2018 IEEE Conference on Decision and Control (CDC)
Miami Beach, FL, USA, Dec. 17-19, 2018

Learning nonlinear state-space models using deep autoencoders

Daniele Masti*†, Alberto Bemporad†

***Abstract*— We introduce a new methodology for the identification of nonlinear state-space models using machine-learning techniques based on deep autoencoders for dimensionality reduction and neural networks. By learning a direct acyclic computational graph, our framework simultaneously identifies the nonlinear output and state-update maps, and optionally a neural state observer. After formulating the approach in detail and providing guidelines for tuning the related hyperparameters and reducing the model order, we show its capability of fitting a nonlinear model from an input/output dataset generated by a benchmark nonlinear system. Performance is assessed in terms of the ability of filtering and predicting output signals ahead, and of controlling the system via nonlinear model predictive control (MPC) based on the identified model.**

I. Introduction

Nonlinear systems identification has gained increasing popularity in the last years [1], also due to recent advances in machine-learning methods for nonlinear function regression. These have been employed with high success either as an extension of classical techniques, such as neural autoregressive models with exogenous inputs (ARX) and reproducing kernel Hilbert space (RKHS) models [1], or as novel approaches, such as long short-term memory (LSTM) neural networks [2] and piecewise-affine regression [3].

Most of the above techniques, however, are based on input-output representations and thus do not involve an explicit definition of a Markovian state. On the other hand, state-space models are the basis for most modern control design techniques, such as model predictive control (MPC), sliding mode control, as well as for noise filtering and smoothing, such as extended Kalman filtering (EKF).

In this paper we introduce a new methodology based on autoencoders (AEs) [4] that is able to learn a nonlinear model in state-space representation from a given input/output dataset. Autoencoders are a particular type of artificial neural networks (ANNs) that can be successfully employed for various tasks, such as dimensionality reduction and denoising [5]. To the best of our knowledge, the use of AEs in systems identification has not been explored in the literature, except for the very recent contribution [6], which we became aware of after preparing this manuscript, that proposes an approach to learn autoregressive models of autonomous systems.

In our approach, the order of the model is a design parameter that can be set arbitrarily, trading off modeling accuracy versus model complexity. Therefore, our method can be also employed to reduce the order of a given nonlinear model, such as a model of a distributed parameter system.

* University of Florence, Department of Information Engineering, Italy. daniele.masti@stud.unifi.it
† IMT School for Advanced Studies Lucca, Italy. alberto.bemporad@imtlucca.it

II. Nonlinear identification problem

We are given a training dataset of input/output samples $Z = \{u_1, y_1, \dots, u_N, y_N\}$ 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. For a given desired number of past outputs $n_a \geq 1$, of past inputs $n_b \geq 1$, and a chosen state dimension $n_x \geq 1$, our goal is to find maps e, f, g, with $e : \mathbb{R}^{n_I} \to \mathbb{R}^{n_x}$, $n_I \triangleq n_a n_y + n_b n_y$, $f : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_x}$, $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, optimizing the fitting criterion

$$\min_{e,f,g} \mathcal{L}(e, f, g, Z) \tag{1a}$$

where

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

$L : \mathbb{R}^{2n_y} \to \mathbb{R}$ is a given loss function, with $L(0, 0) = 0$, $x_k \in \mathbb{R}^{n_x}$ is the state vector, I_k is the following information vector

$$I_k = [y_k' \ \dots \ y_{k-n_a+1}' \ u_k' \ \dots \ u_{k-n_b+1}']' \tag{2}$$

and $k_0 \triangleq \max\{n_a, n_b\}$. In (1) f is the nonlinear state-update map, g the nonlinear output map[1], and e is a state-space coordinate transformation function mapping the information vector I_k of past inputs and outputs into the current state x_k. We will assess the quality of the model on a new *validation* dataset $\tilde{Z} = \{\tilde{u}_1, \tilde{y}_1, \dots, \tilde{u}_M, \tilde{y}_M\}$.

In general problem (1) has infinitely many solutions. 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 $\tilde{y}_k$ is of main interest and has been widely explored in the literature [7], [8].

