Autoencoder Neural Networks for LPV Embedding of Nonlinear Systems

Arash Sadeghzadeh, Pierre-Loïc Garoche

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 56-2 (2023) 9062–9067

Autoencoder Neural Networks for LPV Embedding of Nonlinear Systems ⋆

Arash Sadeghzadeh * **Pierre-Loïc Garoche** *

* *École Nationale de l'Aviation Civile, Université de Toulouse, France (e-mail: {pierre-loic.garoche, arash.sadeghzadeh}@enac.fr).*

Abstract: In this paper, the problem of automated generation of linear parameter-varying (LPV) state-space models is addressed. A deep neural network (DNN) is developed to embed the dynamical behavior of a nonlinear (NL) system into an LPV model with predefined number of scheduling variables which are the NL functions of the states. Leveraging the Autoencoder (AE) neural networks (NN) and using the input-output plant data, a scheduling NL mapping is defined. The developed LPV model depends affinely on the scheduling variables. Since the proposed method to derive LPV model is based on input-output plant data, the explicit NL equations of the plant are not required. The upper and lower bounds on the scheduling variables can be computed by solving convex optimization problems. The effectiveness of the proposed method is evaluated on a benchmark example.

Keywords: Deep Neural Networks, Autoencoder Neural Networks, Linear Parameter-Varying Systems, Nonlinear Systems

1. INTRODUCTION

The linear parameter-varying (LPV) framework has received significant attention for performance analysis and control synthesis for nonlinear (NL) and time-varying (TV) systems in the last few decades. This methodology is capable to address versatile practical applications in diverse fields (Hoffmann and Werner, 2015). However a major limitation to a wider utilization of this powerful tool is the lack of systematic approaches to embed a NL model into an LPV one. Most of the available methods are ad-hoc approaches and usually depend on the expertise of the user. In the available automated procedures, the obtained LPV model usually depends either affinely or polynomially on the scheduling variables, see e.g. Schoukens and Tóth (2018); Sadeghzadeh et al. (2020); Kwiatkowski et al. (2006); Sadeghzadeh and Tóth (2022). Another challenging issue of the existing embedding procedures is that they usually lead to an LPV model with a large amount of scheduling variables. This has triggered a research line for scheduling dimension reduction. Some methods based on principle component analysis (PCA) (Kwiatkowski and Werner, 2008), Kernel PCA (Rizvi et al., 2016), and Autoencoders (AE) (Rizvi et al., 2018; Koelewijn and Tóth, 2020; de Lange et al., 2022) have already been developed for the scheduling dimension reduction. If a first-principles NL model of a system exists, then one may resort to conversion approaches to derive an LPV model for the NL system, see e.g. Caigny et al. (2011); Kwiatkowski et al. (2006); Robles et al. (2019); Petersson and Löfberg (2009); Leith and Leithead (1998).

In this paper, thanks to AE NN, a novel method to embed the dynamical behavior of a NL system into an LPV model is presented. An AE is a NN structure capable of compressing input data, and then reconstructing it as an output. AE comprises two parts, encoder and decoder. Using the compressing feature of the AE performed by encoder part, the state variables can be encoded into scheduling variables. This implies that the encoder can be considered as a scheduling NL mapping based on which the information of the state variables are encoded into a few number of scheduling variables. The input to the decoder are scheduling variables $\theta(k)$, plant input signal $u(k)$, and the current state variables $x(k)$, and the output is $x(k+1)$. The decoder is trained based on these input and output signals.Training an AE is considered an unsupervised learning technique since the training is performed without explicit labeled data.

In Koelewijn and Tóth (2020), a deep neural network (DNN) is employed for the scheduling dimension reduction. The input to the DNN are the state-space matrices of an initially given LPV model, while the output of the DNN are the state-space matrices of the desired LPV model having a reduced number of scheduling variables. The embedding procedure proposed in de Lange et al. (2022) is based on an AE NN. The input to the AE are the state and plant input variables and the output of the AE are the state space matrices of the desired LPV model. The training is carried out by minimizing a cost function representing the difference between an initially given LPV model and the desired LPV model. In both these methods, one requires an initially given LPV model for the original nonlinear system. However, taking advantage of AE NN, it is possible to bypass the requirement for having an initial LPV model which can be problematic when an explicit description of the NL dynamical behavior of the NL system is not available. We provide a method that just requires the input-output data of the NL system to train an AE NN,

⋆ This work was partially supported by the projects ANR FEANICSES (ANR-17-CE25-0018) and ESA AITIVE-GNC.

