LEARNING NEURAL STATE-SPACE MODELS: DO WE NEED A STATE ESTIMATOR?

Marco Forgione, Manas Mejari, and Dario Piga

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

ABSTRACT

In recent years, several algorithms for system identification with neural state-space models have been introduced. Most of the proposed approaches are aimed at reducing the computational complexity of the learning problem, by splitting the optimization over short sub-sequences extracted from a longer training dataset. Different sequences are then processed simultaneously within a minibatch, taking advantage of modern parallel hardware for deep learning. An issue arising in these methods is the need to assign an initial state for each of the sub-sequences, which is required to run simulations and thus to evaluate the fitting loss. In this paper, we provide insights for calibration of neural state-space training algorithms based on extensive experimentation and analyses performed on two recognized system identification benchmarks. Particular focus is given to the choice and the role of the initial state estimation. We demonstrate that advanced initial state estimation techniques are really required to achieve high performance on certain classes of dynamical systems, while for asymptotically stable ones basic procedures such as zero or random initialization already yield competitive performance.

***K*eywords** deep learning, state-space models, system identification.

1 Introduction

While neural networks have been employed for dynamical systems modeling since (at least) the early 90s of the last century [4], the recent impressive achievements of deep learning in diverse domains [24] has sparked renewed interest from the system identification community.

Nowadays, the combined availability of flexible open-source software for deep learning [21] and cheap hardware supporting massive parallel computing enables rapid prototyping and testing of different neural architectures. Leveraging these tools, tailor-made neural model structures designed for dynamical system modeling have been developed. To cite a few examples, state-space models embedding *a priori* system knowledge, and using neural networks to describe the uncertain system components have been showcased in [6]. Novel deep-learning architectures embedding linear transfer function as elementary building blocks have been introduced in [7], along with a specialized and efficient implementation of the back-propagation computations for the transfer-function operator. Finally, [9, 13] propose modeling approaches based on Koopman operator theory. The system state is lifted to a high-dimensional space where the dynamics evolve linearly. This linear dynamics, along with the non-linear state transformation (parametrized as a neural network) are jointly learned with deep-learning tools.

System identification researchers have also worked towards the development of efficient algorithms to fit these neural model structures to experimental data. In particular, when the training dataset consists of a single, long (w.r.t. the system's time constants) sequence, standard simulation-error minimization has been show to be either computationally inefficient or even infeasible [6, 23]. Conversely, one-step-ahead approaches may be sensitive to measurement noise, and in general they are less effective in producing reliable long-term predictions.

An intermediate approach has thus emerged, where an m-step-ahead prediction error is minimized over *sub-sequences* extracted from the training dataset [6, 1, 15]. The length m of the considered sub-sequences is an important hyper-parameter of the training algorithm. This approach presents several advantages. First, from a computational perspective, several sub-sequences can be processed simultaneously, thus taking full advantage of modern parallel hardware [6]. Second, as demonstrated in [23], the optimization problem to be solved when minimizing the prediction error over a very long horizon may be ill-posed, notably in the case of unstable and chaotic dynamics. The multi-step-ahead prediction method enables controlling this effect through tuning of the sub-sequence length m. Lastly, the ultimate aim of modeling is often to generate accurate predictions over a specific time horizon. Notably, this occurs when the aim is to design a model predictive controller [3]. In this case, m-step ahead prediction error minimization with m equal to the envisaged prediction horizon of the controller matches the final goal closely.

A difficulty in the implementation of m-step-ahead prediction-error minimization is the need to start simulations at different time instants along the full training trajectory. For state-space models, in particular, we need to assign an *initial state* to all the sub-sequences considered during training. Thus, in [6], the full state trajectory is considered as an optimization variable to be estimated along with the parameters of the model dynamics. In [1], an *encoder* network is used as a deadbeat state estimator, and its parameters are trained together with the ones of the model to minimize the m-step-ahead prediction error. The work may be seen as an extension (to the m-step-ahead case) of the one-step method previously described in [14].

In the above-mentioned papers, neural state-space model learning approaches are introduced and tested on a limited number of configurations. The contributions do not include extensive experimentations exploring the algorithm hyper-parameters in a systematic fashion, nor a statistical analysis of their main effects and interactions.

The aim of this paper is to shed light on the role of different user choices and algorithm settings for the training of state-space neural models through m-step-ahead simulation error minimization, with a particular focus on the initial state estimation strategy. Compared to the previous contributions, we also report the performance obtained with *dummy* estimation strategies such as *random* and *zero* state initialization, to be considered as baselines. Interestingly, these two baselines are shown to be competitive on the Wiener-Hammerstein benchmark [12] considered for neural state-space identification in [1]. We then perform another extensive analysis involving the pick-and-place machine benchmark [10], which (to the best of our knowledge) has not been previously considered to asses neural state-space models. On the pick-and-place benchmark, we show that more advanced strategies for initial state estimation are really required to attain high fitting performance. With these strategies, we have obtained the highest model fit on this benchmark w.r.t. the ones reported in the literature thus far.

Finally, we analyse and infer from the numerous results obtained on the considered benchmarks taking into account properties of the underlying systems, and provide guidelines for system identification practitioners.

2 Settings

2.1 Notation

Consider an n-length sequence of variables $z = \{z_0, z_1, \dots, z_{n-1}\}$, where each element z_i is either scalar- or vector-valued. We adopt (python-like) *slicing* notation to address a portion of the sequence, *i.e.*, for integers h, k, with $h \leq k \leq n-1$, $z_{h:k} = \{z_h, z_{h+1}, \dots, z_{k-1}\}$ denotes the sub-sequence with start index h (included) and stop index k (excluded).

2.2 Dataset and model structure

We are given a dataset $\mathcal{D} = (u_{0:n}, y_{0:n})$ containing n input samples $u_i \in R^{n_u}$ and (possibly noisy) output samples $y_i \in R^{n_y}$, collected from a dynamical data-generating system $\mathcal{S}$. Our goal is to estimate a model M of the unknown dynamics of $\mathcal{S}$ using the dataset $\mathcal{D}$.

We consider the following model structure:

$$x_{k+1} = \mathcal{N}_f(x_k, u_k; \theta) \quad (1a)$$
$$y_k = \mathcal{N}_g(x_k; \theta), \quad (1b)$$

where $\mathcal{N}_f$ and $\mathcal{N}_g$ are feed-forward neural networks having compatible dimensions, $x_k \in \mathbb{R}^{n_x}$ is the state at time k and $\theta \in \mathbb{R}^{n_\theta}$ is a vector of parameters to be identified from data. Overall, (1) defines a neural state-space model structure, often referred to as *recurrent neural network*.

