sell from grid
- Disturbances (predictions available):
  - Solar radiation
  - External temperature
  - Internal gains



Energy management system

The parametric MPC problem **has 76 parameters:**

- 4 for the states
- 72 for the disturbances trajectories with a prediction horizon of 24

Very **challenging for traditional explicit MPC** approaches

$$
\begin{aligned}
\underset{(x,u)}{\text{minimize}} \quad & \sum_{k=0}^{N-1} (P_{\text{grid}}^k + \gamma (E_{\text{bat}}^k - E_{\text{bat}}^{\text{ref}})^2) \\
\text{subject to} \quad & x_{k+1} = Ax_k + Bu_k + Ed_k, \\
& x_{\text{lb}} \leq x_k \leq x_{\text{ub}}, \\
& u_{\text{lb}} \leq u_k \leq u_{\text{ub}}, \\
& \alpha \in \{0, 1\}, \\
& m_{\text{lb}} \leq Du_k + Gd_k \leq m_{\text{ub}}.
\end{aligned}
$$

$$
A = \begin{bmatrix} 0..8511 & 0.0541 & 0.0707 & 0 \\ 0.1293 & 0.8635 & 0.0055 & 0 \\ 0.0989 & 0.0003 & 0.0002 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0035 \\ 0.0003 \\ 0.0002 \\ -5 \end{bmatrix},
$$

$$
E = 10^{-3} \begin{bmatrix} 22.217 & 1.7912 & 42.212 \\ 1.5376 & 0.6944 & 2.9214 \\ 103.18 & 0.1032 & 196.04 \\ 0 & 0 & 0 \end{bmatrix}
$$

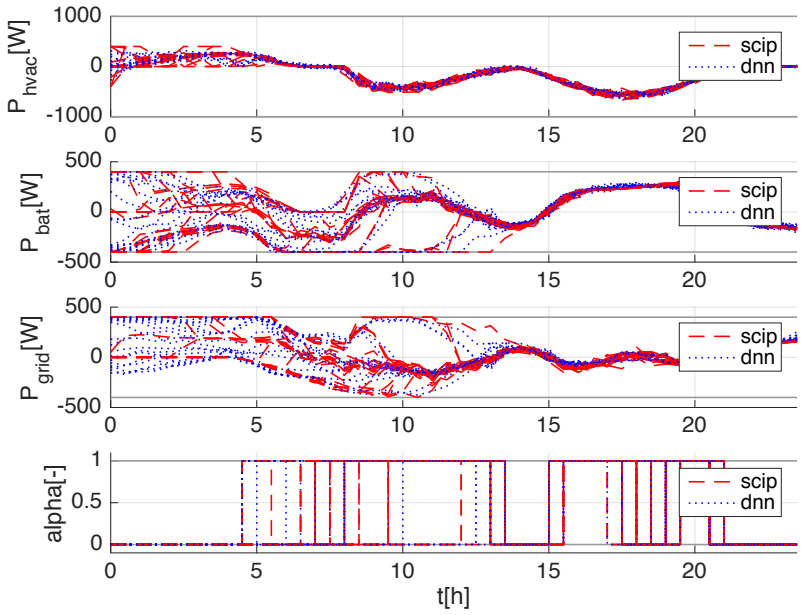
$$
x = [T_{\text{r}}, T_{\text{w,i}}, T_{\text{w,e}}, E_{\text{bat}}]^T
$$

$$
u = [P_{\text{grid}}, P_{\text{hvac}}, P_{\text{bat}}, \alpha]^T
$$

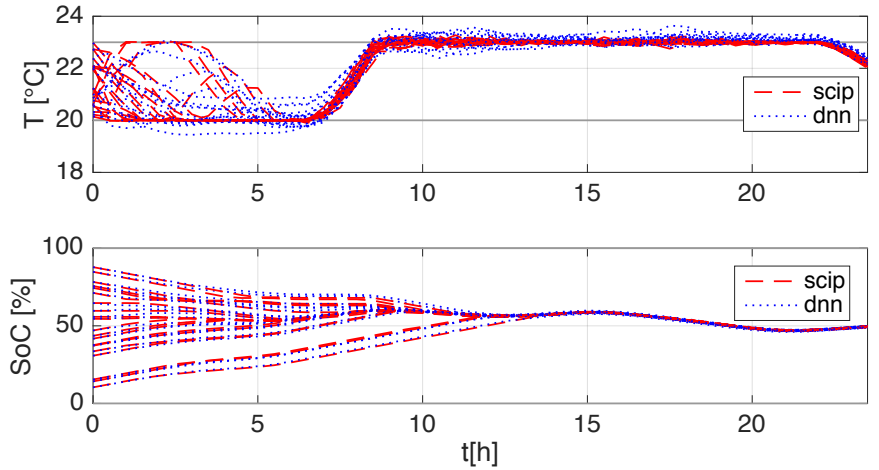
$$
d = [d_{\text{amb}}, d_{\text{sol}}, d_{\text{int}}]^T
$$

DNN controller vs. exact MPC

Optimal inputs computed every 20 minutes



State trajectories



[Karg and Lucia, ECC 2018]

Exact solution →

Deep-learning based solution →

Controller	Average integrated cost	Average violations
SCIP	72.64	0
DNN	73.32	0.11

Many other works and extensions possible

- Large-scale QP approximations [Kumar, Rawlings, Wright, CACE 2021]
- Diffussion-based approximate MPC [Marquez Julbe et al., IROS 2025]
- …

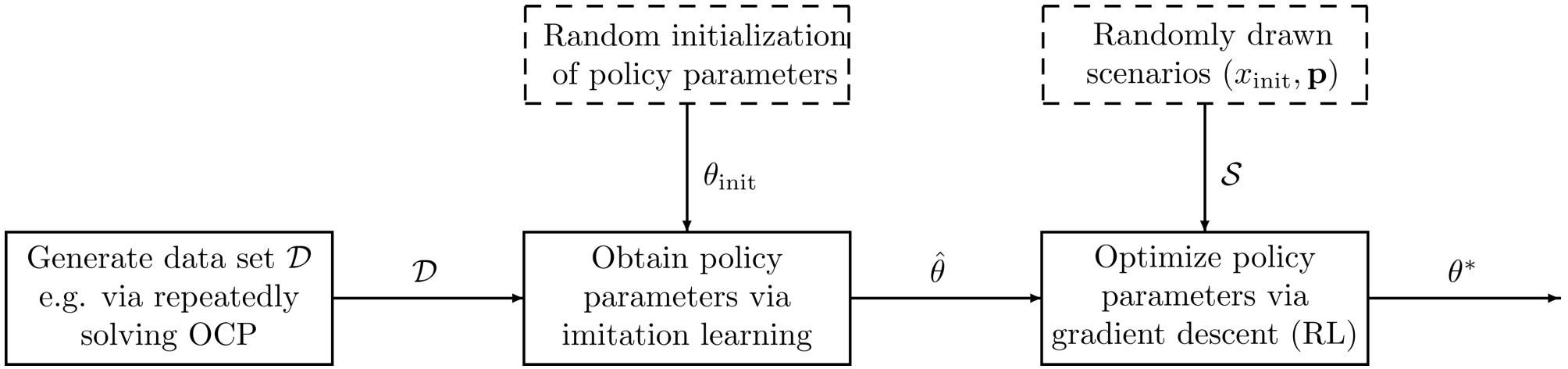
And especially many works dealing with guarantees…

Can we improve approximate controllers once trained?

Improving performance after training

Assuming that a **detailed simulator is available**, is it possible to adapt the NN controller to consider:

- Changes in the model or optimal control problem formulation
- Non-considered effects in the MPC model, or estimation errors…



[Karg and Lucia, *Int. Conference in Process Control*, 2021, **Best Young Author Paper Award**]

Reinforcing a parametric control law

- Closed-loop trajectory is a parametric function:

$$\begin{aligned} x_{k+1} &= f_{\mathrm{hf}}(x_k, \kappa(x_k;\theta), p_k), \\ &= f_{\mathrm{hf,cl}}(x_k, p_k;\theta), \end{aligned}$$

- Closed-loop performance index:

$$\mathcal{P}_{\mathrm{cl}}(x_{\mathrm{init}}, \mathbf{p}; N, \theta) = \mathcal{P}(\mathbf{x}, \kappa(\mathbf{x};\theta), \mathbf{p}; N),$$

- Control problem in closed-loop form:

$$\begin{aligned} \arg\min_{\theta} \quad & \mathbb{E}\left[\mathcal{P}_{\mathrm{cl}}(x_{\mathrm{init}}, \mathbf{p}; N, \theta)\right] \\ \text{subject to:} \quad & 0 \geq g_{\mathrm{cl}}(x_{\mathrm{init}}, \mathbf{p}; N, \theta) \end{aligned}$$

Reinforcing a parametric control law

The constraints can be included in the closed-loop performance index:

$$\mathcal{P}_{\mathrm{cl,aug}}(x_{\mathrm{init}}, \mathbf{p}; N, \theta) = \mathcal{P}_{\mathrm{cl}}(x_{\mathrm{init}}, \mathbf{p}; N, \theta) + w^T \max(0, g_{\mathrm{cl}}\left(x_{\mathrm{init}}, \mathbf{p}; N, \theta\right))$$

Draw random set of scenarios w.r.t. to probability distribution:

$$\mathcal{S} = \{(x_{\mathrm{init},1}, \mathbf{p}_1), \ldots, (x_{\mathrm{init},|\mathcal{S}|}, \mathbf{p}_{|\mathcal{S}|})\}$$