10.1016/j.ifacol.2023.10.137

which, in turn, results in the state-space representation of an LPV model.

The paper is structured as follows, Section 2 is devoted to problem definition. The details of the proposed method are provided in Section 3. Application of the provided approach on an inverted pendulum benchmark is explored in Section 4. Finally, some conclusions are drawn in the last section.

2. PROBLEM DEFINITION

Consider a NL discrete-time dynamical system with the following state-space representation:

$$\begin{aligned} x(k+1) &= f(x(k), u(k)), \\ y(k) &= g(x(k), u(k)), \end{aligned} \tag{1}$$

where $x(k) \in \mathbb{X} \subset \mathbb{R}^n$ is the state vector, $u(k) \in \mathbb{R}^p$ is the input, and $y(k) \in \mathbb{R}^m$ is the output of the system. $\mathbb{X}$ is assumed to be a compact polyhedron with known vertices. $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n$ and $g : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ are bounded and smooth static real-valued nonlinear functions of x and u. We consider the problem of embedding a NL model into an LPV representation. This means that the goal is to obtain the following LPV representation for the NL system (1):

$$\begin{aligned} x(k+1) &= A(\theta(k))x(k) + B(\theta(k))u(k), \\ y(k) &= C(\theta(k))x(k) + D(\theta(k))u(k), \end{aligned} \tag{2}$$

such that $\theta(k) \coloneqq \mu(x(k))$ where the nonlinear scheduling map $\mu : \mathbb{X} \to \Theta \subseteq \mathbb{R}^{n_\theta}$ is automatically constructed, and the scheduling variable $\theta(k) \coloneqq [\, \theta_1(k) \;\; \theta_2(k) \;\; \cdots \;\; \theta_{n_\theta}(k) \,]^\top$ is supposed to belong to the hyperrectangle Θ defined by

$$\underline{\theta}_i \le \theta_i(k) \le \overline{\theta}_i, \quad i = 1, \cdots, n_\theta \tag{3}$$

with $\underline{\theta}_i, \overline{\theta}_i \in \mathbb{R}$ which are defined in the procedure. Furthermore, it is assumed that the embedding of (1) is performed such that the state space matrices A, B, C, and D of compatible dimension have affine dependence on $\theta(k)$, i.e.,

$$\begin{bmatrix} A(\theta) & B(\theta) \\ C(\theta) & D(\theta) \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \sum_{j=1}^{n_\theta} \begin{bmatrix} A_j & B_j \\ C_j & D_j \end{bmatrix} \theta_j. \tag{4}$$

Suppose that we have a representative dataset $\mathcal{D}$ defined as

$$\mathcal{D} : \{x(k), u(k), y(k)\}_{k=0}^{N} \tag{5}$$

representing typical operational behavior of the system. The error or the distance between the LPV model and the original one can be captured by the error between the output of the encoder, i.e., $[\, \hat{x}(k+1)^\top \;\; \hat{y}(k)^\top \,]^\top$, and the ground truth $[\, x(k+1)^\top \;\; y(k)^\top \,]^\top$. It can be expressed as a mean squared error (MSE) or any relevant appropriate error. Here we use the following index (note that it corresponds to MSE when $\alpha = 1$):

$$\begin{aligned} \min_{A,B,C,D,\mu} \; & \frac{1}{N} \sum_{j=1}^{N} \|\hat{x}(k+1) - x(k+1)\|_2^2 + \alpha \, \|\hat{y}(k) - y(k)\|_2^2 \\ s.t. \quad & x(k), u(k), y(k) \in \mathcal{D} \end{aligned} \tag{6}$$

where

$$\begin{aligned} \hat{x}(k+1) &= A(\theta(k))x(k) + B(\theta(k))u(k), \\ \hat{y}(k) &= C(\theta(k))x(k) + D(\theta(k))u(k), \end{aligned}$$

The state space matrices A, B, C, D and the scheduling map function μ are obtained by solving this optimization problem with an appropriate weighting scalar α. Note that $\mathcal{D}$ comprises system state, input, and output trajectories obtained by closed-loop simulation of system (1) using any arbitrary stabilizing controller.

