Manifold meta-learning for reduced-complexity neural system identification

Marco Forgione[1], Ankush Chakrabarty[2], Dario Piga[1], Matteo Rufolo[1], and Alberto Bemporad[3]

[1]SUPSI, IDSIA - Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland.
[2]Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA.
[3]IMT School for Advanced Studies Lucca, Lucca, Italy.

April 16, 2025

Abstract

System identification has greatly benefited from deep learning techniques, particularly for modeling complex, nonlinear dynamical systems with partially unknown physics where traditional approaches may not be feasible. However, deep learning models often require large datasets and significant computational resources at training and inference due to their high-dimensional parameterizations. To address this challenge, we propose a meta-learning framework that discovers a low-dimensional manifold within the parameter space of an over-parameterized neural network architecture. This manifold is learned from a meta-dataset of input-output sequences generated by a class of related dynamical systems, enabling efficient model training while preserving the network's expressive power for the considered system class. Unlike bilevel meta-learning approaches, our method employs an auxiliary neural network to map datasets directly onto the learned manifold, eliminating the need for costly second-order gradient computations during meta-training and reducing the number of first-order updates required in inference, which could be expensive for large models. We validate our approach on a family of Bouc-Wen oscillators, which is a well-studied nonlinear system identification benchmark. We demonstrate that we are able to learn accurate models even in small-data scenarios.

1 Introduction

In recent years, system identification has advanced significantly with the adoption of deep learning tools and techniques [3,8,9,14]. These methods have proven particularly effective in modeling complex, nonlinear dynamical systems where the underlying physics is partially or entirely unknown, relying on flexible and expressive black-box components such as neural networks.

However, a fundamental challenge of black-box system identification is the need for large datasets to ensure model reliability across a wide range of operating conditions. Even when such datasets are available, training deep models with numerous parameters can be computationally demanding, limiting their practical applicability. This contrasts with physics-based modeling, which, when feasible, enables data-efficient learning and typically results in models that generalize well beyond the training regime.

To bridge the gap between black-box and physics-based approaches, recent research has explored the opportunity to include domain knowledge into deep-learning architectures and training algorithms, while preserving their flexibility. These efforts include designing custom model structures that satisfy known physical constraints by design [9,14,21] and introducing regularization terms in the loss function to promote the satisfaction of known dynamics, as seen in physics-informed machine learning [22,24]. While highly successful, these approaches still require expert knowledge and manual tuning for the design of appropriate architectures and regularizers.

This paper addresses the challenge of learning simplified model architectures in an automated manner using meta-learning techniques [19]. In line with the meta-learning setting for system identification [7,15], we assume access to a *meta-dataset*, namely, a collection of datasets generated by different,

yet related, dynamical systems representative of a target population that we aim to describe. The meta-dataset may originate either from a high-fidelity simulator with physical parameters randomized within plausible ranges or from historical real-world data collected in prior experimental campaigns or through monitoring of processes deployed in series. Additionally, we assume that a given black-box architecture, referred to as the *base architecture*, is sufficiently expressive to capture the dynamics of each system in the meta-dataset for at least one (system-dependent) choice of its tunable parameters. However, this base architecture is highly over-parameterized, meaning it has significantly more degrees of freedom than the actual factors governing variability across systems in the meta-dataset. Given the meta-dataset and the base architecture, our goal is to meta-learn a *low-dimensional manifold* embedded in the base architecture's parameter space such that the base architecture, with parameters restricted to this manifold, can still describe all the systems within the class of interest.

Conceptually, our approach is inspired by and builds upon classic approaches for model-based meta-learning, which aim at *optimizing the algorithm* (or parts thereof) that estimates a model within a given base architecture [19]. For instance, the celebrated model-agnostic meta-learning (MAML) [12] seeks an *initialization* of the base architecture's parameters such that, with a few steps of gradient descent starting from that point, effective adaptation to each dataset in the meta-dataset is achieved. This meta-learning problem is tackled through bilevel optimization, where the initial parameters are optimized at the outer level, while adaptation to the specific datasets is performed via gradient descent at the inner level. Unlike MAML, our methodology meta-learns a low-dimensional manifold in the base architecture's parameter space instead of providing an initialization for optimization, and can be related to learning an effective *inductive bias* [17] to facilitate the system identification procedure, in terms of both fitting the model parameters and required training data.

An existing meta-learning algorithm that even more closely resembles our approach is latent embedding optimization (LEO) [25] which, similarly to our approach, meta-learns a low-dimensional embedding of the datasets. Compared to LEO (and MAML), we circumvent the need to perform bilevel optimization by introducing an *auxiliary neural network*, estimated together with the manifold, that directly maps each dataset onto its low-dimensional representation, bypassing the need for gradient-based inner-loop optimization. Furthermore, we formalize our methodology in a regression setting rather than a classification one and demonstrate its application to system identification using the Bouc-Wen benchmark [23].

Meta-learning has been recently applied for system identification, among other works, in [6, 10, 15]. However, [10, 15] build a unified *meta-model* for the entire meta-dataset, relying on the *in-context* learning capabilities of transformer architectures. Conversely, [6] is based on the MAML algorithm and its variants; unlike the current approach, MAML-like approaches involve a bilevel training procedure. While this is effective, a major drawback is the complexity of the bilevel training procedure. To avoid this, workarounds such as implicit MAML (iMAML) has been investigated by [27], which leverages the implicit function theorem to simplify the inner-loop training. Also, [5] demonstrates the potential of almost-no-inner-loop (ANIL), which restricts the inner-loop training to a small set of layers or components in the network. In contrast to ANIL, our method enables the meta-learning procedure to discover a low-dimensional manifold in parameter space where adaptation occurs, rather than specifying in advance which components of the base architecture should be adapted.