2.3 Full simulation error minimization

To estimate the model parameters θ, a straightforward approach is to minimize the simulation error norm with respect to both θ and the initial state x_0:

$$\hat{\theta}, \hat{x}_0 = \arg \min_{\theta, x_0} \sum_{k=0}^{n-1} \left\| y_k - y_k^{\text{sim}} \right\|^2, \tag{2}$$

where y_k^{sim} is the output obtained by iterating (thus, simulating) the model dynamics (1) over n time steps. From a theoretical perspective, minimization of (2) corresponds, in the case of additive white Gaussian noise on the measured outputs y_k, to the *maximum likelihood* estimation of θ and x_0. Thus, in principle, the methodology enjoys desirable properties, *e.g.*, consistency and asymptotic optimality [8]. However, from a practical perspective, the method suffers from two severe limitations. First, repeated (forward and backward) evaluation of the full simulation error minimization is often computationally heavy for long training sequences, and it offers limited opportunities for parallelization due to the *sequential* nature of the simulation over time [6]. Second, the simulation error norm may be numerically hard to optimize (*i.e.*, the conditioning of the optimization problem is often poor), notably in the case of unstable or chaotic underlying dynamics [23].

2.4 Truncated simulation error minimization

To overcome the above limitations, it is possible to minimize the simulation error over shorter sequences extracted from the dataset. Considering all possible m-length contiguous sequences from $\mathcal{D}$, the optimization problem becomes:

$$\hat{\theta}, \hat{x}_{0:n-m} = \arg \min_{\theta, x_{0:n-m}} \sum_{i=0}^{n-m-1} \overbrace{\sum_{j=0}^{m-1} \left\| y_{i+j} - y_{i+j|i}^{\text{sim}} \right\|^2}^{=J_i}, \tag{3}$$

where $y_{i+j|i}^{\text{sim}}$ is the simulated output at time step $i+j$ obtained by initializing the simulation at time i with state x_i. Computation of the loss (3) can be easily parallelized for the $(n-m)m$ different m-length sequences as the terms J_i do not depend on each other and can be processed independently (possibly exploiting parallel hardware). However, the formulation requires the inclusion of an optimization variable $x_{0:n-m}$ representing the state sequence over $n-m$ time steps. The larger number of optimization variables may then lead to an increased variance of the identified parameters $\hat{\theta}$, and thus make the algorithm less sample-efficient.

To counter the variance increase, an effective regularization approach enforcing *compatibility* between the estimated state sequence $\hat{x}_{0:n-m}$ and the dynamics (1) was introduced in [6]. However, the approach in [6] requires careful tuning of the trade-off loss (fitting vs. regularization), and it is not further discussed here.

2.5 State estimator

Instead of treating the full state sequence as an optimization variable, it is possible to learn a *state estimator* that maps previous input/output data samples to the current (estimated) state. We consider, in particular, *deadbeat* estimators which generate an estimate $\hat{x}_i$ of the state at time i from a *finite* window of past input/output measurements of length m_e:

$$x_i = \mathcal{N}_e(u_{i-m_e:i}, y_{i-m_e:i}; \phi). \tag{4}$$

Exploiting such a deadbeat structure, it is still possible to define a multi-step cost function that can be split into independent contributions for the sequences:

$$\hat{\theta}, \hat{\phi} = \arg \min_{\theta, \phi} \sum_{i=0}^{n-m-1} \sum_{j=m_e}^{m-1} \left\| y_{i+j} - y_{i+j|i}^{\text{sim}} \right\|^2 \tag{5a}$$

$$\hat{x}_{i+m_e} = \mathcal{N}_e(u_{i:i+m_e}, y_{i:i+m_e}; \phi), \tag{5b}$$

with $m = m_e + m_f$. Note that the m-length training sequence starting at time index i may be seen as divided into an initial m_e-lenght portion used just to reconstruct the state at time $i + m_e$, and a trailing m_f-length portion used to compute the fitting loss.

The strategy above may be more attractive than an explicit estimation of the whole state sequence $x_{0:n-m}$ (as done in (3)) when the training dataset is relatively large and the dimension of the parameter vector ϕ required to describe the estimator is (expected to be) smaller than the sequence $x_{0:n-m}$ itself.

Remark 1 *In [26][**Lemma 1**], exact state reconstruction is shown to be possible (in the noise-free case) by choosing $m_e \geq n_x$. Specifically, the Lemma states that: if the data-generating system is of the form (1), and it is strongly locally observable, then under mild conditions, for all $m_e \geq n_x$ there exists a smooth function $\gamma : \mathcal{U} \times \mathcal{Y} \rightarrow \mathcal{X}$, where $\mathcal{U} \subset \mathbb{R}^{n_u m_e}, \mathcal{Y} \subset \mathbb{R}^{n_y m_e}, \mathcal{X} \subset \mathbb{R}^{n_x}$, such that the state at time step i is given by $x_i = \gamma(u_{i-m_e:i}, y_{i-m_e:i})$. In this sense, the estimator $x_i = \mathcal{N}_e(u_{i-m_e:i}, y_{i-m_e:i}; \phi)$ given in (4) can be seen as a machine-learning surrogate of the smooth function γ. Classic universal approximation theorems for neural networks [5] guarantee that, for a sufficiently complex architecture, the approximation error of $\mathcal{N}_e$ can be made arbitrarily small.*

Remark 2 *In this paper, we restrict ourselves to* causal *estimators producing the state estimate $\hat{x}_i$ at time i based on* previous *input/output samples $u_{i-m_e:i}, y_{i-m_e:i}$. In principle, current and future input/output samples could also be considered to construct the state estimate. We leave analysis and experimentation with non-causal state reconstruction techniques as future work.*

2.5.1 State estimator structure

In principle, the state estimator $\mathcal{N}_e$ could be designed using traditional techniques from the control literature [25] (*e.g.*, Luenberger, Kalman, moving horizon *etc.*). However, as argued in [1], standard feed-forward (FF) neural networks (which are easily implemented and integrated in deep-learning software frameworks) may be applied for this task as well.

Building upon [1], in this paper we also explore the case where $\mathcal{N}_e$ is parametrized as a long-short-term memory (LSTM) network processing the m_e input/output samples sequentially, and we provide comparative results with the FF case.

