A non-autonomous equation discovery method for time signal classification[*]

Ryeongkyung Yoon[†], Harish S. Bhat[‡], and Braxton Osting[§]

Abstract. Certain neural network architectures, in the infinite-depth limit, lead to systems of nonlinear differential equations. Motivated by this idea, we develop a framework for analyzing time signals based on non-autonomous dynamical systems. We view the time signal as a forcing function for a dynamical system that governs a time-evolving hidden variable. As in equation discovery, the dynamical system is represented using a dictionary of functions and the coefficients are learned from data. This framework is applied to the time signal classification problem. We show how gradients can be efficiently computed using the adjoint method, and we apply methods from dynamical systems to establish stability of the classifier. Through a variety of experiments, on both synthetic and real datasets, we show that the proposed method uses orders of magnitude fewer parameters than competing methods, while achieving comparable accuracy. We created the synthetic datasets using dynamical systems of increasing complexity; though the ground truth vector fields are often polynomials, we find consistently that a Fourier dictionary yields the best results. We also demonstrate how the proposed method yields graphical interpretability in the form of phase portraits.

Key words. Time signal analysis; classification; equation discovery; neural networks; adjoint method

AMS subject classifications. 34H05, 68T07, 62L10

1. Introduction.

Time series classification has been applied in a variety of fields including predicting the genre of music based on a sound recording [36], recognizing human activity using mobile sensors [37], diagnosing disease based on electrical biosignals (*e.g.*, EEG, ECG, and EMG) [35, 33, 34], detecting natural phenomena such as earthquakes or volcanic eruptions using geophysical signals [26], and automatically distinguishing between mosquito species using wing-beat recordings [8].

One promising approach to time series classification involves recurrent neural networks (RNNs) [29, 9], which are now commonly used to process sequential data. For input data, $x_t \in \mathbb{R}^n$ and a hidden state vector $h_t \in \mathbb{R}^m$ (typically initialized with $h_0 = 0$), a traditional sequence-to-label RNN architecture can be represented abstractly as a discrete-time map:

$$h_t = f(h_{t-1}, x_t; \theta) \text{ for } t \in [T]. \tag{1.1}$$

with parameter vector θ—see [13, Eq. (10.5)]. The output layer is then formulated as $\hat{y} = \sigma(Ah_T + b)$, where σ is a user-specified activation function, and for classification problems, σ is typically the softmax function. To train an RNN, we learn parameters θ, A, and b. RNNs are Turing complete; for any function $\mathscr{F}$ computable by a Turing machine, there exists a finite RNN that can compute $\mathscr{F}$ [31, 32]. Note also that (1.1) encapsulates a large class of

[*]Submitted to the editors October 31, 2021.

Funding: B. Osting acknowledges partial support from NSF DMS 17-52202. H. S. Bhat acknowledges partial support from NSF DMS 17-23272.

[†]Department of Mathematics, University of Utah, Salt Lake City, UT (rkyoon@math.utah.edu).

[‡]Department of Applied Mathematics, University of California, Merced, CA (hbhat@ucmerced.edu).

[§]Department of Mathematics, University of Utah, Salt Lake City, UT (osting@math.utah.edu).

This manuscript is for review purposes only.

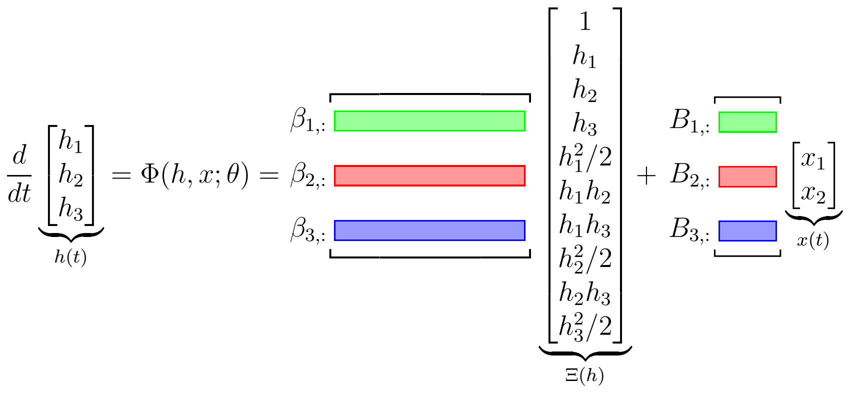
$\frac{d}{dt}\underbrace{\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}}_{h(t)} = \Phi(h,x;\theta) =$
$\beta_{1,:}$
$\beta_{2,:}$
$\beta_{3,:}$
$\underbrace{\begin{bmatrix} 1 \\ h_1 \\ h_2 \\ h_3 \\ h_1^2/2 \\ h_1h_2 \\ h_1h_3 \\ h_2^2/2 \\ h_2h_3 \\ h_3^2/2 \end{bmatrix}}_{\Xi(h)}$
$+$
$B_{1,:}$
$B_{2,:}$
$B_{3,:}$
$\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x(t)}$

Figure 1. *We illustrate the hidden state model* (1.4) *together with* $\Phi(h, x; \theta) = \beta\Xi(h) + Bx$*—see* (3.2)*—and* Ξ *set to a polynomial dictionary. In* section 4*, we refer to this as a Poly (3,2) dictionary;* $m = 3$ *is the dimension of* h*, while* $k = 2$ *means that the dictionary includes terms up to quadratic order. The matrices* β *and* B *have dimensions* 3×10 *and* 3×2*, respectively; the colored bars, labeled* $\beta_{i,:}$ *and* $B_{i,:}$*, denote the* i*-th rows of these matrices. Note that the right-hand side* Φ *can be* nonlinear *in* h *while remaining* linear *in the parameters* β *and* B *that we seek to learn. The total set of parameters for the NAED model is* $\Theta = \{\beta, B, A, b\}$ *where* A *and* b *are defined in* (3.1c)*.*

discrete-time, non-autonomous dynamical systems with state variable h_t and forcing x_t. Given their universality and capacity, it is not surprising that RNNs can serve as accurate models for sequential data, including time series [1, Chap. 7]. However, RNNs can be difficult to train due to long-term dependencies and suffer from computational issues in backpropagation through time, called *exploding* or *vanishing* gradients [25]. Gated RNNs, such as the Long Short Term Memory (LSTM) network were developed in [16] to overcome the challenge of long-term dependencies.

In this paper, we propose a non-autonomous dynamical systems framework that addresses challenges in training RNNs. Note that we distinguish between time signals and time series; time signals are continuous in time, while time series are discrete in time. Frequently time series are obtained from sampling a time signal at discrete times. The *time signal classification* problem is to learn a mapping that assigns a distribution over labels $y \in \mathbb{R}^{|\mathcal{Y}|}$ to a vector-valued, continuous-time signal $x\colon [0, T] \to \mathbb{R}^n$. Here $\mathcal{Y}$ is a finite set of labels.

From (1.1), we derive a continuous-time model as follows. We first insert $N - 1$ hidden layers between h_{t-1} and h_t and consider the discrete-time map

$$h_t = f(h_{t-1/N}, x_t; \theta) \quad \text{for} \quad t = i/N \quad \text{with} \quad i \in [NT]. \tag{1.2}$$

When $N = 1$, we recover (1.1). For $N \gg 1$, the model has a deep hidden-to-hidden transition [24]; N layers must be traversed to go from h_{t-1} to h_t. Next,

$$h_t = h_{t-1/N} + N^{-1}\Phi(h_{t-1/N}, x_t; \theta) \quad \text{for} \quad t = i/N \quad \text{with} \quad ri \in [NT]. \tag{1.3}$$

Finally, we take the *infinite-depth* limit $N \to \infty$ and obtain the central equation in the *non-autonomous equation discovery (NAED) method*:

$$\frac{d}{dt}h(t) = \Phi\left(h(t), x(t); \theta\right) \qquad \text{for } t \in [0, T]. \tag{1.4}$$

This manuscript is for review purposes only.

We view the input signal $x(t)$ as a forcing term in a non-autonomous dynamical system governing a hidden variable $h\colon [0,T] \to \mathbb{R}^m$. We introduce a function $\pi\colon \mathbb{R}^m \to \mathbb{R}^{|\mathcal{Y}|}$ to assign a class label to the hidden variable evaluated at the final time, $\hat{y} = \pi\left(h(T)\right)$. The objective is to learn the right-hand side Φ, parameterized by θ, and the function π, so that given a new time signal $x(t)$, $t \in [0,T]$ we can estimate its class label, y.

In the NAED method, we represent the right-hand side function Φ using a predetermined *dictionary*, a set of candidate functions, which is sufficiently large to capture a wide class of dynamics. There are a variety of choices for dictionaries; here we employ polynomial and Fourier basis functions. See Figure 1 for an illustration of (1.4) with (3.2)—our model for Φ which is *linear* in the parameters to be learned—in the special case of a quadratic polynomial dictionary. In addition to a general NAED method, we also propose a sparse NAED method, which drops less relevant dictionary functions from the learned expression of the classifier. By iteratively thresholding the dictionary coefficients, we improve generalizability (reduce overfitting), robustness to noise and interpretability of learned dynamics.

In section 3, we describe the NAED method in more detail, including an efficient computation of the gradient of the loss function using the adjoint method. In practice, we are given a time series, which we think of as a discretized time signal and we must also discretize the dynamical system to obtain a discrete-time approximation of the hidden variable. In this paper, we employ the optimize-then-discretize approach, where the gradient is computed analytically (see Theorem 3.3) using the continuous-time hidden variable and input time signal, and then evaluated using the time series and discretized hidden variable. This in contrast to a discretize-then-optimize approach that begins with discrete-time models such as (1.1), and then optimizes using gradients computed via backpropagation-through-time.

In section 3, we prove several theoretical results about the NAED method. We prove a sufficient condition for existence/uniqueness of the proposed method's solutions (Theorem 3.2). We also quantify the method's stability, that is, we show that the outputs of the classifier are stable with respect to both deterministic and random perturbations (Theorem 3.4, Theorem 3.5).

In section 4, we report the results of several computational experiments that demonstrate the competitive performance of NAED with respect to RNN-based methods. We carry out these experiments both for synthetic data and for real data from the UCR Time Series Classification Archive [7]. In these experiments, NAED achieves similar or better accuracy than recurrent neural network methods (including LSTM and CFN architectures) and neural controlled differential equations (NCDE). We also show how the NAED method finds a principled and parsimonious dictionary representation of the dynamical system's vector field by training orders of magnitude fewer parameters. NAED seeks to blend the high accuracy of deep RNN architectures with the interpretability of continuous-time dynamical system methods. In particular, we illustrate that trained NAED models can be interpreted graphically using phase portraits.

We conclude in section 5 with a discussion of the NAED method and ideas for future directions.

2. Related Work.

In this section, we discuss two motivations for the NAED method: an infinite depth, continuous-time limit of RNNs and the equation discovery method.