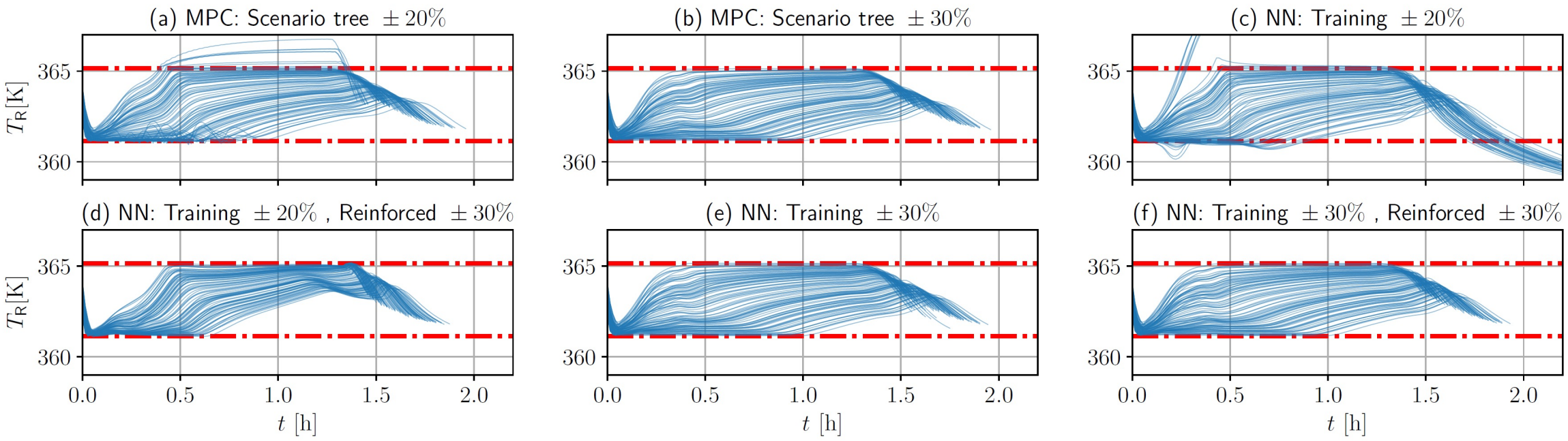
Optimize the policy parameters:

$$\theta \leftarrow \theta - \frac{\alpha}{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} \nabla_\theta \mathcal{P}_{\mathrm{cl,aug}}(x_{\mathrm{init},j}, \mathbf{p}_j; N, \theta)),$$

Reinforced approximate predictive control

Controller trained with uncertain parameters +-20 %

- Real system has uncertain parameters at +- 30%



[Karg and Lucia, *Int. Conference in Process Control*, 2021]

How to do **safe** approximate NMPC?

Safe performance with NN controllers

In general, only an approximate solution is obtained and thus there are **no guarantees** about constraint satisfaction or stability

Many research groups have worked on these issues in recent years

Two main possibilities to achieve guarantees:

- Deterministic guarantees
- Probabilistic guarantees

Deterministic guarantees available mostly for linear systems only

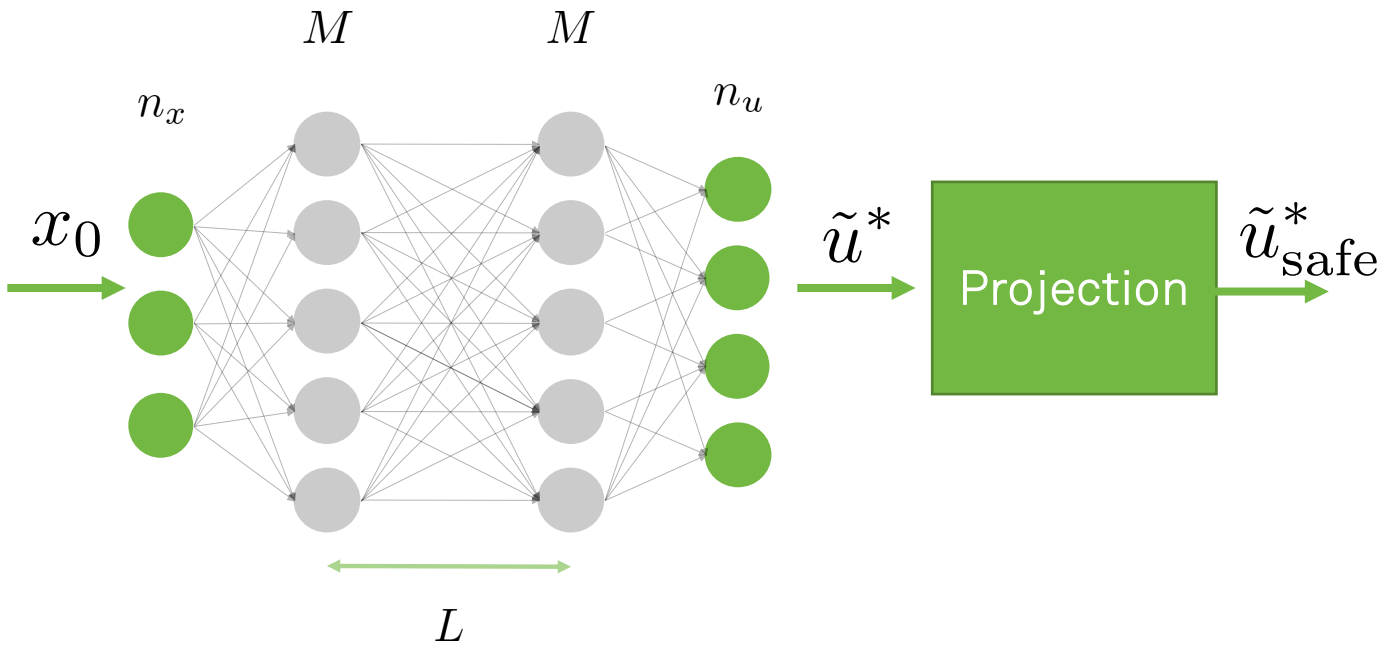
Deterministic guarantees with NN controllers

Linear systems:

- Projection-based methods [Chen et al., 2018, Karg & Lucia, 2020]
- Optimization layers [Maddalena et al., 2020]
- Worst-case approximation errors and Lipschitz constants [Fabiani and Goulart, TAC 2023], [Schwan, Jones and Kuhn, TAC 2023]
- A-priori verification via output-range analysis [Karg & Lucia, 2020]

Deterministic guarantees: Projection

- Projection onto a feasible reagion

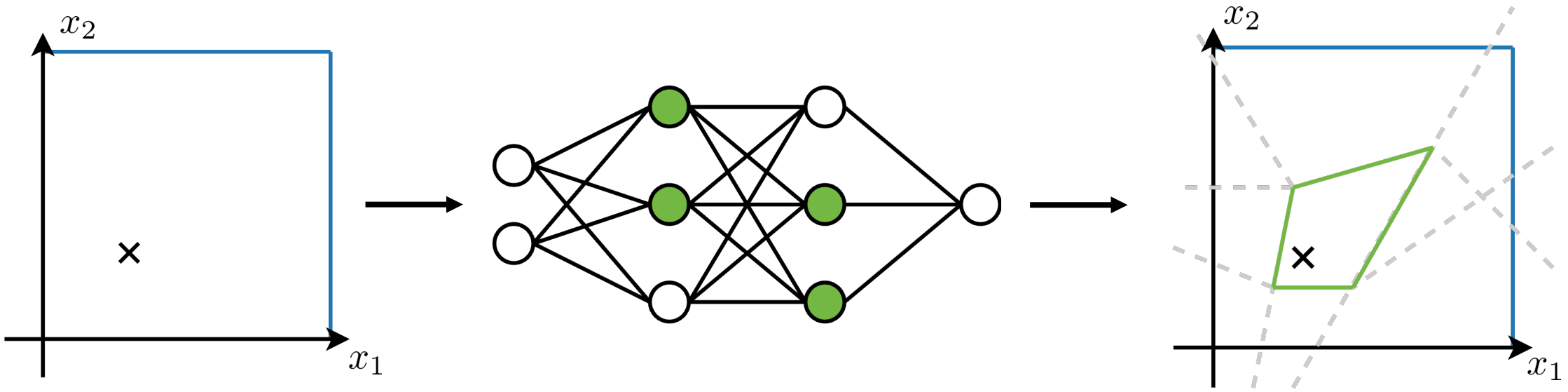


$$\begin{aligned} \underset{\hat{u}}{\text{minimize}} \quad & \|\mathcal{N}(x;\theta,M,L) - \hat{u}\|_2^2 \\ \text{subject to} \quad & C_{\text{inv}}(Ax_{\text{init}} + B\hat{u}) \leq c_{\text{inv}}, \\ & C_u\hat{u} \leq c_u, \end{aligned}$$

[Chen et al., ACC 2018, Karg and Lucia IEEE Tran. Cyb. 2020]

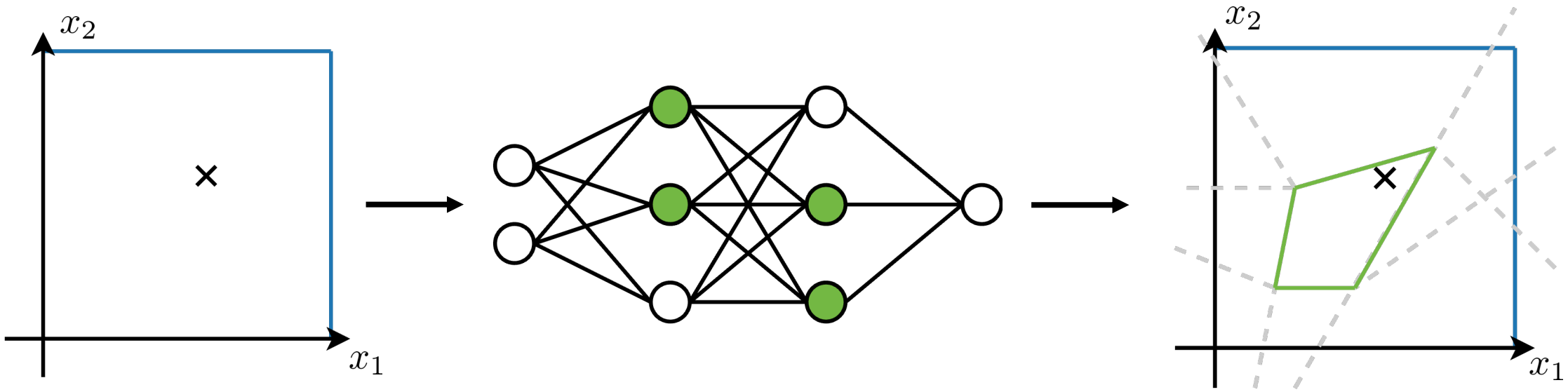
Activation pattern

Each activation pattern defines a region within the state space with an affine state feedback



