Continuous-time system identification with neural networks: model structures and fitting criteria

Marco Forgione and Dario Piga

IDSIA Dalle Molle Institute for Artificial Intelligence USI-SUPSI,
Via la Santa 1, CH-6962 Lugano-Viganello, Switzerland.

April 16, 2021

Abstract

This paper presents tailor-made neural model structures and two custom fitting criteria for learning dynamical systems. The proposed framework is based on a representation of the system behavior in terms of continuous-time state-space models. The sequence of hidden states is optimized along with the neural network parameters in order to minimize the difference between measured and estimated outputs, and at the same time to guarantee that the optimized state sequence is consistent with the estimated system dynamics. The effectiveness of the approach is demonstrated through three case studies, including two public system identification benchmarks based on experimental data.

To cite this work, please use the following bibtex entry:

```
@article{forgione2021a,
  title={Continuous-time system identification
  with neural networks: Model structures and fitting criteria},
  author={Forgione, M. and Piga, D.},
  journal={European Journal of Control},
  volume={59},
  pages={69--81},
  year={2021},
  publisher={Elsevier}
}
```

Using the plain bibtex style, the bibliographic entry should look like:

M. Forgione and D. Piga. *dynoNet*: A neural network architecture for learning dynamical systems. *European Journal of Control*, 59:69–81, 2021.

---

1 Introduction

In recent years, deep learning has advanced at a tremendous pace and is now the core methodology behind cutting-edge technologies such as speech recognition, image classification and captioning, language translation, and autonomous driving (**???**). These impressive achievements are attracting ever increasing investments both from the private and the public sector, fueling further research in this field.

A good deal of the advancement in the deep learning area is publicly accessible, both in terms of scientific publications and software tools. For instance, highly optimized and user-friendly deep learning frameworks are readily available (**??**), and are often distributed under permissive open-source licenses. Using the high-level functionalities of a deep learning framework and following good practice, even a novice user can tackle *standard* machine learning tasks (once considered extremely hard) such as image classification with moderate effort.

An experienced practitioner can employ the same deep learning framework at a lower level to tackle non-standard learning problems, by defining customized models and objective functions to be optimized, and using operators such as *neural networks* as building blocks. The practitioner is free from the burden of writing optimization code from scratch for every particular problem, which would be tedious and error-prone. In fact, as a built-in feature, modern deep learning engines can compute the derivatives of a supplied objective function with respect to free tunable parameters by implementing the celebrated *back-propagation* algorithm (**?**). In turn, this enables convenient setup of any gradient-based optimization method.

An exciting, challenging—and yet largely unexplored—application field is *system identification* with tailor-made model structures and fitting criteria. In this context, neural networks can be used to describe uncertain components of the dynamics, while retaining structural (physical) knowledge, if available. Furthermore, the fitting criterion can be specialized to take into account the modeler's ultimate goal, which could be prediction, failure detection, state estimation, control design, simulation, *etc.*

The choice of the cost function may also be influenced by computational considerations. In this paper, in particular, models are assessed in terms of their simulation performance. In this setting, from a theoretical perspective, simulation error minimization is generally the best fitting criterion. However, evaluating the simulation error loss and its derivative may be prohibitively expensive from a computational perspective for dynamical models containing neural networks. Furthermore, simulation over time has an intrinsically sequential nature and offers limited opportunities for parallelization.

In this paper, we present two fitting algorithms whose run-time is significantly faster than full simulation error minimization, but still provide models with high simulation performance.

In the first approach, called *truncated simulation error minimization*, the neural dynamical model is simulated over (mini)batches of *subsequences* extracted from the training dataset. This allows parallelization of the model sim-

ulation over the different subsequences in a batch and also results in reduced back-propagation cost with respect to a full open-loop simulation. Special care is taken to provide consistent initial conditions for all the simulated subsequences. In fact, these initial conditions are optimized along with the neural network parameters according to a dual objective. Specifically, the batch cost function takes into account both the distance between the simulated output and the measured output—the fitting objective—and the consistency of all the initial condition variables with the neural model equation—the initial state consistency objective. This cost function is iteratively minimized over the randomly extracted batches through a gradient-based optimization algorithm. Preliminary ideas of this approach are presented in **?** for discrete-time neural model structures and are extended here to the continuous-time case, with dynamics described in terms of Ordinary Differential Equations (ODE).

In the second approach, called *soft-constrained integration*, the neural dynamical model is enforced by a regularization term in the cost function penalizing the violation of a numerical ODE integration scheme applied to the system's (hidden) state variables. These state variables, together with the neural network parameters, are tuned with the dual objective of fitting the measured data and minimizing the penalty term associated with the violation of the numerical integration scheme. In the soft-constrained integration method, simulation through time is thus completely circumvented and the loss function splits up into independent contribution for each time step. This enables a fully parallel implementation of gradient-based optimization.

1.1 Related works

The use of neural networks in system identification has a long history. For instance, neural *AutoRegressive Moving Average with eXogenous inputs* models are discussed in **?**. Training is performed with the one-step prediction error method (**?**) previously developed in the context of linear dynamical systems. In (**?**), several *Recurrent Neural Network* structures trained by *Back-Propagation Through Time* are evaluated on a nonlinear system identification task. Minimization of the simulation error loss is performed over short training sequences, with known initial conditions. Although the overall reasoning in these earlier works is similar to our, their results are hardly comparable, given the huge gap of hardware/software technology.

More recently, a few interesting approaches exploiting modern deep learning concepts and tools for system identification have been proposed. For instance, **?** introduces a technique to identify neural state-space model structures using deep autoencoders for state reconstruction, while **?** and **?** discuss the use of *Long Short-Term Memory* (LSTM) recurrent neural networks for system identification. A recent comparative study on modern neural architectures for system identification is also presented in **?**.

Compared to these recent contributions, our work focuses on specialized neural model structures for the identification task at hand. Furthermore, all the above mentioned contributions consider discrete-time (DT) dynamical systems,

while we deal with the continuous-time (CT) case in this work.

Nowadays, direct identification of CT dynamical systems is an active research topic attracting the attention of many researchers in the system and control community. There are indeed multiple advantages in direct CT identification, as argued in **?**. First, the majority of physical systems are naturally modeled in a continuous-time framework. Embedding physical knowledge in continuous-time model structures is thus more intuitive, and inspecting a continuous-time identified model is more insightful as some parameters may retain a physical meaning. Second, continuous-time models can handle the case of non-uniformly sampled data. Last, continuous-time identification is generally immune from the numerical issues affecting discrete-time methodologies in the case of high sampling frequency. Complete reviews of direct CT identification algorithms and applications can be found in the papers **??????**, in the book **?**, and in the special issue **?**. However, the above contributions only focus on dynamical systems with linear relationships in the input and output signals, such as linear time-invariant, linear parameter-varying and linear time-varying systems. Identification of general nonlinear CT dynamical models is still a challenging and open problem.

A few contributions have tackled non-linear CT system identification using certain special neural model structures. For instance, in **?** an identification scheme for CT dynamical systems described in a non-linear observability canonical form is proposed, while **?** consider dynamic neural networks based on feedback linearization theory. Both methods require high-order derivatives of the system output for training, whose estimate could be problematic in the presence of measurement noise.

The connection between deep learning and dynamical system theory is currently under intensive investigation, see *e.g.*, (**?**) and cross-contamination is yielding substantial advances to both fields. On the one hand, certain modern deep learning architectures are now interpreted as discrete-time approximations of an underlying continuous-time neural dynamical system. Equivalently, continuous-time neural network architectures like the ones considered for modeling in this paper are seen as the limit case of deep learning architectures for an infinite number of layers. This is discussed, for instance, in the introduction of **?**. Exploiting this parallel, (**?**) and (**?**) analyze the stability properties of existing deep neural architectures through the lens of system theoretic tools. Modified architectures guaranteeing stability *by design* are also proposed. On the other hand, contributions such as **???** showcase the potential of neural networks for data-driven modeling of dynamical systems described by ordinary and partial differential equations. With respect to these contributions, our aim is to devise computationally efficient fitting strategies for neural dynamical models that are robust to the measurement noise.

1.2 Paper structure

The rest of this paper is structured as follows. The overall settings and problem statement is outlined in Section 2. The neural dynamical model structures are introduced in Section 3 and criteria for fitting these model struc-

tures to training data are described in Section 4. Simulation results are presented in Section 5 and can be replicated using the codes available at `https://github.com/forgi86/sysid-neural-continuous`. Conclusions and directions for future research are discussed in Section 6.

2 Problem Setting

We are given a training dataset $\mathcal{D}$ consisting of N sequential input samples $U = \{u_0,\ u_1, \dots,\ u_{N-1}\}$ and output samples $Y = \{y_0,\ y_1, \dots,\ y_{N-1}\}$, gathered at time instants $T = \{t_0 = 0,\ t_1,\ \dots,\ t_{N-1}\}$ from an experiment on a dynamical system $\mathcal{S}$. The data-generating system $\mathcal{S}$ is assumed to have the continuous-time state-space representation

$$\dot{x}(t) = f(x(t), u(t)) \tag{1a}$$

$$y^{\mathrm{o}}(t) = g(x(t)), \tag{1b}$$

where $x(t) \in \mathbb{R}^{n_x}$ is the system state at time t; $\dot{x}(t)$ denotes the time derivative of $x(t)$; $y^{\mathrm{o}}(t) \in \mathbb{R}^{n_y}$ is the noise-free output; $u(t) \in \mathbb{R}^{n_u}$ is the system input; $f(\cdot, \cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ and $g(\cdot) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ are the state and output mappings, respectively. The measured output y_k at time instant t_k, $k = 0, 1, \dots, N-1$, is corrupted by a zero-mean additive noise η_k, *i.e.*, $y_k = y^{\mathrm{o}}(t_k) + \eta_k$. We assume that the input $u(t)$ can be reconstructed (or reasonably approximated) for all time instants $t \in [t_0\ t_{N-1}] \subset \mathbb{R}$ from the samples U.

