EPFL

CDC 2023 SINGAPORE

Unconstrained Parametrization of Dissipative and Contracting Neural Ordinary Differential Equations

Daniele Martinelli,

Clara Lucía Galimberti,

Ian R. Manchester,

Luca Furieri,

Giancarlo Ferrari-Trecate

EPFL

THE UNIVERSITY OF SYDNEY

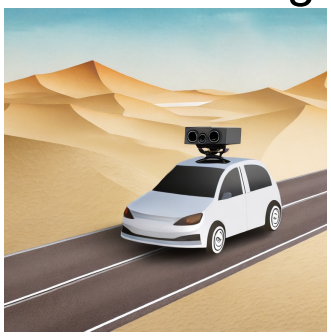
NCCR Automation

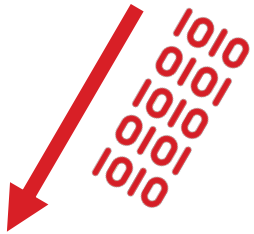
Learning for Complex Systems

Robotics

Power Grids

Autonomous Driving Vehicles

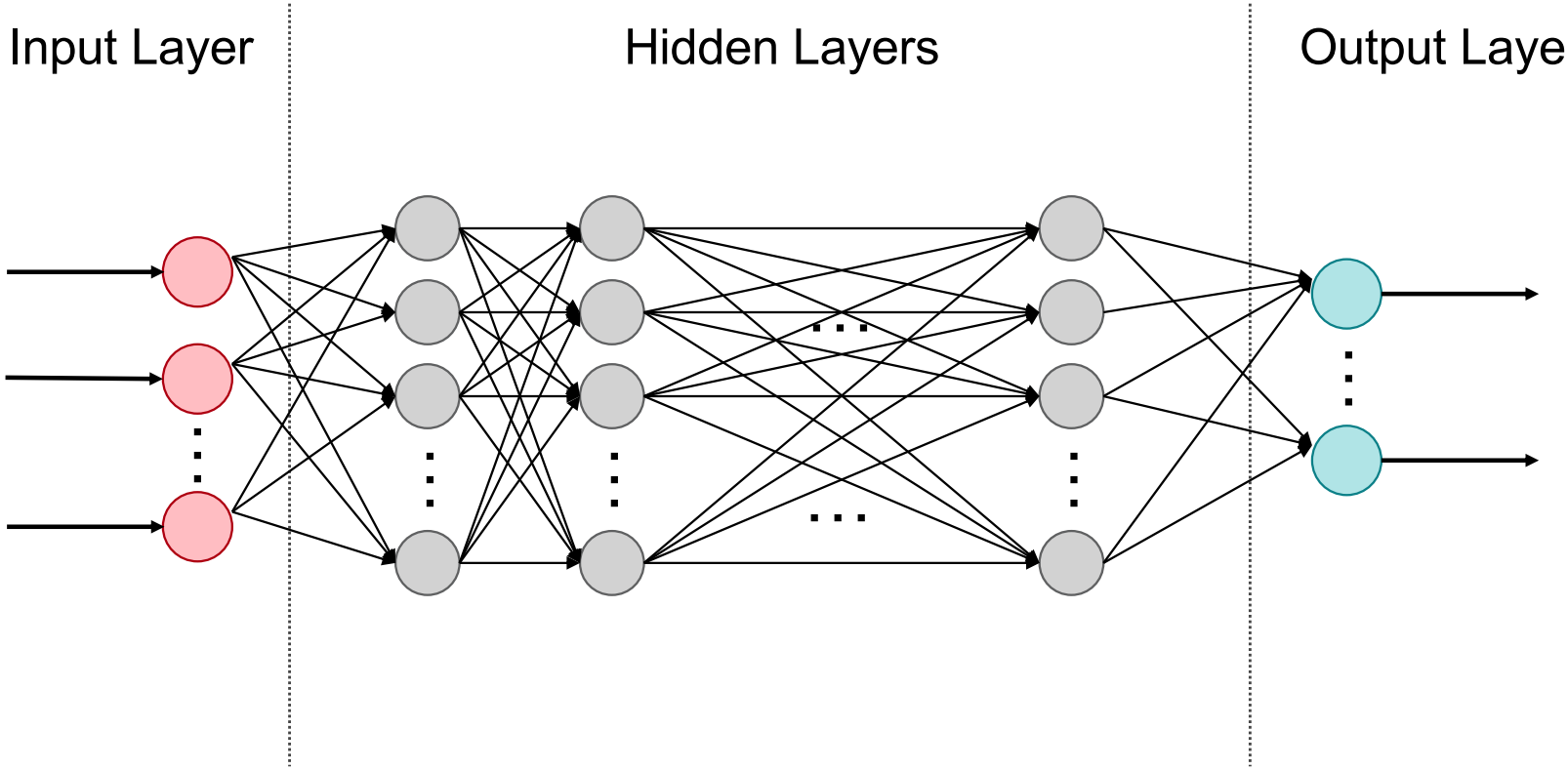




Machine Learning

Deep Neural Networks

Deep Neural Networks (DNNs)



