RECONSTRUCTING A DYNAMICAL SYSTEM AND FORECASTING TIME SERIES BY SELF-CONSISTENT DEEP LEARNING

A PREPRINT

Zhe Wang
Energy research institute @NTU
Nanyang Technological University
637141 Singapore
zhe.wang@ntu.edu.sg

Claude Guet
Energy research institute @NTU
Nanyang Technological University
637141 Singapore
and School of Materials Science and Engineering
Nanyang Technological University
639798 Singapore
cguet@ntu.edu.sg

August 5, 2021

ABSTRACT

We introduce a self-consistent deep-learning framework which, for a noisy deterministic time series, provides unsupervised filtering, state space reconstruction, identification of the underlying differential equations and forecasting. Without a priori information on the signal, we embed the time series in a state space, where deterministic structures, i.e. attractors, are revealed. Under the assumption that the evolution of solution trajectories is described by an unknown dynamical system, we filter out stochastic outliers. The embedding function, the solution trajectories and the dynamical systems are constructed using deep neural networks, respectively. By exploiting the differentiability of the neural solution trajectory, the neural dynamical system is defined locally at each time, mitigating the need for propagating gradients through numerical solvers. On a chaotic time series masked by additive Gaussian noise, we demonstrate the filtering ability and the predictive power of the proposed framework.

Keywords Neural ordinary differential equations, State space reconstruction, Filtering and forecasting

1 Introduction

Time series analysis and time series forecasting have been studied to extract information about the data and the underlying dynamics, and to predict the future of observables from past measurements. The first systematic modeling of time series dates back to 1927 when Yule [1927] introduced a linear autoregression model to reveal the dynamics of sunspot numbers. The model which writes as

$$u(k+1) = \sum_{n=0}^{m-1} \boldsymbol{a}(n)u(k-n) + \boldsymbol{e}(k),　\tag{1}$$

takes the form of a linear difference equation, which states that the future $u(k+1)$ is a weighted sum of the past values in the sequence, with the error term $\boldsymbol{e}(k)$. Here, m denotes the regression order.

Since linear equations lead to only exponential or periodic motions, analyzing systems characterized by irregular motions calls for nonlinear autoregression models which advantageously employ neural networks. Retaining the basic form of Eq. (1), neural autoregression models can be expressed as

$$u(k+1) = \mathcal{N}[u(k), u(k-1), ..., u(k-(m-1)); \boldsymbol{\theta}] + \boldsymbol{e}(k),　\tag{2}$$

where the model parameters $\boldsymbol{\theta}$ of the neural network, denoted by $\mathcal{N}(\cdot)$, are determined by minimizing the deviation $\boldsymbol{e}(k)$. Since the first application of neural networks to reconstruct the governing equations underlying a chaotic time

series by Lapedes and Farber [1987], various network architectures including multilayer perceptrons [Weigend et al., 1990, Weigend, 1991], time delayed neural networks [Wan, 1994, Saad et al., 1998], convolution neural networks [Borovykh et al., 2017], recurrent neural networks [Gers et al., 2002, Mirowski and LeCun, 2009, Dubois et al., 2020], and more recently transformers [S. Li et al., 2019, Lim et al., 2021], have been explored to model time series arising from physics, engineering, biological science, finance,...

Using neural networks as a nonlinear autoregression model stems from the universal approximation theorem which states that a sufficiently deep neural network can approximate any arbitrary well-behaved nonlinear function with a finite set of parameters [Gorban and Wunsch, 1998, Winkler and Le, 2017, Lin and Jegelka, 2018] and from the Takens [1981] theorem in the following manner. Let $\boldsymbol{v}(t)$ be a state vector on the solution manifold and let

$$\frac{\mathrm{d}\boldsymbol{v}(t)}{\mathrm{d}t} = f[\boldsymbol{v}(t)], \tag{3}$$

be the governing equation. One seldom has access to $\boldsymbol{v}(t)$. Instead, the state of dynamical systems is partially observed and inferred through a sequence of scalar measurements $u(k)$ sampled at discrete times and, frequently, masked by noise. For a sufficiently large dimension $m \in \mathbb{Z}^+$ and an arbitrary delay time $\tau \in \mathbb{R}^+$, Takens [1981] theorem affirms the existence of a diffeomorphism between a delay vector $\tilde{\boldsymbol{u}}$ and the underlying state $\boldsymbol{v}(t)$ of the dynamical system. This implies that there exists a nonlinear mapping $u(k+1) = g[\tilde{\boldsymbol{u}}(k)]$ which models the time series exactly. In virtue of the universal approximation theorem, neural networks could capture the $g(\cdot)$ mapping.

Going beyond merely finding the underlying difference equations (2), inspired by recent works in learning differential equations [Chen et al., 2018, Ayed et al., 2019, Raissi et al., 2019, Rackauckas et al., 2020], we propose to reconstruct a latent continuous-time dynamical system

$$\frac{\mathrm{d}\boldsymbol{u}(t)}{\mathrm{d}t} = \boldsymbol{A} \cdot \boldsymbol{u}(t) + \boldsymbol{e}(t) \quad \text{with} \quad \boldsymbol{A} \in \mathbb{R}^{m^2}, \tag{4}$$

for the evolution of the solution trajectory $\boldsymbol{u}(t)$ in a reconstructed state space

$$\boldsymbol{u}(t) \approx \mathcal{E}[u(t)] \quad \text{with} \quad \mathcal{E} : \mathbb{R} \to \mathbb{R}^m. \tag{5}$$

The discrete measurements are parameterized by a continuous and differentiable function $u(t)$.

The aim of this paper is to propose an algorithmic scheme enabling the reconstruction of the latent continuous dynamics. Through a self-consistent process to be discussed in Sec. 3, the matrix $\boldsymbol{A}$, the embedding $\mathcal{E}(\cdot)$, and the fitting function $u(t)$ are constructed using deep neural networks, which are learned jointly. The proposed scheme is tested on a synthetic time series, which is sampled from the Lorenz attractor [Lorenz, 1963] and masked by additive Gaussian noise, in Sec. 4. Finally, the limitations of the proposed scheme are discussed in Sec. 5, where conclusions are drawn.