3. PROPOSED METHOD

To tackle the optimization problem (6), we take advantage of AE NN structure. AE NN has two main parts, i.e., encoder and decoder. Encoder maps the input into a lower dimensional latent representation, then the decoder reconstructs the output from the latent representation. Conventional AE NN is meant to learn unlabeled data and it typically used to reduce dimensionality. Here we rely on this NN architecture to specifically address the LPV embedding problem of NL systems. We specialize the AE NN as follows: the encoder part encodes the state variables of the original NL system into the scheduling variables, so in our AE NN the latent representation is the scheduling vector. Then, resorting to the obtained scheduling variables $\theta(k)$ and using the state vector $x(k)$ and the input $u(k)$ of the system, the decoder reconstruct the next state vector , i.e., $x(k+1)$ and the output $y(k)$. Figure 1 reveals the idea for our modified AE NN. Without loss of generality and just for the ease of representation, we assume that the encoder has $n_e = 2$ layers, and we consider $n_\theta = 2$ scheduling variables in Fig. 1.

The encoder is a feed-forward fully-connected NN with n_e layers. Therefore, it can be represented as follows:

$$\mu : \begin{cases} a_e^{[0]}(k) = x(k), \\ v_e^{[l]}(k) = W_e^{[l]} a_e^{[l-1]}(k) + b_e^{[l]}, & l = 1, \cdots, n_e \\ a_e^{[l]}(k) = \Phi_e^{[l]}(v_e^{[l]}(k)), & l = 1, \cdots, n_e \\ \theta(k) = a_e^{[n_e]}(k), \end{cases} \tag{7}$$

where $a_e^{[l]}$ is the output of the lth layer of the encoder. Let n_l denote number of neurons in layer l; thus, we have $W_e^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $b_e^{[l]} \in \mathbb{R}^{n_l}$. Here, $\Phi_e^{[l]}(\cdot)$ represents the activation function of layer l and is applied element-wise on $v_e^{[l]}$. Note that the input to the encoder is the state vector $x(k)$ of the original nonlinear system and the output is the scheduling vector $\theta(k)$, and the encoder represents the NL scheduling map μ. The weighting matrix $W_e^{[l]}$ and the bias vector $b_e^{[l]}$ for all $l = 1, \cdots, n_e$ are obtained by training the AE NN.

Note that one may consider both $x(k)$ and $u(k)$ as the input to the encoder. This way, the scheduling variables would also be nonlinear functions of $u(k)$.

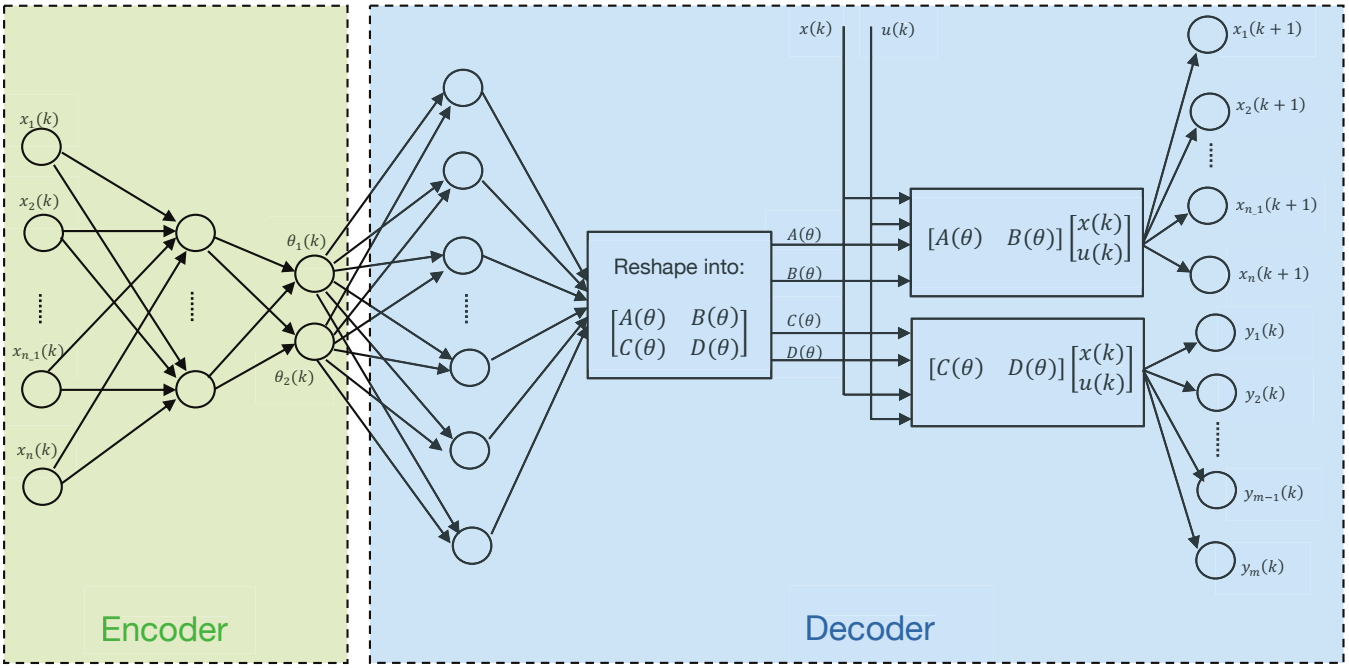

Fig. 1. Modified AE NN for embedding of NL systems into LPV model.

