Integrated Neural Networks for Nonlinear Continuous-Time System Identification

Bojan Mavkov [1], Marco Forgione[1] and Dario Piga[1]

***Abstract*— This paper introduces a novel neural network architecture, called *Integrated Neural Network* (INN), for direct identification of nonlinear continuous-time dynamical models in state-space representation. The proposed INN is used to approximate the continuous-time state map, and it consists of a feed-forward network followed by an integral block. The unknown parameters are estimated by minimizing a properly constructed dual-objective criterion. The effectiveness of the proposed methodology is assessed against the Cascaded Tanks System benchmark.**

I. INTRODUCTION

Direct identification of *continuous-time* (CT) dynamical systems from sampled data is nowadays a mature research topic which has attracted the attention of many researchers in the system and control community. Successful applications and complete reviews of direct CT identification methods can be found in the works [1–6], in the special issue [7], and in the book [8]. However, the methodologies proposed in the above-cited works can only be applied to the identification of dynamical systems with linear input-output relationships, such as linear time-invariant, linear time-varying and linear parameter-varying systems. On the other hand, the identification of general nonlinear dynamical systems is still an open and challenging research problem. In this paper, we address the problem of direct continuous-time identification of general non-linear state-space models, where the state and the output mappings are described by artificial neural networks.

For continuous-time neural state-space identification, a straightforward approach is to discretize the *ordinary differential equation* (ODE) representing the state mapping function of the dynamical model and then minimize the simulation error with respect to the model parameters [9]. The resulting procedure turns out to be very similar to the ones used for *discrete-time* (DT) system identification with *Recurrent Neural Network* (RNN) architectures. Training is performed by applying the *back-propagation through time* algorithm [10], [11]. Other approaches for discrete-time nonlinear system identification with neural networks could alternatively be applied, such as the one-step prediction error method [12], or identification schemes using *Long Short-Term Memory* networks [13]. The main limitation of these approaches is that they cannot be parallelized, due to the inherently sequential nature of a simulation through time. Thus, their implementation on modern hardware optimized for parallel computing is often inefficient. A certain degree of parallelization can be achieved by carrying out the training procedure simultaneously on several sub-sequences extracted from the training dataset, according to the widespread (mini)batch optimization paradigm. However, special care has to be taken to ensure that the initial conditions of all these sub-sequences are compatible with the identified model dynamics over the whole training dataset. For instance, a *multiple shooting* approach has been discussed in [14], while a regularization-based approach is presented in [15].

In order to estimate a continuous-time state-space model, a novel neural network architecture, called *integrated neural network* (INN), is presented in this paper. The INN architecture combines feed-forward neural networks and the integral form of the Cauchy problem defined by the state-space model dynamics. The output mapping function is modelled as a standard feed-forward neural network. The weights of the two networks and the estimate of the (hidden) state sequence are computed simultaneously by minimizing a properly-defined non-convex multi-objective loss function. To solve this non-convex optimization problem, we take advantage of modern deep learning libraries that provide a combination of back-propagation algorithm [16] for gradient evaluation and gradient-based solvers for optimization. The proposed methodology has a flexible structure and allows easy adaptation to non-uniformly sampled data. The latter is a typical advantage of continuous-time identification with respect to DT identification.

With the INN architecture introduced in this paper, we circumvent the need to run time simulations by considering the unknown state sequence as a tunable variable to be optimized along with the state and output mappings. This allows for a full parallelization of the training algorithm. Furthermore, the derivatives of the loss function required for optimization are obtained through a plain back-propagation procedure, as opposed to the back-propagation through time required in simulation error minimization. For completeness, it is worth mentioning that the approach in [17] based on 1-D Convolutional Neural Networks can be also highly parallelized. However, unlike the proposed INN architecture, the approach in [17] considers identification of discrete-time models represented by the interconnection of finite impulse response dynamic blocks with static non-linearities.

The rest of the paper is organized as follows. The overall identification setting is outlined in Section II. The proposed methodology is described in Section III, where the *integrated*

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

[1] Bojan Mavkov, Dario Piga and Marco Forgione are with IDSIA Dalle Molle Institute for Artificial Intelligence, SUPSI-USI, Manno, Switzerland E-mail: {bojan.mavkov, marco.forgione, dario.piga}@supsi.ch

neural network is introduced and details for numerical implementation of the training algorithm are provided. Results on the cascaded two-tank system identification benchmark are reported in Section IV to show the effectiveness of the approach. Finally, conclusions and directions for future works are discussed in Section V.