Our objective is to estimate from $\mathcal{D}$ a *simulation* model M of the unknown dynamical system $\mathcal{S}$, *i.e.*, the model M, given the input sequence $u(t)$, should generate an output sequence $y^{\mathrm{sim}}(t)$ close to the true system output $y^{\mathrm{o}}(t)$.[1]

To be of practical use, the model should *generalize* to conditions that are not encountered in the training dataset. For this reason, we shall evaluate the model performance in terms of the "discrepancy" between $y(t)$ and $y^{\mathrm{sim}}(t)$ on an independent test dataset, where the system (1) is driven by an input sequence $u(t)$ different from the one used for training. Specifically, metrics such as the Root Mean Squared Error (RMSE) and the R^2 index shall be considered.

The dynamic nature of the system at hand calls for specialized models and learning techniques that can deal with the inherently sequential nature of the training sequences. Furthermore, these techniques have to be tailored for the considered simulation objective. In this paper, we introduce flexible neural model structures $\mathcal{M}$ that are suitable to represent generic dynamical systems as (1), allowing the modeler to embed domain knowledge to various degrees and to exploit neural networks to describe unknown model components. Furthermore, we present fitting criteria and algorithms to train these neural dynamical model structures, namely to select a model M within the model structure $\mathcal{M}$ with good simulation performance, using the training dataset $\mathcal{D}$.

Overall, we aim at fitting a neural network model that exploits the dynamic constraints and different forms of available prior knowledge of the data-generating system (1).

[1] We are not considering in this work *prediction* models, which may also exploit previous *measured* output values to predict $y^{\mathrm{o}}(t)$.

3 Neural dynamical models

Let us consider a *model structure* $\mathcal{M} = \{M(\theta),\ \theta \in \mathbb{R}^{n_\theta}\}$, where $M(\theta)$ represents a dynamical model parametrized by a real-valued vector θ. We refer to *neural model structures* as structures $\mathcal{M}$ where some components of the model $M(\theta)$ are described by neural networks. In the following, we introduce possible neural model structures $\mathcal{M}$ for dynamical systems.

3.1 General state-space model

A general *state-space neural model structure* has form

$$\dot{x} = \mathcal{N}_f(x, u; \theta) \tag{2a}$$
$$y^{\text{o}} = \mathcal{N}_g(x; \theta), \tag{2b}$$

where $\mathcal{N}_f$ and $\mathcal{N}_g$ are fully connected, feedforward neural networks **?** of compatible size parametrized by $\theta \in \mathbb{R}^{n_\theta}$. For notational convenience, the time dependence of all signals in (2) is omitted.[2]

For the sake of exposition, in the case of neural networks $\mathcal{N}_f$ and $\mathcal{N}_g$ with a single hidden layer, the network $\mathcal{N}_f$ is characterized by an input layer with $n_x + n_u$ units corresponding to the system state x and system input u; a certain number of hidden layers; and a linear output layer containing n_x units corresponding to state derivatives. The network $\mathcal{N}_g$ has an input layer with n_x units corresponding to the states x; a certain number of hidden layers; and a linear output layer with n_y units corresponding to the system outputs. Thus, the single-hidden-layer neural networks $\mathcal{N}_f$ and $\mathcal{N}_g$ have structure

$$\mathcal{N}_f(x, u; \theta) = \mathcal{W}_{f,2}\left(\varepsilon(\mathcal{W}_{f,1}[x\ u]^\top + \beta_{f,1}) + \beta_{f,2}\right) \tag{3a}$$
$$\mathcal{N}_g(x; \theta) = \mathcal{W}_{g,2}\left(\varepsilon(\mathcal{W}_{g,1}x + \beta_{g,1}) + \beta_{g,2}\right). \tag{3b}$$

In (3), the weight matrices $\mathcal{W}_{f,2}$, $\mathcal{W}_{f,1}$, $\mathcal{W}_{g,2}$, $\mathcal{W}_{g,1}$, and bias vectors $\beta_{f,2}$, $\beta_{f,1}$, $\beta_{g,2}$, $\beta_{g,1}$ are the tunable model parameters collectively represented in the parameter vector $\theta \in \mathbb{R}^{n_\theta}$ for notational compactness, while $\varepsilon(\cdot)$ is the neural network's non-linear *activation function*, which is applied *element-wise* to each element of its (matrix) input, producing an output of the same dimensionality. Note that the number of rows of $\mathcal{W}_{f,1}$ must be equal to the number of columns of $\mathcal{W}_{f,2}$ (and to the number of elements in the vector $\beta_{f,1}$) and corresponds to the number of hidden units in the (single) hidden layer of $\mathcal{N}_f$. Similar dimensionality constraints apply to the weight matrices and bias terms of the neural network $\mathcal{N}_g$.

In the following, fully connected feedforward neural networks are always denoted with the letter $\mathcal{N}$ (using different subscripts to distinguish among them) and referred to in short as "neural networks". The number of input and output layers of these neural network should be clear to the reader from context. The

[2] In the rest of the paper, the time dependence of signals will be specified only when necessary.

exact network structure (number of hidden layers, number of hidden units per layer, type of activation function) is only specified for the examples in Section 5 and not in the main sections of the paper, as it is not required in the general discussion.

The general neural model structure (2) can be tailored for the identification task at hand. Examples are illustrated in the remainder of this section.

3.2 Incremental model

If a linear approximation of the system is available, an appropriate model structure could be

$$\dot{x} = A_L x + B_L u + \mathcal{N}_f(x, u; \theta) \tag{4a}$$
$$y^o = C_L x + \mathcal{N}_g(x; \theta), \tag{4b}$$

where A_L, B_L, and C_L are matrices of compatible size describing the linear system approximation. For example, the values of these matrices can be estimated from the available training dataset $\mathcal{D}$ using well-established algorithms for linear system identification **??**. Although model (4) is not more general than (2), it could be easier to train as the neural networks $\mathcal{N}_f$ and $\mathcal{N}_g$ are supposed to capture only residual (nonlinear) dynamics.

3.3 Fully-observed state model

If the system state is known to be fully observed, the most convenient representation is

$$\dot{x} = \mathcal{N}_f(x, u; \theta) \tag{5a}$$
$$y^o = x, \tag{5b}$$

where only the state mapping neural network $\mathcal{N}_f$ is learned, while the output mapping is fixed to identity.

3.4 Physics-based model

Tailor-made architectures could be used to embed specific physical knowledge in the neural model structure.

For instance, let us consider the *Cascaded Tanks System* (CTS) schematized in Figure 1. The CTS is a fluid level control system consisting of two tanks with free outlets fed by a pump. Water is pumped from a bottom reservoir into the upper tank by a controlled pump. The water in the upper tank flows through a small opening into the lower tank, and from another small opening from the lower tank to the reservoir.

The system input u is the water flow from the bottom reservoir feeding the upper tank. The state variables x_1 and x_2 are the water level in the upper and

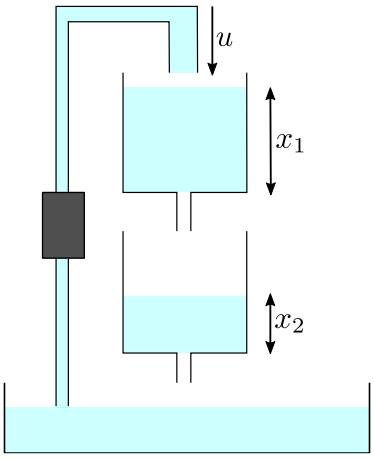


Figure 1: Schematics of the cascaded two-tank system.

lower tanks, respectively.[3] As in the illustrative example further discussed in Section 5, we consider the case where only the second state x_2 is measured.

The following dynamical model for the CTS can be derived from conservation laws and Bernoulli's principle **?**:

$$\dot{x}_1 = -k_1\sqrt{x}_1 + k_4 u \tag{6a}$$
$$\dot{x}_2 = k_2\sqrt{x}_1 - k_3\sqrt{x_2} \tag{6b}$$
$$y = x_2. \tag{6c}$$

Based on this physical knowledge, an appropriate neural dynamical model for the CTS is

$$\dot{x}_1 = \mathcal{N}_{f_1}(x_1, u) \tag{7a}$$
$$\dot{x}_2 = \mathcal{N}_{f_2}(x_1, x_2) \tag{7b}$$
$$y = x_2 \tag{7c}$$

capturing the information that: (i) the system dynamics can be described by a two-dimensional state-space model; (ii) the state x_2 is measured; (iii) the state x_1 does not depend on x_2; and (iv) the state x_2 does not depend directly on u, given the current value of x_1. This neural model takes advantage of the available process knowledge, while leaving representational capabilities to describe unmodeled effects such as fluid viscosity, nonlinearities of the actuators, and water overflow that may happen when the tanks are completely filled.

Embedding physical knowledge in neural model structures is a very active and promising trend in deep learning (**?**). For instance, recent contributions propose specialized structures that are suitable to describe systems satisfying

[3] With some abuse of notation, the subscripts in x_1 and x_2 simply denote the variable name and not a time index. The same notation will be used for the examples in Section 5.

general physical principles such a energy conservation. In these cases, a physics-based neural network may be used to learn the system's Hamiltonian or Lagrangian function, instead of the individual components its ODE representation as independent terms (**??**).

In general, including domain knowledge in the model structure is useful to restrict the search space, while leaving enough representation capacity. Therefore, a better generalization performance is expected from the trained models. Furthermore, the estimated models exactly satisfy (by design) the prior assumptions and thus they are generally easier to diagnose, interpret and exploit for advanced tasks such as state estimation, fault detection and closed-loop control.

4 Training neural dynamical models

In this section, we present algorithms aimed at fitting the model structures introduced in Section 3 to the training dataset $\mathcal{D}$.

4.1 Simulation error minimization

For *fixed* values of neural network parameters θ, for *given* initial condition $x_0 = x(0)$, and under the model structure (2), the open-loop state simulation $x^{\text{sim}}(t)$ is the solution of the Cauchy problem:

$$\dot{x}^{\text{sim}}(t) = \mathcal{N}_f\big(x^{\text{sim}}(t), u(t);\ \theta\big) \tag{8a}$$

$$x^{\text{sim}}(0) = x_0, \tag{8b}$$

and the simulated output $y^{\text{sim}}(t)$ is

$$y^{\text{sim}}(t;\ \theta, x_0) = \mathcal{N}_g(x^{\text{sim}}(t;\ \theta, x_0);\ \theta). \tag{9}$$

Different ODE solution schemes (**?**), may be applied to numerically solve problem (8). Formally,

$$x^{\text{sim}}(t;\ \theta, x_0) = \text{ODEINT}\left(t;\ \mathcal{N}_f\left(\cdot,\ \cdot;\ \theta\right), u(\cdot), x_0\right) \tag{10}$$

will represent the solution of the Cauchy problem (8) obtained using a numerical scheme of choice (explicit or implicit, single-step or multi-step, single-stage or multi-stage) denoted as ODEINT.