To proceed further, let a be defined as follows:

$$a = \begin{bmatrix} a_1 & a_2 & \cdots & a_{qr} \end{bmatrix}^\top \in \mathbb{R}^{qr},$$

then

$$\underset{q \times r}{\overleftarrow{a}} := \begin{bmatrix} a_1 & a_2 & \cdots & a_r \\ a_{r+1} & a_{r+2} & \cdots & a_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ a_{qr-r+1} & a_{qr-r+2} & \cdots & a_{qr} \end{bmatrix} \in \mathbb{R}^{q \times r}.$$

Additionally, let us define q and r as:

$$q = n + p, \qquad r = n + m.$$

The decoder part in Fig. 1 can be formulated as follows:

$$\begin{cases} a_d = W_d\ \theta(k) + b_d, \\ \mathcal{M} = \underset{r \times q}{\overleftarrow{a_d}}, \\ \begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} = \mathcal{M} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}, \end{cases} \tag{8}$$

It is worth mentioning that the encoder just has one layer having $n_d = (n+m)(n+q)$ neurons, and the layer has a linear activation function. The output of this layer is denoted by a_d. Therefore, the elements of the matrices $A(\theta), B(\theta), C(\theta)$, and $D(\theta)$ depend affinely on the scheduling vector $\theta(k)$. The weighting matrix W_d and the bias vector b_d are obtained by training the AE NN.

It is worth mentioning that one may consider more layers for the decoder. In that case, the state-space matrices of the final LPV model would be nonlinear functions of the scheduling variables.

To train the AE NN, we consider a MSE cost function and the input and output data as $(x(k), u(k))$ and $(x(k+1), y(k))$, respectively. The training data are constructed utilizing the dataset $\mathcal{D}$, given in (5). After training the AE NN, the state space matrices of the LPV system are obtained as follows:

$$\begin{bmatrix} A(\theta) & B(\theta) \\ C(\theta) & D(\theta) \end{bmatrix} := \mathcal{M} \tag{9}$$

where $A(\theta) \in \mathbb{R}^{n \times n}$, $B(\theta) \in \mathbb{R}^{n \times p}$, $C(\theta) \in \mathbb{R}^{m \times n}$, and $D(\theta) \in \mathbb{R}^{m \times p}$.

One can obtain the upper and lower bounds on the scheduling variables, defined in (3), as explained in the sequel.

Let $\overline{v}_i^{[l]}$ and $\underline{v}_i^{[l]}$ denote the upper and lower bounds on $v_e^{[l]}$, respectively. Then, for the first layer in the encoder we have

$$\begin{aligned} \overline{v}_i^{[1]} = \max_x \quad & W_e^{[1]} x + b_e^{[1]} \\ s.t. \quad & x \in \mathbb{X} \end{aligned} \tag{10}$$

$$\begin{aligned} \underline{v}_i^{[1]} = \min_x \quad & W_e^{[1]} x + b_e^{[1]} \\ s.t. \quad & x \in \mathbb{X} \end{aligned} \tag{11}$$

for $i = 1, \cdots, n_1$ (n_1 is the number of neurons in the first layer of encoder). This way, we obtain $\overline{v}^{[1]}$ and $\underline{v}^{[1]}$ as follows:

$$\overline{v}^{[1]} := \begin{bmatrix} \overline{v}_1^{[1]} & \overline{v}_2^{[1]} & \cdots, \overline{v}_{n_1}^{[1]} \end{bmatrix}, \quad \underline{v}^{[1]} := \begin{bmatrix} \underline{v}_1^{[1]} & \underline{v}_2^{[1]} & \cdots, \underline{v}_{n_1}^{[1]} \end{bmatrix}.$$