Furthermore, we include in our experiments two "dummy" estimators called ZERO and RAND, to be used as baseline for the most advanced ones (FF, LSTM). The ZERO estimator uses the m_e input/output estimation samples to start a simulation with the initial state set equal to 0. The simulated state at the last step of the estimation sequence is then used as initial state for fitting. Similarly, the RAND estimator initializes the simulation in the estimation sequence with a random value extracted from a standard Gaussian distribution and generates the initial state for the optimization in fitting sequence as the state obtained at the last step of the estimation horizon.

2.6 Implementation aspects

We briefly state here implementation aspects followed in this paper. The presentation is rather concise as they are in line with standard "good practice" for deep learning, see [19].

Optimization of the loss (5) is performed by stochastic gradient descent over (mini)batches of sequences extracted from the training dataset. At each iteration of the optimization loop, the following minibatch loss is computed:

$$\tilde{J}(\theta, \phi) = \frac{1}{bm} \sum_{s=0}^{b-1} \sum_{j=m_e}^{m-1} \left\| y_{i_s+j} - y^{\text{sim}}_{i_s+j|i_s} \right\|^2 \tag{6a}$$

$$\hat{x}_{i_s+m_e} = \mathcal{N}_e(u_{i_s:i_s+m_e}, y_{i_s:i_s+m_e}; \phi), \tag{6b}$$

where the batch size b is the number of sequences included in the batch and $i_s \in \{0, \dots, n - m - 1\}$ is a random integer defining the starting index of the sequence s, for $s = 0, 1, \dots, b - 1$.

The gradients of (6a) with respect to the optimization variables θ and ϕ are computed with reverse-mode automatic differentiation, as implemented in standard deep learning software [21] and used for iterative gradient-based optimization. For numerical optimization, the Adam algorithm [11] is used, since it is generally recognized as more iteration-efficient than plain gradient descent, while not introducing significant additional computational burden.

A 20% portion of the training data is used as hold-out validation dataset. For each configuration, the model having the lowest validation loss over the optimization iterations is saved.

Instead of a fixed number of iterations, we run the training algorithm for a fixed amount of *time*. This allows for a fair comparison among algorithm configurations characterized by widely varying computational cost per iteration.

3 Experiments

The methodologies described in the paper are evaluated on two standard benchmarks for system identification, namely the Wiener-Hammerstein circuit [12] and the pick-and-place machine [10]. For the two examples, extensive experi-

Factor	Values
Estimator type (`est_type`)	FF, LSTM, ZERO RAND
Training time (s) (`max_time`)	300, 1800, 3600
Size b of training (mini)batch (`batch_size`)	32, 128, 512, 1032
Fitting seq. length (`seq_fit_len`)	40, 80, 160, 320
Estimation seq. length m_e (`seq_est_len`)	10, 20, 40, 80

Table 1: WH circuit: Parameter configuration of the experiments.

mental campaigns exploring the effect of different training algorithm settings on the obtained model's performance are carried out.

The trained models are then evaluated in terms of their FIT index on a test dataset:

$$\text{FIT} = 100 \times \left(1 - \frac{\sqrt{\sum_{t=0}^{n-1}\left(y_t - y_t^{\text{sim}}\right)^2}}{\sqrt{\sum_{t=0}^{n-1}\left(y_t - \overline{y}\right)^2}}\right)(\%), \tag{7}$$

where y is the measured output; y^{sim} is the model's simulated output; and $\overline{y}$ is the mean value of y, i.e. $\overline{y} = \frac{1}{n}\sum_{t=0}^{n-1} y_t$. We then inspect the results of our experimental campaigns using standard tools for statistical analysis [20], taking the settings of the training algorithm as controlled *factors* and the model's FIT index in the test dataset as measured *response*.

All the codes required to reproduce our results are available in the GitHub repository `https://github.com/forgi86/sysid-neural-estimator`. The user may also extend the analysis to additional factors (and new benchmarks) with minor modifications to the provided software.

Computations are performed on a deep-learning server of the IDSIA laboratory equipped with 2 64-core AMD EPYC 7742 Processors, 256 GB of RAM, and 4 Nvidia RTX 3090 GPUs. In all the experiments, the hardware resources of the server have been limited to 10 CPU threads and 1 GPU.

3.1 Wiener-Hammerstein circuit

This benchmark was introduced in [12] and it contains real measurement from an electronic circuit which implements a Wiener-Hammerstein (WH) dynamics. Input and output samples are collected at a constant frequency of 51200 Hz. The training and test datasets contain 100000 and 87000 samples, respectively.

In line with [1], the number of states of the model (1) is set to $n_x = 6$ and the neural networks $\mathcal{N}_f$ and $\mathcal{N}_g$ defining the state update and output mapping are chosen having a single hidden layer with 15 nodes and $\tanh$ activation function, plus a direct linear term from input to output. We consider in our experiments the cases of state estimators $\mathcal{N}_e$ with FF, LSTM, ZERO, and RAND structure (see Section 2.5.1). For the FF estimator, we use a single-hidden-layer neural architecture with 15 nodes and a direct input/output linear term. For the LSTM estimator, we employ a standard implementation (as available in PyTorch) with a single hidden layer and 16 nodes. The LSTM processes the samples in the estimation sequence forward in time, and its output at the last time step defines the initial state for fitting. Finally, the learning rate of Adam is fixed to $1 \cdot 10^{-3}$.

The factors explored in the experimental campaign are reported in Table 1, together with their considered levels. The estimator type (est_type), estimation sequence length m_e (seq_est_len) and fitting sequence length m_f (seq_fit_len) are included, as well as the batch size b (batch_size) and the total training time (train_time). A *full factorial* experimentation including all combinations of the algorithm settings has been performed, totaling 768 training runs which required approximately 17 days.[1]

The top-5 configurations in terms of FIT index over the test dataset are reported in Table 2. All these configurations are characterized by the longest training time (train_time=3600 s). Interestingly, the best result is obtained with an LSTM estimator, and the 4 other instances in the top-5 list are based on dummy (ZERO/RAND) estimators.

The measured output y, the output y^{sim} simulated from the best obtained model, and the error signal $y - y^{\text{sim}}$ are shown in Fig. 1. For better visualization, only a short portion of the test dataset is displayed.

[1]Note that the total experimental time is determined *a priori* as we execute the training algorithm for a fixed amount of time, and not for a fixed number of iterations.

est_type	batch_size	seq_fit_len	seq_est_len	FIT
LSTM	128	80	40	98.96 %
RAND	512	80	40	98.89 %
ZERO	512	40	80	98.87 %
RAND	128	40	40	98.82 %
RAND	1024	80	80	98.81 %

Table 2: WH circuit: Top-5 configurations. For all the configurations, train_time=3600 s.