II. Problem setting

Let us consider a data-generating system $\mathcal{S}$ described by the continuous-time state-space representation:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{1a}$$
$$\mathbf{x}(0) = \mathbf{x}_0 \tag{1b}$$
$$\mathbf{y}^{\mathrm{o}}(t) = g(\mathbf{x}(t)), \tag{1c}$$

where $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ and $\dot{\mathbf{x}}(t) \in \mathbb{R}^{n_x}$ are the state vector and its time derivative, respectively; $\mathbf{x}_0 \in \mathbb{R}^{n_x}$ is the state initial condition; $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ is the system input; $\mathbf{y}^{\mathrm{o}}(t) \in \mathbb{R}^{n_y}$ is the (noise-free) system output at time $t \in \mathbb{R}$; $f(\cdot,\cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, and $g(\cdot) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ are the state and output mappings, respectively.

A training dataset $\mathcal{D}$ consisting of N input samples $\{\mathbf{u}(t_0),\ \mathbf{u}(t_1), \dots,\ \mathbf{u}(t_{N-1})\}$ and (noisy) output samples $\{\mathbf{y}(t_0),\ \mathbf{y}(t_1), \dots,\ \mathbf{y}(t_{N-1})\}$, gathered at time instants $T = \{t_0 = 0, t_1, \dots,\ t_{N-1}\}$ from an experiment on the dynamical system $\mathcal{S}$ is available. The measured output samples $\mathbf{y}(t_k)$ at time instant t_k, $k = 0, 1, \dots, N-1$ are corrupted by a zero-mean noise η_k , *i.e.*, $\mathbf{y}(t_k) = \mathbf{y}^{\mathrm{o}}(t_k) + \eta_k$. We assume that the input $\mathbf{u}(t)$ can be reconstructed (or reasonably approximated) for all time instants $t \in [0\ \ t_{N-1}] \subset \mathbb{R}$ from the measured samples $\{\mathbf{u}(t_0),\ \mathbf{u}(t_1), \dots,\ \mathbf{u}(t_{N-1})\}$.

III. Integrated neural network

A. Modelling

Our modelling approach relies on the representation of f as a feed-forward neural network $\mathcal{N}_f$, which is fed by the system input $\mathbf{u}(t)$ and the (estimated) state $\hat{\mathbf{x}}(t)$ at time t, and returns the estimated state time-derivative $\dot{\hat{\mathbf{x}}}(t)$, *i.e.*,

$$\dot{\hat{\mathbf{x}}} = \mathcal{N}_f(\hat{\mathbf{x}}(t), \mathbf{u}(t); \mathcal{W}_f), \tag{2}$$

where $\mathcal{W}_f \in \mathbb{R}^{n_f}$ denotes the neural network parameters to be identified. Similarly, the output mapping g is represented by a feed-forward neural network $\mathcal{N}_g$, fed by $\hat{\mathbf{x}}(t)$ and returning the model output $\hat{\mathbf{y}}(t)$ at time t, *i.e.*,

$$\hat{\mathbf{y}}(t) = \mathcal{N}_g(\hat{\mathbf{x}}(t); \mathcal{W}_g), \tag{3}$$

where $\mathcal{W}_g \in \mathbb{R}^{n_g}$ denotes the parameters of the network. Note that, in many cases, the output map g is actually known (*e.g.*, when the system's outputs correspond to a subset of the state variables). Obviously, in these cases, the output mapping can be fixed and there is no need to estimate the network $\mathcal{N}_g$.

The resulting neural dynamical model is then given by:

$$\dot{\hat{\mathbf{x}}}(t) = \mathcal{N}_f(\hat{\mathbf{x}}(t), \mathbf{u}(t); \mathcal{W}_f) \tag{4a}$$
$$\hat{\mathbf{x}}(0) = \mathbf{x}_0 \tag{4b}$$
$$\hat{\mathbf{y}}(t) = \mathcal{N}_g(\hat{\mathbf{x}}(t); \mathcal{W}_g). \tag{4c}$$

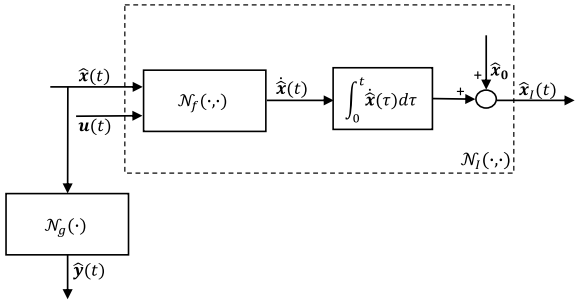


Fig. 1: Block diagram representing the solution $\hat{\mathbf{x}}(t)$ of (4).