Subsequently, since $\Phi_e^{[l]}(\cdot)$ is a non-decreasing function (usually ReLU, tanh, and sigmoid activation functions are used) we can obtain the upper and lower bounds on $a_e^{[1]}$ as

$$\overline{a}_e^{[1]} = \Phi_e^{[1]}(\overline{v}_e^{[1]}), \qquad \underline{a}_e^{[1]} = \Phi_e^{[1]}(\underline{v}_e^{[1]}).$$

Afterwards, resorting to forward propagation through all the layers of the encoder NN, we can obtain $\overline{v}^{[l]}$ and $\underline{v}^{[l]}$ for the other layers. Using affine arithmetic (de Figueiredo and Stolfi, 2004), we can define the center

$$v_c^{[l]} \coloneqq \frac{1}{2} W_e^{[l]} \left(\overline{a}_e^{[l-1]} + \underline{a}_e^{[l-1]} \right)$$

and the radius

$$v_r^{[l]} \coloneqq \frac{1}{2} \text{abs} \left(W_e^{[l]} \left(\overline{a}_e^{[l-1]} - \underline{a}_e^{[l-1]} \right) \right)$$

and then

$$\underline{v}_e^{[l]} \coloneqq v_c^{[l]} - v_r^{[l]}, \qquad \overline{v}_e^{[l]} \coloneqq v_c^{[l]} + v_r^{[l]}.$$

for $l = 2, \cdots, n_e$. Note that

$$\underline{a}_e^{[l]} = \Phi_e^{[l]}(\underline{v}_e^{[l]}), \qquad \overline{a}_e^{[l]} = \Phi_e^{[l]}(\overline{v}_e^{[l]})$$

for $l = 2, \cdots, n_e$. This way, we can eventually obtain

$$\overline{\theta} \coloneqq \overline{a}_e^{[n_e]}, \qquad \underline{\theta} \coloneqq \underline{a}_e^{[n_e]}$$

where

$$\overline{\theta} \coloneqq \left[\, \overline{\theta}_1 \;\; \overline{\theta}_2 \;\; \cdots \;\; \overline{\theta}_{n_\theta} \, \right]^\top \qquad \underline{\theta} \coloneqq \left[\, \underline{\theta}_1 \;\; \underline{\theta}_2 \;\; \cdots \;\; \underline{\theta}_{n_\theta} \, \right]^\top.$$

4. NUMERICAL EXAMPLE

To demonstrate the effectiveness of our proposed method, we consider the LPV embedding problem of an inverted pendulum on a cart. To train the AE NN, we used Keras (Chollet et al., 2015) in the TensorFlow library (Abadi et al., 2016) used in Jupyter notebooks (Kluyver et al., 2016) for Python [1].

The dynamics of the inverted pendulum are described as follows:

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = \dfrac{h_1(x_1, x_2)}{\Delta(x_1)} + \dfrac{h_2(x_1)}{\Delta(x_1)} \text{sat}(u), \\ \dot{x}_3 = x_4, \\ \dot{x}_4 = \dfrac{h_3(x_1, x_2)}{\Delta(x_1)} + \dfrac{h_4}{\Delta(x_1)} \text{sat}(u), \end{cases} \tag{12}$$

where

$$\begin{aligned} \Delta(x_1) =& (M+m)(J+ml^2) - m^2 l^2 \cos^2(x_1) \\ h_1(x_1, x_2) =& - f_1(M+m)x_2 - m^2 l^2 x_2^2 \sin(x_1)\cos(x_1) + \\ & f_0 m l x_4 \cos(x_1) + (M+m)mgl\sin(x_1) \\ h_2(x_1) =& - ml\cos(x_1) \\ h_3(x_1, x_2) =& f_1 m l x_2 \cos(x_1) + (J+ml^2)mlx_2^2 \sin(x_1) - \\ & f_0(J+ml^2)x_4 - m^2 g l^2 \sin(x_1)\cos(x_1) \\ h_4 =& J + ml^2 \end{aligned}$$

[1] All the codes are available on Github at: https://github.com/AraSadz/Autoencoder-Neural-Networks-for-LPV-Embedding-of-Nonlinear-Systems.git

where x_1 and x_2 represent the angle along the vertical and the angular velocity of the pendulum, respectively. x_3 and x_4 denote the displacement and the velocity of the cart, respectively. g is the gravity constant; m and M respectively refer to the mass of the pendulum and cart. f_0 and f_1 stand for the friction factor of the the cart and pendulum, respectively. l is the length from the center of mass of the pendulum to the shaft axis; J is the moment of inertia of the pendulum round its center of mass; u denote the force applied to the cart. The numerical values for the parameters are given in Table 1.