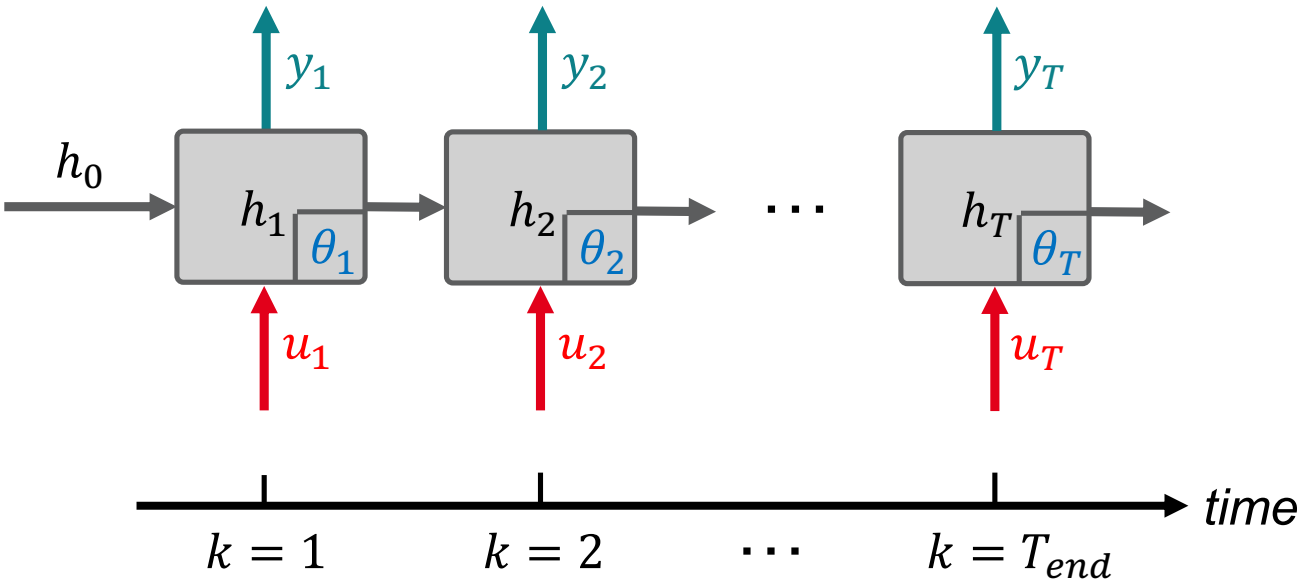
ODE-based DNNs

Ordinary Differential Equation (ODE)

$$\frac{d\,h(t)}{dt} = f(h(t), t, \theta(t), u(t))$$
$$h(0) = h_0$$

Discretization

$$h_{k+1} = h_k + f(h_k, \theta_k, u_k),\ k \in \{0 \dots T_{end}\}$$



Can be seen as a DNN!

u_k: Inputs
y_k: Outputs
h_k: States
θ_k: Parameters

Neural-ODEs

Neural Ordinary Differential Equations

Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*, David Duvenaud
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

- Continuous time system
- Use of modern ODE solvers for the evolution of the model
- Integration methods with
  - Fixed-steps
  - Variable-steps
- Open-Source library

Drawbacks for system identification & control:

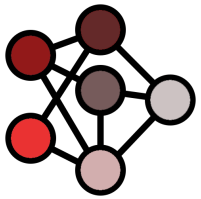
- No guarantees
  - Sensitive to perturbations in I/O
  - No Stability guarantees

Contributions & Contents

Goal: Design class of NeuralODE that enjoys stability properties by design

Targeted Properties

Stability notions



NodeRENs

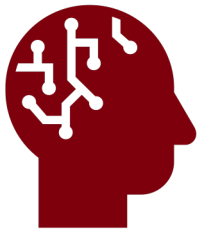
Guaranteed properties through
Unconstrained Parametrization

Numerical Example

Nonlinear System Identification

EPFL

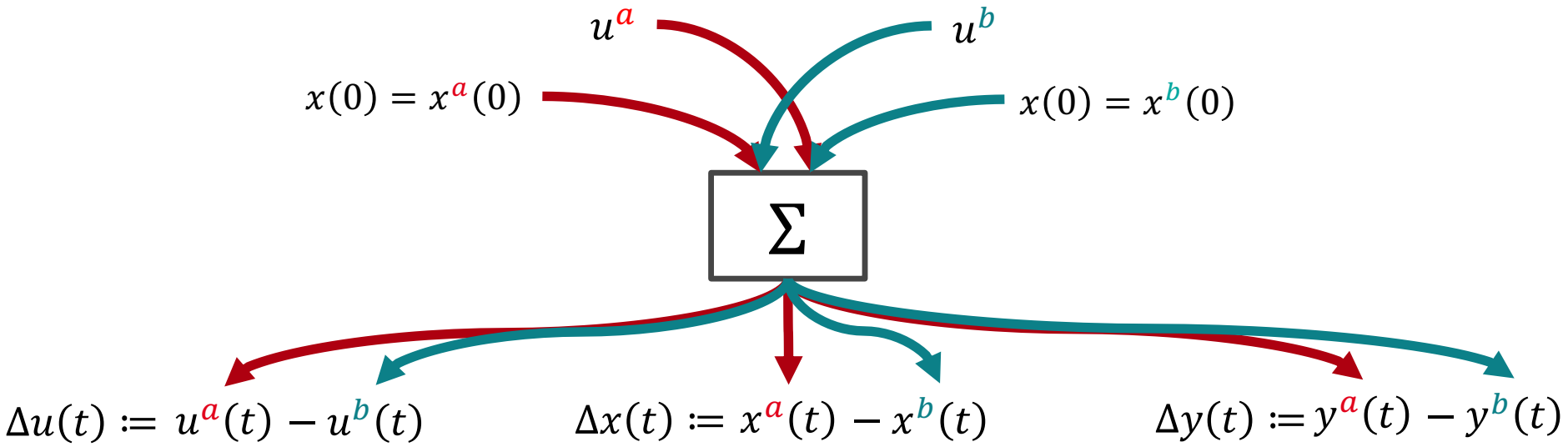
Targeted Properties



Incremental Properties

MIMO system $\Sigma = \begin{cases} \dot{x}(t) = f(x(t), u(t)) \\ y(t) = g(x(t), u(t)) \end{cases}$