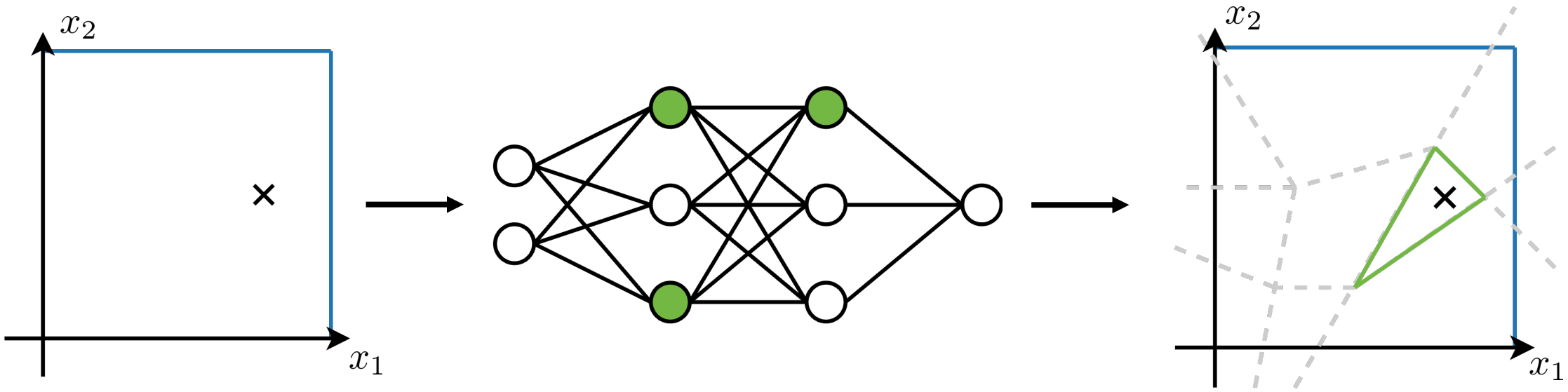
Activation pattern

Each activation pattern defines a region within the state space with an affine state feedback



Activation pattern

Each activation pattern defines a region within the state space with an affine state feedback



Activation region

Activation pattern corresponding to a state x:

$$G(x) := \{\beta \circ f_l(\xi_{l-1}) \in \{0,1\}^{n_l} \,|\, l \in [L], \xi_0 = x\}$$

With

$$\beta \circ f_l(\xi_{l-1})^{(i)} = \begin{cases} 1 & \text{if } W_l^{(i)} \xi_{l-1} + b_l^{(i)} \geq 0, \\ 0 & \text{else.} \end{cases}$$

Region defined by $\Gamma_i = G(x)$:

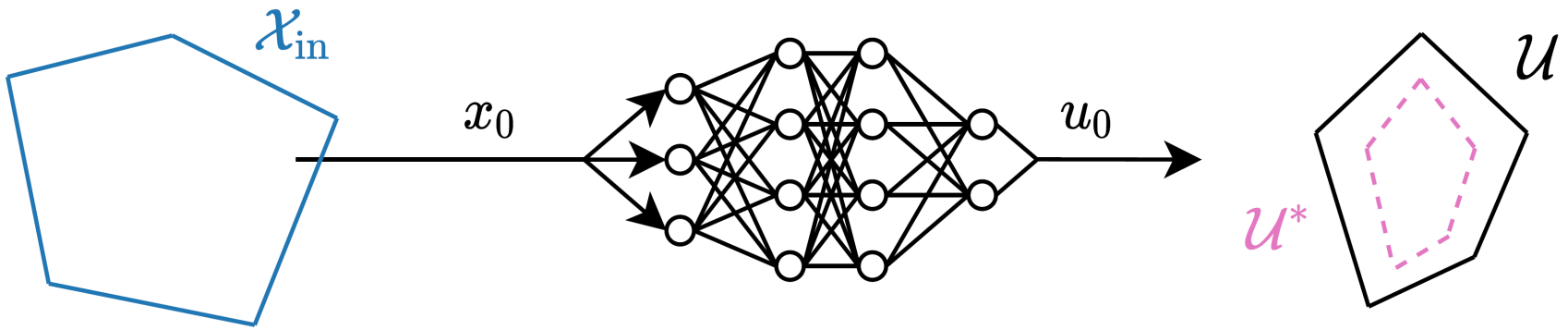
$$\mathcal{R}_{\Gamma_i} = \{x \in \mathbb{R}^{n_x} \,|\, \Gamma_i = G(x)\}, \text{ or } \mathcal{R}_{\Gamma_i} = \{x \in \mathbb{R}^{n_x} \,|\, F_i x \leq g_i\}$$

Activation feedback

Parametric description of neural network based on $\Gamma_i = G(x)$:

$$\mathcal{P}(x, \Gamma_i, l) = \begin{cases} f_l \circ \gamma_{i,l-1} \odot f_{l-1} \circ \cdots \circ \gamma_{i,1} \odot f_1(x), & \text{if } l = L+1, \\ \gamma_{i,l} \odot f_l \circ \cdots \circ \gamma_{i,1} \odot f_1(x), & \text{else.} \end{cases}$$
$$= W_{\Gamma_i,l} x + b_{\Gamma_i,l}$$

Output-range analysis (via MILPs)



- Compute (an over-approximation $\mathcal{U}^*$ of) the output set $\mathcal{U}_{\text{true}} = \mathcal{N}(\mathcal{X})$ of a neural network for a given input set $\mathcal{X}_{\text{in}}$
- Check if $\mathcal{U}^*$ satisfies desired properties, e.g. $\mathcal{U}^* \subseteq \mathcal{U}$

$$\mathcal{U} = \{u \in \mathbb{R}^{n_u} | C_u u \leq c_u\}$$

Modeling logical constraints in optimization

- To model a RELU network via optimization it is necessary to model a logical constraint for the activation pattern
- Big-M formulations are one possibility

Imagine you want to model as constraint:

either $a^T x \leq b$ or $c^T x \leq d$

This can be modelled with an additional integer opt. variable

$$
\begin{aligned}
a^T x &\leq b + My \\
c^T x &\leq d + M(1-y) \\
y &\in \{0,1\}
\end{aligned}
$$

Input constraints

For each direction $C_u^{(i)}$:

- $z = [z_0, \ldots, z_L]$:
  Output of layers
- $t = [t_1, \ldots, t_L]$:
  Activation hidden layers
- u_0 :
  Largest control input in the direction of the normal of the hyperplane
- x_0 :
  Feasible state leading to largest control input
- $M \in \mathbb{R}$

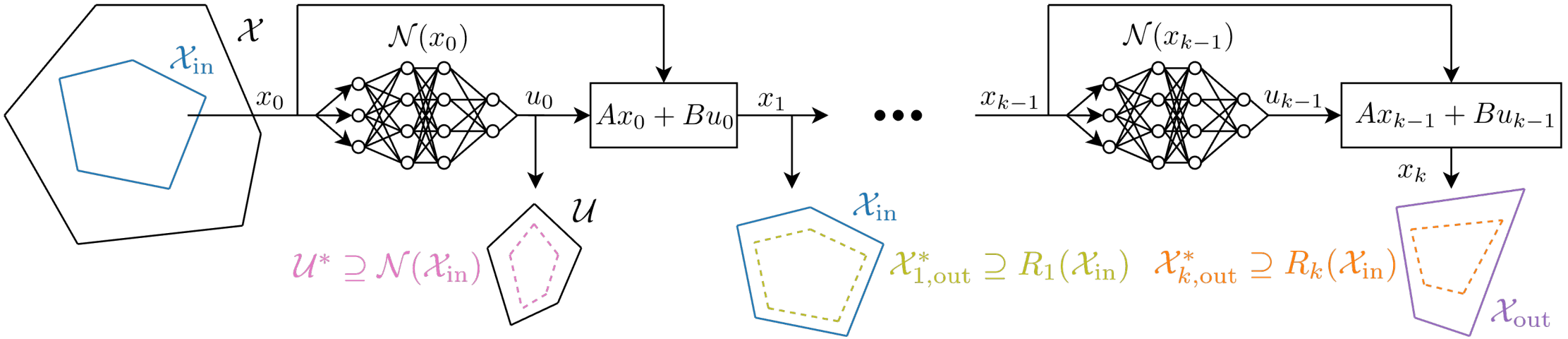
[Dutta et al., NASA Formal Methods Symposium, 2018]

$$
\begin{aligned}
\underset{z,t,u_0,x_0}{\text{maximize}} \quad & C_u^{(i)} u \\
\text{subject to} \quad & C_{\text{in}} x_0 \leq c_{\text{in}}, \\
& z_0 = x_0, \\
& u_0 = W_{L+1} z_L + b_{L+1}, \\
& \text{for all } l \in [L] : \\
& \quad z_l \geq W_l z_{l-1} + b_l, \\
& \quad z_l \leq W_l z_{l-1} + b_l + M t_l, \\
& \quad z_l \geq 0, \\
& \quad z_l \leq M(\mathbf{1} - t_l), \\
& \quad t_l \in \{0,1\}^{n_l},
\end{aligned}
$$

