Noname manuscript No.
(will be inserted by the editor)

Learning POD of Complex Dynamics Using Heavy-ball Neural ODEs

Justin Baker Elena Cherkaev Akil Narayan Bao Wang

Received: date / Accepted: date

Abstract Proper orthogonal decomposition (POD) allows reduced-order modeling of complex dynamical systems at a substantial level, while maintaining a high degree of accuracy in modeling the underlying dynamical systems. Advances in machine learning algorithms enable learning POD-based dynamics from data and making accurate and fast predictions of dynamical systems. In this paper, we leverage the recently proposed heavy-ball neural ODEs (HBNODEs) [Xia et al. NeurIPS, 2021] for learning data-driven reduced-order models (ROMs) in the POD context, in particular, for learning dynamics of time-varying coefficients generated by the POD analysis on training snapshots generated from solving full order models. HBNODE enjoys several practical advantages for learning POD-based ROMs with theoretical guarantees, includ-ing 1) HBNODE can learn long-term dependencies effectively from sequential observations and 2) HBNODE is computationally efficient in both training and testing. We compare HBNODE with other popular ROMs on several complex dynamical systems, including the von Kármán Street flow, the Kurganov-Petrova-Popov equation, and the one-dimensional Euler equations for fluids modeling.

Keywords Neural ODE Momentum Reduced-order modeling Deep learning

Mathematics Subject Classification (2020) 65P99 68T07

J. Baker, E. Cherkaev
Department of Mathematics
University of Utah
E-mail: baker@math.utah.edu, E-mail: elena@math.utah.edu

A. Narayan, B. Wang
Department of Mathematics, and Scientific Computing and Imaging (SCI) Institute
University of Utah
E-mail: akil@sci.utah.edu, E-mail: bwang@math.utah.edu

1 Introduction

Numerical long-time simulation of full-order models (FOMs) of complex dynamical systems is computationally costly. This is particularly true for physical systems that contain a wide range of spatial and temporal scales, including direct numerical simulation (DNS) [37] or large eddy simulation (LES) in fluid mechanics [17, 60, 12] and chaotic systems [27, 52, 53]. Reduced-order models (ROMs) have been utilized as alternative scientific simulation tools, which are computationally much more efficient than FOMs and retain comparable accuracy for simulating complex dynamical systems. ROMs have played crucial roles in designing, optimizing, and controlling dynamical systems [18, 1, 2, 5].

Several data-driven numerical algorithms have been proposed for reduced-order modeling, including dynamic mode decomposition (DMD) [51] and proper orthogonal decomposition (POD) [6]. These models leverage some FOM simulation data to construct low-dimensional simplified models that describe the underlying dynamics, with the goal of using these simplified models in generalization regimes to predict the unseen dynamics. Classical projection-based reduced-order modeling techniques (of which DMD and POD are examples) are among the most popular approaches for constructing ROMs of dynamical systems. This approach transforms the simulation results of FOM into a suitable low-dimensional subspace that preserves the largest variance of the training data. In, e.g., POD, classical numerical algorithms (such as Galerkin methods), are subsequently used to rewrite the state variable in the governing equation of the underlying dynamics into a system of ODEs, resulting in a substantially reduced degree of freedom for describing the complex dynamics. Both DMD and POD have been widely used in scientific simulations, particularly for fluid simulations.

ROMs generated from projection-based approaches can preserve crucial physical structures of the dynamics system. However, inappropriate truncation of the POD modes in governing equations can severely degrade modeling accuracy and result in unexpected, unphysical predictive results. Moreover, the precise strategy for mode truncation is task-dependent and is typically limited to explicit and closed definitions of the mathematical models [50]. Another drawback of direct projection-based approaches is that they require knowledge of governing equations that model the dynamical system, and this information is often absent for real-world problems. As such, data-driven reduced-order modeling has drawn significant recent attention. For instance, the learning of closure models to compensate for information loss due to mode truncation [49, 48, 38, 36], and data-driven reduced basis representations have been learned from simulation data that provides significantly improved predictive performance of the dynamics compared to classical models [39, 31, 58]. More recently, “vanilla” versions of machine learning approaches such as neural ODEs (NODEs) and recurrent neural networks (RNNs) have been used to learn temporal coefficients of the POD of a given complex dynamical system [45, 24, 25].

1.1 Our contribution

We employ the recently developed heavy-ball neural ODE (HBNODE) [59], an extension of NODE [9], to learn the temporal coefficients of the POD of complex physical systems with a focus on time-dependent simulations in scientific computing. In particular, our examples include the von Kármán Street (VKS) flow, the Kurganov-Petrova-Popov (KPP) equation, and the one-dimensional Euler equations for fluids modeling. There are three major advantages of learning PODs using HBNODEs over the existing deep learning approaches:

- HBNODEs are a class of continuous-depth neural networks, and they are suitable for learning irregularly-sampled simulation data or physical observations. Hence, observation protocols that entail missing or sparse data are easily tackled in this framework.
- Certain spectral properties of HBNODEs enable them to capture long-term dependencies from sequential data, which is crucial for learning PODs of complex dynamics.
- Both HBNODEs and their adjoint ODEs are much less stiff than other NODEs, and thus learning HBNODEs is computationally much more efficient than other NODEs.

We provide numerical validation on benchmark tasks and a detailed empirical and theoretical analysis of why HBNODEs are beneficial for learning dynamics of POD modes. Our numerical results show the adjoint state of HBNODEs does not vanish, confirming that HBNODEs do learn long-term dependencies, which results in remarkable performance gain over the baseline NODEs. Moreover, our experimental results show significant computational advantages in training and testing HBNODEs over the baseline ROM models.

1.2 Related work

There is a healthy amount of recent work on learning POD mode dynamics using deep neural networks, particularly RNNs and vanilla NODEs. Perhaps the most related papers to this article are [45, 16, 15], which study the NODE framework for learning ROMs. In [45], the authors developed a POD-NODE ROM framework for learning POD coefficients, which starts from FOM snapshots and then uses an autoencoder to encode the POD representations of FOM snapshots, followed by NODE training and forecasting. The POD-NODE ROM framework achieves appealing results for learning reduced dynamics of the VKS model, and it significantly outperforms the direct application of a long short-term memory (LSTM) network for sequential learning. In [16, 15], the authors study the effectiveness of NODEs for reduced-order modeling and predicting environment hydrodynamics. On the one hand, they find that NODEs provide an elegant framework for the stable and accurate evolution of latent-space dynamics with promising generalizability. On the other hand,

they noticed that in order to facilitate the widespread adoption of NODEs for large-scale systems, significant effort needs to be directed at accelerating training time. This limitation motivates this article's study and utilization of HBNODEs.

In addition to the NODE paradigm of continuous-depth neural networks for reduced-order modeling, the RNN — a natural sequential deep learning model — has also been successfully used for learning-assisted model reduction. Many advanced RNN algorithms can also be leveraged to enhance learning ROMs, e.g., LSTM networks [21]. RNN-based ROMs have achieved remarkable success in many applied domains, including multiphase flow simulation [24, 25], learning advection-dominated systems [35], learning chaotic dynamics [32], and learning nonlinear aeroelastic models [33]. Compared to NODEs for learning ROMs, RNNs cannot learn irregularly-sampled time series effectively and can even depart from the underpinning physics due to their discrete nature.

1.3 Organization

We organize the paper as follows: In Sections 2 and 3, we briefly review the POD-based reduced-order modeling and HBNODE for continuous-depth deep learning, respectively. We present the benchmark physical models of the complex dynamical systems and full-order modeling for data generation in Section 4. Section 5 shows the detailed deep learning model and pipeline for learning POD-based ROMs. We verify the efficacy of our proposed machine learning models and contrast them with several baseline models in Section 6, followed by concluding remarks.

2 POD-based Reduced-order Modeling

In this section, we briefly review key ideas and procedures of POD-based reduced-order modeling.

2.1 Notation

We denote vectors and matrices by lower- and upper-case boldface letters, respectively. For a vector $\mathbf{x} = (x_1; \ldots; x_d)^\top \in \mathbb{R}^d$, where $(x_1; \ldots; x_d)^\top$ denotes the transpose of the row vector $(x_1; \ldots; x_d)$, we use $\|\mathbf{x}\| = (\sum_{i=1}^{d} x_i^2)^{1/2}$ to denote its ℓ_2 norm, and use $\mathbf{0}$ to denote the zero vector. In cases when $d = 2$ and $\mathbf{x}$ is a spatial vector, we will write the components instead as $\mathbf{x} = (x; y)^\top$. For a matrix $\mathbf{A}$, we use $\mathbf{A}^\top$, $\mathbf{A}^{-1}$, and $\|\mathbf{A}\|$ to denote its transpose, inverse, and spectral norm, respectively. We use $\mathbf{I}$ to denote the identity matrix, whose size will be clear based on context.

We will consider the approximation of a space-time function $u = u(\mathbf{x}; t)$ where $\mathbf{x}$ is a spatial vector (typically of 1 or 2 dimensions) and t is a scalar

on [0; T] for some fixed and finite terminal time T. The function u may be vector-valued. In much of our discussion we will take the concrete example of u being a solution to a discretized VKS problem, whose details are given in Section 4.1. For the VKS problem, u 2 R^2 contains the horizontal (x) and vertical (y) components of a fluid velocity field. We will write u = (u_x; u_y) to denote these two components.

2.2 POD snapshots

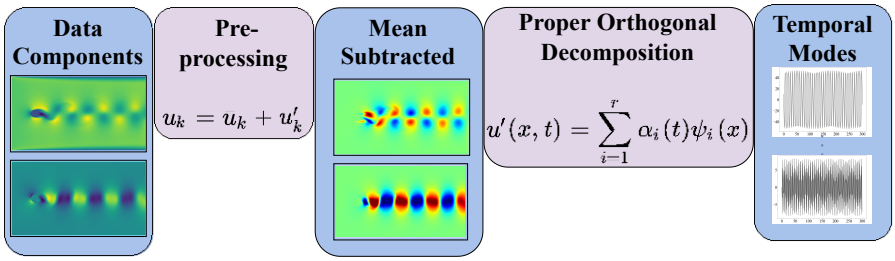


Fig. 1 POD pipeline: We first pre-process the data, from experimental observation or FOM simulation, by subtracting the mean. Then we apply spectral decomposition of the covariance matrix and only keep the first r eigenmodes, resulting in the reduced representation — P $_{i=1}^{r}$ $_i$(t) $_i$(x) — of u^0(x; t), where (t)s are the temporal coefficients and (x)s are the eigenmodes.