2.1. Dynamical systems and RNNs. Continuous-time RNNs were proposed by Hopfield [17] and studied by many authors—see [11, 2] and references therein. Early continuous-time RNNs were proposed as models of associative memory and hence are not directly comparable to the classifiers studied here. Still, early continuous-time RNNs share two features with NAED: the models are expressed as systems of nonlinear differential equations, and inputs are treated as non-autonomous forcing terms. Compared to NAED, early continuous-time RNNs have a rigid right-hand side structure that guarantees Lyapunov stability of the unforced system [17]. In contrast, NAED has a flexible right-hand side Φ that we can often represent as a sparse linear combination of dictionary functions.

More recently, there has been a growing literature that connects deep and recurrent neural networks with ordinary differential equations (ODEs). One branch of this literature seeks to apply ideas from dynamical systems theory to determine stable feedforward architectures [14], RNNs that do not exhibit chaotic dynamics [21], and RNNs that are constrained to be linearly stable [5]. The RNNs considered in these works [21, 5] do not involve ODEs.

Another branch stems from Neural ODEs or ODE-Nets [6]. We view both NAED and Neural ODEs as infinite-depth limits of deep networks that are trained via the adjoint method rather than backpropagation. In Neural ODEs, the vector field is typically modeled using a (static) feedforward neural network (rather than with a dictionary), and the input is used as an initial condition (rather than a forcing term) to the ODE system. Recent efforts have sought to make Neural ODE techniques more practical for large-scale problems [10, 12, 27] and also to better understand the learning of genuinely continuous-time dynamics [23]; we may be able to apply similar ideas to NAED in future work.

Recently, there has been some effort to generalize Neural ODE models to the RNN context. Instead of relying purely on ODEs as in NAED, [28] combines ODE-Net and RNN layers. We also find continuous-time versions of GRU and LSTM models [3, 18, 15]. Compared with NAED, these architectures have more constraints on the right-hand side vector field Φ. Finally, the NAED dynamical system (1.4) can be viewed as a special case of the recently proposed neural controlled differential equation (NCDE) model [20]. Compared with NAED, the controlled differential equation allows for more general dependency of the hidden state $h(t)$ on the input $x(t)$. While NCDE uses a neural network model of the vector field, NAED uses a dictionary.

2.2. Equation Discovery. The dictionary representation of the vector field Φ is motivated by the literature on equation discovery [4]. The problem formulation and goal in equation discovery differs from ours; there one assumes that the data consists of observations of the state vector $h(t)$ of a continuous-time dynamical system. Using this data, the goal is to learn the vector field Φ. This is a nonparametric regression problem, equivalent to finding a system of ordinary differential equations that fit the observations $h(t)$. The Sparse Identification of Nonlinear Dynamics (SINDy) method assumes that Φ can be represented as a sparse linear combination of elements from a dictionary Ξ [4]. In SINDy, training proceeds via an iteratively thresholded least squares method whose convergence has been established [38].

In both the general NAED method (Algorithm 3.1) and the sparse NAED method (Algorithm 3.2), we generalize SINDy in the following way: we do not assume access to $h(t)$ at all, but rather the forcing function $x(t)$. Learning Φ is a byproduct of our method, but the goal is to train a model whose predictions $\hat{y}$ match the true labels y. In Algorithm 3.2, we retain the iterative thresholding step from SINDy. However, in Algorithm 3.1—the algorithm that we use in all of the examples in subsection 4.2—we do not promote sparsity of any of the coefficient matrices. As we show, even with dense coefficient matrices, representing Φ with a dictionary requires fewer parameters than with a neural network.

3. Non-autonomous equation discovery (NAED) method.

In this section, we describe our proposed non-autonomous equation discovery (NAED) method for time series classification, a gradient-based method for training it, our choice of dictionary in the NAED method, stability of the classifier, and a sparse version of the method.

3.1. NAED model for time signal classification.

We assume that we are given data of the form $\{x_i, T_i, y_i\}_{i\in[N]}$, where $x_i\colon [0,T_i] \to \mathbb{R}^n$ is a time signal and $y_i \in \mathbb{R}^{|\mathcal{Y}|}$ is a probability mass function over the classes. In practice, y_i will be a unit vector and $\arg\max_i y_i$ will be the class or label. Note that we allow for the possibility that the time signals have different lengths. We consider the following non-autonomous dynamical system:

$$\frac{d}{dt}h_i(t) = \Phi\left(h_i(t), x_i(t); \theta\right), \qquad t \in [0,T] \tag{3.1a}$$

$$h_i(0) = h_0. \tag{3.1b}$$

For each time signal $x_i(t)$, we interpret the solution to (3.1), $h_i(t) \in \mathbb{R}^m\ \forall t \in [0,T_i]$ as a time-dependent hidden variable that is being *forced* by the function $x_i(t)$. The solution at time T_i is used to make a class prediction $\hat{y}_i$ via

$$\hat{y}_i = \sigma\left(Ah_i(T_i) + b\right), \tag{3.1c}$$

where $A \in \mathbb{R}^{|\mathcal{Y}|\times m}$, $b \in \mathbb{R}^{|\mathcal{Y}|}$, and $\sigma\colon \mathbb{R}^{|\mathcal{Y}|} \to \mathbb{R}^{|\mathcal{Y}|}$ is the softmax function, defined by $[\sigma(x)]_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$.

We parameterize the vector field $\Phi\colon \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ using a dictionary $\mathcal{D} = \{\xi_j\}_{j\in[d]}$, with $\xi_j\colon \mathbb{R}^m \to \mathbb{R}$. We discuss specific choices for the dictionary, $\mathcal{D}$, in subsection 3.3, but we have in mind, *e.g.*, multivariate polynomials. Let $\theta = (\beta, B)$. Concatenating the dictionary elements in a dictionary, $\Xi(h) = (\xi_1(h), \xi_2(h), \cdots, \xi_d(h)) \in \mathbb{R}^d$, we write

$$\Phi(h,x;\theta) = \beta\Xi(h) + Bx, \tag{3.2}$$

where $\beta \in \mathbb{R}^{m\times d}$ and $B \in \mathbb{R}^{m\times n}$ are unknown coefficients.

To train the classifier, we must learn $\theta = (\beta, B)$, determining Φ via (3.2), together with the parameters A and b in (3.1c). We frame this learning problem as one of minimizing the following cross-entropy loss between labels y_i and predictions $\hat{y}_i$. For a regularization term $R(\Theta)$, the objective function is then

$$J(\Theta) = -\frac{1}{N}\sum_{i\in[N]}\sum_{j\in\mathcal{Y}}[y_i]_j \log[\hat{y}_i]_j + R(\Theta) \tag{3.3a}$$

$$= -\frac{1}{N}\sum_{i\in[N]}\sum_{j\in\mathcal{Y}}[y_i]_j \log[\sigma\left(Ah_i(T_i) + b\right)]_j + R(\Theta), \tag{3.3b}$$

where $\Theta = \{\beta, B, A, b\}$ represents all parameters to be learned. It is understood that h_i satisfies (3.1) for the forcing $x_i(t)$, $t \in [0, T_i]$.

Remark 3.1. *The regularization term, $R(\Theta)$, in (3.3) could be Tikhonov (ℓ^2) or sparsity promoting (e.g., ℓ^1). In our numerical experiments, we use a relatively small dictionary and, for simplicity, take $R = 0$. In subsection 3.5, we will discuss a different approach to regularization.*

An important consideration is whether there exists a solution of the dynamical system in (3.1) with right-hand side given by (3.2). The following theorem gives a sufficient condition for the existence and uniqueness of a solution.

Theorem 3.2. *Assume $x\colon [0,t] \to \mathbb{R}^n$ is a continuous function. Let $K \subset \mathbb{R}^m$ be a compact set containing the initial point h_0 such that $\xi_i\colon \mathbb{R}^m \to \mathbb{R}$ is a locally Lipschitz continuous function on K with Lipschitz constant $\mathcal{L}$ for every $i \in [d]$, i.e., $\forall h_1, h_2 \in K$, $|\xi_i(h_1) - \xi_i(h_2)| \leq \mathcal{L}\|h_1 - h_2\|$. Then there is an $\varepsilon > 0$ such that the initial value problem in (3.1) has a unique solution defined on the interval $[-\varepsilon, \varepsilon]$.*

Proof. Let $\Phi(h, x(t))$ be rewritten as $\Gamma(h,t) = \Phi(h, x(t))$. For some $r > 0$ and $a > 0$, define $B_r = \{\|h - h_0\| \leq r\} \subset K, I_a = \{|t| \leq a\}$. Since x is continuous in time, there exists a constant $M > 0$ such that

$$M = \max_{(h,t)\in B_r \times I_a} \|\Gamma(h,t)\|.$$

Also for every $t \in I_a$, $h \mapsto \Gamma(h,t)$ satisfies the local Lipschitz condition on K: for every $h_1, h_2 \in B_r$,

$$\begin{aligned} \|\Gamma(h_1,t) - \Gamma(h_2,t)\| &= \|\beta(\Xi(h_1) - \Xi(h_2))\| \\ &\leq \|\beta\| \, \|(\Xi(h_1) - \Xi(h_2))\| \leq d\mathcal{L}\|\beta\| \, \|h_1 - h_2\|. \end{aligned}$$

From the existence/uniqueness theorem in ordinary differential equations (see, *e.g.*, [30, Theorem 3.2]), there exists a unique solution to (3.1) on the interval $[-\varepsilon, \varepsilon]$, where ε is chosen as $\varepsilon = \min\{a, \frac{r}{M}, \frac{1}{2d\|\beta\|\mathcal{L}}\}$. ■

3.2. Gradient computation and the adjoint method.

For the NAED time signal classifier, training can be formulated as the ODE-constrained optimization problem,

$$\min_{\Theta=\{\beta,B,A,b\}} J(\Theta), \tag{3.4}$$

subject to (3.1) where the objective function $J(\Theta)$ is defined in (3.3). To employ a gradient-based optimization method, we need to compute $\nabla_\Theta J$. However, directly computing the gradient of J with respect to Θ is complicated and computationally expensive because J involves $h_i(T; \Theta)$, the solution to (3.1) at time $t = T$. An alternative method to compute $\nabla_\Theta J$ is to use the adjoint method, as we do in the following theorem.