Input constraints satisfied if for all optimal costs $C_u u \leq c_u$

[Karg and Lucia, CDC 2020]

Output-range analysis via MILPs



$$\mathcal{R}_k(\mathcal{X}_{\text{in}}) = f_{\text{cl}} \circ \cdots \circ f_{\text{cl}}(\mathcal{X}_{\text{in}}) \text{ with } f_{\text{cl}}(x_k) = Ax_k + B\mathcal{N}(x_k)$$

Control inputs:

$$\mathcal{N}(\mathcal{X}_{\text{in}}) \subseteq \mathcal{U}$$

Control-invariance:

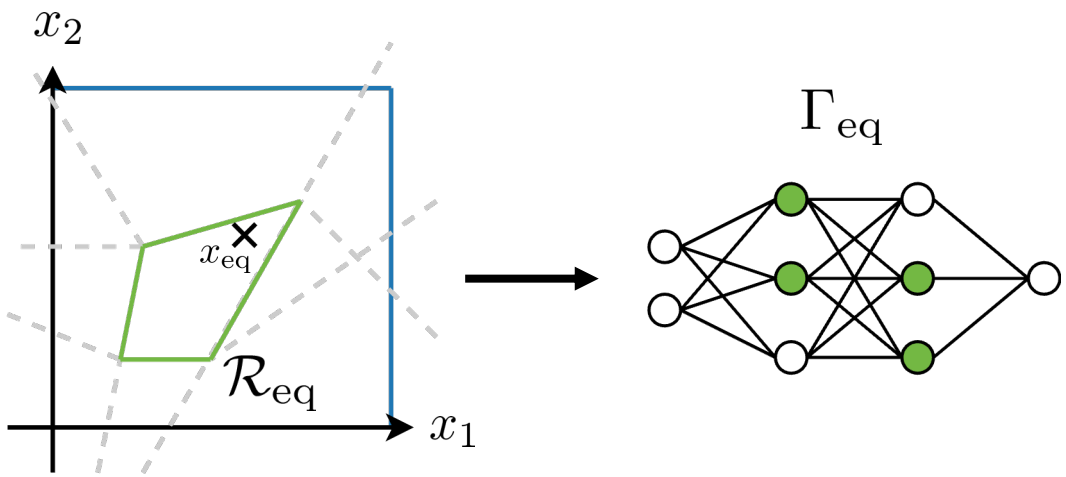
$$\mathcal{R}_1(\mathcal{X}_{\text{in}}) \subseteq \mathcal{X}_{\text{in}}$$

Convergence:

$$\mathcal{R}_k(\mathcal{X}_{\text{in}}) \subseteq \mathcal{X}_{\text{out}}$$

Extended to uncertain systems in [Karg and Lucia, CDC 2022]

Feedback in the neighborhood of equilibrium



Feedback: $\mathcal{P}(x, \Gamma_{\text{eq}}, L+1) =$

$$W_{L+1}(W_{\Gamma_{\text{eq}},L}x + b_{\Gamma_{\text{eq}},L}) + b_{L+1} =$$

$$(W_{L+1}W_{\Gamma_{\text{eq}},L})x + (W_{L+1}b_{\Gamma_{\text{eq}},L} + b_{L+1})$$

State feedback

- Feedback:

$$u = -Kx$$

- Stability condition:

$$\|A - BK\| < 1.$$

Feedback in the neighborhood of equilibrium

Neural network

- Feedback:

$$\mathcal{P}(x, \Gamma_{\text{eq}}, L+1) = W_{L+1}(W_{\Gamma_{\text{eq}},L}x + b_{\Gamma_{\text{eq}},L}) + b_{L+1}$$

- Stability condition:

$$W_{L+1}b_{\Gamma_{\text{eq}},L} + b_{L+1} = 0$$
$$\|A + BW_{L+1}W_{\Gamma_{\text{eq}},L}\| < 1$$

State feedback

- Feedback:

$$u = -Kx$$

- Stability condition:

$$\|A - BK\| < 1.$$

▶ Which neighborhood guarantees asymptotic stability?

Asymptotic stability

Theorem 1 [Karg and Lucia, CDC 2020]

If the neural network controller satisfies

$$W_{L+1} b_{\Gamma_{\mathrm{eq}},L} + b_{L+1} = 0,$$
$$\|A + B W_{L+1} W_{\Gamma_{\mathrm{eq}},L}\| < 1,$$

and there exists a $k \in \mathbb{N}$ such that $\mathcal{X}^*_{k,\mathrm{out}} \subseteq \mathcal{R}_{\mathrm{as}}$,

then the closed-loop system $x_{k+1} = f_{\mathrm{cl}}(x_k)$ is asymptotically stable for all $x \in \mathcal{X}_{\mathrm{in}}$.

- In general, neural network controllers do not satisfy these conditions!
- But we can modify the last layer without affecting the shape of the equilibrium region!

LQR-optimal adaptation

Theorem 2 [Karg and Lucia, CDC 2020]

The convex optimization problem

$$\begin{aligned} \underset{\hat{W}_{L+1},\hat{b}_{L+1}}{\text{minimize}} \quad & \sum_{i,j=1}^{n_u,n_L} (\hat{W}_{L+1}^{(i,j)} - W_{L+1}^{(i,j)})^2 + \sum_{i=1}^{n_u} (\hat{b}_{L+1}^{(i)} - b_{L+1}^{(i)})^2 \\ \text{subject to} \quad & \hat{W}_{L+1} W_{\Gamma_{\text{eq}}} = -K_{\text{lqr}}, \\ & \hat{W}_{L+1} b_{\Gamma_{\text{eq}}} + \hat{b}_{L+1} = 0, \end{aligned}$$

admits a solution which provides weights for the last layer of the neural network controller, such that it behaves like an LQR in the neighborhood of the equilibrium.

Oscillating mass

Two states:

- Displacement s
- Velocity v

One control input:

- Force u

System matrices:

$$A = \begin{bmatrix} 0.5403 & -0.8415 \\ 0.8415 & 0.5403 \end{bmatrix}, B = \begin{bmatrix} -0.4597 \\ 0.8415 \end{bmatrix}$$

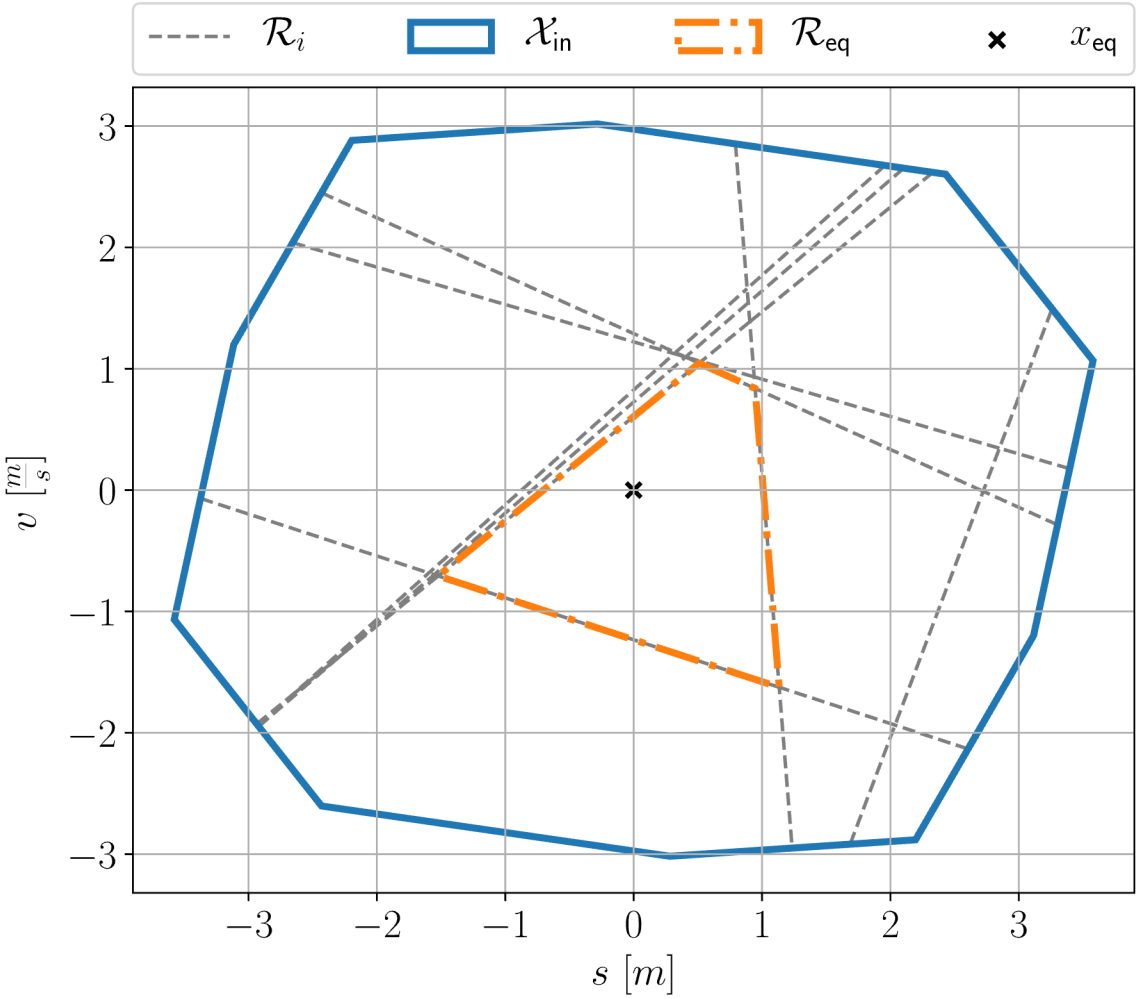
State constraints:

$$\mathcal{X} = \{x \in \mathbb{R}^{n_x} \mid -5 \leq x \leq 5\}$$

Input constraints:

