

MODEL STRUCTURES AND FITTING CRITERIA FOR SYSTEM IDENTIFICATION WITH NEURAL NETWORKS

Marco Forgione, Dario Piga

IDSIA Dalle Molle Institute for Artificial Intelligence SUPSI-USI, Lugano, Switzerland

14th IEEE International Conference on Application of Information and
Communication Technologies

Motivations

Neural Networks are commonly used to model dynamical systems. However, they seldom exploit available **a priori** knowledge.

In this work:

- We present **tailor-made model structures** for system identification with Neural Networks
- We develop **efficient algorithms** to fit these model structures to data.

Motivations

Neural Networks are commonly used to model dynamical systems. However, they seldom exploit available **a priori** knowledge.

In this work:

- We present **tailor-made model structures** for system identification with Neural Networks
- We develop **efficient algorithms** to fit these model structures to data.

Settings

The **data-generating system** S_o is assumed to have the discrete-time state-space representation:

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

$$y_k^o = g(x_k)$$

$$y_k = y_k^o + e_k$$

Training dataset \mathcal{D} consisting of N input samples $U = \{u_0, u_1, \dots, u_{N-1}\}$ and output samples $Y = \{y_0, y_1, \dots, y_{N-1}\}$ available.

Objective: estimate a dynamical model of S_o .

Neural Dynamical Models

A very **generic** neural model structure:

$$\begin{aligned}x_{k+1} &= \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= \mathcal{N}_g(x_k; \theta)\end{aligned}$$

where \mathcal{N}_f , \mathcal{N}_g are feed-forward neural networks. Can be **specialized**:

- Linear approximation available \Rightarrow

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= Cx_k + \mathcal{N}_g(x_k, u_k; \theta)\end{aligned}$$

- State fully observed \Rightarrow

$$\begin{aligned}x_{k+1} &= \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= x_k\end{aligned}$$

Neural Dynamical Models

A very **generic** neural model structure:

$$\begin{aligned}x_{k+1} &= \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= \mathcal{N}_g(x_k; \theta)\end{aligned}$$

where \mathcal{N}_f , \mathcal{N}_g are feed-forward neural networks. Can be **specialized**:

- Linear approximation available \Rightarrow

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= Cx_k + \mathcal{N}_g(x_k, u_k; \theta)\end{aligned}$$

- State fully observed \Rightarrow

$$\begin{aligned}x_{k+1} &= \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= x_k\end{aligned}$$

Neural Dynamical Models

A very **generic** neural model structure:

$$\begin{aligned}x_{k+1} &= \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= \mathcal{N}_g(x_k; \theta)\end{aligned}$$

where \mathcal{N}_f , \mathcal{N}_g are feed-forward neural networks. Can be **specialized**:

- Linear approximation available \Rightarrow

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= Cx_k + \mathcal{N}_g(x_k, u_k; \theta)\end{aligned}$$

- State fully observed \Rightarrow

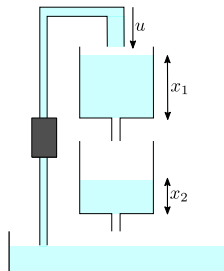
$$\begin{aligned}x_{k+1} &= \mathcal{N}_f(x_k, u_k; \theta) \\ y_k &= x_k\end{aligned}$$

Neural Dynamical Models

Physics-inspired model structures

Two-tank system, input=flow u in upper tank, output=lower tank level x_2 .

- The system has two states: x_1 and x_2
- The state x_1 does not depend on x_2
- The state x_2 does not depend directly on u
- The state x_2 is observed



The observations above lead to the neural physics-inspired model structure

$$\dot{x}_1 = \mathcal{N}_1(x_1, u)$$

$$\dot{x}_2 = \mathcal{N}_2(x_1, x_2)$$

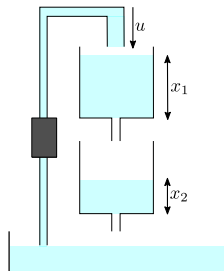
$$y = x_2$$

Neural Dynamical Models

Physics-inspired model structures

Two-tank system, input=flow u in upper tank, output=lower tank level x_2 .

- The system has two states: x_1 and x_2
- The state x_1 does not depend on x_2
- The state x_2 does not depend directly on u
- The state x_2 is observed



The observations above lead to the neural **physics-inspired** model structure

$$\dot{x}_1 = \mathcal{N}_1(x_1, u)$$

$$\dot{x}_2 = \mathcal{N}_2(x_1, x_2)$$

$$y = x_2$$

Training Neural Dynamical Models

In principle, **simulation error minimization** is a valid strategy:

$$\theta^o = \arg \min_{\theta, \hat{x}_0} \sum_{k=0}^{N-1} \|\hat{y}_k(\theta, \hat{x}_0) - y_k\|^2$$

where

$$\begin{aligned}\hat{x}_{k+1} &= \mathcal{N}_f(\hat{x}_k, u_k; \theta) \\ \hat{y}_k &= \mathcal{N}_g(\hat{x}_k; \theta)\end{aligned}$$

for $k = 0, 1, \dots, N - 1$.