POD shares a similar spirit and implementation as the celebrated principal component analysis (PCA), the latter of which has been a very popular tool for data analysis [43, 30]. The key idea of PCA is to project high-dimensional data into a lower-dimensional space that is spanned by the eigenvectors corresponding to the leading eigenvalues of the covariance matrix of the data. PCA preserves the largest variance of training data and thus contains the most important information contained in the originally high-dimensional data. POD has been introduced in accelerating fluids simulation and reducing the complexity of fluid models since the pioneering work of Berkooz et al. [7]. Once the leading eigenmodes are obtained via analysis of training data, it is possible to reduce the order (the computational complexity and degrees of freedom) of the complex FOMs. The POD-based dimension reduction approach starts with some training samples of physically observed or numerically simulated snapshots of dynamics. These sample snapshots are aggregated into an ensemble matrix Y , where each row contains the state of a dynamical system at a fixed time step. Next, we compute the covariance matrix of the rows of the matrix Y , and the eigenvectors — sorted according to the corresponding decreasing-ordered eigenvalues — are used as the new orthogonal basis for representing the ROM. Below we summarize

the crucial steps of identifying the low-dimensional representations via the POD approach, which has also been visualized in Fig. 1.

- Data generation. We simulate the FOM, which is computationally expensive, for a short time to obtain the training data at time steps $t_1; t_2; \ldots; t_{N_t}$. For the VKS problem, when our solution u contains two components $u = (u_x; u_y)$ that depend on the two-dimensional spatial variable $x = (x; y)$, we assume that FOM snapshots $u_x(t_j)$, $u_y(t_j)$ are vectorized representations of N spatial degrees of freedom. Then we have

$$u_x(t_j); u_y(t_j) \in \mathbb{R}^N; \qquad j = 1; 2; \ldots; N_t:$$

  For real-world dynamical systems for which we do not know the exact governing equation, we sample the true dynamics via experimental measurements as training data. In either case, we assume that training data is available to us (as the snapshots above), and our goal is to efficiently leverage this data to learn reduced-order dynamics without recourse to the FOM, which we assume is unknown.
- Linearly center the data dynamics. With our VKS-centric notation above, according to the Reynolds decomposition of the flow, we have for fixed t_j,

$$u_x = \bar{u}_x + u'_x; \quad u_y = \bar{u}_y + u'_y; \tag{1}$$

  where $\bar{u}_x$ and $\bar{u}_y$ are the temporal mean of the solutions, computed over our N_t snapshots. The components u'_x and u'_y are the fluctuating components of the data.
- Data assembling. Concatenate the simulated and centered FOM snapshots into the following matrix Y,

$$Y = \begin{pmatrix} — & (u'(t_1))^\top & — & — & (u'_y(t_1))^\top & — \\ — & (u'_x(t_2))^\top & — & — & (u'_y(t_2))^\top & — \\ & \vdots & & & \vdots & \\ — & (u'_x(t_{N_t}))^\top & — & — & (u'_y(t_{N_t}))^\top & — \end{pmatrix}; \tag{2}$$

  so that row j contains the concatenated snapshot $u'_x(t_j); u'_y(t_j)$, i.e., the two flattened velocity components at time step j. The size of the matrix Y is $N_t \times 2N$.
- Perform a spectral decomposition of the covariance matrix. We construct the covariance matrix K of the rows of Y and compute its eigendecomposition:

$$K = Y Y^\top; \qquad K = A \Lambda A^T; \qquad A = (\varphi_1; \ldots; \varphi_{N_t}); \tag{3}$$

  where φ_j is the jth eigenvector, and the matrix Λ is diagonal containing entries λ_j, the associated non-negative eigenvalues of K. We assume the eigenvalues are listed in non-increasing order, $\lambda_j \geq \lambda_{j+1}$.

– Identify reduced-order modes and truncate. Larger eigenvalues of K are directly related to the dominant characteristics of the dynamical system, while small eigenvalues correspond to small perturbations of the dynamical behavior. The matrix K has N_t eigenvalues, and we choose the order of the reduced-order model to be $r \ll N_t$ by inspecting a relative information content $I(r)$, defined as,

$$I(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{N_t} \lambda_i}; \tag{4}$$

so that $1 - I(r)$ is a relative Frobenius norm error between K and its rank-r spectral approximation. As we will see in Section 6, $I(r)$ is close to one for practical problems, even for very small r. As output of the procedure, we can construct the following (discretized) ROM of the fluctuating component of the dynamics

$$u'(t_j) \approx \sum_{i=1}^{r} (\alpha_i)_j \, \phi_{\gamma,i}; \tag{5}$$

where $\gamma \in \{x, y\}$, and $(\phi_{x,i}^\top, \phi_{y,i}^\top)^\top \in \mathbb{R}^{2N}$ is a vector denoting a discretized spatial function; the entries of $\phi_{\gamma,i}$ correspond to the N degrees of freedom in the snapshots u, and is a subvector of the i-th right-singular vector of Y. Equivalently, it is defined as,

$$\phi_{\gamma,i} = \frac{1}{\sigma_i} Y_\gamma^\top \alpha_i = \frac{1}{\sigma_i} \sum_{j=1}^{N_t} (\alpha_i)_j \, u_\gamma'(t_j) \quad \gamma \in \{x, y\}$$

A "standard" POD approach would next project the (assumed known) dynamical model onto $\mathrm{span}\{\phi_{\gamma,i}\}_{i=1}^{r}$. We will proceed assuming that such a dynamical model is unknown to us, and will instead use machine learning models to predict dynamics.

Remark 1 With the training data α_i available at time steps $t_1, t_2, \ldots, t_{N_t}$ through the above procedure, extrapolation of the FOM dynamics or experimental measurements amounts to predicting the POD coefficients $\alpha_i(t)$ for future time t accurately, in our case using machine learning models. Notice that $\alpha_i(t)$ is observed sequentially and has a continuous profile, indicating the potential advantages of using NODE for learning $\alpha_i(t)$, as we describe next.

3 Heavy-ball Neural ODEs

In this section, we briefly review NODE and HBNODE and algorithms for their training and testing. Moreover, we provide some simple analysis of why HBNODE is better for learning POD coefficients for reduced-order modeling.

3.1 Neural ODEs

NODEs [9] are a class of continuous-depth (-time) neural networks [46, 11]. The continuous-time nature of NODEs makes them particularly suitable for learning complex dynamics from irregularly-sampled sequential data, see, e.g., [9, 47, 14, 34, 41]. Mathematically, a NODE is formulated as the following first-order ODE:

$$\frac{dh(t)}{dt} = f(h(t); t;); \tag{6}$$

where $f(h(t); t;) \in R^d$ is specified by a neural network parameterized by , e.g., a two-layer feed-forward neural network. Starting from the input $h(0)$, NODEs learn the representation and perform prediction by solving (6) from $t = 0$ to T using a numerical integrator with a given error tolerance, often with adaptive step size solver or adaptive solver for short [13]. Solving (6) from $t = 0$ to T in a single pass with an adaptive solver requires evaluating $f(h(t); t;)$ at various timestamps, with computational complexity measured by the number of function evaluations in a time-forward sweep ("forward NFEs") [9].

The adjoint sensitivity method, or the adjoint method [8], is a memory-efficient method for training NODEs through optimization of . We regard the output $h(T)$ as the prediction and denote the loss between the prediction $h(T)$ and the ground truth as L. Let $a(t) := @L=@h(t)$ be the adjoint state, then we have (see [9, 8] for details)

$$\frac{dL}{d} = \int_0^T a(t)^> \frac{@f(h(t); t;)}{@} dt; \tag{7}$$

with $a(t)$ satisfying the following adjoint ODE

$$\frac{da(t)}{dt} = \quad a(t)^> \frac{@}{@h} f(h(t); t;); \tag{8}$$

which is solved numerically from $t = T$ to 0 and also requires the evaluation of the right-hand side of (8) at various timestamps, with the number of NFEs during this time-backward sweep ("backward NFEs") measuring the computational complexity.

There are several critical problems with NODEs, including (i) Given an error tolerance, the NFEs required in a single forward pass can be excessive. Moreover, solving the adjoint ODE (8) often requires more NFEs than solving the forward ODE (6). (ii) In training NODEs, the adjoint state $a(t)$ often vanishes, i.e., the norm of $a(t)$ tends to 0, impeding NODEs from learning long-term dependencies [29], resulting in poor predictive performance.

3.2 Heavy-ball neural ODEs

The authors of [59, 56] proposed HBNODEs and their generalized version, named generalized HBNODEs (GHBNODEs). HBNODEs are motivated by

ideas from momentum-accelerated gradient descent [44] and they can be regarded as the continuous limit of the MomentumRNN model [40]. Mathematically, the HBNODE is a special second-order neural ODE of the following form

$$\frac{d^2h(t)}{dt^2} + \gamma\frac{dh(t)}{dt} = f(h(t), t, \theta), \tag{9}$$

where $\gamma \geq 0$ is the damping parameter, which can be set as a tunable or a learnable hyperparameter with positivity constraint. In the trainable case, we adopt the one used in [59], that is $\gamma = \bar{\gamma}\,\mathrm{sigmoid}(\omega)$ for a trainable $\omega \in \mathbb{R}$ and a fixed tunable upper bound $\bar{\gamma}$, e.g., $\bar{\gamma} = 1$. The HBNODE (9) can be rewritten as the following system of first-order NODEs

$$\frac{dh(t)}{dt} = m(t), \quad \frac{dm(t)}{dt} = -\gamma m(t) + f(h(t), t, \theta). \tag{10}$$

3.2.1 Computational advantages of HBNODE vs. NODE

To show why HBNODE enjoys computational efficiency in training and testing, let us first consider the adjoint equation of (9), which will again be solved using adaptive numerical ODE solvers. First, the following theoretical result [59] shows that the adjoint of an HBNODE is also an HBNODE.