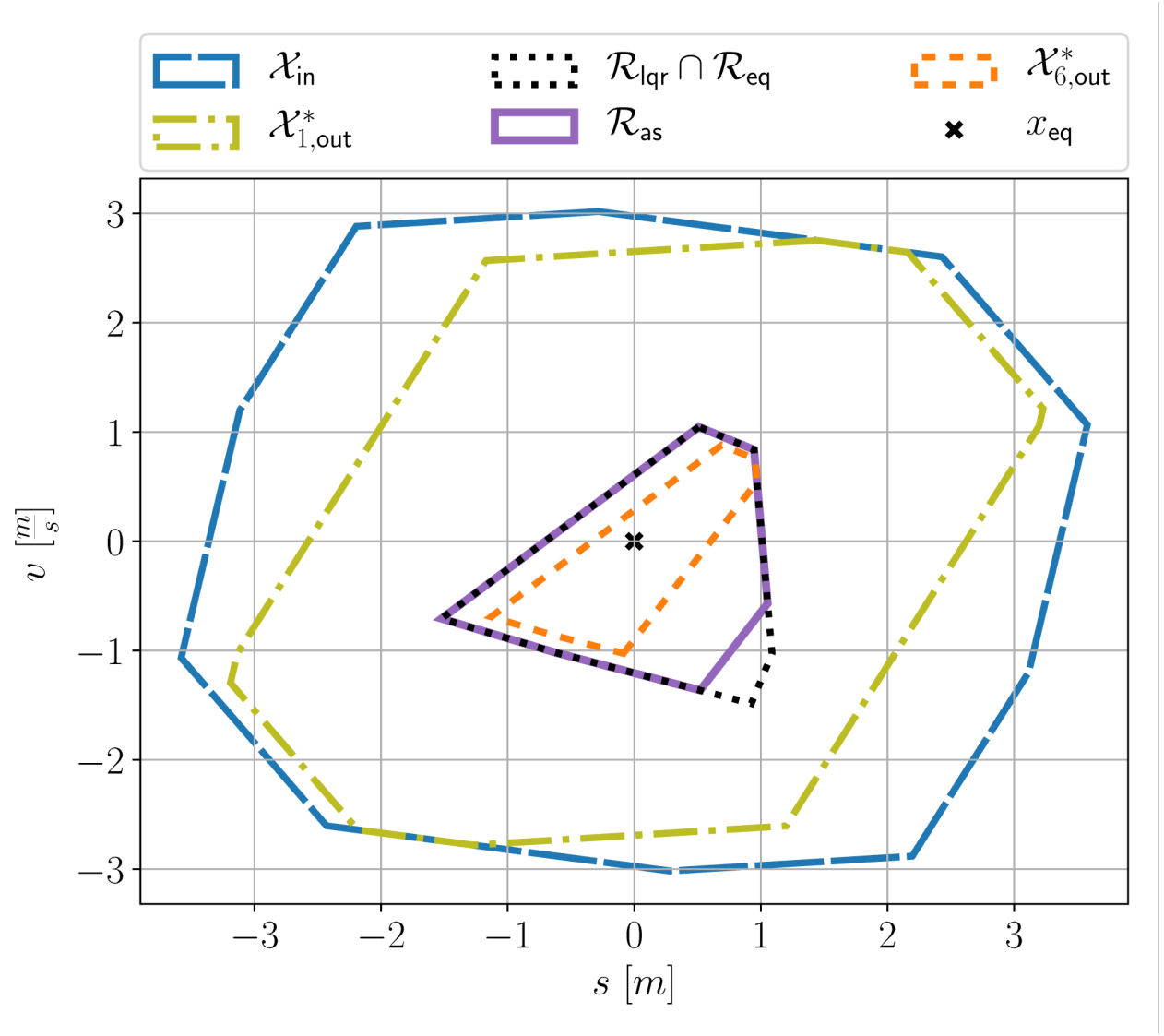
$$\mathcal{U} = \{x \in \mathbb{R}^{n_u} \mid -1 \leq u \leq 1\}$$

Regions of the neural network controller



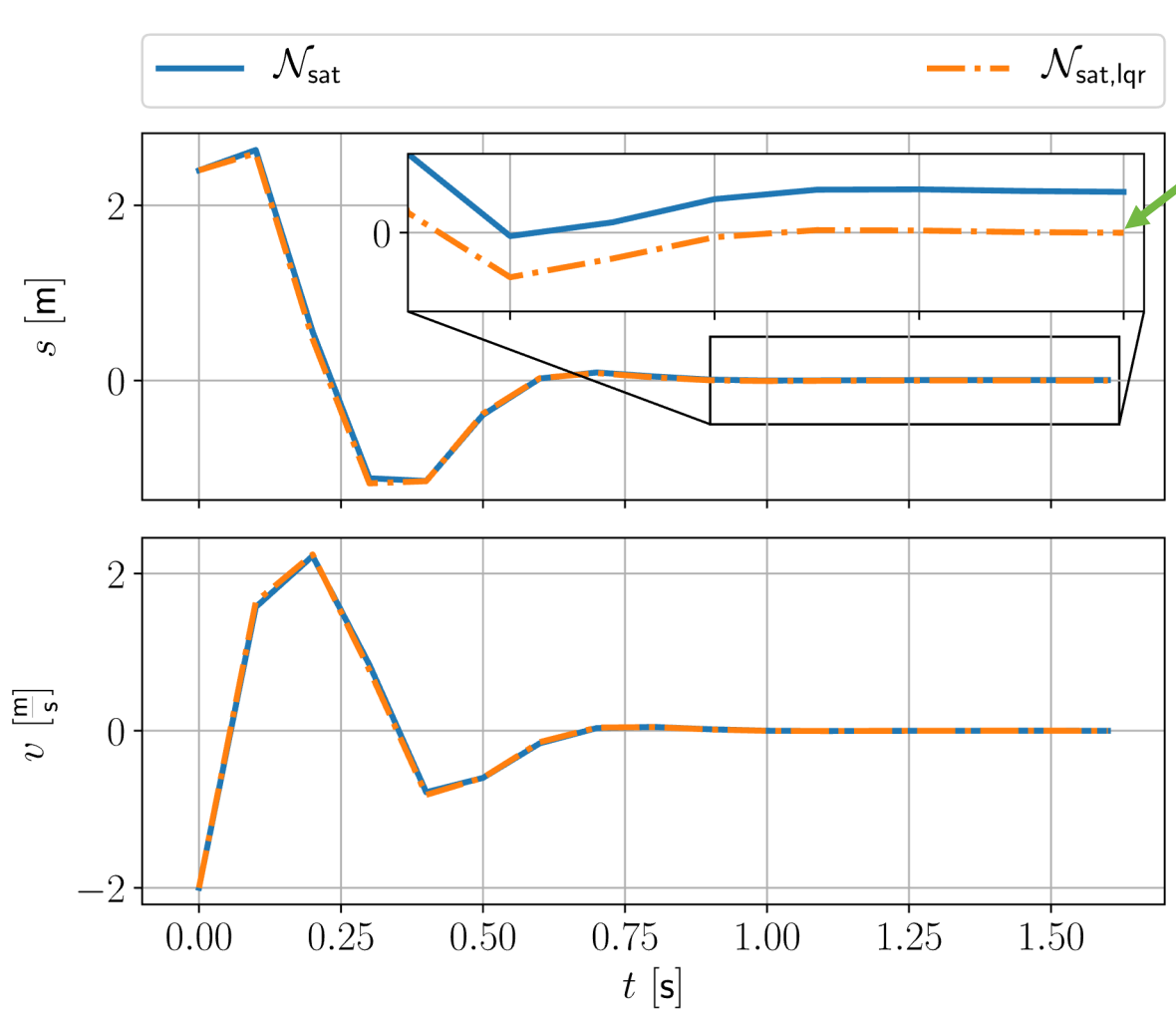
- Regions defined by the ReLU activations
- $\mathcal{X}_{\text{in}}$ obtained from MPT Toolbox
- Box input constraints satisfied via simple modification of neural network
- LQR modification for $K_{\text{lqr}} = \begin{bmatrix} 0.2501 & 0.8290 \end{bmatrix}$ with a cost of 5.4836e-4

Control-invariance and stability



- $\mathcal{X}_{\text{in}}$ is a control-invariant set for neural network controller
- Convergence to $\mathcal{R}_{\text{as}}$ within 6 steps guaranteeing asymptotic stability

Asymptotic stability



Convergence to origin

- LQR-optimized network converges to equilibrium
- Original network has remaining offset error

Discussion

- ReLU neural networks can exactly represent MPC control lawys in the linear case
- Approximate robust NMPC via NNs enables very fast evaluation of approximate robust controllers
- Guarantees can be recovered in some situations

Other points:

- This enables very fast closed-loop simulations of MPC-controlled systems

This is the end

How difficult is it to sample properly?