A similar problem has also been widely studied in machine learning, where discovering a "compressed" description of a given information vector has been extensively studied in the context of *feature extraction* [9]. 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. How to adapt such

[1] We restrict our analysis to strictly causal discrete-time nonlinear systems, as these are most often encountered when modeling physical systems.

techniques to the posed problem of identification of nonlinear state-space models will be discussed in the next sections.

To summarize, solving problem (1) amounts to determining a suitable state-space dimension n_x, the relation e between x_k and past input/output pairs, and the state-update and output maps f, g. For tackling both tasks, we will employ ANNs due to their universal approximation properties [10], [11] and efficient numerical packages available for training them such as Tensorflow [12].

III. State selection via autoencoders

The idea behind an autoencoder is simply to train an ANN to reproduce the unit mapping from a certain information vector $I_k \in \mathbb{R}^{n_I}$ to I_k itself, under the topological constraint that one of the 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, the corresponding value $z_k \in \mathbb{R}^{n_x}$, taken by the neurons of the bottleneck layer, represents the desired lower-dimensional vector concentrating the information contained in I_k.

A. Partial predictive autoencoders

Given the dataset consisting of input/output samples Z, applying a standard autoencoder to compress the information vector I_k defined by (2) into a reduced-order vector $z_k \in \mathbb{R}^{n_x}$ would not be optimal 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}, $i = 1, \dots, n_b$. 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

$$O_k = [y_k' \ \dots \ y_{k-m}']' \tag{3}$$

with $0 \leq m \leq \max\{n_a, n_b\} - 1$. By fitting an ANN with a hidden layer of size n_x, $n_x \leq n_a n_y + n_b n_u$, 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 state x_k of the model captures the information required to predict y_k from I_{k-1}, filter y_{k-1} (if $m \geq 1$), and smooth past outputs (if $m > 1$). Note that we could make x_k depend on y_k too, although our numerical experiments have shown that this leads to worse fitting results.

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 $d : \mathbb{R}^{n_x} \to \mathbb{R}^{m n_y}$ from x_k to O_k, whose first n_y components constitute the desired output function $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$.

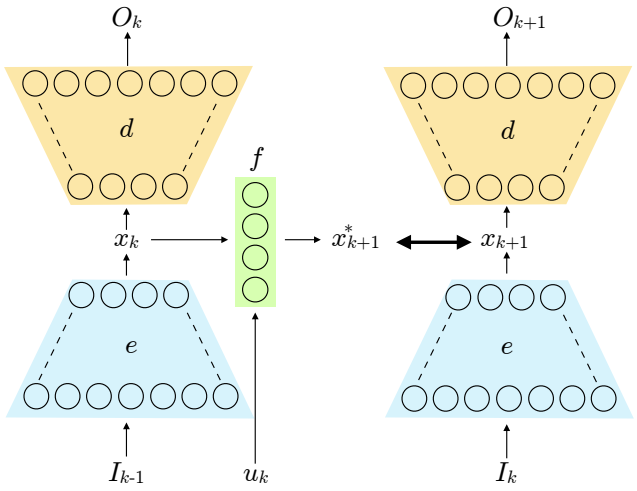

Fig. 1. Schematic representation of the computational graph of the proposed nonlinear model structure

IV. Model learning

Having defined a structure to map I_{k-1} into x_k, we also need a structure to fit a function $f : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_x}$ 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 $f : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_x}$ mapping (x_k, u_k) to x_{k+1}. Our numerical experience is that this approach leads to poor models.