Proposition 1 (Adjoint equation for HBNODE [59]) The adjoint state $a(t) := \partial L/\partial h(t)$ for the HBNODE (9) satisfies the following HBNODE with the same damping parameter γ as that in (9),

$$\frac{d^2a(t)}{dt^2} - \gamma\frac{da(t)}{dt} = a(t)\frac{\partial f}{\partial h}(h(t), t, \theta). \tag{11}$$

Notice that we solve the adjoint equation (11) from $t = T$ to 0 via backward propagation. By letting $\tau = T - t$ and $b(\tau) = a(T - \tau)$, we can rewrite (11) as follows,

$$\frac{d^2b(\tau)}{d\tau^2} + \gamma\frac{db(\tau)}{d\tau} = -b(\tau)\frac{\partial f}{\partial h}(h(T-\tau), T-\tau, \theta). \tag{12}$$

Therefore, the adjoint of the HBNODE is also an HBNODE and they have the same damping parameter.

The above result indicates that the adjoint problem for HBNODE is of the same type as the forward problem, accelerating backward propagation provided the forward propagation is accelerated.

Next, for a very simple linear case, we show that HBNODE can reduce the stiffness of the original NODE significantly. In particular, we consider the following two linearized high-dimensional ODE systems

$$\frac{dh(t)}{dt} = Ah(t), \tag{13}$$

and

$$\begin{aligned}\frac{dh(t)}{dt} &= m(t)\\ \frac{dm(t)}{dt} &= -m(t) + Ah(t)\end{aligned} \quad , \quad \frac{d}{dt}\begin{bmatrix} h(t)\\ m(t)\end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I\\ A & (-I)\end{bmatrix}}_{:=B}\begin{bmatrix} h(t)\\ m(t)\end{bmatrix} \quad : \quad (14)$$

We assume A is negative definite to simplify our analysis and reveal intuition of the advantages of HBNODE over NODE. Let the eigenvalues and eigenvectors of A be given by λ_i, v_i respectively. One trick to find the eigenvalues and the corresponding eigenvectors of the matrix B is to assume B has eigenvectors of the following form $\begin{bmatrix} v_i\\ cv_i\end{bmatrix}$ for some constant c. Then B satisfies the following eigenvalue-eigenvector equation

$$\begin{bmatrix} 0 & I\\ A & -I\end{bmatrix}\begin{bmatrix} v_i\\ cv_i\end{bmatrix} = \begin{bmatrix} cv_i\\ \lambda_i v_i - cv_i\end{bmatrix} \quad :$$

In which case if $\tilde{\lambda}$ is an eigenvalue of B then it is given by the value c satisfying,

$$\tilde{\lambda} = c = \frac{\lambda_i - c}{c} \quad :$$

Notice that c satisfies a quadratic equation in terms of λ_i, $c^2 + c - \lambda_i = 0$. Therefore the eigenvalues of B are given by

$$\tilde{\lambda} = \frac{-1 \pm \sqrt{1 + 4\lambda_i}}{2} \quad :$$

Let λ_{max} and λ_{min} be the eigenvalues of A of the largest and smallest magni-tude. Then the largest ($\tilde{\lambda}_{max}$) and smallest ($\tilde{\lambda}_{min}$) eigenvalues, in magnitude, of B satisfy the following

$$|\tilde{\lambda}_{max}| = \frac{-1 + \sqrt{1 + 4\lambda_{max}}}{2} \quad , \quad |\tilde{\lambda}_{min}| = \frac{1 - \sqrt{1 + 4\lambda_{min}}}{2} \quad :$$

Therefore, we have

$$\frac{|\tilde{\lambda}_{max}|}{|\tilde{\lambda}_{min}|} = \frac{-1 + \sqrt{1 + 4\lambda_{max}}}{1 - \sqrt{1 + 4\lambda_{min}}} \leq \sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \quad , \quad (15)$$

where we obtained the last inequality by simply setting $\alpha = 0$.

Note that the ratio $|\lambda_{max}|/|\lambda_{min}|$ and $|\tilde{\lambda}_{max}|/|\tilde{\lambda}_{min}|$ are the stiffness ratio of the linear ODE model (13) and the corresponding linear HBNODE counterpart (14). Thus (15) implies that the the heavy-ball ODE can be much less stiff than the original NODE. If the stiffness of the ODE model is κ, using a heavy-ball model results in a stiffness of at most $\sqrt{\kappa}$, which is a substantial reduction.

Recall that we use the adaptive step size explicit solver to solve both forward and backward ODEs, from $t = 0$ to T, in training NODEs and HBNODEs. A less stiff model allows the adaptive solver to use a much large step size and thus can significantly reduce NFEs. Moreover, Proposition 1 indicates that the adjoint equation of an HBNODE is also an HBNODE, and therefore we reap the computational advantages of relaxed stiffness in both forward and backward propagation phases. Our previous analysis only considers very simple linear ODE models. How to extend the analysis to the neural network is a very interesting future direction. One particular idea is analyzing the NODE and HBNODE when they are overparameterized, in which case one could leverage neural tangent kernel theory [23].

3.2.2 Generalized HBNODEs (GHBNODEs)

Compared to vanilla NODEs, high-order NODEs include HBNODEs usually suffer from uncontrolled aggregation of the hidden state, deteriorating model performance at best and blowing up training at worst. To alleviate this issue, in [59] the authors propose the following generalized HBNODE

$$\frac{dh(t)}{dt} = (m(t)); \qquad \frac{dm(t)}{dt} = \quad m(t) + f(h(t); t;) \quad h(t); \tag{16}$$

where () is a nonlinear activation, which is set as tanh by default. The positive hyperparameters ; > 0 are two tunable or learnable hyperparameters. In the trainable case, we let $=$ sigmoid(!) as in HBNODE, and $=$ softplus() to ensure that ; 0. Compared to HBNODEs, GHBNODEs integrate two ideas to improve the neural network architecture design: (i) Incorporating the gating mechanism used in LSTM [22] and GRU [10], which can suppress the aggregation of $m(t)$; (ii) Following the idea of skip connections [20], HBNODEs add the term $h(t)$ into the governing equation of $m(t)$, which benefits training and generalization of GHBNODEs. It has been extensively verified that GHBNODE can indeed control the growth of $h(t)$ effectively, which significantly improve the performance of machine learning models on various sequential learning tasks.

Another interesting result is that though the adjoint state of the GHBNODE does not satisfy the exact heavy-ball ODE, it also significantly reduces the backward NFEs in practice. We observe that sometimes GHBNODEs are computationally more efficient than HBNODEs.

3.2.3 (G)HBNODEs learn long-term dependencies effectively

Learning long-term dependencies is crucial for the success of deep learning for sequential data, and vanishing and exploding gradients are two bottlenecks for training RNNs to learn long-term dependencies [4, 42]. The exploding gradients issue can be effectively resolved via gradient clipping, training loss regularization, etc [42]. The vanishing gradient phenomenon in training RNNs materializes in continuous-depth neural networks as vanishing of the adjoint

state [59]. In particular, we consider $a(t) := \partial L/\partial h(t)$, and when the vanishing gradient phenomenon occurs, $a(t)$ goes to 0 quickly as $T - t$ increases, so that dL/d in (7) will be essentially independent of $a(t)$ for larger $T - t$. We have the following expressions for the adjoint states of the NODE and HBNODE (see [59] for details):

– For NODE, we have

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T}\frac{\partial h_T}{\partial h_t} = \frac{\partial L}{\partial h_T}\exp\left(\int_T^t \frac{\partial f}{\partial h}(h(s);s;)ds\right): \tag{17}$$

– For GHBNODE[1], we have

$$\begin{aligned}\frac{\partial L}{\partial h_t}\frac{\partial L}{\partial m_t} &= \frac{\partial L}{\partial h_T}\frac{\partial L}{\partial m_T}\begin{bmatrix}\frac{\partial h_T}{\partial h_t}\frac{\partial h_T}{\partial m_t}\\ \frac{\partial m_T}{\partial h_t}\frac{\partial m_T}{\partial m_t}\end{bmatrix} \\ &= \frac{\partial L}{\partial h_T}\quad\frac{\partial L}{\partial m_T}\exp\left(\int_T^t \underbrace{\begin{bmatrix}\frac{\partial f}{\partial h} & 0\\ \frac{\partial}{\partial m}ds & \end{bmatrix}}_{:=M}\right):\end{aligned} \tag{18}$$

For the matrix M, we have the following useful property about its spectrum.

Proposition 2 ([59]) The eigenvalues of M can be paired so that the sum of each pair equals $(t - T)$.

Following the argument in [59], Proposition 2 can be used to show that the adjoint state of NODE in (17) may vanish when $T - t$ is large, but the adjoint state of (G)HBNODEs in (18) will not vanish. This property supports the claim that HBNODEs benefit in learning long-term dependencies, which in turn further boosts the accuracy in learning POD of complex dynamical systems.

In Section 6, we will validate the above theoretical merits of HBNODEs over NODEs using the benchmark problems listed in Section 4 below.

4 Benchmarks and Data Preparation

In this section, we will present some details of the three benchmark physical models — VKS, KPP, and Euler equations — used for validating the efficacy of learning POD with HBNODEs. Our training data for the KPP and Euler equations are generated by solving FOMs; the training data for the VKS dataset is adopted from a publicly available dataset.

[1] HBNODE can be seen as a special GHBNODE with $= 0$ and be the identity map.

4.1 VKS model

The von Kármán vortex street (VKS) is a fluid dynamics phenomenon where vortices appear in a periodic fashion in the wake of flow past a blunt object, frequently a cylinder. A very small Reynolds number results in a laminar smooth flow past the cylinder, and very large Reynolds numbers result in a turbulent flow. In an appropriate middle regime, the VKS phenomenon appears and can be simulated. The associated dynamical model is the two-dimensional Navier-Stokes equations; with $u = (u_x; u_y)$ the fluid velocity, these equations read,

$$\frac{@u}{@t} = r^2 u \quad (u \ r) u \quad r\frac{1}{0}p;$$

where $_0$ is the spatially uniform pressure, and is the kinematic viscosity, which is inversely related to the Reynolds number. Our experimental setup concerns flow past a cylinder in two spatial dimensions with conditions that result in steady-state VKS flow after a initial transient period. We follow the experimental setting used in [45] to acquire simulation data.