In order to fit the neural network parameters to the training dataset $\mathcal{D}$, one possibility is to simulate (4) using a numerical ODE solution scheme, and then to minimize the *simulation error* norm penalizing the distance between measured and simulated outputs, with respect to the neural network parameters $\mathcal{W}_f$ and $\mathcal{W}_g$.

In this paper, we introduce an alternative method exploiting the *integral form* of the Cauchy problem (4a)-(4b), by defining an *integrated neural network* network $\mathcal{N}_I$ as:

$$\hat{\mathbf{x}}_I(t) = \mathcal{N}_I(\hat{\mathbf{x}}(t), \mathbf{u}(t), \mathcal{W}_f) \tag{5}$$

with $\mathcal{N}_I(\hat{\mathbf{x}}(t), \mathbf{u}(t), \mathcal{W}_f) = \hat{\mathbf{x}}(0) + \int_0^t \mathcal{N}_f(\hat{\mathbf{x}}(\tau), \mathbf{u}(\tau); \mathcal{W}_f) d\tau$.

If the state sequence $\hat{\mathbf{x}}(t)$ feeding the integrated neural network $\mathcal{N}_I$ is actually generated by model (4), then the signal $\hat{\mathbf{x}}_I(t)$ exactly matches $\hat{\mathbf{x}}(t)$, i.e.,

$$\hat{\mathbf{x}}(t) = \hat{\mathbf{x}}_I(t) \qquad \forall t \in [t_0\ t_{N-1}]. \tag{6}$$

The block diagram in Fig. 1 is a representation of (5), along with the output equation (4c) producing $\hat{\mathbf{y}}(t)$.

B. Fitting criterion

In our approach, the neural network weights $\mathcal{W}_f, \mathcal{W}_g$ and the state signal $\hat{\mathbf{x}}(t), t \in [t_0\ t_{N-1}]$ are tunable parameters to be optimized *altogether* according to a dual objective. First, the model output should be close to the measurement in $\mathcal{D}$. This objective is represented by a *fitting term* in the cost function penalizing the distance between $\hat{\mathbf{y}}(t_k)$ and $\mathbf{y}(t_k)$, $k = 0, 1, \dots, N-1$. Second, the state signal $\hat{\mathbf{x}}$ should be compatible with the model dynamics (4). This is promoted by an additional *regularization term* in the cost function penalizing the distance between $\hat{\mathbf{x}}_I(t)$ and $\hat{\mathbf{x}}(t)$, where $\hat{\mathbf{x}}_I$ is defined as in (5).

The following minimization problem is thus formulated:

$$\min_{\hat{\mathbf{x}}, \mathcal{W}_f, \mathcal{W}_g} J(\hat{\mathbf{x}}, \mathcal{W}_f, \mathcal{W}_g), \tag{7a}$$

where

$$J = \underbrace{\sum_{k=0}^{N-1} \|\overbrace{\hat{\mathbf{y}}(t_k) - \mathbf{y}(t_k)}^{\mathbf{e}_y}\|^2}_{J_{\mathbf{y}}} + \alpha \underbrace{\int_{t_0}^{t_{N-1}} \|\overbrace{\hat{\mathbf{x}}_I(\tau) - \hat{\mathbf{x}}(\tau)}^{\mathbf{e}_x}\|^2\, d\tau}_{J_{\mathbf{x}}}, \tag{7b}$$

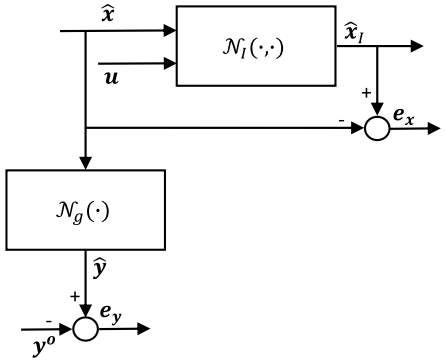


Fig. 2: Block diagram representing the loss function for the proposed integrated neural network.

$$\hat{\mathbf{y}}(t_k) = \mathcal{N}_g(\hat{\mathbf{x}}(t_k);\ \mathcal{W}_g), \tag{7c}$$

$$\hat{\mathbf{x}}_I(t) = \hat{\mathbf{x}}(0) + \int_0^t \mathcal{N}_f(\hat{\mathbf{x}}(\tau), \mathbf{u}(\tau);\ \mathcal{W}_f)\ d\tau. \tag{7d}$$

The weighting constant $\alpha > 0$ acts as a regularization hyper-parameter balancing the relative importance of the fitting objective J_y and the regularization objective J_x. Fig. 2 is a block diagram representation of the operations required to compute the cost J in (7b).