Comparison of different trajectories

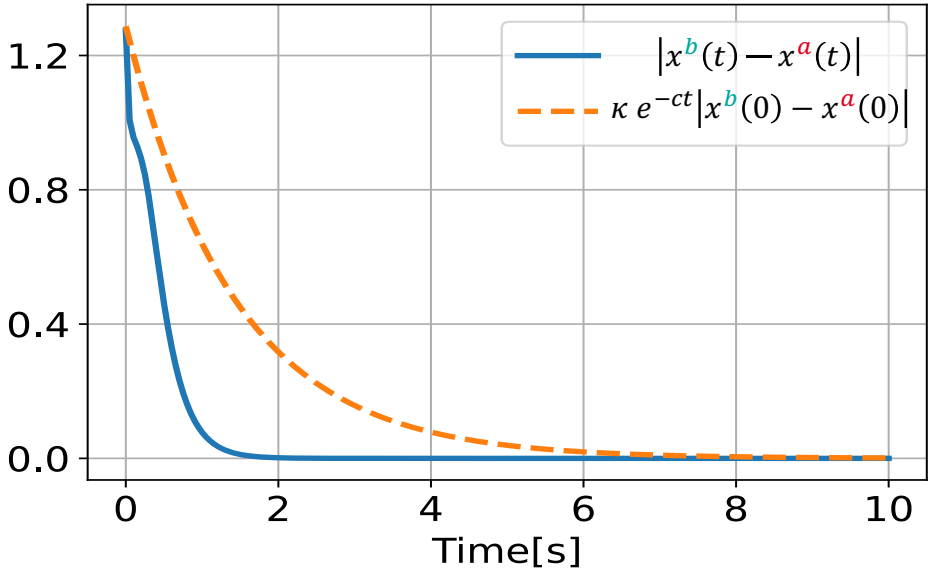


Contractivity

Definition

The system Σ is contracting if for any $x^a(0),\ x^b(0)$ and $u^a = u^b$:

$$\left|x^b(t) - x^a(t)\right| \leq \kappa\, e^{-ct}\left|x^b(0) - x^a(0)\right| \qquad \forall t \geq 0 \qquad \kappa, c > 0$$



If the system has an equilibrium point:

Contractivity

↓

(Globally) Exponential Stability!

Dissipativity

Internal energy → Storage function: $V(\cdot)$

Exchange through inputs/outputs → supply rate: $s(\cdot,\cdot)$

Definition

The system Σ is dissipative w.r.t. a supply rate $s(\cdot,\cdot)$, if there exists a storage function $V(\cdot)$ such that:

$$V\big(x(t_1)\big) - V\big(x(t_0)\big) \leq \int_{t_0}^{t_1} s\big(u(t), y(t)\big)dt \ , \qquad \forall t_1 \geq t_0$$

Why dissipativity?

- Tight connection with *Lyapunov stability* *(under mild conditions)*

Common choices of supply rate $s(\cdot,\cdot)$?

Quadratic Supply Rates

$$s\big(\Delta u(t), \Delta y(t)\big) = \begin{bmatrix}\Delta y(t)\\ \Delta u(t)\end{bmatrix}^{\mathsf{T}} \begin{bmatrix}Q & S^{\mathsf{T}}\\ S & R\end{bmatrix} \begin{bmatrix}\Delta y(t)\\ \Delta u(t)\end{bmatrix}$$

Common choices of (Q, S, R):

ℓ_2 Lipschitz bound:

$\|\Delta y(t)\|_T \leq \gamma \|\Delta u(t)\|_T$ → $Q = -\frac{1}{\gamma} I, \quad R = \gamma I, \quad S = 0$

Incremental Input Passive:

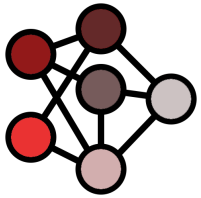
$s = \Delta u(t)^{\mathsf{T}} \Delta y(t) - \nu \|\Delta u(t)\|^2$ → $Q = 0, \quad R = -2\nu I, \quad S = I$

Incremental Output Passive:

$s = \Delta u(t)^{\mathsf{T}} \Delta y(t) - \rho \|\Delta y(t)\|^2$ → $Q = -2\rho I, \quad R = 0, \quad S = I$

EPFL

NODE-RENs



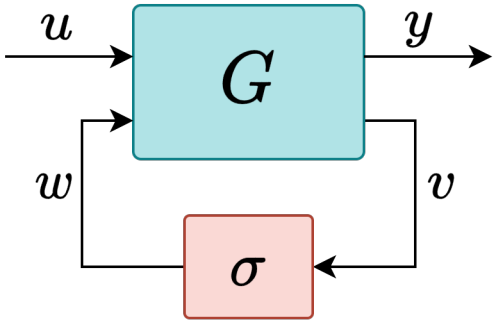
Neural ODE Recurrent Equilibrium Networks (NodeRENs)

Recurrent Equilibrium Networks: Flexible Dynamic Models with Guaranteed Stability and Robustness

Max Revay*, Ruigang Wang*, Ian R. Manchester

Continuous-Time settings + REN (Discrete-Time) = **NodeREN**

NodeREN Model:



=

Linear part:

$$\begin{bmatrix} \dot{x}_t \\ v_t \\ y_t \end{bmatrix} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} x_t \\ w_t \\ u_t \end{bmatrix} + \begin{bmatrix} b_x \\ b_v \\ b_y \end{bmatrix}$$

Nonlinear part:

$$w_t = \sigma(v_t)$$

Contracting & Dissipative NodeRENs

Theorem 1 (*Contracting NodeREN*).
A NodeREN is contracting if there exists a $P \succ 0$ and a $\Lambda \in \mathbb{D}_+$ such that:

$$\begin{bmatrix} -A^\top P - PA & -C_1^\top \Lambda - PB_1 \\ -\Lambda C_1 - B_1^\top P & W \end{bmatrix} \succ 0$$

$$\textit{with: } W = 2\Lambda - \Lambda D_{11} - D_{11}^\top \Lambda$$

Theorem 2 (*Dissipative NodeREN*).
A NodeREN is incrementally dissipative w.r.t. quadratic supply rate if there exists a $P \succ 0$ and a $\Lambda \in \mathbb{D}_+$ such that:

$$\begin{bmatrix} -A^\top P - PA & -C_1^\top \Lambda - PB_1 & -PB_2 + C_2^\top S^\top \\ -\Lambda C_1 - B_1^\top P & W & D_{21}^\top S^\top - \Lambda D_{12} \\ -B_2^\top P + SC_2 & SD_{21} - D_{21}^\top \Lambda & R + SD_{22} + D_{22}^\top S^\top \end{bmatrix} + \begin{bmatrix} C_2^\top \\ D_{21}^\top \\ D_{22}^\top \end{bmatrix} Q \begin{bmatrix} C_2^\top \\ D_{21}^\top \\ D_{22}^\top \end{bmatrix}^\top \succ 0$$

Computationally Expensive!