4.2 KPP model

The Kurganov-Petrova-Popov (KPP) model is a scalar, two-dimensional conservation law, first proposed in [28]. This system is difficult to simulate since it features a non-convex flux, and is given by,

$$\frac{@u}{@t} + r \quad f(u) = 0; t > 0; \qquad x \ 2 \ [\ 2; 2]; \ y \ 2 \qquad \frac{5}{2}; \frac{3}{2} \ ;$$

where $f(u) = (\sin u; \cos u)^T$ and $r := \frac{@}{@x}; \frac{@}{@y}$. Our setup mirrors that in [28], so that we use the following initial data

$$u(x; y; 0) = \begin{matrix} \frac{14}{4}; x \ +_2 y^2 < 1; \\ s \\ _4; \text{el e} \end{matrix}$$

We employ a finite volume scheme utilizing a Lax-Friedrichs flux with a 5th-order WENO reconstruction over the two-dimensional rectangular domain with a Cartesian mesh up to time $T = 10$. The simulation uses a tensorial grid with $N_x = 50$, $N_y = 50$ (corresponding to $N = N_x N_y = 2500$ total spatial degrees of freedom), and $N_t = 1250$.

4.3 Euler equations for fluids modeling

The one-dimensional Euler equations of gas dynamics are a system of conservation laws. We consider the simulation of a parameterized shock-entropy

problem from this differential equation, whose setup is given by,

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0; \quad t > 0; x \in [-5; 5];$$
$$\text{with } f(u) = \begin{bmatrix} u \\ u^2 + p \\ (E + p)u \end{bmatrix}^T;$$

where $u := (\rho \; u \; E)^\top \in \mathbb{R}^3$ is the unknown with $(\rho; u; p; E)$ denoting the gas density, velocity, pressure, and energy, respectively. The system is closed via the following relationship between E and p:

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho u^2\right);$$

where γ is the heat capacity ratio, a gas-dependent constant.[2] We take boundary conditions at $x = -5$ as those given by the initial data. The shock-entropy problem features smoothly oscillating as well as discontinuous features.

We again employ a finite volume scheme to solve the Euler equations, using a Harten-Lax-van Leer (HLL) flux, which is an approximated Riemann solver [19]. Our simulations integrate up to terminal time $T = 1.8$, with a uniform grid having $N = 1000$ degrees of freedom in the scalar spatial variable x.

This last example differs from the previous two in that we consider this a parametric equation, where $\mu = (\mu_u; \mu_\rho) \in \mathbb{R}^2$ is a parameter for the initial conditions. We initialize the dynamics using the parameter μ as follows, where μ_u varies on the interval $[2; 3]$ and μ_ρ varies on the interval from $[3; 4]$. The parametric initial data $(u(x; 0); \rho(x; 0); p(x; 0)) = (u_0; \rho_0; p_0)$ are given by

$$u_0 = \begin{cases} \mu_u & x < -4; \\ 0 & \text{else} \end{cases}; \quad \rho_0 = \begin{cases} & x < -4; \\ 1 + 0.2\sin(x) & \text{else} \end{cases}; \quad p_0 = \begin{cases} \frac{3}{1}\frac{1}{3} & x < -4 \\ 1 & \text{else} \end{cases}$$

We generate training data by gathering an ensemble of trajectories for the above problem over a grid of μ values and attempt to learn dynamics on unseen values of μ. Thus, in this example we not only seek to predict to future times, but also trajectories on parameter values not in the training set.

5 Learning Pipeline

In this section, we describe the detailed pipeline of using deep learning for reduced-order modeling accompanied by the baseline ROMs.

[2] This γ is distinct from the γ discussed in Section 3.2.

5.1 Learning-based reduced-order modeling

Our machine learning-based reduced-order modeling framework is flexible for machine learning model selection, e.g., using either HBNODE or NODE as shown in Fig. 2 and Fig. 3, respectively. In our learning-based reduced-order modeling framework, we first apply POD outlined in Section 2 on the training data to extract (discretized) temporal coefficients (t)'s and the eigenmodes (x)'s following (5). Next, we will use machine learning models to predict future dynamics u(x; t) leveraging these coefficients and modes. In particular, the main task is extrapolation of the temporal coefficients (t)'s using NODEs or HBNODEs.

To predict future values of the POD data, we consider two different machine learning architectures, showing in Fig. 2 and Fig. 3, respectively. The first architecture is a one-to-one architecture that predicts the value at t_{k+1} based on the data at t_k. The second architecture is a sequence-to-sequence architecture that uses sequence data points to predict the following sequence of data points. The overlap in the sequence prediction can be adjusted so that the predicted sequence is entirely new or that only one new data point is predicted.

The first architecture under our study is adapted from [45], which was originally used to compare the performance of NODE and LSTM in model reduction. We replace the vanilla NODE used in [45] with the HBNODE, and we depict the modified architecture in Fig. 2. Compared to the pipeline used in [45], after the RNN encoding of the temporal coefficients we have to sample both h and m to accommodate learning using HBNODE. In contrast, the vanilla NODE used in [45] only needs to sample the state h. The above encoding and sampling procedure is accomplished via a variational autoencoder [26].

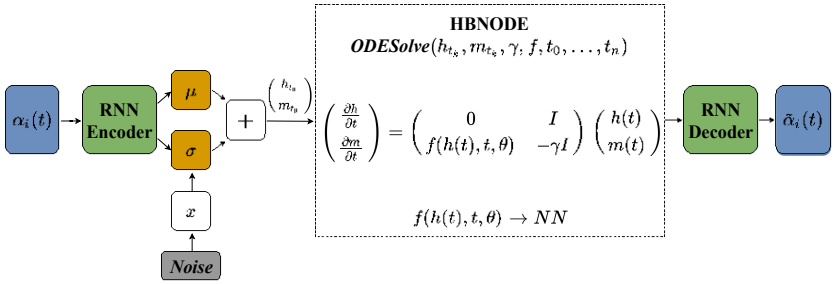


Fig. 2 The pipeline of predicting the temporal coefficients for a single step forward using HBNODE leveraging a variational autoencoder. We first use a RNN encoder to encode the input data and then sample the states h and m and evolve them using a HBNODE. Finally, we apply a RNN decoder to the final representation to get the prediction.

We plot the second architecture in Fig. 3, where the vanilla NODE can be replaced with (generalized) HBNODE. For the second architecture, i.e., the sequence-to-sequence architecture, takes a sequence of length n inputs and predicts a sequence of outputs, we encode the input sequence $f_i(t_j)g_{j=0}^{n}$ into the latent sequence $fz_i(t_j)g^{n}{}_{j=0}^{1}$ using an RNN encoder, then we use NODE or HBNODE to evolve the latent sequence to get the desired representation, followed by an RNN decoder to get the final long-term prediction $f_i(t_j)g_{j=n}^{N}$.

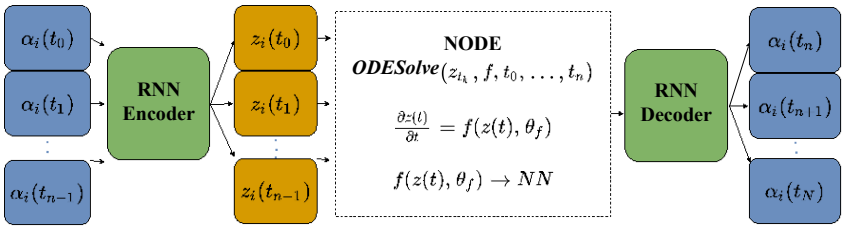


Fig. 3 The pipeline of predicting the temporal coefficients for multi-steps ahead. First, we encode the input sequence $f_i(t_j)g_{j=0}^{n}{}^{1}$ using a RNN encoder to obtain the latent sequence $fz_i(t_j)g_{j=0}^{n}{}^{1}$. Second, we use a NODE to learn a "good" representation of the input sequence by evolving the latent sequence $fz_i(t_j)g^{n}{}_{j=0}^{1}$ Third, we apply a RNN decoder to the "good" representation to get the final prediction. Notice that NODE can be replaced with (generalized) HBNODE, in which case we need to obtain another sequence of momentum states from the RNN encoder.

5.2 A baseline comparison: Dynamic Mode Decomposition (DMD)

We employ DMD as another baseline model reduction method to demonstrate the effectiveness of learning-based model reduction using HBNODEs. In this part, we briefly review the idea of DMD for reduced-order modeling. To compare DMD to the learning-based reduced-order modeling using HBNODE, we consider only modeling the fluctuating components u^0 of the snapshots, see (1). The predictions of DMD are generated by a linear operator A corresponding to a linear difference equation $u^0_{k+1} = Au^0_k$, where A must be learned. In DMD, dominant eigenvalues and eigenvectors of A are computed via the singular value decomposition (SVD). Although the true underlying dynamics may be nonlinear, the Koopman operator formalism concludes that a lifted version of the dynamics is indeed linear. For nonlinear problems, DMD attempts to learn these lifted linear dynamics.

Let $U^{(k)}$ be the snapshot matrix for the time interval $t_0; : : : ; t_{train\ 1}$ and $U^{(k+1)}$ be the snapshot matrix for the time interval $t_1; : : : ; t_{train}$, i.e., column j of $U^{(k)}$ corresponding to time snapshot $t_{j\ 1}$. In particular, let $U^{(k+1)}$ A $U^{(k)}$ where $U^{(k)}$ is given by the SVD $U^{(k)} = X V$. We further denote X , V , and as the rank-r truncation of X , V , and , respectively. Then we may

compute an approximation $\tilde{A}$ directly from $U^{(k+1)}$ by the following,

$$\tilde{A} = \tilde{X} U^{(k+1)} \tilde{V} \tilde{\Sigma}^{-1} \tag{19}$$

The reduced matrix $\tilde{A}$ is composed of the dominant r eigenvalues $_1; \dots; _r$ and eigenvectors $_1; \dots; _r$. These eigenvectors are also known as the DMD modes. Given training data on the training interval $t_0; \dots; t_{train}$, the matrix $\tilde{A}$ is formulated by partitioning the snapshot matrix u^0 into two time intervals. Validation data on the interval $t_{train+1}; \dots; t_{valid}$ is generated by solving $u^0(t_{train+k}) = \tilde{A}^k u^0(t_{train})$. We depict DMD-based reduced-order modeling in Fig. 4. More details of DMD can be found at e.g., [51].

