Computer Physics Communications 277 (2022) 108382

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Forecasting of nonlinear dynamics based on symbolic invariance ☆

Zhao Chen [a], Yang Liu [b], Hao Sun [c,d,*]

[a] Department of Civil and Environmental Engineering, Northeastern University, 360 Huntington Avenue, Boston, 02115, MA, United States of America
[b] School of Engineering Sciences, University of the Chinese Academy of Sciences, 1 Yanqihu East Road, Huairou, Beijing, 100872, China
[c] Gaoling School of Artificial Intelligence, Renmin University of China, 59 Zhongguancun Street, Haidian, Beijing, 100872, China
[d] Beijing Key Laboratory of Big Data Management and Analysis Methods, 59 Zhongguancun Street, Beijing, 100872, China

ARTICLE INFO

Article history:
Received 11 November 2021
Received in revised form 22 March 2022
Accepted 21 April 2022
Available online 25 April 2022

Keywords:
Dynamical systems
Time series
Symbolic regression
Machine learning
Delay coordinate embedding

ABSTRACT

Forecasting unknown dynamics is of great interest across many physics-related disciplines. However, data-driven machine learning methods are bothered by the poor generalization issue. To this end, a forecasting model based on symbolic invariance (i.e., symbolic expressions/equations that represent intrinsic system mechanisms) is proposed. By training and pruning a symbolic neural network wrapped in a numerical integrator, we develop an invariant symbolic structure that represents the evolution function and thus can generalize well to unseen data. To counter noise effect, an algorithmic framework for probabilistic forecasting has also been developed by leveraging a non-parametric Bayesian inference method. Additionally, to account for univariate forecasting that is partially observed from a system with multiple state variables, we further leverage the delay coordinate embedding to find symbolic invariance of the partially observed system in a more self-contained embedding. The performance of the proposed framework has been demonstrated on both synthetic and real-world nonlinear dynamics and shown better generalization over popular deep learning models in short/medium forecasting horizons. Moreover, comparison with dictionary-based symbolic regression methods suggests better-behaved and more efficient optimization of the proposed framework when the function search space is enormous.

1. Introduction

The evolution of a nonlinear dynamic system is typically governed by one (or a set of) ordinary differential equation(s), e.g. $d\mathbf{x}/dt = f(\mathbf{x})$, where $\mathbf{x}$ denotes multiple state variables, t is time and f is a evolution function. If f is known analytically, the dynamic evolution for a given initial condition can be computed based on the numerical analysis. However, an explicit evolution function for an observed system may be unknown, and the forecasting of dynamic evolution is mainly based on measurement data from the past.

Fortunately, much progress has been made for time-series forecasting. There are many classical data-driven forecasting approaches built on the statistical patterns of measurement data. For example, the family of autoregressive–moving-average models (e.g. ARMA [1], ARIMA [2], ARMAX [3], NARMAX [4], etc.) identifies invariant (and parsimonious) relationships for inputs, outputs and noise. Another perspective for forecasting nonlinear dynamics is transforming the observed and usually low-dimensional nonlinear embedding to a high-dimensional embedding that is either linear or more tractable. Such methods include the support vector machine that uses the kernel trick [5,6] and the Koopman operator [7–9] that is embodied by linear or nonlinear dynamic mode decomposition. A comprehensive literature review for classical time-series models could be found in [10].

More recently, novel deep neural networks' strong nonlinear approximation capability has been leveraged to model time series data. Among many deep learning models, recurrent neural networks, such as the long short-term memory (LSTM) [11] and the gated recurrent unit (GRU) [12], can memorize temporal invariance and have been very popular for time series forecasting applications [13–15]. Sequence-to-sequence (seq2seq) models, in the structure of auto-regressive encoder-decoders, are good at exploiting the transformed embedding for system evolution in the original embedding [16]. This type of auto-regressive encoder-decoders is similar to the classical dimension transformation methods such as support vector regression and the Koopman operator. To further improve training efficiency when dealing

☆ The review of this paper was arranged by Prof. Weigel Martin.

* Corresponding author at: Gaoling School of Artificial Intelligence, Renmin University of China, 59 Zhongguancun Street, Haidian, Beijing, 100872, China.
E-mail address: haosun@ruc.edu.cn (H. Sun).

https://doi.org/10.1016/j.cpc.2022.108382
0010-4655/

with very long-term forecasting, the Transformer model [17] was introduced by replacing the recurrent components in both encoder and decoder with modified attention components. The Transformer model's variants have shown state-of-the-art performance for long-term time series forecasting [18,19]. Despite their remarkable accuracy based on the trained statistic distributions, these pure data-driven approaches still suffer from the issues of generalization in the forecasting of nonlinear dynamics, mainly when the test data is quite different from the training data.

To address the generalization issue of data-driven models, a novel line of forecasting approaches constrains data-based learning with inductive biases. A conventional family for such inductive bias is through regularization on model parameters, e.g., the Tikhonov regularization [20] and the dropout technique [21]. Lately, constraining data-driven models by differential equations has drawn lots of attention. Among them, the physics-informed neural networks (PINNs) [22] with their variants [23–26] have achieved remarkable long-term forecasting even for a new initial/boundary condition (I/BC) by absorbing the complete or dominant information of governing differential equations and I/BCs into a soft-constrained/hard-encoded network component.

However, the PINN family requires knowing the exact governing equation or at least the structure of the governing equation with some unknown coefficients. This level of prior knowledge can be impractical in many situations. To relax this restriction, a second deep neural network is introduced to the original PINN to approximate the unknown evolution function and substitute the role of governing equations in PDEs [27]. Similarly, [28] discretized the evolution of ODEs by the 4th-order Runge Kutta (RK4) method [29] and replaced the unknown evolution function with a fully-connected neural network. Interestingly, similar to the handling of the ARMA family, they explicitly model noise variables at every time step, leading to robust performance against noise. A trunk-branch neural network framework (DeepONet) [30] is also proposed to learn an arbitrary mapping from a space of input functions (i.e., excitations, parametric equation coefficients, parametric I/BCs, etc.) to a space of output functions (quantities of interest or QoIs) based on the universal approximation theorem for operator [31]. The DeepONet has been verified on various PDEs and ODEs, including linear and nonlinear explicit and implicit operators. These approaches [27,28,30] although achieve remarkable performance to unseen datasets in a variety of systems, they still require a large number of training data from a parametric distribution and the unseen data have to belong to the same parameterization as the training data to guarantee generalizability [32].

To reduce the measurement data quantity, more inductive biases have been introduced in the design of the computational framework as a trade-off. A family of sparse regression approaches [33–35] has been developed for equation discovery and system forecasting, where measurement data are sparsely projected to a customized dictionary of candidate functions to formulate the governing equation of the system. As long as the discovered governing equation is equal/close to the ground truth, their models can generalize well to all types of new I/BCs and excitations. However, its approximation of derivatives through finite difference is fragile against noise perturbation. Follow-up approaches address this fragility via improvement such as a reformulation into the variational form [36], uncertainty quantification [37], noise identification [38,39] and neural-network-denoising [40,41]. However, the core of these ideas is still based on dictionary regression, and using a large dictionary can be computationally expensive. As the number of QoIs or the power order of candidate functions increases, the number of function combinations grows in a polynomial scale (Please see the Method section for explanation). A great number of candidate combinations lead to a great number of trainable variables and consequently more computational burden for optimization. Another line of methods that align more with the purpose of equation discovery are anchored on genetic algorithms [42–44] that mutate or crossover parent symbolic expressions to produce the most accurate yet compact offspring expression after many generations. An outstanding trait of these methods is their remarkable expressiveness for almost arbitrary types of equations. However, this type of approaches is even less efficient than the dictionary methods for large-scale systems and high dimensional function space [33].

In this paper, we propose a hybrid forecasting model that finds the symbolic invariance (i.e., symbolic expressions/equations that represent intrinsic system mechanism) in unknown evolution functions of an ODE system, namely the symbolic forecasting model or SymFM. The benefits of SymFM are three-fold.

- The pivotal idea is to approximate the unknown evolution function in numerical integration by a symbolic neural network (SNN) [45–47]. The SNN is a fully-connected neural network whose activation functions and input features can be replaced by customized mathematical operators (e.g., multiplication, trigonometric operators, an exponential operator, etc.). In this way, the connections from inputs to outputs in SNN are symbolic expressions, which restrain the approximation capability of SNN but also limit the unknown evolution function to a customized function search space. An extra pruning for the SNN weights can remove misleading function combinations while further improving model predictability.

  Remark 1. While SNNs have been studied for learning dynamics models in previous work [47], their approach is different from ours, and new endeavors could be inspired from both approaches. For example, instead of entirely relying on SNNs to identify a discrete temporal evolution, a temporal integration scheme is hard-encoded in our proposed framework, and SNNs are responsible for learning the system's continuous flow map. An elegant dynamics encoder-decoder architecture in their work can disclose parametric properties from different datasets, such as varying resonant frequency. Such interesting features, together with the ResNet-like structure, the adaptive pruning, the non-parametric Bayesian inference and the delay coordinate embedding from this work can pave the way for more practical exploration in scientific computing.

- To deal with the noise effect on the approximation of derivative $d\mathbf{x}/dt$, we discretize the system evolution by an RK4 integrator (similar to [28,39]) and optionally convert the deterministic model to a non-parametric Bayesian model supported by the Stein variational gradient descent [48] (SVGD). Compared with [28,39] that explicitly identify noise values in each time step, our method depicts noise perturbation by distribution statistics such that its computational memory does not scale with the number of time steps. Furthermore, the SVGD is a non-parametric method not constrained by the parametric assumption for the family of posterior distributions. Therefore, it enjoys more expressive approximation.
- Considering that we may have observability for only a single state variable out of a multi-state-variable system, we turn to the delay coordinate embedding [49] to transform the partial observation to a more self-contained embedding in preprocessing. Unlike the original work [49] where a linear system with an intermittent force is employed for evolution in the new embedding, to achieve more flexibility, we let an SNN learn the possible nonlinear evolution of the new embedding.