2 Related works

State Space reconstruction. State space reconstruction with the time delayed vector was first proposed by Ruelle (cf. Weigend and Gershenfeld [1994]) and Packard et al. [1980], and later proved by Takens [1981]. In order to remove noise, Sauer [1994] applied a low-pass filter to the delayed vector space. Recently, [Jiang and He, 2017, Lusch et al., 2018, Gilpin, 2020, Ouala et al., 2020] have shown that the state space reconstruction from noisy data can be achieved using an autoencoder as the embedding function $\mathcal{E}(\cdot)$. Gilpin [2020] incorporated the false-nearest-neighbor (FNN) algorithm [Kennel et al., 1992] into a loss function to penalize the encoder outputs in the redundant dimensions. As a consequence, the reconstructed attractor is confined to a subspace smaller than the configuration space. However, by incorporating the FNN algorithm into the loss function, one penalizes the dynamics not only in the redundant dimensions, but also on the solution manifold, leading to a dimensionality collapse of the reconstructed attractor reported in [Gilpin, 2020].

Neural ordinary differential equation (ODE). The canonical approach for learning neural ODEs from data uses the adjoint method and calls for numerical solvers [Chen et al., 2018, Ayed et al., 2019, Rackauckas et al., 2020]. Depending on the selected numerical schemes, the differential equation (4) allows for various difference representations, signifying an error. Numerical error accumulates, or even diverges, as one iterates over time, and ultimately affects the loss function. Moreover, in order to obtain the gradients of the loss with respect to network parameters, one needs to solve Eq. (4) forwards and a corresponding adjoint ODE backwards, at each iteration. This can be computationally prohibitive for complex network architectures.

3 Methods

3.1 Fitting and filtering

Time series data are often unevenly spaced. In order to obtain a continuous limit, we parameterize the measured time series using neural networks

$$u(t) = \mathcal{N}_u(t; \boldsymbol{\theta}_u), \quad \mathcal{N}_u(\cdot) : \mathbb{R} \to \mathbb{R}. \tag{6}$$

Let $s_N = [s(1), s(2), ..., s(N)]$ be a segment of the time series measured at times $[t_1, t_2, ..., t_N]$, respectively. The deviation loss associated with the fitting which reads

$$L_{\text{fit}} = \frac{1}{N\sigma_s^2} \sum_{i=1}^{N} [s(i) - \mathcal{N}_u(t_i; \boldsymbol{\theta}_u)]^2, \tag{7}$$

is normalized by the batch variance, $\sigma_s^2 = \frac{1}{N} \sum_{i=1}^{N} [s(i) - \text{mean}(s_N)]^2$.

For noisy datasets, a direct minimization of L_{fit} with respect to $\boldsymbol{\theta}_u$ will lead to an overfitting. Assuming that the dynamics of $u(t)$ is characterized by the underlying neural ODE (4), we include the deviation $e(t)$ as a regularizer

$$L_{\text{ode}} = \frac{1}{Md} \sum_{i=1}^{M} \sum_{j=1}^{m} [e_j(t_i)]^2, \tag{8}$$

where the embedding dimension, $d \leq m$, is learned during the training. The fitting process is self consistent as the error vector $e(t) \in \mathbb{R}^m$ is an implicit function of $\boldsymbol{\theta}_u$. Moreover, we augment the data by randomly sampling M points from $u(t)$ for $t \in [t_1 + (m-1)\tau, t_N]$ at each iteration, where the interval $(m-1)\tau$ stems from the delay vector. A successive re-sampling covers the entire solution manifold of Eq. (4). We defer our discussion on the learning algorithm for $\mathcal{E}(\cdot)$ and d to Sec. 3.2 and on the functional form of $e(t)$ to Sec. 3.3, respectively. Therefore, the total loss function for $\mathcal{N}_u(\cdot)$ is

$$L_u = L_{\text{fit}} + \lambda_u L_{\text{ode}}, \tag{9}$$

where the weight λ_u is a hyper-parameter. A joint minimization of L_{fit} and L_{ode} ensures the smoothness of the solution trajectory and filters out additive noise.

3.2 State space reconstruction

Adopting the idea from Gilpin [2020], we start with a reasonably large value m for the configuration space, $\mathbb{R}^m$, where Eq. (4) lives, and search for an embedding subspace $\mathbb{R}^d$, with $d \leq m$, which contains the attractor. Given $u(t)$, the delay vector can be expressed as

$$\tilde{\boldsymbol{u}}(t)^T = [u(t), u(t - \tau), ..., u(t - (m-1)\tau)]. \tag{10}$$

To treat the redundancy associated with large values of m, we calculate the fraction of false nearest neighbors γ, a heuristic first proposed by Kennel et al. [1992]. False neighbors of a trajectory point in too low an embedding dimension will separate as the embedding dimension increases until all neighbors are real. Therefore, an appropriate embedding dimension d can be inferred by examining how γ varies as a function of dimension. See supplementary material for details.

Let $\boldsymbol{\gamma} = [\gamma_1, ..., \gamma_m]$ be a list containing the fraction of false nearest neighbors for each subspace in $\mathbb{R}^m$. Instead of incorporating $\boldsymbol{\gamma}$ into a loss function [Gilpin, 2020], we introduce a binary mask

$$\boldsymbol{w}^T = \lceil \text{relu}(\boldsymbol{\gamma} - \epsilon) \rceil = [\underbrace{1, ..., 1}_{d}, \underbrace{0, ..., 0}_{m-d}], \tag{11}$$

which decomposes the configuration space $\mathbb{R}^m$ into an embedding $\mathbb{R}^d$ and $(m-d)$ redundant dimensions. Here, $\lceil \cdot \rceil$ denotes the ceiling function. Following Kennel et al. [1992], we take $\epsilon = 0.01$.

Let us consider a state space reconstruction using either the method of delay where the state vector is:

$$\boldsymbol{u} = \boldsymbol{w}^T \tilde{\boldsymbol{u}}, \tag{12}$$

or the autoencoder where the bottleneck is created by the mask $\boldsymbol{w}^T$

$$\text{Encoder}: \quad \boldsymbol{u}(t) = \boldsymbol{w}^T \mathcal{N}_e[\tilde{\boldsymbol{u}}(t); \boldsymbol{\theta}_e], \quad \mathcal{N}_e(\cdot) : \mathbb{R}^m \to \mathbb{R}^m, \tag{13a}$$

$$\text{Decoder}: \quad \hat{\boldsymbol{u}}(t) = \mathcal{N}_d[\boldsymbol{u}(t); \boldsymbol{\theta}_d], \quad \mathcal{N}_d(\cdot) : \mathbb{R}^m \to \mathbb{R}^m. \tag{13b}$$