Theorem 3.3. *The gradients of the objective function in (3.3) with respect to the unknown* *parameters:* $\beta \in \mathbb{R}^{m\times d}$, $B \in \mathbb{R}^{m\times n}$, $A \in \mathbb{R}^{|\mathcal{Y}|\times m}$, *and* $b \in \mathbb{R}^{|\mathcal{Y}|}$ *are given by*

$$\nabla_\beta J = -\sum_{i\in[N]} \int_0^{T_i} \lambda_i(t)\Xi(h_i(t))^t \, dt + \nabla_\beta R \tag{3.5a}$$

$$\nabla_B J = -\sum_{i\in[N]} \int_0^{T_i} \lambda_i(t) x_i(t)^t \, dt + \nabla_B R \tag{3.5b}$$

$$\nabla_A J = -\frac{1}{N}\sum_{i\in[N]} \left(y_i - \sigma(Ah_i(T_i)+b)\right) h_i(T_i)^t + \nabla_A R \tag{3.5c}$$

$$\nabla_b J = -\frac{1}{N}\sum_{i\in[N]} \left(y_i - \sigma(Ah_i(T_i)+b)\right) + \nabla_b R \tag{3.5d}$$

where $\lambda_i(t)$ *for* $t \in [0, T_i]$ *is a solution to the adjoint equation,*

$$\frac{d}{dt}\lambda_i(t) = -[\beta D_h \Xi(h)]^t \lambda_i(t) \tag{3.6a}$$

$$\lambda_i(T_i) = -\frac{1}{N} A^t (y_i - \sigma(Ah_i(T_i)+b)). \tag{3.6b}$$

Proof. We introduce the Lagrange multipliers, $\lambda_i \colon [0, T_i] \to \mathbb{R}^m$, for $i \in [N]$, and the Lagrangian,

$$\begin{aligned} L(\Theta, h_i, \lambda_i) &= J(\Theta) + \sum_{i\in[N]} \int_0^{T_i} \lambda_i^t(t) \left(\dot{h}_i(t) - \Phi(h_i(t), x_i(t))\right) \, dt \\ &= J(\Theta) + \sum_{i\in[N]} \lambda_i^t(T_i) h_i(T_i) - \int_0^{T_i} \dot{\lambda}_i^t(t) h_i(t) + \lambda_i^t(t)\Phi(h_i(t), x_i(t)) \, dt. \end{aligned}$$

Here, we have used integration by parts to rewrite the Lagrangian. Taking the variation of the Lagrangian with respect to $h_k(t)$ gives

$$\delta L = \partial_{h_k(T)} J \delta h_k(T) + \lambda_k^t(T_k)\delta h_k(T_k) - \int_0^{T_k} \left(\dot{\lambda}_k^t(t) + \lambda_k^t D_h \Phi\right) \delta h_k(t) \, dt,$$

where $D_h\Phi = \beta D_h \Xi(h)$ is the Jacobian of Φ with respect to the h. Setting the variation to zero, we find that $\lambda_k(t)$ satisfies the adjoint equation given in (3.6).

The gradients of the objective in (3.5) are then obtained by taking the partial derivatives of the Lagrangian with respect to the unknown parameters, $\Theta = \{\beta, B, A, b\}$. The gradient with respect to β and B are given by

$$\nabla_\beta J = \nabla_\beta L = -\sum_{i\in[N]} \int_0^{T_i} \lambda_i(t)\Xi(h_i(t))^t \, dt + \nabla_\beta R$$

$$\nabla_B J = \nabla_B L = -\sum_{i\in[N]} \int_0^{T_i} \lambda_i(t) x_i(t)^t \, dt + \nabla_B R.$$

This manuscript is for review purposes only.

For the cross-entropy loss function in (3.3), a short computation shows that

$$\nabla_A J = -\frac{1}{N}\sum_{i\in[N]} \left(y_i - \sigma(Ah_i(T_i)+b)\right) h_i(T_i)^t + \nabla_A R$$

$$\nabla_b J = -\frac{1}{N}\sum_{i\in[N]} \left(y_i - \sigma(Ah_i(T_i)+b)\right) + \nabla_b R.$$

Combining these results concludes the proof. ■

The gradients from Theorem 3.3 are used with an optimization method to minimize the cross-entropy loss function (3.1c) and thereby train the model.

3.3. Dictionary choice. In the NAED model, the right-hand side of the dynamical system is given by $\Phi(h,x;\theta) = \beta\Xi(h) + Bx$; see (3.2). The first term is a linear combination of dictionary functions, $\Xi(h) = (\xi_1(h), \xi_2(h), \cdots, \xi_d(h)) \in \mathbb{R}^d$. There is tremendous freedom in selecting the dictionary functions and this choice is paramount to the model. We tested NAED using two different dictionaries, a *polynomial dictionary* and a *Fourier dictionary*, described now in turn.

The polynomial dictionary consists of all possible polynomials of $h \in \mathbb{R}^m$ up to k-th order. For $h \in \mathbb{R}^m$, the dictionary is $\Xi(h) = [1, P_1(h), P_2(h), \ldots P_k(h)]$, where $P_k(h)$ is a basis for homogeneous polynomials of degree k. In this section, we use subscripts to denote scalar components of a vector, *e.g.*, $h = [h_1, h_2, \ldots, h_m]$. We choose the basis $P_k(h)$ to consist of the $\binom{k+m-1}{m-1}$ basis elements of the form $\frac{1}{\alpha_1!\cdots\alpha_m!} h_1^{\alpha_1}\cdots h_m^{\alpha_m}$, where $\sum_{i=1}^m \alpha_i = k$, as appearing in Taylor's theorem. For instance, if $m = 2$, $P_2(h)$ refers to the quadratic polynomials $P_2(h) = \left[h_1^2/2,\ h_1h_2,\ h_2^2/2\right]$. In Figure 1, we illustrate the hidden state model for the polynomial dictionary of order $k = 2$, in the case where $m = 3$ (*i.e.*, with $h \in \mathbb{R}^3$).

Alternatively, we can consider a Fourier dictionary. Using separation of variables for the function $\Phi\colon \mathbb{R}^m \to \mathbb{R}$, we write $\Phi(h) = f_1(h_1)f_2(h_2)\cdots f_m(h_m)$, where $f_i\colon \mathbb{R} \to \mathbb{R}$ for $i = 1, \ldots, m$. We approximate $f_i(x)$ by a finite linear combination of Fourier basis functions, $f_i(x) = a_0^i + \sum_{k=1}^K a_k^i \cos(2\pi kx/L) + b_k^i \sin(2\pi kx/L)$, where L is the period of $f_i(x)$. Each row of the vector $\beta\Xi(h)$ appearing in the RHS of the dynamical system (3.1) can be written as $\prod_{i=1}^m f_i(h_i)$, where the coefficients a_0^i, a_k^i, and b_k^i correspond to entries of β. In other words, our dictionary $\mathcal{D}$ consists of functions given by the outer product of harmonic functions,

$$\Xi(h) = \begin{pmatrix} \sin\left(2\pi k_1 h_1/L\right) \\ \cos\left(2\pi k_1 h_1/L\right)\end{pmatrix} \otimes \begin{pmatrix} \sin\left(2\pi k_2 h_2/L\right) \\ \cos\left(2\pi k_2 h_2/L\right)\end{pmatrix} \otimes\cdots\otimes \begin{pmatrix} \sin\left(2\pi k_m h_m/L\right) \\ \cos\left(2\pi k_m h_m/L\right)\end{pmatrix},$$

where $k_i \in [K]$. For instance, for $m = 2$, and $K = 1$, the dictionary consists of the following 9 functions:

$$\Xi(h) = \Big[1, \cos(2\pi h_1/L), \sin(2\pi h_1/L), \cos(2\pi h_2/L), \sin(2\pi h_2/L), \cos(2\pi h_1/L)\cos(2\pi h_2/L),$$
$$\sin(2\pi h_1/L)\cos(2\pi h_2/L), \cos(2\pi h_1/L)\sin(2\pi h_2/L), \sin(2\pi h_1/L)\sin(2\pi h_2/L)\Big].$$

By the Stone-Weierstrass theorem, the polynomial dictionary and Fourier dictionary are dense in the space of continuous functions and L^2, in the limiting case where $k \to \infty$ and

Algorithm 3.1 General NAED method for time signal classification

```
Input: initial parameters, $\Theta = \{\beta, B, A, b\}$.
for epoch = $1, \dots, N_{epoch}$: do
    Shuffle data and create batches of size $N_{batch}$
    for each batch: do
        (Solve the forward ODE for $h_i$) For the current parameters $\beta$ and $B$, solve the forward ODE in (3.1), i.e., for each example $i \in [N_{batch}]$ and discrete times $t_k$, $k \in [K]$, find $h_i(t_k)$.
        (Make predictions) Assign predictions via (3.1c), i.e., $\hat{y_i} = \sigma(Ah_i(T) + b)$.
        (Solve the adjoint equation for $\lambda_i$) Use the hidden state at the final time $h_i(T)$, to compute the terminal condition and solve the backward ODE in (3.6) i.e. for each example $i \in [N_{batch}]$ and discrete times $t_k$, $k \in [K]$, find $\lambda_i(t_k)$.
        (Compute gradients) Using $h_i(t_k)$ and $\lambda_i(t_k)$, evaluate the gradient of the objective function with respect to the parameters $\nabla_\Theta J$, as in (3.5).
        (Update parameters) Use a gradient-based optimization method, e.g., ADAM method or gradient descent, to update the parameters, $\Theta$.
```

$K \to \infty$, respectively. By choosing these parameters sufficiently large, all smooth dynamical systems can be represented as accurately as is needed.

We would like to apply Theorem 3.2 to guarantee the existence of a unique solution to (3.1). Assuming that the time signal x is continuous, it is enough to choose a dictionary that satisfies the Lipschitz continuity assumption. If we use the Fourier dictionary, then the Lipschitz constant is approximately $\mathcal{L} \approx \frac{2\pi K}{L}$. In this case, our model (3.1) has a unique solution until time T, provided we initialize β with sufficiently small values. On the other hand, if we use the Polynomial dictionary, there are two cases. If only linear terms are used in the dictionary, then the right-hand side is Lipschitz continuous and a unique solution exists on the time interval $[0, T]$. However, if we use higher-order polynomials in the dictionary, the right-hand side is only locally Lipschitz and Theorem 3.2 can only guarantee a solution on a short time interval; the solution may blow up in finite time. In the numerical results in section 4, we will observe that models with the Fourier dictionary are generally more accurate and less sensitive to initialization than models with a nonlinear Polynomial dictionary. In algorithm Algorithm 3.1, we present the process of training the general NAED method.

3.4. Stability of the NAED method. Dynamical systems theory can be used to prove that a given NAED classifier $x \overset{\mathscr{C}}{\longmapsto} y$ is stable to noise; below we do this for both deterministic and stochastic perturbations. For $p \in [1, \infty)$, let $L^p([0, T]; \mathbb{R}^n)$ denote the Bochner space of continuous $\mathbb{R}^n$-valued functions with norm $\|x\|_{L^p([0,T];\mathbb{R}^n)} := \left(\int_0^T |x(t)|^p dt\right)^{\frac{1}{p}}$.