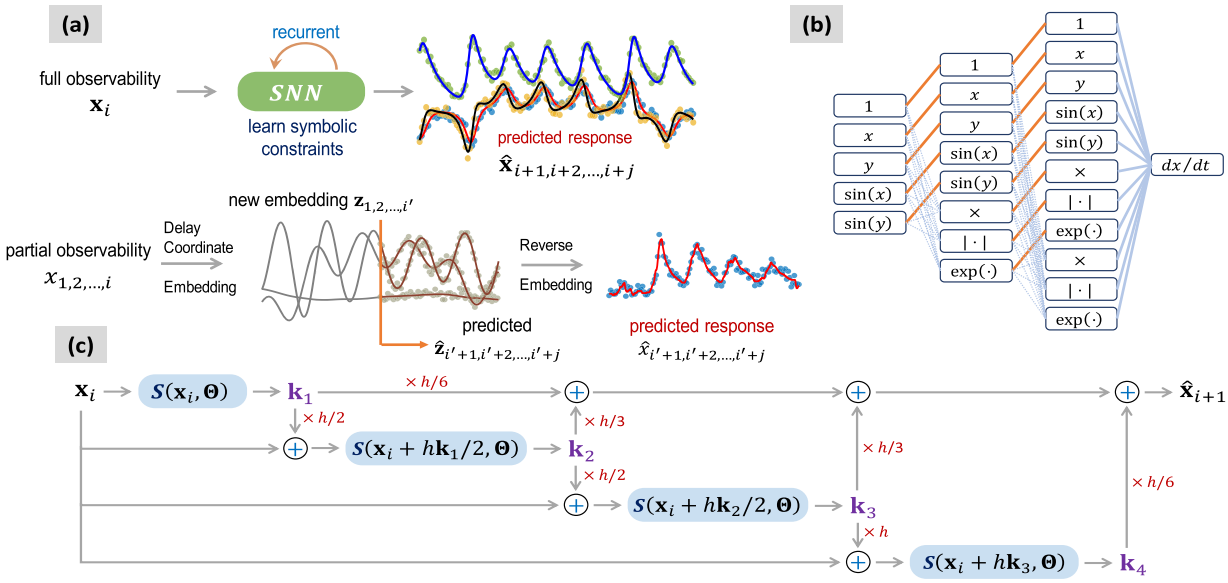

Fig. 1. A schematic of the proposed symbolic forecasting model (SymFM) in the Bayesian version. (a) When all state variables $\mathbf{x}_i$ are observable (i.e., full observability) at time step t_i, the symbolic neural network (SNN) [45–47] learns the unknown evolution function approximated by symbolic constraints. Together with Stein variational gradient descent (SVGD) [48], they enable iterative probabilistic forecasting $\hat{\mathbf{x}}_{i+1,\ldots,i+j}$. When only a single state variable $x_{1,\ldots,i}$ is observable for the past time steps $t_{1,\ldots,i}$ and the rest state variables are hidden, the measured variable is first transformed to a self-contained delay coordinate embedding $\mathbf{z}_{1,\ldots,i'}$ (note that $i' < i$) [49]. Then, the SNN is trained and forecasts $\mathbf{z}_{i'+1,\ldots,i'+j}$ in the new embedding. Lastly, we reverse the dimension transformation to obtain the prediction in the original embedding $x_{i'+1,\ldots,i'+j}$. (b) A generic SNN [46] inspired by ResNet [50]. Inputs and activation operators can be modified according to the domain knowledge. (c) The fourth-order Runge Kutta [29] time integration scheme.

The structure of the paper is as follows. In the second Section, we elaborate on the methodology of the SymFM. In the third Section, the proposed method is verified by two numerical and one experimental examples. In the last Section, the proposed method's strengths and weaknesses are discussed, guiding exploration for future work.

2. Method

This section introduces the proposed symbolic forecasting model (SymFM) (Fig. 1). First of all, we describe the concept of symbolic neural network (SNN) [45–47], how to wrap the SNN by the numerical integrator - the fourth-order Runge Kutta (RK4) [29], and an adaptive pruning strategy to find a parsimonious representation out of the redundant SNN. Secondly, we elaborate on the Bayesian framework braced by the Stein variational gradient descent (SVGD). Lastly, we integrate the delay coordinate embedding [49] in the preprocessing to handle the partial observability.

2.1. Backbone: symbolic neural network wrapped in a fourth-order Runge Kutta integrator

2.1.1. Explicit fourth-order Runge Kutta integrator

In numerical analysis, the explicit RK4 is a popular approach for predicting the evolution of ordinary differential equations (ODEs) by harnessing the evaluations of a known evolution function $d\mathbf{x}/dt = f(\mathbf{x})$ at intermediate time steps. $\mathbf{x} \in \mathbb{R}^{N_{dim} \times N_{ts}}$ denotes the measured state variables, and N_{dim} is the number of these observed state variables, e.g., $N_{dim} = 3$ for $\mathbf{x} = \{x, y, z\}$. N_{ts} is the number of time steps. As one of the widely used explicit numerical methods for solving differential equations, the local truncation error of RK4 is on the order of $\mathcal{O}(h^5)$, and the global accumulative error $\mathcal{O}(h^4)$, where h is the time step size.

Remark 2. Besides the explicit RK4, a familiar integrator like the forward Euler could also propel temporal evolution. However, the RK4 has better properties in terms of accuracy and stability at the cost of a higher computational cost. Other attractive substitutes are multiple-step integrators and implicit methods. Unlike single-step methods, multi-step methods strengthen memory and reduce the risk of gradient vanishing or explosion during the back-propagation. Implicit methods possess natural stability merit compared with the explicit methods while sacrificing the computational efficiency.

2.1.2. Symbolic neural network

However, in many cases, we do not know the evolution function except for measurements for the system. Assuming that we have measurements for every state variable of the system, an SNN can describe the mapping between the velocity $d\mathbf{x}/dt$ and state variables $\mathbf{x}$. The original SNN [45,47] is a fully-connected neural network (FCNN) with individual activation functions customized to mathematical operators/functions, like multiplication, trigonometric, absolute, and exponential functions. Besides activation functions, the mathematical operators can also appear in inputs (e.g., the sinusoidal functions in Fig. 1(b)). Additionally, we find directly passing the layer inputs to layer outputs can emphasize the effects of monomials and make simplified representation more obtainable. Therefore, we adopt a similar architecture to ResNet [50,46].

2.1.3. Comparison with dictionary methods

The densely-connected nature of SNN provides a larger function search space than the dictionary method [33,35,39] for the same amount of trainable variables. For example, if we want to form a search space that has up-to-fourth-order polynomials (e.g., $1, x, y, ..., x^2, xy, ..., x^3, x^2y, ..., x^4, x^3y$) from four atomic components $1, x, y, z$, the dictionary method will require 35 trainable coefficients according to the formula of combination with repetition [51]. Meanwhile, by using an SNN with inputs $1, x, y, z$, and two hidden layers that each has a multiplicative operator, the total number of trainable weights will be 24. If the search space is a fifth-order polynomial from five atomic components, the dictionary method will need 126 trainable variables, and the SNN will only need 44 trainable variables. Generally, if a function search space that has at most N_{poly}th-order polynomials from a collection of N_{atom} atomic components needs to be formed, the dictionary method will require $\binom{N_{atom}+N_{poly}-1}{N_{poly}}$ trainable variables and an SNN that only has one multiplicative operator in each hidden layer will need $N_{atom} + \mathcal{C}(\log_2 N_{poly})\left[2N_{atom} + \mathcal{C}(\log_2 N_{poly})\right]$ trainable variables where $\mathcal{C}(x)$ is the smallest integer greater than x. If the highest order of polynomials N_{poly} is fixed, the number of trainable variables is $\mathcal{O}(N_{atom}^{N_{poly}})$ for the dictionary method versus $\mathcal{O}(N_{atom})$ in the SNN. Similarly, if the number of atomic components N_{atom} is constant, the numbers of trainable variables for the dictionary and the SNN are $\mathcal{O}(N_{poly}^{N_{atom}-1})$ and $\mathcal{O}(\mathcal{C}^2(\log_2 N_{poly}))$, respectively. Therefore, as N_{atom} and N_{poly} increase, the number of trainable parameters in an SNN grows much slower than that in a dictionary. However, the gradient-based training required by SNNs is instinctively more expensive than the linear regression in dictionary methods, which is true for small-scale systems (such as the Lorenz system [33]) with a small number of function combinations. In contrast, the speed and the stability of classical dictionary methods deteriorate rapidly for high dimensional function search space, as shown in the real-world case study. Therefore, the SNN possesses more competitive edges in scalability than dictionary methods.

Besides the multiplicative operators, the SNN can also include many other differential functions (or operators) such as exponential, trigonometric and absolute operators. The SNN can create the same number of combinations using fewer model parameters than the dictionary method, which assigns all possible combinations a model parameter.

2.1.4. Forecasting and loss function

For an unknown dynamics, we express the discrete evolution function at a time step t_i as an SNN, i.e., $f(\mathbf{x}_i) = S(\mathbf{x}_i, \mathbf{\Theta})$, where $\mathbf{\Theta}$ is the collection of all SNN weights. We can write the RK4 propagation as (see Fig. 1 (c)):

$$\hat{\mathbf{x}}_{i+1} = \mathbf{x}_i + \frac{1}{6}h\left[\mathbf{k}_1(\mathbf{x}_i) + 2\mathbf{k}_2\left(\mathbf{x}_i + \frac{h\mathbf{k}_1}{2}\right) + 2\mathbf{k}_3\left(\mathbf{x}_i + \frac{h\mathbf{k}_2}{2}\right) + \mathbf{k}_4\left(\mathbf{x}_i + h\mathbf{k}_3\right)\right] \tag{1}$$

where the slopes $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ and $\mathbf{k}_4$ are read as $\mathbf{k}_1(\mathbf{x}_i) = S(\mathbf{x}_i, \mathbf{\Theta}), \mathbf{k}_2(\mathbf{x}_i + h\mathbf{k}_1/2) = S(\mathbf{x}_i + h\mathbf{k}_1/2, \mathbf{\Theta}),\ \mathbf{k}_3(\mathbf{x}_i + h\mathbf{k}_2/2) = S(\mathbf{x}_i + h\mathbf{k}_2/2, \mathbf{\Theta}), \mathbf{k}_4(\mathbf{x}_i + h\mathbf{k}_3) = S(\mathbf{x}_i + h\mathbf{k}_3, \mathbf{\Theta})$. Note that we use $\hat{\mathbf{x}}_{i+1}$ to denote the predicted system behavior at time step t_{i+1}. When the variable $\mathbf{x}$ is predicted from time step t_i to time step t_{i+1}, the intermediate slopes $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ and $\mathbf{k}_4$ are outputs from the SNN that has the same weights but different input samples. If we define a composite slope

$$\mathcal{R}(\hat{\mathbf{x}}_i, \mathbf{\Theta}) = \frac{1}{6}\left[\mathbf{k}_1(\mathbf{x}_i) + 2\mathbf{k}_2\left(\mathbf{x}_i + \frac{h\mathbf{k}_1}{2}\right) + 2\mathbf{k}_3\left(\mathbf{x}_i + \frac{h\mathbf{k}_2}{2}\right) + \mathbf{k}_4\left(\mathbf{x}_i + h\mathbf{k}_3\right)\right] \tag{2}$$

we can generalize Eq. (1) to a j-step-ahead forecasting equation:

$$\hat{\mathbf{x}}_{i+j} = \mathbf{x}_i + h\sum_{l=0}^{j-1}\mathcal{R}(\hat{\mathbf{x}}_{i+l}, \mathbf{\Theta}) \tag{3}$$

This forecast model has a temporally recurrent structure in the sense that: within one time step, the last three intermediate slopes $\mathbf{k}_2, \mathbf{k}_3, \mathbf{k}_4$ is a function of the first slope $\mathbf{k}_1$; across all time steps, the composite slope $\mathcal{R}(\hat{\mathbf{x}}_i, \mathbf{\Theta})$ depends on composite slopes from previous predicted time steps.

By computing the mean square error (MSE) between the predicted and the measured system trajectories and adding a sparsity penalty, we obtain the loss function in a deterministic setting:

$$\mathcal{J}(\Theta) = \mathbb{E}\left[(\hat{\mathbf{x}}(\mathbf{\Theta}) - \mathbf{x})^2\right] + \alpha\|\mathbf{\Theta}\|_1 \tag{4}$$

where $\mathbb{E}$ is the mean value of square errors of all measurement data; $\|\mathbf{\Theta}\|_1$ is the ℓ_1 norm of SNN weights; and α is its coefficient. The ℓ_1 penalty encourages trivial weights to have small values and paves the way for the following pruning treatment. In all case studies, $\alpha = 10^{-7}$ to prevent inhibition to the data fitting. Even though the sparsity penalty is not very strong in pretraining, the pruning-retraining alternation in the adaptive network pruning can identify and eliminate more minor weights in the later stage.

