PARAMETER-VARYING NEURAL ORDINARY DIFFERENTIAL EQUATIONS WITH PARTITION-OF-UNITY NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this study, we propose parameter-varying neural ordinary differential equations (NODEs) where the evolution of model parameters is represented by partition-of-unity networks (POUNets), a mixture of experts architecture. The proposed variant of NODEs, synthesized with POUNets, learn a meshfree partition of space and represent the evolution of ODE parameters using sets of polynomials associated to each partition. We demonstrate the effectiveness of the proposed method for three important tasks: data-driven dynamics modeling of (1) hybrid systems, (2) switching linear dynamical systems, and (3) latent dynamics for dynamical systems with varying external forcing.

1 INTRODUCTION

1.1 NEURAL ORDINARY DIFFERENTIAL EQUATIONS AND THEIR VARIANTS

Neural ordinary differential equations (NODEs) (Chen et al., 2018; Weinan, 2017; Haber & Ruthotto, 2017; Lu et al., 2018) are a class of continuous-depth neural network architectures that learn the dynamics of interest as a form of systems of ODEs:

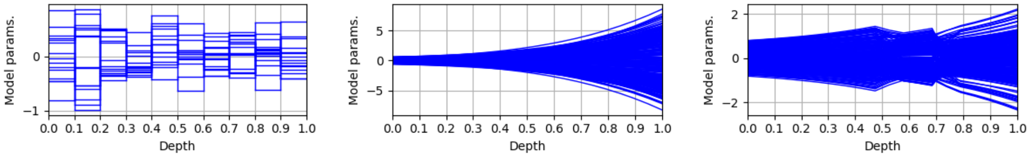
$$\frac{\mathrm{d}\boldsymbol{h}(s)}{\mathrm{d}s} = \boldsymbol{f}(\boldsymbol{h}(s); \Theta),$$

where $\boldsymbol{h}$ denotes a hidden state, s represents a continuous depth, and the velocity function $\boldsymbol{f}$ is parameterized by a feed-forward neural network with learnable model parameters Θ.

As pointed out in (Massaroli et al., 2020), the original NODE formulation (Chen et al., 2018), is limited to incorporate the depth variable s into dynamics as it is, e.g., by concatenating s and $\boldsymbol{h}$, which are then fed to $\boldsymbol{f}(\boldsymbol{h}, s; \Theta)$, rather than constructing the map $s \mapsto \Theta(s)$. Recent studies investigate strategies to extend NODEs to be depth-variant. ANODEV2 (Zhang et al., 2019) proposes a hypernetwork-type approach which builds a coupled system of NODEs, where one NODE defines an evolution of state variables, while another NODE defines an evolution of model parameters. In (Massaroli et al., 2020), stacked NODEs and Galerkin NODEs (GalNODEs) have been proposed where the evolution of model parameters are modeled as piecewise constants and a set of orthogonal basis, respectively. The idea of spectrally modeling model parameters has been further extended to enable basis transformation leading to stateful layers and compressible model parameters (Queiruga et al., 2021).

In this work, following the work by (Massaroli et al., 2020) which has proposed two depth-variant NODEs: stacked NODEs (i.e., a piecewise constant representation of model parameters, e.g., Figure 1a) and Galerkin NODEs (i.e., spectral representation of model parameters, e.g., Figure 1b). Inspired by these two variants, we propose an a combination of stacked and Galerkin NODEs leading to spectral-element-like (Patera, 1984) or hp-finite-element-like (Solin et al., 2003) methods, which we denote by Partition-of-Unity NODEs (POUNODEs, e.g., Figure 1c). We decompose the domain of model parameters (e.g., depth) into disjoint learnable partitions, with model parameters approximated on each as polynomials.

Our main contributions include 1) development of an hp-element-like method for representing the evolution of model parameters of NODEs and 2) to showcase the effectiveness of POUNODEs



(a) Stacked NODE (b) Galerkin NODE (c) POUNODE

Figure 1: An illustrative example depicting evolution of model parameters for Stacked NODE, Galerkin NODE, and the proposed POUNODE. Models are trained to perform the binary classification task of two concentric circles.

with three different important applications: learning hybrid systems, switching linear dynamics, and latent-dynamics modeling with varying external factor.

2 POUNETS INTO NODES

We begin by introducing partition-of-unity networks (POUNets) (Lee et al., 2021b), which a particular type of deep neural network developed for approximating functions with exponential convergence. POUNets automatically learn partitions of the domain and simultaneously compute the coefficients of polynomials associated in each partition. Then we introduce a method to use POUNets for representing the evolution of model parameters for NODEs.

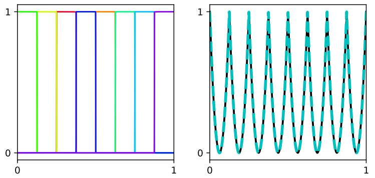
2.1 PARTITION-OF-UNITY NETWORKS

Several recent works (He et al., 2018; Yarotsky, 2017; 2018; Opschoor et al., 2019; Daubechies et al., 2019) on approximation theory of deep neural networks (DNNs) investigate the role of width and depth to the performance of DNNs and have theoretically proved the existence of model parameters of DNNs that emulate algebraic operations, a partition of unity (POU), and polynomials to exponential accuracy in the depth of the network. That is, in theory, with a sufficiently deep architecture, DNNs should be able to learn a spectrally convergent hp-element space by constructing a POU to localize polynomial approximation without a hand-tailored mesh. As has seen in (Fokina & Oseledets, 2019; Adcock & Dexter, 2020; Lee et al., 2021b), however, such convergent behaviours in practice are not realized due to many reasons (e.g., gradient-descent-based training). In (Lee et al., 2021b), a novel neural network architecture, POUNets, has been proposed, which explicitly incorporates a POU and polynomial elements into a neural network architecture, leading to exponentially-convergent DNNs.