Open-loop vs. Closed-loop sampling

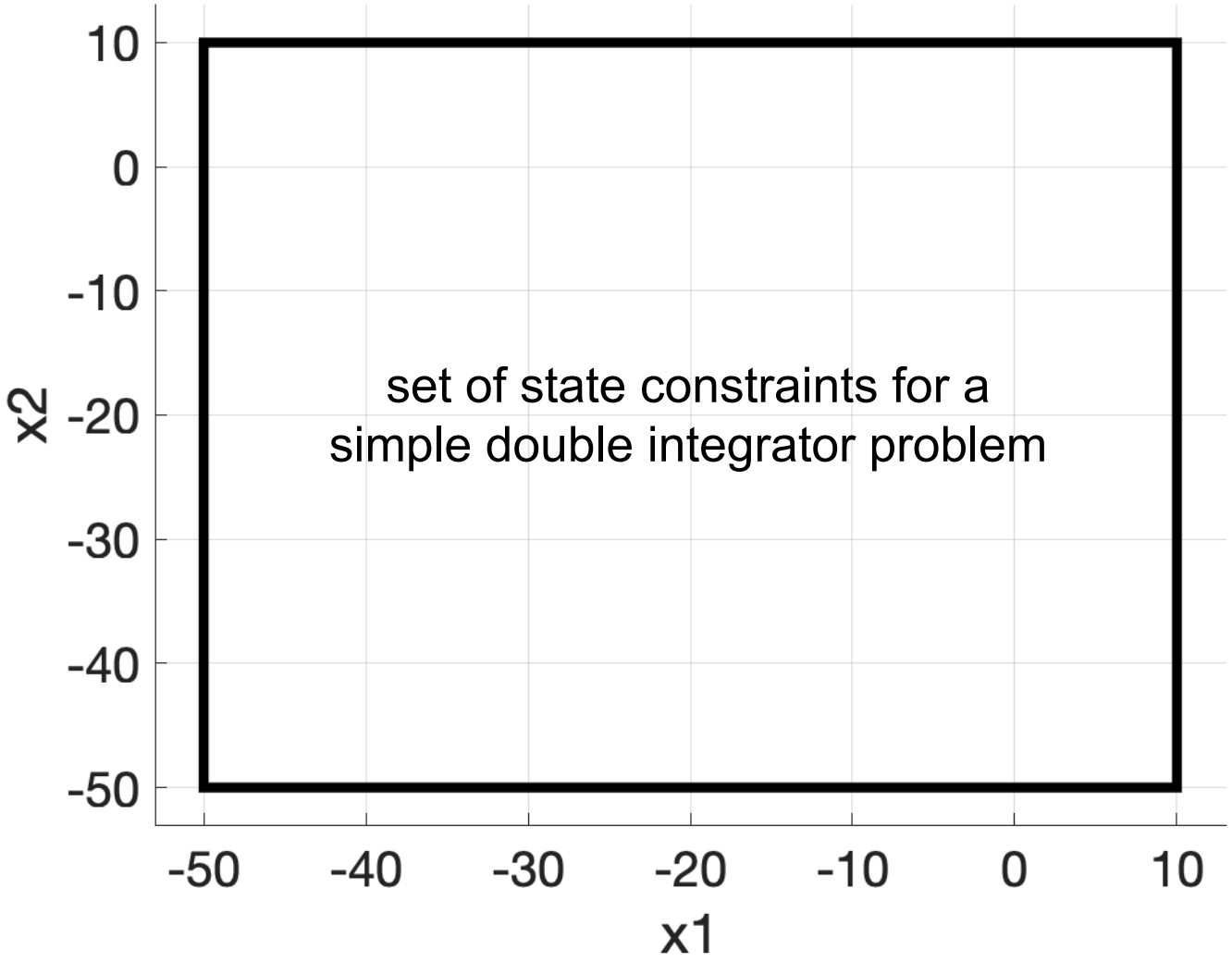
To train the approximate MPC one needs to obtain training data

Training data can be obtained via:

- **Open-loop sampling**: consider all possible feasible initial conditions, sample one and compute the corresponding optimal input
- **Closed-loop sampling:** consider all possible feasible initial conditions. Sample one and perform a closed-loop simulation using MPC. Use the corresponding state and optimal input pairs as data
  - Closed-loop trajectories can include some process noise or uncertainties to explore also the neighborhood of optimal trajectories

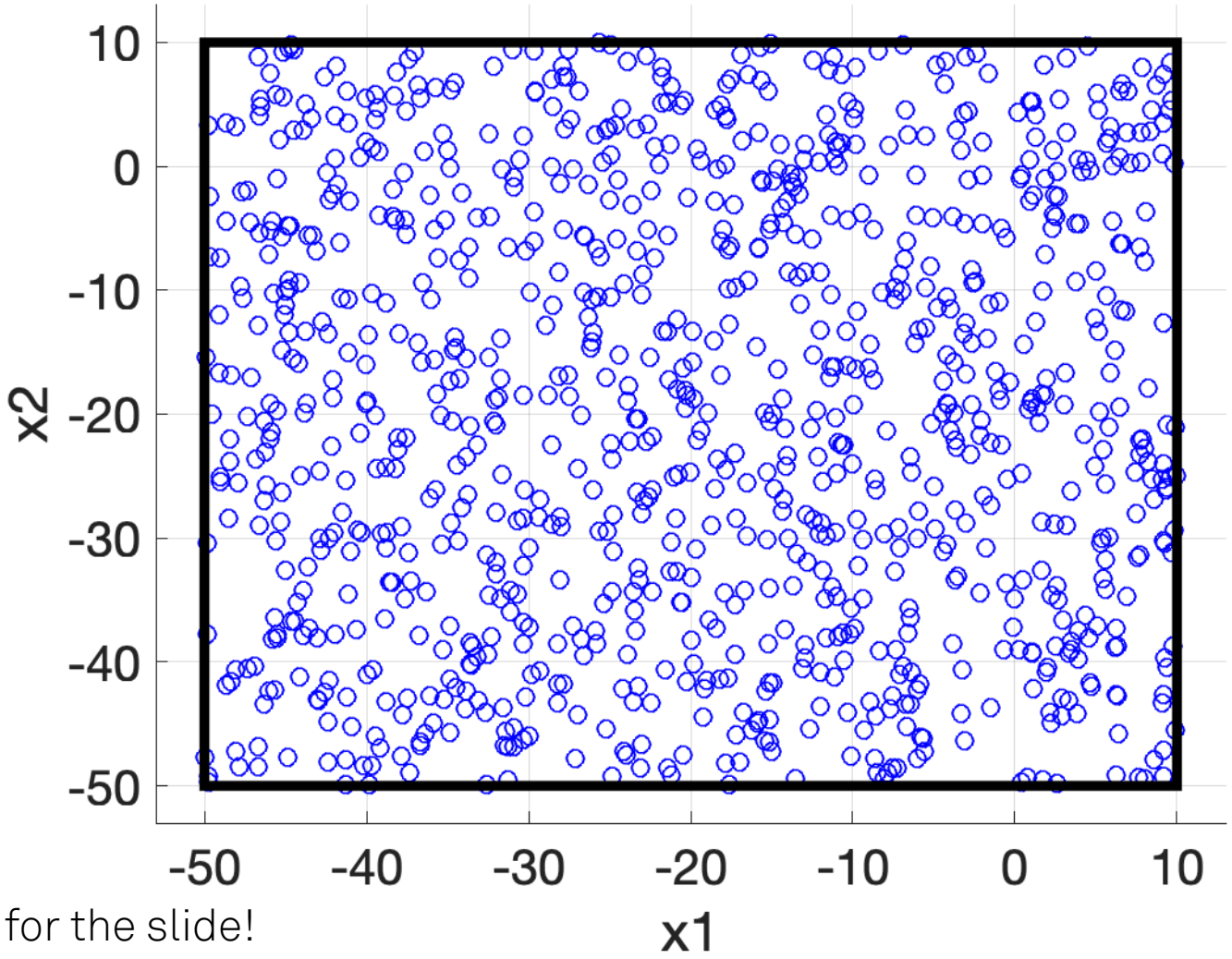
- Closed-loop sampling helps significantly for large state spaces

Illustration: Closed-loop vs. Open-loop Training



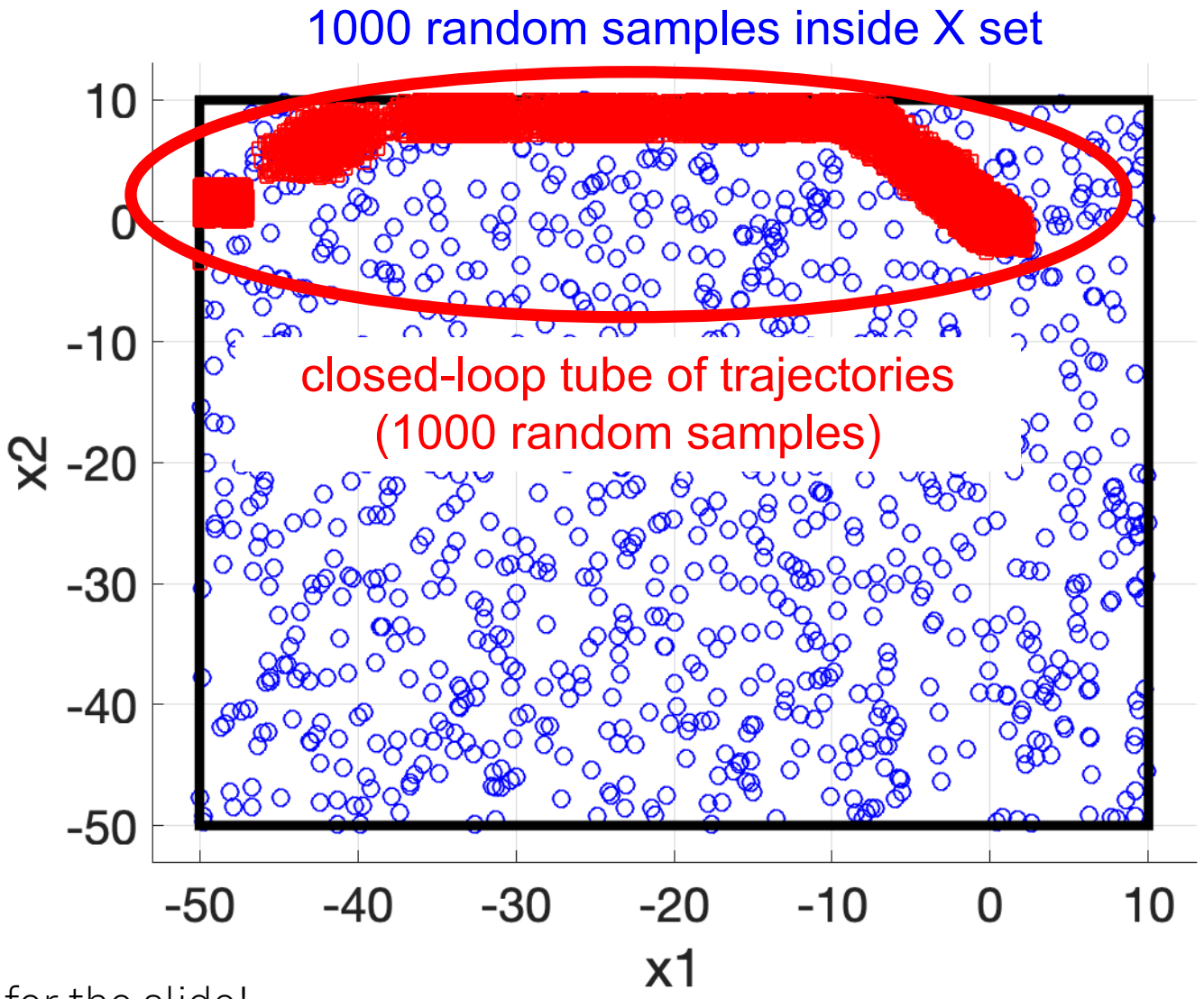
Thanks to Joel Paulson for the slide!