The $\text{sat}(u)$ is defined as follows:

$$\text{sat}(u) = \begin{cases} u & \underline{u} \le u \le \overline{u} \\ \overline{u} & u > \overline{u} \\ \underline{u} & u < \underline{u} \end{cases} \tag{13}$$

where $\overline{u} = -\underline{u} = 105$ is considered. It is also assumed that

$$\begin{aligned} -\frac{\pi}{6} \le x_1(k) \le \frac{\pi}{6}, \\ -5 \le x_2(k) \le 5. \end{aligned} \tag{14}$$

Table 1. Parameter values for the pendulum-cart system

M	m	g	f_0	f_1	l	J
1.3282	0.22	9.8	22.915	0.007056	0.304	0.004963

Now, using Euler's first-order approximation considering $T = 0.05sec.$, the following discrete-time model for the pendulum-cart system is obtained:

$$x(k+1) = \begin{bmatrix} x_1(k) + Tx_2(k) \\ x_2(k) + T\dfrac{h_1(x_1(k), x_2(k))}{\Delta(x_1(k))} + T\dfrac{h_2(x_1(k))}{\Delta(x_1(k))}\text{sat}(u(k)) \\ x_3(k) + Tx_4(k) \\ x_4(k) + T\dfrac{h_3(x_1(k), x_2(k))}{\Delta(x_1(k))} + T\dfrac{h_4}{\Delta(x_1(k))}\text{sat}(u(k)) \end{bmatrix} \tag{15}$$

where $x(k) \coloneqq \left[\, x_1(k) \;\; x_2(k) \;\; x_3(k) \;\; x_4(k) \, \right]^\top$.

To obtain an LPV model for (15) using our proposed method, an operational behaviour dataset $\mathcal{D}$, given by (5), is required. One can derive this dataset exploiting closed-loop time-domain simulations. To do so, we need first a stabilizing controller. To design the controller, (5) is linearized around the equilibrium point $x = 0$ to obtain an approximate linear model. Then, using the *dlqr* function in Matlab, an stabilizing state feedback controller is designed as follows:

$$K = \left[\, -109.3936 \;\; -21.1989 \;\; -8.0214 \;\; -43.1398 \, \right] \tag{16}$$

Now, using time-domain simulation, we obtain 10000 trajectories of $x(t)$ and $u(t)$ considering different initial states in the ranges given in (14). For the simulations, we assume $x_3(0) = x_4(0) = 0$. The details of the encoder NN is provided in Table 2.

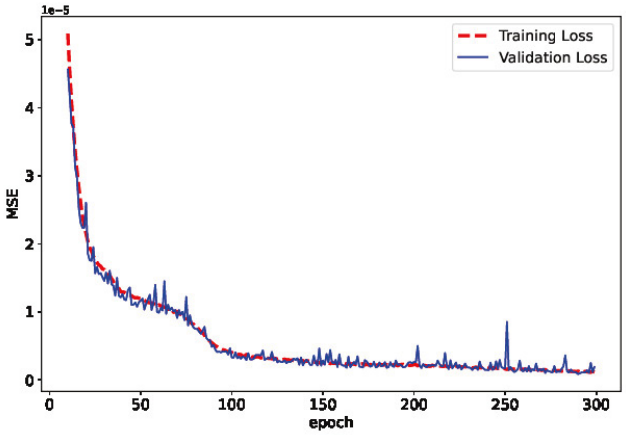

Fig. 2. Training loss and validation loss of training AE NN for LPV embedding of pendulum-cart system.

Note that for this example $g(\cdot)$ is not defined. Therefore, we can ignore the y's in the output of the decoder by modifying $\mathcal{M}$, given in (8), as follows:

$$\mathcal{M} = \underbrace{\begin{bmatrix} a_d \end{bmatrix}}_{n \times q},$$

and then

$$x(k+1) = \mathcal{M} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}.$$

For this case, the number of neurons in the decoder layer equals to $n(n+p) = 4 \times 5 = 20$.

Note that in our dataset $\mathcal{D}$, the state variables are of different scales. Therefore, to improve the trained model and make the training procedure faster, we need to scale the state variables. One can use *MaxAbsScaler* from the scikit-learn library (Pedregosa et al., 2011) for the Python to scale each state variable by its maximum absolute value. This way, the maximum absolute value of each state in the training set will be 1. However, it does not shift the data, and thus does not shift the state values from 0 (the final state values in our time-domain simulations).