We propose instead a better method that learns e, d and f simultaneously. By exploiting the fact that neural networks are direct acyclic graphs, 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 Figure 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 minimization problem

$$\begin{aligned} \min_{f,d,e} \quad & \sum_{k=k_0}^{N-1} L_1(\hat{O}_k, O_k) + L_1(\hat{O}_{k+1}, O_{k+1}) \\ & + \beta L_2(x^\star_{k+1}, x_{k+1}) + \gamma L_3(O_{k+1}, O^\star_{k+1}) \\ \text{s.t.} \quad & x_k = e(I_{k-1}), \ k = k_0, \dots, N \\ & x^\star_{k+1} = f(x_k, u_k), \ k = k_0, \dots, N-1 \\ & \hat{O}_k = d(x_k), \ k = k_0, \dots, N \\ & O^\star_k = d(x^\star_k), \ k = k_0+1, \dots, N \end{aligned} \tag{4}$$

where L_i are loss functions, $\beta, \gamma > 0$ are scalar weights, O_k is defined by (3), and I_k by (2).

A. Choice of the loss functions

Clearly (4) may not solve our original objective (1) exactly, although it may provide a good suboptimal solution to it. In particular, the loss function L_1 must be related with the original loss L in (1b), such as $L_1(\hat{O}_k, O_k) = \sum_{j=k-m}^{k} L(\hat{y}_j, y_j)$. The loss L_2 can be seen as a relaxation of the state update equation $x_{k+1} = f(x_k, u_k)$ in (1b). 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 deviations between the predicted outputs $O^\star_{k+1}$ and the measured outputs O_{k+1}. Minimizing L_3 is the objective with the most priority, as it captures the one-step ahead properties of the model. We set

$$L_3(O_{k+1}, O^\star_{k+1}) = \|O_{k+1} - O^\star_{k+1}\|_2^2 \tag{5a}$$

The "horizontal" objective related to L_2 ensures the consistency of the two copies of PPE learned. We set

$$L_2(x^\star_{k+1}, x_{k+1}) = \|x_{k+1} - x^\star_{k+1}\|_2^2 \tag{5b}$$

The "vertical" objectives related to L_1 is only ancillary to guide the process of learning the correct bridge/decoder pair. For this reason, we choose L_1 as the margin-loss function commonly used in SVM regression problems with soft-margins [13]

$$L_1(O_k, \hat{O}_k) = \sum_{i=1}^{(m+1)n_y} \max\{\varepsilon, \zeta|O^i_k|, |O^i_k - \hat{O}^i_k|\}^2 \tag{5c}$$

where ε, ζ are nonnegative hyper-parameters determining the component-dependent soft threshold $\max\{\varepsilon, \zeta|O^i_k|\}$, with the superscript i denoting the ith component of the corresponding vector.

B. Tuning parameters

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 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 I.

In particular, while in standard AEs the dimension n_x is related to keeping the variance of the error between the value of the input and replicated input layers limited, here it is enough to choose n_x so that the ANN is able to predict just O_k satisfactorily. Clearly, as introduced earlier, as we assume that x_k is a compressed version of the information contained in I_k, we require $n_x \leq n_a n_y + n_b n_u$. Moreover, we restrict $m \leq \max\{n_a, n_b\} - 1$ as we assume that older outputs cannot be inferred from I_k.

V. Nonlinear state estimation and control

A. Filtering and state reconstruction

The encoding part e of the PPE network can be interpreted as a deadbeat observer, in that it provides x_k as a function of the last n_a outputs and n_b inputs. However, the unavoidable presence of measurement noise and modeling errors often asks for a state-observer to reconstruct x_k recursively, which also reduces the storage requirements from the past n_a outputs and n_b inputs to the last n_x components of the estimated state.

We propose here two alternative state-reconstruction methods: (i) a standard extended Kalman filter (EKF) technique [14] based on the learned model f, g and suitable covariance matrices, and (ii) a state observer based on a neural network, denoted in the sequel as "neural observer". Employing an EKF poses a limit in the choice of activation functions, due to the fact that the ANN needs to be differentiated with respect to x_k and u_k in order to get the required Jacobian matrices.

While often the issue can be circumvented, in some cases one knows in advance that the data-generating process has switching properties or other forms of nonsmoothness, that would be highly desirable to capture in our learned model for maximum fit performance.