According to the simulation error minimization criterion, the neural network parameters θ can be obtained by minimizing the simulation error norm

$$J(\theta, x_0) = \frac{1}{N}\sum_{k=0}^{N-1} \left|\left| \overbrace{y^{\text{sim}}(t_k;\ \theta, x_0) - y_k}^{e_k} \right|\right|^2 \tag{11}$$

with respect to both the network parameters θ and the state initial condition x_0, with y^{sim} defined by Equations (9)-(10).

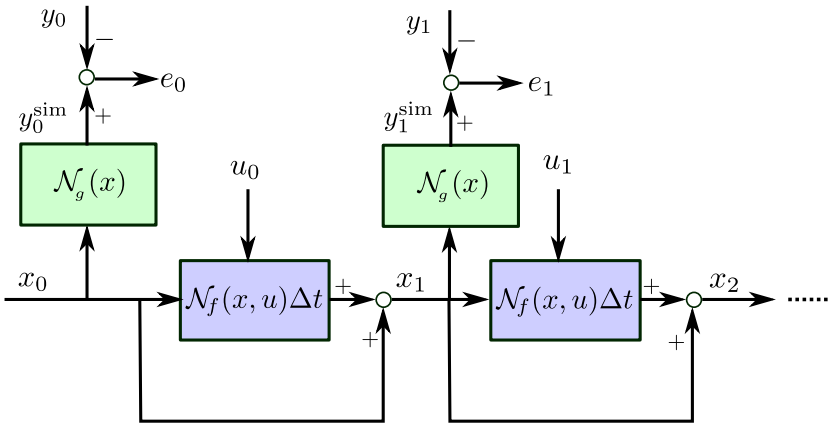


Figure 2: Computational graph associated with the forward Euler ODE integration scheme with constant step size Δt applied to system (2). Measurements are assumed to be equally spaced with the same constant rate Δt.

From an implementation perspective, when ODEINT is an explicit ODE solver, the full computational graph producing the cost function (11) can be constructed using standard differentiable blocks. For example, Figure 2 represents the computational graph obtained by applying a forward Euler scheme with constant step size Δt, assuming that the measurements in $\mathcal{D}$ are collected at the same rate Δt. It is interesting to note the similarity between this computational graph and the one of the *residual network* structure **?**. Note that number of repeated blocks in the computational graph (thus, the depth of the architecture) corresponds to the length of the training sequence.

Therefore, in the case of an explicit ODE solver, the derivatives of the loss $J(\theta, x_0)$ with respect to θ and x_0 required for gradient-based minimization can be obtained using standard back-propagation through the elementary solver steps. Thus, a procedure minimizing $J(\theta, x_0)$ can be implemented using available deep learning software.

Recently, an alternative approach to differentiate through the ODE solution based on backward time integration of *adjoint sensitivities* has been proposed **?**. Following this approach, implicit ODE solvers may be adopted as well.

In either case, from a computational perspective, simulating over time has an intrinsically sequential nature and offers scarce opportunities for parallelization. Thus, in practice, minimizing the simulation error with a gradient-based method over the entire training dataset $\mathcal{D}$ may be inconvenient or even unfeasible in terms of memory allocation and computational time.

Moreover, as thoroughly analyzed in **?**, numerical minimization of the simulation-error norm may be hard as its Lipschitz constant might blow up exponentially with the simulation length for certain system having for instance unstable or chaotic dynamics.

Remark 1 *In many cases, optimizing the initial condition x_0 is not necessary*

in simulation error minimization as this quantity may be available from physical considerations. For instance, in the cascaded tanks system presented above, the initial values of the tank levels may be known. Even when the initial condition is unknown, its effect may be negligible, as in the case for measurements collected from a fading memory *system $\mathcal{S}$ on a sufficiently long time horizon $[t_0\ t_{N-1}]$.*

Remark 2 *As in all non-convex optimization problems, a gradient-based optimization algorithm is not guaranteed to reach a global minimizer of the simulation error loss. This prevents us from deriving formal proofs of convergence for gradient-based simulation error minimization, as well as for all other algorithms presented in this paper. Nonetheless, this fundamental theoretical limitation does not seem to constitute a severe limitation for deep learning applications in practice. As discussed in (**?**, Ch. 8.2), among the stationary points of typical loss functions involving deep neural networks, local minimizers with a "high" cost (w.r.t. the global minimum of the loss) are rather uncommon compared, for instance, to saddle points. Empirically, gradient descent seems to be able to escape from these saddle points in many cases.*

4.2 Truncated simulation error minimization

In order to reduce the computational burden and circumvent the numerical issues possibly affecting full simulation error minimization approach previously discussed, the simulated output $y^{\mathrm{sim}}(t;\ \theta, x_0)$ can be obtained by simulating the system state $x^{\mathrm{sim}}(t;\ \theta, x_0)$ in (10) on several smaller portions of the training set $\mathcal{D}$.

For efficient implementation, the *truncated* simulation error minimization algorithm presented in this section processes *batches* containing q subsequences extracted from $\mathcal{D}$. In principle, the simulations can be carried out simultaneously for all the subsequences in the batch by exploiting parallel computing.

A batch is completely specified by a *batch starting index vector* $s \in \mathbb{N}^q$ defining the initial sample of each subsequence and a *sequence duration* $m \in \mathbb{N}$ defining the number of samples contained in each subsequence, where each element s_j of s satisfies $s_j \leq N-m-1$. Thus, for instance, the j-th output subsequence in a batch contains the measured output samples $\{y_{s_j}, y_{s_j+1}, \ldots, y_{s_j+m-1}\}$ (see Figure 3 for graphical representation).

For notational convenience, we can arrange the batch data in the following *tensors*:

- output $\boldsymbol{y} \in \mathbb{R}^{q\times m\times n_y}$, where $\boldsymbol{y}_{j,h} = y_{s_j+h}$
- input $\boldsymbol{u} \in \mathbb{R}^{q\times m\times n_u}$, where $\boldsymbol{u}_{j,h} = u_{s_j+h}$
- relative time $\boldsymbol{\tau} \in \mathbb{R}^{q\times m}$, where $\boldsymbol{\tau}_{j,h} = t_{s_j+h} - t_{s_j}$,

for $j = 0, 1, \ldots, q-1$ and $h = 0, 1, \ldots, m-1$. Furthermore, we define the tensors

- simulated state $\boldsymbol{x}^{\mathrm{sim}} \in \mathbb{R}^{q\times m\times n_x}$
- simulated output $\boldsymbol{y}^{\mathrm{sim}} \in \mathbb{R}^{q\times m\times n_y}$,

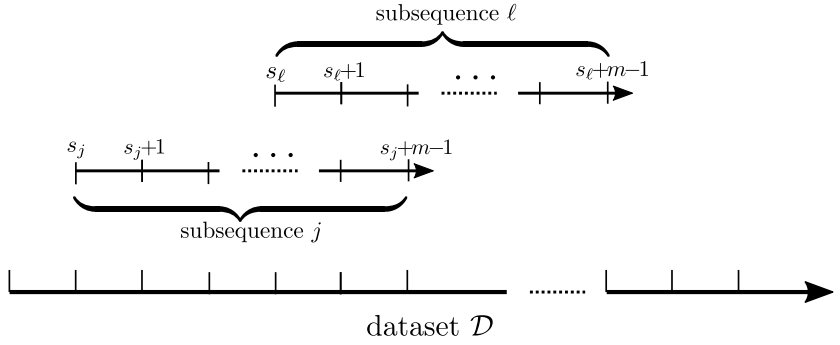


Figure 3: Representation of two subsequences of length m extracted from the training dataset $\mathcal{D}$. s_j and s_ℓ represent the starting indexes of subsequences j and ℓ, respectively.

that will be used to store simulated state and output values at the corresponding batch time instants.

Note that the third dimension of the output tensor $\boldsymbol{y} \in \mathbb{R}^{q \times m \times n_y}$ corresponds to the output channel. With some abuse of notation, the tensor $\boldsymbol{y}$ is addressed by only two indexes because we do not need to specify a particular output channel. The same notation is used for tensors $\boldsymbol{u}$, $\boldsymbol{x}^{\text{sim}}$, and $\boldsymbol{y}^{\text{sim}}$.

For the subsequences $j = 0, 1, \ldots, q-1$, the state evolution in the relative time intervals $[0 \;\; \boldsymbol{\tau}_{j,m-1}]$ (which corresponds to the absolute time interval $[t_{s_j} \; t_{s_j+m-1}]$) is the solution of the Cauchy problem

$$\dot{x}_j^{\text{sim}}(\tau) = \mathcal{N}_f(x_j^{\text{sim}}(\tau), u(t_{s_j} + \tau);\, \theta) \tag{12}$$

for a given initial condition $x_j^{\text{sim}}(0)$. The solution of the Cauchy problem (12) is numerically approximated by

$$x_j^{\text{sim}}(\tau) = \text{ODEINT}(\tau;\, \mathcal{N}_f(\cdot, \cdot;\, \theta), u(\cdot), x_j^{\text{sim}}(0)), \tag{13a}$$

and the simulated output for the j-th subsequence is then given by

$$y_j^{\text{sim}}(\tau) = \mathcal{N}_g(x_j^{\text{sim}}(\tau);\, \theta). \tag{13b}$$

We can now arrange the simulated state and output at the measurement time instants in the tensors $\boldsymbol{y}^{\text{sim}}$ and $\boldsymbol{x}^{\text{sim}}$, respectively:

$$\boldsymbol{y}_{j,h}^{\text{sim}} = y_j^{\text{sim}}(\boldsymbol{\tau}_{j,h}) \tag{14a}$$

$$\boldsymbol{x}_{j,h}^{\text{sim}} = x_j^{\text{sim}}(\boldsymbol{\tau}_{j,h}), \tag{14b}$$

for $j = 0, 1, \ldots, q-1$ and $h = 0, 1, \ldots, m-1$.