C. Numerical implementation

Note that the optimization problem (7) is actually *infinite-dimensional* as one of the optimization variable is the continuous-time state signal $\hat{\mathbf{x}}(t) \in \mathbb{R}^{n_x},\ t \in [t_0\ t_{N-1}]$. In order to transform (7) into a finite-dimensional problem amenable for numerical optimization, $\hat{\mathbf{x}}(t)$ is approximated using a finite-dimensional parametrization. For the sake of exposition, we adopt in this work a simple *piecewise constant* parametrization, where $\hat{\mathbf{x}}(t)$ is constant on the intervals $[t_k\ t_{k+1}]$, $k = 0, 1, \ldots, N-1$. In general, different parametrizations for $\hat{\mathbf{x}}$ such as piecewise linear of polynomial could be adopted. Moreover, the intervals for the piecewise definition of $\hat{\mathbf{x}}$ do not necessarily correspond to the ones induced by the measurements in $\mathcal{D}$.

Furthermore, the integrals in (7b) and (7d) are approximated by applying a numerical integration scheme. For the sake of simplicity, we adopt the classical *rectangular approximation* rule for the numerical integration of both integrals. Thus, the *piecewise constant* parametrization of $\hat{\mathbf{x}}(t)$ and the *rectangular quadrature* of the integrals yield:

$$\int_{t_0}^{t_{N-1}} \|\hat{\mathbf{x}}_I(\tau) - \hat{\mathbf{x}}(\tau)\|^2 d\tau \approx \sum_{k=1}^{N-1} \|\hat{\mathbf{x}}_I(t_k) - \hat{\mathbf{x}}(t_k)\|^2 \Delta t_k, \tag{8}$$

where $\Delta t_k = t_k - t_{k-1}$, and (7d) is approximated as:

$$\hat{\mathbf{x}}_I(t_k) \approx \mathbf{x}_0 + \sum_{j=0}^{k-1} \Delta t_{j+1} \mathcal{N}_f(\hat{\mathbf{x}}(t_j), \mathbf{u}(t_j);\ \mathcal{W}_f). \tag{9}$$

Algorithm 1 Training neural network through gradient descent

Inputs: training dataset $\mathcal{D}$; number of gradient-descent iterations n; learning rate λ; weighting constant α.

1. **Initialize** the neural network parameters $\mathcal{W}_f$, $\mathcal{W}_g$ and the state sequence $\hat{\mathbf{x}}$.
2. **for** $j = 0, .., n-1$ **do**
   2.1. $\hat{\mathbf{x}}_I(t_0) \leftarrow \hat{\mathbf{x}}(t_0)$
   2.2. **for** $k = 0, 1, \ldots, N-1$ **do**

$$\hat{\mathbf{y}}(t_k) \leftarrow \mathcal{N}_g(\hat{\mathbf{x}}(t_k); \mathcal{W}_g)$$
$$\Delta \mathbf{x}_k \leftarrow \mathcal{N}_f(\hat{\mathbf{x}}(t_k), \mathbf{u}(t_k); \mathcal{W}_f)(t_{k+1} - t_k)$$
$$\hat{\mathbf{x}}_I(t_{k+1}) \leftarrow \hat{\mathbf{x}}_I(t_k) + \Delta \mathbf{x_k}$$

3. **Define** the cost J from (7b) and (8), i.e.,

$$J(\mathcal{W}_f, \mathcal{W}_g, \hat{\mathbf{x}}) \leftarrow \sum_{k=0}^{N-1} \|\hat{\mathbf{y}}(t_k) - \mathbf{y}(t_k)\|^2 + \alpha \sum_{k=1}^{N-1} \|\hat{\mathbf{x}}_I(t_k) - \hat{\mathbf{x}}(t_k)\|^2 (t_k - t_{k-1})$$

4. **Compute** the gradients $\nabla_{\mathcal{W}_f} J,\ \nabla_{\mathcal{W}_g} J,\ \nabla_{\mathbf{x}} J$ through back-propagation
5. **Update** the network parameters and the hidden state:

$$\mathcal{W}_f \leftarrow \mathcal{W}_f - \lambda \nabla_{\mathcal{W}_f} J$$
$$\mathcal{W}_g \leftarrow \mathcal{W}_g - \lambda \nabla_{\mathcal{W}_g} J$$
$$\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} - \lambda \nabla_{\hat{\mathbf{x}}} J$$

Output: neural network parameters $\mathcal{W}_f$ and $\mathcal{W}_g$.