Illustration: Closed-loop vs. Open-loop Training



Thanks to Joel Paulson for the slide!

Illustration: Closed-loop vs. Open-loop Training



Thanks to Joel Paulson for the slide!

Regardless of open or closed-loop, is there something we can do to improve data efficiency?

Data-Efficient Approximate NMPC

- Augmented loss using knowledge about constraints[1]
- NLP sensitivities for data augmentation[2]
- Fitting gradients of linear MPC control law[3]

➢Use of inherent NLP information

All details in [Lüken, Brandner and Lucia, IFAC WC 2023]

- Our work
  - Use of parametric NLP sensitivities
  - Fitting gradients: Sobolev training

$$\min_{\boldsymbol{\Theta}} \; \mathcal{L}_{\text{nom.}}(\boldsymbol{x}_0, \boldsymbol{u}_0; \boldsymbol{\Theta}) + \left\| \frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{x}_0}(\boldsymbol{x}_0) - \frac{\partial \hat{\boldsymbol{u}}_0}{\partial \boldsymbol{x}_0}(\boldsymbol{x}_0; \boldsymbol{\Theta}) \right\|_{\text{F}}^2$$

- Extension of Ideas of[2,3]

➢How to obtain sensitivities?

➢How to train NN with sensitivities?

[1] S. Adhau, V. Naik, S. Skogestad, 2021, CDC
[2] D. Krishnamoorthy, 2022, TAC
[3] R. Winqvist, A. Venkitaraman, B. Wahlberg, 2021, IFAC Papers Online

Sobolev Training[1,2]

Lukas Lüken

All details in [Lüken, Brandner and Lucia, IFAC WC 2023]

- „Sobolev Spaces“:
  - Metric spaces containing derivatives of function[1]
- **Data**

$$\mathbb{D} = \{(\boldsymbol{x}_0^i, \boldsymbol{u}_0^i, \left(\frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{x}_0}\right)^i) | i = 1, \ldots, N_s\}$$

- **Predictions**

$$\hat{\boldsymbol{u}}_0^i = NN(\boldsymbol{x}_0^i; \boldsymbol{\Theta}) \in \mathbb{R}^{n_u}$$

$$\left(\frac{\partial \hat{\boldsymbol{u}}_0}{\partial \boldsymbol{x}_0}\right)^i = \frac{\partial NN}{\partial \boldsymbol{x}_0}(\boldsymbol{x}_0^i; \boldsymbol{\Theta}) \in \mathbb{R}^{n_u \times n_x}$$

- Sobolev Loss

$$\mathcal{L}_{\text{sob}}(\boldsymbol{x}_0, \boldsymbol{u}_0, \frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{x}_0}; \boldsymbol{\Theta}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \|\boldsymbol{u}_0^i - \hat{\boldsymbol{u}}_0^i\|_2^2 + \gamma \cdot \frac{1}{N_s} \sum_{i=1}^{N_s} \left\| \left(\frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{x}_0}\right)^i - \left(\frac{\partial \hat{\boldsymbol{u}}_0}{\partial \boldsymbol{x}_0}\right)^i \right\|_{\text{F}}^2$$

- Training (via **SGD**)

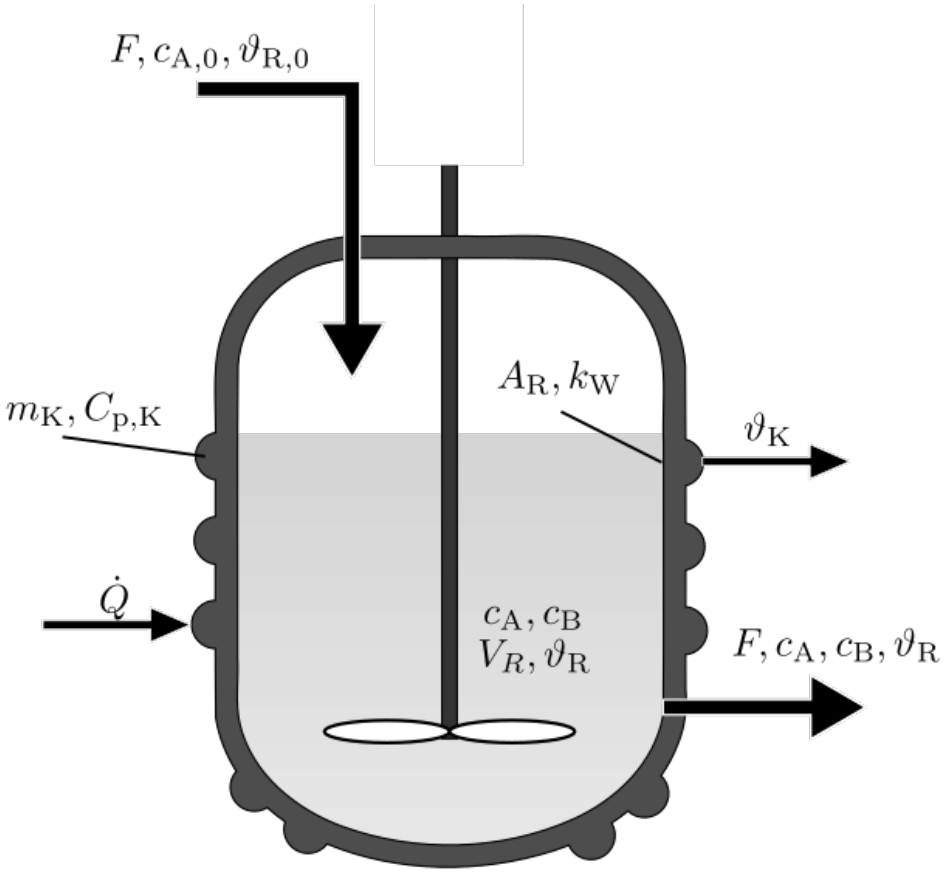
$$\min_{\boldsymbol{\Theta}} \mathcal{L}_{\text{sob}}(\boldsymbol{x}_0, \boldsymbol{u}_0, \frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{x}_0}; \boldsymbol{\Theta})$$

[1] W. Czarnecki et al., 2017, NeurIPS
[2] J. Cocola and P. Hand, 2020, LOD

Continuous stirred tank reactor (CSTR)

- Nonlinear dynamics
- 4 states $(c_\mathrm{A}, c_\mathrm{B}, \vartheta_\mathrm{R}, \vartheta_\mathrm{K})$
- 2 control inputs $\left(F, \dot{Q}\right)$

$$
\begin{aligned}
\frac{\partial c_\mathrm{A}}{\partial t} &= F\left(c_{\mathrm{A},0} - c_\mathrm{A}\right) - k_1 c_\mathrm{A} - k_3 c_\mathrm{A}^2 \\
\frac{\partial c_\mathrm{B}}{\partial t} &= -F c_\mathrm{B} + k_1 c_\mathrm{A} - k_2 c_\mathrm{B} \\
\frac{\partial \vartheta_\mathrm{R}}{\partial t} &= F\left(\vartheta_{\mathrm{R},0} - \vartheta_\mathrm{R}\right) + \frac{k_\mathrm{W} A_\mathrm{R}}{\varrho C_{p,\mathrm{R}} V_\mathrm{R}}\left(\vartheta_\mathrm{K} - \vartheta_\mathrm{R}\right) \\
&\quad - \frac{k_1 c_\mathrm{A} \Delta H_{R,1} + k_2 c_\mathrm{B} \Delta H_{R,2} + k_3 c_\mathrm{A}^2 \Delta H_{R,3}}{\varrho C_{p,\mathrm{R}}} \\
\frac{\partial \vartheta_\mathrm{K}}{\partial t} &= \frac{\dot{Q} + k_\mathrm{W} A_\mathrm{R}\left(\vartheta_\mathrm{R} - \vartheta_\mathrm{K}\right)}{m_\mathrm{K} C_{p,\mathrm{K}}} \\
k_i &= k_{0,i} \exp\left(\frac{-E_{A,i}}{R\left(\vartheta_\mathrm{R} + 273.15\right)}\right)
\end{aligned}
$$