Mathematically, a POU can be defined as $\Phi(x) = \{\phi_i(x)\}_{i=1}^{n_{\text{part}}}$ satisfying $\sum_i \phi_i(x) = 1$ and $\phi_i \leq 0$ for all x. Then POUNets can be represented as

$$y_{\text{POU}}(x) = \sum_{i=1}^{n_{\text{part}}} \phi_i(x;\pi) \sum_{j=1}^{\dim(V)} \alpha_{i,j}\phi_j(x),$$

where $V = \text{span}(\{\psi_j\})$, typically taken as the space of polynomials of order m, and $\Phi(x;\pi) = [\phi_1(x;\pi), \ldots, \phi_{n_{\text{part}}}(x;\pi)]$ is parameterized by a neural network with the model parameters π. To ensure the properties of the partition-of-unity, the output layer of the neural network Φ is designed to produce positive and normalized output (i.e., $\phi_i(x;\pi) \geq 0$ and $\sum_i \phi_i(x;\pi) = 1$). Figure 2 depicts an example of regressing a quadratic wave with a POUNet, where standard MLPs exhibit poor performance: the left panel shows the learned partitions and the right panel shows the ground truth target function (solid black) and the prediction (dashed cian). In each partition, a set of monomials with the maximal degree 2 is fitted optimally by solving local linear least-squares problems.



(a) Partitions (b) Quadratic wave

Figure 2: Learned partitions (left) and predictions (cian dashed) depicted with the ground truth target function (black solid).

2.2 Partition-of-unity-based neural ordinary differential equations

Now, we introduce the proposed partition-of-unity-based neural ordinary differential equations, where the model parameters are represented as a POUNet: $\Theta(\boldsymbol{s}) \in \mathbb{R}^{n_\Theta}$:

$$\Theta(\boldsymbol{s}; \alpha, \pi) = \sum_{i=1}^{n_{\text{part}}} \phi_i(\boldsymbol{s}; \pi) p_i(\boldsymbol{s}) = \sum_{i=1}^{n_{\text{part}}} \phi_i(\boldsymbol{s}; \pi) \sum_{j=1}^{n_{\text{poly}}} \alpha_{i,j} \psi_j(\boldsymbol{s}), \tag{1}$$

where $\boldsymbol{s}$ denotes a set of variables whose domains are expected to have a set of partitions (e.g., $\boldsymbol{s}$ can be a depth variable in depth-continuous neural network architectures), $\phi_i(\boldsymbol{s}; \pi) \in \mathbb{R}$ denotes a partition of unity network, parameterized by π, $\psi_j(\boldsymbol{s}) \in \mathbb{R}$ denotes a polynomial basis, and $\alpha_{\cdot,j} \in \mathbb{R}^{n_\Theta}$ denote the polynomial coefficients. Thus, collectively, there is a set of parameters $\alpha = (\alpha_1, \ldots, \alpha_{n_{\text{part}}})$ with $\alpha_i = [\alpha_{i,1} \cdots, \alpha_{i,n_{\text{poly}}}] \in \mathbb{R}^{n_\theta \times n_{\text{poly}}}$. In the following, we present a couple example cases of the types of the variables $\boldsymbol{s}$.

Temporally varying dynamics / depth variance As in the typical settings of NODEs, when an MLP is considered to parameterize the velocity function, $\boldsymbol{f}(\cdot; \Theta)$, the model parameters can be represented as a set of constant-valued variables, $\Theta = \{(W_\ell, \boldsymbol{b}_\ell)\}_{\ell=1}^{L}$, where W_ℓ and $\boldsymbol{b}_\ell$ denote weights and biases of the ℓ-th layer. As opposed to the depth-invariant NODE parameters Θ, POUNODEs represent depth-variant NODEs (or non-autonomous dynamical systems) by setting the model parameters as

$$\Theta(t) = \{(W_\ell(t), \boldsymbol{b}_\ell(t))\}_{\ell=1}^{L},$$

where t denotes the time variable or the depth of the neural network and represent, and by representing $\Theta(t)$ as a POUNet as in Eq. (1) with $\boldsymbol{s} = t$.

Spatially varying dynamics Another example dynamical systems that can be represented by POUNODEs is a class of dynamical systems whose dynamics modes are defined differently on different spatial regions. In this case, the model parameters can be set as spatially-varying ones:

$$\Theta(\boldsymbol{x}) = \{(W_\ell(\boldsymbol{x}), \boldsymbol{b}_\ell(\boldsymbol{x}))\}_{\ell=1}^{L}.$$

and can be represented as a POUNet as in Eq. (1) with $\boldsymbol{s} = \boldsymbol{x}$.

Remark 2.1. *Although not numerically tested in this study, the idea of representing the evolution of model parameters via POUNets can be applied to different neural network architectures, e.g., POU-Recurrent Neural Networks (POU-RNNs).*

3 Use cases

This section exhibits example use cases where the benefits of using POUNODE can be pronounced. All implementations are based on PyTorch (Paszke et al., 2019) and the TorchDiffeq library (Chen et al., 2018) for the NODEs capability.

For all following experiments, we consider a POUNet, $\Phi = \{\phi_i\}_{i=1}^{n_{\text{part}}}$, based on a radial basis function (RBF) network (Broomhead & Lowe, 1988; Billings & Zheng, 1995); for each partition, there is an associated RBF layer, defined by its center and shape parameter, and then the output of the RBF layers is normalized to satisfy the partition-of-unity property (refer to Appendix for more details).

3.1 System identification of a hybrid system

As a first set of use cases, we apply POUNODEs for data-driven dynamics modeling. In particular, we aim to learn a dynamics model for a hybrid system, where the different dynamics models are mixed in a single system: a system consisting of multiple smooth dynamical flows (SDFs), each of which is interrupted by sudden changes (e.g., jump discontinuities or distributional shifts) (Van Der Schaft & Schumacher, 2000).

Following (Shi & Morris, 2021), we are interested in modeling a hybrid system, where external factors exist and results in sudden changes in the dynamics modes, which makes the applications of traditional dynamics modeling approach challenging.

Again, similar to (Shi & Morris, 2021), as a benchmark, we consider the Lotka–Volterra (LV) equation:

$$\dot{x} = a(t)x - b(t)xy,$$
$$\dot{y} = d(t)xy - c(t)y,$$

where $(a(t), b(t), c(t), d(t))$ are time-varying ODE parameters that define the dynamics. As a system identification benchmark problem, we generate a trajectory consisting of four different dynamics (SDFs) as depicted in Figure 3. The ODE parameters are chosen to be piecewise constant and the values of the parameters are listed in Table 1. There are three change points at 35.85, 57.34, and 88.07 seconds, which are non-uniformly distributed over time.