Note that, for implementation convenience, the sum in (9) can be constructed recursively as:

$$\hat{\mathbf{x}}_I(t_k{+}1) = \hat{\mathbf{x}}_I(t_k) + \overbrace{\Delta t_{k+1} \mathcal{N}_f(\hat{\mathbf{x}}(t_k), \mathbf{u}(t_k); \mathcal{W}_f)}^{\Delta \mathbf{x}_k}. \tag{10}$$

In general, other *quadrature rules* such as the trapezoidal rule or *Gaussian quadrature* [18] could be adopted for approximation of the integrals in (7b) and (7d).

D. Algorithm overview

Algorithm 1 describes the steps required to estimate the network parameters $\mathcal{W}_f$ and $\mathcal{W}_g$ by minimizing the loss J defined in (7b) over a number of n gradient-descent iterations.

At the beginning of the procedure (Step 1), all the optimization parameters $\mathcal{W}_f$, $\mathcal{W}_g$, and $\hat{\mathbf{x}}$ are initialized to some selected (or random) values. It is import to remark that, for numerical implementation and with some abuse of notation, the optimization variable $\hat{\mathbf{x}}$ in Algorithm 1 denotes the finite-dimensional representation of the state signal $\hat{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} = \{\hat{\mathbf{x}}(t_0), \ldots, \hat{\mathbf{x}}(t_{N-1})\}$. The measured input and output values in the dataset $\mathcal{D}$ may also be normalized, if required.

Then, at each iteration $j = 0, \ldots, n-1$ of the gradient-based optimization, the following operations are repeated.

In Step 2, the neural network output $\hat{\mathbf{y}}$ and the integrated state $\hat{\mathbf{x}}_I$ are evaluated according to (7c) and (10),

respectively.[1] The cost function J is then computed (Step 3) as in (7b) and (8), and its gradient with respect to the optimization variables $\mathcal{W}_f$, $\mathcal{W}_g$, and $\hat{\mathbf{x}}$ is obtained through back-propagation (Step 4), based on the computational graph in Fig. 2. Lastly, the optimization variables $\mathcal{W}_f$, $\mathcal{W}_g$, and $\hat{\mathbf{x}}$ are updated according to the *gradient descent* optimization algorithm with learning rate λ (Step 5). Enhanced variants of the vanilla gradient descent algorithm such as RMSProp and Adam [19] can also be adopted in Step 5.

The computational graph corresponding to the loss minimized in Algorithm 1 is sketched in Fig. 3. It is evident from this graph that all the neural network function evaluations can be performed in parallel, as the state sequence $\hat{\mathbf{x}}(t)$ is a free optimization variable. Only the cumulative sum required to obtain recursively $\hat{\mathbf{x}}_I(t_{k+1})$ as $\hat{\mathbf{x}}_I(t_{k+1}) = \hat{\mathbf{x}}_I(t_k) + \Delta x_k$ has to be computed sequentially. However, this simple operation has a negligible computational cost compared to neural network evaluations.

For the sake of comparison, the computational graph of a simulation error minimization training strategy based on the forward Euler numerical scheme for ODE integration is shown in Figure 4. It is evident that in this case the neural network function evaluations have to be performed sequentially as the simulated state $\hat{\mathbf{x}}(t_{k+1})$ at time step t_{k+1} depends on the previously simulated state $\hat{\mathbf{x}}(t_k)$ at time step t_k. This does not allow a parallel implementation and generally results in a longer algorithm runtime.

Remark 1: It is possible to take adavantage of a previously available model to initialize the optimization variables in Step 1 of Algorithm 1. For instance, an initial estimate of state sequence $\hat{\mathbf{x}}$ may be obtained by optimizing the cost function J with respect to $\hat{\mathbf{x}}$ only, while fixing the system dynamics to the initial model. This type of initialization is proposed in [20] for general nonlinear identification of state space models, with an initial linear model obtained as the Best Linear Approximation of the training data.

Remark 2: In Algorithm 1, the entire training dataset is processed at each iteration j of the optimization loop. Unlike in simulation error minimization, (mini)-batch training is not required in our approach to increase the level of parallelization. Nonetheless, executing the training procedure on shorter training sequences could still be beneficial for different reasons. First, as shown in [14], the Lipschitz constant of the objective function may blow up exponentially with the training sequence length for *non-contractive* system dynamics. Thus, the optimization problem (7) may be better posed when solved over subsequences extracted from the training dataset. Furthermore, for large-scale training datasets, minibatch training may be required due to memory limitations of the hardware.