Therefore, an alternative approach, similar to what employed in Section IV to learn the state-update function f, is to extend the overall learning objective (4) to also train a neural observer. Similar to the bridge function introduced to forward the state x_k and new input u_k to estimate the next state x^*_{k+1}, we can introduce a similar structure to build, together with e, d, f, an additional map $s : \mathbb{R}^{n_x+n_u+n_y} \to \mathbb{R}^{n_x}$ from the current state estimate $\hat{x}_k$, the input u_k, *and* the new measured output y_k to the updated state estimate $\hat{x}_{k+1}$. This is achieved by replacing $L_3(O^\star_{k+1}, O_{k+1})$ in problem (4) with

$$\beta L_4(\hat{x}_{k+1}, x_{k+1}) + \gamma L_3(\hat{O}^\star_{k+1}, \hat{O}_{k+1}) \tag{6}$$

where

$$\begin{aligned} \hat{x}_{k+1} &= s(x_k, u_k, y_k) \\ \hat{O}^\star_k &= d(\hat{x}_k),\ k = 0, \ldots, N-1 \end{aligned} \tag{7}$$

and

$$L_4(\hat{x}_{k+1}, x_{k+1}) = \|\hat{x}_{k+1} - x_{k+1}\|_2^2 \tag{8}$$

A benefit of this approach 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 data set. On the other hand, having coupled the fit of model and observer leaves no freedom in retuning the observer without fitting again both; moreover, it may lead to sacrificing the quality of prediction/smoothing of the output in favor of better reconstructability of the state.

B. Nonlinear model predictive control

Having a state-space model and a state observer learned from the available dataset, any model-based state-feedback control technique can be employed to design a control system. Among these, Model Predictive Control (MPC) is probably the most flexible [15], [16] for dealing with multivariable systems and constraints on process variables, and relatively easy to tune. Here we focus on a linear time-varying (LTV) MPC scheme that exploits the Jacobian matrices already used by the EKF.

The standard Algorithm 1 summarizes the LTV-MPC control algorithm based on the learned model f, g and EKF (see, e.g., [17]).

parameter	symbol	meaning
input window size	n_a, n_b	size of information vector I_k employed to construct the state x_{k+1}
autoencoding window	m	size of output vector O_k, $m \leq \max\{n_a, n_b\} - 1$. Increasing m usually provides a regularization effect
state dimension	n_x	number of components of the state x_k, corresponding to the number of neurons in the central layer of the hourglass ANN defining the PPE, $n_x \leq n_a n_y + n_b n_u$
soft threshold parameters	ζ, ε	parameters defining the loss function L_1, $\zeta, \varepsilon \geq 0$
relative weights	β, γ	relative weights scalarizing the multi-objective fitting criterion in (4)

TABLE I

PARAMETERS OF THE PROPOSED LEARNING METHOD TO BE TUNED

VI. EXAMPLE: NONLINEAR TANK SYSTEM

A. Benchmark problem

We generate data from the following process

$$\begin{cases} x_1(k+1) = x_1(k) - k_1\sqrt{x_1(k)} + k_2(u(k) + w(k)) \\ x_2(k+1) = x_2(k) + k_3\sqrt{x_1(k)} - k_4\sqrt{x_2(k)} \\ y(k) = x_2(k) + v(k) \end{cases} \tag{9}$$

which is a discrete-time approximation of the well-known nonlinear tank system [18] with possible overflows neglected. In (9) $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$, $k_4 = 0.3$, and $v(k)$, $w(k)$ are zero-mean Gaussian white noise signals with standard deviation $\sigma_v = 0.15$ and $\sigma_w = 0.35$, respectively. The training dataset consists of $N = 80000$ samples collected by exciting (9) with a sequence of step signals of length 7 steps each of random amplitude, normally distributed with zero mean and standard deviation of 5. Since we employ an early-stopping strategy, 10% of the training dataset is used to check the stopping criterion.

The test set used for cross-validation consists of 1000 samples and is instead collected by supplying the harmonic input $u(k) = 2.5\sin(k/7) + 1.5\cos(k/3) + 1$ and noise generated with the same distributions used when collecting the training dataset. The acquired signals are rescaled by a multiplicative factor to have similar ranges, although bias is not removed as motivated in [19].