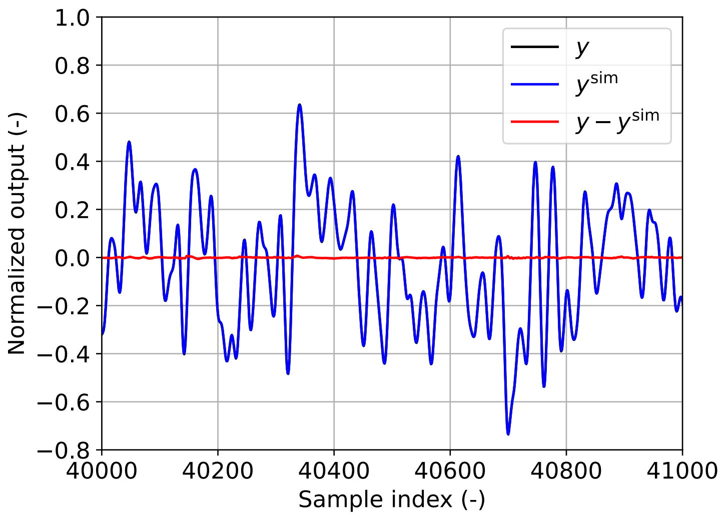


Figure 1: WH circuit: Measured output y, simulated output y^{sim}, and error signal $y - y^{\mathrm{sim}}$ on a portion of the test dataset.

In order to assess the intrinsic variability of the training algorithm, 100 repetitions of the best configuration with different random initializations of the model (and estimator) parameters have been executed.[2] The histogram of the FIT index achieved over the test dataset by the 100 trained models is shown in Fig. 2. The empirical mean and standard deviation of the 100 FIT values are 98.44% and 0.32%, respectively. In this sense, FIT index differences larger than $3 \times 0.32\% \approx 1.0\%$ between different configurations are expected to be statistically significant.

In Fig. 3, the main effects of the factors are illustrated. For each level of a factor, the graph shows the average (w.r.t. all the other factor values) of the FIT index as a round dot, and a 95% confidence interval as a vertical band.

The training time has by far the largest impact and it has a positive effect on the outcome, in line with our expectation. The fitting sequence length m_f has also a significant effect, with a negative impact for the largest considered value $m_f = 320$. The influence of the state estimator choice (est_type) is not visible at this level.

In Fig. 4, the interactions between the training time (which is the dominant factor of the main effect analysis) and all the other factors are highlighted. From the figure, it is evident that the negative effect of the longest fitting sequence length (seq_fit_len $m_f = 320$) is particularly strong in conjunction with the shortest training time (train_time=300 s), and it is almost negligible in the case of the largest training time (train_time=3600 s). Thus, the lower performance is likely related to the increasing computational cost per optimization iteration, which in turn resulted in a lower number total iterations executed in the fixed time budget.

In this benchmark, the effect of the state estimator is only visible by restricting the analysis to the shortest value of seq_fit_len $m_f = 40$ and a longest training time (train_time=3600 s). The main effects of the remaining factors are visualized in Fig. 5, where it appears that higher average performance with lower variance is associated with the experiments where LSTM or FF state estimators are used. Remarkably, a significantly low average FIT and high variability is also associated with the shortest estimation sequence length $m_e = 10$.

[2]Parameter initialization is the main source of non-determinism in the training algorithm.

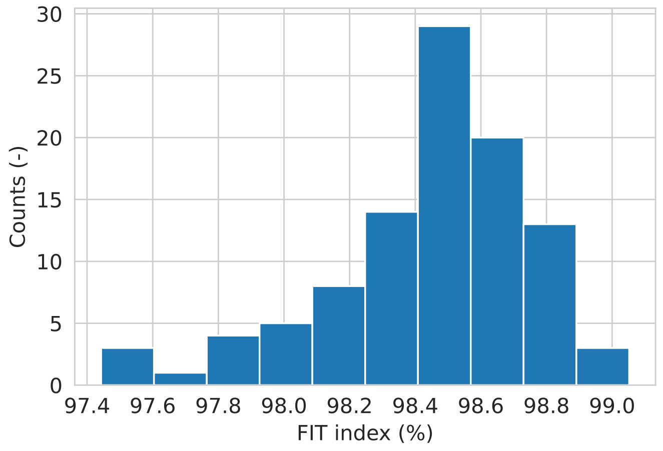


Figure 2: Histogram of the test FIT index achieved by repeating the best configuration 100 times.

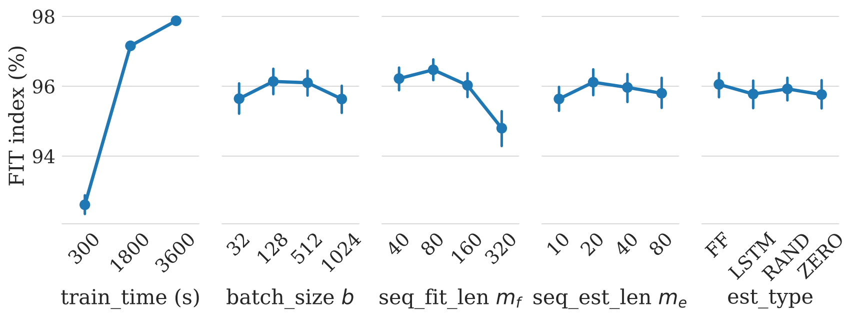


Figure 3: WH circuit: Main effects (and 95% confidence intervals) of different factors.

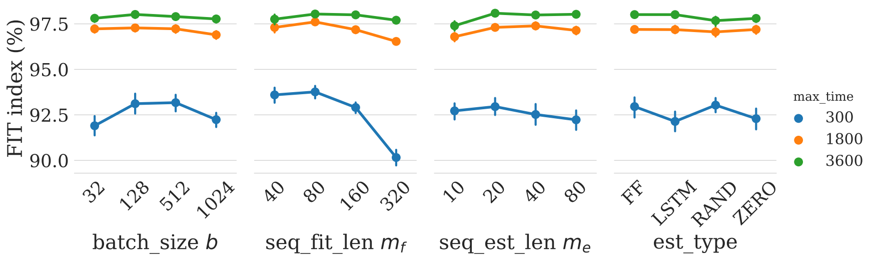


Figure 4: WH circuit: Interactions between train_time and other factors.