IV. Case study

A. System description

The proposed method is evaluated on a publicly available system identification benchmark proposed in `www.nonlinearbenchmark.com`. In particular, the *Cascaded Tanks System* (CTS) benchmark thoroughly described in [21], [22] is considered here.

The CTS (schematized in Fig. 5) consists of two cascaded tanks with free outlets. The system input is the water flow provided by a pump feeding water from a reservoir into the upper tank, while the measured output is the water level in the lower tank. When one of the tanks is completely filled, water overflow occurs. Furthermore, when an overflow occurs in the upper tank, part of the water flows into the lower tank, while the rest leaves the system.

An approximate nonlinear state-space model of the CTS

[1] Note that (10) is an approximation of (7d).

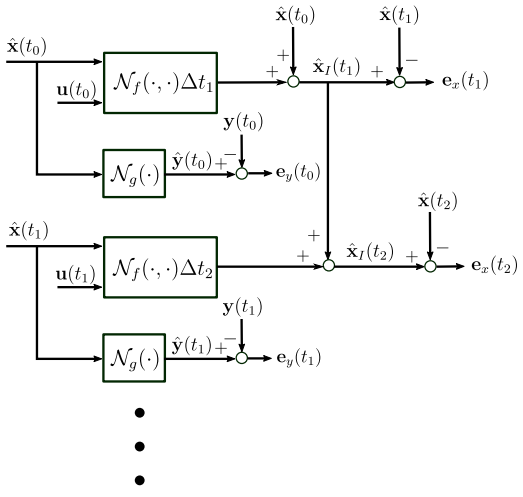


Fig. 3: Computational graph corresponding to the Integrated Neural Network training described in Algorithm 1.

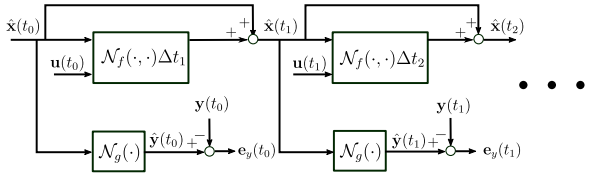


Fig. 4: Computational graph corresponding to simulation error minimization, with system dynamics integrated using the forward Euler numerical scheme.

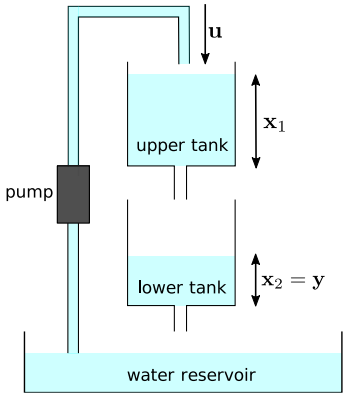


Fig. 5: Schematics of the cascaded two-tank system.

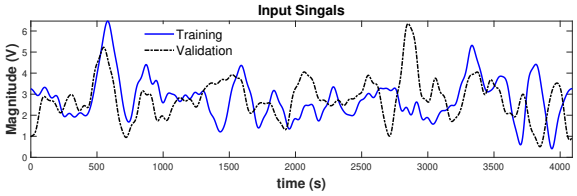


Fig. 6: Plot of the inputs of the system. The blue trace represents the input in the training set and the black dashed line represents the input in the validation set.

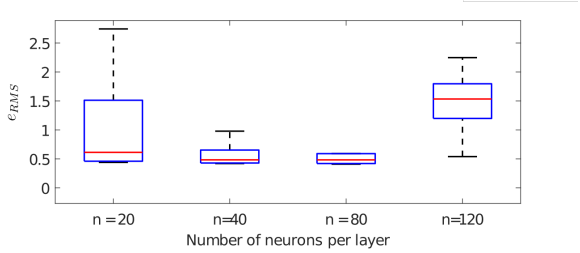


Fig. 7: Boxplot of e_{RMS} for different number of neurons per layer.

can be derived as in [21], exploiting Bernoulli's principle and conservation of mass (but ignoring the water overflow effect):

$$\dot{\mathbf{x}}_1(t) = -k_1\sqrt{\mathbf{x}_1(t)} + k_4\mathbf{u}(t) \tag{11a}$$

$$\dot{\mathbf{x}}_2(t) = k_2\sqrt{\mathbf{x}_1(t)} - k_3\mathbf{u}(t) \tag{11b}$$

$$\mathbf{y}(t) = \mathbf{x}_2(t), \tag{11c}$$

where $\mathbf{u}(t)$ is the input signal that controls the water pump flow from the reservoir into the upper tank; $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ are the state variables of the system representing water level in the upper and lower tank, respectively; $\mathbf{y}(t)$ is the output signal that measures the water level in the lower tank; and k_1, k_2, k_3, k_4 are unknown coefficients depending on the CTS configuration.