To circumvent gradient vanishing (or exploding) problems when j is large, we set $j = 1$ to maintain computational efficiency during training. In particular, we formulate the training measurements in the form of $\{\mathbf{x}_0; \mathbf{x}_1\}$, $\{\mathbf{x}_1; \mathbf{x}_2\}$, ..., $\{\mathbf{x}_i; \mathbf{x}_{i+1}\}$, where $\mathbf{x}_i$ before semicolon denotes the input and $\mathbf{x}_{i+1}$ behind semicolon is the output. From experiments, we found that the value of j has little effect on inference accuracy.

2.1.5. Adaptive network pruning

To improve the generalizability of SNN, we prune out trivial weights in SNN after training. The pruning principle is to balance the forecasting accuracy and the model complexity. In particular, minor weights whose absolute values are smaller than a threshold are nullified, and the remaining weights are retrained. The pruning-retraining cycle stops when the number of non-zero weights converges. The threshold is adaptively and iteratively updated based on a score function s that balances the goodness-of-fit on training data and the SNN complexity.

Algorithm 1 Adaptive Symbolic Neural Network Pruning.

Input: Trained SNN weights $\mathbf{\Theta}$, training datasets $\mathbf{x}^{tr}$, validation data $\mathbf{x}^{val}$, ℓ_0 coefficient η, initial pruning percentile $\delta_0 = 5\%$ and initial upscale factor for pruning percentile $\beta = 2$.
Output: Best threshold $th(\delta_{best})$
$\hat{\mathbf{\Theta}}_0 \leftarrow \mathbf{\Theta}\,(|\theta| < th(\delta_0)) = 0$ // initial pruning by $th(\delta_0)$
$s_{best} = \mathbb{E}\left[(\hat{\mathbf{x}}^{val}(\hat{\mathbf{\Theta}}_0) - \mathbf{x}^{val})^2\right] + \eta||\hat{\mathbf{\Theta}}_0||_0$ // baseline score
$\delta_{best} = \delta_0$ // baseline pruning percentile
$\delta_1 \leftarrow \delta_0$
for $m = 1, 2, ..., N_{adp}$ **do** // loop for new thresholds
  $\hat{\mathbf{\Theta}}_{m,0} \leftarrow \mathbf{\Theta}\,(|\theta| < th(\delta_m)) = 0$
  for $n = 1, 2, ..., N_{prune}$ **do** // loop for pruning-retraining
    $\hat{\mathbf{\Theta}}_{m,n} \leftarrow \hat{\mathbf{\Theta}}_{m,n-1}\,(|\theta| < th(\delta_m)) = 0$
    $\hat{\mathbf{\Theta}}_{m,n} \leftarrow \arg\min \mathbb{E}\left[(\hat{\mathbf{x}}^{tr}(\hat{\mathbf{\Theta}}_{m,n}) - \mathbf{x}^{tr})^2\right]$ // retrain non-zero weights
  end for
  $s_m = \mathbb{E}\left[(\hat{\mathbf{x}}^{val}(\hat{\mathbf{\Theta}}_{m,n}) - \mathbf{x}^{val})^2\right] + \eta||\hat{\mathbf{\Theta}}_{m,n}||_0$ // new score
  if $s_m \le s_{best}$ **then**
    $s_{best} \leftarrow s_m, \delta_{best} \leftarrow \delta_m, \delta_{m+1} \leftarrow \min(\beta\delta_{best}, 99\%)$ // increase threshold
  else
    $\delta_{m+1} \leftarrow 0.75\delta_m, \beta \leftarrow \max(1.2, \sqrt{\beta})$ // penalize threshold
  end if
end for

$$s = \mathbb{E}\left[(\hat{\mathbf{x}}(\mathbf{\Theta}(\delta)) - \mathbf{x})^2\right] + \eta||\mathbf{\Theta}(\delta)||_0 \tag{5}$$

where η is a hyper-parameter decided by the equilibrium between model predictability and simplicity, $||\mathbf{\Theta}(\delta)||_0$ is the ℓ_0 norm for SNN weights, and δ is the percentile of all SNN weights' absolute values. In the beginning, $\delta = 5\%$, meaning that 5% SNN weights will become zeros. If a new δ can achieve an equal or more miniature score than the current best score, we will increase δ. Otherwise, we will decrease δ. The score function s is similar to the Akaike information criterion [52]. A more specific workflow of the adaptive network pruning is shown in Algorithm 1.

2.2. Uncertainty quantification: non-parametric Bayesian inference

When measurement data are noisy, we develop a non-parametric Bayesian formulation to quantify the uncertainty propagation in forecasting by the Stein variational gradient descent (SVGD) [48].

2.2.1. Bayesian inference

We model the noise effect as a heteroscedastic Gaussian distribution $\mathbf{n} \sim \mathcal{N}(0, \mathbf{\Omega}^{-1})$ additive to the deterministic model

$$\hat{\mathbf{x}}_{i+j} \approx \mathbf{x}_i + h\sum_{l=0}^{j-1} \mathcal{R}(\hat{\mathbf{x}}_{i+l}, \mathbf{\Theta}) + \mathbf{n} \tag{6}$$

Elements in the diagonal noise precision matrix $\mathbf{\Omega}$ vary from one state variable to another, and each of them presumably follows a Gamma distribution $\mathbf{\Gamma}(a_1, b_1)$, where a_1 (shape) and b_1 (rate) are determined heuristically according to estimated noise statistics. For example, estimated noise, decoupled from the actual measurement by data smoothing, has an approximate variance of 0.01. Then, $a_1 = 1$ and $b_1 = 0.01$ so that the mean of the noise precision prior is 0.01^{-1}. However, a more accurate noise representation needs to consider the noise's influence inside $\mathcal{R}(\hat{\mathbf{x}}_{i+l}, \mathbf{\Theta})$ since $\hat{\mathbf{x}}_{i+l}$ is also subjected to noise. Here we alleviate this drawback by setting $j = 1$ during training.

To quantify the posterior distribution of the SNN weights $\mathbf{\Theta}$ as well as the noise precision $\mathbf{\Omega}$ from measurement data $\mathbf{x}$, i.e., $p(\mathbf{\Theta}, \mathbf{\Omega}|\mathbf{x})$, we minimize its Kullback-Leibler ($\mathbb{KL}$) divergence [53] with a proxy distribution $q(\mathbf{\Theta}, \mathbf{\Omega})$, given by

$$\mathbb{KL}\left[q(\mathbf{\Theta}, \mathbf{\Omega})||p(\mathbf{\Theta}, \mathbf{\Omega}|\mathbf{x})\right] = \mathbb{E}_q \log q - \mathbb{E}_q \log \overline{p} + \log p(\mathbf{x}) \tag{7}$$

Here, q is a shorthand for $q(\mathbf{\Theta}, \mathbf{\Omega})$; $\mathbb{E}_q \log q$ denotes the expectation of $\log q$ with respect to q; $\mathbb{E}_q \log \overline{p}$ denotes the expectation of $\log \overline{p}$ to q; and $\overline{p} = \overline{p}(\mathbf{\Theta}, \mathbf{\Omega}|\mathbf{x}) = p(\mathbf{x}|\mathbf{\Theta}, \mathbf{\Omega})p(\mathbf{\Theta})p(\mathbf{\Omega})$ is the unnormalized posterior based on hierarchical Bayesian inference. In the unnormalized posterior $\overline{p}$, $p(\mathbf{x}|\mathbf{\Theta}, \mathbf{\Omega}) = \mathcal{N}(\mathbf{x}, \mathbf{\Omega}^{-1})$ is a Gaussian likelihood; $p(\mathbf{\Theta}) = \mathcal{L}(\mathbf{\Theta}|a_2, b_2)$ is a sparsity-promoted Laplace prior for $\mathbf{\Theta}$, where heuristically $a_2 = 0$ (location) and $b_2 = 2$ (scale) to gently promote sparsity; and $p(\mathbf{\Omega})$ is the product of Gamma priors $\mathbf{\Gamma}(a_1, b_1)$ of the noise precision for each state variable as described in the last paragraph. Note that the SNN model prior $p(\mathbf{\Theta})$ and the noise prior $p(\mathbf{\Omega})$ are independent; thus, we use $p(\mathbf{\Theta})$ instead of $p(\mathbf{\Theta}|\mathbf{\Omega})$.

2.2.2. Stein variational gradient descent

The posterior $p(\mathbf{\Theta}, \mathbf{\Omega}|\mathbf{x})$ does not have a closed-form due to the complicated relationship between $\hat{\mathbf{x}}$ and $\mathbf{\Theta}$ in the Gaussian likelihood. Usually, the proxy distribution $q(\mathbf{\Theta}, \mathbf{\Omega})$ is assumed to follow some parametrization, such as an exponential family. Then its hyper-parameters will be optimized to fit the measurement data as much as possible. However, this type of restriction is not ideal for our case since the relationship between $\hat{\mathbf{x}}$ and $\mathbf{\Theta}$ is implicit and the SNN is highly customizable. Alternatively, we turn to a non-parametric method, so-called Stein variational gradient descent (SVGD) [48], developed on the theory of Stein operator. In SVGD, iterative transformation (or gradient descent) from an uninformative trial distribution to the target distribution can be obtained in a regularized space [48]. By assuming $\mathbf{\Lambda} = \{\mathbf{\Theta}, \mathbf{\Omega}\}$, the transformation is stated as follows

$$\mathbf{\Lambda}_{n_e+1} \leftarrow \mathbf{\Lambda}_{n_e} + \epsilon\,\Psi(\mathbf{\Lambda}_{n_e}) \tag{8}$$

Algorithm 2 Probabilistic Symbolic Forecasting Model.

Input: Measured trajectories $\mathbf{x}$, hyper-parameter b_1 for noise prior $p(\mathbf{\Omega})$, the learning rate ϵ, time step size h.
Output: N_p sets of SVGD particles $\mathbf{\Lambda}$ approximating the posterior $p(\mathbf{\Lambda}|\mathbf{x})$
Sample N_p sets of SVGD particles $\mathbf{\Lambda}_1 = \{\mathbf{\Theta}_1, \mathbf{\Omega}_1\}$ from priors $p(\mathbf{\Theta})$ and $p(\mathbf{\Omega})$;
for $n_e = 1, 2, ..., N_e$ **do** // loop for SVGD training epochs
  for $n_p = 1, 2, ..., N_p$ **do** // loop for SVGD particle sets
    Compute the log of unnormalized posterior $\log \overline{p}(\boldsymbol{\lambda}_{n_e}^{n_p})$ using a set of particles;
    Compute its gradient with respect to this set of particle via automatic differentiation: $\nabla_{\boldsymbol{\lambda}_{n_e}^{n_p}} \log \overline{p}(\boldsymbol{\lambda}_{n_e}^{n_p}|\mathbf{x})$
  end for
  Compute the transformation function $\Psi(\mathbf{\Lambda}_{n_e})$
  for $n_p = 1, 2, ..., N_p$ **do** // loop for SVGD particle sets
    Update particles: $\boldsymbol{\lambda}_{n_e+1}^{n_p} \leftarrow \boldsymbol{\lambda}_{n_e}^{n_p} + \epsilon \Psi(\boldsymbol{\lambda}_{n_e}^{n_p})$
  end for
end for

where the subscript n_e denotes the n_eth epoch for the transformation; ϵ is the learning rate; and Ψ is the transformation function or the steepest gradient descent in a unit ball of a reproducing kernel Hilbert space, namely,

$$\Psi(\mathbf{\Lambda}) = \mathbb{E}_q \left[K(\mathbf{\Lambda}, \mathbf{\Lambda}') \nabla_{\mathbf{\Lambda}} \log \overline{p}(\mathbf{\Lambda}|\mathbf{x}) + \nabla_{\mathbf{\Lambda}} K(\mathbf{\Lambda}, \mathbf{\Lambda}') \right] \tag{9}$$

where $\nabla_{\mathbf{\Lambda}}$ is gradient with respect to $\mathbf{\Lambda}$; and $K(\mathbf{\Lambda}, \mathbf{\Lambda}')$ is the radial basis function kernel [54]. The transformation function Ψ is the analytical solution of the kernelized Stein discrepancy between the real posterior p and the proxy q in a unit ball of a reproducing kernel Hilbert space [48]. In other words, Eq. (9) gives the steepest descent for the $\mathbb{KL}$ divergence in a regularized space. In practice, the proxy $q(\mathbf{\Lambda})$ is discretized with N_p sets of particles, i.e., N_p copies of $\boldsymbol{\lambda}$ ($\boldsymbol{\lambda} \subseteq \mathbf{\Lambda}$) sampled from their priors or other uninformative distributions. Then, these SVGD particles gradually cover the real posterior by adaptively adjusting their distance. The first term in the update function $\Psi(\mathbf{\Lambda})$ pulls particles to the high probability area of the unnormalized posterior $\overline{p}(\mathbf{\Lambda}|\mathbf{x})$. In contrast, the second term prohibits all particles from falling into one local mode. Compared with other Bayesian methods, SVGD does not need to specify the parametric proxy distribution or require a considerable amount of samples. The algorithm for SVGD is shown in Algorithm 2.