However, it is not convenient from a computational perspective:

- Simulation is **not parallelizable**. Several neural network evaluations have to be performed **sequentially**.
- **Back-propagation cost** increases with the sequence length

In this work, we minimize instead the simulation error over **batches** of q subsequences, each one of length $m \ll N$.

Training Neural Dynamical Models

In principle, **simulation error minimization** is a valid strategy:

$$\theta^o = \arg \min_{\theta, \hat{x}_0} \sum_{k=0}^{N-1} \|\hat{y}_k(\theta, \hat{x}_0) - y_k\|^2$$

where

$$\begin{aligned}\hat{x}_{k+1} &= \mathcal{N}_f(\hat{x}_k, u_k; \theta) \\ \hat{y}_k &= \mathcal{N}_g(\hat{x}_k; \theta)\end{aligned}$$

for $k = 0, 1, \dots, N - 1$.

However, it is not convenient from a computational perspective:

- Simulation is **not parallelizable**. Several neural network evaluations have to be performed **sequentially**.
- **Back-propagation cost** increases with the sequence length

In this work, we minimize instead the simulation error over **batches** of q subsequences, each one of length $m \ll N$.

Training Neural Dynamical Models

In principle, **simulation error minimization** is a valid strategy:

$$\theta^o = \arg \min_{\theta, \hat{x}_0} \sum_{k=0}^{N-1} \|\hat{y}_k(\theta, \hat{x}_0) - y_k\|^2$$

where

$$\begin{aligned}\hat{x}_{k+1} &= \mathcal{N}_f(\hat{x}_k, u_k; \theta) \\ \hat{y}_k &= \mathcal{N}_g(\hat{x}_k; \theta)\end{aligned}$$

for $k = 0, 1, \dots, N - 1$.

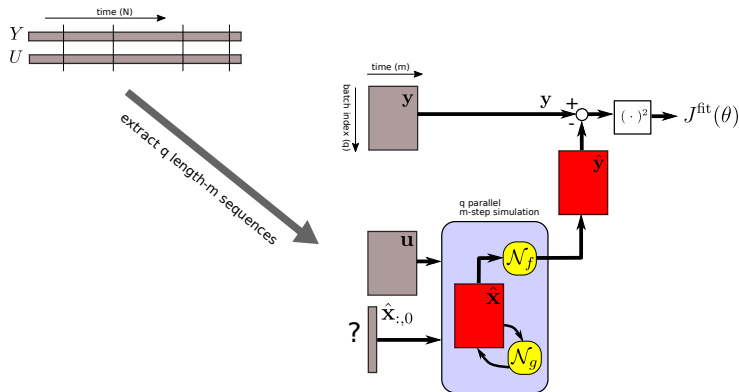
However, it is not convenient from a computational perspective:

- Simulation is **not parallelizable**. Several neural network evaluations have to be performed **sequentially**.
- **Back-propagation cost** increases with the sequence length

In this work, we minimize instead the simulation error over **batches** of q **subsequences**, each one of length $m \ll N$.

Training Neural Dynamical Models

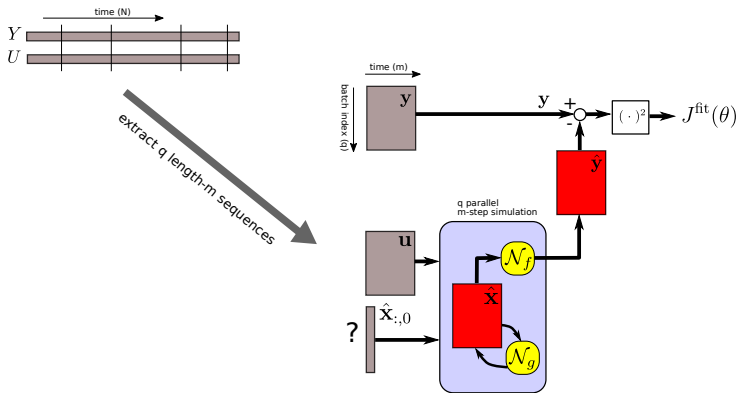
for each iteration of **gradient-based** optimization:



Problem: how do we choose $\hat{x}_{:,0}$, the initial state for each batch?
We do not know it, but we need it to initialize all simulations.