Can we remove constraints?

Unconstrained Parametrization

Contractivity matrix inequality

$$\begin{bmatrix} -A^\top P - PA & -C_1^\top \Lambda - PB_1 \\ -\Lambda C_1 - B_1^\top P & W \end{bmatrix} \succ 0$$

NodeREN Model:

$$\begin{bmatrix} \dot{x}_t \\ v_t \\ y_t \end{bmatrix} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} x_t \\ w_t \\ u_t \end{bmatrix} + \begin{bmatrix} b_x \\ b_v \\ b_y \end{bmatrix}$$

$$w_t = \sigma(v_t)$$

$$H = X^\top X + \epsilon I \succ 0 \text{ always!}$$

$$P = X_P^\top X_P + \epsilon_P I \succ 0$$

How we obtain a C-NodeREN

I) Choose freely X, X_p, build H, P II) Get matrices from H, P III) Choose freely the remaining ones

Enforcing Dissipativity follows similar steps!

$H = X^\top$ …
$P = X_P^\top X_P + \epsilon_P I$

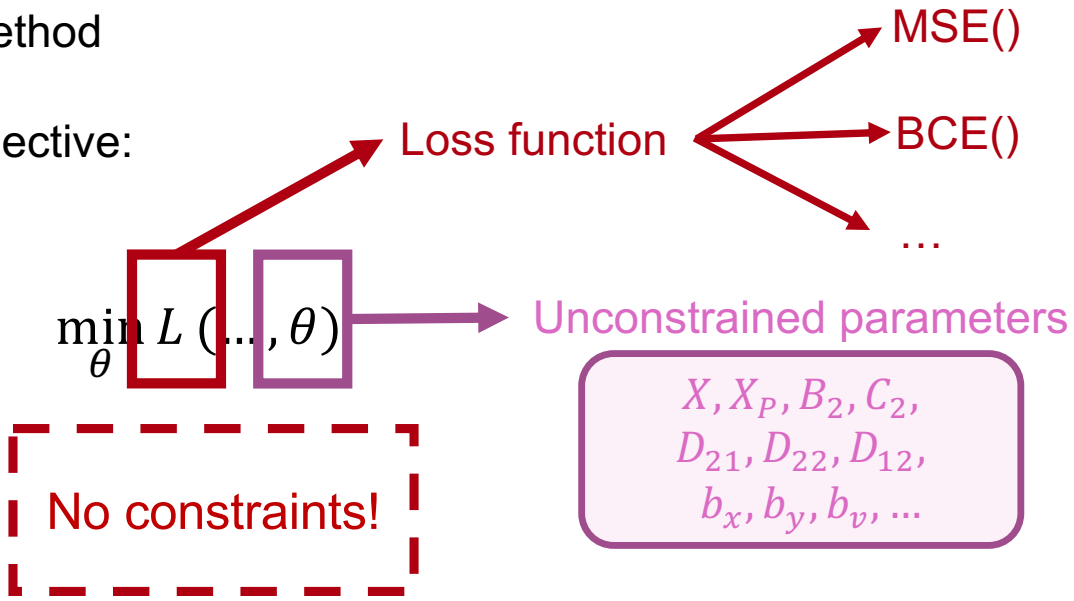
$A, B_1,$
C_1, D_{11}, Λ

$D_{12},$
$D_{21}, D_{22},$
$b_x, b_y, b_v, \ldots$

Unconstrained Parameters

How To Train NodeRENs

- Choose the property to be guaranteed: contractivity and/or incremental dissipativity
- Choose an integration method
- Define a minimization objective:



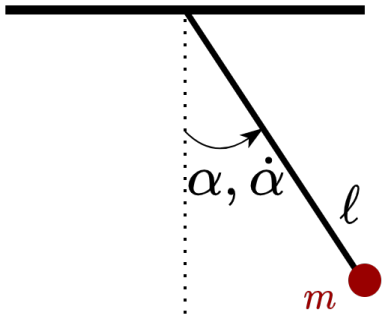
- Easy to optimize over with gradient descent or stochastic variants (e.g., SGD, ADAM)

PyTorch

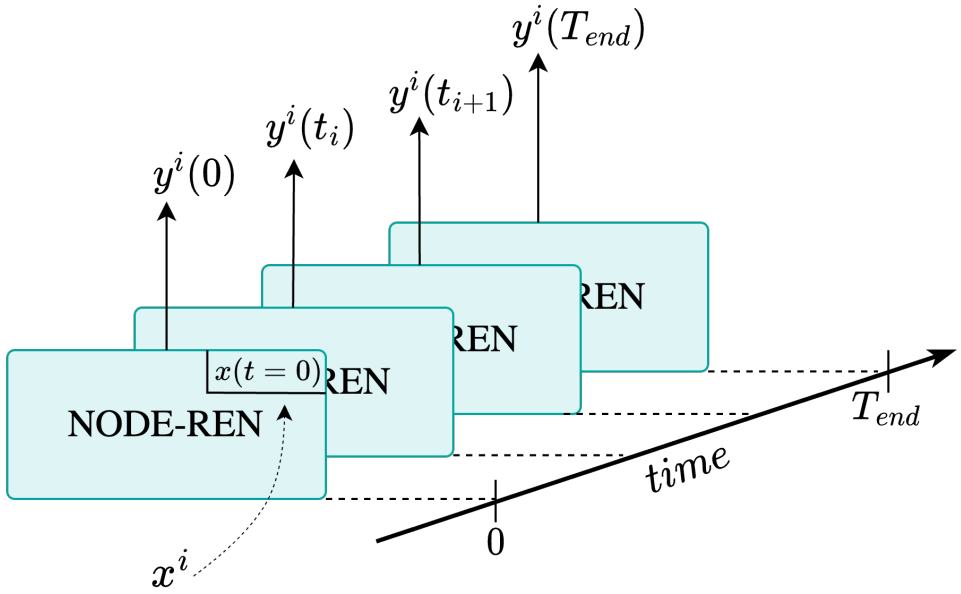
Numerical Example

System Identification (SYSID)