As opposed to the full simulation error minimization case, the choice of appropriate initial conditions $x_j^{\text{sim}}(0)$ in the Cauchy problem (12) is here critical. Indeed, the effect of these initial conditions cannot in general be neglected as

the duration of a subsequence is typically much shorter than the whole dataset $\mathcal{D}$. Furthermore, the system state is unlikely to be known *a priori* at all different time instants. For this reason, we introduce an additional set of free variables $\overline{X} = \{\overline{x}_0, \overline{x}_1, \ldots, \overline{x}_{N-1}\}$, where $\overline{x}_k \in \mathbb{R}^{n_x}$ represents the (unmeasured) system state at the measurement time t_k. For a given batch, the set of subsequences' initial conditions $x_j^{\text{sim}}(0)$ is then constructed as $x_j^{\text{sim}}(0) = \overline{x}_{s_j}$ (with $j = 0, \ldots, q-1$) and optimized along with the neural network parameters θ in order to minimize the fitting cost

$$J_{\text{fit}}(\theta, \overline{X}) = \frac{1}{qm} \sum_{j=0}^{q-1} \sum_{h=0}^{m-1} \left\| \boldsymbol{y}_{j,h}^{\text{sim}}(\theta, \overline{X}) - \boldsymbol{y}_{j,h} \right\|^2. \tag{15}$$

It is important to remark that the simulated output $\boldsymbol{y}_{j,h}^{\text{sim}}(\theta, \overline{X})$ is a function of both the neural network parameters θ and the initial conditions $\overline{x}_{s_j} \in \overline{X}$.

Since the fitting cost $J_{\text{fit}}(\theta, \overline{X})$ defined above has N additional degrees of freedom w.r.t. the full simulation error minimization case, a price could be paid in terms of lack of generalization of the estimated model.

In order to reduce the degrees of freedom in the minimization of the loss J_{fit}, the variable $\overline{X}$ used to construct the initial conditions for the Cauchy problem (12) can be enforced to represent the unknown system state and thus to be *consistent* with the neural model structure (2a) (namely, the state sequence $\overline{X}$ should satisfy the neural ODE equation (2a)). To this aim, we introduce a regularization term $J_{\text{reg}}(\theta, \overline{X})$ penalizing the distance between the state $\boldsymbol{x}^{\text{sim}}$ (simulated through (13b)) and the optimization variables in $\overline{X}$. Specifically, the regularization term $J_{\text{reg}}(\theta, \overline{X})$ is defined as

$$J_{\text{reg}}(\theta, \overline{X}) = \frac{1}{qm} \sum_{j=0}^{q-1} \sum_{h=0}^{m-1} \left\| \boldsymbol{x}_{j,h}^{\text{sim}}(\theta, \overline{X}) - \overline{\boldsymbol{x}}_{j,h}(\overline{X}) \right\|^2, \tag{16}$$

where $\overline{\boldsymbol{x}}$ is a tensor with the same size and structure as $\boldsymbol{x}$, but populated with samples from $\overline{X}$, *i.e.*, $\overline{\boldsymbol{x}}_{j,h} = \overline{x}_{s_j+h}$.

The overall loss is then constructed as a weighted sum of the two objectives, namely:

$$J_{\text{tot}}(\theta, \overline{X}) = J_{\text{fit}}(\theta, \overline{X}) + \alpha J_{\text{reg}}(\theta, \overline{X}) \tag{17}$$

with regularization weight $\alpha \geq 0$.

Algorithm 1 details the steps required by the proposed truncated simulation error method to train a dynamical neural model via gradient-based optimization.

In Step 1, the neural network parameters θ and the "hidden" state variables in $\overline{X}$ are initialized. For instance, the initial values of θ and $\overline{X}$ can be set to small random numbers. Alternatively, if a measurement/estimate of some of the system's state variables is available, it can be used to initialize certain components of $\overline{X}$.

Then, at each iteration $i = 0, \ldots, n-1$ of the gradient-based training algorithm, the following steps are executed. Firstly, the batch start vector $s \in \mathbb{N}^q$ is

Algorithm 1 Truncated simulation error minimization

Inputs: training dataset $\mathcal{D}$; number of iterations n; batch size q; length of subsequences m; learning rate $\lambda > 0$; regularization weight $\alpha \geq 0$.

1. **initialize** the neural network parameters θ and the hidden state sequence $\overline{X}$;
2. **for** $i = 0, \ldots, n-1$ **do**
   2.1. **select** batch start index vector $s \in \mathbb{N}^q$;
   2.2. **populate** tensors

$$\begin{aligned} &\boldsymbol{y}_{j,h} = y_{s_j+h}, \qquad \overline{\boldsymbol{x}}_{j,h} = \overline{x}_{s_j+h}, \qquad \boldsymbol{\tau}_{j,h} = t_{s_j+h} - t_{s_j} \\ &\text{for } j = 0, 1, \ldots, q-1 \;\text{ and }\; h = 0, 1, \ldots, m-1 \end{aligned}$$

   and set of initial conditions

$$x_j^{\mathrm{sim}}(0) = \overline{x}_{s_j}, \;\; \text{for } j = 0, 1, \ldots, q-1;$$

   2.3. **simulate** state and output

$$\begin{aligned} x_j^{\mathrm{sim}}(\tau) &= \mathrm{ODEINT}(\tau;\, \mathcal{N}_f(\cdot, \cdot; \theta), x_j^{\mathrm{sim}}(0)) \\ y_j^{\mathrm{sim}}(\tau) &= \mathcal{N}_g(x_j^{\mathrm{sim}}(\tau); \theta) \end{aligned}$$

   for $j = 0, 1, \ldots, q-1$ and $\tau \in [0 \; \tau_{s_{j+m}}]$;
   2.4. **populate** tensors $\boldsymbol{x}^{\mathrm{sim}}$ and $\boldsymbol{y}^{\mathrm{sim}}$ as

$$\begin{aligned} \boldsymbol{x}_{j,h}^{\mathrm{sim}} &= x_j^{\mathrm{sim}}(\boldsymbol{\tau}_{j,h}) \\ \boldsymbol{y}_{j,h}^{\mathrm{sim}} &= y_j^{\mathrm{sim}}(\boldsymbol{\tau}_{j,h}) \end{aligned}$$

   for $j = 0, 1, \ldots, q-1$ and $h = 0, 1, \ldots, m-1$;
   2.5. **compute** the cost

$$J_{\mathrm{tot}}(\theta, \overline{X}) = \overbrace{\frac{1}{qm} \sum_{j=0}^{q-1} \sum_{h=0}^{m-1} \left\| \boldsymbol{y}_{j,h}^{\mathrm{sim}}(\theta, \overline{X}) - \boldsymbol{y}_{j,h} \right\|^2}^{J_{\mathrm{fit}}} + \\ + \alpha \overbrace{\frac{1}{qm} \sum_{h=0}^{m-1} \left\| \boldsymbol{x}_{j,h}^{\mathrm{sim}}(\theta, \overline{X}) - \overline{\boldsymbol{x}}_{j,h}(\overline{X}) \right\|^2}^{J_{\mathrm{reg}}};$$

   2.6. **evaluate** the gradients $\nabla_\theta J_{\mathrm{tot}} = \frac{\partial J_{\mathrm{tot}}}{\partial \theta}$ and $\nabla_{\overline{X}} J_{\mathrm{tot}} = \frac{\partial J_{\mathrm{tot}}}{\partial \overline{X}}$ at the current values of θ and $\overline{X}$;
   2.7. **update** optimization variables θ and $\overline{X}$:

$$\begin{aligned} \theta &\leftarrow \theta - \lambda \nabla_\theta J_{\mathrm{tot}} \\ \overline{X} &\leftarrow \overline{X} - \lambda \nabla_{\overline{X}} J_{\mathrm{tot}}; \end{aligned}$$

Output: neural network parameters θ.

selected with $s_j \in [0 \;\; N-m-1],\; j = 0, 1, \ldots, q-1$ (Step 2.1). The indexes in s can be either (pseudo)randomly generated, or chosen deterministically.[4] Then, tensors $\boldsymbol{y}$, $\overline{\boldsymbol{x}}$, and $\boldsymbol{\tau}$ are populated with the corresponding values in $\mathcal{D}$ and $\overline{X}$. The initial conditions $x_j^{\text{sim}}(0)$, with $j = 0, 1, \ldots, q-1$ are also obtained from $\overline{X}$ (Step 2.2). Subsequently, for each subsequence j, the system state and output are simulated within the time interval $[0 \;\; \tau_{s_{j+m-1}}]$, starting from the initial condition $x_j^{\text{sim}}(0)$ (Step 2.3). Then, the values of the simulated state and output at the (relative) measurement times $\boldsymbol{\tau}_{j,h}$ (with $h = 0, \ldots, m-1$) are collected in tensors $\boldsymbol{x}^{\text{sim}}$ and $\boldsymbol{y}^{\text{sim}}$ (Step 2.4), and used to construct the loss $J_{\text{tot}}(\theta, \overline{X})$ (Step 2.5). Next, the gradients of the cost with respect to the optimization variables θ, $\overline{X}$ are obtained in (Step 2.6), either by back-propagation through the solver steps or exploiting the adjoint sensitivity method suggested in **?**. Lastly, the optimization variables are updated via gradient descent with *learning rate* λ (Step 2.7). Improved variants of gradient descent such as *Adam* (**?**) can be alternatively adopted at Step 2.7.

Note that, at each iteration i of the gradient descent algorithm, the cost $J_{\text{tot}}(\theta, \overline{X})$ may depend on just a subset of hidden state variables $\overline{X}$. Thus, only for those components at each iteration the gradient vector $\nabla_{\overline{X}} J_{\text{tot}} = \frac{\partial J_{\text{tot}}}{\partial \overline{X}}$ is non-zero and an update is performed.