Theorem 3.4. *Consider a NAED classifier $\mathscr{C} \colon L^1([0, T]; \mathbb{R}^n) \to \mathbb{R}^{|\mathcal{Y}|}$, equipped with a dictionary $\Xi \colon \mathbb{R}^m \to \mathbb{R}^d$ that is Lipschitz continuous with constant $\mathcal{L}$. The classifier $\mathscr{C}$ is Lipschitz continuous with constant $L > 0$ defined in the proof. That is, if we have a time signal $x(t)$*

and a noise corrupted version, $\tilde{x}(t) = x(t) + \eta(t)$, then $|\mathscr{C}(\tilde{x}) - \mathscr{C}(x)| \le L\|\eta\|_{L^1([0,T];\mathbb{R}^n)}$.

Proof. In the NAED method with dictionary Ξ, the unperturbed and perturbed hidden variables, h and $\tilde{h}$ satisfy

$$\frac{d}{dt}h = \beta\Xi(h) + Bx$$
$$\frac{d}{dt}\tilde{h} = \beta\Xi(\tilde{h}) + B\tilde{x},$$

with $h(0) = \tilde{h}(0) = h_0$. Let $\mathcal{L}$ denote the Lipschitz constant for the dictionary Ξ. Subtracting these equations, we estimate

$$|h(t) - \tilde{h}(t)| \le \int_0^t \mathcal{L}\|\beta\|\,|h(\tau) - \tilde{h}(\tau)|\,d\tau + \int_0^t \|B\|\,|\eta(\tau)|\,d\tau.$$

Since $\int_0^t \|B\|\,|\eta(\tau)|\,d\tau$ is a non decreasing function, Gronwall's inequality yields

$$|h(T) - \tilde{h}(T)| \le \|B\|\left(\int_0^T |\eta(\tau)|\,d\tau\right) e^{\mathcal{L}T\|\beta\|} = C\int_0^T |\eta(\tau)|\,d\tau,$$

where $C = \|B\|e^{\mathcal{L}T\|\beta\|}$. The softmax prediction function in (3.1c) is Lipschitz continuous with constant that we denote by L_σ. We have

$$|\mathscr{C}(\tilde{x}) - \mathscr{C}(x)| \le L_\sigma|\tilde{h}(T) - h(T)| \le L\|\eta\|_{L^1([0,T];\mathbb{R}^n)}, \tag{3.7}$$

where $L = L_\sigma C$, as desired. ■

Theorem 3.5. *Consider a NAED classifier $\mathscr{C}\colon L^1([0,T];\mathbb{R}^n) \to \mathbb{R}^{|\mathcal{Y}|}$, equipped with a dictionary $\Xi\colon \mathbb{R}^m \to \mathbb{R}^d$ that is Lipschitz continuous with constant $\mathcal{L}$. Let W_t denote the Wiener process in $\mathbb{R}^d$. Consider a time signal $x(t)$ and a version corrupted by Gaussian white noise, $\tilde{x}(t) = x(t) + \eta(t)$, where $\eta(t)dt = dW_t$. Then $|\mathscr{C}(\tilde{x}) - \mathscr{C}(x)| \le L\sup_{0\le s\le T}|W_s|$ and $P(|\mathscr{C}(\tilde{x}) - \mathscr{C}(x)| \ge r) \le 2de^{-r^2/2dTL^2}$, with constant $L > 0$ defined in the proof.*

Proof. In the NAED method with dictionary Ξ, the unperturbed and perturbed hidden variables, h and $\tilde{h}$ satisfy

$$h(t) = h_0 + \int_0^t \beta\Xi(h(\tau))\,d\tau + B\int_0^t x(\tau)\,d\tau$$
$$\tilde{h}(t) = h_0 + \int_0^t \beta\Xi(\tilde{h}(\tau))\,d\tau + B\int_0^t x(\tau)\,d\tau + B\int_0^t dW_\tau$$

with $h(0) = \tilde{h}(0) = h_0$. Subtracting these equations, we first obtain

$$|\tilde{h}(t) - h(t)| = \left|\int_0^t \beta\left[\Xi(h(\tau)) - \Xi(\tilde{h}(\tau))\right]d\tau + BW_t\right|.$$

Let $\mathcal{L}$ denote the Lipschitz constant for the dictionary Ξ. We estimate

$$\begin{aligned} |\tilde{h}(t) - h(t)| &\leq \int_0^t \mathcal{L}\|\beta\| \, |h(\tau) - \tilde{h}(\tau)| \, d\tau + \|B\||W_t| \\ &\leq \int_0^t \mathcal{L}\|\beta\| \, |h(\tau) - \tilde{h}(\tau)| \, d\tau + \|B\| \sup_{0\leq s\leq t} |W_s| \end{aligned}$$

The continuity of W_t implies the continuity of $\sup_{0\leq s\leq t} |W_s|$. Note that $\sup_{0\leq s\leq t} |W_s|$ is non-decreasing. Hence Gronwall's inequality yields

$$|\tilde{h}(T) - h(T)| \leq \|B\| \sup_{0\leq s\leq T} |W_s| e^{\mathcal{L}T\|\beta\|} = C \sup_{0\leq s\leq T} |W_s|,$$

where $C = \|B\|e^{\mathcal{L}T\|\beta\|}$. We combine this with the Lipschitz bound on softmax:

$$|\mathscr{C}(\tilde{x}) - \mathscr{C}(x)| \leq L_\sigma |\tilde{h}(T) - h(T)| \leq L \sup_{0\leq s\leq T} |W_s|, \tag{3.8}$$

where $L = L_\sigma C$ as before.

The remaining estimates can be derived from the density computed in [19, §2.8A]; for clarity, we provide a self-contained treatment. Let B_t denote the Wiener process in $\mathbb{R}$, and let $\tau_z = \min\{t : B_t = z\}$, a first passage time. Then

$$P(B_t \geq z) = \underbrace{P(B_t \geq z \,|\, \tau_z \leq t)}_{\text{I}} P(\tau_z \leq t) + \underbrace{P(B_t \geq z \,|\, \tau_z > t)}_{\text{II}} P(\tau_z > t)$$

By symmetry of B_t, term I is 1/2; by continuity of B_t, term II is 0. Hence $P(\tau_z \leq t) = 2P(B_t \geq z) = \operatorname{erfc}(z(2t)^{-1/2})$ where erfc is the complementary error function. Now using the reflection principle, we have

$$\begin{aligned} P\left(\sup_{0\leq s\leq T} |B_s| \geq z\right) &\leq 2P\left(\sup_{0\leq s\leq T} B_s \geq z\right) \\ &\leq 2P(\tau_z \leq T) \\ &\leq 2\operatorname{erfc}(z(2T)^{-1/2}) \\ &\leq 2e^{-z^2/(2T)}. \end{aligned}$$

Let $W_{t,j}$ denote the j-th coordinate of W_t; each $W_{t,j}$ is an independent one-dimensional Wiener process. With $|w|_p$ denoting the p-norm of the vector $w \in \mathbb{R}^d$, we have $|w| = |w|_2 \leq d^{1/2}|w|_\infty$. Putting these facts together, we estimate

$$\begin{aligned} P\left(\sup_{0\leq s\leq T} |W_s| \geq z\right) &\leq P\left(\sup_{0\leq s\leq T} |W_s|_\infty \geq zd^{-1/2}\right) \\ &\leq P\left(\sup_{1\leq j\leq d} \sup_{0\leq s\leq T} |W_{s,j}| \geq zd^{-1/2}\right) \\ &\leq dP\left(\sup_{0\leq s\leq T} |B_s| \geq zd^{-1/2}\right) \\ &\leq 2de^{-z^2/(2dT)}. \end{aligned}$$

Combining this with (3.7) yields the conclusion of the theorem. ■

Theorem 3.4 and Theorem 3.5 can be further interpreted in terms of classification stability as follows. Suppose that for a given time series, x, the NAED classifier gives the estimate $\hat{y} = \mathscr{C}(x)$ (a probability vector). Further, suppose that $\max_i \mathscr{C}(x)_i$ is uniquely attained so that the distance between $\mathscr{C}(x)$ and the decision boundary is positive. Then there exists a positive constant, $\varepsilon > 0$, such that for any corrupted time signal $\tilde{x}(t) = x(t) + \eta(t)$ with $\|\eta\| \leq \varepsilon$ the two estimates $\mathscr{C}(x)$ and $\mathscr{C}(\tilde{x})$ have the same maximum component, and so the assigned class does not change for the corrupted time signal. Here, the corruption can be either deterministic (Theorem 3.4) or stochastic (Theorem 3.5).

3.5. Sparse NAED method. The main task in our proposed learning method is to find the right-hand side (rhs) of the underlying non-autonomous dynamical system in (3.2), where the rhs is assumed to be a linear combination of dictionary terms. Here we explore the idea of imposing sparsity on the dictionary coefficients, with the goal of finding a *simple representation* of the underlying dynamics. As in equation discovery methods, we are motivated by the observation that most equations describing physical phenomena involve only a few relevant terms so that the rhs is sparse in the set of all possible functions. Imposing this assumption, we learn a model that balances accuracy and parsimony. Additionally, the sparsity assumption on the dictionary coefficients helps to prevent overfitting on the training dataset, leading to a method that is more robust to noise. Moreover, by assuming sparsity, we also obtain more interpretable dynamical models.

To develop a practical method to promote sparsity in the dictionary coefficients, we adopt the idea of iterative thresholding from [4, 38]. The resulting algorithm is given in Algorithm 3.2. In Algorithm 3.2, entries of β with magnitude less than $\nu > 0$ are thresholded to zero. This procedure is repeated until β has converged. In general, increasing ν trades accuracy for sparsity. The optimal value of ν will thus depend on the problem and data at hand. In practice, we use cross-validation to tune the value of ν; we find that the convergence of the algorithm depends on the value of ν. We examine the effectiveness of Algorithm 3.2 using the perturbed dataset in subsection 4.3 and real dataset in subsection 4.4.

4. Computational experiments.

In this section, we demonstrate our proposed NAED method on a variety of datasets: synthetic datasets derived from dynamical systems and partial differential equations (subsection 4.2), a noisy synthetic dataset derived from a dynamical system (subsection 4.3), and UCR Archive Datasets (subsection 4.4). We demonstrate that our method is interpretable and attains results with accuracy comparable to or better than the RNN, LSTM, CFN and NCDE methods, using substantially fewer parameters. Next we describe details of our implementation; our source code is available online[1].

4.1. Implementation details. We implemented the NAED and sparse NAED methods, described in section 3, using TensorFlow; pseudocode for these two algorithms is given in Algorithms 3.1 and 3.2.