2.2.3. Probabilistic forecasting

Besides computing the posterior for model parameters, the uncertainty for evolution can also be quantified. Given training data $\mathbf{x}$, the mean for propagating a new initial condition $\mathbf{x}_0^\star$ can be determined by the law of total expectation [55]:

$$\begin{aligned} \mathbb{E}[\hat{\mathbf{x}}^\star|\mathbf{x}, \mathbf{x}_0^\star] &= \mathbb{E}_q \left[\mathbb{E}[\hat{\mathbf{x}}^\star|\mathbf{x}_0^\star, \mathbf{\Lambda}] \right] \\ &= \mathbb{E}_q \left[\mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}) \right] \\ &\approx \frac{1}{N_p} \sum_{n_p=1}^{N_p} \mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}^{n_p}) \end{aligned} \tag{10}$$

where $\mathcal{F}$ represents the deterministic forecasting model as shown in Eq. (3).

Based on the law of total variance, the predictive covariance is as follows [55]:

$$\begin{aligned} & Cov(\hat{\mathbf{x}}^\star|\mathbf{x}, \mathbf{x}_0^\star) \\ &= \mathbb{E}_q \left[Cov(\hat{\mathbf{x}}^\star|\mathbf{x}_0^\star, \mathbf{\Lambda}) \right] + Cov_q \left(\mathbb{E}[\hat{\mathbf{x}}^\star|\mathbf{x}_0^\star, \mathbf{\Lambda}] \right) \\ &= \mathbb{E}_q \left[\mathbf{\Omega}^{-1} \right] + Cov_q \left(\mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}) \right) \\ &= \mathbb{E}_q \left[\mathbf{\Omega}^{-1} + \mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}) \mathcal{F}^\intercal(\mathbf{x}_0^\star, \mathbf{\Theta}) \right] - \mathbb{E}_q \left[\mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}) \right] \mathbb{E}_q^\intercal \left[\mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}) \right] \end{aligned} \tag{11}$$

where $\intercal$ denotes matrix transposition. The predictive variance is further obtained from the main diagonal of the covariance matrix

$$\begin{aligned} & \operatorname{diag} \left\{ Cov(\hat{\mathbf{x}}^\star|\mathbf{x}, \mathbf{x}_0^\star) \right\} \\ &\approx \frac{1}{N_p} \sum_{n_p=1}^{N_p} \left(\left(\mathbf{\Omega}^{n_p} \right)^{-1} + \mathcal{F}^2(\mathbf{x}_0^\star, \mathbf{\Theta}^{n_p}) \right) - \left(\frac{1}{N_p} \sum_{n_p=1}^{N_p} \mathcal{F}(\mathbf{x}_0^\star, \mathbf{\Theta}^{n_p}) \right)^2 \end{aligned} \tag{12}$$

Remark 3. The pruning strategy in Algorithm 1 is developed for the deterministic SNNs. However, it can also be extended for the Bayesian framework with two modifications. One modification is that a new weight pruning method needs to be based on SNN weights' distributions. Inspired by the sparse Bayesian learning [56], means of weights' posteriors or variances (need to be trainable in this case) of weights' priors can be a good indicator for redundancy. If the mean of a weight's posterior is too small, the weight can be regarded as possibly trivial; If the variance of a weight's prior is too small, it means this weight's prior is heavily concentrated at zero, and the weight is possibly redundant. The threshold for means or variances can be determined in an adaptive manner similar to Algorithm 1. The other modification is that the first component (an indicator for accuracy) of the score function can now base on the negative logarithmic likelihood of the validation data with predicted mean and variance from Eqs. (10) and (12).

2.3. Univariate observation: delay coordinate embedding

When a system is governed by multiple state variables but only one of them has measurement data, we can no longer design an SNN correlating this single state variable to its temporal derivative. However, we can use dimension transformation to find a new self-contained embedding from the measurement data.

The delay coordinate embedding [57] is a popular family of approaches to reveal the full state space from the history of observations. Assume we have measurement data for a single state variable $x \in \mathbb{R}^{1\times N_{ts}}$ with the time step size h. We can find a new multivariate embedding $\mathbf{y}$ by formulating a collection of delay coordinates, such as $\mathbf{y}_i = \{x_i, x_{i+h}, x_{i+2h}, ..., x_{i+(N_{dc}-1)h}\}$ at time step t_i. When the dimension of delay coordinates is N_{dc}, the total number of time steps for the embedding $\mathbf{y}$ will be $N_{ts} - N_{dc} + 1$, which is smaller than that of the original measurement data x. The length of delay between two neighboring coordinates does not have to be one-time step size h, and multiple times of h is also possible, i.e., $\mathbf{y}_i = \{x_i, x_{i+3h}, x_{i+6h}, ..., \}$. According to Takens' theorem [58], the new embedding is diffeomorphic to the original system's dimension (whose state dimension is N_{dim}) if $N_{dc} > 2N_{dim} + 1$ with some mild conditions.

One way to determine the new embedding's dimension is to conduct a singular value decomposition (SVD) [59] on a Hankel matrix $\mathbf{H}$ and only maintain significant singular values for the new embedding. In particular, the Hankel matrix is read as

$$\begin{aligned}\mathbf{H} &= \begin{bmatrix} x_i & x_{i+h} & \cdots & x_{i+(N_{ts}-N_{dc})h} \\ x_{i+h} & x_{i+2h} & \cdots & x_{i+(N_{ts}-N_{dc}+1)h} \\ \vdots & \vdots & \vdots & \vdots \\ x_{i+(N_{dc}-1)h} & x_{i+N_{dc}h} & \cdots & x_{i+(N_{ts}-1)h} \end{bmatrix} \\ &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{T}} \end{aligned} \tag{13}$$

where $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ are the left singular matrix, the diagonal singular-value matrix and the right singular matrix. Then, we can line up the significance of singular values and truncate trivial singular values at the rank r based on thresholding principles (such as [60]).

The Hankel matrix $\mathbf{H}$ has an interesting connection to the Koopman operator, as explored in the Hankel alternative view of Koopman [49]. Suppose the original system is ergodic [61]. In that case, the left singular vectors in $\mathbf{U}$ can converge to the Koopman eigenmodes. The right singular vectors $\mathbf{z}$ in $\mathbf{V}$ are corresponding time-variant coordinates of these eigenmodes. For chaotic systems, vector components $\mathbf{z}$ from the right singular matrix $\mathbf{V}$ can be formulated into a linear system with intermittent force [49]

$$\frac{d\mathbf{z}}{dt} = \mathbf{A}\mathbf{z} + \mathbf{B}z_r \tag{14}$$

where $\mathbf{A}$ and $\mathbf{B}$ are constant matrices, and z_r is the forcing term also from $\mathbf{V}$. However, in practice, the truncation of SVD and the fact that not all systems are ergodic introduce modeling errors such that the system of $\mathbf{z}$ may not be a forced system [62]. To account for these errors and find an adaptive architecture for $\mathbf{z}$ from systems with different nonlinearities, we use the flexible SNN to simulate the evolution of $\mathbf{z}$ in Eq. (14).

3. Experiments

In this section, we investigate the efficacy of the proposed method in two numerical systems and one real-world system. First of all, we show that an SNN can improve forecasting generalizability compared with baseline deep learning models. Second, we explore the robustness of the model's forecasting ability when the SNN is redundant and show the effect of pruning. Third, we present the probabilistic forecasting ability of the proposed method for noisy data. Fourth, we demonstrate how the delay coordinate embedding augments the proposed method to deal with univariate observation for a system with multiple state variables and the advantages of the proposed framework over popular dictionary methods [33,39]. All tests were computed on an Nvidia V100 GPU.

3.1. System description

3.1.1. Synthetic system: chaotic Lorenz

The Lorenz system [63] is a three-dimensional chaotic dynamics whose attractor resembles a butterfly shape under certain coefficients. The chaotic governing functions for our case are read as

$$\begin{aligned}\frac{dx}{dt} &= 10(y - x) \\ \frac{dy}{dt} &= x(28 - z) - y \\ \frac{dz}{dt} &= xy - \frac{8}{3}z\end{aligned}$$

For training and validation data, we generated 15 trajectories using the `ode45` function in MATLAB with initial conditions (ICs) randomly sampled from $\mathbf{x}_0 \sim U(-18, 2), \mathbf{y}_0 \sim U(-3, 17), \mathbf{z}_0 \sim U(17, 37)$. All trajectories lasted for 15 seconds with a sampling rate of 1000 Hz. 10 of them were allocated to training data and the rest to validation data. For test data, another 5 trajectories were generated from different distributions of ICs $\mathbf{x}_0 \sim U(-50, 50), \mathbf{y}_0 \sim U(-50, 50), \mathbf{z}_0 \sim U(-50, 50)$ with the same duration and sample rate. Visual comparison for the training/validation vs. the test data is illustrated in Fig. 2(a). It is observed that the test data have longer tails in their distribution than the training/validation data.