Remark 3 *The computational cost of evaluating the gradient of* J_{tot} *is proportional to the number of solver steps executed in* (13a)*. In the case the solver step is equal to the sampling intervals* $t_{k+1} - t_k$ *(which corresponds to* m *solver steps), running truncated simulation error minimization with* $m \ll N$ *is thus significantly faster than full simulation error minimization. Furthermore, the computations for the* q *subsequences can be carried out independently, and thus parallel computing can be exploited.*

Remark 4 *The use of a regularization term enforcing consistency of the hidden variable* $\overline{X}$ *with the system dynamics has analogies with the* multiple shooting *numerical method for optimal control of dynamical systems* **?***. As in multiple shooting, the original optimization problem defined over a long time window is split into smaller intervals, which are then processed independently. While in multiple shooting the intervals are fixed and continuity of the solution is enforced by equality constraints at the interval boundaries, in our approach the subsequences may have different starting points, and the regularization term promoting consistency of the hidden variable* $\overline{X}$ *is defined for all time steps.*

Remark 5 *Our regularized truncated simulation error minimization criterion* (17) *has also similarities with the cost function minimized by the neural moving horizon estimator presented in* **?***. Furthermore, a by-product of our identification algorithm is a state estimate* $\overline{X}$ *for all the time instants in the training dataset that may be interpreted, for each time instant, as an average of all* m*-length moving horizon estimators including that time instant, based on*

[4] For an efficient use of the training data $\mathcal{D}$, s has to be chosen in such a way that all samples are visited with equal frequency during the iterations of Algorithm 1.

the learned model dynamics. Note that while in **?** *the true system dynamics is assumed to be known and a neural network is used to model the state estimation function, in our approach the true dynamics is unknown and jointly approximated as neural networks together with the hidden state sequence* $\overline{X}$*, by minimizing the regularized fitting criterion* (17)*.*

In the next paragraph, we introduce an alternative method for fitting neural dynamical models that does not require simulation through time, and thus is even more convenient from a computational perspective, allowing full parallelization at time step level.

4.3 Soft-constrained integration

The optimization variables $\overline{X}$ previously introduced for truncated simulation error minimization are regularized to be consistent with the fitted system dynamics through the cost J_{reg} (16). Therefore, $\overline{X}$ implicitly provides another estimate of the unknown system state, and thus of the output Y for given $\mathcal{N}_g$. This estimate can be compared with the measured samples Y to define an alternative fitting objective. This suggests the following variant for the fitting term J_{fit}:

$$J_{\text{fit}}(\overline{X}) = \frac{1}{qm} \sum_{j=0}^{q-1} \sum_{h=0}^{m-1} \left\| \overline{\boldsymbol{y}}_{j,h} - \boldsymbol{y}_{j,h} \right\|^2, \tag{18}$$

where $\overline{\boldsymbol{y}}_{j,h} = \mathcal{N}_g(\overline{\boldsymbol{x}}_{j,h})$.

Experimentally, using (18) instead of (15) as fitting term does not significantly alter the properties of truncated simulation error minimization. Furthermore, the wall-clock execution time of Algorithm 1 with the modified fitting term (18) is still dominated by the m-step simulation (and back-propagation) still required to compute the gradient of the regularization term J_{reg} in (16). In order to formulate a faster learning algorithm, an alternative regularization term J_{reg} promoting consistency of the hidden state variables, but not requiring time simulation should be devised.

The consistency-promoting regularizer J_{reg} considered in the soft-constrained integration method penalizes the violation of a numerical ODE integration scheme applied to the hidden state variables $\overline{X}$, independently at each integration step. For instance, the forward Euler scheme can be enforced by means of the regularization term

$$J_{\text{reg}}(\overline{X}, \theta) = \sum_{j=0}^{q-1} \sum_{h=1}^{m-1} \left\| \overline{\boldsymbol{x}}_{j,h} - \overline{\boldsymbol{x}}_{j,h-1} - \Delta t \mathcal{N}_f(\overline{\boldsymbol{x}}_{j,h-1}, \boldsymbol{u}_{j,h-1}) \right\|^2, \tag{19}$$

assuming for notation simplicity a constant step size Δt. If the regularization term (19) is reduced to a "small value" through optimization, then the hidden variables $\overline{X}$ will (approximately) satisfy the forward Euler scheme.

Algorithm 2 details the training steps when the forward Euler numerical scheme is used to enforce consistency of the hidden state variables $\overline{X}$ with the

Algorithm 2 Soft-constrained integration for the forward Euler scheme

Inputs: training dataset $\mathcal{D}$; number of iterations n; batch size q; length of subsequences m; learning rate $\lambda > 0$; regularization weight $\alpha \geq 0$.

1. **initialize** the neural network parameters θ and the hidden state sequence $\overline{X}$;
2. **for** $i = 0, \ldots, n-1$ **do**
   2.1. **select** batch start index vector $s \in \mathbb{N}^q$;
   2.2. **populate** tensors

$$\begin{aligned} &\boldsymbol{y}_{j,h} = y_{s_j+h}, \qquad \overline{\boldsymbol{x}}_{j,h} = \overline{x}_{s_j+h}, \qquad \boldsymbol{u}_{j,h} = u_{s_j+h}, \\ &\text{for } j=0,1,\ldots,q-1 \;\text{ and }\; h=0,1,\ldots,m-1; \end{aligned}$$

   2.3. **compute** the cost

$$J_{\mathrm{tot}}(\theta, \overline{X}) = \overbrace{\frac{1}{qm} \sum_{j=0}^{q-1} \sum_{h=0}^{m-1} \left\| \overline{\boldsymbol{y}}_{j,h} - \boldsymbol{y}_{j,h} \right\|^2}^{J_{\mathrm{fit}}} + \\ + \alpha \overbrace{\frac{1}{qm} \sum_{j=0}^{q-1} \sum_{h=1}^{m-1} \left\| \overline{\boldsymbol{x}}_{j,h} - \overline{\boldsymbol{x}}_{j,h-1} - \Delta t \mathcal{N}_f(\overline{\boldsymbol{x}}_{j,h-1}, \boldsymbol{u}_{j,h-1}) \right\|^2}^{J_{\mathrm{reg}}},$$

   with $\overline{\boldsymbol{y}}_{j,h} = \mathcal{N}_g(\overline{\boldsymbol{x}}_{j,h})$;
   2.4. **evaluate** the gradients $\nabla_\theta J_{\mathrm{tot}} = \frac{\partial J_{tot}}{\partial \theta}$ and $\nabla_{\overline{X}} J_{\mathrm{tot}} = \frac{\partial J_{q,m}}{\partial \overline{X}}$ at the current values of θ and $\overline{X}$;
   2.5. **update** optimization variables θ and $\overline{X}$:

$$\begin{aligned} \theta &\leftarrow \theta - \lambda \nabla_\theta J_{\mathrm{tot}} \\ \overline{X} &\leftarrow \overline{X} - \lambda \nabla_{\overline{X}} J_{\mathrm{tot}}; \end{aligned}$$

Output: neural network parameters θ.

model dynamics. In Step 1, the neural network parameters θ, and the sequence of hidden variables $\overline{X}$ are initialized. Then, at each iteration $i = 0, 1, \ldots, n-1$ of the gradient-based training algorithm, the following steps are executed. Firstly, the batch start vector $s \in \mathbb{N}^q$ is selected with $s_j \in [0 \;\; N-m-1],\; j = 0, 1, \ldots, q-1$ (Step 2.1) and the tensors $\boldsymbol{y}$, $\overline{\boldsymbol{x}}$, $\overline{\boldsymbol{u}}$ are populated with the corresponding samples in $\mathcal{D}$ (Step 2.2), similarly as in Algorithm 1

Then, the loss J_{tot} is computed (Step 2.3) as a weighted sum of the fitting cost J_{fit} in (18) and the regularizer J_{reg} (19). Note that, unlike Algorithm 1, Algorithm 2 does not require m-step simulation. The gradients of J_{tot} are obtained using standard back-propagation and used to perform a gradient-based minimization step (Steps 2.4 and 2.5).

The potential advantage of the proposed soft-constrained integration method over the truncated simulation error minimization is twofold. Firstly, implicit integration schemes can be enforced with no additional computational burden with respect to explicit ones. For instance, the backward Euler integration scheme can be implemented simply by modifying the consistency term J_{reg} to

$$J_{\text{reg}}(\overline{X}, \theta) = \sum_{j=0}^{q-1} \sum_{h=1}^{m-1} \left\| \overline{\boldsymbol{x}}_{j,h} - \overline{\boldsymbol{x}}_{j,h-1} - \Delta t \mathcal{N}_f(\overline{\boldsymbol{x}}_{j,h}, \boldsymbol{u}_{j,h}) \right\|^2, \tag{20}$$

while the Crank-Nicolson scheme corresponds to

$$J_{\text{reg}}(\overline{X}, \theta) = \sum_{j=0}^{q-1} \sum_{h=1}^{m-1} \left|\left| \overline{\boldsymbol{x}}_{j,h} - \overline{\boldsymbol{x}}_{j,h-1} - \frac{\Delta t}{2} (\mathcal{N}_f(\overline{\boldsymbol{x}}_{j,h}, \boldsymbol{u}_{j,h}) + \mathcal{N}_f(\overline{\boldsymbol{x}}_{j,h-1}, \boldsymbol{u}_{j,h-1})) \right|\right|^2. \tag{21}$$

Other implicit schemes such as the multi-step Backward Differentiation Formula (BDF) and Adams-Moulton (AM), or multi-stage implicit Runge Kutta (RK) methods may be similarly implemented, leading to a potential increase in the accuracy of the ODE numerical solution **?**. Secondly, the resulting cost function splits up as a sum of independent contributions for each time step, thus enabling the fully parallel implementation of gradient-based optimization. This leads to significant computational advantages and a reduced wall-clock execution time.

On the other hand, in the proposed soft-constrained integration method, the numerical scheme is only approximately satisfied at each solver step, and the degree of violation eventually depends on the weighting constant α in the cost function. The tuning of the weight α is thus more critical as compared to truncated simulation error minimization.

5 Case studies

The performance of the model structures and fitting algorithms introduced in this paper are tested on three cases studies considering the identification of a nonlinear RLC circuit; a *Cascaded Tanks System* (CTS); and an *Elecro-Mechanical Positioning System* (EMPS).

Code availability The software implementation is based on the *PyTorch* Deep Learning Framework (**?**). All the codes required to reproduce the results reported in the paper are available in the on-line repository `https://github.com/forgi86/sysid-neural-continuous`.

Dataset availability The RLC circuit dataset is simulated and available in the on-line repository. Experimental datasets for the CTS and the EMPS case studies are obtained from the website `http://www.nonlinearbenchmark.org`, which hosts a collection of public benchmarks widely used in system identification.