The inclusion of $\boldsymbol{w}^T$ compresses the outputs of $\mathcal{N}_e(\cdot)$ to the embedding, while the decoder ensures information conservation. The associated reconstruction loss is

$$L_{\text{rec}} = \frac{d}{Mm\sum_{i=1}^{d}\sigma_{u_i}^2}\sum_{i=1}^{M}\sum_{j=1}^{m}\left[\hat{u}_j(t_i) - \tilde{u}_j(t_i)\right]^2, \tag{14}$$

where $\sigma_{\boldsymbol{u}}$ denotes the standard deviation of $\boldsymbol{u}$ in the batch direction. To enforce an isotropic expansion of the attractor, we consider minimizing the following loss function

$$L_{\text{exp}} = \frac{2}{d(d-1)}\sum_{i=1}^{d-1}\sum_{j=i+1}^{d}K_{u_i u_j}^2 + \frac{1}{d}\sum_{i=1}^{d}\left[\sigma_{u_i} - \text{mean}(\sigma_{\boldsymbol{u}})\right]^2, \tag{15}$$

where $\boldsymbol{K_{uu}} \in \mathbb{R}^{m\times m}$ denotes the covariance matrix of $\boldsymbol{u}$, such that the outputs of the encoder span an orthogonal basis, while the second term forces the unfolding to be isotropic. Once more, we include L_{ode} as a regularizer. Thus, with the weights $\lambda_{e,1}$ and $\lambda_{e,2}$ being hyper-parameters, the loss functions for the encoder and the decoder are, respectively

$$L_e = L_{\text{rec}} + \lambda_{e,1}L_{\text{ode}} + \lambda_{e,2}L_{\text{exp}} \quad \text{and} \quad L_d = L_{\text{rec}}. \tag{16}$$

3.3 Neural dynamical system

To confine the dynamics of Eq. (4) to the d-dimensional embedding, we introduce an FNN-informed attention $\boldsymbol{w}\cdot\boldsymbol{w}^T$ to the output matrix of a neural network $\mathcal{N}_f(\cdot) : \mathbb{R}^m \to \mathbb{R}^{m^2}$, such that

$$\boldsymbol{A} = (\boldsymbol{w}\cdot\boldsymbol{w}^T)\odot\mathcal{N}_f[\boldsymbol{u}(t);\boldsymbol{\theta}_f] = \begin{bmatrix} a_{11} & \cdots & a_{1d} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{d1} & \cdots & a_{dd} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Given $\boldsymbol{u}(t)$ defined in Sec. 3.2, the deviation vector is given by

$$\boldsymbol{e}(t) = \frac{d\boldsymbol{u}(t)}{dt} - \boldsymbol{A}\cdot\boldsymbol{u}(t) = \frac{d\boldsymbol{u}(t)}{dt} - \left\{(\boldsymbol{w}\cdot\boldsymbol{w}^T)\odot\mathcal{N}_f[\boldsymbol{u}(t);\boldsymbol{\theta}_f]\right\}\cdot\boldsymbol{u}(t), \tag{17}$$

where the time derivative is calculated using auto-differentiation [Baydin et al., 2017].

The dimensionality reduction from the configuration space to the embedding one is a common feature of dissipative systems [Temam, 2012]. Thus, we impose that the divergence of the vector field $\boldsymbol{F}(\boldsymbol{u}) = \boldsymbol{A}\cdot\boldsymbol{u}$ is negative. Being a global property of the dynamical system (4), at each iteration, we sample the configuration space with M points, denoted by $\boldsymbol{u}_s$, leading to a loss function

$$L_{\text{div}} = \frac{1}{Md}\sum_{i=1}^{M}\left\{\text{relu}\left[\text{div}\boldsymbol{F}(\boldsymbol{u}_s(t_i))\right]\right\}^2. \tag{18}$$

With λ_f being a hyper-parameter, a joint minimization of L_{ode} and L_{div}

$$L_f = L_{\text{ode}} + \lambda_f L_{\text{div}}, \tag{19}$$

enforces the long-term dynamics of solution trajectories on the attractor.

3.4 Algorithm

The schematic algorithm 1 shown below brings together all pieces introduced in the previous sections. For neural dynamical systems whose state vector is reconstructed using the method of delay (12), one mitigates the need for learning an autoencoder (13).

Algorithm 1 Joint learning for neural dynamical systems with autoencoder

---

Input: Mini-batched training samples $\{s_N[1], s_N[2], ..., s_N[S]\}$ and the corresponding time labels.
Guess initial parameters $\{\boldsymbol{\theta}_u[1], \boldsymbol{\theta}_u[2], ..., \boldsymbol{\theta}_u[S]\}$, $\boldsymbol{\theta}_e, \boldsymbol{\theta}_d\ \boldsymbol{\theta}_f$ and initialize $\boldsymbol{\gamma} = \boldsymbol{w}^T = [1, ..., 1]$.

1: **while** not converged **do**
2:　　Create an empty list: U.
3:　　**for** $i = 1, ..., S$ **do**
4:　　　　Randomly draw M samples from $u(t)[i]$ and append to list U.
5:　　　　Randomly draw M samples from the configuration space†.
6:　　　　Compute loss functions: $L_{\text{fit}}, L_{\text{rec}}, L_{\text{ode}}$ and L_{div}.
7:　　　　Optimize $\boldsymbol{\theta}_u[i]$ using gradients: $\partial_{\boldsymbol{\theta}_u[i]} L_{\text{fit}} + \lambda_u \partial_{\boldsymbol{\theta}_u[i]} L_{\text{ode}}$.
8:　　　　Optimize $\boldsymbol{\theta}_e$ using gradients: $\partial_{\boldsymbol{\theta}_e} L_{\text{rec}} + \lambda_{e,1} \partial_{\boldsymbol{\theta}_e} L_{\text{ode}} + \lambda_{e,2} \partial_{\boldsymbol{\theta}_e} L_{\text{exp}}$.
9:　　　　Optimize $\boldsymbol{\theta}_d$ using gradients: $\partial_{\boldsymbol{\theta}_d} L_{\text{rec}}$.
10:　　　　Optimize $\boldsymbol{\theta}_f$ using gradients: $\partial_{\boldsymbol{\theta}_f} L_{\text{ode}} + \lambda_f \partial_{\boldsymbol{\theta}_f} L_{\text{div}}$.
11:　　**end for**
12:　　Compute epoch-wise $\hat{\boldsymbol{\gamma}}$ using U and update $\boldsymbol{\gamma}$ using a moving average: $\boldsymbol{\gamma} = (1 - \alpha)\boldsymbol{\gamma} + \alpha\hat{\boldsymbol{\gamma}}$.
13:　　Compute and update $\boldsymbol{w}^T$ using Eq. (11).
14: **end while**
　　Output: Optimized parameters $\{\boldsymbol{\theta}_u[1], \boldsymbol{\theta}_u[2], ..., \boldsymbol{\theta}_u[S]\}$, $\boldsymbol{\theta}_e, \boldsymbol{\theta}_e, \boldsymbol{\theta}_f$ and the weights $\boldsymbol{w}^T$.
Hyper-parameters in this paper: $S = 128$, $M = 64$, $\lambda_u = \lambda_{e,1} = \lambda_{e,2} = \lambda_f = 1$ and $\alpha = 0.1$.
† The configuration space is confined by the output activation function of the encoder, e.g. `tanh`.
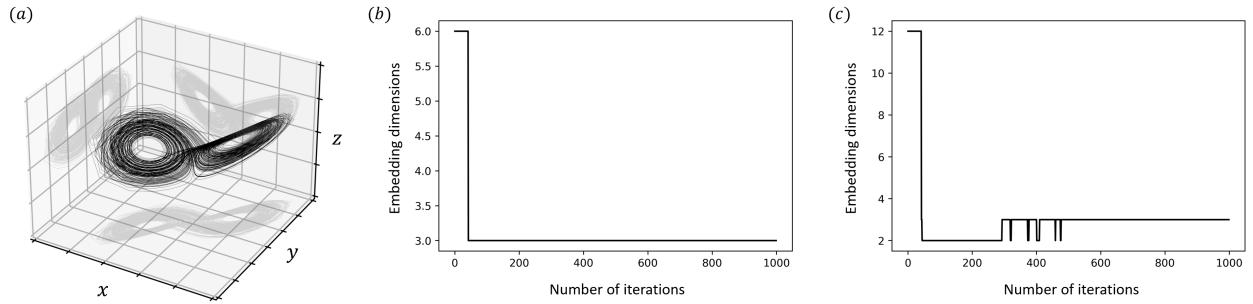