To solve the optimization problem, we used the mini-batched ADAM optimizer with gradient computed as in Theorem 3.3. For each epoch, we shuffled the data using the Tensorflow function `tf.random.shuffle` and split the training data into small batches that were used to compute the gradients and update the parameters. For the synthetic datasets, we used

[1] https://github.com/rkyoon12/NAED

Algorithm 3.2 Sparse NAED method for time signal classification. The modification to the NAED method Algorithm 3.1 is the threshold step.

```
Input: initial parameters, $\Theta = \{\beta, B, A, b\}$ and cut-off value, $\nu > 0$.
for epoch = 1, ..., $N_{epoch}$: do
    Shuffle data and create batches of size $N_{batch}$
    for each batch: do
        (Solve the forward ODE for $h_i$) For the current parameters $\beta$ and $B$, solve
        the forward ODE in (3.1), i.e., for each example $i \in [N_{batch}]$ and discrete times $t_k$,
        $k \in [K]$, find $h_i(t_k)$.
        (Make predictions) Assign predictions via (3.1c), i.e., $\hat{y}_i = \sigma(Ah_i(T) + b)$.
        (Solve the adjoint equation for $\lambda_i$) Use the hidden state at the final time $h_i(T)$,
        to compute the terminal condition and solve the backward ODE in (3.6) i.e. for
        each example $i \in [N_{batch}]$ and discrete times $t_k$, $k \in [K]$, find $\lambda_i(t_k)$.
        (Compute gradients) Using $h_i(t_k)$ and $\lambda_i(t_k)$, evaluate the gradient of the ob-
        jective function with respect to the parameters $\nabla_\Theta J$, as in (3.5).
        (Update parameters) Use a gradient-based optimization method, e.g., ADAM
        method or gradient descent, to update the parameters, $\Theta$.
        (Threshold step) We threshold the $\beta$ parameter values by setting
            $$\beta_{ij} \leftarrow \begin{cases} \beta_{ij}, & \text{if } |\beta_{ij}| \geq \nu \\ 0, & \text{if } |\beta_{ij}| < \nu \end{cases}.$$
```

batch size 800 and for the UCR datasets, we used batch size $200, 20, 50$ and 50 respectively. We used a learning rate $\in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$ and the report the results of the best performance.

The gradient computation requires us to solve both the *forward ODE* (3.1) for $h\colon [0,T] \to \mathbb{R}^m$ and the *adjoint ODE* (3.6) for $\lambda\colon [0,T] \to \mathbb{R}^m$. To approximate the solution of the forward and adjoint ODEs, we used the fourth-order Runge–Kutta (RK4) method, implemented via `tfs.integrate.odeint_fixed` in the `tensorflow_scientific` library. To approximate $x(t)$ at times t not in the sampled time series data, we use linear interpolation

$$x(t) \approx \frac{(t_{n+1} - t)x_n + (t - t_n)x_{n+1}}{t_{n+1} - t_n}, \quad t_n \leq t \leq t_{n+1}.$$

In all numerical examples we fixed the initial condition for the hidden state in (3.1b) to be $h_0 = 0$.

For each dataset, we train the NAED model several times for different dimensions, m, of the hidden state, largest degree of polynomials k, or the maximum multiplier of Fourier basis terms K. We report the results for several such models.

Initialization. As described at the end of subsection 3.3, the choice of dictionary functions and coefficients has a significant effect on the convergence of the method. In particular, large values of β can cause the solution of the forward ODE (3.1) to blow up in finite time. The time duration ε, as guaranteed by Theorem 3.2 for a bounded solution, is inversely proportional to the norm of β and B. Hence we initialize parameters to be small to guarantee a bounded solution to (3.1) until the final time T. Let $\mathcal{U}[r,s]$ denote the uniform distribution on the interval $[r,s]$. For either the linear polynomial dictionary or the Fourier basis dictionary, the Lipschitz constant for each dictionary function is approximately $\mathcal{L} \approx 1$, so we initialize the parameters $\beta, B, A \overset{iid}{\sim} \mathcal{U}[-1,1]$ and $b \overset{iid}{\sim} \mathcal{U}[0,1]$. When the dictionary involves higher-degree polynomials, we initialize $\beta, B \overset{iid}{\sim} \mathcal{U}[-0.1, 0.1]$ and $A, b \overset{iid}{\sim} \mathcal{U}[-1,1]$.

Competing Methods and Hyperparameters. We implemented the RNN and LSTM methods in TensorFlow, using `tf.keras.sequential`, `keras.layers.rnn` and `keras.layers.LSTM`. Using the description in [21], we developed our own implementation of the CFN method in TensorFlow. We trained the NCDE method using the published code [20]; we implemented this in PyTorch using the `torchcde` library. We trained these competing methods using the cross-entropy loss function, the ADAM optimization method, and the default initialization. The models were trained until convergence of the loss function. For the RNN, LSTM, CFN, and NCDE methods, we conducted extensive parameter sweeps to select hyperparameters such as network depth (number of layers) and width (number of units per layer). For each data set, and for each of these four methods, we report the best result that we found.

4.2. Synthetic datasets.

4.2.1. Forced harmonic oscillator. We consider a forced oscillator with position $u(t)$ satisfying

$$\ddot{u} + \gamma\dot{u} + \omega^2 u = x(t) \tag{4.1a}$$

$$u(0) = \dot{u}(0) = 0, \tag{4.1b}$$

where γ is the damping coefficient, ω^2 is the undamped angular frequency, and $x(t)$ is a specified forcing. To form the ground-truth labels, we record whether the position of $u(T)$ at the final time $t = T$ is positive or negative,

$$y = \begin{cases} (1,0) & u(T) > 0 \\ (0,1) & u(T) < 0 \end{cases}. \tag{4.2}$$

With the above framework, we generate a synthetic dataset as follows. Fix $K \in \mathbb{N}$, $\gamma > 0$, $\omega > 0$, and $T > 0$. For a forcing of the form $x(t) = \sum_{k=1}^{K} A_k \sin(\alpha_k t)$, $t \in [0,T]$, where A_k are randomly chosen amplitudes and α_k are randomly chosen forcing frequencies, we numerically solve (4.1) for $u(t)$, $t \in (0,T]$ and compute y via (4.2). We choose $K = 2$, $\gamma = 0.2$, $\omega = 1$, $T = 10$, $A_k \overset{iid}{\sim} \mathcal{N}(0,1)$, and $\alpha_k \overset{iid}{\sim} \mathcal{N}(0,1)$. In this paper, we use $\mathcal{N}(\mu, \sigma^2)$ to denote the normal distribution with mean μ and variance σ^2. The process is repeated $N = 10000$ times to create a dataset with 8000 training examples and 2000 test examples.

In Table 1, we tabulate the accuracy and number of trained parameters for various methods on this dataset. The total number of parameters for the NAED method is given by

$$\#\text{params} = \dim(\beta) + \dim(B) + \dim(A) + \dim(b) = d \times m + n \times m + m \times |\mathcal{Y}| + |\mathcal{Y}|.$$

Methods	Train	Test	# params
NAED Poly (2,1)	**0.9994**	**0.9905**	14
NAED Poly (3,1)	0.9831	0.9585	23
NAED Poly (4,1)	0.9800	0.9040	34
NAED Poly (2,2)	0.9817	0.9700	20
NAED Fourier (2,1)	0.9614	0.9670	26
NAED Fourier (2,2)	0.9365	0.9345	58
RNN (1,5)	0.9741	0.9715	41
LSTM (1,5)	0.9791	0.9750	146
CFN (7,5)	0.9113	0.9180	891
NCDE (32,32-1)	0.9919	0.9854	1221

Table 1

A comparison of the accuracy (training and test datasets) and number of parameters for various methods on the synthetic dataset based on the forced harmonic oscillator. In this and in subsequent tables, we use boldface to indicate methods with the highest accuracy. NAED refers to Algorithm 3.1 with no sparsity promotion. See subsection 4.2.1.

In the first column of Table 1, additional information about each method is summarized. For the NAED Method with Polynomial dictionary, the parenthetical numbers are (# of units in hidden layer, maximum degree of polynomial in dictionary). The first row block of Table 1 is for the NAED method while varying either the dimension of the hidden units or the maximum degree of the polynomial entries. We observe that the Polynomial dictionary with a two-dimensional hidden state and polynomials up to degree one produces the best accuracy. This model also has the smallest number of parameters of all methods tested. This result might be expected as it agrees with the ground-truth model (harmonic oscillator) up to conjugation by an orthogonal matrix. For the NAED method with the Fourier dictionary, the parenthetical numbers refer to (# of units in hidden layer, largest multiplier K) where the dictionary consists of Fourier terms with frequency $\omega = \frac{L}{K}, \ldots, L$. It is natural to choose $L = 10$ because we handle the hidden state h on the time interval $[0, 10]$. For the RNN, LSTM, and CFN methods, the parenthetical numbers represent (# of hidden layers, # of units). For the NCDE method, the parenthetical numbers represent (# of units, *width-depth* of neural network for vector field). As described in subsection 4.1, for the RNN, LSTM, CFN, and NCDE methods, we carried out repeated runs with different values of these hyperparameters, but we report only the hyperparameters that yield the best test accuracy. We observe that all methods performed remarkably well for this simple dataset.

We can visualize our model using phase portraits; examples are given in Figure 2. Here, the black arrows represent the autonomous part of the learned vector field, $h \mapsto \beta\Xi(h)$. Also plotted in color are solution trajectories, all of which begin at the origin. The final positions at $T = 10$ are indicated by a square. The class associated with each sample is indicated in the legend. The classification decision is made using the final state of a trajectory via the probability vector, $\hat{y} = \sigma\left(Ah(T) + b\right)$. Writing $Ah(T) + b = A\left(h(T) + A^{-1}b\right) = A\left(h(T) - h_0\right)$ where $h_0 = -A^{-1}b$, we see that the softmax function is being applied to the