The rest of the paper is organized as follows. The meta-learning framework is introduced in Section 2 and the meta-learning procedure is detailed in Section 3. Numerical results are presented in Section 4, and conclusions are drawn in Section 5.

2 Framework

2.1 Data stream and distribution

In line with the meta-modeling framework introduced in [15], we assume to have access to a (possibly unlimited) collection $\mathcal{D}$ of datasets referred to as the meta-dataset, described by

$$\mathcal{D} = \{D^{(i)}\}_{i \in \mathbb{N}}, \text{ where } D^{(i)} = (\mathbf{u}^{(i)}_{0:N-1}, \mathbf{y}^{(i)}_{0:N-1}). \tag{1}$$

Each dataset $D^{(i)}$ comprises N inputs $\mathbf{u}^{(i)}_k \in \mathbb{R}^{n_u}$ and corresponding targets $\mathbf{y}^{(i)}_k \in \mathbb{R}^{n_y}, k = 0, 1, \ldots, N$. In the context of system identification, $\mathbf{y}^{(i)}_{0:N-1}$ is the sequence of outputs generated by a dynamical

system $S^{(i)}$ driven by the sequence of control actions $\mathbf{u}_{0:N-1}^{(i)}$. This results in a causal dependency $\mathbf{u}_{0:k}^{(i)} \to \mathbf{y}_k^{(i)}$. In the case of static regression, the dependency is simply $\mathbf{u}_k^{(i)} \to \mathbf{y}_k^{(i)}$.

In the following, we adopt the shorthand notation $\mathbf{u} = \mathbf{u}_{0:N-1}, \mathbf{y} = \mathbf{y}_{0:N-1}$ to denote the sequence of N input and output samples, respectively, contained in a dataset. The inputs $\mathbf{u}^{(i)}$ of each dataset are assumed to be drawn from a distribution with density $p(\mathbf{u})$, while the dependency $\mathbf{u}^{(i)} \to \mathbf{y}^{(i)}$ is different, yet related across datasets. Formally, the collection of datasets (1) may be interpreted as samples drawn from a distribution (in the space of datasets) with density function:

$$p(D) = p(\mathbf{u}, \mathbf{y}) = p(\mathbf{u}) \int_z p(z) p(\mathbf{y}|\mathbf{u}, z)\, dz, \tag{2}$$

where $z \in \mathbb{R}^{n_z}$ is some unknown or hidden variable that characterizes the data-generating mechanism $p(\mathbf{y}|\mathbf{u}, z)$.

In a synthetic data generation setting, z could correspond to parameters of a physical simulator randomized within admissible ranges. Samples $D^{(i)}$ are drawn from $p(D)$ by first sampling a random $z^{(i)}$, corresponding to a random system $S^{(i)}$, from $p(z)$, together with a random input $\mathbf{u}^{(i)}$ from $p(\mathbf{u})$. Then, the output $\mathbf{y}^{(i)}$ is obtained by processing input $\mathbf{u}^{(i)}$ through the system $S^{(i)}$.

In the case of real data, z may correspond to the operating conditions that vary (intentionally or not to the experimenter) across different runs, such as different masses attached to the end-effector of a robot, environment temperature in a chemical reaction, line voltage for electrical equipment, etc. Furthermore, the exact meaning and (minimum) dimension of z might be unknown.

Remark 1 *The latent variable z corresponds to the aspects of the data-generating system that vary across datasets. For instance, in the case of the robot where only the end effector mass changes, a scalar z representing such mass suffices, even though the robot itself is characterized by many other physical parameters, that, however, remain fixed for all runs. Those non-varying parameters may be thought of as part of the functional form of the conditional distribution $p(\mathbf{y}|\mathbf{u}, z)$ in (2).*

2.2 Base architecture

Consider the base architecture given by the black-box full-order model structure

$$\hat{\mathbf{y}} = F(\mathbf{u}; \theta),\ \theta \in \mathbb{R}^{n_\theta}, \tag{3}$$

which we assume is prescribed to describe the input/output dependency $\mathbf{u} \to \mathbf{y}$ for each dataset D in the support of $p(D)$ with practically no bias, for some system-dependent (unknown) value of the parameter θ. We argue that this assumption is not particularly restrictive given the availability of expressive function approximators such as deep neural networks, that, with appropriate selection of hyperparameters, can describe wide classes of nonlinear functions with high accuracy, and have been reported to exhibit competitive performance across different system identification benchmarks [8].

The model structure (3) is typically characterized by a number of parameters $n_\theta \gg n_z$ for two reasons. First, (3) has a black-box structure, not necessarily tailored to a specific class of system. For example, accurate description of robot dynamics with $l = 8$ links may require thousands of black-box parameters ($n_\theta \approx 10^3$) with neural-network models [3], whereas a physics-based representation only requires $10l = 80$ terms [16]. Second, just a subset of these physical parameters may be varying across the simulations/experimental runs within the meta-dataset. For instance, as in the example of Remark 1, if the only physical parameter varying across dataset is the robot's end-effector mass, then $n_z = 1$ suffices. Consequently, black-box architectures used to describe physical systems are often heavily over-parameterized.