Figure 1: (a) Lorenz attractor. Dimension $d = 3$ obtained from the FNN method for embedding in an (b) $m = 6$; and an (c) $m = 12$ dimensional configuration space over 1000 iterations.

4   Experiments with synthetic time series

State space reconstruction [Sauer, 1994, Jiang and He, 2017, Ouala et al., 2020, Gilpin, 2020] and recovered neural ODEs [Chen et al., 2018] are commonly assessed by visualizing the reconstructed attractor and by forecasting the continuation of the time series. In all herein cited works, synthetic time series generated by solving the Lorenz [1963] equations

$$\frac{dx}{dt} = \sigma(y - x), \quad \frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y, \quad \frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z, \tag{20}$$

with $\sigma = 10$, $\rho = 28$, $\beta = 8/3$, were considered. Our dataset is the x-coordinate of a trajectory generated by solving Eqs. (20) for $10,200$ time steps from one initial condition $(x, y, z) = (0, 1, 1.05)$. With a step size $\Delta t = 0.05$, a visualization of the Lorenz attractor is shown in Fig. 1 (a), where the time series $t \in [0, 510]$ completes one oscillation every 15 to 20 time steps. In addition, the time series are masked by an Gaussian white noise, with η being a ratio:

$$s'[k] = s[k] + N(0, \sigma^2) \quad \text{where} \quad \sigma = \eta\sigma_s. \tag{21}$$

In the model, we employ a residual connection [He et al., 2016] around two sub-layers made of 32 neurons regularized with batch normalization [Ioffe and Szegedy, 2015] and activated by a `tanh` function. Networks $\mathcal{N}_u, \mathcal{N}_e, \mathcal{N}_d$ and $\mathcal{N}_f$ consist of a stacking of 3 and 5 residual blocks, respectively. By rescaling the training dataset to a range $[-1, 1]$, the output activation functions for $\mathcal{N}_u, \mathcal{N}_e, \mathcal{N}_d$ are `tanh`. Since there is no a priori information on $\boldsymbol{A}$ (except for the divergence of $\boldsymbol{A} \cdot \boldsymbol{u}$), we use a linear activation for the output of $\mathcal{N}_f$. Then, we corrupt the input of the encoder with
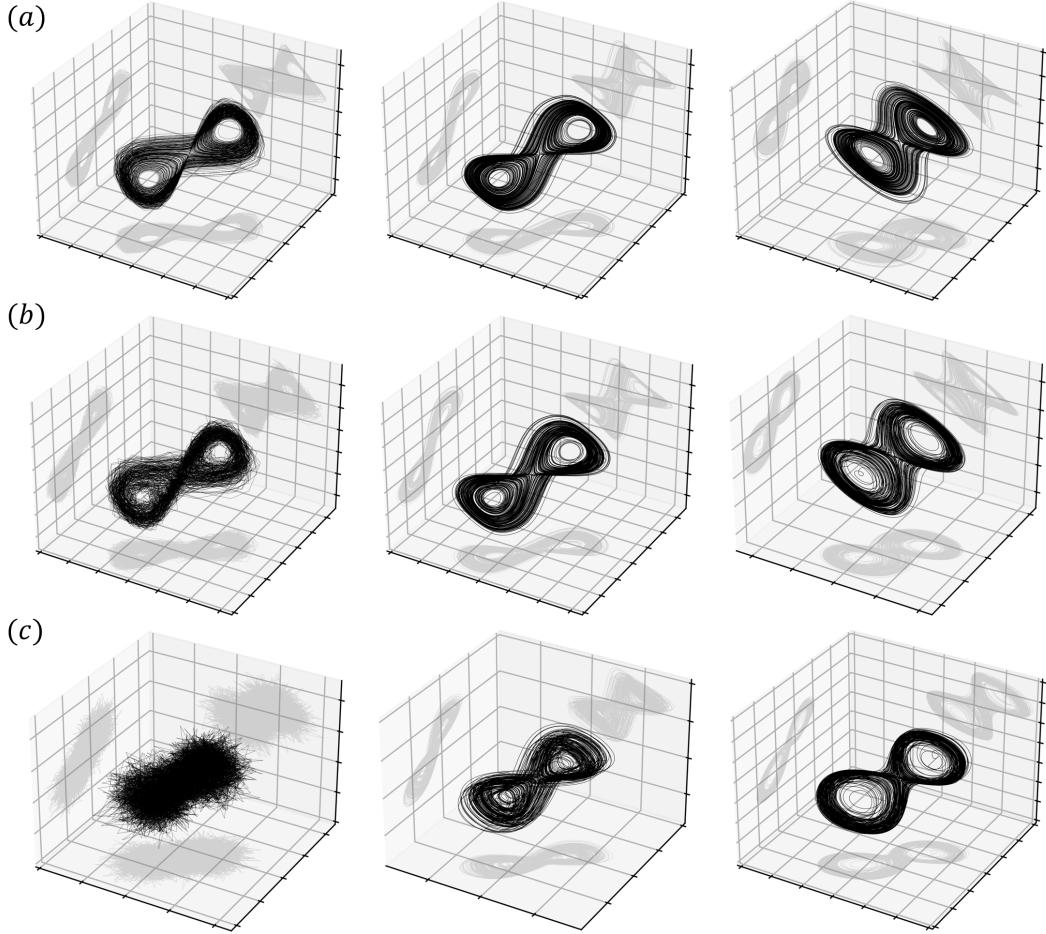
Figure 2: Delay attractors reconstructed from noisy measurements (Left) or the parameterization $u(t)$ (Middle). Right: latent attractors reconstructed using the autoencoder. The time series were masked by Gaussian noise with noise ratio (a) $\eta = 0$; (b) $\eta = 0.15$; and (c) $\eta = 0.3$, cf. Eq. (21)

Gaussian noise $N(0, 0.5^2)$ as in [Vincent et al., 2008, 2010, Gilpin, 2020]; to reduce overfitting, we apply a dropout regularization [Srivastava et al., 2014] with a rate 0.1 just before each network's output layer.