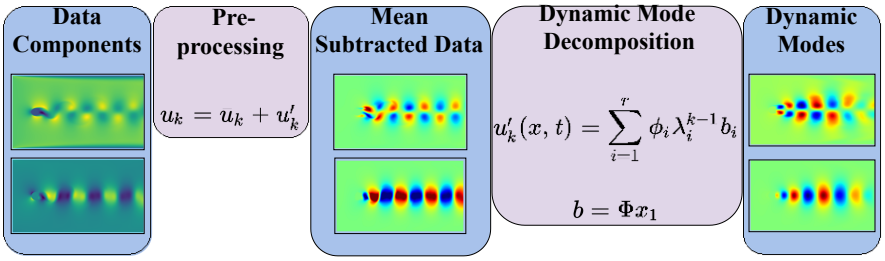


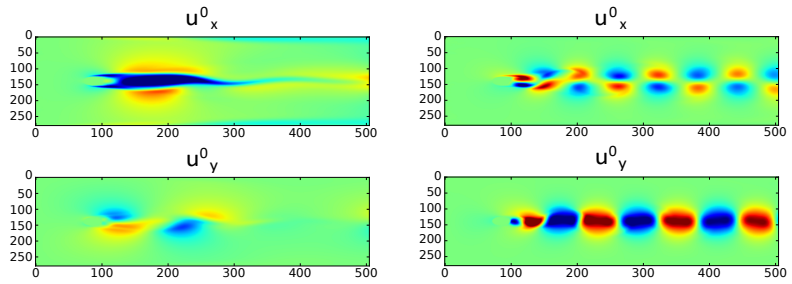
Fig. 4 DMD pipeline: The data is pre-processed by subtracting the mean to capture the fluctuating components of the data. In addition the following lifts $f\cos(x); \sin(x); x^2; x^3 g$ were applied to the data, and then vectorized along the snap-shot axis. We then generate the full-order model according to the spectral decomposition of the linear transform between the two snapshot matrices at subsequent time intervals. To reduce the order only the dominant r eigenmodes are selected, resulting in a representation $u^0(x;t) = \sum_{i=1}^{r} {}_i {}^k {}_i^1 b_i$ for the lifted data $u^c(x;t)$.

6 Experimental Results

In each experiment below, we contrast the performance of HBNODE-based ROM to two baseline ROMs, namely, NODE-based and DMD-based ROMs. We observe consistently improved predictive performance of HBNODE over baseline ROMs. We interpret the improved performance using HBNODEs by inspecting the stiffness and adjoint state of HBNODEs, confirming the theoretical results. Animated comparisons of the data reconstructions can be found at [3].

6.1 Transient and steady-state VKS

The VKS dataset is obtained by simulating the FOM in Section 4.1 on the time interval $[0; 400]$, containing two different regimes. When $t < 100$, the dynamics lie in the transient state and approaches the steady-state as t increases; while the dynamics maintain a steady-state when $t \geq 100$, as shown in Fig. 5.



(a) Transient state t = 50 (b) Steady-state t = 150

Fig. 5 Comparison of transient and steady-state phases of the VKS dataset. The steady-state phase contains quasi-periodic solutions conducive to machine learning. The transient phase does not contain such well-behaved dynamics.

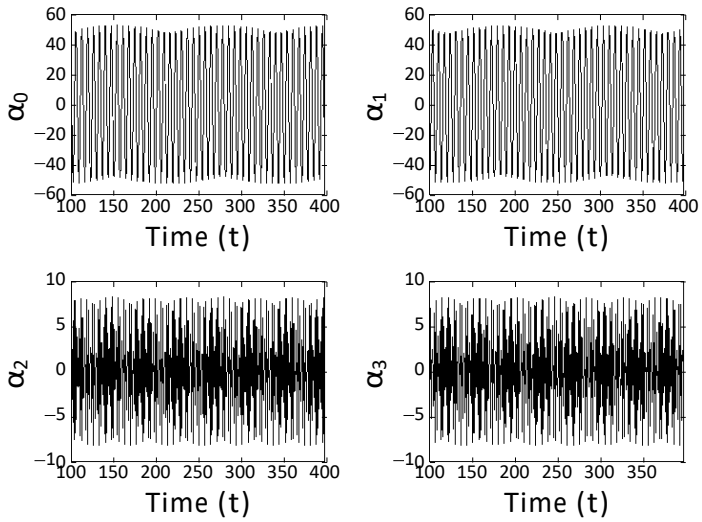
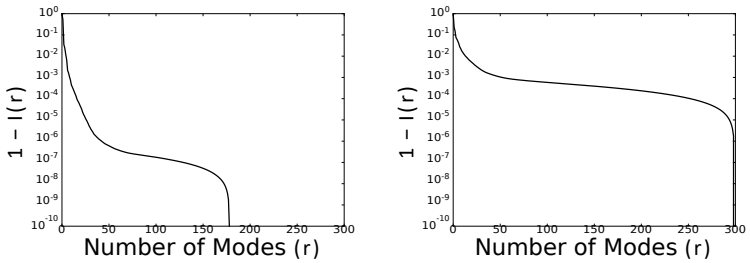


Fig. 6 The steady-state POD modes are highly oscillatory with quasi-periodic patterns.

ROMs for steady state dynamics. We contrast different ROMs for simulating VKS flow in the steady-state regime. In particular, both the DMD and POD training is taken over the time interval from t = 100 to 400. The POD modes for the steady-state flow oscillate quasi-periodically, see Fig. 6, and the relative information content $I(r)$ in (4) decays rapidly in r. The POD relative information content for 8 leading modes is 99%, as illustrated in Fig. 7 (a). In contrast, the lifted DMD model, using the lifts $f\cos(x); \sin(x); x^2; x^3 g$ with $x = (x; y)$, requires 24 modes to achieve 99% relative information content, shown in Fig. 7 (b). The quasi-periodic nature of the POD modes indicates that a model with high training accuracy will continue to perform well on the validation data.



(a) POD decay (b) DMD decay

Fig. 7 Comparison of the relative information decay for POD and DMD over the steady-state VKS. The POD modes decay far more rapidly than those of DMD. Therefore, one can expect better results with a smaller order of ROM using POD than DMD.

ROMs for transient to steady state dynamics. ROMs behave very differently over the entire time interval from t = 0 to 400. The POD modes do not oscillate over the entire interval but only over the steady-state phase. For both POD and DMD, the relative information decays much slower. The POD relative information decreases to 96% for the dominant 8 modes. While for the dominant 24 lifted DMD modes, the relative information is reduced to 94% over the full dynamics. This suggests that a machine learning-based ROM which is able to train on the transient phase to predict the steady-state phase accurately captures the intrinsic patterns of the underlying dynamics.

Hyperparameter	Value
Latent dimension	6
Layers encoder	4
Units encoder	10
Layers ODE	12
Units decoder	41
Layers decoder	4
Learning rate	.00153
Epochs	2000

Table 1 The hyperparameters for the VAE architecture — shown in Fig. 2 — for NODE and HBNODE-based ROMs. The parameters are tuned to the best NODE specification.

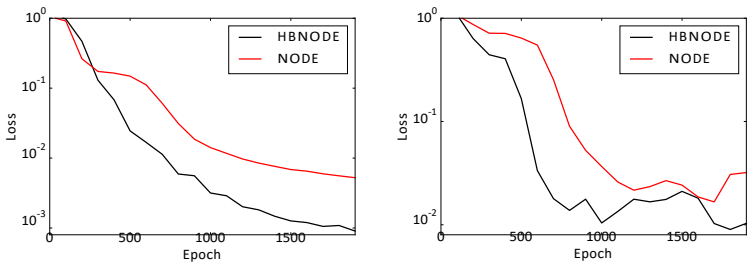
Learning steady-state dynamics. In this task, we train the pipeline shown in Fig. 2 for single-input-single-output dynamics prediction. Following the baseline in [45], we train over the steady-state dynamics starting from t = 100 using the dominant 8 POD modes. The training data consists of the POD modes from t = 100 to 174, and the training labels consist of the POD modes from t = 101 to 175. The validation data consists of the POD modes from t = 175 to 199, with the objective to predict the POD modes at time steps from t = 176 to 200. We use the mean squared error to measure the loss between the labeled data and the predictions. We utilized an AdamW optimizer to train the network

based on this loss criteria. For the black-box integration method, we selected DOPRI-5 [13] with a relative tolerance of 1e 8. The model's hyperparameters are tuned to best the NODE as outlined in [45] and restated in Table 1.

Hyper-parameter	Value
Layers	12
Hidden layers	64
Sequence length	9
Learning rate	.001
Epochs	500

Table 2 Hyperparameters of NODE and HBNODE for learning ROMs from transient to steady-state VKS dynamics.

Learning transient to steady-state dynamics. In this task, we train the pipeline outlined in Fig. 3 for multi-input-single-output dynamics prediction. The objective of this task is to capture the phase transition at t = 100. The data consisted of the dominant 8 POD modes for the time interval from t = 0 to 400. The data was sequenced in a multi-input-single-output structure so that 9 preceding time steps were used to predict the 10-th time step. The training data consists of the POD modes for the transient time interval from t = 0 to 79. The training labels consisted of the POD modes from t = 10 to 80. The validation data utilizes the POD modes from steady-state time interval t = 80 to 119, and the validation labels consist of data from t = 90 to 120. The other experimental settings follow the above single-input-single-output scenario. The model's hyperparameters are the same for NODE and HBNODE components and are given in Table 2.

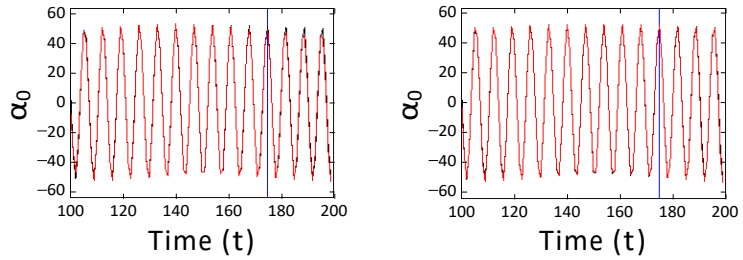


(a) Training loss (b) Validation loss

Fig. 8 Contrasting NODE and HBNODE-based ROMs for learning steady-state VKS dynamics. HBNODE outperforms NODE in both training and validation loss.