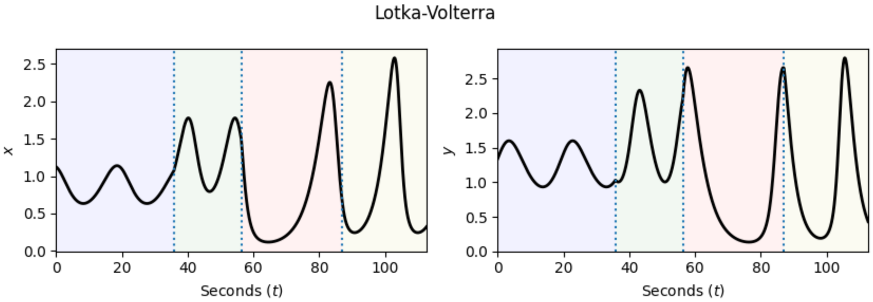


Figure 3: A trajectory of four different dynamics generated from solving the LV equation.

Now, we identify the system of the given trajectory using the proposed POUNODE. For the parameterization of the velocity function, we consider a dictionary-based approach:

$$f_{\Theta(t)}(x) = \left(\Phi(x)^{\mathsf{T}}\Xi(t)\right)^{\mathsf{T}}, \tag{2}$$

where $\Phi(x) \in \mathbb{R}^{p\times 1}$ denotes a vector of dictionaries and $\Xi(t) \in \mathbb{R}^{p\times n}$ denotes a trainable time-dependent coefficients (i.e., $\Theta(t) = \Xi(t)$). For the following experiments, we choose a set of polynomials as our dictionaries, i.e., $\Phi(x) = [1, x, x^2, xy, y, y^2]^{\mathsf{T}}$ and $p = 6$. The coefficients $\Theta(t)$ are modeled as a set of piecewise constant model parameters using POUNets such that

$$\Theta(t; \alpha, \pi) = \sum_{i=1}^{n_{\text{part}}} \phi_i(t; \pi)\left(\alpha_{i,1}\psi_1(t)\right) = \sum_{i=1}^{n_{\text{part}}} \alpha_i \phi_i(t; \pi).$$

That is, there is a set of constant coefficients associated with each partition, $\alpha_i \in \mathbb{R}^{p\times n}$, $i = 1, \ldots, n_{\text{part}}$. Note that the above equation is a special case of the expression in Eq. (1) with $n_{\text{poly}} = 1$.

t (seconds)	[0, 35.85]	[35.86, 57.34]	[57.35,88.07]	[88.08,113.68]
Ground truth	$\dot{x} = 0.3543x - 0.2867xy$ $\dot{y} = 0.3492xy - 0.3011y$	$\dot{x} = 0.4301x - 0.2731xy$ $\dot{y} = 0.3847xy - 0.4695y$	$\dot{x} = 0.2500xy - 0.2966y$ $\dot{y} = 0.3548xy - 0.2568y$	$\dot{x} = 0.3256x - 0.3364xy$ $\dot{y} = 0.4213xy - 0.4176y$
POUNODE ($n_{\text{part}} = 4$)	$\dot{x} = 0.3604x - 0.2895xy$ $\dot{y} = 0.3447xy - 0.2950y$	$\dot{x} = 0.4334x - 0.2754xy$ $\dot{y} = 0.3822xy - 0.4612y$	$\dot{x} = 0.2500x - 0.2950xy$ $\dot{y} = 0.3532xy - 0.2565y$	$\dot{x} = 0.3285x - 0.3384xy$ $\dot{y} = 0.4134xy - 0.4110y$
POUNODE ($n_{\text{part}} = 8$)	$\dot{x} = 0.3530x - 0.2856xy$ $\dot{y} = 0.3512xy - 0.3007y$	$\dot{x} = 0.4353x - 0.2712xy$ $\dot{y} = 0.3829xy - 0.4676y$	$\dot{x} = 0.2508x - 0.2958xy$ $\dot{y} = 0.3561xy - 0.2576y$	$\dot{x} = 0.3225x - 0.3342xy$ $\dot{y} = 0.4213xy - 0.4154y$

Table 1: A hybrid Lotka–Volterra system consisting of four different dynamics. The coefficients for each dynamics of the ground truth system, and learned systems are listed.

For training, we use the training algorithm proposed in the work of sparse nonlinear dynamics identification method (Lee et al., 2021a). The essence is that a sparsity promoting L1 penalty (or L1 weight decay (LASSO) (Tibshirani, 1996)), is applied to the weight $\alpha = [\alpha_1, \ldots, \alpha_{n_{\text{part}}}]$ and an element of the weight whose magnitude is smaller than a certain threshold is pruned over the course of gradient-based training. We leave the details in Appendix. As we use the zero initialization (i.e., all elements of α_i are set to zero), we do not repeat the same experiments.

Table 1 reports the coefficients for the ground-truth systems (the second row) and the coefficients identified by using the proposed methods: POUNODE ($n_{\text{part}} = 4$), and POUNODE ($n_{\text{part}} = 8$). POUNODE ($n_{\text{part}} = N$) indicates that the model starts with N partitions; some of them are expected to vanish as training proceeds, e.g., the right panel in Figure 4. Figure 4 also depicts the trajectory of the learned dynamics (dashed cyan color on two left panels) that is almost overlapped with the ground-truth trajectory and the learned partitions (on the right panel).

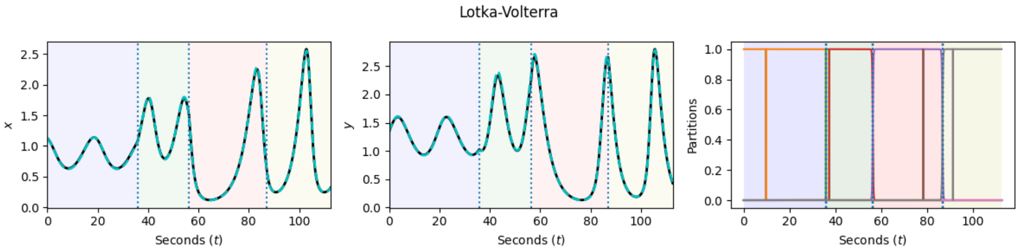


Figure 4: A trajectory (dashed cyan color on two left panels) generated by solving learned dynamics model with eight beginning partitions ($n_{\text{part}} = 8$) and the learned partitions (right).