We separated the time series into a training $t \in [0, 490]$, a validation $t \in [490, 500]$ and a test $t \in [500, 510]$ set; the training set was divided into 128 batches with an $(m-1)\tau$ overlapping between each batch. The FNN-algorithm converged to and remained on a $d = 3$ embedding during the training, cf. Fig. 1 (b) and (c). Without loss of generality, we selected $m = 6$ and $\tau = 0.1$. We trained our model using the ADAM optimizer Kingma and Ba [2015] and on a NVIDIA 2080Ti GPU; iterating over an epoch took around 2 and 4 seconds without and with the autoencoder, respectively. Thus, we pretrained the models $\mathcal{N}_u$ and $\mathcal{N}_f$ with $\boldsymbol{u}$ reconstructed using the method of delay (12) for $15,000$ iterations, and then turn on the autoencoder (13) for a fine tuning for $15,000$ iterations. The results were compared with a direct training using the method of delay for $30,000$ iterations. We retained the models with the lowest normalized mean square error (NMSE) on the validation set

$$\text{NMSE} = \sum_{i=1}^{N} (\text{truth}_i - \text{prediction}_i)^2 / \sum_{i=1}^{N} \left[\text{truth}_i - \text{mean}(\text{truth})\right]^2 . \tag{22}$$

4.1 Visualization of the latent attractor

Fig. 2 shows the reconstructed attractors from noisy measurements. Even for the case $\eta = 0.3$, the L_{ODE} regularizer enables us to recover an attractor resembling that observed in noise-free conditions. Note that the filtering process can occur at two stages. First, the regularized parameterization removes, in principle, outliers that are not governed by the

latent ODEs (4). This features a physics-informed filtering, despite the underlying physics is not known a priori, but it is learned in parallel with the filtering process. Second, since the reconstruction loss, cf. Eq. (14), is not exactly zero at the end of the training, the deviation of the decoded signal from $u(t)$ may contribute to another filtering process. In order to assess the filtering ability, we compare in Table 1 the NMSE of the noisy signal, the learned $u(t)$, and the output of the decoder with the noise-free Lorenz time series. An overall 80% noise reduction has been achieved; however, such a reduction is mainly due the inclusion of L_{ode} during the trajectory fitting.

Table 1: Normalized mean square error of the two-stage filtering process

Case	Raw measurements	$u(t)$	$\mathcal{N}_d\{\mathcal{N}_e[\tilde{\boldsymbol{u}}(t)]\}$
$\eta = 0.00$	0.0	5.8×10^{-5}	9.6×10^{-5}
$\eta = 0.15$	2.3×10^{-2}	2.6×10^{-3}	2.6×10^{-3}
$\eta = 0.30$	9.1×10^{-2}	1.6×10^{-2}	1.3×10^{-2}

For the Lorenz system, each time the signal x goes to zero, the trajectory passes through a sensitive region, i.e. the center, of the attractor where the signal is susceptible to error [Sauer, 1994, Dubois et al., 2020, Brunton et al., 2017]. Compared with delay attractors and works using autoencoders [Jiang and He, 2017, Gilpin, 2020], the L_{ODE}-regularized encoder unfolds the central region of the attractor, which may be the reason for the enhanced forecasting horizon discussed in Sec. 4.2.

4.2   Continuation of the training time series

Two neural dynamical systems were identified in the state spaces reconstructed either with method of delay (12) or with the autoencoder (13). For both systems, we first solved Eq. (4) with given initial conditions $\boldsymbol{u}(t_0)$ in the configuration space, and then convert the solutions to the measurement space. With the method of the delay, the conversion reduces to an identity operation; whereas with the autoencoder, the conversion is made using the decoder (13b). Taken $t_0 = 500$, we predict the continuation of the time series for the next 200 time steps with $\Delta t = 0.05$, and compare both predictions with the exact solution of the Lorenz model, see Fig. 3. Our best result was obtained when integrating the neural dynamical system reconstructed in a latent state space and trained on a noise-free dataset. As Fig. 3 shows, the predicted continuation remains close to the exact signal up to 11 Lorenz time, which is at least 2 times longer than previously achieved in [Sauer, 1994, Chen et al., 2018, Dubois et al., 2020, Gilpin, 2020] and on par with Ouala et al. [2020] who implemented a neural autoregression model, cf. Eq. (2) in the latent space. The inclusion of noise leads to an deterioration in the prediction horizon, as expected. With increasing noise ratio, our algorithm fails finding the precise initial conditions. The sensitive dependence on initial conditions of chaotic dynamical systems prohibits forecasting.

Fig. 4 shows continuations of the noise-free training dataset $t \in [0, 490]$ into the future. The initial condition at, e.g. $t_0 = 600$, is determined from measurements in an interval $t \in [599, 601]$ by minimizing L_u, cf. Eq. (9). The parameters $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_e$ and $\boldsymbol{\theta}_d$ are fixed during the inference phase. Despite a degradation of prediction horizon for all cases, a better performance is achieved for state space reconstruction using the autoencoder, coming with the cost of doubled computation time.