To train the AE NN, we use the well known Adam optimization (Kingma and Ba, 2014). Our dataset comprises 1980000 data samples. To train the AE NN, we consider a batch size of 1024 samples with 300 epochs and the MSE loss function. We consider 20% of the data for cross-validation. In Fig. 2, the training loss and validation loss versus epoch are depicted.

Consider a time-domain simulation of the original NL system using the state feedback controller (16) from the initial state $x(0) = [\,\pi/6 \;\; 5 \;\; 0 \;\; 0\,]^\top$. This way, we obtain two time series for $x(k)$ and $u(k)$. Now, for the comparison purposes, let compare the output of the AE NN, i.e. $\hat{x}(k)$, with $x(k)$. In Fig. 3, $x(k)$ and $\hat{x}(k)$ are depicted. One can

Table 2. Details of the encoder NN.

	Layer 1	Layer 2	Layer 3
No. neurons/Activation	20/ReLU	10/ReLU	1/ReLU

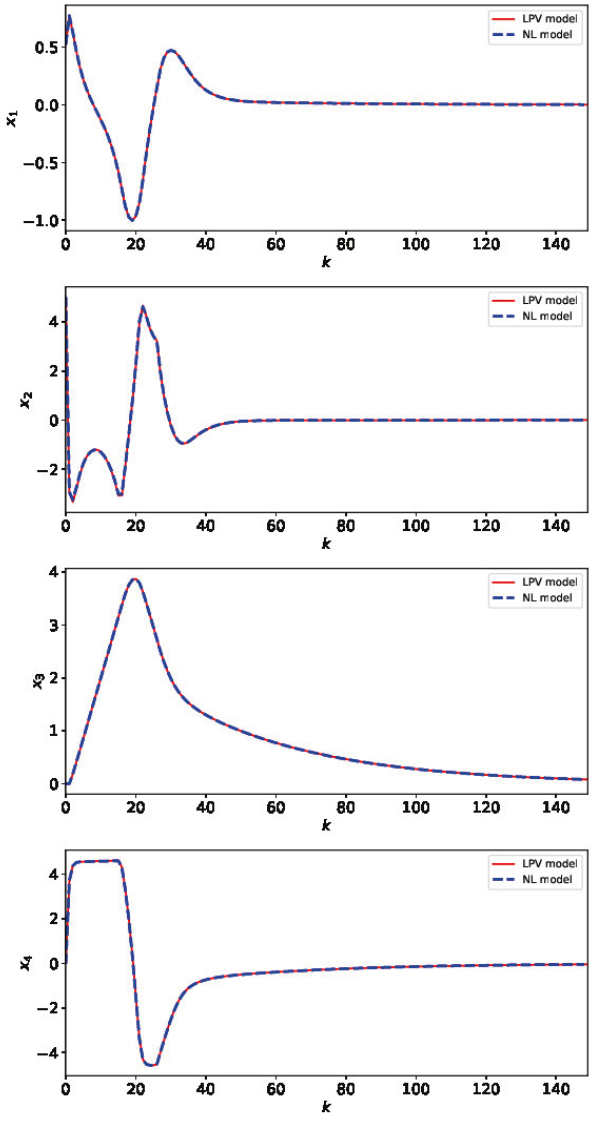

Fig. 3. Comparison between the LPV model and the original NL model.

see that the output of the AE NN perfectly matches the ground truth.

As we mentioned earlier, the output of the encoder NN provides the scheduling variables. In Fig. 4, the scheduling variable θ_1 is depicted for the time-domain simulation we mentioned earlier. One can easily see that the proposed method provides an accurate LPV model for this benchmark example.

5. CONCLUSION

A novel method to systematically embed the dynamical behavior of a NL system into an LPV model is proposed. Using AE NN, a NL mapping from the state variables to a predefined number of scheduling variables is introduced. Even though the introduced NL scheduling mapping just depends on the state variables, extension to the case that the mapping depends on both state and input signals is also straightforward. Leveraging this NL mapping, an LPV model is developed using the input-output data. The

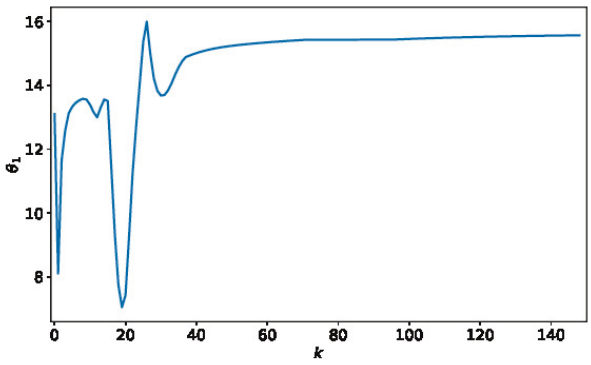

Fig. 4. Scheduling variable θ_1 for the obtained LPV model for the inverted pendulum benchmark.