Metrics For all case studies, the performance of the fitting algorithms is assessed in terms of the R^2 index of the model simulation:

$$R^2 = 1 - \frac{\sum_{k=0}^{N-1} \left(y_k - y^{\mathrm{sim}}(t_k)\right)^2}{\sum_{k=0}^{N-1} \left(y_k - y^{\mathrm{mean}}\right)^2},$$

where $y^{\mathrm{mean}} = \frac{1}{N}\sum_{k=0}^{N-1} y_k$.

For the CTS, the *Root Mean Squared Error* (RMSE) of the model simulation is also provided, as this is the performance index suggested in the description of the benchmark **?**:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} \left(y_k - y^{\mathrm{sim}}(t_k)\right)^2}.$$

The performance indexes are evaluated both on the training dataset and on a separate test dataset. In the case of systems with multiple output channels, the metrics are computed channel-wise.

Data pre-processing For the EMPS example, basic data pre-processing steps described in the corresponding section have been applied on the training dataset to improve the algorithm performance. In the other cases, the original training datasets have been used for training as the available signals are already in a reasonable numerical range to apply our methods. Even though pre-processing steps may provide marginal performance improvements also in these examples, we decided to present the results obtained using the original datasets for the sake of simplicity. In all examples, the metrics and plots are reported for variables in the original unit of measure.

Algorithm settings In truncated simulation error minimization (Algorithm 1), the batch size q and sequence length m are chosen within the integer range [32 128], while the regularization weight α in (17) is always set to 1.

In the soft-constrained integration method (Algorithm 2), we consider instead a single subsequence containing all the dataset samples, *i.e.*, $q = 1$ and

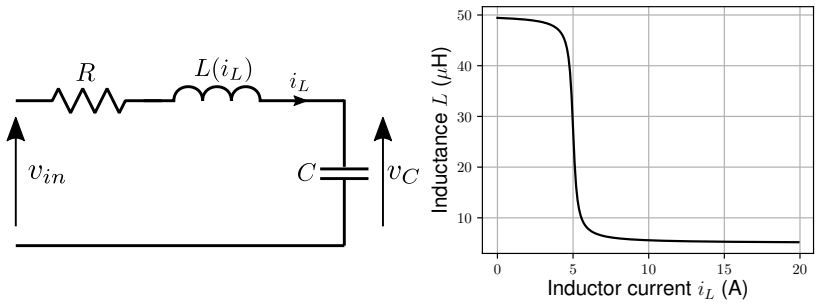


Figure 4: Nonlinear series RLC circuit used in the example (left) and nonlinear dependence of the inductance L on the inductor current i_L (right).

$m = N$. The weight constant α is tuned based on the simulation performance of the identified model in the training dataset.

In all the examples, the Adam optimizer is used for gradient-based optimization. The learning rate λ is adjusted through a rough trial-and-error within the range $[10^{-2}\ 10^{-6}]$, while the other optimizer parameters are left to default. The number of training steps n is chosen sufficiently high to reach a cost function plateau.

The neural networks' weight parameters are initialized to random Gaussian variables with zero mean and standard deviation 10^{-4}, while the bias terms are initialized to zero. The hidden state variables $\overline{X}$ are initialized differently for the three examples, exploiting available process knowledge.

Hardware configuration All computations are carried out on a PC equipped with an Intel i5-7300U 2.60 GHz processor and 32 GB of RAM.

5.1 Nonlinear RLC circuit

We consider the nonlinear RLC circuit in Figure 4 (left). The circuit behavior is described by the continuous-time state-space equation

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ \frac{-1}{L(i_L)} & \frac{-R}{L(i_L)} \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L(i_L)} \end{bmatrix} v_{in}, \tag{22}$$

where v_{in} (V) is the input voltage; v_C (V) is the capacitor voltage; and i_L (A) is the inductor current. The circuit parameters $R = 3\ \Omega$ and $C = 270$ nF are fixed, while the inductance L depends on i_L as shown in Figure 4 (right) and according to

$$L(i_L) = L_0 \left[0.9 \left(\frac{1}{\pi} \arctan\big(-5(|i_L| - 5) + 0.5 \big) \right) + 0.1 \right],$$

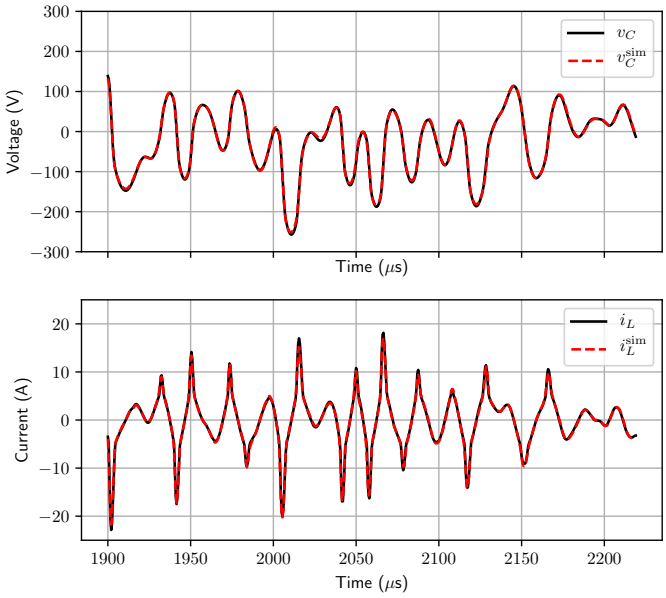


Figure 5: RLC circuit: true output (black) and estimated output (red) obtained by the state-space model trained by truncated simulation error minimization.

with $L_0 = 50$ μH. This kind of dependence is typically encountered in ferrite inductors operating in partial saturation (**?**).

In this case study, we assume both state variables v_C and i_L to be measured. A training dataset $\mathcal{D}$ with $N = 4000$ samples is built by simulating the system for 2 ms with a fixed step $\Delta t = 0.5$ μs. The input v_{in} is a filtered white noise with bandwidth 150 kHz and standard deviation 80 V. An independent test dataset is generated using as input v_{in} a filtered white noise with bandwidth 200 kHz and standard deviation 60 V. In the training dataset, the observations of v_C and i_L are corrupted by an additive white Gaussian noise with zero mean and standard deviation 10 V and 1 A, respectively. This corresponds to a *Signal-to-Noise Ratio* (SNR) of 20 dB and 13 dB on v_C and i_L, respectively.

Neural Model Structure By considering as state vector $x = [v_C \; i_L]^\top$ and input $u = v_{in}$, we adopt for this system the fully observed state model structure (5) introduced in Section 3 reported for convenience below:

$$\dot{x} = \mathcal{N}_f(x, u)$$
$$y = x.$$

This model structure embeds the knowledge that (i) the system has a second-order state-space representation and (ii) the whole state vector is measured.

The neural network $\mathcal{N}_f$ used in this example has feed-forward structure with three input units (corresponding to v_C, i_L, and v_{in}); a hidden layer with 64

Subsequence length m (-)	Run-time (s)	R^2 index (-)
2	16	0.95/0.87
4	19	0.98/0.92
8	27	0.99/0.97
64	138	0.99/0.98
128	268	0.99/0.99
256	536	0.99/0.98

Table 1: RLC circuit: run-time and R^2 index in test on the voltage/current channels for truncated simulation error minimization with increasing value of the subsequence length m.

linear units followed by ReLU nonlinearity; and two linear output units—the components of the state equation to be learned.

Truncated simulation error minimization Algorithm 1 is executed with learning rate $\lambda = 10^{-4}$, number of iterations is $n = 10000$, and batches containing $q = 64$ subsequences, each one of size $m = 64$. The hidden state variables $\overline{X}$ are initialized to the values of the noisy output measurements. Model equations (12) are numerically integrated using the forward Euler numerical scheme. The total run time of Algorithm 1 is 142 seconds.

Time trajectories of the true and model output are reported in Figure 5. For the sake of visualization, only a portion of the test dataset is shown. The fitted model describes the system dynamics with high accuracy. On both the training and the test datasets, the R^2 index in simulation is above 0.99 for v_C and 0.98 for i_L. For the sake of comparison, a second-order linear *Output Error* model estimated using the *System Identification Toolbox* (**?**) achieves an R^2 index of 0.96 for v_C and 0.77 for i_L on the training dataset, and 0.94 for v_C and 0.76 for i_L on the test dataset.

Table 1 reports the run-time and the achieved R^2 index in test on v_C/i_L obtained by re-running truncated simulation error minimization algorithm with different values of the subsequence length m, while keeping all other settings constant. The run-time depends approximately linearly on m. This trend is more clear for the larger values of m considered. The model performance generally improves with increasing values of m, with the exception of a small performance decrease observed for $m = 256$.

Full simulation error minimization Full simulation error minimization is also tested. This method yields the same performance of 64-step truncated simulation error minimization in terms of R^2 index of the fitted model. However, the run-time required to execute $n = 10000$ iterations and reach a cost function

plateau is about two hours.

Soft-constrained integration The soft-constrained integration method described in Algorithm 2 is tested. The regularization constant α and the learning rate λ are set to 100 and 10^{-3}, respectively. The algorithm run-time for $n = 20000$ gradient descent iterations is 135 seconds. The estimated model achieves R^2 index of 0.99 and 0.98 for v_C and i_L, both in the training and in the test datasets.

One-step prediction error minimization For the fully-observed neural model structure, a straightforward fitting criterion may be defined by taking the noisy output measurement as a state estimate and by minimizing the *one-step prediction error loss*

$$J_{\text{pred}}(\theta) = \sum_{t=1}^{N-1} \|y_t - y_{t-1} - \Delta t \mathcal{N}_f(y_{t-1}, u_{t-1})\|^2 . \tag{23}$$

Minimization of (23) corresponds to the training of a *standard* feedforward neural network with target $\frac{y_t - y_{t-1}}{\Delta t}$ and features y_{t-1}, u_{t-1}.

Since the neural network is fed with noisy input data and only the one-step ahead is minimized, this approach is not robust to the measurement noise, as illustrated by the authors in (**?**). On this RLC example, the fully-observed state neural model structure trained by minimizing $J_{\text{pred}}(\theta)$ achieves an R^2 index of 0.73 for v_C and 0.03 for i_L in the test dataset (see time trajectories in Figure 6).

By repeating the fitting procedure on a noise-free training dataset, one-step prediction error minimization recovers the same performance of simulation error minimization and soft-constrained integration (R^2 index of 0.99 and 0.98 for v_C and i_L, respectively).