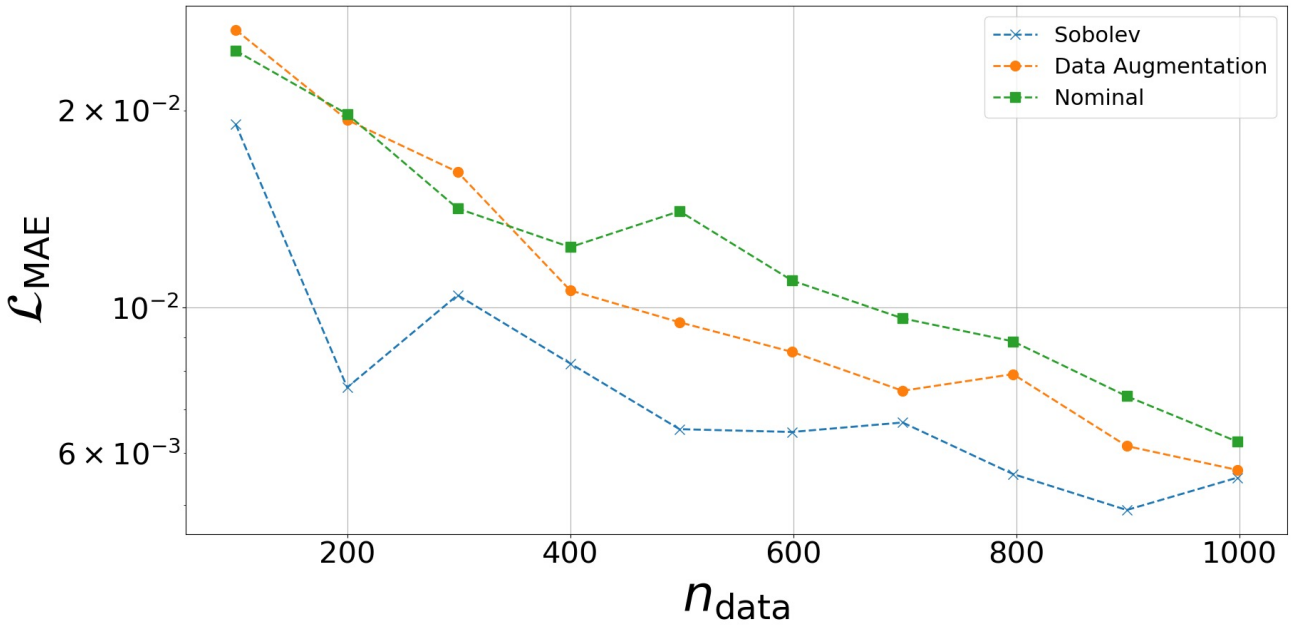


[K. Klatt and S. Engell, 1998, Computers & Chemical Engineering]

Sobolev Training with NLP Sensitivities

All details in [Lüken, Brandner and Lucia, IFAC WC 2023]

- Hyperparameter
  - Grid search on approx. NMPC with nominal training
  - 3 Layers, 100 Neurons
  - Tanh activation
- Evaluation on mean absolute error (MAE)
  - Predictions only: $\|\boldsymbol{u}_0 - \hat{\boldsymbol{u}}_0\|$
  - Less sensitive to outliers



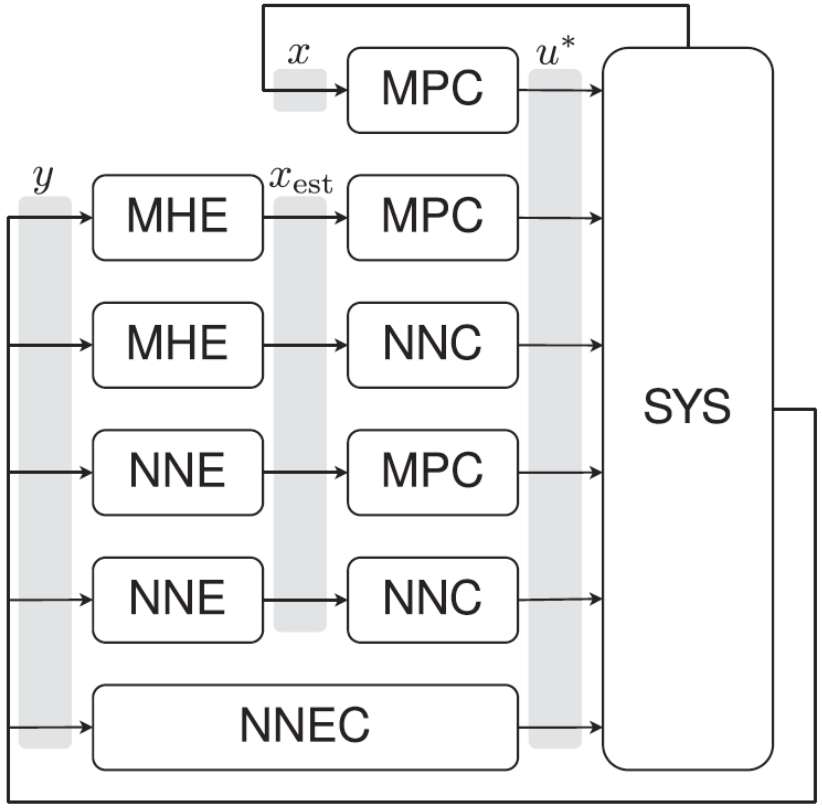
- Sobolev training useful in low data regime
- Large sensitivities might lead to high approximation error

Constraints in the state space

$$
\begin{aligned}
&\underset{\mathbf{z},\mathbf{t},\mathbf{u},x_0,x_k}{\text{maximize}} && a_x x_k \\
&\text{subject to} && C_{\text{in}} x_0 \leq c_{\text{in}}, \\
& && x_k = A z_{k,0} + B u_k \\
& && z_{1,0} = x_0, \\
& && z_{j,0} = A z_{j-1,0} + B u_{j-1} \, \forall j \in [k] \setminus 1, \\
& && \text{for all } j \in [k] \text{ and } l \in [L]: \\
& && \quad z_{j,l} \geq W_l z_{j,l-1} + b_l, \\
& && \quad z_{j,l} \leq W_l z_{j,l-1} + b_l + M t_{j,l}, \\
& && \quad z_{j,l} \geq 0, \\
& && \quad z_{j,l} \leq M(\mathbf{1} - t_{j,l}), \\
& && \quad t_{j,l} \in \{0,1\}^{n_l}, \\
& && \quad u_j = W_{L+1} z_{j,L} + b_{L+1}
\end{aligned}
$$

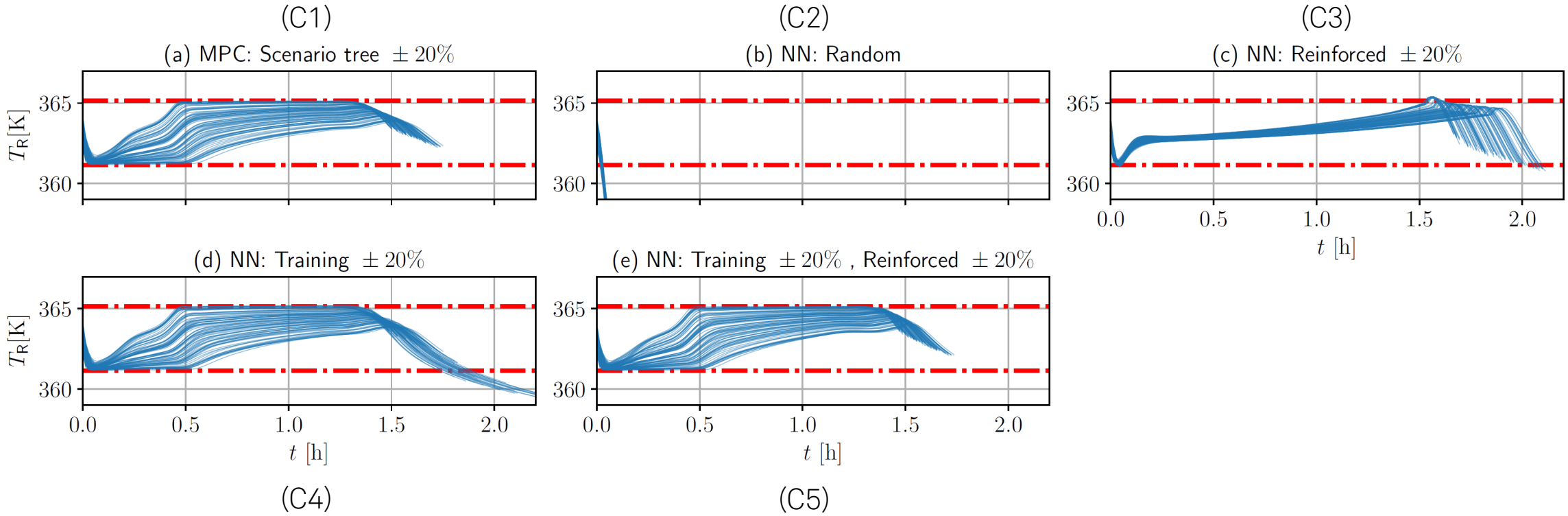
- If $a_x x_k^* \leq b_x$, $\mathcal{R}_k(x_0) \leq b_x \, \forall x_0 \in \mathcal{X}_{\text{in}}$

Approximating NMPC and MHE



[Karg and Lucia, *Computers and Chemical Engineering*, 2020]

Reinforced approximate predictive control



- Direct reinforcement learning leads to very conservative results
- Imitation learning is strongly depending on provided data