4.3   Forecasting from different initial conditions

To assess the transferability of our prediction model, we study forecasting for time series initiated with different initial conditions in the training dataset. The Lorenz model (20) is integrated with given initial conditions for the period $t \in [0, 110]$, wherein the interval $t \in [99, 101]$ is used to determine the state vector $\boldsymbol{u}(t_0)$ at $t_0 = 100$. Then, the reconstructed dynamical system is solved for $t \in [100, 110]$. The results are shown in Fig. 5. Compared with direct continuations (Figs. 3, 4), the deterioration in prediction horizon suggests that the reconstructed dynamical system overfits to the training dataset, which is seeded with a single initial condition. Therefore, should a reconstruction of the universal dynamical system be the goal, one needs to augment the training dataset with time series seeded with different initial conditions. Nevertheless, compared with the method of delay, our results show that state space reconstruction using the regularized autoencoder (13) mitigates the overfitting.
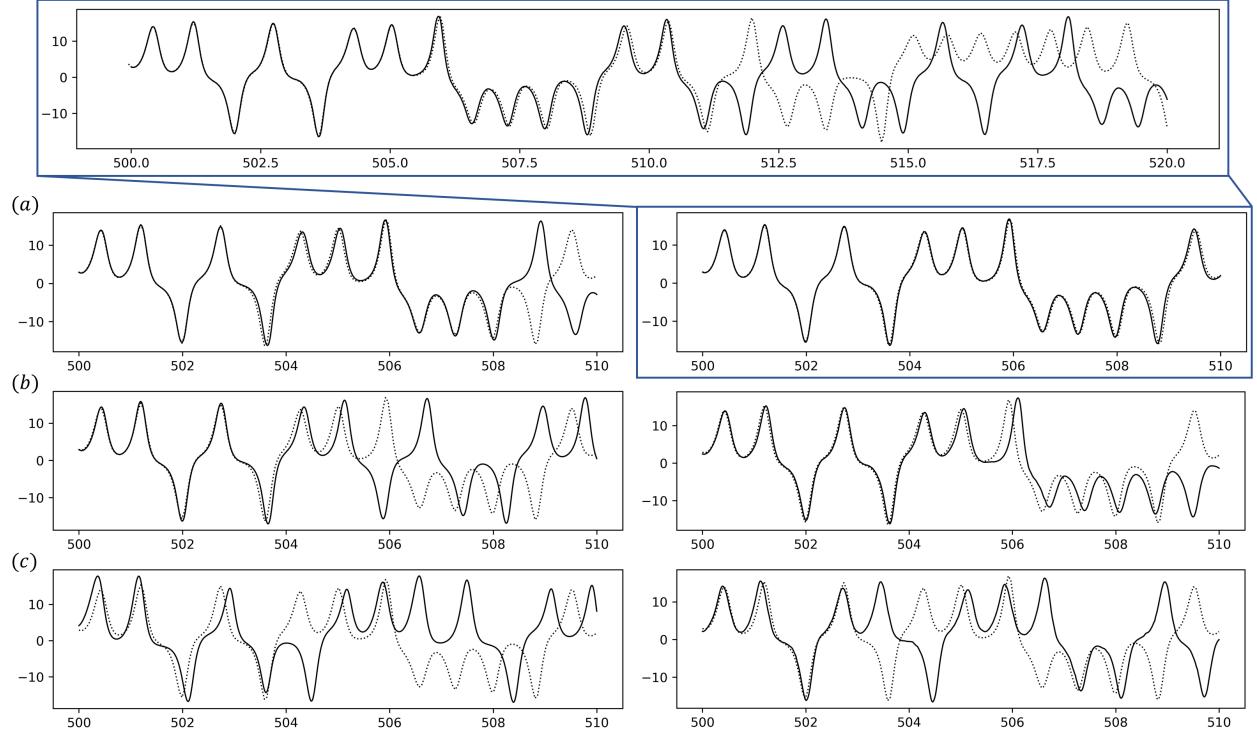
Figure 3: Time series prediction for signal from Lorenz attractor masked by an additive Gaussian noise with (a) $\eta = 0$; (b) $\eta = 0.15$; and (c) $\eta = 0.3$ without (left) and with (right) the autoencoder. Solid curve denotes the predicted time series and dashed curve is the true time series. The inset reveals that with the autoencoder and a clean training dataset $\eta = 0$, the prediction stays near the correct signal for around 220 time steps with $\Delta t = 0.05$.

5   Discussion and conclusions

We have introduced a general method which allows us to filter a noisy time series without a priori information on the measurement, to reconstruct an attractor from the filtered results, and to learn a latent dynamical system which underlies the time series in a self-consistent manner. First, the measured time series is parameterized as a continuous function $u(t)$ using deep neural networks. The measurement is then augmented and used to reconstruct the state space using either the method of delay and an autoencoder. The embedding dimension, which creates a bottleneck, is automatically searched using the FNN algorithm during the training. Finally, an approximated dynamical system underlying the temporal evolution serves as a regularizer for $u(t)$ to filter out the noise and for the encoder to determine an optimal embedding, completing the self-consistent cycle.

The proposed framework has been tested by forecasting the continuation of an univariate time series sampled from the Lorenz attractor. To the best of our knowledge, our prediction horizon is significantly longer than most published ones. Moreover, with an additive Gaussian noise, an overall 80% noise reduction is achieved for a chaotic signal. While we adopted the same architecture for all neural networks and keep the regularizer strength constant $\lambda_u = \lambda_{e,1} = \lambda_{e,2} = \lambda_f = 1$, Gilpin [2020], Wang et al. [2021a,b] recently showed that a careful tuning of the regularizer strength and a selection of the network architecture can speed up the convergence and improve the quality of the learned model to a large degree. Thus, we trust that our results can still be improved. It also remains to test the algorithm on a broader class of synthetic deterministic time series. Eventually, we hope to apply it as an auxiliary tool to explore hidden physics from experimental data and to extract relevant information from direct numerical simulations.