Noise sensitivity For the proposed truncated simulation error minimization and soft-constrained integration approaches, the result of a sensitivity analysis with respect to increasing levels of measurement noise is summarized in Table 2. In particular, we consider the cases where the measurement noise on the two channels is multiplied by factors 0x, 1x, 2x, 3x, and 6x. Note that the factors 0x and 1x correspond to the noise-free and to the noisy datasets considered above.

The truncated simulation error minimization approach is clearly less sensitive to measurement noise w.r.t. to soft-constrained integration. Still, soft-constrained integration is more noise-tolerant than the one-step prediction error approach, which is only applicable to the noise-free dataset (0x), as shown in the previous paragraph.

5.2 Cascaded Tanks System

We consider the CTS already introduced in Section 3 and described in details in (**?**).

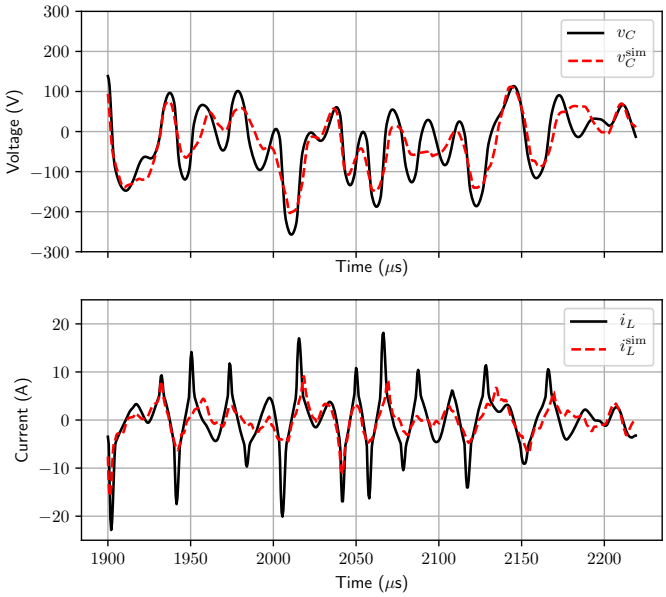


Figure 6: RLC circuit: true output (black) and estimated output (red) obtained by the state-space model trained by one-step prediction error minimization.

The training and test datasets contain 1024 points each, collected at a constant sampling time $\Delta t = 5$ s. In both datasets, the input is a multisine signal with identical power spectrum, but different realization. Input and output values are in Volts as they correspond to the actuator commands and the raw sensor readings, respectively.

The initial state of the system is not provided, but it is known to be the same for both datasets. Thus, as suggested in (**?**), we use the initial state estimated on the training dataset (which is a by-product of the proposed fitting procedures) as initial state for model simulation in test.

	Noise level				
Algorithm	0x	1x	2x	3x	6x
TSEM	0.99/0.98	0.99/0.98	0.99/0.97	0.98/0.94	0.96/0.91
SCI	0.99/0.99	0.99/0.98	0.89/0.7	0.38/-0.027	0.035/-0.23

Table 2: RLC circuit: R^2 index for the voltage/current channels on the test set for truncated simulation error minimization (TSEC, first row) and soft-constrained integration (SCI, second row) for increasing noise level.

Neural Model Structure The neural model structure used for this system is

$$\dot{x}_1 = \mathcal{N}_{f_1}(x_1, u) \tag{24a}$$
$$\dot{x}_2 = \mathcal{N}_{f_2}(x_1, x_2, u) \tag{24b}$$
$$y = x_2. \tag{24c}$$

Compared to (7), model (24) also includes a direct dependency on u in the second state equation. This dependency is added to take into account that in this experimental setup, in case of water overflow from the upper tank, part of the overflowing water may go directly in the lower tank (**?**).

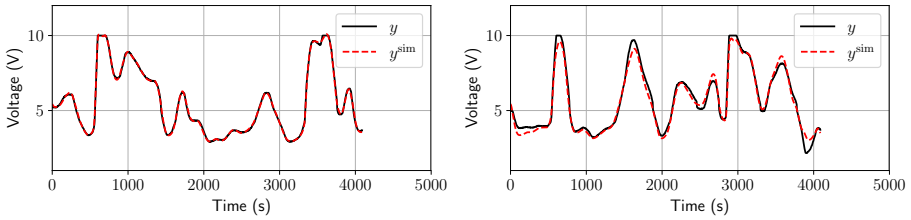
The neural networks $\mathcal{N}_{f_1}$ and $\mathcal{N}_{f_2}$ have two and three input units, respectively. Both networks one hidden layer with 100 linear units followed by ReLU nonlinearity and a linear output unit.

Truncated Simulation Error Minimization Algorithm 1 is executed with learning rate $\lambda = 10^{-3}$, number of iterations $n = 10000$, batch size $q = 64$, and subsequence length $m = 128$. The components of the hidden state variables $\overline{X}$ associated to the state variable x_1 are initialized to 0, while the components associated to x_2 are initialized to the noisy output measurements y. Model equations (10) are numerically integrated using the forward Euler numerical scheme. The total run-time of Algorithm 1 is 533 seconds.

Time trajectories of simulated and true output are reported in Figure 7. The achieved R^2 index is 0.99 and 0.97 on the training and on the test dataset, respectively. The RMSE index is 0.08 V and 0.33 V on the identification and on the test dataset, respectively. These results compare favorably with state-of-the-art black-box nonlinear identification methods applied to this benchmark (**???**). To the best of our knowledge, the best previously published result was obtained in **?** using non-linear basis expansion state-space models and exploiting a state initialization procedure (called NLSS2) based on a preliminary (linear) subspace identification for optimization. The authors of (**?**) report an RMSE index of 0.34 V on the test dataset. A higher performance was reported only in (**?**) for grey-box models including an explicit physical description of the water overflow phenomenon (RMSE index of 0.18 V).

Note that the largest discrepancies between measured and simulated output are noticeable towards the end of the test experiment, where the measured output y is close to 2 V. This condition is not encountered in the training dataset and therefore a mismatch can be expected for a black-box nonlinear model such as a neural network.

Soft-constrained integration method Algorithm 2 is executed with the regularization term J_{reg} in (21) enforcing the Crank-Nicolson integration scheme and a regularization constant $\alpha = 50000$. For this small dataset, all time steps fit into the memory and can be processed altogether in a batch. Thus, we consider batches with a single subsequence ($q = 1$) containing the whole training



(a) Training dataset.

(b) Test dataset.

Figure 7: CTS: measured output y (black) and model simulation y^{sim} (red) obtained by the neural model trained by truncated simulation error minimization.

Method	RMSE (V)
Best linear approximation **?**	0.75
Volterra model **?**	0.54
State-space with GP-inspired prior **?**	0.45
Non-linear SS + NLSS2 initialization **?**	0.34
Grey-box with physical overflow model **?**	0.18
TSEM [this paper]	0.33
SCI [this paper]	0.40

Table 3: CTS: comparison of truncated simulation error minimization (TSEM) and soft-constrained integation (SCI) with other published methods in the literature in terms of the RMSE index in the test dataset.

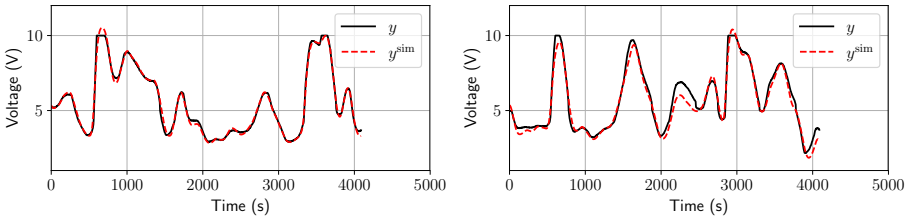
dataset $\mathcal{D}$ ($m = 1024$). Optimization is performed over $n = 50000$ iterations of the Adam algorithm, with learning rate $\lambda = 10^{-5}$. The total run-time of Algorithm 2 is 271 seconds—approximately half the run-time of Algorithm 1.

Time trajectories of the output are reported in Figure 8 for both the training and the test dataset. The R^2 index of the model is 0.99 and 0.96 on the training and on test dataset, respectively. The RMSE index is 0.18 V and 0.40 V on the training and on the test datasets, respectively. The results are thus in line with the ones achieved by truncated simulation error minimization.

Table 3 summarizes the results obtained in the literature on the CTS benchmark, together with the ones of truncated simulation error minimization and soft-constrained integration in terms of the RMSE index in test.

5.3 Electro-Mechanical Positioning System

As a last case study, we consider the identification of the Electro-Mechanical Positioning System (EMPS) described in (**?**).



(a) Training dataset.

(b) Test dataset.

Figure 8: CTS: measured output y (black) and model simulation y^{sim} (red) obtained by the neural model trained using the soft-constrained integration method.

The system is a controlled prismatic joint, which is a common component of robots and machine tools. In the benchmark, the system input is the motor force τ (N) expressed in the load side and the measured output is the prismatic joint position p (m). A physical state-space model for the system is

$$\dot{p} = v \tag{25a}$$

$$\dot{v} = -\frac{\tau}{M} - \frac{f_v}{M} v - \frac{F_c(v)}{M}, \tag{25b}$$

where M (kg) is the joint mass, f_v (Ns/m) is the dynamic friction coefficient and F_c (N) is the static friction.

The identification and test dataset are constructed from closed-loop experiments performed with the same reference position trajectory. A force disturbance is acting on the system in the test experiment only. The two datasets have the same duration (approximately 25 seconds) and are collected at a sampling frequency of 1 kHz.

For training, the original EMPS signals are decimated by a factor 5. Thus, each dataset contains $N = 4968$ points with sampling time $\Delta t = 5$ ms. Furthermore, the output position is normalized to the numerical range $[-1\ 1]$. Finally, an initial estimate of the joint velocity to be used for initialization of the hidden state variables is obtained by numerical differentiation (using a forward finite differences scheme) of the output position signal.