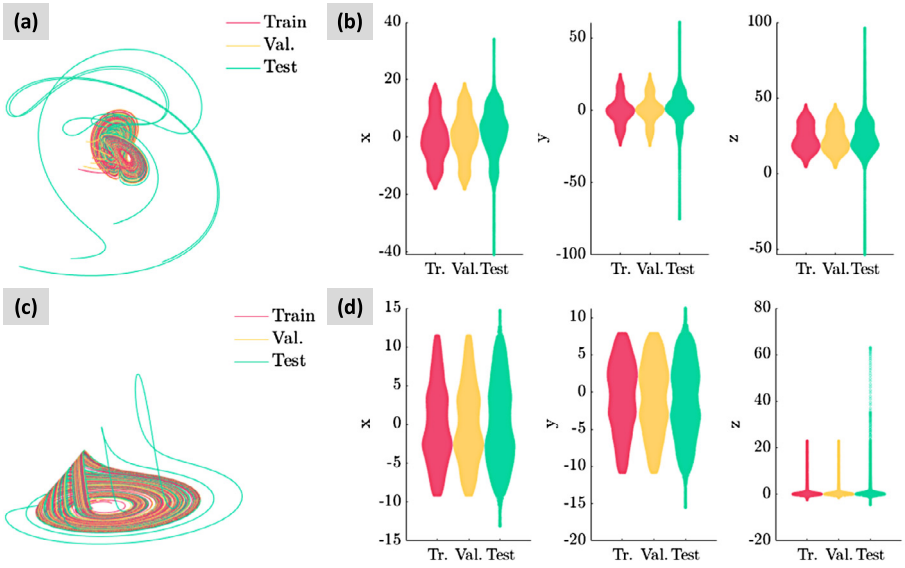

Fig. 2. Comparison of training, validation, and test data generated by different distributions of initial conditions (ICs). Test data have more wide-ranging ICs, leading to longer tails than other data in their distributions. (a) 3D trajectories of all data for the Lorenz system. (b) Distributions of all data for the Lorenz system. (c) 3D trajectories of all data for the Rössler system. (d) Distributions of all data for the Rössler system. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

3.1.2. Synthetic system: chaotic Rössler

The Rössler system [64] is another chaotic system whose trajectory starts from a fixed point in the $x-y$ plane and evolves in outgoing orbits until later being attracted by another fix point and rising high in the z-direction. Its governing functions are given by

$$\frac{dx}{dt} = -y - z$$
$$\frac{dy}{dt} = x + 0.2y$$
$$\frac{dz}{dt} = 0.2 + z(x - 5.7)$$

15 trajectories were generated synthetically using the `ode45` in MATLAB for training and validation data, whose ICs were randomly sampled from $\mathbf{x}_0 \sim U(-1.5, 3.5), \mathbf{y}_0 \sim U(-1.5, 3.5), \mathbf{z}_0 \sim U(-2.5, 2.5)$. Each trajectory lasted 80 seconds with a sampling rate of 100 Hz. 10 trajectories were training data, and the rest were validation data. Another 5 trajectories were generated from a broader range of ICs $\mathbf{x}_0 \sim U(-10, 10), \mathbf{y}_0 \sim U(-10, 10), \mathbf{z}_0 \sim U(-10, 10)$ to serve as the test data. A visual comparison between the training/validation and test data is illustrated in Fig. 2(b), where the test data had longer tails.

3.1.3. Real-world system: electrical transformer

The electrical transformer (ET) is a vital component in the electric transmission network. The working condition of an electrical transformer can be indicated by oil temperature, daily power load, weekly power load, etc. Assuming that only oil temperature data [19] was available, we intended to foresee its future trajectory given a history of data collected every 15 minutes for 2 years. Standardization was carried out by setting its mean to zero and variance to one. Then, a moving average with a sliding window of 50 data points was applied to the standardized data. This lengthy trajectory follows a phase-shifted sinusoidal trend governed by seasonal patterns and many higher frequency oscillations due to short-term and long-term variations (Fig. 3). We split the data continuously for training, validation, and test by 60%, 20%, and 20%. The delay coordinate embedding for training data was obtained following the truncated SVD mentioned above. The delay coordinate embedding for validation data and the initial condition of test data were computed based on the left and the diagonal singular matrices from training data.

3.2. Deterministic forecasting with full observability: comparison with deep learning models

First of all, we compared the deterministic SymFM with two deep learning baselines, the long short-term memory (LSTM) model [11] and the Informer model [19], to test their generalizability in the two synthetic systems, i.e., Lorenz and Rössler.

3.2.1. Model setups

Deterministic SymFM. The deterministic SymFM or DSymFM in this comparison consisted of three SNNs since trajectories from Lorenz and Rössler systems were three-dimensional (i.e., x, y, z). All SNNs had four input features $1, x, y, z$, two hidden layers that each had one extra multiplicative operator and one output feature $\frac{dx}{dt}$, $\frac{dy}{dt}$ or $\frac{dz}{dt}$. We would analyze more redundant SNN designs later. The SNN weights were initialized from a normal distribution $N(0, 0.01^2)$ lay a gentle foundation for sparsity.

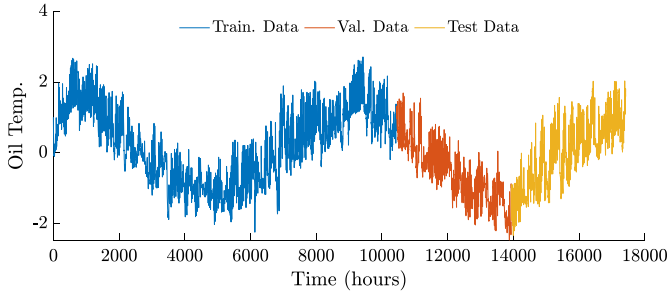

Fig. 3. Standardized and smoothed oil temperature data from the electrical transformer system, where training, validation and test data were continuously split to 60%, 20% and 20% portions.

Table 1
Training efforts for two synthetic examples in deterministic forecasting.

Models	Lorenz		Rössler	
	Learning rate	Epochs	Learning rate	Epochs
DSymFM	10^{-3}(50%)[a]	10k	5×10^{-2}(50%)[a]	0.4k
LSTM	10^{-3}(10%)[b]	1k(2048)[c]	10^{-3}(10%)[b]	8k(8192)[c]
Informer	10^{-4}(10%)[d]	12(32)[c]	10^{-4}(10%)[d]	12(32)[c]

[a] The learning rate decayed to the percentage in the parentheses when the validation error stopped decreasing for 10 epochs.
[b] The learning rate decayed to the percentage in the parentheses in the second half of the training.
[c] In every epoch, the optimizer conducted mini-batch training where the mini-batch size is indicated in the parentheses.
[d] The learning rate decayed to the percentage in the parentheses every two epochs.

Table 2
RMSEs for two synthetic examples in deterministic forecasting (Models with the best test score in bold).

Steps	Lorenz			Rössler		
	100	200	300	100	200	300
DSymFM	0.007(train)	0.011(train)	0.015(train)	0.024(train)	0.037(train)	0.0571(train)
	0.007(val.)	0.011(val.)	0.015(val.)	0.025(val.)	0.036(val.)	0.0495(val.)
	0.245(test)	**0.314(test)**	**0.458(test)**	**0.092(test)**	**0.141(test)**	**0.180(test)**
LSTM	0.643(train)	0.743(train)	1.118(train)	0.019(train)	0.018(train)	0.017(train)
	0.675(val.)	0.781(val.)	1.209(val.)	0.019(val.)	0.018(val.)	0.017(val.)
	2.209(test)	1.223(test)	1.446(test)	0.566(test)	0.449(test)	0.250(test)
Informer	0.100(train)	0.185(train)	0.241(train)	0.149(train)	0.204(train)	0.229(train)
	0.190(val.)	0.333(val.)	0.646(val.)	0.191(val.)	0.251(val.)	0.252(val.)
	1.815(test)	1.246(test)	1.570(test)	1.107(test)	1.285(test)	0.757(test)

LSTM. The LSTM model [11] is a classical recurrent neural network that leverages efficient long-term memory channels to bypass gradient vanishing/explosion problems. The LSTM consisted of 3 stacked LSTM layers with 32 nodes and a linear output layer. All LSTM layers had a 20% dropout rate to promote generalizability. Weights and biases were initialized from a uniform distribution $U(-\sqrt{n^l_{lstm}}, \sqrt{n^l_{lstm}})$, where n^l_{lstm} is the number of features in each LSTM layer. The LSTM predicts the next j time steps given the previous $2j$ time steps during training.

Informer. As a variant of the Transformer model [17], the Informer model developed novel ways to replace the computationally expensive full-attention mechanism with efficient ProbSparse attention and became one of the state-of-the-art models for time-series forecasting. Following the original setting [19], the Informer model had 512 dimensions of latent embeddings, 8 heads, 3 stacked encoder layers, 2 stacked decoder layers, 1024 dimensions of function, a dropout rate of 5%, and a batch size of 32. Weights and biases were initialized from a uniform distribution $U(-\sqrt{n^l_{informer}}, \sqrt{n^l_{informer}})$, where $n^l_{informer}$ is related to the number of input features and the kernel size. Similar to the LSTM, the Informer predicts the next j time steps given the previous $2j$ time steps.

To train all three models, we used Adam optimizer [65]. The learning rates and the training epochs of the three models are listed in Table 1. While the network architectures of the LSTM and the Informer and the learning efforts (learning rates and epochs) were determined by trial and error, they were sufficient for loss convergence and satisfying training and validation accuracy (Table 2).

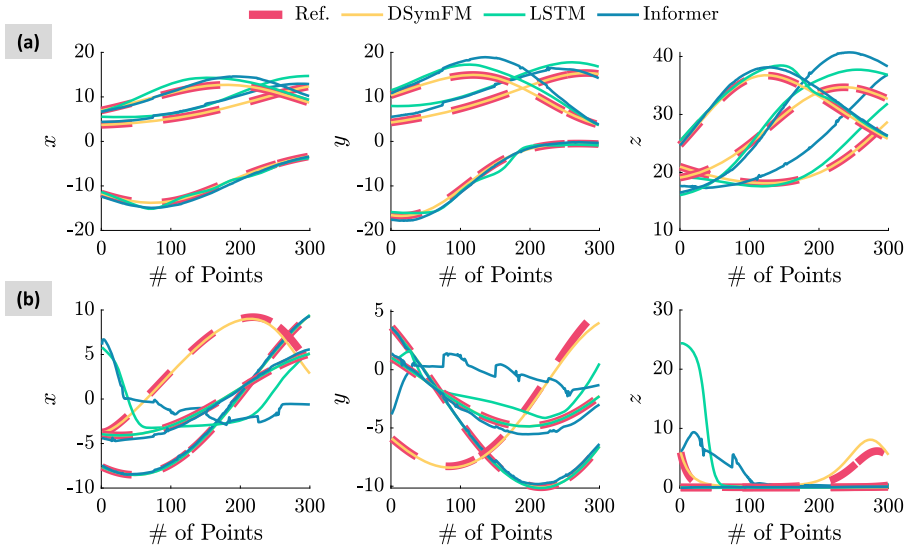

Fig. 4. Visualization of the benchmark comparison for two synthetic systems. (a) The 300-step-ahead forecasting of Lorenz system.(b) The 300-step-ahead forecasting of Rössler system.

3.2.2. Results

We compared the performance of three models under 100/200/300 forecasting steps. The root mean square errors (RMSEs) for training, validation, and test datasets for all the models are shown in Table 2. Some forecasting segments by the DSymFM are visualized in Fig. 4.

The DSymFM had the lowest RMSEs in the two synthetic systems. Despite remarkable accuracy for training and validation datasets in the Lorenz and the Rössler systems (considering the magnitudes of trajectories), the test errors of LSTM and the Informer model were much significant. The better performance of DSymFm was rooted in the fact that SNNs had learned the governing equations of the dynamic systems. Therefore, it could better handle unseen ICs. As for LSTM and Informer, it is found that their errors mainly came from two aspects: out-of-distribution ICs and error propagation. Firstly, the ICs in the test data were quite atypical of those in the training/validation data; thus, both deep learning models failed to make accurate predictions. Secondly, error accumulation in LSTM and Informer models was also more significant than DSymFM.

