Automatica 129 (2021) 109666

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Learning nonlinear state–space models using autoencoders☆

Daniele Masti *, Alberto Bemporad

IMT School for Advanced Studies, Piazza San Francesco 19, Lucca, Italy

ARTICLE INFO

Article history:
Received 11 April 2020
Received in revised form 8 February 2021
Accepted 16 March 2021
Available online 4 May 2021

Keywords:
Identification methods
Model fitting
Identification for control
Neural networks

ABSTRACT

We propose a methodology for the identification of nonlinear state–space models from input/output data using machine-learning techniques based on autoencoders and neural networks. Our framework simultaneously identifies the nonlinear output and state-update maps of the model. After formulating the approach and providing guidelines for tuning the related hyper-parameters (including the model order), we show its capability in fitting nonlinear models on different nonlinear system identification benchmarks. Performance is assessed in terms of open-loop prediction on test data and of controlling the system via nonlinear model predictive control (MPC) based on the identified nonlinear state–space model.

1. Introduction

Nonlinear system identification has gained increasing popularity in recent years (Pillonetto, Dinuzzo, Chen, Nicolao, & Ljung, 2014; Schoukens & Ljung, 2019), also due to massive advances in machine-learning methods for nonlinear function regression. Such methods have been employed with high success for extending classical linear techniques to nonlinear systems, such as for the estimation of neural autoregressive models with exogenous inputs (NARX) (Schoukens & Ljung, 2019) and of reproducing kernel Hilbert space (RKHS) models (Pillonetto et al., 2014), as well as for piecewise-affine regression (Breschi, Piga, & Bemporad, 2016), and for developing novel approaches based on long short-term memory (LSTM) neural networks (Wang, 2017).

Most of the aforementioned techniques, however, identify nonlinear models in *input/output* form, without an explicit definition of a (minimal) Markovian state. On the other hand, *state–space* models are the basis for most modern control design techniques, such as nonlinear control, model predictive control (MPC), as well as for noise filtering and smoothing, such as extended Kalman filtering (EKF).

1.1. Machine-learning methods for the identification of state–space models

The idea of applying machine-learning approaches to identify state–space representations of a dynamical system from input/output data has been widely explored in the literature. For example, we mention here the classical dynamic mode decomposition (DMD) and refer the reader to the review in Lu and Zavala (2020). Learning a state–space model typically requires a nonlinear transformation of a vector of past input/output samples to a state vector. One drawback of many machine-learning methods performing such a dimensionality reduction is that the resulting state–space model (a.k.a. "latent representation") may result in non-regular and stiff mappings, therefore creating issues when the model is used to design a controller and/or an observer, for instance when the linearization of the model is used. Although exceptions exist (Simpson, Dervilis, & Chatzi, 2020), most of the proposed solutions address this issue by relying on variations of two families of approaches (Wehmeyer & Noé, 2018): (*i*) imposing a "regular geometry" on the learned latent space; (*ii*) envision learning schemes in which the capability of the latent representation to predict future output values is directly taken into account during the learning phase.

Among the first family, many works are inspired by the well known variational autoencoders (Kingma & Welling, 2019). Here, the latent representation is learned so that the distribution of the resulting states is a prescribed one, usually a Gaussian one. Different types of VAE have been widely used both for system identification (Gedon, Wahlström, Schön, & Ljung, 2020; Kandukuri, Achterhold, Möller, & Stückler, 2020; Karl, Soelch, Bayer,

☆ This paper was partially supported by the Italian Ministry of University and Research under the PRIN'17 project "Data-driven learning of constrained control systems" , contract no. 2017J89ARP. The material in this paper was partially presented at the 57th IEEE Conference on Decision and Control, December 17–19, 2018, Miami Beach, Florida, USA. This paper was recommended for publication in revised form by Associate Editor Gianluigi Pillonetto under the direction of Editor Torsten Söderström.

* Corresponding author.
E-mail addresses: daniele.masti@imtlucca.it (D. Masti), alberto.bemporad@imtlucca.it (A. Bemporad).

https://doi.org/10.1016/j.automatica.2021.109666
0005-1098/

& Van der Smagt, 2016; Krishnan, Shalit, & Sontag, 2015; Watter, Springenberg, Boedecker, & Riedmiller, 2015) and in reinforcement learning (Okada & Taniguchi, 2020; van Hoof, Chen, Karl, van der Smagt, & Peters, 2016).

The contributions related to the second family of approaches are often extensions of the DMD idea. We mention here papers based on the Koopman formalism (Li, Dietrich, Bollt, & Kevrekidis, 2017; Ping, Yin, Li, Liu, & Yang, 2020; Xiao, Zhang, Xu, Liu, & Liu, 2020) and the contributions (Lusch, Kutz, & Brunton, 2018; Takeishi, Kawahara, & Yairi, 2017), which use autoencoders in conjunction with the Koopman operator for learning representations of autonomous systems. Closely related to the DMD idea is also the so-called "SINDy" framework (Champion, Lusch, Kutz, & Brunton, 2019). In these frameworks, the original input/output signals are lifted to a possibly higher-dimensional space, in which the dynamic evolution of the system can be modeled in linear time-invariant (LTI) form or, more rarely, in bilinear form. A characteristic of the approach is that the resulting state–space dimension can be larger than that of the original vector of past input/output samples.

Some techniques combine methods from both families. For instance, in Fraccaro, Kamronn, Paquet, and Winther (2017) and Rangapuram et al. (2018) recurrent neural networks are trained to predict the evolution of the state-update maps that govern a time-varying time series, while in Chung et al. (2015) a variational recurrent neural network architecture is envisioned.

1.2. Contribution

In this paper we propose a methodology that uses artificial neural networks (ANNs), and in particular autoencoders (AEs) (Hinton & Salakhutdinov, 2006), to learn a nonlinear model in state–space from a given input/output dataset. The main idea is the following: we train an AE that reproduces a collection of output signals from a collection of input and output signals and take the central layer of the AE as the state vector. Such a dimensionality reduction problem is solved jointly with problem of learning the nonlinear state-update function (parameterized by a deep neural network) that maps the values of such a state into its next values.

Contrary to the contributions (Li et al., 2017; Lusch et al., 2018; Otto & Rowley, 2019), we do not require the dynamics to be linear in the learned latent space. On the other hand, to ease the design of controllers and state observers, we also consider a quasi linear parameter-varying (LPV) formulation of the model, in which each of the coefficients of the state-update matrices is the output of an ANN.

In contrast with the works (Chung et al., 2015; Karl et al., 2016; Krishnan et al., 2015; Watter et al., 2015), we do not rely on variational inference arguments, as we do not impose specific structures on the learned latent space. Our approach brings two advantages: (*i*) it avoids making an assumption about the distribution to impose; (*ii*) as we will show in Section 4.3, it enables the use of classical shrinkage operators to tune some of the hyper-parameters of the method, that would be instead hard to adopt within the variational framework.

To improve both the accuracy of the resulting models and the numerical stability of the approach, in this paper we also use a fitting criterion based on multi-step predictions. The main hyper-parameter of the approach is the order of the state–space model, that must be chosen (as typically in system identification methods) to obtain a tradeoff between model accuracy in reproducing test data and model complexity. We provide a heuristic approach based on group sparsification methods to help tuning the number of states to include in the nonlinear model and the number of past input and outputs the autoencoder consumes.