- **System:** Pendulum

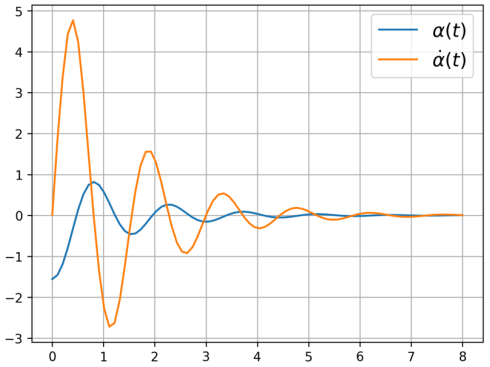


$$\ell\ddot{\alpha}(t) + \beta\dot{\alpha}(t) + g\sin\alpha(t) = 0$$



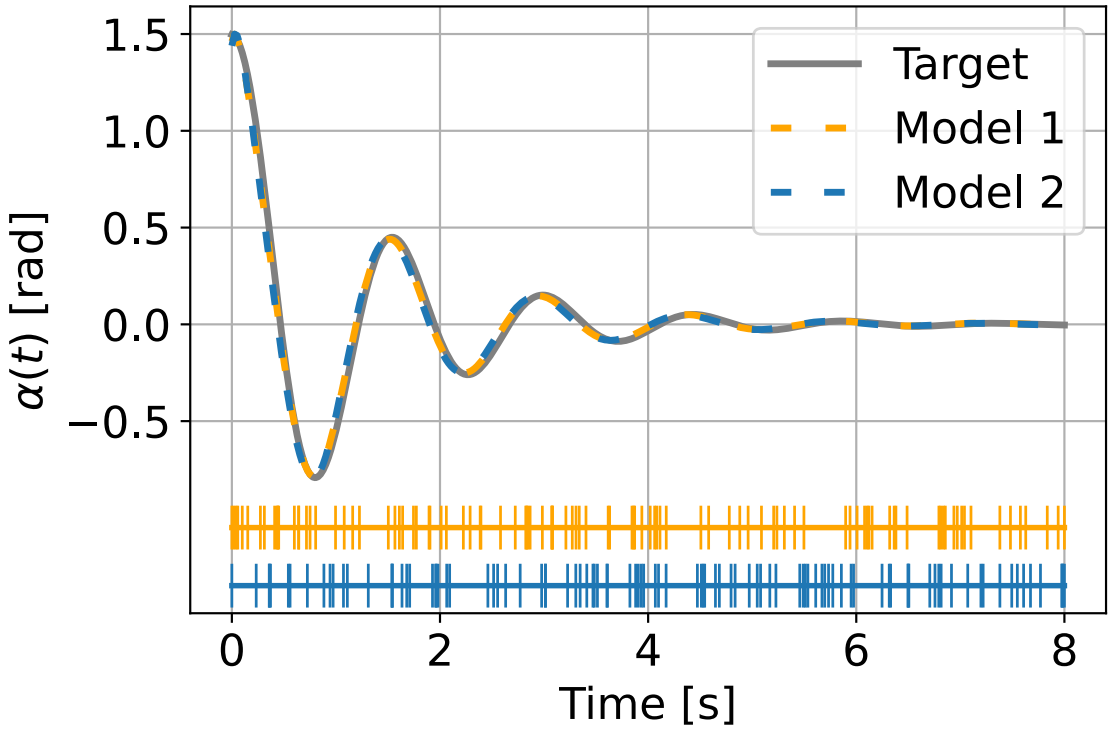
- **Loss Function:** Mean Squared Error (MSE)