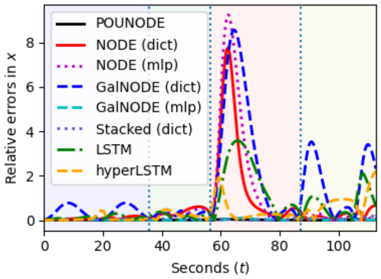


Figure 5: Relative errors in predictions

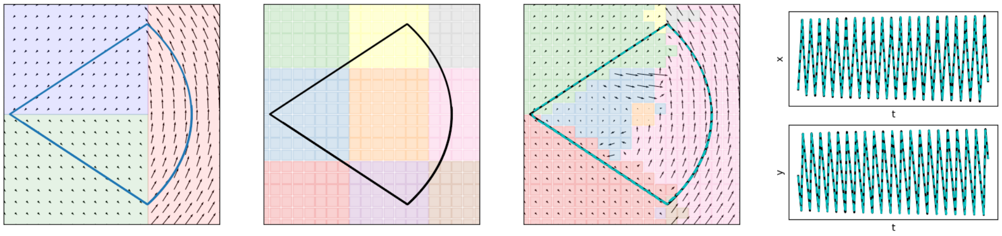
Although the main objective of the system identification is to discover an interpretable model (via imposing strong inductive biases such as the choice of dictionaries), we can also see how well the model fits to the data. As a baseline for comparison, we test RNN-LSTM, RNN-GRU, hyperLSTM (Ha et al., 2017), NODE ($n_{\text{part}} = 1$, $n_{\text{poly}} = 1$), StackedNODE, GalNODE ($n_{\text{part}} = 1$, $n_{\text{poly}} = 3$), and ANODEv2(Gholami et al., 2019)[1]. Here, we employ the same dictionary-based parameterization (Eq. (2)) for NODE, StackedNODE and GalNODE as well as the "black-box" MLP parameterization for NODE and GalNODE. For MLP, we consider 4 layers with 25 neurons in each layer. Figure 5 shows the time-instantaneous relative error of the trajectory of $x(t)$, i.e., $e(t) = \frac{|x(t) - \tilde{x}(t)|}{|x(t)|}$, where $\tilde{x}(t)$ denotes the predictions and $|\cdot|$ denotes an absolute value. As Figure 5 shows POUNODE outperforms other baseline approaches in terms of accuracy and have comparable accuracy with Stacked NODE (dictionary-based, 8 partitions) and GalNODE (mlp): the relative errors measured in L2-norm are 0.0160, 0.0615, and 0.0264 for POUNODE, StackedNODE and GalNODE, respectively. StackedNODE, however, is not capable of pinpointing the change points, and GalNODE requires a much larger number of model parameters ($\times 50$ more parameters, compared to POUNODE).

3.2 Switching linear dynamical systems

As a next set of use cases, we consider switching linear dynamical systems (SLDS) consisting of multiple sequences of simple dynamical modes that change the dynamics mode based on a discrete switch (Ackerson & Fu, 1970; Chang & Athans, 1978; Ghahramani & Hinton; Fox et al., 2008; Linderman et al., 2016). We are interested in data-driven dynamics modeling of SLDS in the continuous-time setting as has considered in (Chen et al., 2020), taking the ground-truth dynamics as described in Table 2. At the boundary of each spatial subdomain (as depicted in Figure 6a), the dynamics changes instantaneously and, thus, the resulting dynamics consists of sequences of different dynamics and can exhibit discontinuities at the moment of switching.

This benchmark problem considered is an SLDS example of a particle moving around a fan-shaped synthetic race track as in (Chen et al., 2020), which has been originally adapted from (Linderman et al., 2016). Figure 6 (left) depicts an example of the ground-truth trajectory and the vector field and the analytical expression of the ODEs can be found in Table 2.

[1] The results of RNN-GRU and ANODEv2 were not reported as the both models did not seem to be trained well under the experimental configuration used for training the proposed method and other baselines.



(a) Ground-truth (b) Initial partitions (c) POUNODE (d) Trajectories over time

Figure 6: A ground-truth trajectory with the ground-truth vector field (left), a computed trajectory computed from the learned vector field (right). Different colors indicate the different partitions.

Coordinates (x, y)	$x < 2, y < 0$	$x < 2, y \geq 0$	$x \geq 2$
Ground truth	$\dot{x} = 1$ $\dot{y} = -1$	$\dot{x} = -1$ $\dot{y} = -1$	$\dot{x} = -y$ $\dot{y} = x + 2$
Partitions	Top-left (green) partition	Bottom-left (red) partition	Right (pink) partition
POUNODE	$\dot{x} = 0.9980$ $\dot{y} = -1.0000$	$\dot{x} = -1.0033$ $\dot{y} = -0.9965$	$\dot{x} = -1.0000y$ $\dot{y} = 0.9977x + 1.9968$

Table 2: A switching linear dynamical system: The ODEs of each dynamics of the ground truth system, and learned systems are listed.

Our goal is to utilize POUNODE to model an SLDS by treating SLDS parameters as parameters that are dependent on the spatial coordinates $\boldsymbol{x}$. That is, we consider a time-continuous model

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}_{\Theta(\boldsymbol{x})}(\boldsymbol{x}) = C(\boldsymbol{x})\boldsymbol{x} + \boldsymbol{d}(\boldsymbol{x}),$$

where $C(\boldsymbol{x})$ and $\boldsymbol{d}(\boldsymbol{x})$ are model parameters, $\Theta(\boldsymbol{x}) = [C(\boldsymbol{x}), \boldsymbol{d}(\boldsymbol{x})] \in \mathbb{R}^{2\times 3}$, that are piecewise constant on each partition:

$$\Theta(\boldsymbol{x}; \alpha, \pi) = \sum_{i=1}^{n_{\text{part}}} \alpha_i \phi_i(\boldsymbol{x}; \pi),$$

where $\alpha_i \in \mathbb{R}^{2\times 3}$ denotes the i-th coefficients defined on the i-th partition, ϕ_i. Our intention is to learn the three disjoint spatial regions as disjoint partitions and the associated piece-wise constant coefficients to correctly identify the vector field.