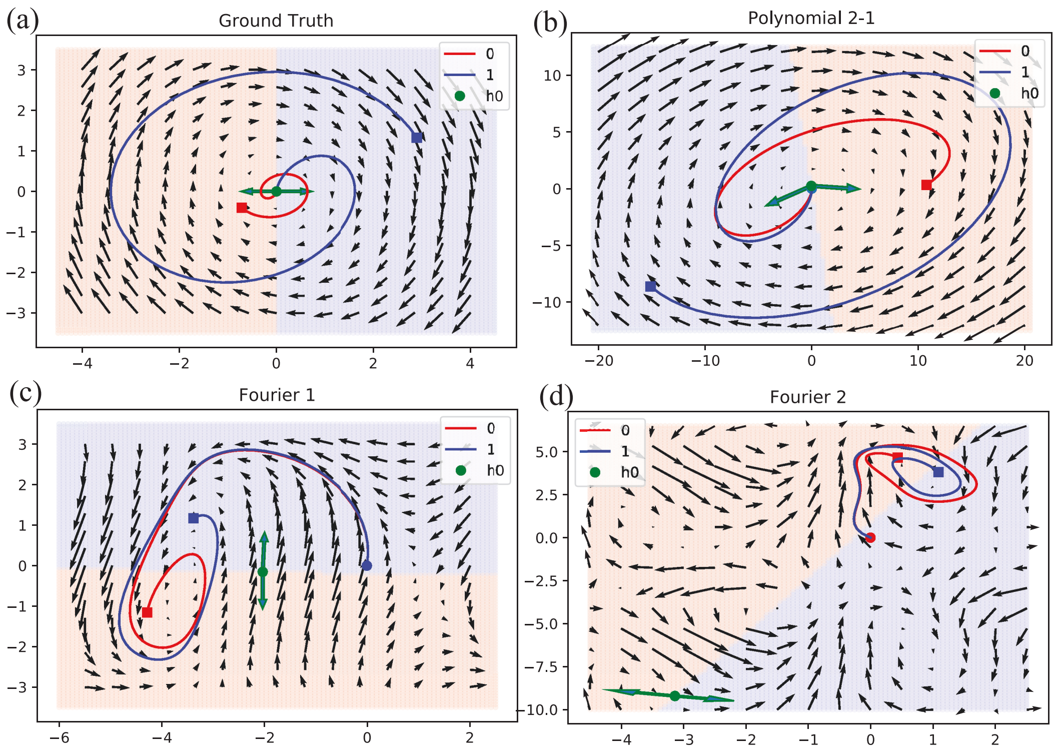

Figure 2. *In four subplots, labeled* **(a)**–**(d)**, *we plot the vector field* $h \mapsto \beta\Xi(h)$ *in* (3.1) *for different choices of dictionary* Ξ. *In each plot, two example solution trajectories are given (one for each class), vectors used for the decision are drawn, and the class regions are shaded in red and blue.* **(a)** *Polynomial dictionary with ground truth initialization.* **(b)** *Polynomial dictionary with random initialization.* **(c)** *Fourier dictionary with* $K = 1$. **(d)** *Fourier dictionary with* $K = 2$. *See subsection 4.2.1 for details.*

vector $\begin{pmatrix} a_1^t(h(T) - h_0) \\ a_2^t(h(T) - h_0) \end{pmatrix}$, where a_i is the i-th row of A. We can visualize this decision in Figure 2 as follows. We draw the two rows of A as green vectors. These vectors partition $\mathbb{R}^2$ into two regions; (each representing a class); we shade the region representing class 0 in red and class 1 in blue. In Figure 2, we observe that the the final states of the chosen trajectories belong to the correctly identified partition component.

We now remark on the identifiability of our model. Recall that a statistical model is said to be *identifiable* if the parameter values uniquely determine the probability distribution of the data. For an identifiable model, it is in principle possible to learn the ground-truth parameters used to construct the data. Also recall that the goal of our algorithm is not to learn the mapping $h \mapsto \beta\Xi(h)$, but rather the mapping $x \mapsto y$. Since only the solution of the forward ODE at the final time is used to make this prediction, the learned vector field can differ from the ground truth vector field; our model is not identifiable. If we consider Figure 2**(b)**, the learned vector field closely agrees with the ground truth vector field **(a)**, up to conjugation by an orthogonal matrix; the eigenvalues of $\beta\Xi(h)$ for in **(a)** are $-0.1 \pm i0.995$ which are close to the eigenvalues of $\beta\Xi(h)$ for **(b)**, given by $-0.1015 \pm i1.001$. However, the vector fields in **(c)** and **(d)** are seen to differ from **(a)** considerably. Nevertheless, for the example trajectories shown in each subplot, the class is correctly predicted (*e.g.*, red trajectory terminates in the red region). So, despite these differences in the vector fields, the classification accuracy for each of the four models is quite good (see (4.1)).

By using the adjoint method to train the NAED method, we avoid the exploding/vanishing gradient problem that often arises when training RNNs. To demonstrate this and also to show that the NAED method captures long-term dependencies in the data, we perform the following experiment. We again generate synthetic data using the forced harmonic oscillator (4.1) and assign the labels $\hat{y}$ using the position of $u(T)$. Here we take a larger final time, $T = 100$ (compared to $T = 10$ previously), while keeping the time step $dt = 0.1$ the same as before. The result is sequential data that is 10 times as long as that considered before. We then train the NAED, RNN, and NCDE methods on the data. With the same set of hyperparameters (entries of the dictionary for NAED and depth and width of networks for RNN/NCDE), we found that the NAED method still achieves similar accuracy as before; 0.9937/0.9890 (train/test), whereas the performance of other methods is degraded; the accuracy of RNN is 0.8313/0.8020, and the accuracy of NCDE is 0.5699/0.5299. To improve the performance of these methods, we can adjust the architecture of the models by increasing the depth and width of layers. Nevertheless, the best accuracy we found for the RNN and NCDE methods, respectively, is 0.8950/0.8888 and 0.6637/0.6399. These results indicate that the NAED method can compute gradients stably across long intervals of time and that it can model long-term dependencies.

4.2.2. Forced Van der Pol oscillator.

Consider the forced Van der Pol oscillator with position $u(t)$ satisfying

$$\ddot{u} - \mu(1 - u^2)\dot{u} + u = x(t) \tag{4.3a}$$

$$u(0) = \dot{u}(0) = 0 \tag{4.3b}$$

where $\mu = 0.3$ controls the strength of nonlinear damping. We choose the forcing $x(t)$ as in subsection 4.2.1 and, at time $T = 10$, we define the label y as in (4.2).

As shown in Table 2, the best accuracy for the forced Van der Pol dataset is obtained with the Fourier (2,2) dictionary. It is a remarkable result in that NAED only uses 58 parameters. On the other hand, the second best result trains roughly 20 times more numbers of parameters. Since the true system is nonlinear, it is not surprising to see strong performance from the Fourier dictionaries, which contain sums and products of trigonometric functions. Due to the presence of nonlinear polynomials in the Van der Pol system, we might expect that the best dictionary would be the Polynomial (2,3) dictionary. However, as discussed in subsection 3.3, the nonlinear entries in the dictionary cause the right-hand side of (3.1) to be only locally Lipschitz continuous, so that Theorem 3.2 can only guarantee a solution on a short time interval. Since the class prediction is made using (3.1c), *i.e.*, it depends on the hidden variable $h(t)$ at time $t = T$, premature blowup of solutions spoils the learning process. The first block in Table 2 shows that linear Polynomial dictionaries beat nonlinear ones. We obtained the best accuracy with a more complex Fourier dictionary; both Fourier dictionaries outperformed all polynomial dictionaries on this problem.

Method	Train	Test	# params
NAED Poly (2,1)	0.9030	0.854	14
NAED Poly (3,1)	0.9100	0.8675	23
NAED Poly (4,1)	0.8790	0.882	34
NAED Poly (2,2)	0.7458	0.765	20
NAED Poly (2,3)	0.8237	0.8215	28
NAED Fourier (2,1)	0.9045	0.8975	26
NAED Fourier (2,2)	**0.9830**	**0.9860**	58
RNN (5,7)	0.9729	0.9600	491
LSTM (1,5)	0.9603	0.9495	146
CFN (7,5)	0.9345	0.9350	1,177
NCDE (32, 32-1)	0.9821	0.9745	1,221

Table 2

A comparison of the accuracy and number of parameters for various methods on the forced Van der Pol synthetic dataset. NAED refers to Algorithm 3.1 with no sparsity promotion. See subsection 4.2.2.

4.2.3. Forced Lorenz system. Consider the forced nonlinear Lorenz system with positive parameters (σ, ρ, β):

$$\dot{u}_1 = \sigma(u_2 - u_3) + x(t) \tag{4.4a}$$
$$\dot{u}_2 = u_1(\rho - u_3) - u_2 \tag{4.4b}$$
$$\dot{u}_3 = u_1 u_2 - \beta u_3 \tag{4.4c}$$
$$u_1(0) = u_2(0) = u_3(0) = 1 \tag{4.4d}$$

The first coordinate is forced by $x(t) = 4\sum_{k=1}^{K} A_k \sin(\alpha_k t)$, $t \in [0, T]$, where $A_k \overset{iid}{\sim} \mathcal{N}(0, 1)$ and $a_k \overset{iid}{\sim} \mathcal{N}(0, 1)$. Using the position of $u_1(t)$ at the final time $T = 10$, we define the label y as in (4.2). To generate the synthetic data, we choose parameters $\sigma = 5, \beta = 1.3$ and $\rho = 10$. This dataset has a class imbalance: the training data consist of 6348 and 1652 instances in classes 0 and 1, respectively, and the test data contains 1566 and 434 instances in classes 0 and 1, respectively.

As shown in Table 3, the highest accuracy for different choices of dictionaries and parameters in the NAED method is obtained by Fourier (3,1). This result demonstrates that complex dictionary entries are required to capture the nonlinearity in the underlying dynamics. It is remarkable that the NAED methods produced comparable results to the other methods using far fewer parameters, although it does not exceeded the classification accuracy of the LSTM method.

4.2.4. Forced Lotka-Volterra equations. Consider the forced Lotka-Volterra system,

$$\dot{u}_1 = \alpha u_1 - \beta x(t) u_1 u_2 \tag{4.5a}$$
$$\dot{u}_2 = \delta x(t) u_1 u_2 - \gamma u_2, \tag{4.5b}$$

with initial condition $(u_1(0), u_2(0)) = (5, 4)$, $x(t) = \left(\sum_{k=1}^{K} A_k \sin(\alpha_k t)\right)^2 \geq 0$, and parameters $(\alpha, \beta, \delta, \gamma) = (0.8, 0.1, 0.01, 1.1)$. We sample $A_k, a_k \overset{iid}{\sim} \mathcal{N}(0, 0.5)$. After numerically

Method	Train	Test	# params
NAED Poly (2,1)	0.8388	0.8365	14
NAED Poly (3,1)	0.8252	0.8160	23
NAED Poly (4,1)	0.8321	0.8215	34
NAED Poly (2,2)	0.8522	0.847	20
NAED Poly (3,2)	0.8546	0.8535	41
NAED Fourier (2,1)	0.8861	0.8945	26
NAED Fourier (3,1)	**0.9051**	**0.9050**	92
NAED Fourier (2,2)	0.8517	0.8439	58
RNN (2,10)	0.7937	0.7799	341
LSTM (1,10)	0.9306	0.9359	491
CFN (2,10)	0.8080	0.7965	781
NCDE (16,16-1)	**0.9434**	**0.9369**	595

Table 3
A comparison of the accuracy and number of parameters for various methods on the synthetic dataset based on the forced Lorenz equation. NAED refers to Algorithm 3.1 with no sparsity promotion. See subsection 4.2.3.

solving up to time $T = 10$, we set the ground truth label y via the indicator function for $\operatorname{argmax}(u_1(T), u_2(T))$. Here $x(t)$ appears as a coefficient in the nonlinear terms. If we introduce the additional variable $u_3(t) = x(t)$, the forcing occurs linearly,

$$\begin{aligned} \dot{u}_1 &= \alpha u_1 - u_1 u_2 u_3 & u_1(0) &= 5 \\ \dot{u}_2 &= u_1 u_2 u_3 - \gamma u_2 & u_2(0) &= 4 \\ \dot{u}_3 &= \dot{x}(t) & u_3(0) &= x(0) = 0 \end{aligned}$$

This system suggests that we consider $\dot{x}(t)$ as the time series input data. Hence we train the model using, in turn, either $x(t)$ or $\dot{x}(t)$ as the input. We generate $\dot{x}(t)$ using the derivative of $x(t)$ computed by hand.