3.3. Effects of redundant SNN design and pruning

In the previous results, the SNNs we used intrinsically represented the temporal derivative (i.e., x_t) in a multivariate power series of x, y, z. Considering that governing equations of lots of chaotic systems could be represented by polynomials [38], it was reasonable to have such prior knowledge when designing the SNNs.

However, in the case that less prior knowledge was known, we designed two different types of SNNs for the Lorenz system: The second type of SNN had extra trigonometric functions $(1, x, y, z, \sin x, \sin y, \sin z)$ as inputs, two hidden layers that each had a multiplicative operator, and a temporal derivative x_t, y_t or z_t); The third type had extra trigonometric, exponential and absolute functions $(1, x, y, z, \sin x, \sin y, \sin z)$ as inputs, two hidden layers that each had a multiplication operator, an exponential operator and an absolute operator, and a temporal derivative x_t, y_t or z_t).

We tested the performance of all three types (i.e., one polynomial type and two redundant types) of SNNs for 200-step-ahead forecasting of the synthetic Lorenz system. Besides the same training efforts as that of DSymFM in Table 1, we further pruned the trained SNNs to improve the forecasting accuracy. As for hyper-parameters N_{adp} (the number of new thresholds to try) and N_{prune} (the times of alternating between pruning and retraining) in the Algorithm 1, it was found that $N_{adp} = 25$ and $N_{prune} = 10$ were enough for the convergence of loss and the stabilization of the number of non-zero weights. The remaining critical hyper-parameter was the ℓ_0 coefficient η, which balanced the pruned model's accuracy and parsimony. Values of η from a geometric sequence (e.g., from 10^{-15} to 10^{-5} with a ratio of 10) were tried. The optimal values of η, as shown in Fig. 5 were those achieving the fewest non-zero SNN weights while not encountering a significant increase in validation loss.

After pruning, the RMSE of the first type (SNNs with polynomial functions) was 5.65×10^{-4} for the unseen test data, the RMSE of the second type (SNNs with polynomial and trigonometric functions) was 8.14×10^{-5}, and the RMSE of the third type (SNN with polynomial, trigonometric, exponential and absolute functions) was 2.511. Compared with the results in Table 2, both the first and the second types had remarkable improvement after minor weights in SNN were pruned. However, the test RMSE of the third SNN was higher than that of either the LSTM or the Informer. By inspecting the three worst trajectory samples, the dominant divergence came from the beginnings of the trajectories, where ICs were very different from those of the training or validation data. As for the rest of trajectories, the third SNN predicted much better since these later portions were much like training and validation data.

We also analyzed the pruned SNNs by reformulating them into governing equations. The parsimonious governing equations for the first type almost agreed exactly with the ground truth, read as $dx/dt = -9.992x + 9.755y, dy/dt = -0.897xz + 25.588x - 0.894y$ and $dz/dt = 0.895xy - 2.489z$. The identified equations from the second type was a bit redundant as $dx/dt = 4 \times 10^{-6}xy - 10x + 10y, dy/dt = -xz + 28x + 3 \times 10^{-6}yz - y$ and $dz/dt = xy - 2.67z$. For the third type, the first equation was read as $dx/dt = -10.000x + 10.000y$ and the third equation was $dz/dt = xy - 2.67|z|$. Considering z in both training and validation data only had positive values (Fig. 2(b)), the confusion between $|z|$ and z was reasonable. However, the biggest problem was that the second identified equation had hundreds of

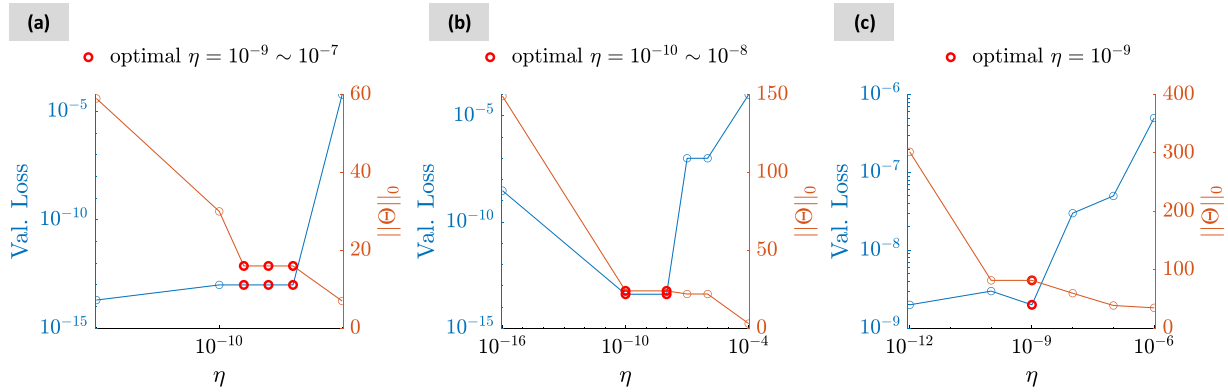

Fig. 5. The selection of ℓ_0 coefficient η (a critical hyper-parameter for pruning) of three symbolic neural network (SNN) types. (a) For the first type (SNNs with polynomial functions), we found $\eta = 10^{-9} \sim 10^{-7}$ was the optimal range. (b) For the second type (SNNs with polynomial and trigonometric functions), when the optimal range was $\eta = 10^{-10} \sim 10^{-8}$, the validation loss could be about the same as the first SNN. (c) For the third type (SNN with polynomial, trigonometric, exponential and absolute functions), the optimal choice was $\eta = 10^{-9}$. Either the validation loss or the model sparsity could reach a level as satisfying as the previous two cases.

minor terms. One reason was that excessive function options make discovery harder - training was more susceptible to local minima and may over-explain training and validation data.

Observations from this set of redundancy experiments are that

- Mild functional redundancy in SNNs is acceptable for generalized forecasting. After pruning, the first and second SNNs found compact and intrinsic symbolic invariance. It is conjectured that the inclusion of multiplication, trigonometric functions, or other functions that can construct series expansion should still endow a pruned SNN satisfying generalizability. However, it warrants further validation in the future.
- Random or excessive functional redundancy poses challenges for generalizable models. The third SNN found cumbersome symbolic equations that only reflected training/validation data. Its high test RMSE was also an indicator of an over-complicated SNN design. Such models more or less imitated a regular multiple layer perceptron, which could perform well within trained distributions.
- Function/Operator designs, optimization methods, and data diversity are essential for generalizable forecasting. On the one hand, SNN design would benefit from first-principle inspirations and offer a cost-effective representation of function space. On the other hand, to reduce the dependency on prior knowledge, the inclusion of more optimization penalties (such as system stability and conservation laws) and diverse, heterogeneous data would be exciting directions.

3.4. Probabilistic forecasting

To verify the ability for probabilistic forecasting, we trained the Bayesian SymFM or BSymFM on noisy data from the Lorenz system. Specifically, Gaussian noise was added to the original data at a 1% or 5% level (the ratio of the noise's standard deviation to the noise-free data's standard deviation). The learned model was asked to forecast for 200-step-ahead for the test data.

The SNN design was the same as the first type in deterministic forecasting. As for the number of SVGD particles, $N_p = 10$ was adequate for the sake of accuracy and constrained computational cost in this example. The same setting with 20 particles was also tried, but no significant performance improvement (5.11% for the test RMSE) was noticed when the computational cost doubled. With 2000 training epochs (N_e) and a learning rate $\epsilon = 0.5$, the loss reached convergence. The RMSEs for forecasting noise-free test data under 1% and 5% noise were 0.487 and 0.979, respectively. Some forecasting segments are visualized in Fig. 6, where the 95% confidence interval covers the discrepancies between the predicted and the true signals; meanwhile, the errors accumulate as the system evolves.

Remark 4. Given the same noisy data and optimization settings, the test RMSEs of the DSymFM for 1% and 5% noisy data were 0.594 and 1.472, respectively, which were 21.97% and 50.36% more than those of BSymFM. It shows that noise modeling in the BSymFM improved robustness against noise disturbance.

3.5. Univariate forecasting with the delay coordinate embedding

3.5.1. Comparison with deep learning models and dictionary models

In this section, the DSymFM was compared with two neural network models (LSTM and Informer) and two dictionary models (sparse identification of nonlinear dynamics or SINDy [33] and time-stepping SINDy [39]). The SINDy is a symbolic regression approach that constructs a parsimonious yet predictive ODE by sparsely regressing a dictionary of possible variables on an estimated temporal derivative of a state variable. The time-stepping SINDy is a variant that encodes the governing ODEs into a temporal integration scheme (RK4 in this case) to prevent the explicit estimation of temporal derivatives. In other words, the time-stepping SINDy shared the same spirit as DSymFM except that its backbone was a dictionary instead of an SNN.

The model setups of DSymFM and two neural network models were similar to those of previous studies of the Lorenz and the Rössler systems, except that the SNNs used all embedding coordinates $\mathbf{z}$ as inputs. As for two dictionary methods, a dictionary of $\mathbf{z}$ with the same function search space as that of the DSymFM was built up. The function space contained 3876 combinations for 15 embedding coordinates forming a polynomial series up to the fourth-order. In the vanilla SINDy test, all temporal derivatives were calculated by the fourth-order central difference [33], and boundary points were trimmed to maintain accuracy. Pruning for the dictionary methods was

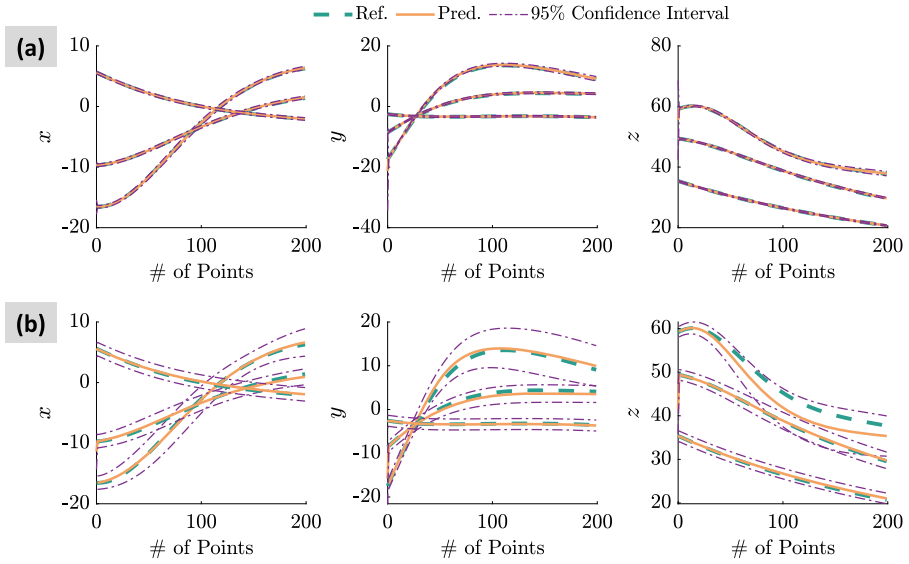

Fig. 6. Probabilistic forecasting of the Lorenz system. (a) Three segments of 200-step-ahead forecasting for test data when the training/validation data had 1% Gaussian noise. (b) Three segments of 200-step-ahead forecasting for test data when the training/validation data had 5% Gaussian noise.

Table 3
Training efforts and forecasting RMSEs of the ET system (Models with the best test score in bold).