- **Experiments:** $\begin{bmatrix}\alpha(0)\\ \dot{\alpha}(0)\end{bmatrix}$ →

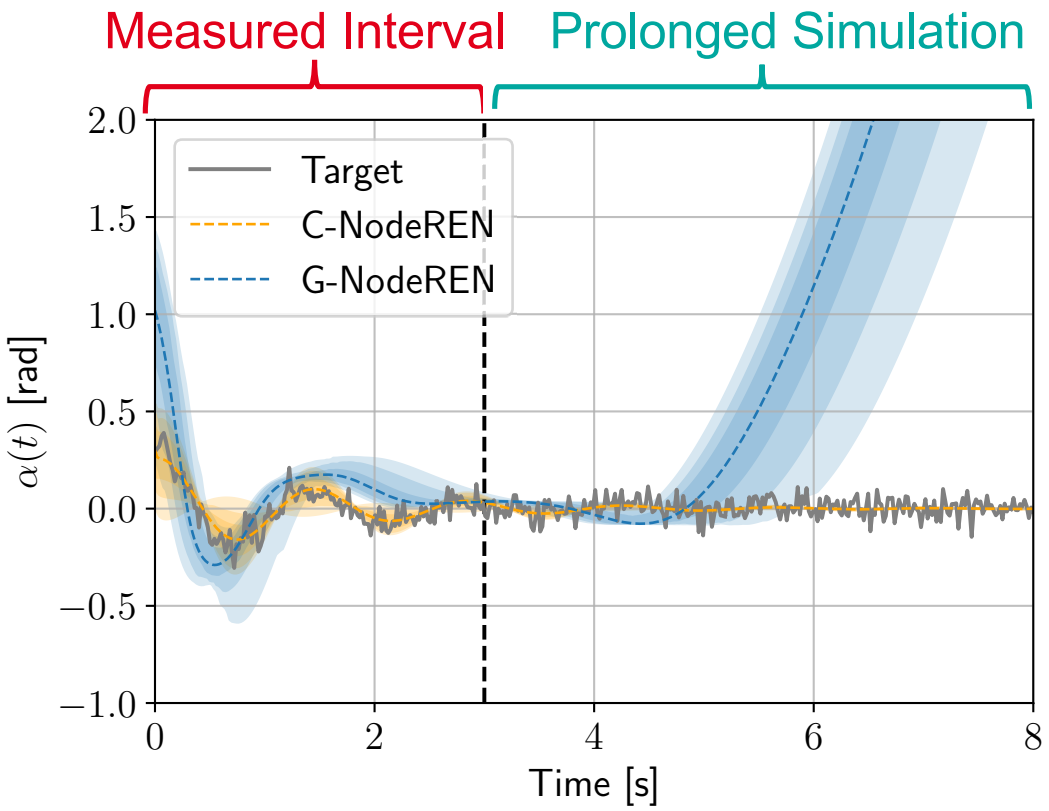


SYSID - Irregularly Sampled Data

Can handle naturally irregularly sampled data! (e.g., from sensor networks)



SYSID – Why Enforcing Contractivity



- We enforce contractivity by design
  ↪ Embedding prior knowledge
- Can be simulated for longer time horizons (w.r.t. training time)
- Improved numerical stability during simulations

Conclusions

Summary

- Class of DNN with guarantees of contractivity and/or inc. dissipativity by design
- Nonlinear system identification embedding contractivity/inc. dissipativity priors

Future Developments

- Beyond SYSID Applications:
  - Optimal control
  - Classification tasks (ML)
- Real-world applications
- Incorporating integration schemes preserving contractivity

Thank you!

Paper

How To Train NodeRENs

COME FACCIO IL TRAINING effettivamente (BACKUP ALLA FINE)

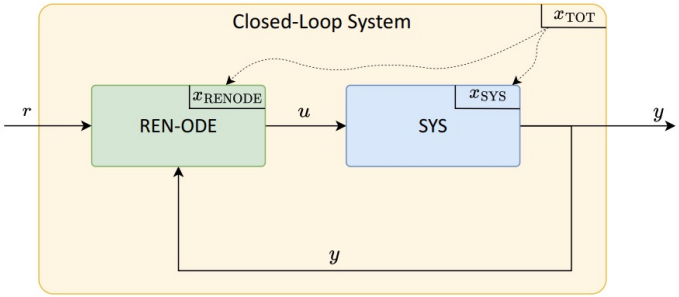


Figure 3.2: Scheme of the implementation of a feedback closed-loop system using Torchdiffeq library. The reference signal r must be defined a priori or generated by a "signal generator" inside the closed-loop system.

Algorithm 3.2 Pseudo-code of the Feed-Forward function in a Closed-Loop system.

Input: model (total system), time instant t, state of the total system $x_{TOT}(t)$.

$x_{\text{RENODE}},\ x_{\text{SYS}} = \text{split}(x_{\text{TOT}}(t))$ ▷ Split state vector in two.

$y(t) = \text{SYS.output}(t, x_{\text{SYS}})$ ▷ output evaluation of SYS

$\dot{x}_{\text{REN-ODE}}(t) = \text{RENODE.forward}(t, x_{\text{SYS}})$ ▷ state evolution of RENODE

$u_{\text{RENODE}}(t) = \text{RENODE.output}(t, x_{\text{SYS}})$ ▷ output evaluation of RENODE

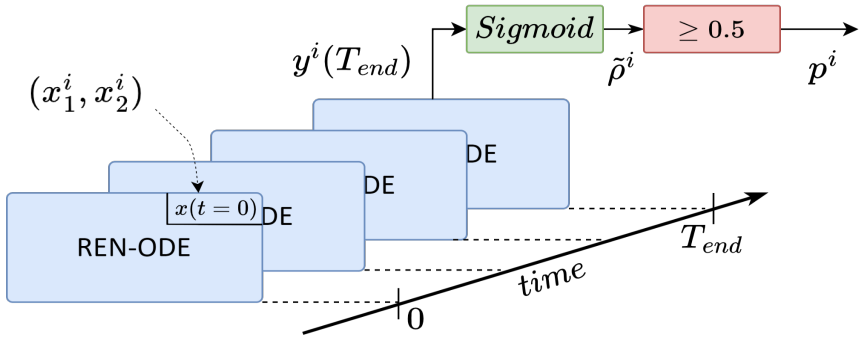
$\dot{x}_{\text{SYS}}(t) = \text{SYS.forward}(t, x_{\text{SYS}}, u_{\text{RENODE}}(t))$ ▷ state evolution of SYS

return $[\dot{x}_{RENODE}\ ,\ \dot{x}_{SYS}]$ ▷ Return the derivative of both states

Binary Classification

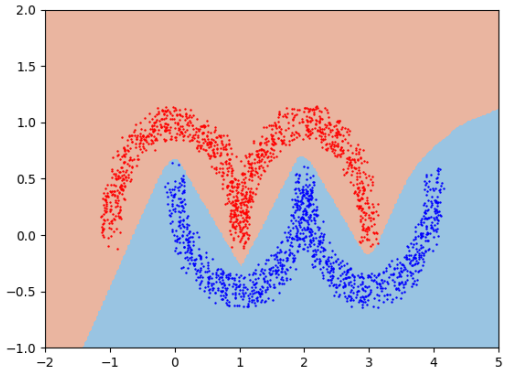
Classification Problem:

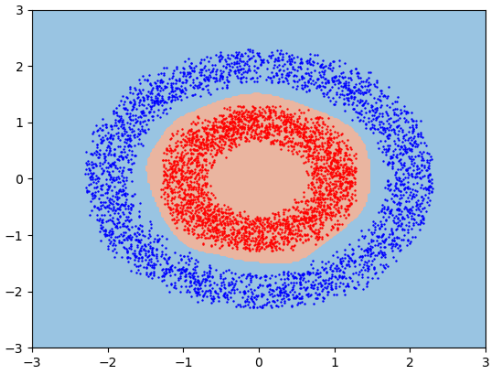
- **Input**: 2 features
- **Label**: binary { 0, 1 }