Both the training and the validation datasets consist of $N = 1024$ samples and the sampling time is $\Delta t = 4$ s. The input $\mathbf{u}$ and the output $\mathbf{y}$ are expressed in Volts, as they correspond to raw actuator and sensor readings, respectively. The plot of the inputs of the training and validation data is given in Fig. 6.

The goal of this benchmark is to estimate the dynamics of the system as accurately as possible, using only the training data. The efficiency of the estimation algorithm is then tested on the validation dataset. As proposed in [21], the *Root Mean Square Error* (RMSE) between the measured output $\mathbf{y}$ and the open-loop simulated output $\hat{\mathbf{y}}$ is used as performance index:

$$e_{RMS} = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} \|\mathbf{y}(t_k) - \hat{\mathbf{y}}(t_k)\|^2}. \tag{12}$$

B. Neural model structure

Based on the proposed identification architecture and on the physics-based model given in (11), the following neural network model structure is chosen:

$$\dot{\hat{\mathbf{x}}}(t) = \mathcal{N}_f(\hat{\mathbf{x}}(t), \mathbf{u}(t);\ \mathcal{W}_f) \tag{13a}$$

$$\hat{\mathbf{y}}(t) = \hat{\mathbf{x}}_2(t), \tag{13b}$$

with $\mathbf{x}(t) \in \mathbb{R}^2$. This model structure is motivated by the observation that (i) the physics-based model has two state variables and (ii) the second one corresponds to the measured output.

The feed-forward neural network $\mathcal{N}_f$ has three input nodes (corresponding to the two state components and the system input $\mathbf{u}$); two hidden layers with 80 nodes with a sigmoid non-linearity; and two linear output nodes corresponding to the two components of the state equation (13a).

C. Algorithm setup

The proposed identification algorithm is implemented using the PyTorch Deep Learning framework [23]. In order to adjust data with different magnitude scales, the input and the output data are normalized to have zero mean and unitary standard deviation. In Algorithm 1, gradient-based optimization is performed using the Adam method [19] over $n = 4 \cdot 10^5$ iterations, fixing the learning rate to $\lambda = 10^{-5}$. The hyper-parameter α is set equal to $1/8$. The code for reproducing this example is available in the following GitHub repository: `https://github.com/bmavkov/INN-for-Identification`

D. Results

The achieved results may vary due to the random initial conditions assigned to the optimization variables and the non-convexity of the objective function. Thus, multiple executions with different initial conditions and a varying number of neurons per layer were performed. The neural network consists of 3 layers, while the number of neurons varied in the range of $20 - 120$ per each layer. The boxplots of the RMSE values of the validation data are shown in Fig. 7.

The measured output and the simulated model output in the training and in the validation datasets are reported in Fig. 8a and Fig. 8b, respectively. The resulting RMSE index for the best preforming model is $e_{RMS} = 0.36$ for the training dataset and $e_{RMS} = 0.41$ for the validation dataset.

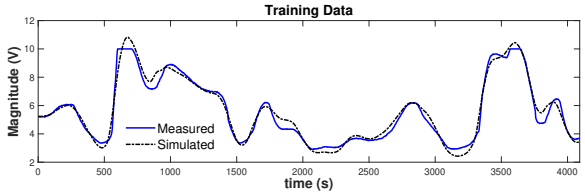
We observe that the achieved performance index is similar in the training and in the validation datasets, and that the results are in line with the ones obtained using other state-of-the-art identification methods applied to this benchmark [24–29]. Detailed listing of the resulting RMSE values of these algorithms are given in Table I.

V. Conclusions

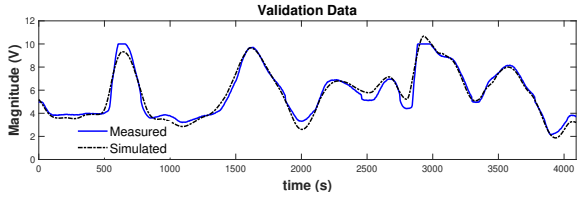
A novel neural network architecture has been presented for the identification of nonlinear dynamical systems in continuous-time state-space forms. The proposed structure consists of a feed-forward neural network followed by an integral block. The network is trained using gradient-descent

Method	e_{RMS}
Svensson et al.[29]	0.45
Volt.FB (Schoukens et at.)[24]	0.39
NOMAD (Brunot et al.) [25]	0.37
PNLSS-I (Relan et al.) [27]	0.45
NLSS2 (Relan et al.) [27]	0.34
PWARX (Mattsson et al.) [28]	0.35
INN Training data	0.36
INN Test data	0.41

TABLE I: Comparison of different identification methods.