Models	Learning rate	Epochs	48-step	96-step	144-step
DSymFM	10^{-2}(50%)[a]	10k	0.177(train)/0.150(val.) **0.135**(test)	0.329(train)/0.213(val.) 0.386(test)	0.358(train)/0.287(val.) 0.414(test)
LSTM	10^{-3}(50%)[b]	10k(2048)[c]	0.360(train)/0.380(val.) 0.421(test)	0.317(train)/0.301(val.) 0.356(test)	0.452(train)/0.468(val.) 0.523(test)
Informer	10^{-4}(10%)[d]	12(32)[c]	0.088(train)/0.138(val.) 0.165(test)	0.193(train)/0.247(val.) **0.281**(test)	0.225(train)/0.291(val.) **0.350**(test)
Vanilla SINDy	Not Applicable[e]	Not Applicable[e]	3.175(train)/2.296(val.) 2.448(test)	2.957(train)/0.744(val.) 2.037(test)	3.333(train)/0.937(val.) 2.293(test)
Time-stepping SINDy	10^{-3}(50%)[a]	400	0.227(train)/0.199(val.) 0.334(test)	0.367(train)/0.357(val.) 0.744(test)	0.429(train)/0.614(val.) 0.796(test)

[a] The learning rate decayed to the percentage in the parentheses in the second half of the training.
[b] The learning rate decayed to the percentage in the parentheses when the validation error stopped decreasing for 10 epochs.
[c] In every epoch, the optimizer conducted mini-batch training where the mini-batch size is indicated in the parentheses.
[d] The learning rate decayed to the percentage in the parentheses every two epochs.
[e] SINDy solved least squares regression while adaptively thresholding minor model parameters to achieve sparsity.

done by grid-searching the best threshold to eliminate the most dictionary coefficients without causing an abrupt decline in the accuracy of validation data. The grid-search interval started from the smallest absolute value and ended at the greatest absolute value from all dictionary coefficients. The interval was uniformly split into ten sub-intervals, and one threshold was tried from each sub-interval. After pruning, trajectory forecasting in the vanilla SINDy method was done by integrating the discovered ODEs with an RK4 scheme.

Training efforts for DSymFM, neural network models, and dictionary models are shown in Table 3. Though these efforts were empirically determined, they enabled convergence of training loss to a stable level. Unlike gradient-based methods, the solution to SINDy was obtained by linear regression, specifically a QR solver [66] in MATLAB. Time-stepping SINDy, similar to the DSymFM, relied on gradient descent to optimize its model parameters (i.e., all dictionary coefficients).

The forecasting performance for all models is shown in Table 3. At first glance, DSymFM was the best for 48-step-ahead, but it became a bit inferior in long-term forecasting (as shown in Fig. 7), especially when we consider that the longest forecasting steps 144 were still much smaller than that of the Lorenz and the Rössler systems. A fundamental reason may be that the current DSymFM did not effectively leverage long-term information from past measurements. Compared with DSymFM, both LSTM and Informer had an input sequence whose length is twice as long as that of the output length, not to mention their recognized memory mechanisms (i.e., the cell state for LSTM and the attention technique for Informer). Even though the delay coordinate embedding condensed some temporal information, its effect seemed limited. To address this issue in long-term forecasting, we may utilize a multi-step numerical method such as the Adams–Bashforth family of methods [67] or build up a more flexible mapping from previous time steps to the next step by training an attention mechanism. Implicit integration is also a good option for stabilizing long-term forecasting.

Results of the vanilla SINDy and the time-stepping SINDy (RK4-SINDy) showed some defects in the dictionary methods. First of all, RMSEs of the vanilla SINDy were the highest largely due to its big dictionary matrix that had 3876 function combinations and over 41k samples. After random downsampling, the number of samples was reduced to about 4k so that the computational time became

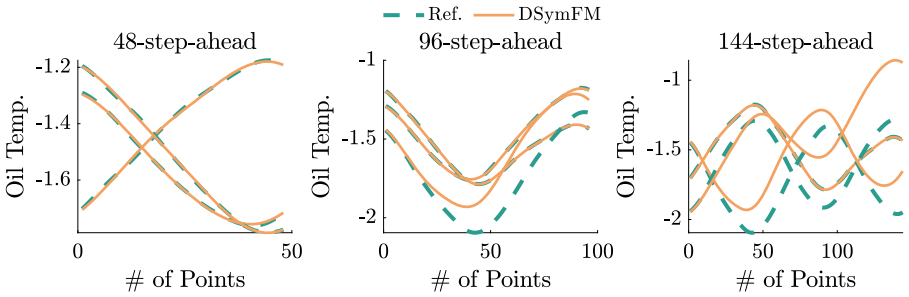

Fig. 7. Forecasting of the electrical transformer (ET) system by the deterministic symbolic forecasting model (DSymFM) with different forecasting horizons.

Table 4
Reconstruction errors of different N_{dc} for the ET system.

	$N_{dc}=50$ $r=4$	$N_{dc}=100$ $r=8$	$N_{dc}=200$ $r=15$	$N_{dc}=300$ $r=21$
Recons. Err.	0.34%	0.17%	0.11%	0.12%

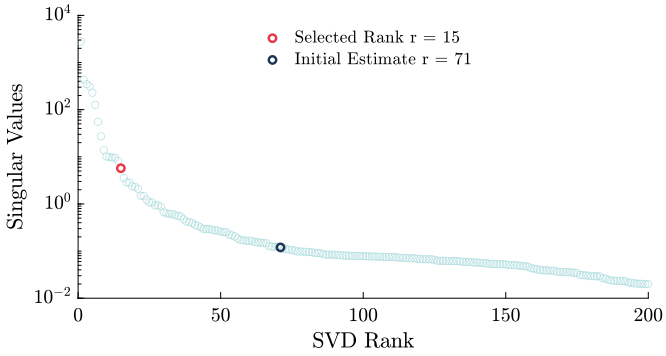

Fig. 8. The singular values of a Hankel matrix whose $N_{dc}=200$ for the electrical transformer (ET) system. The blue circle is the initial estimation by the optimal hard-thresholding principle [60]. The red circle is the final truncation with consideration of the initial estimation, the computational cost and the location of the adjacent energy gaps.

more acceptable. However, regression with such a big matrix was ill-conditioned, and lots of NaNs and Infs appeared in the predicted trajectories, which were removed to provide a valid error metric. Secondly, RMSEs of time-stepping SINDy were better than vanilla SINDy, because there was neither derivative estimation by finite difference nor linear regression of a huge matrix. However, the number of model parameters ($3876\times 15=58140$) was about 45 times more than that of 15 SNNs ($15\times\{16+\mathcal{C}(\log_2 4)[2\times 16+\mathcal{C}(\log_2 4)]\}=1260$ based on Section 2.1.3), which led to a speed contrast of 68.8 sec/epoch to 0.1 sec/epoch. Even though sufficient epochs were provided for achieving loss convergence, many NaNs and Infs still populated the predicted trajectories from time-stepping SINDy, possibly because that a high dimensional parameter space aggravated the optimization non-convexity and stability. In contrast, parameters in SNNs, as a low dimensional manifold of the function space, did not produce invalid values for the same prediction horizons.

3.5.2. Determination of hyper-parameters in the delay coordinate embedding

To construct the delay coordinate embedding, the dimension of delay coordinates N_{dc} and the SVD truncation rank r are critical hyper-parameters.

The dimension of delay coordinates N_{dc}. The bigger N_{dc} is, the more the new embedding will look like Fourier modes since the embedding space will be more linear [62]. In practice, we determined N_{dc} by comparing the reconstruction accuracy of the Hankel matrix for the validation data. Specifically, we first computed the SVD of the Hankel matrix formed by the training data $\mathbf{H}^{tr}=\mathbf{U}^{tr}\mathbf{\Sigma}^{tr}\mathbf{V}^{tr\intercal}$. Secondly, we determined the truncated SVD rank r following the technique mentioned in the following paragraph. Thirdly, the right-singular vector $\hat{\mathbf{V}}^{val}$ for the validation data was approximated by solving $\mathbf{H}^{val}=\mathbf{U}^{tr}\mathbf{\Sigma}^{tr}\hat{\mathbf{V}}^{val\intercal}$. Then, we reconstructed the validation Hankel matrix $\hat{\mathbf{H}}^{val}$ by truncated $\mathbf{U}_r^{tr}$, $\mathbf{\Sigma}_r^{tr}$ and $\hat{\mathbf{V}}_r^{val}$. Lastly, $\hat{\mathbf{H}}^{val}$ was compared with the original validation Hankel matrix $\mathbf{H}^{val}$, and the reconstruction error was computed. For the ET system, the reconstruction errors for different N_{dc} are listed in Table 4, where we chose $N_{dc}=200$ and $r=15$ at the end since its reconstruction error was the lowest.

The SVD truncation rank r. A small r will leave out helpful information, while a big r will include noise perturbation in the new embedding. A big r also induces more computational burden since each embedding dimension needs an SNN to describe its evolution function. We first referred to an optimal hard-thresholding principle [60] (resembling the denoised signal) to initialize the rank guess. We then adjusted this estimate to an evident energy gap in the singular value plot. For instance, in the training data of the ET system, when $N_{dc}=200$, r was suggested to be 71 based on the optimal hard-thresholding principle [60]. By visual inspection (Fig. 8), the adjacent energy gap at $r=15$ was selected eventually.

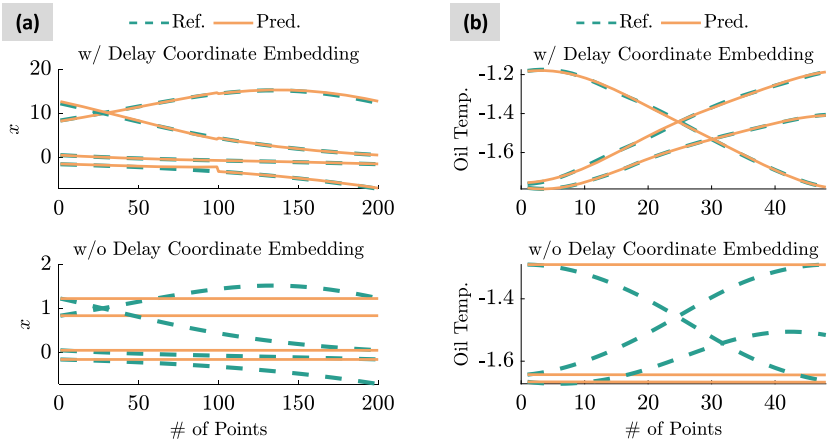

Fig. 9. Univariate forecasting by the deterministic symbolic forecasting model (DSymFM) without delay coordinate embedding. (a) The 200-step-ahead univariate forecasting of Lorenz system. (b) The 48-step-ahead univariate forecasting of the ET system.

3.5.3. Univariate forecasting without the delay coordinate embedding

If we did not resort to a dimension transformation method like the delay coordinate embedding, a direct implementation of the SymFM would result in meaningless forecasting. We verified this concern in Lorenz (where we only had x data) and ET systems, whose results in Fig. 9 show that the forecasting was flat.

4. Conclusion

We introduce a symbolic forecasting model (SymFM) that can project data patterns onto symbolic structure such that SymFm has the potential to forecast well on unseen data. With adaptive pruning, the symbolic neural network (SNN) [45–47] wrapped in a fourth-order Runge Kutta (RK4) integrator can bear high resemblance to the evolution functions of the system. Compared with dictionary methods [33,39] that also identify symbolic invariance in data, an SNN is more cost-effective in terms of the number of model parameters and easier to optimize for high dimensional function search space. To account for noise perturbation, we also develop the probabilistic version of the SymFM based on a non-parametric Bayesian inference method, Stein variational gradient descent (SVGD) [48]. In the case of partial observability, where a single variable is observed from a system with multiple state variables, we leverage the delay coordinate embedding [49] to lift the original measurement into a self-contained high dimensional system.