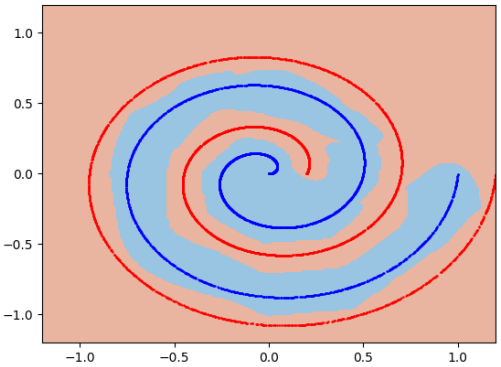


Loss Function:
Binary Classification Entropy (BCE):

$$\ell(\tilde{p}, \hat{y}) = -(\hat{y}\log(\tilde{p}) + (1-\hat{y})\log(1-\tilde{p}))$$

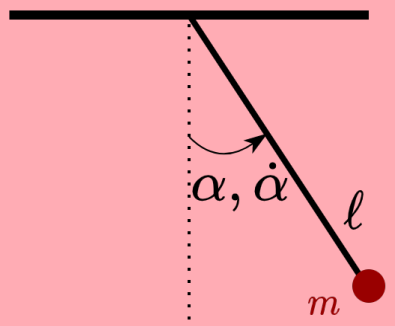




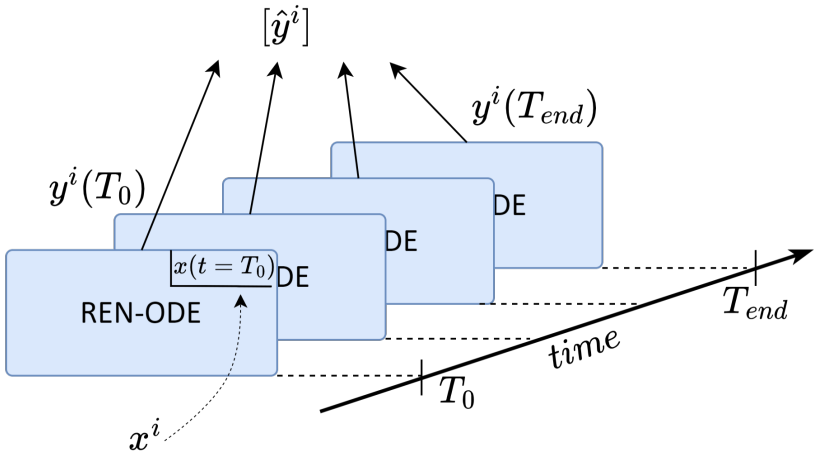


System Identification (SYSID)

Nonlinear Pendulum:



$$\ell\ddot{\alpha}(t) + \beta\dot{\alpha}(t) + g\sin\alpha(t) = 0$$

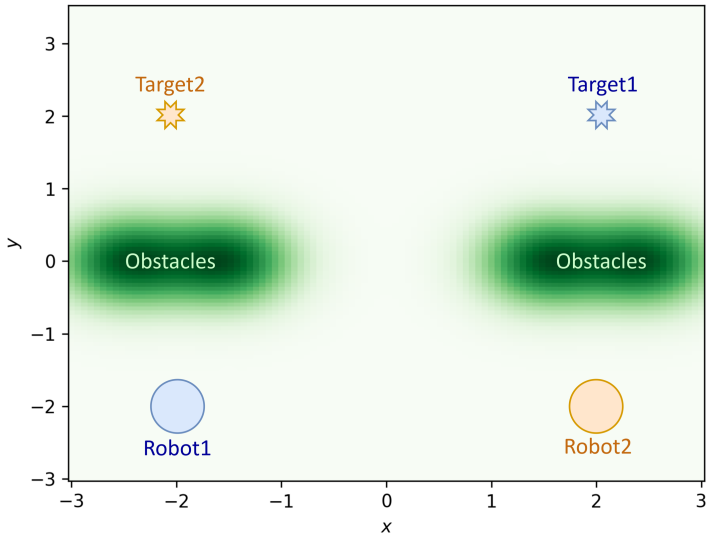


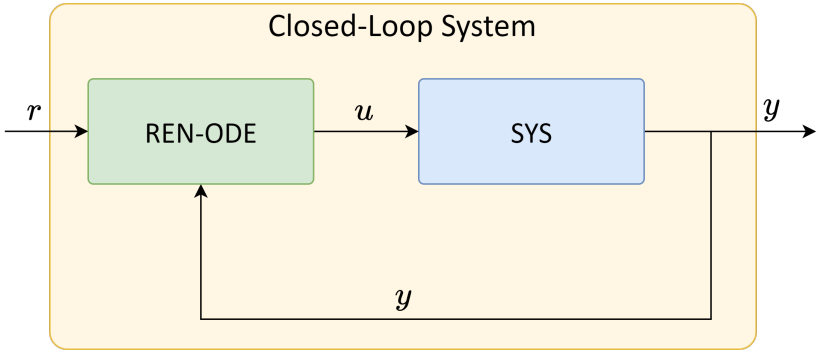
Loss Function:

Mean Squared Error (MSE):

$$L(y,\hat{y}) = MSE(y,\hat{y}) = \frac{1}{\eta}\sum_{i=0}^{\eta}\sum_{t=T_0}^{T_{end}}\left\|y^i(t) - \hat{y}^i(t)\right\|^2$$