Training Neural Dynamical Models

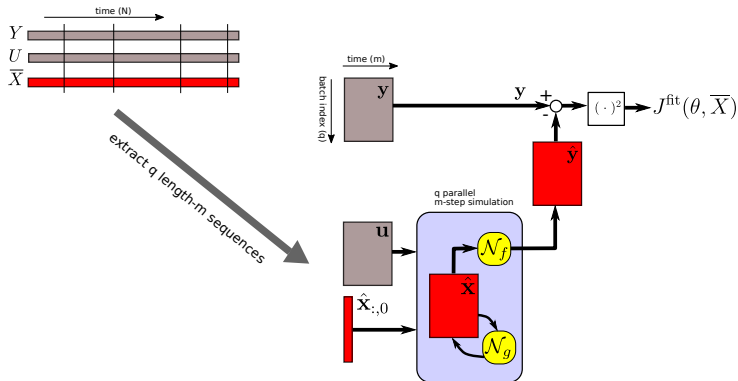
for each iteration of **gradient-based** optimization:



Problem: how do we choose $\hat{x}_{:,0}$, the initial state for each batch?
We do not know it, but we need it to initialize all simulations.

Training Neural Dynamical Models

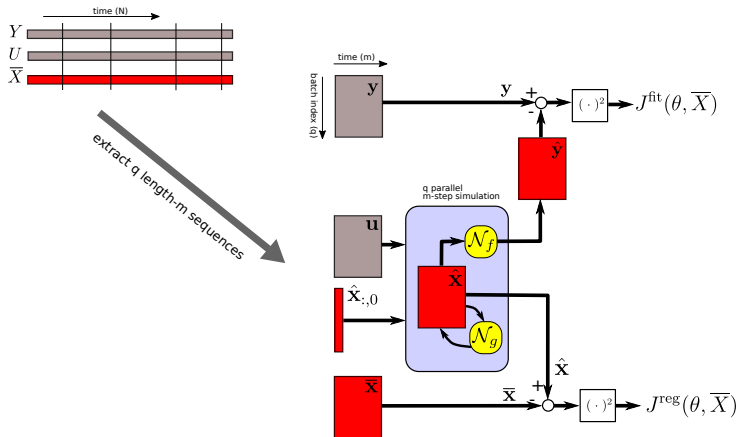
We consider the unknown state sequence \bar{X} as an **optimization variable**.
We sample from \bar{X} to obtain the initial state for simulation in each batch.



J^{fit} is now a function of both θ and \bar{X} . We optimize w.r.t. both!

Training Neural Dynamical Models

The hidden state sequence \bar{X} should also satisfy the identified dynamics!
We enforce this by adding a **regularization term** in the cost function.



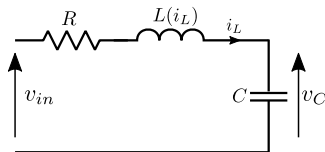
We minimize a **weighted sum** of J^{fit} and J^{reg} w.r.t. both θ and \bar{X} .

Simulation example

RLC circuit

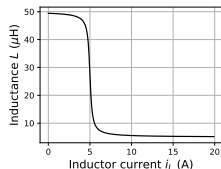
We consider a nonlinear RLC circuit:

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ \frac{-R}{L(i_L)} & 0 \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L(i_L)} \end{bmatrix} v_{in}$$



with nonlinear inductance $L(i_L)$

$$L(i_L) = L_0 \left[\left(\frac{0.9}{\pi} \arctan(-5(|i_L| - 5)) + 0.5 \right) + 0.1 \right]$$



Input: voltage v_{in} . Output: voltage v_C , current i_L . SNR=20

Neural model structure: fully observed state

$$x_{k+1} = \mathcal{N}_f(x_k, u_k; \theta)$$

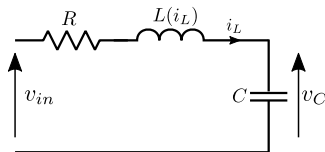
$$y_k = x_k$$

Simulation example

RLC circuit

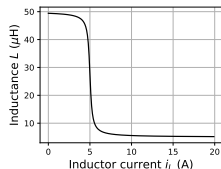
We consider a nonlinear RLC circuit:

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ \frac{-R}{L(i_L)} & \frac{1}{L(i_L)} \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L(i_L)} \end{bmatrix} v_{in}$$



with nonlinear inductance $L(i_L)$

$$L(i_L) = L_0 \left[\left(\frac{0.9}{\pi} \arctan(-5(|i_L|-5)) + 0.5 \right) + 0.1 \right]$$