Preliminary ideas and results related to the contents of this paper were presented in Masti and Bemporad (2018), where we proposed AEs to extract a compressed representation of I/O data to be used as state representation for a classical one-step-ahead Prediction Error Method (PEM) approach. The method we present in this paper is also related to the approaches presented in some very recent contributions (Beintema, Toth, & Schoukens, 2020, 2020; Drgona, Tuor, Chandan, & Vrabie, 2020), which also develop ideas related to the preliminary work presented in Masti and Bemporad (2018).

The paper is organized as follows. In Section 2 we formulate the nonlinear identification problem we want to solve and present a solution method based on autoencoders in Section 3. After detailing the learning algorithm in Section 4 and discussing the use of the identified model for state estimation and control in Section 5, we report results in Section 6 based on several nonlinear benchmark problems. Finally, we draw conclusions in Section 7.

The Python code to reproduce the results described in the paper is available at http://dysco.imtlucca.it/masti/autoencoders.

2. Nonlinear identification problem

We are given a training dataset of input/output samples $Z = \{u_0, y_0, \dots, u_N, y_N\}$ collected from a dynamical system, where $u_k \in \mathbb{R}^{n_u}$ is the vector of exogenous inputs and $y_k \in \mathbb{R}^{n_y}$ the vector of measured outputs. Our goal is to identify a dynamical model of the system in the following state–space form

$$\begin{cases} x_{k+1} &= f(x_k, u_k) \\ \hat{y}_k &= g(x_k) \end{cases} \tag{1}$$

with $x \in \mathbb{R}^{n_x}$, that, starting from an appropriate condition x_{k_0} and excited by the same inputs $u_{k_0}, \dots, u_{N-1}$, produces an output signal $\hat{y}_k$ that is as close as possible to the one y_k recorded on the system. Although we assume that the collected input and output signals may be affected by measurement noise, we do not make any particular assumption about the properties of such noise.

Given a number of past outputs $n_a \geq 1$, of past inputs $n_b \geq 1$, and a desired state dimension $n_x \geq 1$, the problem can be recast to the problem of finding a triplet of maps e, f, g, $e : \mathbb{R}^{n_l} \to \mathbb{R}^{n_x}$, $n_l \triangleq n_a n_y + n_b n_u$, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ that solves the following optimization problem:

$$\min_{e,f,g} \mathcal{L}(e, f, g, Z) \tag{2a}$$

where

$$\begin{aligned} \mathcal{L}(e, f, g, Z) \quad &= \sum_{k=k_0}^{N} L(\hat{y}_k, y_k) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & \hat{y}_k = g(x_k), \quad k = k_0, \dots, N \\ & x_{k_0} = e(I_{k_0-1}) \end{aligned} \tag{2b}$$

In (2), $L : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \to [0, +\infty)$ is a suitable loss function that penalizes the discrepancy between the predicted and the measured output, and I_k is the following information vector

$$I_k = [y_k' \ \dots \ y_{k-n_a+1}' \ u_k' \ \dots \ u_{k-n_b+1}']' \tag{3}$$

where $k_0 \triangleq \max\{n_a, n_b\}$. In (2), e is the dimensionality-reduction mapping from the vector I_k of past inputs and outputs to the state vector x_k, whose role will be further explained in the next sections.

In general problem (2) has infinitely many solutions. For example, if the data were generated by a linear system of order n, for any dimension $n_x \geq n$ all the infinitely many (possibly non-minimal) state–space realizations leading to the same transfer

function would be equally optimal. The problem of recognizing the smallest state-dimension n_x that provides an acceptable mismatch between the predictions $\hat{y}_k$ and the measured outputs y_k is of main interest and has been widely explored in the literature (Baram, 1984; Williams & Lawrence, 2007). A similar problem of data compression is also widely studied in the machine learning literature in the context of *feature extraction* (Guyon & Elisseeff, 2006). There, the goal is to reduce the dimension of the input space by identifying a nonlinear function that projects the original (large) input space into a (smaller dimensional) feature space, without losing significant information content.

In the following, we solve problem (2) by parameterizing the maps e, f, g as ANNs, due to their universal approximation properties (Barron, 1993) and efficient numerical packages available for training them.

3. State selection via autoencoders

The idea behind an autoencoder is to train an ANN to reproduce the identity mapping from a certain information vector $I_k \in \mathbb{R}^{n_I}$ to I_k itself, under the topological constraint that one of its hidden layers contains $n_x < n_I$ neurons. Such a constraint forces the network to learn a description of I_k that lives in the lower-dimensional space $\mathbb{R}^{n_x}$ without losing information. The smaller the fitting error between I_k and the reconstructed I_k, the less information is lost when passing through the network across the hidden "bottleneck" layer. As a result, when excited by an input value I_k, the corresponding value $x_k \in \mathbb{R}^{n_x}$, taken by the neurons of the bottleneck layer represents the desired lower-dimensional vector concentrating the information contained in I_k. This approach has been shown to be successful in a large variety of applications. For completely linear networks, it has been shown in Baldi and Hornik (1989) that it has a strong relation with the standard principal component analysis (PCA) technique.

3.1. Partial predictive autoencoders

Given the dataset Z of input/output samples, applying a standard autoencoder to compress the information vector I_k defined by (3) into a reduced-order vector $x_k \in \mathbb{R}^{n_x}$ would not be optimal to learn a state representation for two reasons: (*i*) it would treat the samples I_k as independent, missing the fact that consecutive samples I_k share common (time-shifted) components, and therefore fail in capturing the capability of predicting the next output y_{k+1}, and (*ii*) it would be redundant, as we are not really interested in reproducing the input signals u_{k-i+1}, $i = 1, \dots, n_b$, that are components of I_k. Therefore, we introduce here a *partial predictive autoencoder* (PPE) that maps I_{k-1} (i.e., the information available up to time $k-1$) into the following vector of outputs

$$O_k = [y_k' \ \dots \ y_{k-m}']' \tag{4}$$

with $0 \le m \le n_a$. By fitting an ANN with a hidden layer of size n_x, $n_x \le n_a n_y + n_b n_u$, that tries to predict O_k given I_{k-1}, we obtain an intermediate compressed representation $x_k \in \mathbb{R}^{n_x}$. Such a vector x_k can be treated as a model state, as it captures the information required to predict y_k from I_{k-1}, and even filter y_{k-1} (if $m \ge 1$) and smooth past outputs (if $m > 1$).

We note here that we could make x_k depend on y_k too, but we do not consider such a case in this work.

The PPE amounts to the cascade of two different ANNs: (*i*) an encoding function $e : \mathbb{R}^{n_I} \to \mathbb{R}^{n_x}$ representing the transformation from I_{k-1} (past inputs and outputs) to x_k (state vector), (*ii*) a decoding mapping $d : \mathbb{R}^{n_x} \to \mathbb{R}^{mn_y}$ from x_k to O_k, whose first n_y components constitute the desired output function $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$.

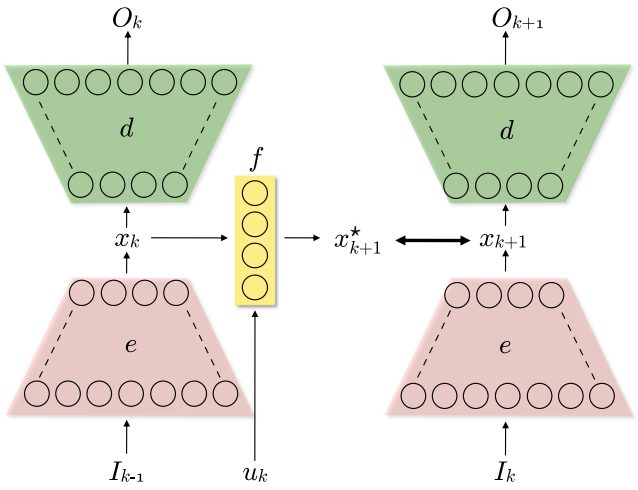


Fig. 1. Schematic representation of the computational graph of the proposed nonlinear model structure.