For a given dataset $D = (\mathbf{u}, \mathbf{y})$, the optimal parameter that best describes the input/output relation is typically obtained by minimizing a supervised learning criterion:

$$\hat{\theta} = \arg\min_\theta \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + r(\theta), \tag{4}$$

where $\mathcal{L}(\cdot, \cdot)$ is a training loss that penalizes the mismatch between measurements $\mathbf{y}$ and predictions $\hat{\mathbf{y}} = F(\mathbf{u}, \theta)$, and $r : \mathbb{R}^{n_\theta} \to \mathbb{R}$ is a regularization term. A common choice for $\mathcal{L}(\cdot, \cdot)$ is the mean squared error $\mathrm{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{k=1}^{N} \|\mathbf{y}_k - \hat{\mathbf{y}}_k\|_2^2$ paired with an ℓ_2-regularization term $r(\theta) = \rho \|\theta\|_2^2$, where $\rho \geq 0$ controls the regularization strength.

However, when the number of parameters n_θ and/or the number of samples N in the training dataset D are large, learning the dynamics represented by F can be computationally intensive. Conversely, when N is small, achieving good *generalization* to unseen data becomes difficult.

2.3 Meta-learning manifolds in the parameter space

To address the computational and generalization challenges related to full-order system identification, in this paper we address the following problem:

Problem 1 *Given a full-order architecture* (3) *and a collection* $\mathcal{D}$ *of datasets sampled from* $p(D)$*, learn a lifting function* $P : \mathbb{R}^{n_\phi} \to \mathbb{R}^{n_\theta}$*, with* $n_\phi \ll n_\theta$ *that defines a* n_ϕ*-dimensional manifold* $M = \{\theta \in \mathbb{R}^{n_\theta} : \theta = P(\phi),\ \phi \in \mathbb{R}^{n_\phi}\}$ *such that each dataset* $D = (\mathbf{u}, \mathbf{y})$ *in the support of* $p(D)$ *can be described accurately by a model* $\hat{\mathbf{y}} = F(\mathbf{u}; P(\phi))$ *for some coordinate vector* $\phi \in \mathbb{R}^{n_\phi}$*.*

We label Problem 1 as a *meta-learning* problem, as our goal is to learn a unique manifold M for the *entire meta-dataset*. Once the manifold is learned, the user can fit a model to any dataset $D = (\mathbf{u}, \mathbf{y})$ contained in the support of $p(D)$ by solving a reduced-complexity training problem, in that the number of free parameters is (significantly) reduced from n_θ to n_ϕ.

Problem 1 admits at least one solution for $n_\phi \geq n_z$. Indeed, we hypothesized in Section 2.2 that each data-generating system in the distribution (equivalently, each $z \in \mathbb{R}^{n_z}$) is represented by (at least) a $\theta \in \mathbb{R}^{n_\theta}$. Then, there exists a function $P^\dagger : \mathbb{R}^{n_z} \to \mathbb{R}^{n_\theta}$ mapping each data-generating system (characterized by a $z \in \mathbb{R}^{n_z}$) to a parameter $\theta \in \mathbb{R}^{n_\theta}$ of the full black-box model structure.[1] Therefore, the model structure $\hat{\mathbf{y}} = F(\mathbf{u}; P^\dagger(z)),\ z \in \mathbb{R}^{n_z}$ solves Problem 1 with $P = P^\dagger$, $\phi = z$. However, the function $P^\dagger$, as well as n_z, are not generally known a priori.

Let us parameterize function P by a vector γ of tunable parameters, $\gamma \in \mathbb{R}^{n_\gamma}$, and denote it by P_γ. As we will detail in Section 3, our meta-learning goal is to learn an optimal lifting parameter vector $\hat{\gamma} \in \mathbb{R}^{n_\gamma}$ for the entire set $\mathcal{D}$ to define the reduced-complexity architecture

$$\hat{\mathbf{y}} = F(\mathbf{u}; P_{\hat{\gamma}}(\phi)),\ \phi \in \mathbb{R}^{n_\phi}, \tag{5}$$

given by the functional composition of the base architecture with the *learned* lifting function $P_{\hat{\gamma}} : \mathbb{R}^{n_\phi} \to \mathbb{R}^{n_\theta}$.

Based on the learned reduced-complexity architecture (5), and given a new dataset $D^* = (\mathbf{u}^*, \mathbf{y}^*)$ that is contained in the support of $p(D)$, the following reduced-complexity optimization problem

$$\hat{\phi} = \arg\min_{\phi} \mathcal{L}(\mathbf{y}^*, F(\mathbf{u}^*, P_{\hat{\gamma}}(\phi))) \tag{6}$$

solves the system identification problem of fitting a model to D^*. Given the reduced number n_ϕ of variables, the optimization problem (6) can be solved much more efficiently than (4), e.g., by taking advantage of quasi-Newton approaches (see, e.g., [2, 3]) that might not be computationally feasible in the full-order structure (3). Furthermore, the risk of overfitting is significantly mitigated by the reduced number of free parameters.

Remark 2 *In case* $p(D)$ *is a synthetic dataset distribution, a parametric architecture with parameters* z *describing the dependency* $\mathbf{u} \to \mathbf{y}$ *is already available a priori, and it corresponds to the conditional density* $p(\mathbf{y}|\mathbf{u}, z)$ *in* (2)*. Nonetheless, such architecture may be defined in terms of complex mathematical objects such as stiff ordinary differential equations, partial differential equations, implicit algebraic constraints, non-differentiable blocks, or components subject to restricted interaction in the case of commercial simulators. Conversely, the learned architecture* (5) *allows for full customization by the user.*