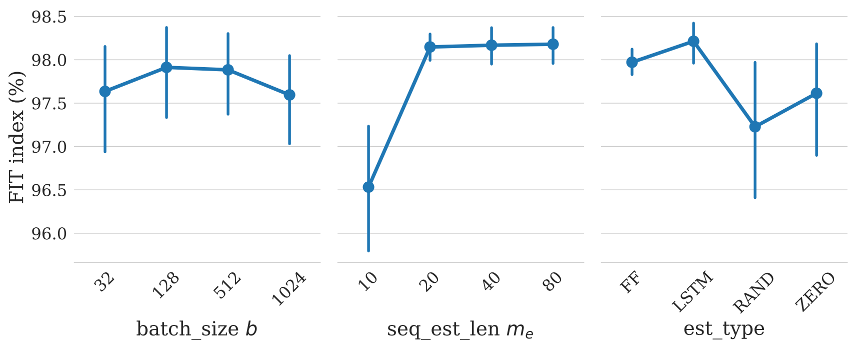


Figure 5: WH circuit: Main effects of factors restricted to train_time=3600 s, seq_fit_len $m_f = 40$.

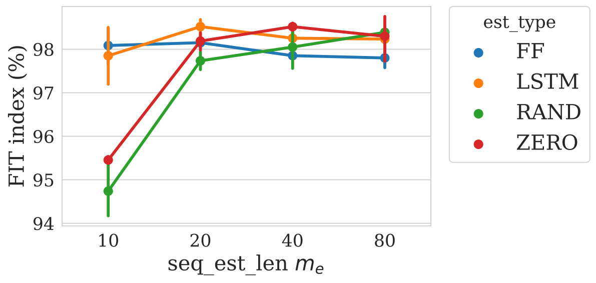


Figure 6: WH circuit: Interaction between factors seq_est_len and est_type for train_time=3600 s and seq_fit_len $m_f = 40$.

Further insight is obtained by looking at the interaction between est_type and seq_fit_len in the same restricted set of configurations (Figure 6). From this figure, it is clear that a significantly lower performance is associated with experiments where the estimation sequence is short (seq_est_len $m_e = 10$) and a dummy estimator (ZERO/RAND) is used.

These observed results are compatible with the following reasoning: the Wiener-Hammerstein benchmark has an asymptotically stable dynamics, and state estimation may be performed simply by simulating an accurate model for a sufficiently long time horizon, starting from *any* initial condition (including zero and random values). Only for very short values of m_e (w.r.t. the system dynamic's time constant), the dummy estimation methods ZERO and RAND are not sufficiently accurate, resulting in a lower overall fitting performance. On the other hand, more powerful state estimation techniques FF and LSTM are able to reconstruct the initial state even from a shorter sequence.

The situation is highlighted in Fig. 7, where the behavior of the models obtained in two similar configurations (train_time=3600 s, batch_size=$b = 128$, seq_est_len $m_e = 10$, seq_fit_len $m_f = 40$) and only differing in the estimator type (LSTM *vs* ZERO) is reported for three random sequences extracted from the training dataset. In the LSTM case, the simulated output is already close to the target at time step 10, suggesting that this estimator is indeed effectively reconstructing the state at that time step. Conversely, the ZERO estimator (which is equivalent to a simulation initialized at time step 0 with state equal to 0) seems to require approximately 40 steps to converge to the true value, thus limiting the effectiveness of the overall approach. Incidentally, the plots also clarifies why, for configurations with seq_est_len $m_e \geq 40$, the fitting performance does not seem to be affected by the choice of the state estimator in a significant way. Furthermore, for longer values of seq_fit_len m_f (excluded in this specific analysis), the negative effect of bad state initialization is diluted over more fitting samples, and its impact on the model performance becomes negligible.

Remark 3 *For smaller values of seq_est_len $m_e < 10$, the performance of the LSTM and FF state estimators may eventually decrease. From a theoretical perspective, as already stated in Remark 1, even in a noise-free context a minimum number of samples is required for state estimation. Specifically, a sufficient condition for exact state reconstructability is $m_e \geq n_x$ [26]. The situation is more complex in a noisy scenario, and a detailed analysis is left as future work.*

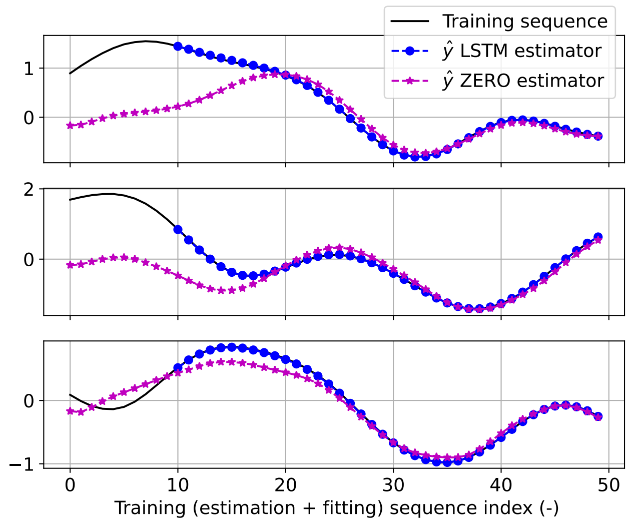


Figure 7: WH circuit: Training sequence (black) vs. model prediction obtained with est_type=LSTM (blue), est_type=ZERO (magenta). The other factor are set to train_time=3600 s, batch_size=128, seq_est_len=10, seq_len=40.

Factor	Values
Estimator type (`est_type`)	FF, LSTM, ZERO, RAND
Training time (sec) (`max_time`)	300, 1800
Length b of (mini)batch in training (`batch_size`)	32, 128, 1032
Length m_f of fitting sequence (`seq_fit_len`)	64, 256, 512
Length m_e of past I/O window (`seq_est_len`)	10, 40, 100
# of hidden nodes of $\mathcal{N}_e$ (`est_hidden_size`)	10, 30

Table 3: Pick-and-place machine: Parameter configuration of the experiments.

3.2 Pick-and-place machine

In this case study, we consider data-driven modeling of an electronic component placement process in a pick-and-place machine. The experimental setup consists of a mounting head which holds an electronic component [10]. The component is pushed down to an impact surface simulating the printed circuit board and then it is released. The system is characterized by two main operating modes: *free* mode and *impact* mode. In the free mode, the mounting head moves the component in an unconstrained environment without any contact with the impacting surface, while in the impact mode, the mounting head is in contact with the impacting surface. This setup is considered as a benchmark for assessing the effectiveness of *hybrid* system identification approaches, see [2, 22].