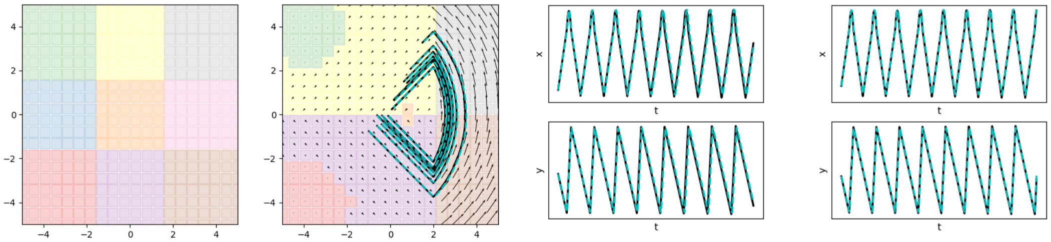
For training, we again use the same algorithm, proposed in (Lee et al., 2021a), which we summarize in Appendix. For the system identification task, we use a single trajectory to train the model.

Figures 6b–6d show the initial 3×3 partitions (Figure 6b), the learned partitions and the trajectory produced by solving the learned dynamics model (Figure 6c), and the trajectories of each state variable (Figure 6d). Table 2 reports the identified systems in each region. As reported in Figures 6b–6d and Table 2, POUNODE successfully identify the benchmark SLDS with the errors in the third/fourth most significant digits.

Figure 7 reports the results of learning dynamics using multiple trajectories and applying the learned dynamics in the predictive setting. For this experiments, we have generated 80 training, 10 validation, and 10 test trajectories with varying initial conditions. Figures 7b–7d depict the ground-truth trajectories (solid black) and the trajectories computed from the learned dynamics model (dashed cyan). Figure 7b shows that there are four remaining partitions, where the learned coefficients are as follows:

(purple partition)	(yellow partition)	(gray partition)	(brown partition)
$\dot{x} = 0.9992,$ $\dot{y} = -0.9982,$	$\dot{x} = -0.9997,$ $\dot{y} = -1.0010,$	$\dot{x} = -0.9995y,$ $\dot{y} = 0.9966x + 2.0034,$	$\dot{x} = -0.9985y,$ $\dot{y} = 0.9962x + 2.0217.$

Compared to the approach where the method learns a differential event function (Chen et al., 2020), the proposed approach directly learns the vector fields that are differently defined in each spatial



(a) Initial partitions

(b) POUNODE

(c) Test trajectory #1

(d) Test trajectory #2

Figure 7: Ground-truth trajectories (solid black) and trajectories computed from the learned vector field (dashed cyan) are depicted. Different colors in background indicate the different partitions: initial partitions (Figure 7a) and learned partitions (Figure 7b).

domain and, thus, does not require to specify in advance how many events of switching dynamics will happen in the simulation run.

3.3 Latent dynamics modeling (reduced-order modeling)

The next use case is a latent-dynamics modeling in the context of reduced-order modeling (ROM), a computational framework that is widely investigated in the field of computational science and engineering (Fulton et al., 2019; Lee & Carlberg, 2020; 2021). The main goal of developing ROMs is to provide a means to perform rapid simulations of complex physical phenomena (typically described in partial differential equations) to support time-critical applications such as control.

As elaborated in (Lee & Carlberg, 2021), a latent-dynamics modeling requires two main components: 1) an *embedding*, i.e., a nonlinear mapping between high-dimensional dynamical-system states and low-dimensional latent states, and (2) a dynamics model, i.e., the time evolution model of the latent states. For learning an embedding, nearly all traditional numerical methods seek a linear embedding, which is typically defined by principal component analysis, or "proper orthogonal decomposition" (POD) (Holmes et al., 2012), performed on measurements of the high-dimensional states. Recent approaches, on the other hand, explore the use of deep neural networks, (autoencoders (Hinton & Salakhutdinov, 2006), in particular), to build a nonlinear embedding (Morton et al., 2018; Wiewel et al., 2019; Fulton et al., 2019; Lee & Carlberg, 2020; 2021). After learning the embedding, a (nonlinear) latent-dynamics model is constructed, representatively, via long short-term memory (Hochreiter & Schmidhuber, 1997), Koopman operators (Li et al., 2019; Azencot et al., 2020), and NODEs (Chen et al., 2018; Lee & Parish, 2020).

In the following experiment, we choose a linear embedding, defined by a POD basis matrix, $\varphi \in \mathbb{R}^{N \times p}$, where N and p denote the dimensions of the high dimensional space and the latent space. The encoding and the decoding are defined as $\boldsymbol{h} = \varphi \boldsymbol{x}$ and $\boldsymbol{x} = \varphi^{\mathsf{T}} \boldsymbol{h}$, where $\boldsymbol{x} \in \mathbb{R}^N$ and $\boldsymbol{h} \in \mathbb{R}^p$. Given the linear encoder and the decoder, we learn the latent dynamics with the proposed POUNODE. As Figure 8 illustrates that an initial high-dimensional state is encoded into the latent initial state, future latent states are computed via the forward pass of POUNODE, and the high-dimensional approximate states are computed via the decoder.

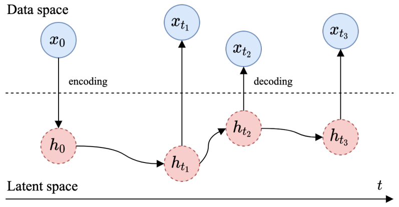


Figure 8: A latent-dynamics model

As a benchmark problem, we consider 1-dimensional inviscid Burgers' equation with a parameterized forcing term; setting different values to the parameter change the dynamics. We generate a 35-seconds-long trajectory that has two change points at $t = [13.4, 22.1]$ seconds, where the value of the forcing parameter changes. We test the latent-dynamics modeling in a reconstructive setting with a single trajectory and we set the original data dimension to be $N = 256$ and the latent dimension to be $p = 3$.