Even though trained with a one-step-ahead prediction, the deterministic SymFM (DSymFM) forecast well on a multi-step-ahead horizon for the synthetic data and generalized better than the long short-term memory (LSTM) [11] model and the state-of-the-art Informer model [19]. The model's generalizability could further improve with extra pruning even when the SNN consisted of mild redundant functions. The Bayesian SymFM (BSymFM) could better handle noisy data than its deterministic counterpart and provide probabilistic assessment for forecasting. With the delay coordinate embedding, the DSymFM could handle real-world data like the ET system well (and better than the two deep learning models) when the forecasting horizon was not long. The DSymFM also displayed faster and better-behaved convergence and forecasting behavior than dictionary methods for high dimensional function space since the SNN was a low dimensional representation of the function space.

To conclude, there are some discussions on limitations and future development.

- It was observed from the case studies that prediction performance was sensitive to SNN design. While mild redundancy like multiplication and trigonometric functions in SNN architecture had a minor impact after pruning, the inclusion of arbitrary functions/operators was ill-advised. In the latter case, the generalized performance of SymFM could deteriorate significantly. Due to the disturbance of optimization errors, this could lead a pruned SNN to non-unique fitting to the training data. To alleviate this problem, a principled SNN design (or rigorous design selection), utilization of diverse and possibly heterogeneous data, and regularization inspired by system properties like stability or physical conservation could all be valuable resources to explore.
- Another limitation was that the current SymFM only leveraged immediate information since its backbone was a single-step integrator. This short memory could still lead to fast divergence and instability even with the approximated symbolic invariance. Multi-step methods such as Adam-Bashforth methods [67] or neural-inspired attention mechanisms could be organically absorbed to develop long-term memory in the model. With its preferred inclination for stability, implicit integration is also worth investigating.
- Finally, there are intrinsic characteristics in the delay coordinate embedding that could potentially regularize SNN training. For example, a matrix representation of the latent embedding has antisymmetric and tridiagonal properties [49,68]. Furthermore, this work considers only a scalar trajectory for embedding transformation, whereas a multi-dimensional trajectory can also leverage delay coordinate embedding [69,70]. Unlike empirical principles used in this work, rigorous non-asymptotic analysis [70] can also refine the determination of hyper-parameters during embedding transformation. Additionally, how to reconstruct or forecast the entire state trajectory in the original embedding from the delay coordinate embedding could be an exciting direction to investigate [71].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The work is supported in part by the Beijing Outstanding Young Scientist Program (No. BJJWZYJH012019100020098) as well as the Intelligent Social Governance Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Renmin University of China. We also wish to acknowledge the support in part by the Engineering for Civil Infrastructure program at National Science Foundation under grant CMMI-2013067, the research award from MathWorks, and the Tier 1 Seed Grant Program at Northeastern University. We highly appreciate the constructive comments and suggestions from anonymous reviewers, which help improve this work's quality to a new level.

References

[1] P. Whittle, Hypothesis Testing in Time Series Analysis, vol. 4, Almqvist & Wiksells boktr., 1951.
[2] G.E. Box, D.A. Pierce, J. Am. Stat. Assoc. 65 (332) (1970) 1509–1526.
[3] H. Spliid, J. Am. Stat. Assoc. 78 (384) (1983) 843–849.
[4] S.A. Billings, Nonlinear System Identification, NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains, John Wiley & Sons, 2013.
[5] B.E. Boser, I.M. Guyon, V.N. Vapnik, in: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, 1992, pp. 144–152.
[6] H. Drucker, C.J. Burges, L. Kaufman, A. Smola, V. Vapnik, et al., Adv. Neural Inf. Process. Syst. 9 (1997) 155–161.
[7] A. Surana, J. Nonlinear Sci. (2018) 1–34.
[8] S. Sinha, B. Huang, U. Vaidya, in: 2018 Annual American Control Conference (ACC), IEEE, 2018, pp. 5491–5496.
[9] J.-C. Hua, F. Noorian, D. Moss, P.H. Leong, G.H. Gunaratne, Nonlinear Dyn. 90 (3) (2017) 1785–1806.
[10] W.W. Wei, in: The Oxford Handbook of Quantitative Methods in Psychology, vol. 2, 2006.
[11] S. Hochreiter, J. Schmidhuber, Neural Comput. 9 (8) (1997) 1735–1780.
[12] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint arXiv:1412.3555, 2014.
[13] B.B. Sahoo, R. Jha, A. Singh, D. Kumar, Acta Geophys. 67 (5) (2019) 1471–1481.
[14] H. Goel, I. Melnyk, A. Banerjee, R2n2: residual recurrent neural networks for multivariate time series forecasting, arXiv preprint arXiv:1709.03159, 2017.
[15] H. Hewamalage, C. Bergmeir, K. Bandara, Int. J. Forecast. 37 (1) (2021) 388–427.
[16] Z. Xiang, J. Yan, I. Demir, Water Resour. Res. 56 (1) (2020) e2019WR025326.
[17] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A.N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, et al., Tensor2tensor for neural machine translation, arXiv preprint arXiv:1803.07416, 2018.
[18] N. Wu, B. Green, X. Ben, S. O'Banion, Deep transformer models for time series forecasting: the influenza prevalence case, arXiv preprint arXiv:2001.08317, 2020.
[19] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: beyond efficient transformer for long sequence time-series forecasting, arXiv preprint arXiv:2012.07436, 2020.
[20] C.M. Bishop, Neural Comput. 7 (1) (1995) 108–116.
[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[22] M. Raissi, P. Perdikaris, G.E. Karniadakis, J. Comput. Phys. 378 (2019) 686–707.
[23] Y. Yang, P. Perdikaris, J. Comput. Phys. 394 (2019) 136–152.
[24] L. Sun, H. Gao, S. Pan, J.-X. Wang, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732.
[25] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, J. Comput. Phys. 394 (2019) 56–81.
[26] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, SIAM Rev. 63 (1) (2021) 208–228.
[27] M. Raissi, J. Mach. Learn. Res. 19 (1) (2018) 932–955.
[28] S.H. Rudy, J.N. Kutz, S.L. Brunton, J. Comput. Phys. 396 (2019) 483–506.
[29] J.R. Dormand, P.J. Prince, J. Comput. Appl. Math. 6 (1) (1980) 19–26.
[30] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[31] T. Chen, H. Chen, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917.
[32] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, G.E. Karniadakis, J. Chem. Phys. 154 (10) (2021) 104118.
[33] S.L. Brunton, J.L. Proctor, J.N. Kutz, Proc. Natl. Acad. Sci. 113 (15) (2016) 3932–3937.
[34] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Sci. Adv. 3 (4) (2017) e1602614.
[35] H. Schaeffer, Proc. R. Soc. A, Math. Phys. Eng. Sci. 473 (2197) (2017) 20160446.
[36] Z. Wang, X. Huan, K. Garikipati, Comput. Methods Appl. Mech. Eng. 356 (2019) 44–74.
[37] S. Zhang, G. Lin, Proc. R. Soc. A, Math. Phys. Eng. Sci. 474 (2217) (2018) 20180305.
[38] G. Tran, R. Ward, Multiscale Model. Simul. 15 (3) (2017) 1108–1129.
[39] K. Kaheman, S.L. Brunton, J.N. Kutz, Automatic differentiation to simultaneously identify nonlinear dynamics and extract noise probability distributions from data, arXiv preprint arXiv:2009.08810, 2020.
[40] Z. Chen, Y. Liu, H. Sun, Deep learning of physical laws from scarce data, 2020, arXiv:e-prints, arXiv–2005.
[41] G.-J. Both, S. Choudhury, P. Sens, R. Kusters, J. Comput. Phys. 428 (2021) 109985.
[42] M. Schmidt, H. Lipson, Science 324 (5923) (2009) 81–85.
[43] H. Xu, H. Chang, D. Zhang, J. Comput. Phys. 418 (2020) 109584.
[44] H. Vaddireddy, A. Rasheed, A.E. Staples, O. San, Phys. Fluids 32 (1) (2020) 015113.
[45] S. Sahoo, C. Lampert, G. Martius, in: International Conference on Machine Learning, PMLR, 2018, pp. 4442–4450.
[46] Z. Long, Y. Lu, B. Dong, J. Comput. Phys. 399 (2019) 108925.
[47] S. Kim, P.Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, M. Soljačić, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[48] Q. Liu, D. Wang, Stein variational gradient descent: a general purpose bayesian inference algorithm, arXiv preprint arXiv:1608.04471, 2016.
[49] S.L. Brunton, B.W. Brunton, J.L. Proctor, E. Kaiser, J.N. Kutz, Nat. Commun. 8 (1) (2017) 1–9.
[50] K. He, X. Zhang, S. Ren, J. Sun, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[51] R.A. Brualdi, Introductory Combinatorics, Pearson Education, India, 1977.
[52] Y. Wang, Q. Liu, Fish. Res. 77 (2) (2006) 220–225.
[53] S. Kullback, R.A. Leibler, Ann. Math. Stat. 22 (1) (1951) 79–86.
[54] J.-P. Vert, K. Tsuda, B. Schölkopf, Kernel Meth. Comput. Biol. 47 (2004) 35–70.
[55] Y. Zhu, N. Zabaras, J. Comput. Phys. 366 (2018) 415–447.
[56] M.E. Tipping, J. Mach. Learn. Res. 1 (Jun 2001) 211–244.
[57] H. Kim, R. Eykholt, J. Salas, Phys. D: Nonlinear Phenom. 127 (1–2) (1999) 48–60.
[58] F. Takens, in: Dynamical Systems and Turbulence, Warwick 1980, Springer, 1981, pp. 366–381.
[59] L. De Lathauwer, B. De Moor, J. Vandewalle, SIAM J. Matrix Anal. Appl. 21 (4) (2000) 1253–1278.
[60] M. Gavish, D.L. Donoho, IEEE Trans. Inf. Theory 60 (8) (2014) 5040–5053.
[61] H. Arbabi, I. Mezic, SIAM J. Appl. Dyn. Syst. 16 (4) (2017) 2096–2126.
[62] K.P. Champion, S.L. Brunton, J.N. Kutz, SIAM J. Appl. Dyn. Syst. 18 (1) (2019) 312–333.
[63] E.N. Lorenz, J. Atmos. Sci. 20 (2) (1963) 130–141.
[64] O.E. Rössler, Phys. Lett. A 57 (5) (1976) 397–398.

[65] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[66] W. Gander, Res. Rep. 80 (02) (1980) 1251–1268.
[67] J.C. Butcher, Numerical Methods for Ordinary Differential Equations, John Wiley & Sons, 2016.
[68] S.M. Hirsh, S.M. Ichinaga, S.L. Brunton, J. Nathan Kutz, B.W. Brunton, Proc. R. Soc. A 477 (2254) (2021) 20210097.
[69] S.P. Garcia, J.S. Almeida, Phys. Rev. E 72 (2) (2005) 027205.
[70] S. Pan, K. Duraisamy, Chaos, Interdiscip. J. Nonlinear Sci. 30 (7) (2020) 073135.
[71] J. Bakarji, K. Champion, J.N. Kutz, S.L. Brunton, Discovering governing equations from partial measurements with deep delay autoencoders, arXiv preprint arXiv:2201.05136, 2022.