An input-output dataset over a time interval of 15 s with sampling frequency of 400 Hz is gathered. The training data consists of 5000 samples taken during the first 12.5 seconds of the experiment and the test dataset consists of 1000 samples gathered in the last 2.5 seconds.

We carry out an extensive experimental campaign in order to study the effects of the estimators as well as different settings of the training algorithm on the identified model quality, quantified in terms of the FIT index (7) on the test dataset.

We consider the state-space neural model structure (1) with order set to $n_x = 2$. The neural networks $\mathcal{N}_f$ and $\mathcal{N}_g$ defining the state update and output are characterized by a single hidden layer with 15 nodes and $\tanh$ activation function. The networks are trained by employing *Adam* algorithm with a learning rate set to $1 \cdot 10^{-3}$.

We run the training algorithm for all the possible combinations of factors reported in Table 3 (full factorial experimentation) and thus we obtain 432 models, each one corresponding to a different unique configuration. Execution of all the training runs required approximately 5 days.

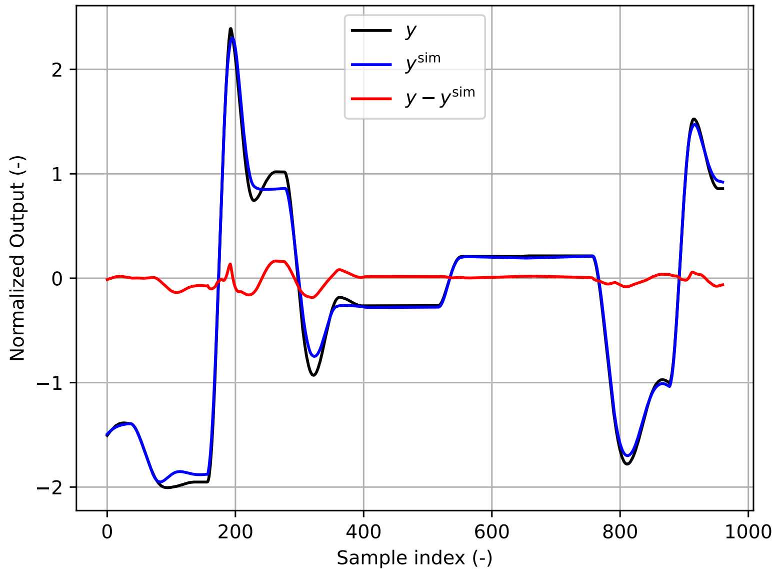


Figure 8: Pick-and-place machine: Measured output y, simulated output y^{sim}, and error signal $y - y^{\text{sim}}$ on the test dataset.

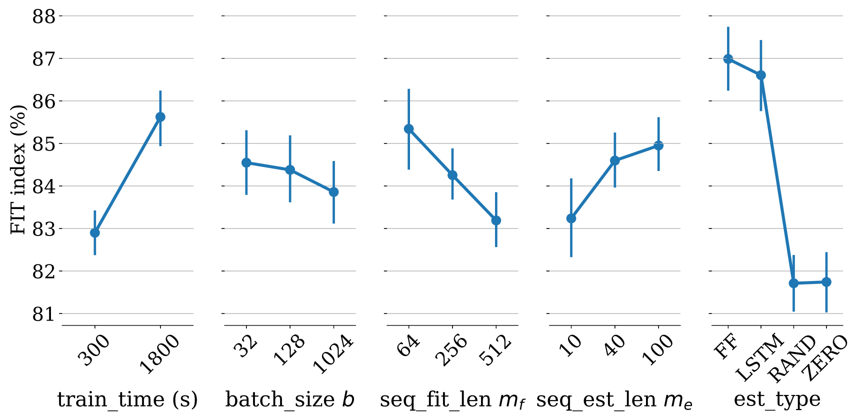


Figure 9: Pick-and-place machine: Effect of different factors on FIT index, Mean (round dot) and 95% confidence intervals (vertical bars).

The top-5 configurations leading to the models with highest FIT index are reported in Table 4. It is interesting to note that, in contrast with the WH case study, all the top-5 configurations have non-dummy state estimators FF, LSTM. To the best of our knowledge, the FIT indices reported in Table 4 are the highest achieved among the ones reported in the literature, see [2, 22, 16, 18, 17] for this benchmark. The measured output y, the output y^{sim} simulated from the best obtained model, and the error signal $y - y^{\text{sim}}$ are shown in Fig. 8.

In Fig. 9, we analyse the effect of different factors on the achieved model performance.[3]

The round dot represents the mean value of the FIT index[4] and the vertical bars represent 95% confidence intervals. In the leftmost subplot, we see that a larger training time (1800 s) gives better FIT on average w.r.t. a shorter training of 300 s, as a consequence of the higher number of iterations allowed for convergence. The (mini)batch size b has a slight negative effect on the FIT index, as seen in the second subplot. A longer fitting sequence length $m_f = 512$

[3]We omit to report results for the factor est_hidden_size, as its effect was found to be negligible.

[4]For each value of a factor, the mean FIT is computed by taking the average over all the other factor values.

est_type	batch_size	seq_fit_len	seq_est_len	FIT
FF	128	64	40	93.61 %
LSTM	32	64	100	93.40 %
FF	32	64	40	92.94 %
LSTM	128	64	40	92.79 %
LSTM	128	64	10	92.51 %

Table 4: Pick-and-place machine: Top-5 configurations. For all configurations, train_time = 1800 s.

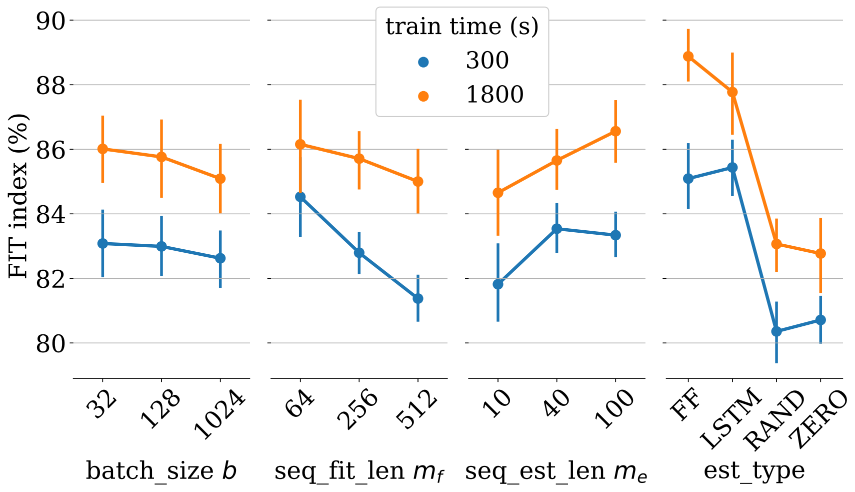


Figure 10: Pick-and-place machine: Effect of training time on FIT index.