4. Model learning

Having defined a structure to map I_{k-1} into x_k, we also need a structure to fit a function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ mapping x_k and u_k into the next state x_{k+1}. A first approach would be to fit the PPE described above to get functions e and d, compute the set of states $x_k = e(I_{k-1})$, $k = \max(n_a, n_b) + 1, \dots, N$, and then fit a model f mapping (x_k, u_k) to x_{k+1}. We propose instead a better method that learns e, d, and f simultaneously. We define a multi-objective learning problem whose solution is a set of sub-networks implementing the state-update and output functions of the desired state–space model. The corresponding structure is schematically depicted in Fig. 1, in which we use two PPEs that share exactly the same weights (to be determined), one fed by I_{k-1} and the other by I_k. The goal is to reproduce, respectively, O_k and O_{k+1}. In this way, the generated state x_k in the first AE and x_{k+1} in the second AE will be coherent. A third ANN must be trained to map u_k and x_k into the shifted state x_{k+1}, therefore getting the state-update mapping f.

The overall training problem described above is formulated as the following optimization problem

$$\begin{aligned}
\min_{e,f,d} \quad & \sum_{k=k_0}^{N-1} \alpha \left(L_1(\hat{O}_k, O_k) + L_1(\hat{O}_{k+1}, O_{k+1}) \right) \\
& + \beta L_2(x^\star_{k+1}, x_{k+1}) + \gamma L_3(O_{k+1}, O^\star_{k+1}) \\
\text{s.t.} \quad & x_k = e(I_{k-1}), \ \ k = k_0, \dots, N \\
& x^\star_{k+1} = f(x_k, u_k), \ \ k = k_0, \dots, N-1 \\
& \hat{O}_k = d(x_k), \ \ k = k_0, \dots, N \\
& O^\star_k = d(x^\star_k), \ \ k = k_0+1, \dots, N
\end{aligned} \tag{5}$$

where L_i are loss functions, $\alpha, \beta, \gamma \ge 0$ are scalar weights, O_k is defined by (4), and I_k by (3), and $d : \mathbb{R}^{n_x} \to \mathbb{R}^{(m+1)n_y}$ is the decoder part of the PPE. Note that the output function g in (1)–(2) is retrieved from d by taking the components corresponding to y_k.

In (5), the loss function L_1 extends the original loss L in (2b), for instance $L_1(\hat{O}_k, O_k) = \sum_{j=k-m}^{k} L(\hat{y}_j, y_j)$. The loss L_2 can be seen as a relaxation of the state update equation $x_{k+1} = f(x_k, u_k)$ in (2b). The loss L_3 attempts avoiding that the error introduced by the bridge function f gets amplified by the nonlinear decoder d and results in a large deviation, on between the predicted outputs $O^\star_{k+1}$ and the measured outputs O_{k+1}. While clearly (5) may not

solve the original problem (2) exactly, it attempts to provide a good sub-optimal solution to it.

Minimizing L_3 and L_2 in (5) is the objective with the most priority, as it captures the one-step ahead properties of the model. The "vertical" objective related to L_1 is instead only ancillary and serves to guide the process of learning the correct bridge/decoder pair. This suggests that a good approach for better solving the posed learning problem (2) is to start the training procedure with small values of β, γ and a high value of α, and then use the solution as the initial guess of another instance of (5) with a smaller value of α, possibly reiterating the procedure multiple times for decreasing values of α.

In the rest of the paper we will consider that L_{1-3} are L_{MAE} loss functions.[1]

Besides deciding the *topology* and *activation functions* of the ANNs employed in the two identical PPEs, and the *learning algorithms* employed in optimizing their weights and bias terms given the available training dataset, several tuning hyper-parameters are involved in the proposed nonlinear system identification method described above. These are summarized in Table 1.

4.1. Multiple-step ahead fitting procedure

Penalizing the *one-step ahead* prediction errors in (5) does not guarantee that the identified model will provide good *open-loop* predictions. An approach to solve this issue would be to resort to a backpropagation through time (BPTT) learning scheme (Werbos et al., 1990), in which the initial estimated condition x_0 is propagated through each step k of the whole dataset. The main issue of BPTT is that it involves optimizing cost functions that are both computationally expensive to evaluate and to differentiate, and are highly nonlinear, which makes the learning problem difficult to solve (Forgione & Piga, 2020; Hochreiter & Schmidhuber, 1997). A good compromise is to resort on the so-called truncated BPTT (Forgione & Piga, 2021; Puskorius & Feldkamp, 1994), where the propagation of the estimate $\hat{x}_k$ is carried only for a limited number of steps $F \ll N$. In our identification scheme, the idea of truncated BPTT translates into the following multi-step ahead modification of (5):

$$
\begin{aligned}
\min_{e,f,d} \quad & \sum_{k=k_0}^{N-F} \left(\sum_{f=0}^{F} \alpha L_1(\hat{O}_{k+f}, O_{k+f}) \right. \\
& \left. + \sum_{f=1}^{F} \sum_{r=0}^{f-1} \beta L_2(\hat{x}_{k+f}^{k+r}, x_{k+f}) + \gamma L_3(O_{k+f}, \hat{O}_{k+f}^{k+r}) \right) \\
\text{s.t.} \quad \hat{x}_{k+f+1}^{k+r} &= f(\hat{x}_{k+f}^{k+r}, u_{k+f}), \ k = k_0, \dots, N-1 \\
& \quad f = r+1, \dots, F, \quad r = 1, \dots, f-1 \\
\hat{x}_{k+r+1}^{k+r} &= f(x_{k+r}, u_{k+r}), \ k = k_0, \dots, N-1 \\
& \quad r = 1, \dots, f-1 \\
x_{k+r} &= e(I_{k+r-1}), \ k = k_0, \dots, N \\
\hat{O}_k &= d(x_k), \ k = k_0, \dots, N \\
\hat{O}_{k+f}^{k+r} &= d(\hat{x}_{k+f}^{k+r}), \ k = k_0+1, \dots, N
\end{aligned}
\tag{6}
$$

where in (6) we consider the multi-step ahead predictions $\hat{x}_{k+f}^{k+r}$ of the state vector at step $k+f$ based on iterating the model in open-loop for $f-r$ steps from the initial state $x_{k+r} = e(I_{k+r-1})$. Note that, compared to a standard truncated BPTT, here we also evaluate explicitly the quality of the predictions generated from any step to any other step within the interval $[k, k+F]$, i.e., from any x_{k+r} to any $x_{k+f}, f > r$. At first sight the use of the index r may seem redundant (for example the index k_1+k_2 is covered by $k = k_1$, $r = k_2$ and by $k = k_1 + k_2$, $r = 0$). However, introducing such a redundancy is useful when stochastic gradient descent (SGD) is adopted to solve (6), as most commonly used in deep learning (Goodfellow, Bengio, & Courville, 2016). In fact, in SGD a set of random, possibly non-consecutive, values of the index k are selected at each optimization step to form the mini-batch. Hence, the time-windows between k and $k+F$ may not overlap in the mini-batch.