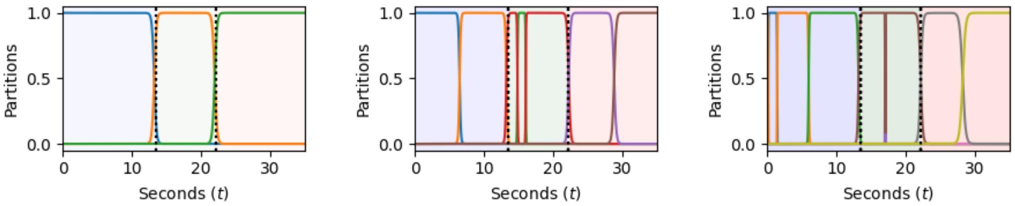
The velocity function is parameterized as an MLP with 2 hidden layers, 25 neurons in each layer, and the hyperbolic Tangent nonlinearity such that $\boldsymbol{f} = W^{(3)}\sigma(W^{(2)}\sigma(W^{(1)}\boldsymbol{h} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}) + \boldsymbol{b}^{(3)}$, where the weights and the biases are function of time $\Theta(t) = \{(W^{(\ell)}(t), \boldsymbol{b}^{(\ell)}(t)\}_{\ell=1}^{3}$, which are modeled as a POUNet (Eq. (1)).

Table 3 reports relative errors in the $L2$-norm, $\frac{\|X-\tilde{X}\|_\mathsf{F}}{\|X\|_\mathsf{F}}$, where $X, \tilde{X} \in \mathbb{R}^{N \times n_{\text{seq}}}$ denote the ground-truth solution measurements and the predicted solutions. POUNODE with $n_{\text{part}} = 1$ and n_{poly} is equivalent to NODE and POUNODE with $n_{\text{part}} = 1$ and $n_{\text{poly}} = 3$ is conceptually same as the GalNODE. We also observed that setting $n_{\text{part}} \geq 6$ does not improve the performance significantly. As baselines of comparisons, we assess the performance of RNN-LSTM, RNN-GRU, HyperLSTM (Ha et al., 2017), ANODEv2 (Gholami et al., 2019), and Stacked NDOEs (with 6 fixed partitions), of which results are reported in Table 3. For each model, we repeat perform 5 runs of experiments with different random seeds. The details of the neural network architecture and hyperparameter choices are in Appendix.

Table 3: Performance

Models	Accuracy
RNN-LSTM	0.4367 ± 0.08620
RNN-GRU	0.3035 ± 0.06128
HyperLSTM (Ha et al., 2017)	0.2244 ± 0.06812
NODE ($n_{\text{part}} = 1, n_{\text{poly}} = 1$)	0.2981 ± 0.00275
ANODEv2 (Gholami et al., 2019)	0.3589 ± 0.08057
GalNODE ($n_{\text{part}} = 1, n_{\text{poly}} = 3$)	0.4783 ± 0.15414
StackedNODE ($n_{\text{part}} = 6$, fixed)	0.3659 ± 0.00472
POUNODE ($n_{\text{part}} = 3, n_{\text{poly}} = 1$)	0.1147 ± 0.00072
POUNODE ($n_{\text{part}} = 6, n_{\text{poly}} = 1$)	0.0731 ± 0.00286
POUNODE ($n_{\text{part}} = 9, n_{\text{poly}} = 1$)	0.0730 ± 0.00057

Figure 9 illustrates the ground truth change points (the black dashed vertical lines), where in between the forcing term remain the same (the regions highlighted with different colors), and the learned partitions. The partitions are learned to have disjoint sections that do not cross over the change points, and some of the unnecessary partitions are eliminated. Figure 10 depicts the ground-truth solution snapshots (solid blue) and the approximated solution snapshots (dashed red) for varying latent dynamics models with $n_{\text{part}} = \{1, 3, 9\}$. The approximate solutions are smooth as they are represented as linear combinations of three principal basis ($\varphi \in \mathbb{R}^{N \times p}$ with $p = 3$), however the approximate solutions that are generated with the latent dynamics models ($n_{\text{part}} \geq 3$) shows that they can match the shock locations (i.e., a place of the discontinuity in each snapshot).



(a) $n_{\text{part}} = 3$ (b) $n_{\text{part}} = 6$ (c) $n_{\text{part}} = 9$

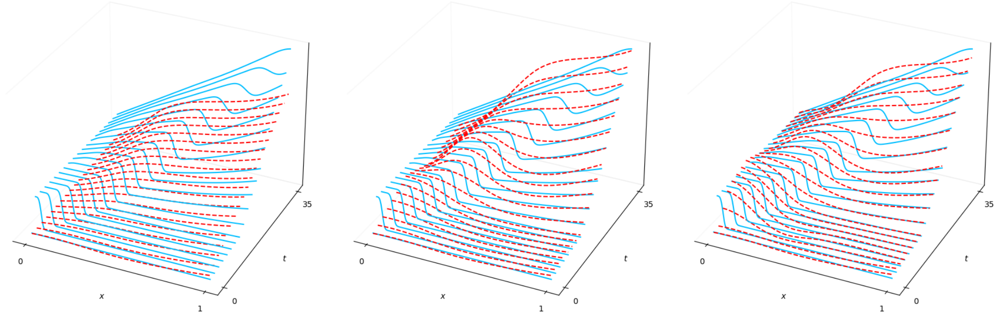
Figure 9: Learned partitions for the latent-dynamics models. The subcaption, $n_{\text{part}} = k$, indicates the number of the beginning partitions.

4 Discussion

4.1 Limitations and future directions

Minibatching Minibatching trajectories with different change points requires multiple POUNets, where each POUNet needs to be a realization of an input-data dependent POUNet, i.e., $\Theta(\boldsymbol{s}, \boldsymbol{x}^{\text{(input)}}; \alpha, \pi)$, where $\boldsymbol{x}^{\text{(input)}}$ denotes the input data. An approach similar to data-controlled NODEs proposed in (Massaroli et al., 2020) can be extended to be equipped with POUNets such that $\frac{\mathrm{d}\boldsymbol{h}(t)}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{h}(t), \boldsymbol{x}^{\text{(input)}}; \Theta(\boldsymbol{s}, \boldsymbol{x}^{\text{(input)}}))$.