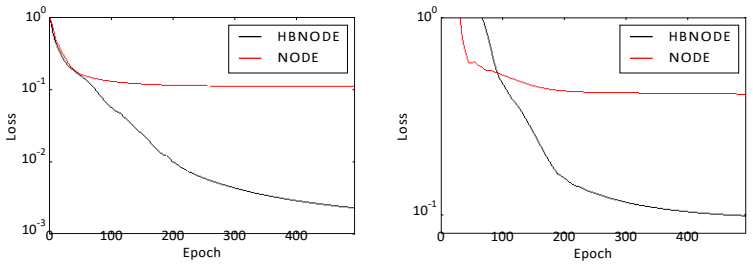
6.1.1 Results and comparison to existing ROMs

Results of learning steady-state dynamics. We contrast HBNODE and NODE-based ROMs in Fig. 8 and Fig. 9. Figure 8 shows that HBNODE-based ROM not only achieves remarkably smaller training loss but also significantly smaller validation loss than NODE-based ROM. In terms of the predictive performance, we see that HBNODE performs better at capturing several of the peaks of the oscillatory modes as shown in Fig. 9.



(a) VAE-NODE dominant mode (b) VAE-HBNODE dominant mode

Fig. 9 Comparison of modes reconstruction for NODE and HBNODE-based ROMs for learning steady-state VKS dynamics. HBNODE captures the peaks of the dominant POD modes better than NODE. Before and after the vertical blue line stands for training and validation, respectively.

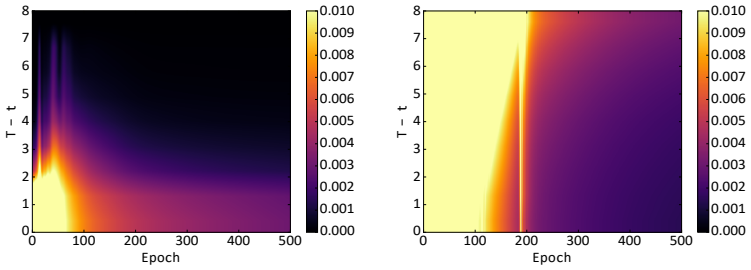


(a) Training loss (b) Validation loss

Fig. 10 Contrasting training and validation loss of NODE and HBNODE for learning ROMs of transient to steady-state VKS dynamics. The progress made by the NODE is significantly reduced by the vanishing gradient. HBNODE has a much slower decaying gradient and is able to continue to make progress in both training and validation sets.

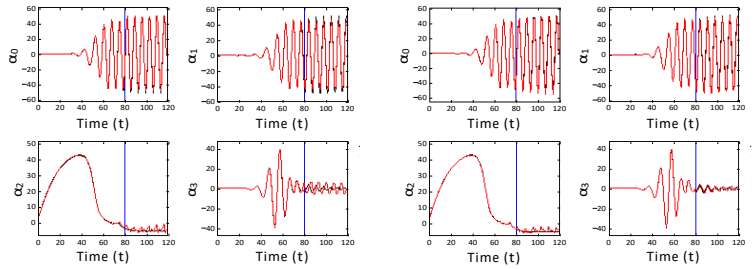
Results of learning transient to steady-state dynamics. Compared to learning steady-state VKS dynamics, HBNODE achieves more significant performance gain over NODE for learning transient to steady-state dynamics in terms of training and validation loss, as shown in Fig. 10. Since we are doing sequential

learning, one interpretation of the improvement in learning dynamics is the effective learning of long-term dependencies. Indeed, the criterion of learning long-term dependencies has been widely used in measuring the efficacy of sequential learning models [42,59]. In NODE and HBNODE, the effectiveness of learning long-term dependencies can be measured by whether the adjoint state vanishes quickly or not. We visualize the evolution of the magnitude of the adjoint states of NODE and HBNODE in Fig. 11, which support the theoretical result in Section 3.2.3. In particular, we see that the adjoint state of NODE vanishes much more rapidly than that of HBNODE as T t increases. A more detailed connection between the adjoint state and learning long-term dependencies is provided in [59].



(a) NODE adjoint state (b) HBNODE adjoint state

Fig. 11 Comparison of the adjoint states for the NODE and HBNODE in learning multi-input-single-output. The NODE adjoint state vanishes substantially faster than HBNODE.



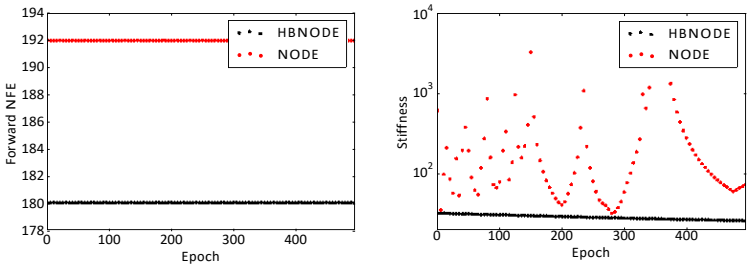
(a) VAE-NODE modes (b) VAE-HBNODE modes

Fig. 12 Comparison of modes reconstruction for NODE and HBNODE-based ROMs for learning multi-input-single-output task. NODE is able to reliably learn the first two dominant modes but is unable to capture the steady-state dynamics, having a much larger frequency and introducing substantial lag into the oscillation frequency. HBNODE is able to more reliably capture the steady-state dynamics with slightly larger frequency and developing lag much later than the NODE component. Before and after the vertical blue line stands for training and validation, respectively.

In terms of the predictive performance, as shown in Fig. 12, the HBNODE predictor captures the peaks of the oscillatory dynamics better than NODEs,

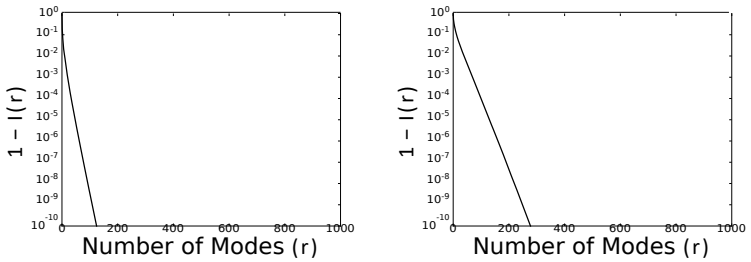
especially in the first two modes $_1$ and $_2$. Moreover, the prediction error using NODE is much larger than that of HBNODE, and the prediction error amplifies as the prediction time goes, in particular, for modes $_3$ and $_4$.

Another primary advantage of HBNODE over NODE-based ROMs lies in computational efficiency, which is theoretically supported by the discussion in Section 3.2.1. As shown in Fig. 13 (a), the forward NFE required in each forward pass by HBNODE is consistent smaller than that of NODE. We also monitor the stiffness of both NODE and HBNODE during the learning process, and Fig. 13 (b) shows that the stiffness of NODE oscillates and maintains much larger than HBNODE.



(a) Forward NFE comparison (b) Stiffness comparison

Fig. 13 Comparison of NFEs and stiffness for the NODE and HBNODE in learning transient to steady-state VKS dynamics. NODE requires more NFEs in each forward pass than HBNODE as the NODE is much more stiff than HBNODE. The stiffness of NODE varies sharply as training goes, while the stiffness of HBNODE decays during the training.



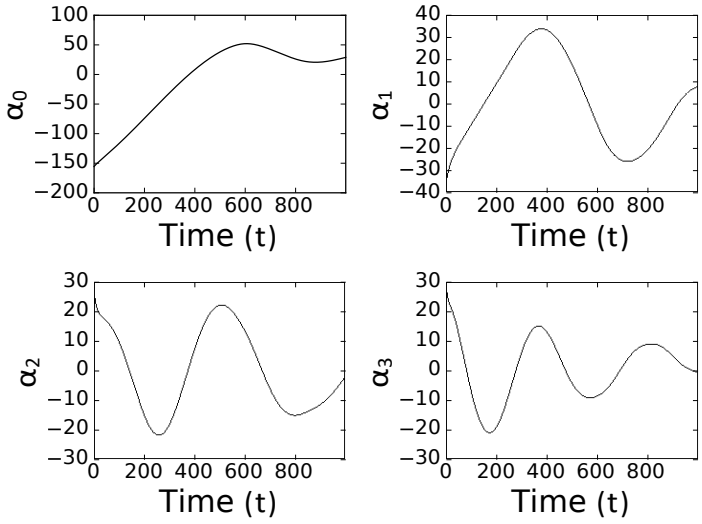
(a) POD mode decay (b) DMD mode decay

Fig. 14 The decay of relative information content for POD and DMD of the KPP dataset. The rapid decay in the modes indicates the problem is suitable to POD and DMD.

6.2 KPP model

We obtain the KPP dataset by simulating the FOM presented in Section 4.2 for 1000 timesteps (N_t = 1000). The KPP model is well-suited for reduced-order

modeling due to the rapidly decaying eigenvalues in both POD and DMD, seeing Fig. 14. However, we found in our experiments that it is particularly difficult to capture the dynamics using machine learning architectures due to the slow decaying ROM dynamics depicted in Fig. 15.



KPP POD modes

Fig. 15 The dominant POD modes for the KPP data. The oscillations of the modes decay rapidly and have a very low frequency, making it challenging for learning compared to the quasi-periodic oscillations of the VKS dataset.