Note that the proposed truncated BPTT approach includes the one-step ahead approach for $F = 1$.

4.2. Network topology

In principle, each sub-network e, f, and d could be designed with different topologies and activation functions, thus allowing the user to incorporate possible prior knowledge of the process to identify. In this work we restrict our analysis to two alternative architectures: (i) a fully connected feed-forward (FF) topology for all the sub-networks e, f, and d; (ii) a quasi-LPV parameterization of the maps f and d while maintaining a general FF topology for the encoder e. The latter option allows us to learn models in the following quasi-LPV form

$$
\begin{cases}
x_{k+1} = A(x_k, u_k)[x_k' \ 1]' + B(x_k, u_k)u_k \\
y_k = C(x_k, u_k)[x_k' \ 1]'
\end{cases}
\tag{7}
$$

whose usefulness will be detailed in Section 5.2. Each coefficient of A, B and C in (7) is in turn defined as the output of a FF network.

4.3. Feature selection and model reduction

The most important hyper-parameters of our approach for achieving a good fit are n_a, n_b and n_x. As it is common in most identification techniques, tuning such parameters often requires physical insight and/or extensive trial and error. Here we propose a simple heuristics to facilitate the selection process.

It is known that the inclusion of ℓ_1-penalties in an optimization problem (a.k.a. the *shrinkage operator*) induces sparse solutions (Tibshirani, 1996). Consider the general FF topology (i). An approach to attempt reducing the number n_x of states of the model is to introduce the following variation of the so-called group LASSO operator (Scardapane, Comminiello, Hussain, & Uncini, 2017)

$$
L_{n_x}(\omega) = \chi_1 \sum_{i=1}^{n_x} i^2 \|\omega_{[i]}\|_1 \tag{8}
$$

where $\chi_1 > 0$, $\chi_1 \in \mathbb{R}$, ω is the vector of weights of the initial layer of the maps f and d, and $\omega_{[i]}$ is the subvector of ω corresponding to the state component $x_{k,i}$. After solving (5) with the additional penalty (8), all $x_{k,i}$ such that $\omega_{[i]}$ is negligible are considered as redundant, and n_x is decreased accordingly. A similar argument can be used in case the quasi-LPV topology (7) is used, by inspecting whether A, B, C depend on $x_{k,i}$ and whether the i-th column of A and C is negligible.

Regarding the encoder e, we penalize more the weights in the first layer of e associated with the components of $I_k = [y_k' \ \dots \ y_{k-n_a+1}' \ u_k' \ \dots \ u_{k-n_b+1}']'$ corresponding to less recent input/output values. To this end, let θ be the vector of weights corresponding to the first layer of e, and let $\theta_{[\ell]}$ the subvector of θ corresponding to weights associated with either $u_{k-\ell}$ or $y_{k-\ell}$, $\ell = 0, \dots, T$, $T = \max\{n_a, n_b\} - 1$. Consider the following group LASSO penalty function

$$
L_e(\theta) = \sum_{\ell=0}^{T} \chi_2 (\ell+1)^2 \|\theta_{[\ell]}\|_1 \tag{9}
$$

[1] For a given dataset $\{Q_k\}_{k=1}^{N_Q}$, $Q_k \in \mathbb{R}^{n_Q}$, and its estimate $\{\hat{Q}_k\}_{k=1}^{N_Q}$, $L_{\text{MAE}}(Q_k, \hat{Q}_k) = \frac{1}{N_Q n_Q} \|Q_k - \hat{Q}_k\|_1$.

Table 1
Hyper-parameters of the proposed learning method.

Parameter	Symbol	Meaning
Past I/O window width	n_a, n_b	Size of the information vector I_k employed to construct the state x_{k+1}
Autoencoding window	m	Time length of the output vector O_k
State dimension	n_x	Number of neurons in the "bottleneck" layer of the PPE
Multistep prediction horizon	F	Length of the prediction horizon used in truncated BPTT
Relative weights	α, β, γ	Relative weights scalarizing the multi-objective fitting criterion in (5)

where $\chi_2 > 0$, $\chi_2 \in \mathbb{R}$, is a new hyper-parameter to choose. A possible way of choosing χ_2 is to solve (5) with the additional penalty (9) for different values of χ_2 and choose the value that best trades off between quality of fit and small values of n_a, n_b. Then, after fixing the best values of n_a, n_b found, problem (5) is solved again without adding (9).

As we will describe in Section 6, we will also include small ℓ_2-regularization terms in the objective function to minimize.

5. Nonlinear state estimation and control

5.1. Filtering and state reconstruction

The encoding part e of the PPE network provides x_k as a static function of the past n_a outputs and n_b inputs, collected in vector I_{k-1}. To avoid storing I_{k-1} and to filter noise out, we consider here standard recursive filtering and state-reconstruction technique to recover x_k iteratively from input and output measurements.

A standard approach to estimate x_k is to use model-based state-estimation techniques such as extended Kalman filters (EKF). However, this would require additional tuning effort and would restrict the choice of the activation functions used in f, g to be differentiable for linearization.

An alternative method is to extend the overall learning objective (5) to also train a "neural observer". Similar to the bridge function f introduced to forward the state x_k and new input u_k to the next state x^*_{k+1}, we can introduce a similar structure to build, together with e, d, f, an additional map $s : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_x}$ from the current state estimate $\hat{x}_k$, input u_k, *and* new measured output y_k to the updated state estimate $\hat{x}_{k+1}$. This can be achieved by adding

$$\beta_4 L_4(\hat{x}_{k+1}, x_{k+1}) + \gamma_5 L_5(\hat{O}^\star_{k+1}, O_{k+1}) \tag{10a}$$

in the loss function in (5), where

$$\begin{aligned} \hat{x}_{k+1} &= s(x_k, u_k, y_k) \\ \hat{O}^\star_k &= d(\hat{x}_k),\ k = 0, \dots, N-1 \end{aligned} \tag{10b}$$

L_4, L_5 are appropriate loss functions, and $\beta_4, \gamma_5 \geq 0$.

A clear benefit of the approach in (10) is that no separate tuning process is required *after* training the process model f, g, in that the observer s is trained directly on the dataset. On the other hand, having coupled the fit of the model with the synthesis of the observer leaves no freedom to re-tune the observer without fitting again both of them. For a detailed comparison between the two approaches we refer the interested reader to Masti and Bemporad (2018).

5.2. Nonlinear model predictive control

Having identified a model in the state–space form (1) or (7), we can employ any state-feedback controller synthesis technique to control the system generating the data. Among such techniques, model predictive control (MPC) is probably the most flexible (Bemporad, 2006; Mayne, 2014) for dealing with multivariable systems under constraints on process variables. To deal with nonlinear systems, one of the most commonly used MPC schemes is the so-called linear time-varying (LTV) formulation (a.k.a. real-time iteration scheme Diehl, Bock, & Schlöder, 2005), which is based on linearizing the model around the previous optimal solution (Gros, Zanon, Quirynen, Bemporad, & Diehl, 2020).

As for EKF, this method requires the activation functions used in f, g to be differentiable, and the computation of the Jacobian matrices of the model along the prediction horizon. This can be mitigated by using the quasi-LPV affine form (7) by directly computing the matrices A, B, C for the nominal values of x_k, u_k along the prediction horizon and neglecting their sensitivities with respect to x_k, u_k. This scheme is summarized in Algorithm 1.