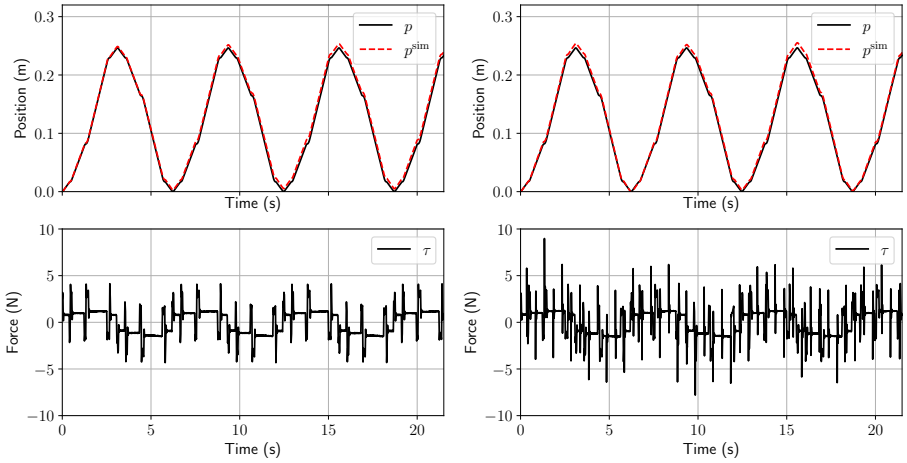
Neural Model Structure According to the physical model (25), the neural model structure used to fit the EMPS system is

$$\dot{x}_1 = x_2 \tag{26a}$$

$$\dot{x}_2 = \mathcal{N}_f(x_2, u) \tag{26b}$$

$$y = x_1, \tag{26c}$$

with state variables $x_1 = p$ and $x_2 = v$; and input $u = \tau$.



(a) Training dataset. (b) Test dataset.

Figure 9: EMPS: measured position p (top panels, black) and model simulation p^{sim} (top panels, red) obtained by the neural model trained by truncated simulation error minimization. The input force τ is shown in the bottom panels.

The neural model structure (26) captures the physical knowledge that: (*i*) the system states are position and velocity; (*ii*) the derivative of position is velocity; and (*iii*) the velocity dynamics does not depend on the position. The neural network is thus used to describe the velocity dynamics (25b), which could be rather complex due to the presence of static friction. Indeed, static friction is highly nonlinear and hard to describe with first-principles formulas. On the other hand, there is no need to use a black-box model to describe the position dynamics (25a). In fact, this equation simply states that velocity is the time derivative of position.

The neural network $\mathcal{N}_f$ used for this benchmark has 2 input units; 64 hidden linear units followed by ReLU nonlinearity; and one linear output unit.

Truncated simulation error method Algorithm 1 is executed with learning rate $\lambda = 10^{-4}$, number of iterations is $n = 10000$, batch size $q = 32$, and sequence length $m = 64$. The components of $\overline{X}$ associated to x_1 are initialized to the position sequence, while the components associated to x_2 are initialized using the initial velocity estimate obtained through numerical differentiation of the position sequence in the pre-processing step.

Model equations (10) are numerically integrated using the fourth-order Runge-Kutta scheme RK44 (**?**). The total run-time of Algorithm 1 is 710 seconds.

Time trajectories of the input and of the output are reported in Figure 9. The achieved R^2 index is above 0.99 on both the identification and test datasets. By comparison, **?** reports an R^2 index of 0.5 for different linear models estimated on this benchmark.

Soft-constrained integration method Algorithm 2 is executed with the regularization term J_{reg} in (20) enforcing the backward Euler integration scheme and regularization weight $\alpha = 1000$. As for the CTS benchmark, we consider batches with a single subsequence ($q = 1$) containing the whole training dataset $\mathcal{D}$ ($m = 4968$). Optimization is performed over $n = 40000$ iterations of the Adam algorithm, with learning rate $\lambda = 10^{-5}$. The identified model achieves an R^2 index above 0.99 both in identification and in test, as for Algorithm 1. The total run-time of Algorithm 2 is 342 seconds (around 2x faster then Algorithm 1).

6 Conclusions and follow-up

In this paper, we have presented neural model structures and two novel methodologies for the identification of continuous-time dynamical systems.

The main strengths of the presented framework are: (*i*) its versatility to describe complex and structured non-linear systems, thanks to the neural network flexibility and the possibility to exploit physical model structures; (*ii*) its robustness to the measurement noise, thanks to the minimization of a (truncated) simulation error criterion and regularization terms that enforce the hidden state variables to be consistent with the estimated neural model; (*iii*) the possibility to exploit parallel computing to train the network and optimize the initial conditions, thanks to the division of the dataset into small-size subsequences.

The proposed case studies have shown the effectiveness of the presented methodologies, and their capabilities to outperform state-of-art algorithms in well-known benchmarks for system identification.

Current and future research activities are devoted to the application of the proposed framework for the identification of systems described by partial differential equations, as well as the formulation of alternative fitting criteria that directly take into account the final usage the estimated dynamical models, like fault detection and control system design.

Acknowledgements

This work was partially supported by the European H2020-CS2 project ADMITTED, Grant agreement no. GA832003.

References

M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

A. Alessandri, M. Baglietto, G. Battistelli, and M. Gaggero. Moving-horizon state estimation for nonlinear systems using neural networks. *IEEE Transactions on Neural Networks*, 22(5):768–780, 2011.

Y. Bengio et al. Learning deep architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

G. Birpoutsoukis, P. Z. Csurcsia, and J. Schoukens. Efficient multidimensional regularization for volterra series estimation. *Mechanical Systems and Signal Processing*, 104:896–914, 2018.

H. G. Bock and K.-J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

S. Chen, S. Billings, and P. Grant. Non-linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214, 1990.

T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

G. Di Capua, N. Femia, K. Stoyka, M. Lodi, A. Oliveri, and M. Storace. Ferrite inductor models for switch-mode power supplies analysis and design. In *2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–4, 2017.

M. Forgione and D. Piga. Model structures and fitting criteria for system identification with neural networks. In *14th IEEE International Conference on Application of ICT*, Tashkent, Uzbekistan, 2020.

H. Garnier. Direct continuous-time approaches to system identification. overview and benefits for practical applications. *European Journal of control*, 24:50–62, 2015.

H. Garnier and L. Wang. *Identification of Continuous-time Models from Sampled Data*. Springer Publishing Company, Incorporated, 1st edition, 2008.

H. Garnier and P. C. Young. The advantages of directly identifying continuous-time transfer function models in practical applications. *International Journal of Control*, 87(7):1319–1338, 2014a. doi: 10.1080/00207179.2013.840053.

H. Garnier and P. C. Young. Special issue on applications of continuous-time model identification and estimation, 2014b.

H. Garnier, M. Mensler, and A. Richard. Continuous-time model identification from sampled data: implementation issues and performance evaluation. *International journal of Control*, 76(13):1337–1357, 2003.

J. Gonzalez and W. Yu. Non-linear system modeling using LSTM neural networks. *IFAC-PapersOnLine*, 51(13):485–489, 2018.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363, 2019.

E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

B. G. Horne and C. L. Giles. An experimental comparison of recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 697–704, 1995.

A. Janot, M. Gautier, and M. Brunot. Data Set and Reference Models of EMPS. In *Nonlinear System Identification Benchmarks*, EINDHOVEN, Netherlands, Apr. 2019. URL `https://hal.archives-ouvertes.fr/hal-02178709`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

R. Kumar, S. Srivastava, J. Gupta, and A. Mohindru. Comparative study of neural networks for dynamic nonlinear systems identification. *Soft Computing*, 23(1):101–114, 2019.

J. Lataire, R. Pintelon, D. Piga, and R. Tóth. Continuous-time linear time-varying system identification with a frequency-domain kernel-based estimator. *IET Control Theory Applications*, 11(4):457–465, 2017. ISSN 1751-8644.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Y. Liu and J. J. Zhu. Continuous-time nonlinear system identification using neural network. In *2008 American Control Conference*, pages 613–618. IEEE, 2008.

L. Ljung. Convergence analysis of parametric identification methods. *IEEE Transactions on Automatic Control*, 23(5):770–783, 1978.

L. Ljung, editor. *System Identification: Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2 edition, 1999. ISBN 0-13-656695-2.

L. Ljung and R. Singh. Version 8 of the Matlab System Identification Toolbox. *IFAC Proceedings Volumes*, 45(16):1826–1831, 2012.

Z. Long, Y. Lu, and B. Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490, 2019*, 2019.

D. Masti and A. Bemporad. Learning nonlinear state-space models using deep autoencoders. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 3862–3867, 2018.

G. Mercère, H. Palsson, and T. Poinot. Continuous-time linear parameter-varying identification of a cross flow heat exchanger: a local approach. *IEEE Transactions on Control Systems Technology*, 19(1):64–76, 2011.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

D. Piga. Finite-horizon integration for continuous-time identification: bias analysis and application to variable stiffness actuators. *International Journal of Control*, In press. doi: 10.1080/00207179.2018.1557348.

A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

A. Ralston. Runge-Kutta methods with minimum error bounds. *Mathematics of computation*, 16(80):431–437, 1962.

R. Relan, K. Tiels, A. Marconato, and J. Schoukens. An unstructured flexible nonlinear model for the cascaded water-tanks benchmark. *IFAC-PapersOnLine*, 50(1):452–457, 2017.

X. Ren, A. B. Rad, P. Chan, and W. L. Lo. Identification and control of continuous-time nonlinear systems via dynamic neural networks. *IEEE Transactions on Industrial Electronics*, 50(3):478–486, 2003.

A. H. Ribeiro, K. Tiels, J. Umenberger, T. B. Schön, and L. A. Aguirre. On the smoothness of nonlinear system identification. *Automatica*, 121:109158, 2020.

T. Rogers, G. Holmes, E. Cross, and K. Worden. On a grey box modelling framework for nonlinear system identification. In *Special Topics in Structural Dynamics, Volume 6*, pages 167–178. Springer, 2017.

D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3), 1988.

L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

M. Schoukens and J. P. Noël. Three benchmarks addressing open challenges in nonlinear system identification. *IFAC-PapersOnLine*, 50(1):446–451, 2017.

A. Svensson and T. B. Schön. A flexible state–space model for learning nonlinear dynamical systems. *Automatica*, 80:189–199, 2017.

P. Van Overschee and B. De Moor. N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1): 75–93, 1994.

Y. Wang. A new concept using lstm neural networks for dynamic system identification. In *2017 American Control Conference (ACC)*, pages 5324–5329, 2017.

E. Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.