B. Choice of topology and tuning of hyper-parameters

In principle each one the four modules (encoder e, decoder d, bridge f, and observer s) can be designed with different ANN topologies and activation functions, which provides an extreme flexibility of the approach. We restrict here our analysis to ANNs with 3 hidden layers for all modules, each one containing 60 exponential linear unit (ELU) neurons [20], encapsulated between a linear input and a linear output layer with no bias enabled, for a total of ≈ 45000 weights equally divided among the four modules. Such a choice, although not the optimal one, allows us to perform all the tests using the same network architecture, and in any case it is a good baseline for further fine tuning.

The network has been implemented in Keras [21] and Tensorflow [12], and has been trained via the AMSgrad algorithm [22] with batch size of 32 samples per batch. The training cost function in (4) is defined by the hyper-parameters $\zeta = 0.03$, $\varepsilon = 0.25$, $\beta = 2$, and $\gamma = 10$.

All the reported results were obtained on a laptop with Intel Core i5 6200u (2.3GHz) with 8 Gb RAM running Ubuntu 16.04 using Pyhton 3.5. We used version 1.5.0rc1 of Tensorflow and 2.1.3 of Keras.

C. Sensitivity to state and information vector dimensions

We evaluate the fit performance obtained when applying our identification method with different sizes n_x of the state x_k and n_a, n_b of the information vector I_k. We set $m = 2$ in all tests as we are only interested in predicting the next output y_{k+1} and filter the current one y_k.

Performance is compared in terms of the best fit rate (BFR)

$$\text{BFR} = 100\left(1 - \frac{\sqrt{\sum_{k=k_0}^{N}(y_t - \hat{y}_t)^2}}{\sqrt{\sum_{k=k_0}^{N}(y_t - \bar{y})^2}}\right)\% \tag{10}$$

and Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=k_0}^{N}(y_t - \hat{y}_t)^2} \tag{11}$$

In (10)–(11) $\bar{y}$ is the average of the output signal y over $N - k_0 + 1$ samples, $k_0 = \max\{n_a, n_b\}$, and we consider different choices for $\hat{y}_t$:

- (i) *one-step ahead prediction* $\hat{y}_k$, extracted from $\hat{O}_k = d(e(I_{k-1}))$;
- (ii) *open-loop prediction*, in which $\hat{y}_k$ is extracted from $\hat{O}_k = d(\hat{x}_k)$, where

$$\begin{array}{rcl} \hat{x}_{k+1} & = & f(\hat{x}_k, u_k),\ k = k_0, \ldots, N-1 \\ \hat{x}_{k_0} & = & e(I_0) \end{array} \tag{12}$$

- (iii) $\hat{y}_k$ generated by the *neural observer*, i.e., extracted from $\hat{O}^\star_k = d(\hat{x}_k)$, where $\hat{x}_{k+1} = s(\hat{x}_k, u_k, y_k)$, $k = k_0, 1, \ldots, N-1$, $x_{k_0} = e(I_0)$;
- (iv) $\hat{y}_k = \hat{y}_{k|k}$ generated by the *extended Kalman filter*, i.e., extracted from $O_{k|k} = d(\hat{x}_{k|k})$, where $\hat{x}_{k|k}$ is the state estimate produced by the EKF, tuned with variances consistent with those of the noise signals $w(k)$, $v(k)$.

Table II reports the results obtained with $n_x = 6$, $n_a = n_b = 12$, which requires ~230 s CPU time to solve the training problem (4), while Table III reports the results achieved instead with $n_x = 2$, $n_a = n_b = 7$, which requires ~25 s CPU time. Figure 2 shows some of the simulation traces corresponding to the results reported in Table II.