Input: voltage v_{in} . Output: voltage v_C , current i_L . SNR=20

Neural model structure: fully observed state

$$x_{k+1} = \mathcal{N}_f(x_k, u_k; \theta)$$

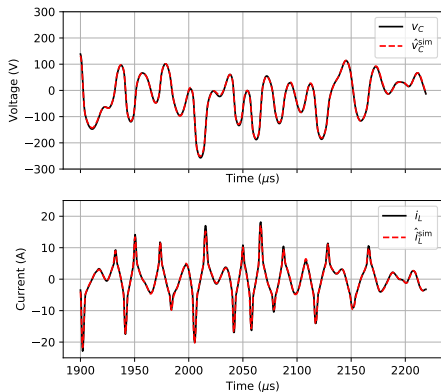
$$y_k = x_k$$

Numerical example

RLC circuit

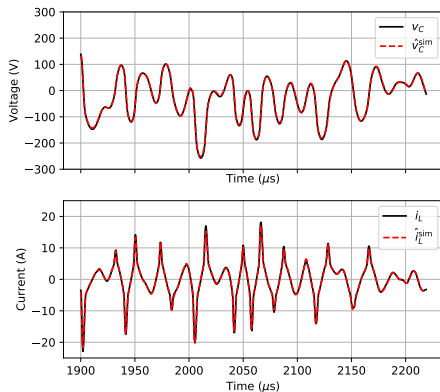
Results in simulation on the test dataset. Training with:

$q = 62$ sequences of length $m = 64$



train time: 320 s

simulation error minimization

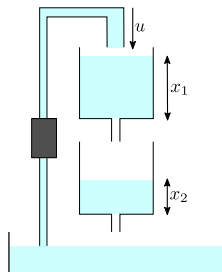
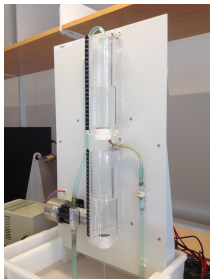


train time: 7000 s

Numerical example

Cascaded Tank System

Dataset with **real measurements** from `www.nonlinearbenchmark.org`



Neural model structure: **physics-inspired**

$$\dot{x}_1 = \mathcal{N}_1(x_1, u)$$

$$\dot{x}_2 = \mathcal{N}_2(x_1, x_2, \textcolor{red}{u})$$

$$y = x_2$$

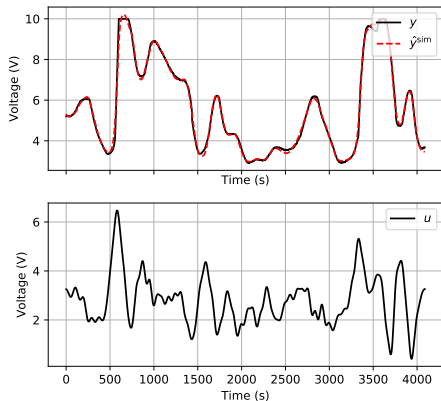
The dependency of \mathcal{N}_2 on u models **water overflow** from upper tank.

Numerical example

Cascaded Tank System

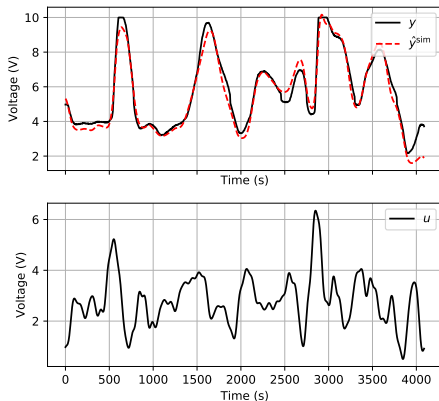
Training with $m = 128, q = 64$. Results on:

Training dataset



$$R^2 = 0.99, \text{RMSE} = 0.08 \text{ V}$$

Test dataset



$$R^2 = 0.97, \text{RMSE} = 0.33 \text{ V}$$

Conclusions

We have presented **tailor-made** neural structures for system identification embedding a priori knowledge.

We have shown how to parallelize the training using batches of short-size **subsequences**, and taking into account the effect of the **initial condition**.

Current/Future work

- Extension to the continuous-time setting
- Learning of Partial Differential Equations

Conclusions

We have presented **tailor-made** neural structures for system identification embedding a priori knowledge.

We have shown how to parallelize the training using batches of short-size **subsequences**, and taking into account the effect of the **initial condition**.

Current/Future work

- Extension to the continuous-time setting
- Learning of Partial Differential Equations

Thank you.
Questions?

`marco.forgione@idsia.ch`