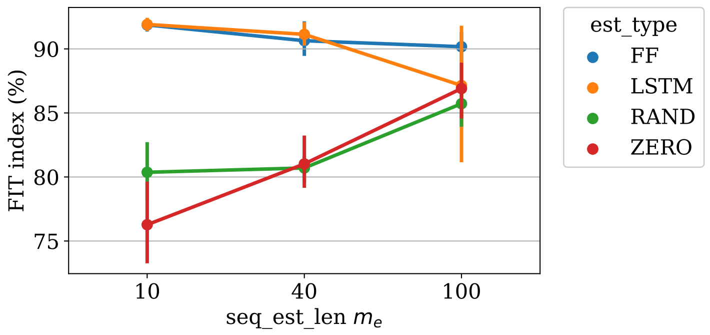
leads to worse performance, while a longer estimation sequence length m_e improves the FIT on average, as seen from subplots 3 and 4 respectively. We remark that this simple evaluation of the effects of m_f and m_e on the FIT is not complete at this stage, as it is influenced by factors such as short training time, estimator type, etc. The effects of m_f and m_e will be analyzed in more details w.r.t. various factors in the subsequent paragraphs.

Finally, the effect of the state estimators on model quality can be seen in the rightmost subplot. This plot clearly shows that models having an estimator for the initial state (either FF or LSTM) significantly outperforms the models with ZERO or RAND initializations.

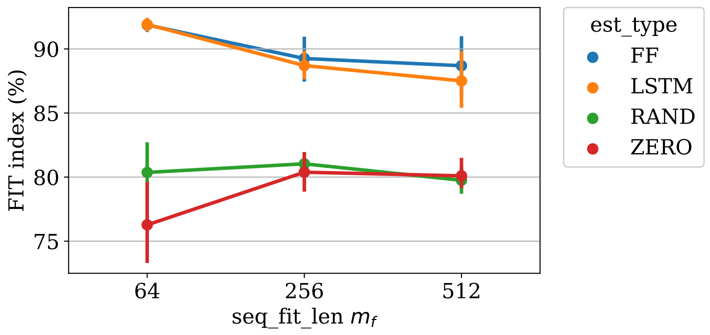
Remark 4 *The significant improvement in the FIT index with FF and LSTM estimators is compatible with the following rationale: We conjecture that for the pick-and-place machine dataset, the state estimators (FF, LSTM) are required due to the presence of an integrator in the dynamics of underlying mechanical system. The estimation error in the initial state may get integrated over time, resulting in a worse performance for models with RAND or ZERO initialization. On the other hand, FF and LSTM are able to provide an accurate estimate of the initial state.*

In Fig. 10, the effect of the training time on FIT index is depicted. Naturally, a longer training time of 1800 s implies a larger number of iterations allowing the algorithm to converge towards a better solution, which results in an improved model quality *vs* models trained for a shorter interval of 300 s. As discussed in the WH case study, for the short training time of 300 s, it is interesting to note the trend of the FIT w.r.t. the fitting sequence length m_f (blue curve shown in subplot 2). It can be seen that the model accuracy decreases fairly uniformly as m_f is increased when models are trained for a short time interval. This phenomenon can be explained as follows: for a long fitting sequence, the training algorithm requires more time/iterations for convergence towards an optimum, thus pre-termination with a short training time results in models having generally lower performance. For a longer training time of 1800 s, this effect is not prominent. From Fig. 10, it is also interesting to note that the average performance is improved with increase in the length of the estimation sequence m_e as seen in the third subplot from the left.

Next, we focus on the effect of the factors seq_fit_len m_f and seq_est_len m_e. In this analysis, we consider the larger training time of 1800 s, omitting the results obtained with a short training interval of 300 s. In Fig. 11, we analyse the effect of short sequence lengths m_e, m_f on the FIT index for different estimators types. In particular, in Fig. 11a, we plot the FIT index *vs* m_e with the seq_fit_len $m_f = 64$, while in Fig. 11b, the FIT index is plotted against m_f for a



(a) Effect of m_e for fixed fitting sequence length $m_f = 64$.



(b) Effect of m_f for fixed fitting sequence length $m_e = 10$.

Figure 11: Pick-and-place machine: Effect of m_e and m_f on FIT index with different estimator types.

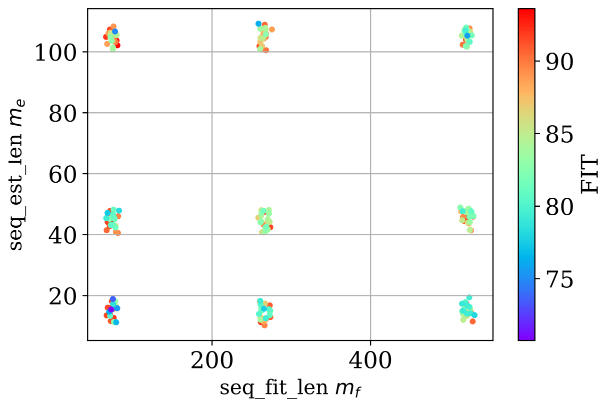


Figure 12: Pick-and-place machine: Effect of seq_est_len and seq_fit_len on FIT index for training time of 1800 s.

short estimation length of $m_e = 10$. We observe that even for short estimation and fitting sequence lengths m_e, m_f, the estimators FF, LSTM result in a high FIT index. On the other hand, the FIT obtained with the dummy estimators ZERO, RAND is worse for short sequence lengths, with a high variability. Nonetheless, it is interesting to note that for the dummy estimators ZERO, RAND, the FIT is improved for increasing values of m_e.

In Fig. 12, we show the overall effect of different configurations of m_e, m_f on the FIT index with different estimators. For the sake of visualization, the scatter plot is obtained by slightly perturbing m_e, m_f (i.e., by adding some jitter to their numerical values on the grid, even though each cluster actually represents the same configuration of m_e, m_f having integer values). Similar to the previous analysis, we observe that the lowest values of m_e, m_f (i.e., lower left cluster with $m_e = 10, m_f = 64$) have the lowest average FIT index.