	BFR	RMSE
1-step ahead	94.0%	0.174
open-loop	87.5%	0.348
neural observer	91.0%	0.264
EKF	93.0%	0.200

TABLE II

FIT RESULTS OBTAINED WITH $n_x = 6$, $n_a = n_b = 12$, 80000 SAMPLES

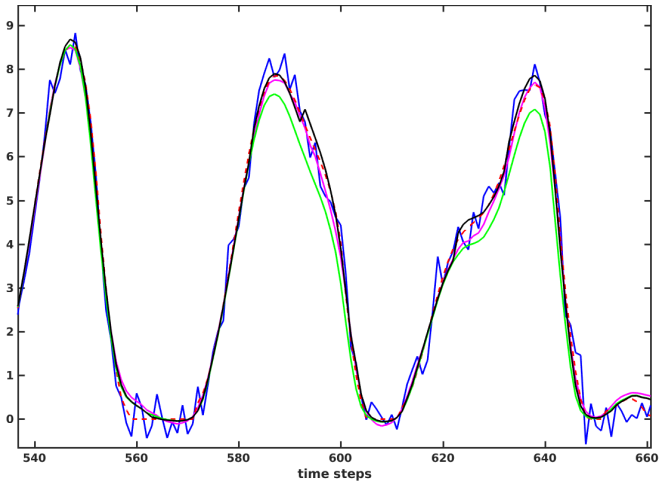

Fig. 2. Estimated output signal $\hat{y}_k$ with $n_x = 6$, $n_a = n_b = 12$: noise-corrupted measured signal (blue), noise-free signal (dashed red), 10-steps ahead w/ reset prediction (black), open-loop simulation (green), EKF filtered output $y_{t|t}$ (magenta).

D. Sensitivity with respect to dataset size

As for most deep-learning approaches, the training algorithm based on (4) may require a high number N of samples to converge to a good model. Table IV report the performance obtained under the same network topology with $n_x = 6$, $n_a = n_b = 12$ when we use a reduced training dataset of $N = 10000$ samples. No attempt has been made to revise the topology of the networks with respect to the previous tests. The CPU time required for training the model is ≈ 30 s.

Although the performance of open-loop prediction deteriorates in a perceivable way, results are still acceptable for limited ahead prediction and using a state observer. This suggests that the obtained model may still good enough for control design, as confirmed in Section VI-E.

E. Linear time-varying MPC based on nonlinear model

To control the tank system (9), we apply the LTV-MPC Algorithm 1 based on the nonlinear model identified with $n_x = 3$, $n_a = n_b = 7$ trained on a dataset further reduced to $N = 8000$ samples, that achieves 61% BFR in open-loop prediction. We employ the neural state observer, although very similar results have been obtained using the EKF.

Figure 3 reports the closed-loop results obtained with the LTV-MPC controller defined by $n_p = n_m = 4$, $W_y = 10$, $W_u = 0.1$, $W_{\Delta u} = 0.3$, under slew-rate constraints $|\Delta u_k| \leq 1.5$. Each simulation step, that includes simulating the system, updating the state observer, and computing the LTV-MPC solution, is carried out in 40 ms.

	BFR	RMSE
1-step ahead	89.2%	0.303
open-loop	82.5%	0.530
neural observer	89.0%	0.314
EKF	87.4%	0.369

TABLE III

FIT RESULTS OBTAINED WITH $n_x = 2$, $n_a = n_b = 7$, 80000 SAMPLES

	BFR	RMSE
1-step ahead	89.2%	0.305
open-loop	67.1%	1.160
neural observer	90.0%	0.296
EKF	82.9%	0.540

TABLE IV

FIT RESULTS OBTAINED WITH $n_x = 6$, $n_a = n_b = 12$, 10000 SAMPLES

F. Comparison with other identification techniques

We compare our approach with more classical nonlinear neural ARX (NNARX) techniques. To this end, we use the `nlarx` function implemented in the Systems Identification Toolbox for MATLAB [23], based on the regressor $[y(t-1)\ \ldots\ y(t-n_a)\ u(t-n_k)\ \ldots\ u(t-n_k-n_b+1)]'$ and a feedforward neural network as the nonlinearity [24]. Table V reports the performance obtained in cross-validation in terms of BFR when reproducing the output signal in open-loop simulation.