Algorithm 1. LTV-MPC algorithm based on model (7)

Input: Prediction horizon n_p, control horizon n_m, weight matrices $W_y \in \mathbb{R}^{n_y \times n_y}$, $W_u \in \mathbb{R}^{n_u \times n_u}$, $W_{\Delta u} \in \mathbb{R}^{n_u \times n_u}$; output and input reference signals $r_t \in \mathbb{R}^{n_y}$, $u^r_t \in \mathbb{R}^{n_u}$, $t = 0, 1, \dots$; current state estimate x_t.

1. **compute** the sequence of predicted states $\{\bar{x}_{t+1}, \dots, \bar{x}_{t+n_p}\}$ given the current guess of the input sequence $\{\bar{u}_t \cdots \bar{u}_{t+n_p-1}\}$, with $\bar{u}_{t+k} = \bar{u}_{t+n_m-1}$ for all $k \geq n_m - 1$;
2. **compute**
$$A_k = A(\bar{x}_{t+k}, \bar{u}_{t+k}), \quad B_k = B(\bar{x}_{t+k}, \bar{u}_{t+k})$$
$$C_k = C(\bar{x}_{t+k}, \bar{u}_{t+k})$$
3. **solve** the quadratic programming problem
$$\begin{aligned} \arg\min_{u_t, \dots, u_{t+n_m}} & \sum_{k=0}^{n_p-1} \|W_y(y_{t+k+1} - r_{t+k+1})\|_2^2 \\ & + \|W_u(u_{t+k} - u^r_{t+k})\|_2^2 + \|W_{\Delta u}\Delta u_{t+k}\|_2^2 \end{aligned}$$
$$\begin{aligned} \text{s.t. } & x_{j+1} = A_k[x_j'\ 1]' + B_k u_j \\ & y_{j+1} = C_k[x_{j+1}'\ 1]' \\ & \Delta u_j = u_j - u_{j-1},\ j = t+k,\ k = 0, \dots, n_p - 1 \\ & \text{linear constraints on } \Delta u_{t+k}, u_{t+k}, x_{t+k}, y_{t+k} \\ & \Delta u_{t+k} = 0,\ n_m \leq k < n_p \end{aligned}$$
to get the optimal sequence $u^\star_t, \dots, u^\star_{t+n_m-1}$;
4. **set** the current input $u_t = u^\star_t$;
5. **update** the nominal input sequence $\bar{u}_{t+k} = u^\star_{t+k}$, $1 \leq k \leq n_m - 1$, $\bar{u}_{t+k} = u^\star_{t+n_m-1}$, $n_m \leq k \leq n_p - 1$.

Output: Command input u_t, updated nominal input sequence $\{\bar{u}_{t+1}, \dots, \bar{u}_{t+n_p}\}$.

6. Experimental results

6.1. Synthetic benchmark problems

We first apply the proposed nonlinear state–space identification approach to the following synthetic benchmark problems:

the Hammerstein–Wiener system

$$\Sigma_{HW} = \begin{cases} x_{k+1} &= \begin{bmatrix} 0.7555 & 0.25 \\ -0.1991 & 0 \end{bmatrix} x_k + \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} v_k \\ v_k &= \begin{cases} \sqrt{u_k} & \text{if } u > 0 \\ u_k & \text{otherwise} \end{cases} \\ w_k &= \begin{bmatrix} 0.6993 & -0.4427 \end{bmatrix} x_k \\ y_k &= w_k + 5\sin(w_k) \end{cases} \tag{11}$$

and the discrete-time nonlinear system

$$\Sigma_T = \begin{cases} x_{k+1,1} &= x_{k,1} - k_1\sqrt{x_{k,1}} + k_2 v_k \\ x_{k+1,2} &= x_{k,2} + k_3\sqrt{x_{k,1}} - k_4\sqrt{x_{k,2}} \\ y_k &= x_{k,2} \end{cases} \tag{12}$$

that describes a tank system with possible overflows neglected (Schoukens & Noël, 2017). In (12) we set $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$, $k_4 = 0.3$ and consider two cases: one where we directly control v_k (i.e: $v_k = u_k$) that will be referred to as Σ_{T1}, and one where $v_k = \text{sign}(u_k)u_k^2$ that will be referred to as Σ_{T2}.

For both Σ_{T1} and Σ_{T2} we collect a training dataset consisting of 20,000 training samples generated by exciting the system with a sequence of step signals of length 5 steps with random amplitudes drawn from the Gaussian distribution $\mathcal{N}(1, 1)$. For Σ_{HW} we consider a training set of 10,000 samples generated using the same methodology but with amplitudes of the input signals distributed in $\mathcal{N}(0, 1)$ and length equal to 7 steps. As test cases, for all systems we consider the open simulation accuracy of the learned model when excited by 1000 test input samples consisting of random-amplitude step signals with the same characteristics of the one used for training.

All the signals are scaled by subtracting the empirical mean and dividing by the standard deviation computed on the training set. Once normalized, a zero-mean Gaussian white-noise with standard deviation $\sigma = 0.02$ is added on both the training and test signals to mimic measurement noise.

6.2. Experimental and simulation benchmarks

We further test the capabilities of the proposed approach on three publicly available datasets. The first is taken from an experimental magneto-rheological fluid damper system (The MathWorks, Inc., 2020a; Wang, Sano, Chen, & Huang, 2009), the second one from a simulated physically-accurate continuous-time two-tank system (The MathWorks, Inc., 2020b), the third one from the well known "Silverbox" system (Schoukens & Noël, 2017). The systems generating the three dataset will be refereed to as Σ_{RH}, Σ_{Tank}, and Σ_S, respectively.

The first two datasets contain 3500 and 3000 samples, respectively. In both cases, the first 2000 samples of the set are used for training/validation purposes while the remaining samples for testing. The Silverbox dataset is extracted as in Ljung, Andersson, Tiels, and Schon (2020). Again the signals in the dataset are normalized and no additional noise is superimposed.

In all cases, since during the training of the neural networks we employ an early-stopping criterion based on validation data, 5% of the training dataset is reserved for validation.

6.3. Hyperparameter selection and implementation details

For both the FF network topology (1) and the affine quasi-LPV one (7), described in Section 4.2 as type (*i*) and type (*ii*), respectively, the hidden part of all the involved networks consist of 3 layers of 30 neurons each with ReLU (Hahnloser, Sarpeshkar, Mahowald, Douglas, & Seung, 2000) activation functions, followed by a final linear output layer.

The network has been implemented in Keras (Chollet et al., 2015) using Tensorflow (Abadi, Agarwal, Barham, Brevdo, Chen, Citro, et al., 2015) as back-end, and trained via the AMSgrad algorithm (Reddi, Kale, & Kumar, 2019). The training procedure is carried out in two stages: first the ANNs are trained with $\alpha = 10, \beta = 0.3, \gamma = 0$ until convergence, and then trained again (using as a starting point the weights obtained in the previous phase) using $\alpha = 0, \beta = 10, \gamma = 1$. Unless otherwise noted, ℓ_2-regularization terms with penalty $\lambda = 0.0001$ are added on all but bias terms coefficients of the nonlinear neurons to smooth the fitting process out. Moreover, the same ℓ_2-penalty is added in the final linear layer of the neural network modeling the last column of matrix $A(x_k, u_k)$ in case the quasi-LPV topology (7) is used. Under the same topology, we also zero the coefficients of the last column of matrix $C(x_k, u_k)$.