Optimal Control





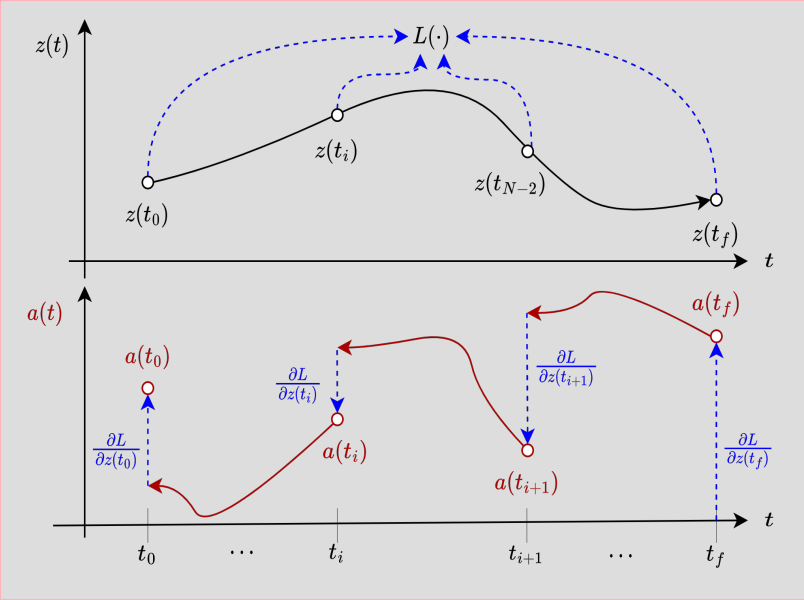
Optimal Control Policy (loss function):

$$L = \sum_{k=0}^{T_{end}/T_s} l_{traj}(x_{t_k}, u_{t_k}) + l_{ca}(x_{t_k}) + l_{obst}(x_{t_k})$$

Neural-ODE

Training Phase

- Gradient are computed by solving another ODE from $T_{end} \rightarrow t_0$.
- Don't backpropagate through the operations of the solver.
- No need to store activations, O(1) memory gradient



$$a(t) = \frac{\partial L}{\partial z(t)} \longrightarrow \frac{d}{dt}a(t) = -a(t)^{\mathsf{T}} \frac{\partial f(z(t), t, \theta)}{\partial z}$$

Assumptions

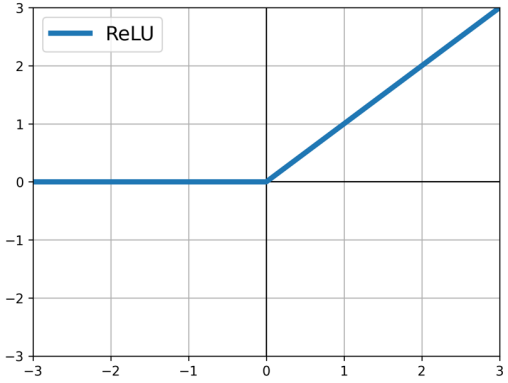
Assumption on activation function $\sigma(\cdot)$:

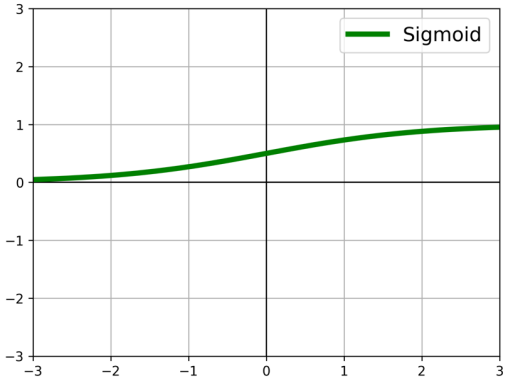
$$0 \leq \frac{\sigma(y)-\sigma(x)}{y-x} \leq 1$$

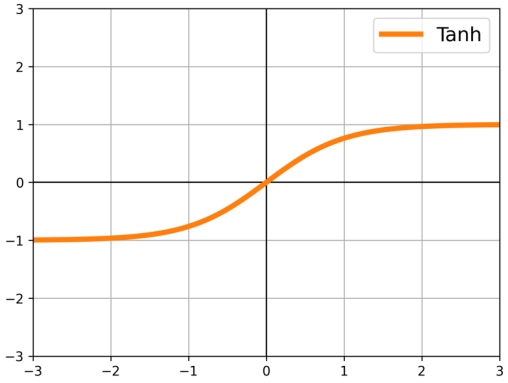
Conic Combination

$$\Gamma_t = \begin{bmatrix} \Delta v_t \\ \Delta w_t \end{bmatrix}^{\top} \begin{bmatrix} 0 & \Lambda \\ \Lambda & -2\Lambda \end{bmatrix} \begin{bmatrix} \Delta v_t \\ \Delta w_t \end{bmatrix} \geq 0,$$

Popular activation functions satisfy it already! (e.g., $ReLU(\cdot), Sigmoid(\cdot), \tanh(\cdot)$ ✓)







Dissipative Direct Parametrization

Dissipativity matrix inequality

$$\begin{bmatrix} -A^\top P - PA & -C_1^\top \Lambda - PB_1 & -PB_2 + C_2^\top S^\top \\ -\Lambda C_1 - B_1^\top P & W & D_{21}^\top S^\top - \Lambda D_{12} \\ -B_2^\top P + SC_2 & SD_{21} - D_{21}^\top \Lambda & R + SD_{22} + D_{22}^\top S^\top \end{bmatrix} + \begin{bmatrix} C_2^\top \\ D_{21}^\top \\ D_{22}^\top \end{bmatrix} Q \begin{bmatrix} C_2^\top \\ D_{21}^\top \\ D_{22}^\top \end{bmatrix}^\top \begin{matrix} Q < 0 \\ > 0 \end{matrix}$$

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} = X^\top X + \epsilon I \quad > 0 \text{ always!}$$

How we obtain from H a RNodeREN?

I) Pick free parameters

$X, \cancel{B_2}, C_2, \cancel{D_{12}}, \cancel{D_{21}}, \cancel{D_{22}}, b_x, b_y, b_v, \ldots$

II) Build matrix H

$H = X^\top X + \epsilon I$

III) Retrieve remaining matrices using H

$A, B_1, B_2, C_1, D_{11}, D_{12}, D_{22}$