3 Meta-learning procedure

In line with the standard meta-learning setting, we split each dataset $D^{(i)}$ into training and test portions, i.e.,

$$D^{(i)} = (\mathbf{u}_{\text{tr}}^{(i)}, \mathbf{y}_{\text{tr}}^{(i)}), (\mathbf{u}_{\text{te}}^{(i)}, \mathbf{y}_{\text{te}}^{(i)}),\ i = 1, 2, \ldots, \tag{7}$$

[1] The relationship $z \to \theta$ might be one-to-many. The function $P^\dagger$ here arbitrary chooses one of the multiple equivalent solutions.

where both $\mathbf{y}_{\text{tr}}^{(i)}$ and $\mathbf{y}_{\text{te}}^{(i)}$ are generated by the *same* system $S^{(i)}$ for different input signals $\mathbf{u}_{\text{tr}}^{(i)}$ and $\mathbf{u}_{\text{te}}^{(i)}$, respectively. Such a data split allows one to learn an optimal manifold M such that, when training is performed on $(\mathbf{u}_{\text{tr}}^{(i)}, \mathbf{y}_{\text{tr}}^{(i)})$, performance measured on test data $(\mathbf{u}_{\text{te}}^{(i)}, \mathbf{y}_{\text{te}}^{(i)})$ is good.

A direct way of learning an optimal parametric function $P_{\hat{\gamma}}$ is to solve the following bilevel optimization problem:

$$\hat{\gamma} = \arg\min_{\gamma} \mathbb{E}_{p(D)} \left[\mathcal{L}(\mathbf{y}_{\text{te}}, \hat{\mathbf{y}}_{\text{te}}) \right] \tag{8a}$$

$$\hat{\mathbf{y}}_{\text{te}} = F\left(\mathbf{u}_{\text{te}};\, P_{\gamma}(\hat{\phi}) \right) \tag{8b}$$

$$\hat{\phi} = \text{SYSID}(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}}), \tag{8c}$$

where SYSID is the system identification algorithm that the modeler intends to use later on the real data to fit ϕ on a given dataset D. The most obvious choice for SYSID is to align the inner-loop optimization with the intended reduced-complexity fitting criterion (6):

$$\text{SYSID}(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}}) = \arg\min_{\phi} \mathcal{L}(F(\mathbf{u}_{\text{tr}};\, P_{\gamma}(\phi)), \mathbf{y}_{\text{tr}}). \tag{9}$$

In essence, (8) explicitly tunes the lifting function $P_{\hat{\gamma}}$ to provide optimal expected performance, when a specific system identification algorithm is applied in training.

3.1 Encoder-decoder architecture

To circumvent the numerical intricacies related to the nested optimization problem formulation (8), we replace the inner optimization with an auxiliary neural network E_{ψ}, which may be interpreted as a *learned* black-box SYSID algorithm:

$$\hat{\gamma}, \hat{\psi} = \arg\min_{\gamma, \psi} \mathbb{E}_{p(D)} \overbrace{\left[\mathcal{L}(\mathbf{y}_{\text{te}}, \hat{\mathbf{y}}_{\text{te}},) \right]}^{J(\gamma, \psi)} \tag{10a}$$

$$\hat{\mathbf{y}}_{\text{te}} = F\left(\mathbf{u}_{\text{te}};\, P_{\gamma}(\hat{\phi}) \right) \tag{10b}$$

$$\hat{\phi} = E_{\psi}(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}}). \tag{10c}$$

Note that E_{ψ} is an (approximate) multiparametric solution [11] to the optimization problem associated with (8c), whose unknowns are the entries of ϕ and parameters are $(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}})$.

The pair (10b)-(10c) may also be seen as an encoder-decoder architecture. The encoder E_{ψ} ingests the input-output pair $(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}})$ and outputs reduced-complexity parameters $\hat{\phi}$. The decoder, defined by the functional composition $F(\cdot, P_{\hat{\gamma}}(\cdot))$, first lifts the reduced parameters $\hat{\phi}$ to the original full-order dimension n_{θ}, and then applies the base architecture with the lifted parameters to the input $\mathbf{u}_{\text{te}}$ to predict the output $\hat{\mathbf{y}}_{\text{te}}$. Figure 1 is a graphical representation of (10b)-(10c) focusing on this encoder-decoder interpretation. Finally, we highlight that the encoder plays the role of a hyper-network [18], as it provides the parameters of another neural network, namely of the base architecture F.

To solve (10), we replace the intractable expectation over $p(D)$ by a Monte Carlo average. In other words, $J(\gamma, \psi)$ in (10) is approximated as a mean over b sampled datasets $D^{(i)}$ from the meta-dataset $\mathcal{D}$ leading to:

$$\tilde{J}(\gamma, \psi) = \frac{1}{b} \sum_{i=1}^{b} \mathcal{L}\left(\mathbf{y}_{\text{te}}^{(i)}, F\left(\mathbf{u}_{\text{te}}^{(i)}, P_{\gamma}(E_{\psi}(\mathbf{u}_{\text{tr}}^{(i)}, \mathbf{y}_{\text{tr}}^{(i)})) \right) \right) \tag{11}$$

The loss $\tilde{J}(\gamma, \psi)$ is minimized with standard gradient-based optimization. In case the datasets $D^{(i)}$ are resampled at each iteration, b plays the role of the batch size.

We emphasize that the main objective of solving (10) is to find the lifting function $P_{\hat{\gamma}}$ to be plugged into (5), and, therefore, the manifold M of dimension n_{ϕ} that best reduces the complexity of identifying a model over the dataset distribution $p(D)$. Once this mapping $P_{\hat{\gamma}}$ is learned, given a new dataset $D^* = (\mathbf{u}^*, \mathbf{y}^*)$, the user may prefer to solve the identification problem (6) explicitly as in (8c), instead of relying on the learned algorithm $E_{\hat{\psi}}$. In fact, solving (6) is arguably more trustworthy than relying on a black-box map. Nonetheless, the learned algorithm $E_{\hat{\psi}}$ may be used to initialize an iterative algorithm to evaluate SYSID in (8c), thereby speeding up the reduced-complexity model learning task (8c).