The first block of Table 4 shows results with $x(t)$ as input, while the second block shows results with $\dot{x}(t)$ as input. Comparing these two blocks in Table 4, we see that across all dictionaries and hyperparameters, the NAED method performs better with $\dot{x}(t)$ as input. We find that the NAED method with Fourier dictionary yields similar or better results than other methods regardless of whether $x(t)$ or $\dot{x}(t)$ is used as input.

4.2.5. Stochastic gated partial diffusion equation.

Consider the one-dimensional stochastic gated partial diffusion equation [22],

$$\begin{aligned} u_t(z,t) &= \kappa u_{zz}(z,t) & z &\in [0,1],\ t \in [0,1] \\ u_z(0,t) &= 0 & z &= 0 \\ x(t)u(1,t) + (1 - x(t))u_z(1,t) &= 0 & z &= 1 \\ u(z,0) &= u_0(z) & t &= 0. \end{aligned}$$

At $z = 0$, we impose the reflecting (Neumann) boundary condition. At $z = 1$, we impose the switching (time-dependent Robin) boundary condition, where for all $t \in [0,1]$, we have

Method	Train	Test	# params
NAED Poly (2,1)	0.8835	0.8860	14
NAED Poly (2,3)	0.8785	0.8835	28
NAED Fourier (2,1)	0.9600	0.9539	26
NAED Fourier (3,1)	**0.9805**	**0.9739**	92
RNN (5,32)	0.8192	0.8045	9,441
LSTM (1,47)	0.9614	0.9595	9,260
CFN (5,20)	0.9295	0.9184	9,081
NCDE (32,32-1)	**0.9838**	**0.9789**	1,221

Method	Train	Test	# params
NAED Poly (2,1)	0.9109	0.850	14
NAED Poly (3,3)	0.9538	0.9435	71
NAED Fourier (2,1)	**0.9737**	**0.9660**	26
NAED Fourier (3,1)	0.9536	0.9505	92
NAED Fourier (2,2)	0.9717	0.9670	58
RNN (5,30)	0.9256	0.9225	8,311
LSTM (2,20)	0.9684	0.9670	5,082
CFN (3,20)	0.9409	0.9275	5,001
NCDE (32,32-1)	**0.9786**	**0.9720**	1,221

Table 4

A comparison of the accuracy and number of parameters for various methods on the synthetic dataset based on the forced Lotka-Volterra equation. Each table presents results trained by input data $x(t)$ and $\dot{x}(t)$ respectively. NAED refers to Algorithm 3.1 with no sparsity promotion. See subsection 4.2.4.

$x(t) \in \{0,1\}$, a *switching function*. For the initial condition, we use an approximation to the Dirac delta $\delta(z-0.5)$, given by $u_0(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-0.5)^2}{2\sigma^2}\right)$, where $\sigma = 0.1$. The solution has an interpretation in terms of a particle experiencing Brownian motion on the interval. The probability of finding the particle at time t and position z is given by $u(z,t)$. The initial condition is interpreted as the particles all starting near $z = 0.5$. The boundary condition at $z = 1$ has the interpretation that when $x(t) = 1$ a particle leaves the interval when it reaches the boundary and when $x(t) = 0$ the particles are reflected. The proportion of particles remaining in the interval at time t, referred to as the *survival probability* is given by $S(t) = \int_0^1 u(z,t)\, dz$. For a given switching function $x(t)$, we assign a label y based on the survival probability at time $t = 1$;

$$y = \begin{cases} (1,0) & S(1) < \frac{1}{2} \\ (0,1) & S(1) \geq \frac{1}{2} \end{cases}. \tag{4.6}$$

The classification problem seeks the mapping from the switching function $x(t)$ to the binary class y.

We generate a synthetic dataset for this problem with 8000 training examples and 2000 testing examples as follows. To generate each switching function $x(t)$ we choose an integer, q, between zero and ten uniformly. We then randomly select q times in the interval $[0,1]$

Method	Train	Test	# params
NAED Poly (2,1)	0.9203	0.9155	14
NAED Fourier (2,1)	**0.9582**	**0.9570**	26
NAED Fourier (2,2)	0.9523	0.9515	58
RNN (3,10)	0.9550	0.9570	146
LSTM (1,5)	**0.9805**	**0.9799**	146
CFN (2,3)	0.9440	0.9309	88
NCDE (32,32-1)	0.9785	0.9750	1,221

Table 5

A comparison of the accuracy and number of parameters for various methods on the synthetic dataset based on the stochastic gated diffusion equation. NAED refers to Algorithm 3.1 with no sparsity promotion. See subsection 4.2.5.

and starting with $x(t) = 0$, we set $x(t)$ to alternate between 0 and 1 at these times. For each switching function, $x(t)$, we approximately solve the heat equation for $u(z, t)$ as follows. We apply a forward difference in time and a second-order central difference scheme for the space derivative. We use a spatial discretization size of $dx = 0.05$ and temporal step size of $dt = 0.01$. To obtain roughly balanced class sizes, we choose the diffusion coefficient to be $\kappa = 0.165$. For this choice of parameters, the CFL condition $\frac{\kappa\ (dt)}{(dx)^2} \approx 0.66 < 1$ is satisfied, so the numerical method is stable. The solution at time $t = 1$ is used to define the label y as in (4.6).

A comparison of the accuracy of various methods is given in Table 5. As shown in the first block of Table 5, the proposed NAED method works well on this dataset generated using a partial diffusion equation. Among the several choices of entries for the dictionary, we achieve the best accuracy with the Fourier (2-1) dictionary. The NAED method provides comparable accuracy to other methods with substantially fewer parameters.

4.3. Synthetic dataset with noise. In this section, we train the sparse NAED method (see subsection 3.5) and show the robustness of this method on a noisy dataset. To generate the noisy data, we contaminate the forced harmonic oscillator input/forcing $x(t)$ from subsection 4.2.1 with noise:

$$\tilde{x}(t) = x(t) + \eta(t), \qquad \eta(t) \overset{iid}{\sim} \mathcal{N}(0, 10^{-4})$$

where $\eta(t)$ is a Gaussian process, mutually independent for different t. Noise is added to the original data $x(t) \in [-5.8, 5.6]$ for $t \in [0, T]$.

For the noisy data, we apply the sparse NAED method within a cross-validation loop to select ν. For each value of $\nu \in \{0.01, 0.03, 0.05, 0.1, 0.5, 1\}$, and within each fold of 5-fold cross-validation, we train with Algorithm 3.2 until convergence. We then choose ν to minimize the cross-validation test error. The last column of the Table 6 records the number of non-zero entries in the trained β. As shown in each block of Table 6, the performance of the sparse NAED method tends to be slightly better than competing methods. In particular, the sparse Fourier (2-2) method achieves the best test error with substantially fewer parameters than competing RNN methods. In this synthetic example, the underlying dynamical system does possess a sparse representation.

Methods	Train	Test	# nnz params
NAED Poly (2,1)	0.7580	0.7505	6
Sparse NAED Poly (2,1)	0.7618	0.7605	3
NAED Fourier (2,1)	0.9192	0.9155	18
Sparse NAED Fourier (2,1)	0.9311	0.928	6
NAED Fourier (2,2)	0.9523	0.9515	50
Sparse NAED (2,2)	**0.9670**	**0.9645**	16
RNN (2,10)	0.9557	0.9530	341
LSTM (2,10)	0.9615	0.9595	1,342
CFN (2,10)	0.9230	0.9180	781
NCDE (16,16-1)	**0.9789**	**0.9674**	595

Table 6

A comparison of the accuracy and number of nonzero (nnz) parameters for various methods on the synthetic dataset based on the forced harmonic oscillator with noise. NAED refers to Algorithm 3.1 with no sparsity promotion, while Sparse NAED refers to Algorithm 3.2. See subsection 4.3.

4.4. UCR archive datasets. In this section, we compare NAED with four competing methods (RNN, LSTM, CFN, and NCDE) on four univariate time series datasets from the UCR archive [7]. Unlike the datasets considered above, these datasets do not arise through numerical solutions of differential equations. For all methods, we follow training, initialization, and hyperparameter selection procedures described in subsection 4.1. In particular, we have trained repeatedly with different choices of hyperparameters. For all methods considered, we report only the results corresponding to hyperparameters that maximize test accuracy.

For the general NAED method (Algorithm 3.1), hyperparameters relate to the dimension m of the hidden variable $h(t)$ and the size d of the dictionary $\Xi(h)$. For the sparse NAED method (Algorithm 3.2), we also search for an optimal value of ν, the thresholding parameter. For the RNN, LSTM, and CFN methods, we varied the depth (number of layers) and width (number of units per layer). For NCDE, we trained with either 32 or 64 hidden channels; the vector field is represented using a feedforward neural network with one hidden layer with either 64 or 128 units.

For each method, we report the best results obtained, the optimal set of hyperparameters and total number of model parameters in Table 7. In Figure 3, we show an example trajectory for each class and use colored partitions to denote the classification regions and decision boundaries.

The *Two Pattern* dataset is synthetically generated and has 1000 training and 4000 test samples. There are four balanced classes and the sequence length for all samples is 128. As recorded in Table 7, NAED achieves its highest accuracy when we use a Fourier (2,1) dictionary. Compared with other methods, the NAED method provides slightly lower accuracy but is still close to 100% on both train and test data.

The *Plane* dataset contains outlines of airplanes measured by a sensor. The classification problem is to distinguish the type of airplane where there are seven airplane shape classes: Mirage, Eurofighter, F-14 wings closed, F-14 wings opened, Harrier, F-22 and F-15. There are 105 instances in both the training and test sets, each having length 144. As presented in

	Dataset		RNN	LSTM	CFN	NCDE	NAED
UCR archive	**Two Patterns**	test	0.7630	**1.0000**	0.9900	0.8420	0.9760
	train/test : 1000/4000	train	0.7473	**1.0000**	1.0000	0.8330	0.9815
	4 classes	info	(5-24-5,428)	(1-35-5,324)	(3-20-5,064)	(64-128-17,030)	(2-1-32)
	Plane	test	0.7048	0.4762	0.4000	**0.8571**	0.7714
	train/test : 105/105	train	0.7429	0.4952	0.5524	**0.8095**	0.7523
	7 classes	info	(5-10-3,867)	(5-10-3,917)	(5-20-9,205)	(32-64-8,905)	(2-1-41)
	Kitchen Appliance	test	0.5973	0.6027	0.5760	0.5306	**0.6133**
	train/test : 375/375	train	0.6027	0.5813	0.5467	0.5040	**0.6053**
	3 classes	info	(5-10-993)	(5-10-3,873)	(2-5-228)	(64-64-8,645)	(2-1-29)
	Computer	test	0.5800	**0.6640**	0.6199	0.6520	0.6599
	train/test : 250/250	train	0.5960	0.6280	0.6199	**0.6800**	0.6200
	2 classes	info	(1-5-47)	(1-3-68)	(2-2-45)	(64-128-16,835)	(2-2-58)

Table 7

A comparison of the accuracy and number of parameters for various methods on four UCR archive datasets. For the Two Patterns and Plane datasets, NAED refers to Algorithm 3.1; for the Kitchen Appliance and Computer datasets, NAED refers to Algorithm 3.2 with iterative thresholding. See subsection 4.4.