The fundamental assumption underlying this work is that the time series is governed by some unknown ordinary differential equations. Therefore, our algorithm is only applicable to deterministic time series masked by additive noise. Being aware that most time series of practical interest, e.g. financial data, are stochastic, one needs to replace Eq. (4) by stochastic differential equations and reformulate the loss functions, accordingly.
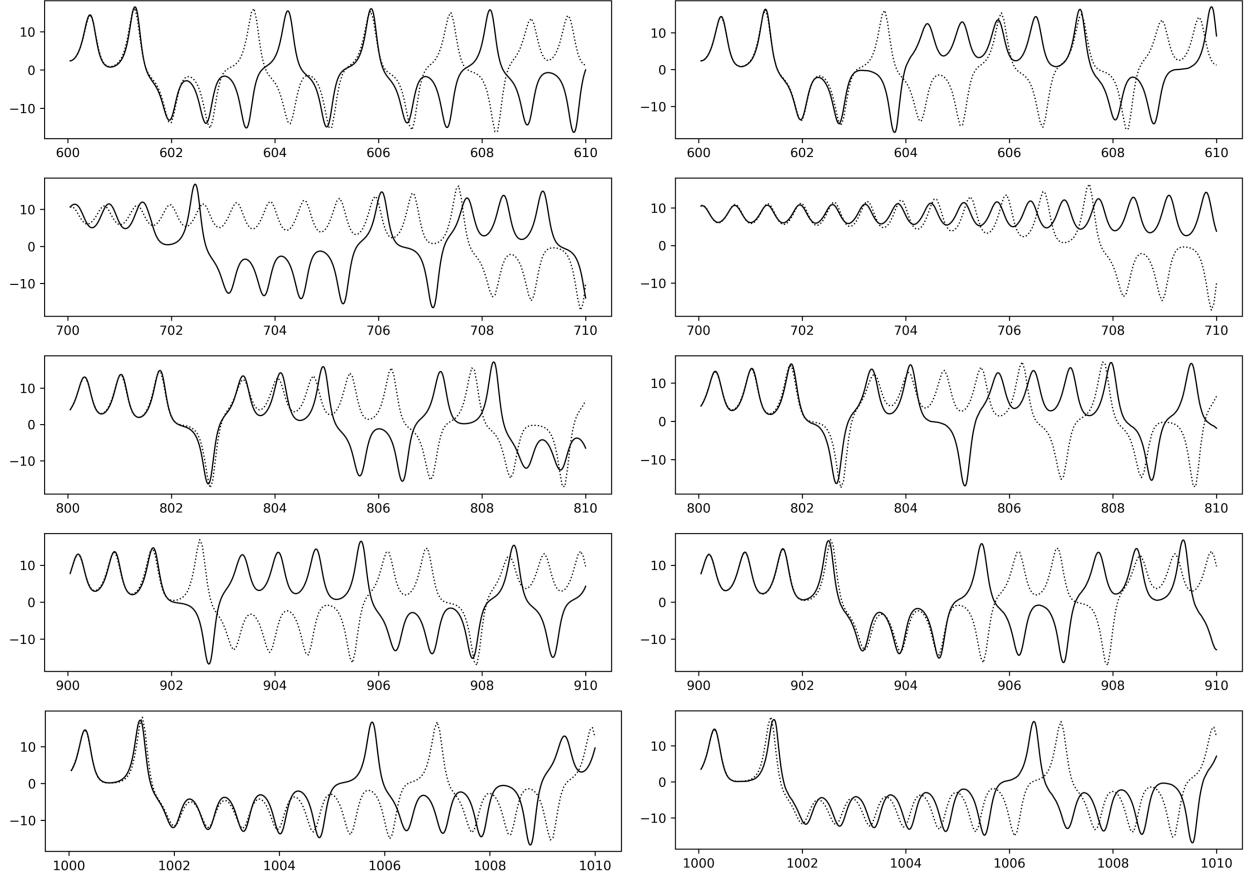
Figure 4: Prediction for five successive continuations of the training dataset $t \in [0, 490]$ further into the future without (left) and with (right) the autoencoder. The solid curve is the predicted continuation and the dashed curve denotes the ground truth.

Acknowledgment

Appendix

A   Calculation for the fraction of false nearest neighbors

Consider a random sampling of M points from an m-dimensional delay vector to form a batch

$$
\begin{bmatrix}
u(t_1) & u(t_1 - \tau) & \dots & u(t_1 - (m-1)\tau) \\
u(t_2) & u(t_2 - \tau) & \dots & u(t_2 - (m-1)\tau) \\
\vdots & \vdots & \ddots & \vdots \\
u(t_M) & u(t_M - \tau) & \dots & u(t_M - (m-1)\tau)
\end{bmatrix},
\tag{23}
$$

Let us reorganize Eq. (23) into the following form:

$$
\boldsymbol{H} = \left[\begin{bmatrix}
u(t_1) & 0 & \dots & 0 \\
u(t_2) & 0 & \dots & 0 \\
\vdots & \vdots & \ddots & 0 \\
u(t_M) & 0 & \dots & 0
\end{bmatrix}, ..., \begin{bmatrix}
u(t_1) & u(t_1 - \tau) & \dots & u(t_1 - (m-1)\tau) \\
u(t_2) & u(t_2 - \tau) & \dots & u(t_2 - (m-1)\tau) \\
\vdots & \vdots & \ddots & \vdots \\
u(t_M) & u(t_M - \tau) & \dots & u(t_M - (m-1)\tau)
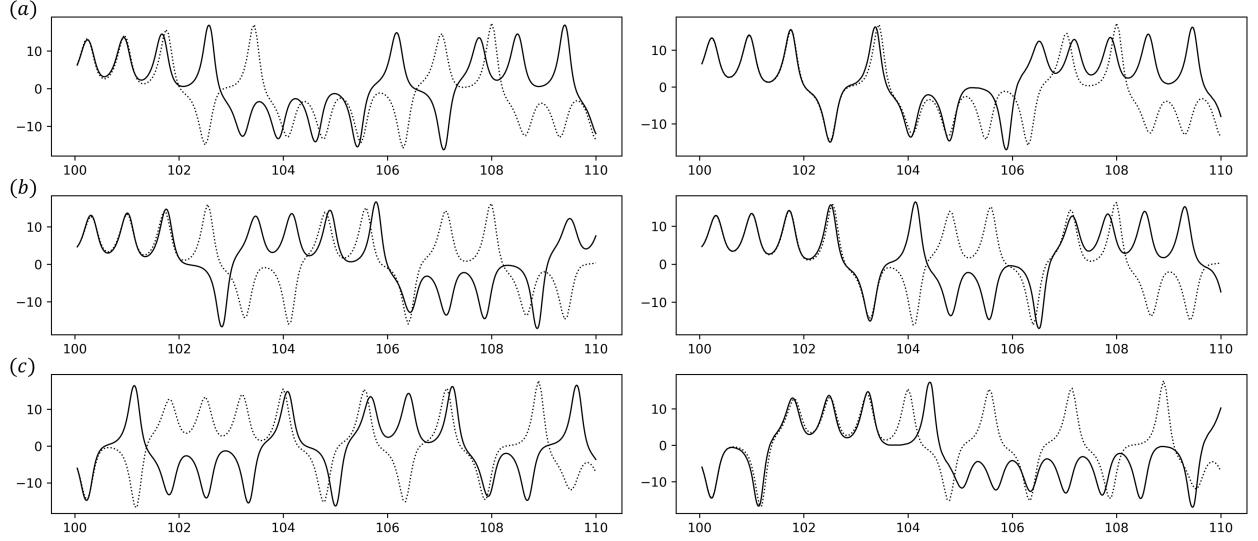\end{bmatrix} \right].
\tag{24}
$$

9

Figure 5: Prediction ($t \in [100, 110]$) for Lorenz time series seeded with different initial conditions ($t = 0$) from the training dataset (a) $(x, y, z) = (1.02, 0.05, -1.67)$; (b) $(x, y, z) = (3.14, -1.59, 2.65)$; and (c) $(x, y, z) = (2.00, 3.00, 4.25)$ without (left) and with (right) the autoencoder. Solid curve denotes the predicted time series and dashed curve is the true time series.