(a) Output of the training data. $RMSE$: 0.36



(b) Output of the validation data. $RMSE$: 0.41

Fig. 8: Plots of the simulated and measured outputs. The black dashed trace represents the simulated output of the estimated model and the blue trace represents the true output.

optimization, with gradients computed through standard *back-propagation*.

Current research activities are focused on the extension of the proposed approach for direct identification of systems described by partial differential equations, which requires handling infinite-dimensional state-space vectors.

REFERENCES

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

[2] H. Garnier, M. Mensler, and A. Richard, "Continuous-time model identification from sampled data: implementation issues and performance evaluation," *International journal of Control*, vol. 76, no. 13, pp. 1337–1357, 2003.

[3] 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*, vol. 19, no. 1, pp. 64–76, 2011.

[4] H. Garnier and P. C. Young, "The advantages of directly identifying continuous-time transfer function models in practical applications," *International Journal of Control*, vol. 87, no. 7, pp. 1319–1338, 2014.

[5] 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*, vol. 11, no. 4, pp. 457–465, 2017.

[6] 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.

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

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

[9] S. Chen and S. Billings, "Neural networks for nonlinear dynamic system modelling and identification," *International journal of control*, vol. 56, no. 2, pp. 319–346, 1992.

[10] B. G. Horne and C. L. Giles, "An experimental comparison of recurrent neural networks," in *Advances in Neural Information Processing Systems*, 1995, pp. 697–704.

[11] R. J. Williams and D. Zipser, *Oxford Handbook of Innovation*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995, ch. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity.

[12] S. Chen, S. Billings, and P. Grant, "Non-linear system identification using neural networks," *International Journal of Control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[13] Y. Wang, "A new concept using LSTM neural networks for dynamic system identification," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5324–5329.

[14] A. H. Ribeiro, K. Tiels, J. Umenberger, T. B. Schön, and L. A. Aguirre, "On the smoothness of nonlinear system identification," *arXiv preprint arXiv:1905.00820*, 2019.

[15] M. Forgione and D. Piga, "Model structures and fitting criteria for system identification with neural networks," *arXiv preprint arXiv:1911.13034*, 2019.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[17] C. Andersson, A. H. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön, "Deep convolutional networks in system identification," in *58th IEEE Conference on Decision and Control*, 2019, pp. 3670–3676.

[18] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Springer Science & Business Media, 2010, vol. 37.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] A. Marconato, J. Sjöberg, J. A. Suykens, and J. Schoukens, "Improved initialization for nonlinear state-space modeling," *IEEE Transactions on instrumentation and Measurement*, vol. 63, no. 4, pp. 972–980, 2013.

[21] M. Schoukens, P. Mattson, T. Wigren, and J. P. Noël, "Cascaded tanks benchmark combining soft and hard nonlinearities," in *Workshop on Nonlinear System Identification Benchmarks*, Brussels, Belgium, 2016, pp. 20–23.

[22] M. Schoukens and J. P. Noël, "Three benchmarks addressing open challenges in nonlinear system identification," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 446–451, 2017.

[23] 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," 2017.

[24] M. Schoukens and F. G. Scheiwe, "Modeling nonlinear systems using a Volterra feedback model," in *Workshop on Nonlinear System Identification Benchmarks*, 2016.

[25] M. Brunot, A. Janot, and F. Carrillo, "Continuous-time nonlinear systems identification with output error method based on derivative-free optimisation," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 464–469, 2017.

[26] K. Worden, R. Barthorpe, E. Cross, N. Dervilis, G. Holmes, G. Manson, and T. Rogers, "On evolutionary system identification with applications to nonlinear benchmarks," *Mechanical Systems and Signal Processing*, vol. 112, pp. 194–232, 2018.

[27] R. Relan, K. Tiels, A. Marconato, and J. Schoukens, "An unstructured flexible nonlinear model for the cascaded water-tanks benchmark," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 452–457, 2017.

[28] P. Mattsson, D. Zachariah, and P. Stoica, "Identification of cascade water tanks using a PWARX model," *Mechanical systems and signal processing*, vol. 106, pp. 40–48, 2018.

[29] A. Svensson and T. B. Schön, "A flexible state–space model for learning nonlinear dynamical systems," *Automatica*, vol. 80, pp. 189 – 199, 2017.