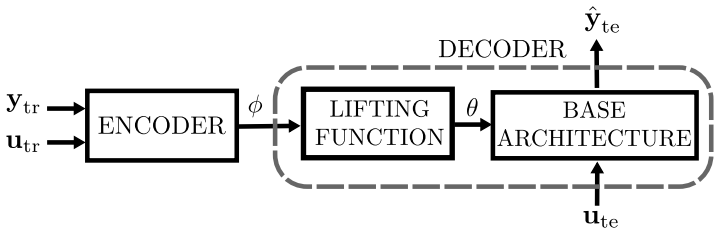

Figure 1: The autoencoder architecture for reduced-complexity architecture learning.

3.2 Specialization for neural state-space models

We now specialize the methodology presented in the previous sections for datasets $D^{(i)}$ that are input/output time series generated by dynamical systems, and the base architecture is a state-space model:

$$x_{k+1} = f(x_k, u_k; \theta) \qquad y_k = g(x_k, u_k; \theta). \tag{12}$$

In (12), f and g are learnable mappings, such as feed-forward neural networks. In this case, the output sequence $\mathbf{y}$, given an input sequence $\mathbf{u}$, also depends on an initial state x_0. Formally, we have: $\hat{\mathbf{y}} = F(\mathbf{u}, x_0; \theta)$. Thus, in applying the previously mentioned methodology, care has to be taken in the initialization of this state, notably for a correct computation of the loss in (10a).

A few practical solutions have been introduced in recent SYSID literature to tackle the initial state issue, see [13] for an overview. Similarly to [1, 3, 20], an additional neural network can be included to processes the first n_{init} samples of $(\mathbf{u}_{\text{te}}, \mathbf{y}_{\text{te}})$ and provide an estimate $\hat{x}_{n_{\text{init}}}$ of the state at time step n_{init}. Starting from this state estimate, the loss (10a) may be computed, excluding the first n_{init} samples from the loss evaluation. We note that, in the case of synthetic data, the two datasets $(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}})$, $(\mathbf{u}_{\text{te}}, \mathbf{y}_{\text{te}})$ may be chosen to have the *same* (random) initial state. In this case, the encoder network E_ψ may be designed to provide initial state estimates, together with the reduced-complexity model parameters, thereby simplifying the architecture.

Finally, as argued in [13], an even simpler approach may be followed when the systems to be modeled are known to have a fading memory. In this case, the state can be initialized to an arbitrary value (such as zero), and the effect of the system's "natural response" can be removed from the loss by ignoring the contribution from the first n_{skip} samples, where n_{skip} is selected large enough. In the numerical examples presented in Section 4, for simplicity, problem (10) is instantiated relying on such a "zero" state initialization scheme. More advanced schemes may be integrated with a moderate effort.

4 Numerical illustration

We apply our approach to the Bouc-Wen system identification benchmark [23]. The software was developed in Python using the JAX automatic differentiation library [4] and it is available at the repository `https://github.com/forgi86/sysid-neural-manifold`. All computations were performed on a server equipped with an AMD EPYC 9754 CPU and an Nvidia 4090 GPU.

4.1 The Bouc-Wen benchmark

The Bouc-Wen benchmark entails the vibrations of a 1-DoF hysteretic mechanical system. The system dynamics are described in state-space form by:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \frac{1}{m_L}\left(u(t) - k_L p(t) - c_L v(t) - z(t)\right) \\ \alpha v(t) - \beta(\gamma |v(t)||z(t)|^{\nu-1} + \delta v(t)|z(t)|^{\nu}) \end{bmatrix} \tag{13a}$$

$$y(t) = p(t), \tag{13b}$$

where $p(t)$ (m) is the measured output position; $v(t)$ (m/s) is the unmeasured velocity; $z(t)$ (N) is an unmeasured hysteretic force term; and $u(t)$ (N) is the known input force. The nominal value of the model coefficients are reported in the first row of Table 1.

Table 1: Coefficients of the Bouc-Wen system. Nominal values (first row) and minimum/maximum values in the dataset distribution (second/third row).

	m_L	c_L	k_L	α	β	γ	δ	ν
nom	2	10	5.0×10^4	5.0×10^4	1000	0.8	-1.1	1
min	1	5	2.5×10^4	2.5×10^4	500	0.5	-1.5	1
max	3	15	7.5×10^4	7.5×10^4	4500	0.9	-0.5	1

The benchmark provides a test dataset where the input $u(t)$ is a random-phase multisine of length 8192 sampled at frequency $f_s = 750$ Hz that excites the frequency range $[5, 150]$ Hz and has a root mean square of 50 N. System (13) is simulated with its nominal coefficients and the test output $y(t) = p(t)$ is provided noise-free.

A "default" training dataset is also provided, comprising 40960 samples generated by exciting the system with a multisine input signal with the same characteristic as the test input. In the training dataset, the output is corrupted by an additive Gaussian noise with bandwidth $[0 - 375]$ Hz and amplitude $8 \cdot 10^{-3}$. Moreover, the benchmark provides simulation code and specifies that the user is free to choose different input signals to excite the system and generate arbitrary training datasets.