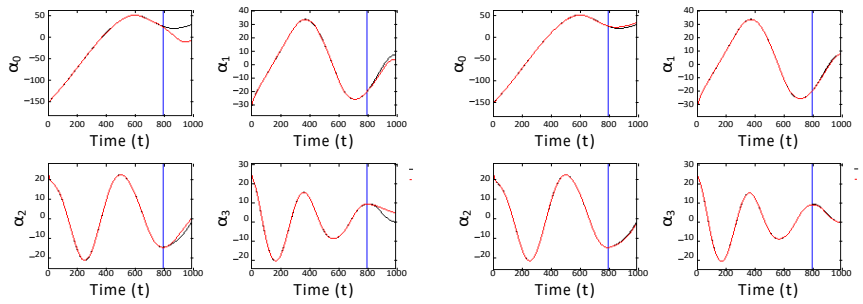
In our experiments, we note that the non-lifted DMD continuously deforms the center of mass in a way that defies physical constraints of the system. A comparison of lifted and non-lifted DMD predictions and POD predictions are shown in [3]. To lift DMD, we utilized the lifting functions $f\cos(x); \sin(x); x^2; x^3 g$ with $x = (x; y)$. Figure 14 shows that the POD modes decay faster than the lifted DMD modes. The dominant 24 DMD modes correspond to 97% of the relative information content; in contrast, the dominant 8 POD modes correspond to 99% of the relative information value.

We train the pipeline depicted in Fig. 3 for learning multi-input-single-output dynamics. The data is constructed from the 8 dominant POD modes on the time interval from $t = 0$ to 1000. The data is sequenced so that every 4 preceding time steps are used to predict the 5-th time step. The training data consists of the POD modes from $t = 0$ to 799 and the training labels consist of POD modes from $t = 5$ to 800. The validation data utilizes data from $t = 800$ to 999 and the validation labels consist of data from $t = 805$ to 1000. The model's hyperparameters are the same for NODE and HBNODE components and are given in Table 3.

Hyper-parameter	Value
Layers	2
Hidden layers	64
Sequence length	4
Learning rate	.01
Epochs	500

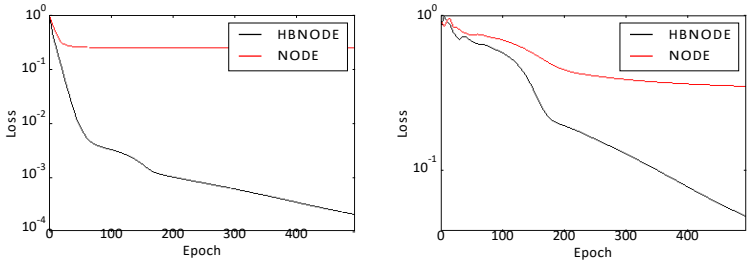
Table 3 The hyperparameters of NODE and NODE for the learning ROMs of KPP model.

6.2.1 Results and comparison to existing ROMs



(a) NODE mode recapture (b) HBNODE mode recapture

Fig. 16 Comparison of the KPP modes prediction using NODE and HBNODE. HBNODE is better in predicting the dynamics for the validation set than NODE. Before and after the vertical blue line stands for training and validation, respectively.



(a) Training loss (b) Validation loss

Fig. 17 Comparison of the training and validation loss of NODE and HBNODE for learning ROMs for the KPP model. NODE is unable to make progress due to a rapidly vanishing gradient, impeding learning long-term dependencies.

We compare the prediction of NODE and HBNODE against ground truth in Fig. 16, and we see that HBNODE performs remarkably better than NODE in predicting the dynamics. In particular, HBNODE is able to properly capturing

the oscillation dynamics of the modes unlike NODE. Figure 17 shows that HBNODE has much smaller training and validation loss than that of NODE.

6.3 Euler equations for fluids modeling

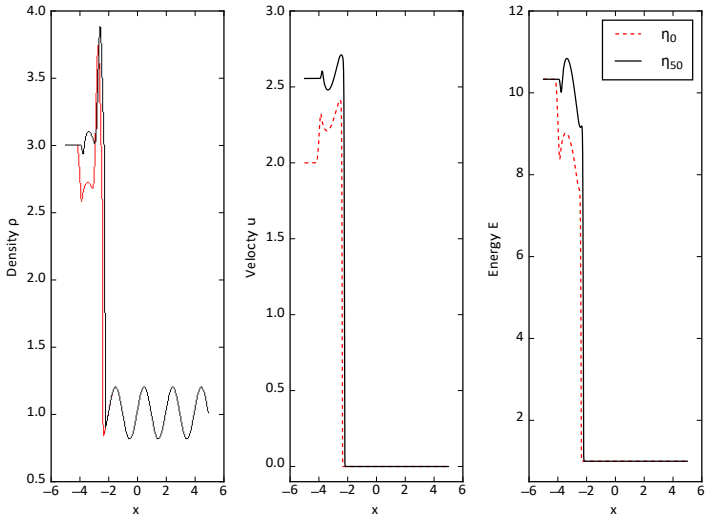


Fig. 18 The Euler Equations data at time step t = 50 with two different initial parameters, $_0$ and $_{50}$. Variations in the parameter produces widely varying dynamics, and as a result varying POD modes. The objective of this task is to predict the dynamics for unseen parameters using a set of training parameters $_{train}$.

We further consider learning reduced-order models for simulating the Euler equations, where the dataset is obtained by simulating the full-order model presented in Section 4.3 with a discrete ensemble of parameters $_1; : : : ; _M$ with M = 100, over 180 timesteps. Two different parameter values can produce rather different dynamics, as as evidenced in Fig. 18. The Euler equations data is unique in the sense that it may be segmented based on these initial conditions. The ROM is generated by taking the dominant 8 POD modes for each parameter $_i$ on the time interval from t = 0 to 180. This data is shuffled randomly among the initial parameter $_i$ to no longer increasing sequentially. The average relative information content across all 100 values of is 95%.

In this task, we train the machine learning pipeline shown in Fig. 3 for learning multi-input-multi-output dynamics. The training dataset comprises the dominant 8 POD modes for each of the training parameters among $_1; : : : ; _{train}$. We use 90 of the 100 parameters, $_1; : : : ; _{90}$, for training and the rest for validation. The training input consists of the dominant 8 POD modes for each of the 90 training parameters on the time interval from t = 0 to 150. The

training labels consist of the dominant 8 POD modes for each of the 90 training parameters time steps from t = 151 to 180. The validation dataset is composed of the validation parameters $_{91};:::;_{100}$. The validation input and labels are segmented using the same intervals as the training data.

The model uses a GHBNODE component with a hyperbolic tangent activation function. All other experimental settings are the same as in the KPP dataset, and the tuned hyper-parameters are listed in table 4. The NODE and HBNODE models are trained and validated over the same data shuffling.

Hyper-parameter	Value
Layers	6
Hidden layers	16
Learning rate	.01
Epochs	100

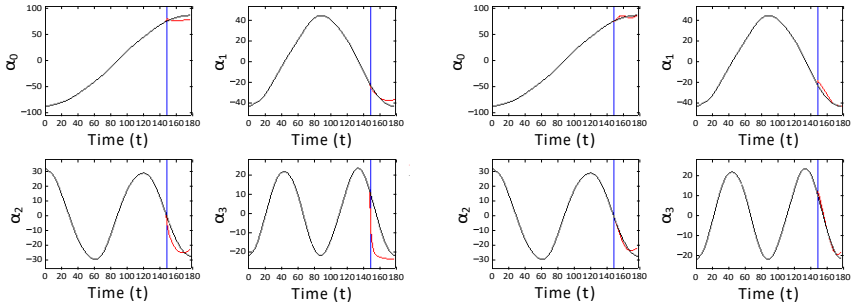
Table 4 The hyperparameters of NODE and NODE for the learning ROMs of Euler equations.

6.3.1 Results and comparison to existing models

We compare the prediction of NODE and GHBNODE for a randomly selected parameter $_{train}$ from the training set and a randomly selected parameter $_{valid}$ from the validation set. The modes for the training parameter $_{train}$ are shown in Fig. 19, and the modes for the validation parameter $_{valid}$ are shown in Fig. 20. We observe that the POD modes for the parameter $_{train}$ in Fig. 19 differ from those for $_{valid}$ primarily in amplitude rather than shape. As a result, a poor prediction model will have a sudden discontinuity between the input and prediction values. The transition point between the input and the prediction is indicated in Fig. 19 and Fig. 20 by the vertical blue line.

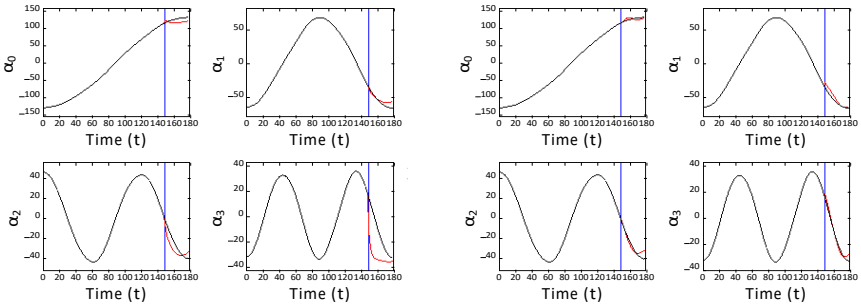
Figure 19 shows the dominant 4 POD modes for $_{train}$ from the training set. The predictive capabilities of GHBNODE significantly outperform the NODE model. In particular, for $_2$ and $_3$, we observe that the NODE has a large jump discontinuity at the transition between the input and prediction. The GHBNODE modes are smoother in the transition region, which indicates the ability of the GHBNODE model to distinguish between separate parameters.

In Figure 20, we observe the same characteristics for NODE and GHBNODE. NODE is unable to accurately predict the output for the parameter $_{valid}$ from the validation set. In particular, for $_3$ of the validation parameter, the NODE prediction is even less smooth than that of the training parameter shown in Fig. 19. In this experiment, we observe that NODE is unable to distinguish data with varying parameters as accurately as GHBNODE.



(a) NODE Training Mode Recapture (b) GHBNODE Training Mode Recapture

Fig. 19 For a randomly selected initial parameter $_{train}$, we plot the dominant four modes and their predicted data using NODE and GHBNODE. The blue vertical line separates the input and output data. The ground truth is in black, while the prediction data is in dashed red. GHBNODE can learn the dynamics of $_2$ and $_1$ for the parameterized data significantly better than NODE. This is evidenced by the fact that NODE predicts the inflection point of $_2$ too early.