Predictive tasks As shown in Sections 3.1 and 3.3, the proposed POUNODEs has demonstrated their effectiveness for identifying or building a surrogate model for a hybrid system. As demon-



(a) $n_{\text{part}} = 1$

(b) $n_{\text{part}} = 3$

(c) $n_{\text{part}} = 9$

Figure 10: The ground-truth solution snapshots (solid blue) and the approximated solution snapshots (dashed red) for varying latent dynamics models with $n_{\text{part}} = \{1, 3, 9\}$.

strated in (Shi & Morris, 2021), the proposed models can be used for extrapolation in a somewhat limited scenario, where the dynamical mode remain unchanged. To be useful in the extrapolation settings, where the future dynamics is under temporal drift, alternative approaches that learn to produce future model parameters based on past sets of model parameters (e.g., hypernetwork-based approaches (Ha et al., 2017)) or that exploit hierarchical structures of time-series for forecasting (e.g., N-BEATS (Oreshkin et al., 2019)).

Fast optimizer In the original work of POUNets (Lee et al., 2021b), a fast optimizer, which alternates between gradient descent updates for updating partition parameters and least-squares solves for computing optimal polynomial coefficients, has been proposed for solving polynomial regression problems and has demonstrated the faster convergence. Thus, as opposed to the gradient-descent-based optimizer used in this work, which updates all model parameters simultaneously, developing an optimizer tailored to POUNODE would allow faster and more accurate training.

POUNODEs as general NN architectures As in previous work (Zhang et al., 2019; Massaroli et al., 2020), the depth-variant neural ODEs have demonstrated increased performance in other downstream tasks, e.g., image classification. We have tested POUNODEs, where convolutional kernels are spectrally represented, for image classification with CIFAR-10 by using the same setting considered in (Dupont et al., 2019; Massaroli et al., 2020) and observe only marginal improvements (1$\sim$2% increase in the test accuracy, but with the increase in the number of function evaluations). We expect that the benefits of using POUNODEs can be more pronounced in more complex settings, e.g., replacing multiple ResBlocks in ResNet-151 (He et al., 2016) with a small number of POUNODE-Blocks, and plan to further investigate the performance of POUNODEs in those settings.

5 Conclusion

In this study, we have introduced a new variant of NODEs (POUNODEs) with evolving model parameters, where the evolution is modeled by using partition-of-unity networks. We have demonstrated the effective of the proposed POUNODEs with three important case studies: learning hybrid dynamical systems, switching linear dynamics, and latent dynamics modeling with varying external factors. In those use-cases, we have demonstrated that the POUNODEs are very effective and outperform the baselines including the previous depth-variant NODEs and hypernetwork-based LSTMs.

6 Reproducibility statement

The code will be publicly released upon acceptance and the hyperparameters to reproduce the results shown in the manuscript will be provided.

REFERENCES

G Ackerson and K Fu. On state estimation in switching environments. *IEEE transactions on automatic control*, 15(1):10–17, 1970.

Ben Adcock and Nick Dexter. The gap between theory and practice in function approximation with deep neural networks. *arXiv preprint arXiv:2001.07523*, 2020.

Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael Mahoney. Forecasting sequential data using consistent Koopman autoencoders. In *International Conference on Machine Learning*, pp. 475–485. PMLR, 2020.

Steve A Billings and Guang L Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8(6):877–890, 1995.

David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

Chaw-Bing Chang and Michael Athans. State estimation for discrete systems with switching parameters. *IEEE Transactions on Aerospace and Electronic Systems*, (3):418–425, 1978.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. In *International Conference on Learning Representations*, 2020.

Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) ReLU networks. *arXiv preprint arXiv:1905.02199*, 2019.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf`.

Daria Fokina and Ivan Oseledets. Growing axons: greedy learning of neural networks with application to function approximation. *arXiv preprint arXiv:1910.12686*, 2019.

Emily Fox, Erik Sudderth, Michael Jordan, and Alan Willsky. Nonparametric Bayesian learning of switching linear dynamical systems. *Advances in neural information processing systems*, 21, 2008.

Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. Latent-space dynamics for reduced deformable simulation. In *Computer graphics forum*, volume 38, pp. 379–391. Wiley Online Library, 2019.

Zoubin Ghahramani and Geoffrey Hinton. Switching state-space models. Technical report, University of Toronto.

Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=rkpACe1lx`.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Philip Holmes, John L Lumley, Gahl Berkooz, and Clarence W Rowley. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

Kookjin Lee and Kevin T Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

Kookjin Lee and Kevin T Carlberg. Deep conservation: A latent-dynamics model for exact satisfaction of physical conservation laws. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 277–285, 2021.

Kookjin Lee and Eric J Parish. Parameterized neural ordinary differential equations: Applications to computational physics problems. *arXiv preprint arXiv:2010.14685*, 2020.

Kookjin Lee, Nathaniel Trask, and Panos Stinis. Structure-preserving sparse identification of nonlinear dynamics for data-driven modeling. *arXiv preprint arXiv:2109.05364*, 2021a.

Kookjin Lee, Nathaniel A Trask, Ravi G Patel, Mamikon A Gulian, and Eric C Cyr. Partition of unity networks: Deep hp-approximation. *arXiv preprint arXiv:2101.11256*, 2021b.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional Koopman operators for model-based control. In *International Conference on Learning Representations*, 2019.

Scott W Linderman, Andrew C Miller, Ryan P Adams, David M Blei, Liam Paninski, and Matthew J Johnson. Recurrent switching linear dynamical systems. *arXiv preprint arXiv:1610.08466*, 2016.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pp. 3276–3285. PMLR, 2018.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asma. Dissecting neural odes. In *34th Conference on Neural Information Processing Systems, NeurIPS 2020*. The Neural Information Processing Systems, 2020.

Jeremy Morton, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden. Deep dynamical modeling and control of unsteady fluid flows. *Advances in Neural Information Processing Systems*, 31, 2018.

Joost AA Opschoor, Philipp Petersen, and Christoph Schwab. Deep ReLU networks and high-order finite element methods. *SAM, ETH Zürich*, 2019.

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

Anthony T Patera. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *Journal of computational Physics*, 54(3):468–488, 1984.

Alejandro Queiruga, N Benjamin Erichson, Liam Hodgkinson, and Michael W Mahoney. Stateful ode-nets using basis function expansions. *Advances in Neural Information Processing Systems*, 34:21770–21781, 2021.

Ruian Shi and Quaid Morris. Segmenting hybrid trajectories using latent odes. In *International Conference on Machine Learning*, pp. 9569–9579. PMLR, 2021.

Pavel Solin, Karel Segeth, and Ivo Dolezel. *Higher-order Finite Element Methods*. Chapman and Hall/CRC, 2003.

Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Arjan J Van Der Schaft and Johannes Maria Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251. Springer London, 2000.

E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, volume 38, pp. 71–82. Wiley Online Library, 2019.

Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017.

Dmitry Yarotsky. Optimal approximation of continuous functions by very deep ReLU networks. *arXiv preprint arXiv:1802.03620*, 2018.

Tianjun Zhang, Zhewei Yao, Amir Gholami, Kurt Keutzer, Joseph Gonzalez, George Biros, and Michael Mahoney. Anodev2: A coupled neural ode evolution framework. *arXiv preprint arXiv:1906.04596*, 2019.

A Training algorithms

For the system identification, where the dictionary-based parameterization of the right-hand side of ODEs is employed (in Sections 3.1–3.2), we use the neural ODE-based sparse nonlinear dynamics identification method (i.e., nerual SINDy) developed in (Lee et al., 2021a).

During the training, the model takes the forward pass by solving initial value problems as in neural ODEs. Then, as a training objective, the $L1$-distance between the data and the prediction is minimized. In addition, to promote the sparsity of the coefficients matrix Ξ, the elements of Ξ is penalized with the $L1$-penalty:

$$L = \frac{1}{n_b n_{\text{seq}}} \sum_{j=1}^{n_b} \sum_{i=1}^{n_{\text{seq}}} \left| \boldsymbol{x}_i^{(j)} - \tilde{\boldsymbol{x}}_i^{(j)} \right| + \lambda \|\Xi\|_1,$$

where n_b and n_{seq} denote the size of a minibatch and the length of sequences in the minibatch, $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ denote the data and the prediction, and λ is the penalty weight, which is set as 10^{-4}.

In addition, over the course of training, neural SINDy prunes the coefficients based on their absolute magnitude with a certain threshold, τ:

$$[\Xi]_{kl} = 0 \quad \text{if} \quad |[\Xi]_{kl}| < \tau.$$

We set $\tau = 10^{-6}$ for all experiments. Pruning is applied to learned Ξ every feeding 100 minibatches.

B RADIAL-BASIS-FUNCTION-BASED POUNETS

For all experiments, we use the simple RBF-based POUNets as follows (in a one-dimension case):

$$\phi_i(x) = \frac{\exp\left(-\frac{|x-c^{(i)}|}{b^{(i)}}\right)}{\sum_k \exp\left(-\frac{|x-c^{(k)}|}{b^{(k)}}\right)},$$

where $\{(c^{(i)}, b^{(i)}\}_{i=1}^{n_{\text{part}}}$ is a set of learnable parameters. Here, $c^{(i)}$ and $b^{(i)}$ denote the center and the bandwidth of the RBF, respectively.

The centers are initialized to be on a uniform grid of the spatial domain.

C MODEL ARCHITECTURES AND HYPERPARAMETERS

System identification of a hybrid system experiments

- Model architectures
  - (Dictionary-based) NODE, StackedNODE, GalNODE, POUNODE: a single layer that linearly combines the output of the dictionaries $\Phi(x) = [1, x, x^2, xy, y, y^2]$
  - (MLP-based) NODE, GalNODE : 4 layers with 25 neurons and Tanh activation
  - LSTM: 4 stacked LSTM cells with 25 neurons for hidden and cell states
  - hyperLSTM: 4 stacked LSTM and hyperLSTM cells with 25 neurons for hidden, cell states and 25 neurons for hyper and embedding units
- Hyperparameters
  - Learning rate: 0.01
  - Max epoch: 3000
  - Batch size: 1 (50 for hyperLSTM)
  - Batched subsequence length: 100
  - ODE integrator: Dormand–Prince (dopri5) (Dormand & Prince, 1980) with the relative tolerance 10^{-7} and the absolute tolerance 10^{-9}

Learning switching linear dynamical systems

- Model architectures
  - POUNODE: a single layer that linearly combines the output of dictionaries, $\Phi(x) = [1, x, y]$
- Hyperparameters
  - Learning rate: 0.01
  - Max epoch: 3000
  - Batch size: 1
  - Batched subsequence length: 100
  - ODE integrator: Runge–Kutta of order 4

Latent dynamics modeling

- Model architectures
  - (MLP-based) NODE, StackedNODE, GalNODE, POUNODE : 2 layers with 25 neurons and Tanh activation
  - ANODEv2: 2 layers with 24 neurons and Tanh for the main NODEs, 2 layers with 50 neurons and Tanh for the weight NODEs
  - LSTM: 2 stacked LSTM cells with 25 neurons for hidden and cell states
  - hyperLSTM: 2 stacked LSTM and hyperLSTM cells with 25 neurons for hidden, cell states and 50 neurons for hyper and embedding units

- Hyperparameters
  - Learning rate: 0.01
  - Max epoch: 500
  - Batch size: 1
  - Batched subsequence length: 50
  - ODE integrator: Dormand–Prince (dopri5) (Dormand & Prince, 1980) with the relative tolerance 10^{-7} and the absolute tolerance 10^{-9}

D 1D INVISCID BURGERS' EQUATION

As a benchmark problem for reduced-order modeling (shown in Section 3.3 latent-dynamics modeling), we consider 1-dimensional inviscid Burgers' equation, which is defined as,

$$\begin{aligned} \frac{\partial w(x,t;\mu)}{\partial t} + \frac{\partial f(w(x,t;\mu))}{\partial x} &= 0.02e^{\mu x}, \qquad \forall x \in [0,100],\, \forall t \in [0,T] \\ w(0,t;\mu) &= 4.5, \qquad \forall t \in [0,T] \\ w(x,0) &= 1, \qquad \forall x \in [0,100], \end{aligned}$$

where μ defines the the forcing term, and we set $\mu = \mu(t)$ to be a time dependent function. In the high-fidelity simulation, we set $\mu(t)$ to be a piecewise-constant function over time such that

$$\mu(t) = \begin{cases} 0.005 & \text{if } t \leq 13.4 \\ 0.015 & \text{if } 13.4 < t \leq 22.2 \\ 0.025 & \text{if } t > 22.2 \end{cases}$$

For the discretization, we apply Godunov's scheme with 256 control volumes (i.e., $N = 256$ degrees of freedom) and the backward-Euler scheme with 600 uniform time steps.