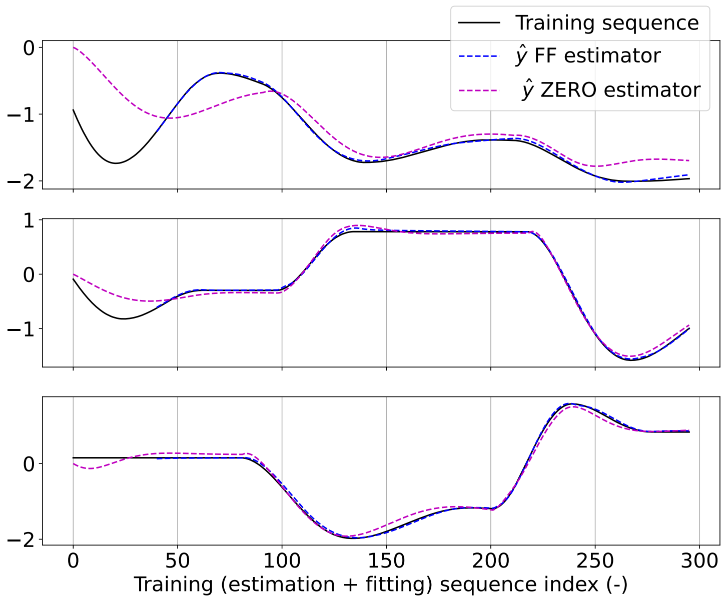


Figure 13: Pick-and-place machine: Training sequence (black) *vs* model prediction obtained with est_type=FF (blue), est_type=ZERO (magenta). The other factor are set to train_time=1800 s, batch_size=128, seq_est_len=40, seq_len=256.

Finally, we highlight the effect of ZERO *vs* FF estimator on the model quality by visualizing the simulated outputs obtained for a specific training configuration. In Fig. 13, we plot 3 different training sequences, along with the simulated output trajectories obtained with ZERO and FF estimator models, where other factors are fixed to $b = 128, m_e = 40, m_f = 256$. It is evident that the simulated trajectories obtained with the FF estimator follow the true sequence closely from the time step 40, viz. the sequence length used to estimate the initial state. This suggests that the initial condition has been correctly reconstructed by the FF estimator. On the other hand, we observe that the simulated outputs with ZERO state initialization at time step 0 requires about 100 time steps to match closely with the true sequence, while still giving an error in the estimation at subsequent time samples. In particular, for the sequence in the first subplot which is further away from the zero initial condition, the mismatch between the trajectory simulated with ZERO initialization and the true sequence is large. This further highlights the need for an estimator for the pick-and-place benchmark example.

Overall, the results obtained on the pick-and-place example demonstrate the need of a state estimator in order to significantly improve the model quality. In our assessment, this need is linked to the presence of an integrator in the underlying dynamics of the considered mechanical system. We conjecture that the overall lower FIT index obtained with ZERO and RAND initialization is a consequence of the integration of the initial state estimation error over time.

4 Conclusion

We have presented algorithms for neural state-space model learning and provided a detailed descriptions of their salient hyper-parameters and settings. Through extensive experimentation and analyses of the obtained results, we have extracted insight about the effects and interactions of several choices available to the system identification practitioner, focusing in particular on the state initialization procedure.

The developed software has been made publicly available not only to allow reproducibility of the outcomes, but also to encourage other researchers to extend the analyses to additional factors, levels, and benchmarks. Possible investigation include: use of non-causal state reconstruction approaches; estimator design with formal strategies from control theory (e.g., Kalman, Luenberger, moving horizon); experimentation with unstable and chaotic dynamics.

We expect that similar studies could shed light on yet unknown properties of the existing algorithms, and possibly lead to improved and more efficient variations thereof.

References

[1] G. Beintema, R. Tóth, and M. Schoukens. Nonlinear state-space identification using deep encoder networks. In *Learning for Dynamics and Control*, pages 241–250. PMLR, 2021.

[2] A. Bemporad, A. Garulli, S Paoletti, and A. Vicino. A bounded-error approach to piecewise affine system identification. *IEEE Transactions on Automatic Control*, 50(10):1567–1580, 2005.

[3] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[4] S. Chen, S. A. Billings, and P. M. Grant. Non-linear system identification using neural networks. *International journal of control*, 51(6):1191–1214, 1990.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[6] M. Forgione and D. Piga. Model structures and fitting criteria for system identification with neural networks. In *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–6. IEEE, 2020.

[7] M. Forgione and D. Piga. *dynoNet*: A neural network architecture for learning dynamical systems. *International Journal of Adaptive Control and Signal Processing*, 35(4):612–626, 2021.

[8] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[9] L. Iacob, G. Beintema, M. Schoukens, and R. Tóth. Deep Identification of Nonlinear Systems in Koopman Form. *arXiv preprint arXiv:2110.02583*, 2021.

[10] A.Lj Juloski, W.P.M.H Heemels, and G Ferrari-Trecate. Data-based hybrid modelling of the component placement process in pick-and-place machines. *Control Engineering Practice*, 12(10):1241–1252, 2004.

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

[12] L. Ljung, J. Schoukens, and J. Suykens. Wiener-Hammerstein benchmark. In *15th IFAC Symposium on System Identification, Saint-Malo, France, July, 2009*, 2009.

[13] B. Lusch, J. N. Kutz, and S. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

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

[15] D. Masti and A. Bemporad. Learning nonlinear state–space models using autoencoders. *Automatica*, 129:109666, 2021.

[16] P. Mattsson, D. Zachariah, and P. Stoica. Recursive nonlinear-system identification using latent variables. *Automatica*, 93:343–351, 2018.

[17] M. Mazzoleni, A. Chiuso, M. Scandella, S. Formentin, and F. Previdi. Kernel-based system identification with manifold regularization: A bayesian perspective. *Automatica*, 142:110419, 2022.

[18] M. Mejari, V.V. Naik, D. Piga, and A. Bemporad. Identification of hybrid and linear parameter-varying models via piecewise affine regression using mixed integer programming. *International Journal of Robust Nonlinear Control*, 30:5802–5819, 2020.

[19] U. Michelucci. Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks. 2018.

[20] D. C. Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[22] D. Piga, A. Bemporad, and A. Benavoli. Rao-Blackwellized sampling for batch and recursive Bayesian inference of Piecewise Affine models. *Automatica*, 117:109002, 2020.

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

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

[25] D. Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[26] V. Verdult, J. A. K. Suykens, J. Boets, I. Goethals, and B. De Moor. Least squares support vector machines for kernel cca in nonlinear state-space identification. In *Proceedings of the 16th international symposium on Mathematical Theory of Networks and Systems (MTNS2004), Leuven, Belgium*. Citeseer, 2004.