Table V also shows the number of states of the identified models, which is n_x in our approach and $n_a + n_b + n_k - 2$ for NNARX models. It is apparent that for the same number of states our method is able to better reproduce the open-loop response on test data. Note also that the NNARX model identified with $n_a = n_b = 12$ corresponds to using the entire uncompressed information vector I_k.

Finally, we remark that in our approach we restricted the activation functions of the neurons to be differentiable. If

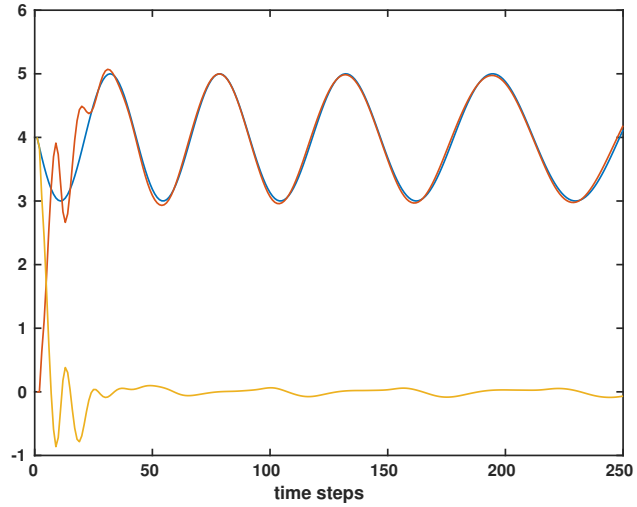

Fig. 3. Nonlinear tank system benchmark: Tracking a sine-sweep signal (red) with LTV MPC based on a neural observer as state observer. The controlled output is shown in blue, the tracking error in yellow

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 $r_t \in \mathbb{R}^{n_y}$, $u_t^r \in \mathbb{R}^{n_u}$, $t = 0, 1, \ldots$; current state estimate $\hat{x}_{t|t-1}$.

1. **estimate** $\hat{x}_{t|t}$ from $\hat{x}_{t|t-1}$, u_t, y_t (e.g., using EKF);
2. **compute** the sequence of predicted states $\{\bar{x}_{t+1}, \ldots, \bar{x}_{t+n_p}\}$ and outputs $\{\bar{y}_{t+1} \cdots \bar{y}_{t+n_p}\}$ given the current guess of the input sequence $\{\bar{u}_t \cdots u_{t+\bar{n}_p-1}\}$, with $\bar{u}_{t+k} = \bar{u}_{t+n_m-1}$ for all $k \geq n_m - 1$;
3. **compute** the Jacobians

$$A_k = \frac{\partial f}{\partial x_k}(\bar{x}_{t+k}, \bar{u}_{t+k}), \quad B_k = \frac{\partial f}{\partial u_k}(\bar{x}_{t+k}, \bar{u}_{t+k})$$
$$C_k = \frac{\partial g}{\partial x_k}(\bar{x}_{t+k}, \bar{u}_{t+k})$$

4. **solve** the quadratic programming problem of LTV-MPC to get the optimal sequence $u_t^\star, \ldots, u_{t+n_m-1}^\star$;
5. **set** the current input $u_t = u_t^\star$;
6. **update** the estimate $\hat{x}_{t+1|t} = f(\hat{x}_{t|t}, u_t)$;
7. **update** the nominal input sequence $\bar{u}_{t+k} = u_{t+k}^\star$, $1 \leq k \leq n_m - 1$, $\bar{u}_{t+k} = u_{t+n_m-1}^\star$, $n_m \leq k \leq n_p - 1$.

Output: Command input u_t, updated nominal input sequence $\{\bar{u}_{t+1}, \ldots, \bar{u}_{t+n_p-1}\}$, updated state estimate $\hat{x}_{t+1|t}$.