In this paper, along the lines of [6], we extend the original scope of the benchmark [23] and use the simulator to generate datasets with perturbed coefficient values, aiming at finding a reduced-complexity architecture that provides high performance over the entire *class* of Bouc-Wen systems with parameters comprised in the min-max range reported in Table 1.

4.2 Base architecture

As base architecture, we consider the neural state-space structure:

$$x_{k+1} = Ax_k + Bu_x + N_f(x_k, u_k; W_f) \tag{14a}$$
$$y_k = Cx_k + N_g(x_k; W_g), \tag{14b}$$

where $n_u = 1$, $n_y = 1$, $n_x = 3$, N_f, and N_g are feed-forward neural networks with one hidden layer and 16 hidden units with weights W_f and W_g, respectively, and `tanh` activation function; and A, B, and C are matrices of proper size. The base architecture parameters are thus $\theta = \text{vec}(W_f,\ W_g,\ A,\ B,\ C) \in \mathbb{R}^{n_\theta}$, with $n_\theta = 244$. Note that a very similar architecture has been utilized in [26] and shown to provide state-of-the-art results on this nonlinear system identification benchmark.

4.3 Metrics and baselines

We evaluate model performance in terms of the indices

$$\text{fit} = 100 \cdot \left(1 - \frac{\sqrt{\sum_{k=0}^{N-1} (\mathbf{y}_k - \hat{\mathbf{y}}_k)^2}}{\sqrt{\sum_{k=0}^{N-1} (\mathbf{y}_k - \overline{\mathbf{y}})^2}}\right) (\%), \qquad \text{rmse} = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} (\mathbf{y}_k - \hat{\mathbf{y}}_k)^2},$$

where $\overline{\mathbf{y}}$ is the sample average of the measured output $\mathbf{y}$.

Fitting the model on the default 40960-length training dataset, [26] reported the state-of-the-art result on this benchmark (fit = 98.9 %, rmse = $7.16 \cdot 10^{-6}$) on the test set. We were able to closely match this result (fit = 98.8 %, rmse = $7.87 \cdot 10^{-6}$) by minimizing the MSE loss of the base architecture over 40000 iterations of `AdamW` with learning rate 10^{-3} followed by 10000 iterations of `BFGS`. The training time was 1.4 hours. The eigenvalues of the Hessian of the full-order training loss at the solution of the optimization problem are visualized in Figure 2. Interestingly, many of those eigenvalues are close to zero, which suggests that the model structure is over-parameterized and further motivates the pursue for a lower-dimensional parameterization.

We also train as a baseline a third-order linear model that provides fit = 77.2 %, underlining the relevance of non-linearities in this benchmark.

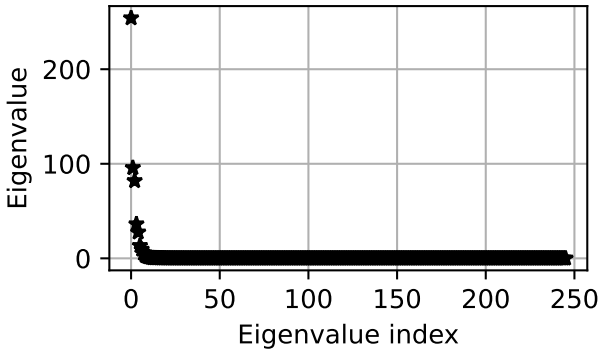

Figure 2: Eigenvalues of the Hessian of the full-order training loss.

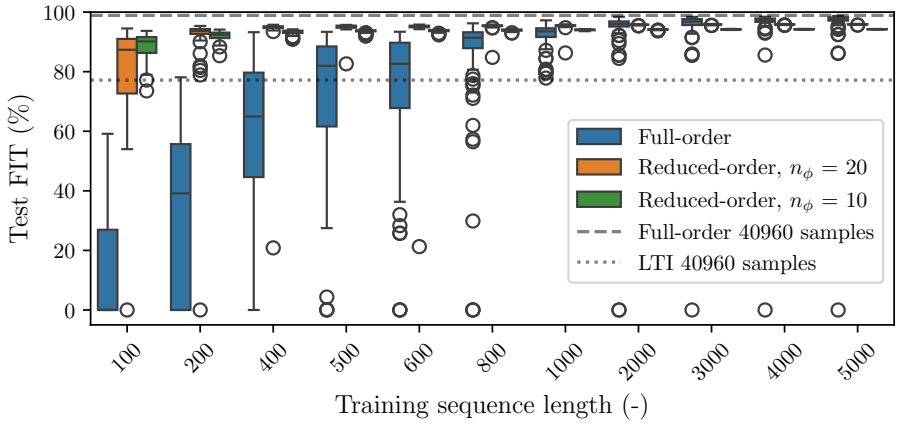

Figure 3: Full and reduced-order models: test FIT vs. training sequence length L.

4.4 Full-order model training

To assess the performance of full-order model learning in the low-data regime, we train models over shorter sequences of length $L = 100, 200, 400, 500, 600, 800, 1000, 2000, 3000, 4000, 5000$ samples. For each length L, we execute a Monte Carlo study and train 100 distinct models over 100 L-length subsequences randomly extracted from the official 40960-length dataset. For optimization, we use the same `AdamW` + `BFGS` approach with the same settings detailed in the previous section. As shown in Figure 3, full-order model learning proves highly inefficient in the low-data regime. In particular, for sequence length $L \leq 400$, the median fit is below the linear baseline. The median performance consistently improves for increasing sequence length, with average fit above 97 % for $L \geq 3000$ samples. However, even for $L = 2000, 3000, 4000$, and 5000, a small number of runs (2, 1, 1, and 1, respectively) fail due to numerical issues in the `BFGS` solver. While these issues might be mitigated with a careful tuning of the solver settings, they highlight the challenges of full-order model training.