The above two-step procedure is used to ease the learning process. As the focus of the first step is mostly on the L_1 objective, the learning of e and d is emphasized ($\alpha = 10$), with a small contribution ($\beta = 0.3$) of L_2 maintained to avoid that the resulting state vector cannot be properly updated over time. Once a good estimate of the decoder/encoder pair has been determined in the first stage, the second stage takes this as the initial guess to completely focus ($\alpha = 0$) on enforcing that the model propagates correctly over time ($\beta = 10, \gamma = 1$).

Note that in our experiments we have not made any attempt to optimize the selected values of α, β, γ. When best identification performance is sought, global optimization algorithms based on surrogate functions (Bemporad, 2020; Snoek, Larochelle, & Adams, 2012) could be used for optimal tuning of α, β, γ, and possibly other hyper-parameters.

Performance results are measured in terms of the best fit ratio (BFR)

$$\text{BFR} = \max\left\{0, 1 - \frac{\|y_t - \hat{y}_t\|_2}{\|y_t - \bar{y}\|_2}\right\} \tag{13}$$

where $\bar{y}$ is the average of the output signal y over all the samples and $\hat{y}_k$ is the *open-loop prediction* extracted from $\hat{O}_{k+1} = d(\hat{x}_{k+1})$ with

$$\begin{aligned} \hat{x}_{k+1} &= f(\hat{x}_k, u_k), \;\; k = k_0, \ldots, N-1 \\ \hat{x}_{k_0} &= e(I_{k_0-1}) \end{aligned} \tag{14}$$

We set $m = 1$ in all tests as we are only interested in predicting the next output y_{k+1} and filter the current y_k. As we plan to never exceed the value 15 for n_a, n_b, all simulations start from $k_0 = 15$ in order to have enough samples to form I_{k_0-1}. Unless otherwise noted, we report the best and average BFR and its standard deviation obtained on 10 different runs with different random seeds.

6.4. Fit results

We evaluate the BFR open-loop simulation performance (13) when the fitting process is either based one-step ahead criterion ($F = 1$) or truncated BPTT ($F = 4$), for both the quasi-LPV and FF topologies. In all cases we set $n_x = 6$, $n_a = n_b = 10$.

Results are reported in Table 2 for the synthetic benchmarks and in Table 3 for the experimental/simulation benchmarks. The results highlight that the quasi-LPV parameterization (7) well reproduces the behavior of the system in all the proposed benchmarks, and slightly outperforms the FF architecture. This is especially evident for Σ_{T2}. The results also confirm the beneficial effect of truncated BPTT.

For comparison, for each problem we train a nonlinear ARX models of the same order, equipped with a dense feedforward ANN regressor with 3 layers of 30 ReLU neurons each, trained using the System Identification Toolbox for MATLAB (The MathWorks, Inc., 2019). We consistently achieve roughly the following BFRs: 0.97 (Σ_{T1}), 0.94 (Σ_{T2}), 0.97 (Σ_{HW}), 0.94 (Σ_{Tank}), 0.55 (Σ_{RH}), and 0.94 (Σ_S).

Table 2
Performance results (BFR) on synthetic benchmarks for different combinations of topologies and training procedures: A = FF topology, B = quasi-LPV topology; 1 = training based on one-step ahead loss ($F = 1$), F = training based on multi-step ahead loss ($F = 4$).

System		A/1	B/1	A/F	B/F
Σ_{HW}	Average	0.983	0.989	0.989	0.991
	Standard deviation	0.002	0.001	0.001	0.001
	Best	0.986	0.991	0.990	0.992
Σ_{T2}	Average	0.873	0.913	0.943	0.961
	Standard deviation	0.024	0.019	0.013	0.015
	Best	0.903	0.938	0.963	0.977
Σ_{T1}	Average	0.944	0.969	0.973	0.980
	Standard deviation	0.020	0.003	0.003	0.004
	Best	0.960	0.976	0.977	0.985

Table 3
Performance results (BFR) on experimental and simulation benchmarks for different combinations of topologies and training procedures: A = FF topology, B = quasi-LPV topology; 1 = training based on one-step ahead loss ($F = 1$), F = training based on multi-step ahead loss ($F = 4$).

System		A/1	B/1	A/F	B/F
Σ_{RH}	Average	0.815	0.815	0.818	0.882
	Standard deviation	0.040	0.044	0.015	0.025
	Best	0.859	0.870	0.839	0.908
Σ_{Tank}	Average	0.896	0.904	0.916	0.915
	Standard deviation	0.008	0.020	0.007	0.009
	Best	0.911	0.923	0.932	0.927
Σ_{S}	Average	0.905	0.951	0.966	0.986
	Standard deviation	0.015	0.013	0.004	0.003
	Best	0.931	0.963	0.974	0.991

6.5. Feature selection and model reduction

We test the model-selection technique presented in Section 4.3 to reduce n_a, n_b and n_x. Starting with $n_x = 6$ and $n_a = n_b = 10$, we attempt reducing n_a, n_b by imposing the group LASSO penalty (9) with $\chi_2 = 0.0003$, while maintaining the ℓ_2-regularization penalty $\lambda = 0.0001$ on the remaining regularized coefficients. Similarly, starting with the same values of n_x, n_a, n_b, we impose the group LASSO penalty (8) to attempt reducing n_x with $\chi_1 = 0.0003$. Numerical tests are performed on Σ_{T2}, with training carried out using truncated BPTT with $F = 4$.

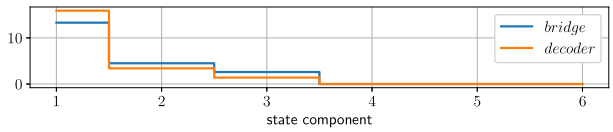
The number of coefficients whose absolute value is larger than 0.001 (which we selected as the threshold used to consider the coefficient as negligible) is reported in Fig. 2. In particular, Fig. 2(a) clearly shows that only 3 state components are commonly used, so that n_x can be reduced to $n_x = 3$. Similarly, from Fig. 2(b) we can note that only $y_k, \ldots, y_{k-2}$ and $u_k, \ldots, u_{k-4}$ contribute significantly to the encoder function e, so that we can decide to limit I_k to them. This suggests setting $n_a = n_b$ to either 3, 4, or 5.

This is confirmed by the results reported in Table 4, which shows the fit performance occurring when the models are trained with $n_x = 2$, $n_a = n_b = 5$, with $n_x = 3$, $n_a = n_b = 5$, and with $n_x = n_a = n_b = 5$, without including ℓ_1-penalties for sparsification. The difference in performance between using $n_x = 3$, $n_a = n_b = 5$, and both using $n_x = n_a = n_b = 5$ and the original values $n_x = 6$, $n_a = n_b = 10$, in the original feedforward topology is negligible. Very good performance is also obtained by $n_x = 2$, $n_a = n_b = 5$, confirming the marginal relevance of introducing a third state shown in Fig. 2(a).

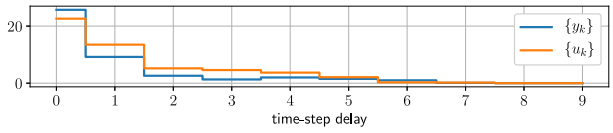
6.6. LTV-MPC based on learned model

We evaluate the performance of the MPC controller presented in Section 5.2 in controlling Σ_{T2}. The MPC controller is based on the quasi-LPV model obtained by training from the same dataset with $n_x = 6$ and $n_a = n_b = 10$, and by setting $n_p = 5$, $W_y = 1$, $W_u = 0.001$, $W_{\Delta u} = 0.01$, under the box constraints $|u_k| \leq 0.8$.