(a) NODE Validation Mode Recapture (b) GHBNODE Validation Mode Recapture

Fig. 20 For the randomly selected initial parameter $_{valid}$, we plot the dominant four modes and their predicted data using NODE and GHBNODE. The blue vertical line separates the input and output data. The ground truth is in black, while the prediction data is in dashed red. HBNODE can predict the data for unseen parameterizations much better than NODE.

7 Concluding Remarks

This paper employs the recently developed HBNODEs and their generalization for learning POD coefficients for model reduction. We analyze through simple linearized models and empirically verify the advantages of HBNODEs over existing NODEs. In particular, HBNODEs enjoy the following advantages that imply practical benefits for learning POD-based ROMs, including 1) The deep learning model is continuous-depth, providing flexibility in learning irregularly-sampled time series and faithful to the continuous profiling of the underlying physical models. 2) Both the forward and adjoint ODEs of HBNODEs are of the heavy-ball style, accelerating both training and testing of the machine learning procedure. And 3) HBNODEs can learn long-term dependencies effectively,

capturing intrinsic patterns from data. There are numerous avenues for future works, and two particular interesting directions in our mind are 1) Improving HBNODEs, particularly replacing the fine-tuned or learned damping parameter with an adaptive one that are motivated by certain optimization algorithms with adaptive momentum [55, 54, 57], and 2) Applying HBNODE-based ROMs to model reduction arising from scientific challenges, especially when we do not have the ground truth governing equation of the dynamical systems.

8 Acknowledgement

This material is based on research sponsored by NSF grants DMS-1848508, DMS-1924935, DMS-1952339, and DMS-2111117, DOE grant DE-SC0021142, and AFOSR FA9550-20-1-0338. We also acknowledge support from a seed grant from the College of Science at the University of Utah.

References

1. A. Antoulas. Approximation of Large-Scale Dynamical Systems. Advances in Design and Control. Society for Industrial and Applied Mathematics, January 2005.
2. Athanasios C. Antoulas, Christopher A. Beattie, and Serkan Gugercin. Interpolatory Model Reduction of Large-Scale Dynamical Systems. In Javad Mohammadpour and Karolos M. Grigoriadis, editors, Efficient Modeling and Control of Large-Scale Systems, pages 3–58. Springer US, Boston, MA, 2010.
3. Justin Baker, Elena Cherkaev, Akil Narayan, and Bao Wang. Learning pod of complex dynamics usingheavy-ball neural odes: Animations. https://www.github.com/JustinBakerMath/pod_hbnode/blob/master/README.md#animations.
4. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157–166, 1994.
5. P. Benner, S. Gugercin, and K. Willcox. A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems. SIAM Review, 57(4):483–531, January 2015.
6. Peter Benner, Serkan Gugercin, and Karen Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. SIAM review, 57(4):483–531, 2015.
7. Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. Annual review of fluid mechanics, 25(1):539–575, 1993.
8. L. Bittner. L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, E. F. Mishechenko, the mathematical theory of optimal processes. VIII + 360 S. New York/London 1962. John Wiley & Sons. Preis 90/–. ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik, 43(10-11):514–515, 1963.
9. Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018.
10. Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

11. Michael A. Cohen and Stephen Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13(5):815–826, 1983.
12. Richard V Craster and Omar K Matar. Dynamics and stability of thin liquid films. Reviews of modern physics, 81(3):1131, 2009.
13. J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. Journal of Computational and Applied Mathematics, 6(1):19–26, 1980.
14. Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.
15. Sourav Dutta, Peter Rivera-Casillas, Orie M Cecil, Matthew W Farthing, Emma Perracchione, and Mario Putti. Data-driven reduced order modeling of environmental hydrodynamics using deep autoencoders and neural odes. arXiv preprint arXiv:2107.02784, 2021.
16. Sourav Dutta, Peter Rivera-Casillas, and Matthew W Farthing. Neural ordinary differential equations for data-driven reduced order modeling of environmental hydrodynamics. arXiv preprint arXiv:2104.13962, 2021.
17. Massimo Germano, Ugo Piomelli, Parviz Moin, and William H Cabot. A dynamic subgrid-scale eddy viscosity model. Physics of Fluids A: Fluid Dynamics, 3(7):1760–1765, 1991.
18. Serkan Gugercin and Athanasios C. Antoulas. A Survey of Model Reduction by Balanced Truncation and Some New Results. International Journal of Control, 77(8):748–766, May 2004.
19. Amiram Harten, Peter D. Lax, and Bram van Leer. On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws. SIAM Review, 25(1):35–61, 1983.
20. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In European Conference on Computer Vision, pages 630–645, 2016.
21. S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
22. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
23. Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: convergence and generalization in neural networks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, pages 8580–8589, 2018.
24. J Nagoor Kani and Ahmed H Elsheikh. Dr-rnn: A deep residual recurrent neural network for model reduction. arXiv preprint arXiv:1709.00939, 2017.
25. J Nagoor Kani and Ahmed H Elsheikh. Reduced-order modeling of subsurface multiphase flow models using deep residual recurrent neural networks. Transport in Porous Media, 126(3):713–741, 2019.
26. Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.
27. Yoshiki Kuramoto. Diffusion-Induced Chaos in Reaction Systems. Progress of Theoretical Physics Supplement, 64:346–367, 02 1978.
28. Alexander Kurganov, Guergana Petrova, and Bojan Popov. Adaptive Semidiscrete Central-Upwind Schemes for Nonconvex Hyperbolic Conservation Laws. SIAM Journal on Scientific Computing, 29(6):2381–2401, January 2007.
29. Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. arXiv preprint arXiv:2006.04418, 2020.
30. Y.C. Liang, H.P. Lee, S.P. Lim, W.Z. Lin, K.H. Lee, and C.G. Wu. Proper orthogonal decomposition and its applications—part i: Theory. Journal of Sound and Vibration, 252(3):527–544, 2002.
31. Hugo FS Lui and William R Wolf. Construction of reduced-order models for fluid flows using deep feedforward neural networks. Journal of Fluid Mechanics, 872:963–994, 2019.
32. Chao Ma, Jianchun Wang, et al. Model reduction with memory and the machine learning of dynamical systems. arXiv preprint arXiv:1808.04258, 2018.
33. Andrea Mannarino and Paolo Mantegazza. Nonlinear aeroelastic reduced order modeling by recurrent neural networks. Journal of Fluids and Structures, 48:103–121, 2014.

34. Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 3952–3963. Curran Associates, Inc., 2020.
35. Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. Physics of Fluids, 33(3):037106, 2021.
36. M. Mohebujjaman, L.G. Rebholz, and T. Iliescu. Physically constrained data-driven correction for reduced-order modeling of fluid flows. International Journal for Numerical Methods in Fluids, 89(3):103–122, 2019.
37. Parviz Moin and Krishnan Mahesh. Direct numerical simulation: a tool in turbulence research. Annual review of fluid mechanics, 30(1):539–578, 1998.
38. Changhong Mou, Honghu Liu, David R Wells, and Traian Iliescu. Data-driven correction reduced order models for the quasi-geostrophic equations: A numerical investigation. International Journal of Computational Fluid Dynamics, 34(2):147–159, 2020.
39. Takaaki Murata, Kai Fukami, and Koji Fukagata. Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. Journal of Fluid Mechanics, 882:A13, 2020.
40. Tan Nguyen, Richard Baraniuk, Andrea Bertozzi, Stanley Osher, and Bao Wang. MomentumRNN: Integrating momentum into recurrent neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 1924–1936. Curran Associates, Inc., 2020.
41. Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Lió. On second order behaviour in augmented neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 5911–5921. Curran Associates, Inc., 2020.
42. Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In International Conference on Machine Learning, pages 1310–1318, 2013.
43. Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 2(11):559–572, 1901.
44. Boris T Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.
45. Carlos J. G. Rojas, Andreas Dengel, and Mateus Dias Ribeiro. Reduced-order Model for Fluid Flows via Neural Ordinary Differential Equations. arXiv:2102.02248 [physics], February 2021. arXiv: 2102.02248.
46. Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.
47. Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.
48. Omer San and Romit Maulik. Neural network closures for nonlinear model order reduction. arXiv preprint arXiv:1705.08532, 2017.
49. Omer San and Romit Maulik. Machine learning closures for model order reduction of thermal fluids. Applied Mathematical Modelling, 60:681–710, 2018.
50. Omer San, Romit Maulik, and Mansoor Ahmed. An artificial neural network framework for reduced order modeling of transient flows. Communications in Nonlinear Science and Numerical Simulation, 77:271–287, 2019.
51. Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. Journal of Fluid Mechanics, 656:5–28, 2010.
52. G. I. Sivashinsky. On flame propagation under conditions of stoichiometry. SIAM Journal on Applied Mathematics, 39(1):67–82, 1980.
53. G.I. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames—i. derivation of basic equations. Acta Astronautica, 4(11):1177–1206, 1977.
54. Tao Sun, Huaming Ling, Zuoqiang Shi, Dongsheng Li, and Bao Wang. Training deep neural networks with adaptive momentum inspired by the quadratic optimization. arXiv preprint arXiv:2110.09057, 2021.

55. Bao Wang, Tan M Nguyen, Andrea L Bertozzi, Richard G Baraniuk, and Stanley J Osher. Scheduled restart momentum for accelerated stochastic gradient descent. arXiv preprint arXiv:2002.10583, 2020.
56. Bao Wang, Hedi Xia, Tan Nguyen, and Stanley Osher. How does momentum benefit deep neural networks architecture design? a few case studies. arXiv preprint arXiv:2110.07034, 2021.
57. Bao Wang and Qiang Ye. Stochastic gradient descent with nonlinear conjugate gradient-style adaptive momentum. arXiv preprint arXiv:2012.02188, 2020.
58. Mingliang Wang, Han-Xiong Li, Xin Chen, and Yun Chen. Deep learning-based model reduction for distributed parameter systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 46(12):1664–1674, 2016.
59. Hedi Xia, Vai Suliafu, Hangjie Ji, Tan Nguyen, Andrea Bertozzi, Stanley Osher, and Bao Wang. Heavy ball neural ordinary differential equation. In Advances in Neural Information Processing Systems, volume 34. Curran Associates, Inc., 2021.
60. Donghyun You and Parviz Moin. A dynamic global-coefficient subgrid-scale eddy-viscosity model for large-eddy simulation in complex geometries. Physics of Fluids, 19(6):065110, 2007.