Figure 4 illustrates the time required to train in parallel the 100 models vs. the sequence length L. We report both the run time for the 40000 `AdamW` iterations and the total training time, which includes the 10000 `BFGS` iterations. The results show that `BFGS` dominates the overall training time.

4.5 Meta-learning the reduced-complexity architecture

To enable higher performance in the low-data regime, we learn a manifold with $n_\phi = 20$ parameters, embedded within the full-order architecture (14) characterized by $n_\theta = 244$ parameters, by applying the methodology described in Section 3. To this aim, we introduce the parameterized lifting network $P_\gamma : \mathbb{R}^{n_\phi} \to \mathbb{R}^{n_\theta} = V\phi + \theta_{\text{bias}}$ with $V \in \mathbb{R}^{n_\theta \times n_\phi}, \theta_{\text{bias}} \in \mathbb{R}^{n_\theta}$, which expands the reduced-complexity

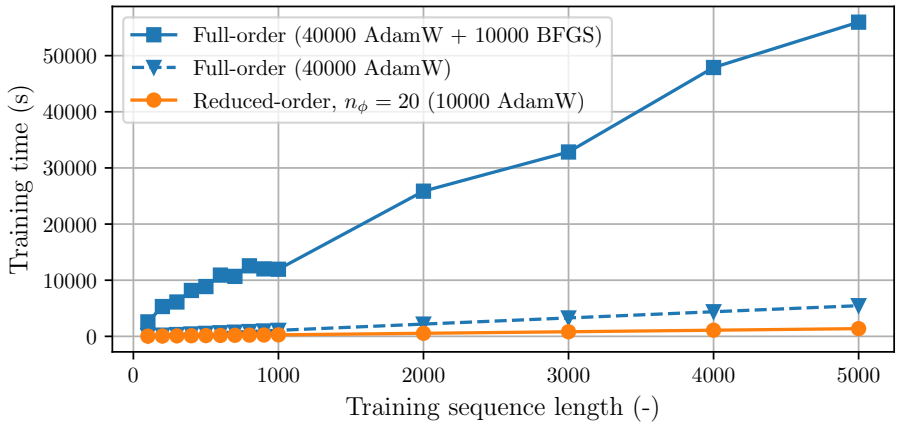

Figure 4: Training time vs. training sequence length L.

dimension n_ϕ to the full-order n_θ. Its tunable parameters are collected in the vector $\gamma = \text{vec}(V, \theta_{\text{bias}}) \in \mathbb{R}^{n_\gamma}$, with $n_\gamma = (n_\phi + 1)n_\theta = 5124$.

To learn the optimal lifting network parameters $\hat{\gamma}$, we adopt the encoder-decoder methodology and solve the optimization problem (10), using the Monte Carlo approximation $\tilde{J}(\gamma, \psi)$ in (11) of the loss $J(\gamma, \psi)$. The meta-dataset $\mathcal{D}$ is obtained by randomly sampling Bouc-Wen systems with coefficients uniformly distributed within the min-max ranges in Table 1, and feeding as input random multisine sequences of length $N = 2000$ with the same properties as the default training input.

The encoder E_ψ is a sequence-to-vector architecture obtained by combining a bi-directional Gated Recurrent Unit (GRU) with a feed-forward neural network. The GRU is configured with $n_u + n_y = 2$ input channels and a single hidden layer with $n_h = 128$ hidden units. It processes the sequences $(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}})$ and produces a sequence of activations $\mathbf{h}_{0:N-1}$, $\mathbf{h}_i \in \mathbb{R}^{n_h}$. This sequence is reduced to a single vector h_m through *average pooling*: $h_m = \frac{1}{N}\sum_i \mathbf{h}_i$ and further processed by the feed-forward neural network configured with $n_h = 128$ inputs, a hidden layer with 128 units, and an output layer with $n_\phi = 20$ units, and `tanh` activation functions. Overall, the network E_ψ has $n_\psi = 136340$ parameters corresponding to the GRU and the feed-forward neural network weights. It takes as input the sequences $(\mathbf{u}_{\text{tr}}, \mathbf{y}_{\text{tr}})$ and produces as output a single vector of size n_ϕ.

For gradient-based minimization of the loss (11), the `Adam` algorithm is applied with a batch size of 128 and initial learning rate of $2 \cdot 10^{-4}$, decreased to $2 \cdot 10^{-5}$ with cosine scheduling over 200000 iterations. The optimization takes approximately 25.3 hours. As a result, we obtain the reduced-order architecture (5) and, as a byproduct, the encoder network $E_{\hat{\psi}}$ which can be used to initialize the reduced-complexity model learning (6).

4.6 Reduced-complexity model training

We repeat the Monte Carlo study of Section 4.4 using the reduced-order architecture meta-learned in Section 4.5, instead of the base architecture. In all the runs, the loss (6) is minimized over 10000 iterations of `AdamW` with learning rate 10^{-3}, using the encoder network $E_{\hat{\psi}}$ to obtain an initial guess of ϕ. Results in Figure 3 show that the reduced-complexity training is significantly more effective than the full-order one for sequences of length up to 2000 samples. Very good results are obtained with just 500 samples (median fit of 95.2%, median rmse $= 3.18 \cdot 10^{-5}$). For training sequences longer than 2000 samples, full-order training generally yields superior results, given the higher flexibility of the model structure.