Fig. 3 shows the obtained closed-loop performance when tracking a shifted sine-sweep reference signal. To close the MPC loop, in this example we used the encoder function to estimate the state x_k from past input/output samples contained in I_{k-1}, although other observers (like a time-varying Kalman filter based on the same quasi-LPV model) could be used as well.



(a) Average number of non-negligible neurons in f and d as a function of the state component $x_{k,i}$, $i = 1, \ldots, n_x$



(b) Average number of non-negligible neurons in function e acting on I_k, with respect to the time-step delay i of either the input sample u_{k-i} or the output sample y_{k-i}, $i = 0, \ldots, n_a - 1 = n_b - 1$, the neuron is acting on.

Fig. 2. Number of active neurons in e, f, d.

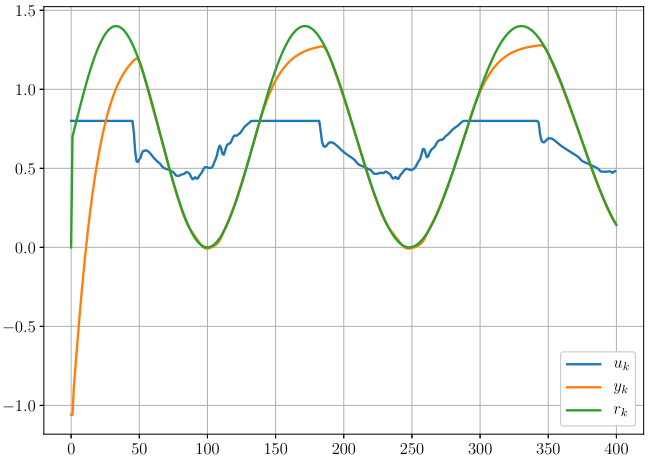


Fig. 3. Tracking performance of LTV-MPC (Algorithm 1) applied to control system Σ_{T2} (quantities are in normalized units).

Table 4
Performance (BFR) of the reduced-order models in learning Σ_{T2}. FF-5 = feedforward topology with $n_x = n_a = n_b = 5$. FF-3/5 = feedforward topology with $n_x = 3$, $n_a = n_b = 5$. FF-2/5 = feedforward topology with $n_x = 2$, $n_a = n_b = 5$. FF-6 = feedforward topology with $n_x = 6$, $n_a = n_b = 10$ (reported for comparison).

System		FF-2/5	FF-3/5	FF-5	FF-6
Σ_{T2}	Average	0.905	0.903	0.925	0.943
	Std. deviation	0.059	0.061	0.026	0.013
	Best	0.948	0.952	0.964	0.963

6.7. Computational aspects

The overall CPU time to simulate the closed-loop LTV-MPC system, running the decoder, and computing the control action

via the general purpose L-BFGS-B solver (Oliphant, 2007) over 400 sampling steps is ≈ 5 seconds on a laptop equipped with an Intel Core i5 6200u CPU and 16 GB of RAM. Note that we did not make any attempt towards efficiency of implementation, for example a more efficient solver like the one proposed in Saraf and Bemporad (2020) could be used to compute the MPC action.

Regarding the CPU time spent for training the models, the proposed procedure is carried out on average in $\approx$ 10 minutes on the aforementioned machine for dataset consisting of 20,000 samples. Interestingly, the computation effort is not very sensitive to the particular choice of n_x, n_a, and n_b.

Models with $n_x = 6$, $n_a = n_b = 10$ require $\approx$ 10,000 single precision coefficients. Clearly, the number of model coefficients heavily depends on the chosen topology, so that it can be drastically reduced if memory footprint and/or throughput are of concern. Moreover, we mention that one could apply recent results on model reduction for ANNs, see, e.g., Han, Mao, and Dally (2015) and references therein. In general, one must find the best tradeoff between the number of optimized model coefficients and the obtained closed-loop tracking performance when training a nonlinear state–space model for control purposes.

7. Conclusions

In this paper we have proposed a viable approach to learn nonlinear state–space models from input/output data for model-based control systems design. The method can be applied either to identify a nonlinear model from experimental data, or from a high-fidelity simulator, or to reduce the order of an existing nonlinear model. Indeed, the approach allows the direct control of the number of states, which in turn dictates the complexity of model-based nonlinear estimators and MPC controllers.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. arXiv preprint arXiv:1603.04467, Software available from http://www.tensorflow.org.

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, *2*(1), 53–58.

Baram, Y. (1984). Minimal order representation, estimation and feedback of continuous-time stochastic linear systems. In *Mathematical theory of networks and systems* (pp. 24–41). Springer.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, *39*(3), 930–945.

Beintema, G. I., Toth, R., & Schoukens, M. (2020). Non-linear state-space model identification from video data using deep encoders. arXiv preprint arXiv:2012.07721.

Beintema, G., Toth, R., & Schoukens, M. (2020). Nonlinear state-space identification using deep encoder networks. arXiv preprint arXiv:2012.07697.

Bemporad, A. (2006). Model-based predictive control design: New trends and tools. In *Proceedings of 45th IEEE conference on decision and control* (pp. 6678–6683). San Diego, CA.

Bemporad, A. (2020). Global optimization via inverse distance weighting and radial basis functions. *Computational Optimization and Applications*, *77*, 571–595, Code available at http://cse.lab.imtlucca.it/~bemporad/glis.

Breschi, V., Piga, D., & Bemporad, A. (2016). Piecewise affine regression via recursive multiple least squares and multicategory discrimination. *Automatica*, *73*, 155–162.

Champion, K., Lusch, B., Kutz, J. N., & Brunton, S. L. (2019). Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, *116*(45), 22445–22451.

Chollet, F., et al. (2015). *Keras*. GitHub, https://github.com/keras-team/keras.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., & Bengio, Y. (2015). A recurrent latent variable model for sequential data. In *Advances in neural information processing systems* (pp. 2980–2988).

Diehl, M., Bock, H. G., & Schlöder, J. P. (2005). A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, *43*(5), 1714–1736.

Drgona, J., Tuor, A. R., Chandan, V., & Vrabie, D. L. (2020). Physics-constrained deep learning of multi-zone building thermal dynamics. arXiv preprint arXiv:2011.05987.

Forgione, M., & Piga, D. (2020). Model structures and fitting criteria for system identification with neural networks. In *Proceedings of 2020 IEEE 14th international conference on application of information and communication technologies* (pp. 1–6).

Forgione, M., & Piga, D. (2021). Continuous-time system identification with neural networks: Model structures and fitting criteria. *European Journal of Control*.

Fraccaro, M., Kamronn, S., Paquet, U., & Winther, O. (2017). A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in neural information processing systems* (pp. 3601–3610).

Gedon, D., Wahlström, N., Schön, T. B., & Ljung, L. (2020). Deep state space models for nonlinear system identification. arXiv preprint arXiv:2003.14162.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Gros, S., Zanon, M., Quirynen, R., Bemporad, A., & Diehl, M. (2020). From linear to nonlinear MPC: Bridging the gap via the real-time iteration. *International Journal of Control*, *93*(1), 62–80.

Guyon, I., & Elisseeff, A. (2006). An introduction to feature extraction. In *Feature extraction: Foundations and applications* (pp. 1–25). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, S. H. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*(6789), 947.

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Kandukuri, R. K., Achterhold, J., Möller, M., & Stückler, J. (2020). Learning to identify physical parameters from video using differentiable physics. arXiv preprint arXiv:2009.08292.