Algorithm 1: LTV-MPC algorithm

method	parameters	# states	BFR
NNARX	$n_a = 2$, $n_b = 1$, $n_k = 1$	2	75.0%
NNARX	$n_a = 3$, $n_b = 4$, $n_k = 1$	6	82.9%
NNARX	$n_a = n_b = 12$, $n_k = 1$	23	94.9%
Eq. (4)	$n_x = 6$, $n_a = n_b = 12$	6	87.5%
Eq. (4)	$n_x = 2$, $n_a = n_b = 7$	2	82.5%

TABLE V

COMPARISON WITH NONLINEAR NEURAL ARX REGRESSION: BFR OBTAINED IN OPEN-LOOP PREDICTION (80000 SAMPLES)

such an assumption is relaxed, better open-loop fit figures can be obtained. However, we have not tested non-smooth models extensively, as we are interested in obtaining the Jacobian matrices required by the EKF and MPC controller.

VII. CONCLUSIONS

In this paper we introduced a novel methodology based on autoencoders and neural networks to estimate nonlinear state-space models from input/output data. The approach well captures nonlinear models with a direct control of the order of the state vector. The technique is especially appealing for model-based control design and opens the avenue to the synthesis of nonlinear model predictive controllers from black-box input/output data. Due to the flexibility of the proposed approach, we envision several extensions, such as for the identification of gray-box models, for the synthesis of virtual sensors, and for model reduction of nonlinear state-space models.

ACKNOWLEDGEMENTS

The authors thank Professor Alberto Tesi for providing invaluable suggestions during the development of this work.

REFERENCES

[1] G. Pillonetto, F. Dinuzzo, T. Chen, G. D. Nicolao, and L. Ljung, "Kernel methods in system identification, machine learning and function estimation: A survey," *Automatica*, vol. 50, no. 3, pp. 657 – 682, 2014.

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

[3] V. Breschi, D. Piga, and A. Bemporad, "Piecewise affine regression via recursive multiple least squares and multicategory discrimination," *Automatica*, vol. 73, pp. 155–162, Nov. 2016.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, July 2006.

[5] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010.

[6] N. Takeishi, Y. Kawahara, and T. Yairi, "Learning Koopman invariant subspaces for dynamic mode decomposition," *CoRR*, vol. abs/1710.04340, 2017.

[7] R. L. Williams and D. A. Lawrence, *Minimal Realizations*, pp. 185–197. John Wiley Sons, Inc., 2007.

[8] Y. Baram, "Minimal order representation, estimation and feedback of continuous-time stochastic linear systems," in *Mathematical Theory of Networks and Systems* (P. A. Fuhrmann, ed.), (Berlin, Heidelberg), pp. 24–41, Springer Berlin Heidelberg, 1984.

[9] I. Guyon and A. Elisseeff, *An Introduction to Feature Extraction*, pp. 1–25. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[10] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, pp. 930–945, May 1993.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec 1989.

[12] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[13] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Comput.*, vol. 12, pp. 1207–1245, May 2000.

[14] B. Anderson and J. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[15] A. Bemporad, "Model-based predictive control design: New trends and tools," in *Proc. 45th IEEE Conf. on Decision and Control*, (San Diego, CA), pp. 6678–6683, 2006.

[16] D. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[17] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear MPC: bridging the gap via the real-time iteration," *Int. Journal of Control*, 2016. In press.

[18] M. Schoukens and J. Noël, "Three benchmarks addressing open challenges in nonlinear system identification," *20th World Congress of the International Federation of Automatic Control, pp.448-453, Toulouse, France, July 9-14, 2017.*

[19] K. K. Chen, J. H. Tu, and C. W. Rowley, "Variants of dynamic mode decomposition: Boundary condition, Koopman, and Fourier analyses," *Journal of Nonlinear Science*, vol. 22, pp. 887–915, Dec 2012.

[20] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," *CoRR*, vol. abs/1511.07289, 2015.

[21] F. Chollet *et al.*, "Keras." https://github.com/keras-team/keras, 2015.

[22] S. K. Sashank J. Reddi, Satyen Kale, "On the convergence of Adam and beyond," *International Conference on Learning Representations*, 2018.

[23] L. Ljung, *System Identification Toolbox for MATLAB – User's Guide*. The Mathworks, Inc.

[24] The MathWorks Inc., "Class representing neural network nonlinearity estimator for nonlinear arx models." https://it.mathworks.com/help/ident/ref/neuralnet.html.