We remark that none of the reduced-order training runs exhibited numerical difficulties, and the obtained performance is largely insensitive to the choice of the optimizer and its settings. This underscores the stability and reliability of the reduced-order model learning problem.

The run time of reduced-order training is visualized in Figure 4 and corresponds to approximately 25 % the `AdamW` portion of the full-order case. This is because the cost of a single `AdamW` iteration remains nearly identical for both cases, as evaluating the lifting function P is negligible compared to

computing the training loss. Due to the lower problem dimensionality and to the effective optimization initialization using $E_{\hat{\phi}}$, 10000 `AdamW` iterations are sufficient for reduced-order model training.

To analyze the sensitivity of the methodology to the choice of the manifold dimensionality, we repeat the reduced-complexity architecture meta-learning and the Monte Carlo study for $n_\phi = 10$. Results in Figure 3 highlight that this choice leads to the best results for the very short sequences of length $L = 100$, while it leads to worse performance against $n_\phi = 20$ for all other lengths.

5 Conclusions

This paper introduced a meta-learning framework for system identification that discovers a low-dimensional manifold within the parameter space of an over-parameterized neural architecture. Our results on the Bouc-Wen benchmark highlight the potential of this approach, particularly in extremely data-scarce scenarios.

Several extensions of this framework can be explored. While our methodology was introduced in the context of a black-box base architecture, it could be extended to models, constraints, and fitting criteria informed by physical knowledge. Additionally, integrating variational learning techniques into the meta-learning process could enable the discovery of not only a deterministic low-dimensional representation but also a prior distribution over this space. This would support a principled probabilistic approach to reduced-complexity modeling, where uncertainty is propagated from a meta-learned prior, enhancing robustness, generalization, and interpretability.

References

[1] G. I. Beintema, M. Schoukens, and R. Tóth. Deep subspace encoders for nonlinear system identification. *Automatica*, 156:111210, 2023.

[2] A. Bemporad. Training recurrent neural networks by sequential least squares and the alternating direction method of multipliers. *Automatica*, 156:111183, 2023.

[3] A. Bemporad. An L-BFGS-B approach for linear and nonlinear system identification under ℓ_1- and group-lasso regularization. *IEEE Transactions on Automatic Control*, 2025. In press.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] A. Chakrabarty, V. Deshpande, G. Wichern, and K. Berntorp. Physics-constrained meta-learning for online adaptation and estimation in positioning applications. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 1561–1566. IEEE, 2024.

[6] A. Chakrabarty, G. Wichern, V. M. Deshpande, A. P. Vinod, K. Berntorp, and C. R. Laughman. Meta-learning for physically-constrained neural system identification. *arXiv preprint arXiv:2501.06167*, 2025.

[7] A. Chakrabarty, G. Wichern, and C. R. Laughman. Meta-learning of neural state-space models using data from similar systems. *IFAC-PapersOnLine*, 56(2):1490–1495, 2023.

[8] M. D. Champneys, G. I. Beintema, R. Tóth, M. Schoukens, and T. J. Rogers. Baseline results for selected nonlinear system identification benchmarks. *arXiv preprint arXiv:2405.10779*, 2024.

[9] V. M. Deshpande, A. Chakrabarty, A. P. Vinod, and C. R. Laughman. Physics-constrained deep autoencoded kalman filters for estimating vapor compression system states. *IEEE Control Systems Letters*, 7:3483–3488, 2023.

[10] Z. Du, H. Balim, S. Oymak, and N. Ozay. Can transformers learn optimal filtering for unknown systems? *IEEE Control Systems Letters*, 7:3525–3530, 2023.

[11] A. V. Fiacco. *Introduction to sensitivity and stability analysis in nonlinear programming*. Academic Press, London, U.K., 1983.

[12] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[13] M. Forgione, M. Mejari, and D. Piga. Learning neural state-space models: Do we need a state estimator? *arXiv preprint arXiv:2206.12928*, 2022.

[14] M. Forgione and D. Piga. Continuous-time system identification with neural networks: Model structures and fitting criteria. *European Journal of Control*, 59:69–81, 2021.

[15] M. Forgione, F. Pura, and D. Piga. From system models to class models: An in-context learning paradigm. *IEEE Control Systems Letters*, 2023.

[16] C. Gaz, M. Cognetti, A. Oliva, P. R. Giordano, and A. De Luca. Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 4(4):4147–4154, 2019.

[17] A. Goyal and Y. Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266):20210068, 2022.

[18] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[19] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

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

[21] S. Moradi, N. Jaensson, R. Tóth, and M. Schoukens. Physics-informed learning using hamiltonian neural networks with output error noise models. *IFAC-PapersOnLine*, 56(2):5152–5157, 2023.

[22] T. X. Nghiem, J. Drgoňa, C. Jones, Z. Nagy, R. Schwan, B. Dey, A. Chakrabarty, S. Di Cairano, J. A. Paulson, A. Carron, et al. Physics-informed machine learning for modeling and control of dynamical systems. In *2023 American Control Conference (ACC)*, pages 3735–3750. IEEE, 2023.

[23] J.-P. Noel and M. Schoukens. Hysteretic benchmark with a dynamic nonlinearity. In *Workshop on nonlinear system identification benchmarks*, pages 7–14, 2016.

[24] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[25] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[26] M. Schoukens. Improved initialization of state-space artificial neural networks. In *2021 European Control Conference (ECC)*, pages 1913–1918. IEEE, 2021.

[27] J. Yan, A. Chakrabarty, A. Rupenyan, and J. Lygeros. MPC of uncertain nonlinear systems with meta-learning for fast adaptation of neural predictive models. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pages 1910–1915. IEEE, 2024.