Karl, M., Soelch, M., Bayer, J., & Van der Smagt, P. (2016). Deep variational bayes filters: Unsupervised learning of state space models from raw data. arXiv preprint arXiv:1605.06432.

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. In *An introduction to variational autoencoders*. Now Foundations and Trends.

Krishnan, R. G., Shalit, U., & Sontag, D. (2015). Deep Kalman filters. arXiv preprint arXiv:1511.05121.

Li, Q., Dietrich, F., Bollt, E. M., & Kevrekidis, I. G. (2017). Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator. *Chaos. An Interdisciplinary Journal of Nonlinear Science*, *27*(10), Article 103111.

Ljung, L., Andersson, C., Tiels, K., & Schon, T. B. (2020). Deep learning and system identification. In *Proceedings of the 21st IFAC world congress*.

Lu, Q., & Zavala, V. M. (2020). Image-based model predictive control via dynamic mode decomposition. arXiv preprint arXiv:2006.06727.

Lusch, B., Kutz, J. N., & Brunton, S. L. (2018). Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, *9*(1), 1–10.

Masti, D., & Bemporad, A. (2018). Learning nonlinear state-space models using deep autoencoders. In *Proceedings of 57th conference on decision and control* (pp. 3862–3867).

Mayne, D. Q. (2014). Model predictive control: Recent developments and future promise. *Automatica*, *50*(12), 2967–2986.

Okada, M., & Taniguchi, T. (2020). Dreaming: Model-based reinforcement learning by latent imagination without reconstruction. arXiv preprint arXiv:2007.14535.

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, *9*(3), 10–20.

Otto, S. E., & Rowley, C. W. (2019). Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, *18*(1), 558–593.

Pillonetto, G., Dinuzzo, F., Chen, T., Nicolao, G. D., & Ljung, L. (2014). Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*, *50*(3), 657–682.

Ping, Z., Yin, Z., Li, X., Liu, Y., & Yang, T. (2020). Deep Koopman model predictive control for enhancing transient stability in grids. *International Journal of Robust and Nonlinear Control*.

Puskorius, G. V., & Feldkamp, L. A. (1994). Truncated backpropagation through time and Kalman filter training for neurocontrol. In *Proceedings of 1994 IEEE international conference on neural networks (vol. 4)* (pp. 2488–2493). IEEE.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., & Januschowski, T. (2018). Deep state space models for time series forecasting. In *Advances in neural information processing systems* (pp. 7785–7794).

Reddi, S. J., Kale, S., & Kumar, S. (2019). On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237.

Saraf, N., & Bemporad, A. (2020). A bounded-variable least-squares solver based on stable QR updates. *IEEE Transactions on Automatic Control*, *65*(3), 1242–1247.

Scardapane, S., Comminiello, D., Hussain, A., & Uncini, A. (2017). Group sparse regularization for deep neural networks. *Neurocomputing*, *241*, 81–89.

Schoukens, J., & Ljung, L. (2019). Nonlinear system identification: A user-oriented road map. *IEEE Control Systems Magazine*, *39*(6), 28–99.

Schoukens, M., & Noël, J.-P. (2017). Three benchmarks addressing open challenges in nonlinear system identification. In *Proceedings of 20th world congress of the international federation of automatic control* (pp. 448–453). Toulouse, France.

Simpson, T., Dervilis, N., & Chatzi, E. (2020). On the use of nonlinear normal modes for nonlinear reduced order modelling. arXiv preprint arXiv:2007.00466.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems (vol. 25)* (pp. 2951–2959). Curran Associates, Inc.

Takeishi, N., Kawahara, Y., & Yairi, T. (2017). Learning Koopman invariant subspaces for dynamic mode decomposition. In *Advances in neural information processing systems* (pp. 1130–1140).

The MathWorks, Inc. (2019). System identification toolbox. Natick, Massachusetts, United State.

The MathWorks, Inc. (2020a). Nonlinear modeling of a magneto-rheological fluid damper. https://mathworks.com/help/ident/ug/nonlinear-modeling-of-a-magneto-rheological-fluid-damper.html.

The MathWorks, Inc. (2020b). Two tank system: C MEX-file modeling of time-continuous SISO system. https://it.mathworks.com/help/ident/ug/two-tank-system-c-mex-file-modeling-of-time-continuous-siso-system.html.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *58*(1), 267–288.

van Hoof, H., Chen, N., Karl, M., van der Smagt, P., & Peters, J. (2016). Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ international conference on intelligent robots and systems* (pp. 3928–3934).

Wang, Y. (2017). A new concept using LSTM Neural Networks for dynamic system identification. In *Proceedings of 2017 American control conference* (pp. 5324–5329).

Wang, J., Sano, A., Chen, T., & Huang, B. (2009). Identification of Hammerstein systems without explicit parameterisation of non-linearity. *International Journal of Control*, *82*(5), 937–952.

Watter, M., Springenberg, J., Boedecker, J., & Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems* (pp. 2746–2754).

Wehmeyer, C., & Noé, F. (2018). Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *Journal of Chemical Physics*, *148*(24), Article 241703.

Werbos, P. J., et al. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Williams, R. L., & Lawrence, D. A. (2007). Minimal realizations. In *Linear state-space control systems* (pp. 185–197). John Wiley Sons, Inc.

Xiao, Y., Zhang, X., Xu, X., Liu, X., & Liu, J. (2020). A deep learning framework based on Koopman operator for data-driven modeling of vehicle dynamics. arXiv preprint arXiv:2007.02219.

Daniele Masti received his Bachelor's degree in Computer and Information Engineering from the University of Siena, Italy, in 2015, and his Master's degree in Electric and Automation Engineering in 2018 from the University of Florence, Italy.

He is currently a Ph.D. student in Computer Science and System Engineering at IMT School for Advanced Studies Lucca, Italy. His research interests include system identification, data-driven control, and machine learning.

Alberto Bemporad received his Master's degree in Electrical Engineering in 1993 and his Ph.D. in Control Engineering in 1997 from the University of Florence, Italy. In 1996/97 he was with the Center for Robotics and Automation, Department of Systems Science & Mathematics, Washington University, St. Louis. In 1997–1999 he held a postdoctoral position at the Automatic Control Laboratory, ETH Zurich, Switzerland, where he collaborated as a senior researcher until 2002. In 1999–2009 he was with the Department of Information Engineering of the University of Siena, Italy, becoming an Associate Professor in 2005. In 2010–2011 he was with the Department of Mechanical and Structural Engineering of the University of Trento, Italy. Since 2011 he is Full Professor at the IMT School for Advanced Studies Lucca, Italy, where he served as the Director of the institute in 2012–2015. He spent visiting periods at Stanford University, University of Michigan, and Zhejiang University. In 2011 he co-founded ODYS S.r.l., a company specialized in developing model predictive control systems for industrial production. He has published more than 350 papers in the areas of model predictive control, hybrid systems, optimization, automotive control, and is the co-inventor of 16 patents.

He is author or coauthor of various software packages for model predictive control design and implementation, including the Model Predictive Control Toolbox (The Mathworks, Inc.) and the Hybrid Toolbox for MATLAB. He was an Associate Editor of the IEEE Transactions on Automatic Control during 2001–2004 and Chair of the Technical Committee on Hybrid Systems of the IEEE Control Systems Society in 2002–2010. He received the IFAC High-Impact Paper Award for the 2011–14 triennial and the IEEE CSS Transition to Practice Award in 2019. He is an IEEE Fellow since 2010.