wherein the d-th component of $\boldsymbol{H}$, denoted by $\boldsymbol{H}_d \in \mathbb{R}^{M \times m}$, represents the batched d-dimensional delay vector for $d = 1, ..., m$. The Euclidean length of each delay vectors in the batch $\boldsymbol{H}_d$ is

$$
\boldsymbol{L}_d = \left[\sum_{j=1}^{d} (H_d[i, j])^2 \right]^{1/2} \in \mathbb{R}^{M \times 1}.
\tag{25}
$$

Using $\boldsymbol{H}[d]$ and $\boldsymbol{L}[d]$, the Euclidean distances between each pairs of two vectors in $\boldsymbol{H}[d]$ can be expressed by the following symmetric matrix

$$
\boldsymbol{D}_d = \left[\boldsymbol{L}_d^2 + \left(\boldsymbol{L}_d^2 \right)^T - 2\boldsymbol{H}_d \cdot \boldsymbol{H}_d^T \right]^{1/2} \in \mathbb{R}^{M \times M}.
\tag{26}
$$

Collecting the nearest neighbors, i.e. the smallest off-diagonal elements, along the horizontal direction

$$
R_d[i] = \min \left(D_d[i, j] \right) \quad \text{for} \quad j > i,
\tag{27}
$$

the FNN-algorithm considers the element i as a false nearest neighbor if either of the following two tests fails. They are

$$
\left[\frac{R_d[i]^2 - R_{d-1}[i]^2}{R_d[i]^2} \right]^{1/2} > R_{\text{tol}},
\tag{28a}
$$

$$
\frac{R_d[i]}{R_A} > A_{\text{tol}},
\tag{28b}
$$

for $d \geq 2$. Here the recommended values for the thresholds in Kennel et al. [1992] are $R_{\text{tol}} = 10$ and $A_{\text{tol}} = 2$, respectively, and

$$
R_A^2 = \frac{1}{M} \sum_{i=1}^{M} \left[u(t_i) - \text{mean}[u(t)] \right]^2.
\tag{29}
$$

Denoting

$$
\gamma_d[i] = \begin{cases} 1, & \text{if Eq. (28a) is True or Eq. (28b) is True or } d = 1, \\ 0, & \text{otherwise}, \end{cases}
\tag{30}
$$

10

the fraction of the false nearest neighbors associated with d-dimensional embedding is

$$\gamma_d = \frac{1}{M} \sum_{i=1}^{M} \gamma_d[i]. \tag{31}$$

Therefore, the fraction of false nearest neighbors associated with each sub-dimension $d \leq m$ is

$$\gamma = [\gamma_1, \gamma_2, ..., \gamma_m]. \tag{32}$$

References

G. U. Yule. Vii. on a method of investigating periodicities disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A*, 226:267–298, 1927.

A. Lapedes and R. Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical report, Los Alamos National Laboratory, United States, 1987.

A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *International journal of neural systems*, 1(03):193–209, 1990.

A. S. Weigend. *Connectionist architectures for time series prediction of dynamical systems*. PhD thesis, Stanford University, 1991.

E. A. Wan. Time series prediction by using a connectionist network with internal delay lines. In *Time Series Prediction: forecasting the future and understanding the past*, pages 195–217, 1994.

E. W. Saad, D. V. Prokhorov, and D. C. Wunsch. Comparative study of stock trend prediction using time delay,recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks*, 9(6):1456–1470, 1998.

A. Borovykh, S. Bohte, and C. W. Oosterlee. Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691, 2017.

F. A. Gers, D. Eck, and J. Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, London, 2002.

P. Mirowski and Y. LeCun. Dynamic factor graphs for time series modeling. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, Berlin, Heidelberg, 2009.

P. Dubois, T. Gomez, L. Planckaert, and L. Perret. Data-driven predictions of the lorenz system. *Physica D: Nonlinear Phenomena*, 408:132495, 2020.

S. Li et al. Enhancing the locality and breaking the memory ottleneck of transformer on time series forecasting. In *Advances in NeuralInformation Processing Systems (NeurIPS)*, 2019.

B. Lim, N. Loeff, S. Arik, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 2021.

A. N. Gorban and D. C. Wunsch. The general approximation theorem. In *Proceedings of the International Joint Conference on Neural Networks*, 1998.

D. A. Winkler and T. C. Le. Performance of deep and shallow neural networks, the universal approximation theorem, activity cliffs, and qsar. *Molecular Informatics*, 36(1-2):1600118, 2017.

H. Lin and S. Jegelka. Resnet with one-neuron hidden layers is a universal approximator. In *Advances in Neural Information Processing Systems*, 2018.

F. Takens. Detecting strange attractors in turbulence." dynamical systems and turbulence. In *Dynamical systems and turbulence*, pages 366–381. Springer, Berlin, Heidelberg, 1981.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. arXiv preprint arXiv:1806.07366, 2018.

I. Ayed, E. de Bézenac, A. Pajot, J. Brajard, and P. Gallinari. Learning dynamical systems from partial observations. arXiv preprint arXiv:1902.11136, 2019.

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

C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

E. N. Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

A. S. Weigend and N. A. Gershenfeld, editors. *Time Series Prediction: forecasting the future and understanding the past*. Addison-Wesley, 1994.

N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Physical review letters*, 45 (9):712–716, 1980.

T. Sauer. Time series prediction by using delay coordinate embedding. In *Time Series Prediction: forecasting the future and understanding the past*, 1994.

H. Jiang and H. He. State space reconstruction from noisy nonlinear time series: An autoencoderbasedapproach. In *International Joint Conference on Neural Networks (IJCNN)*, 2017.

B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings ofnonlinear dynamics. *Nature communications*, 9:1–10, 2018.

W. Gilpin. Deep reconstruction of strange attractors from time series. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

S. Ouala, D. Nguyen, L. Drumetz, B. Chapron, A. Pascual, F. Collard, L. Gaultier, and R. Fablet. Learning latent dynamics for partially-observed chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(10): 103121, 2020.

M. B. Kennel, R. Brown, and H. D. I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A*, 45:3403, 1992.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

R. Temam. *Infinite-Dimensional Dynamical Systems in Mechanics and Physics*. Springer Science & Business Media, 2012.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 2015.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12):3372–2408, 2010.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

S. L. Brunton, B W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz. Chaos as an intermittently forced linear system. *Nature Communications*, 8(19), 2017.

S. Wang, X. Yu, and P. Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. arXiv:2007.14527, 2021a.

S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv:2001.04536, 2021b.