proposed method does not require the exact NL state-space representation of the original NL system; in return, the input-output data of the operational behavior of the NL system is employed. The state-space matrices of the LPV model are affine functions of the scheduling variables. Nevertheless, by increasing the number of the layers in the decoder part, one can attain a more complex dependency on the introduced scheduling variables. This method has already been applied with success in an ESA project on a larger use case of a Mars lander scenario. It was able to provide a faithful LPV model for a large and complex input Simulink model containing many nested subsystems, lookup tables and nonlinearities. For future work, we aim to perform more comparison with the other available approaches, and to quantify the impact of increasing the number of the layers in the encoder and decoder on the overall quality of the model.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 265–283.

Caigny, J.D., Camino, J.F., and Swevers, J. (2011). Interpolation-Based Modeling of MIMO LPV Systems. *IEEE Transactions on Control Systems Technology*, 19(1), 46–63.

Chollet, F. et al. (2015). Keras. URL `https://github.com/fchollet/keras`.

de Figueiredo, L.H. and Stolfi, J. (2004). Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37, 147–158.

de Lange, M.H., Verhoek, C., Preda, V., and Tóth, R. (2022). LPV modeling of the atmospheric flight dynamics of a generic parafoil return vehicle. URL `https://arxiv.org/abs/2205.09555`.

Hoffmann, C. and Werner, H. (2015). A survey of linear parameter-varying control applications validated by experiments or high-fidelity simulations. *IEEE Transactions on Control Systems Technology*, 23(2), 416–433.

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. URL `http://arxiv.org/abs/1412.6980`.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 87 – 90. IOS Press.

Koelewijn, P. and Tóth, R. (2020). Scheduling dimension reduction of lpv models - a deep neural network approach. In *2020 American Control Conference (ACC)*, 1111–1117. doi:10.23919/ACC45564.2020.9147310.

Kwiatkowski, A., Boll, M., and Werner, H. (2006). Automated generation and assessment of affine LPV models. In *Proc. of the 45th IEEE Conference on Decision and Control*, 6690–6695. San Diego, CA, USA.

Kwiatkowski, A. and Werner, H. (2008). PCA-based parameter set mappings for LPV models with fewer parameters and less overbounding. *IEEE Transactions on Control Systems Technology*, 16(4), 781–788.

Leith, D.J. and Leithead, W.E. (1998). Gain-scheduled and nonlinear systems: Dynamic analysis by velocity-based linearization families. *International Journal of Control*, 70(2), 289–317.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Petersson, D. and Löfberg, J. (2009). Optimization based LPV-approximation of multi-model systems. In *Proc. of the European Control Conference*, 3172–3177. Budapest, Hungary.

Rizvi, S.Z., Abbasi, F., and Velni, J.M. (2018). Model reduction in linear parameter-varying models using autoencoder neural networks. In *2018 Annual American Control Conference (ACC)*, 6415–6420. doi: 10.23919/ACC.2018.8431912.

Rizvi, S.Z., Mohammadpour, J., Tóth, R., and Meskin, N. (2016). A kernel-based pca approach to model reduction of linear parameter-varying systems. *IEEE Transactions on Control Systems Technology*, 24(5), 1883–1891.

Robles, R., Sala, A., and Bernal, M. (2019). Performance-oriented quasi-LPV modeling of nonlinear systems. *International Journal of Robust and Nonlinear Control*, 29(5), 1230–1248.

Sadeghzadeh, A., Sharif, B., and Tóth, R. (2020). Affine linear parameter-varying embedding of nonlinear models with improved accuracy and minimal overbounding. *IET Control Theory and Applications*, 14, 3363–3373(10).

Sadeghzadeh, A. and Tóth, R. (2022). Improved embedding of nonlinear systems in linear parameter-varying models with polynomial dependence. *IEEE Transactions on Control Systems Technology*, 1–13. doi: 10.1109/TCST.2022.3173891.

Schoukens, M. and Tóth, R. (2018). Linear parameter varying representation of a class of MIMO nonlinear systems. In *Proc. of the 2nd IFAC Workshop on Linear Parameter Varying Systems*, 94 – 99. Florianopolis, Brazil.