Table 7, the NAED method with Fourier (2,1) dictionary surpasses the test accuracy of the RNN, LSTM, and CFN. It does this even with 100-200 times fewer parameters than these competing methods. The NCDE method is the best on this dataset; it has over 200 times more parameters than NAED. This dataset shows that NAED works well on a multiclass classification problem.

The *Kitchen Appliance* dataset is behavioral data recorded from 251 households and measured by a device in two-minute intervals over a month. Each series has length 720. This problems classifies how consumers use electricity within their home, so there are three classes: Kettle, Microwave and Toaster. This data contains 375 instances in the training and test sets. In Table 7, sparse NAED with a Fourier (2,1) dictionary returns the best accuracy on this dataset with only 29 parameters. Here, the cutoff value is set to $\nu = 0.03$ and two entries of β are dropped to zero.

In Figure 3(d), we observe that some of the trajectories appear to jump in the phase plot. Since the trajectory is a solution to a *forced* dynamical system, very strong or highly oscillatory forcing can cause this type of behavior.

The *Computer* dataset consists of 250 train and test instances for a consumer's electricity usage behavior in a home. Each sample consists of recordings made every two minutes over a month so that total length is 720. There are two classes: Desktop and Laptop. According to Table 7, the best accuracy is obtained by the LSTM method. Sparse NAED with Fourier (2,2) dictionary and cutoff value $\nu = 0.05$ nearly matches the LSTM's accuracy. The imposed sparsity condition replaces 16 entries in β with zero; consequently, the trained vector field is relatively simple and interpretable.

For the *Kitchen Appliance* dataset, NAED achieves the best test set results; for the remaining three datasets, NAED's parameter count is on average $>$ 200 times less than that of the method with the best test set performance. For the first three datasets considered in

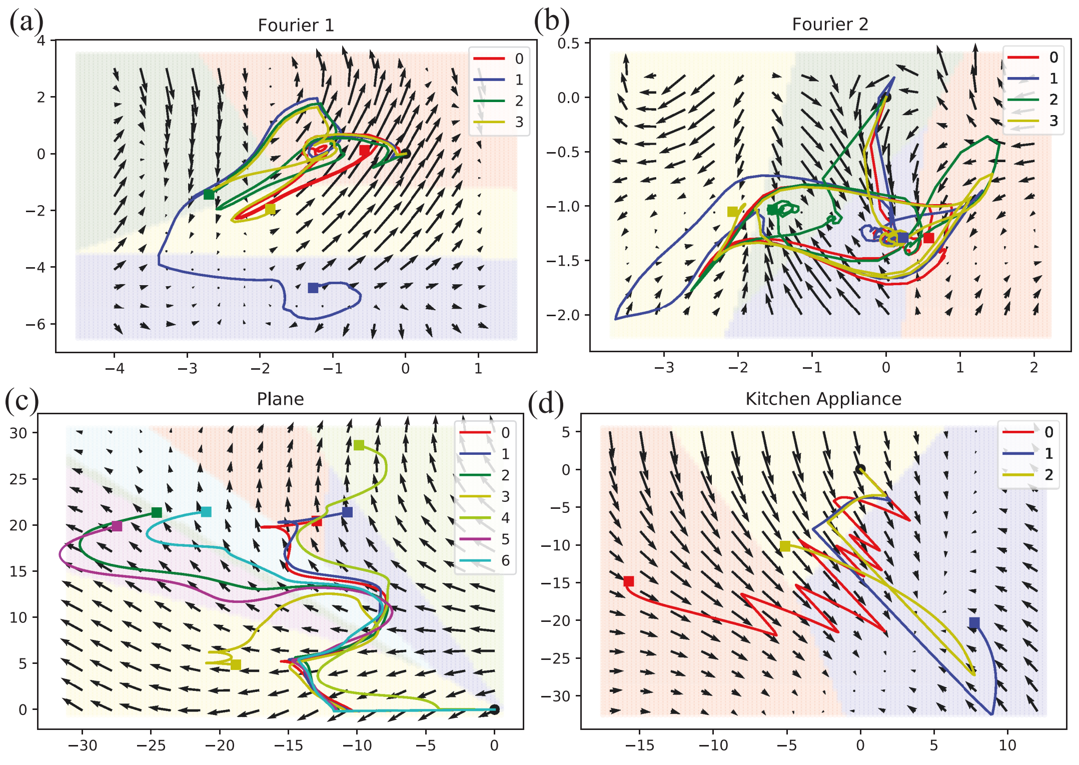

Figure 3. *In four subplots, labeled* **(a)**–**(d)***, we plot the vector field* $h \mapsto \beta\Xi(h)$ *in* (3.1) *trained on different UCR archive datasets. In each plot, example solution trajectories for each class are displayed and the classification partition is colored.* **(a)** *Two Patterns dataset using NAED with Fourier dictionary with* $K = 1$*.* **(b)** *Two Patterns dataset using NAED with Fourier dictionary with* $K = 2$*.* **(c)** *Plane dataset using NAED with Fourier dictionary with* $K = 1$*.* **(d)** *Kitchen Appliance dataset using sparse NAED with Fourier dictionary with* $K = 1$ *and* $\nu = 0.03$*.*

Table 7, NAED is the only method that achieves competitive test accuracies with a small number of parameters. For *Computer*, the parameter counts for NAED and LSTM are similar. Based on the RNN results here, we conjecture that NAED underfits this dataset; a more scalable implementation of the NAED method would enable us to explore larger values of the dimension of h and the largest Fourier multiplier K.

As we described in section 3, the NAED method learns a representation of the underlying vector field based on a prespecified dictionary. With polynomial or harmonic basis functions, these vector fields can be approximated using only a few terms. By promoting sparsity, Algorithm 3.2 can further enhance parsimony. As shown in experiments, competing methods require at least 2 times and up to 500 times the number of parameters required by NAED.

5. Discussion.

In this paper, we developed a framework for analyzing time signals based on non-autonomous dynamical systems. A time signal, $x(t)$, is interpreted as a forcing function for a dynamical system (3.1) that governs a time-evolving hidden variable, $h(t)$. As in equation discovery, the dynamical system is represented using a dictionary of prespecified candidate functions and the coefficients are learned from data. We refer to the resulting model as non-autonomous equation discovery (NAED). This framework is applied to the time signal classification problem, where the hidden variable, at a final time, $h(t = T)$, is used to make a prediction via the composition of the softmax function and an affine function. Using a cross-entropy loss function, we train the NAED model using a gradient based optimization method, where the gradients are efficiently computed using the adjoint method; see Theorem 3.3.

Through a variety of experiments—on both synthetic and real datasets—we demonstrated that the NAED method achieves accuracy that is comparable to RNN, LSTM, CFN and NCDE methods on binary and multi-class classification problems; see section 4. Note that [20] shows that NCDE itself outperforms other RNN architectures, including continuous-time/ODE-like GRU models [3, 18] and a method that merges an RNN with a neural ODE [28]. The NAED method generally requires far fewer parameters than neural network-based methods and the number of parameters can further be reduced by using a sparse version of the algorithm; see Algorithm 3.2. We also show in subsection 4.4 that sparsity improves the trainability of the method and its robustness to noise in the data. Finally, by construction, our method is interpretable using the theory of dynamical systems. For example, using phase plots, we can visualize the trajectories of the underlying dynamical system and how they navigate the decision boundaries between classes.

Since our model is built on dynamical systems, we can generate synthetic labelled data from a dynamical system and then pose the *inverse problem* of trying to recover the ground-truth labels from the data. For a synthetic dataset based on the forced harmonic oscillator (subsection 4.2.1), we showed that the NAED method for classification is not generally identifiable, *i.e.*, the method does not always recover the ground-truth parameters. However, in the case of a linear dictionary, we recover the ground-truth parameters up to conjugation by an orthogonal matrix.

There are a variety of natural future directions for this work. Since the NAED method is built on dynamical systems, we could use dynamical systems theory to further analyze a particular trained NAED model. For example, one could use stability theory to further sharpen and generalize the misclassification estimates in Theorem 3.4 and Theorem 3.5. To enhance the method's ability to deal with noisy time signals, one could combine the NAED method with filtering methods (*e.g.*, the Kalman filter). Since we interpret time signals as continuous objects and discretize within the method (the optimize-then-discretize approach), multi-scale methods could be used in training. A slight generalization of the model would be to let B in (3.2) be a parameterized operator, $B = \sum_{k=0}^{K} B_k \partial_t^k$, where $B_k \in \mathbb{R}^{m\times n}$ are unknown coefficients. In the forced Lotka-Volterra equations (subsection 4.2.4), we considered using as forcing either x or $\dot{x}$ and this generalization would avoid this. Another generalization would be to use the hidden state over the entire interval $[0, T]$, rather than just the final time; that is, rather than (3.1c), we could assign labels using an integral operator

$$\hat{y}_i = \sigma\left(\int_0^T A(t)h_i(t) + b(t)\ dt\right),$$

where $A\colon [0,T] \mapsto \mathbb{R}^{|\mathcal{Y}|\times m}$, $b\colon [0,T] \mapsto \mathbb{R}^{|\mathcal{Y}|}$, and $\sigma\colon \mathbb{R}^{|\mathcal{Y}|} \to \mathbb{R}^{|\mathcal{Y}|}$ is the softmax function. Finally, the NAED framework developed here could be applied to other time signal analysis tasks, such as prediction and forecasting, classification, segmentation, and denoising.

Acknowledgments. We would like to thank Dong Wang and Rebecca Hardenbrook for helpful discussions in the early stages of this work. We would also like to thank the anonymous referees for their helpful comments.

REFERENCES

[1] C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer International Publishing, 2018, https://books.google.com/books?id=achqDwAAQBAJ.

[2] R. D. Beer, *On the dynamics of small continuous-time recurrent neural networks*, Adapt. Behav., 3 (1995), pp. 469–509, https://doi.org/10.1177/105971239500300405.

[3] E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau, *Gru-ode-bayes: Continuous modeling of sporadically-observed time series*, in Advances in Neural Information Processing Systems, vol. 32, 2019, pp. 7377–7388.

[4] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proceedings of the National Academy of Sciences, 113 (2016), pp. 3932–3937, https://doi.org/10.1073/pnas.1517384113.

[5] B. Chang, M. Chen, E. Haber, and E. H. Chi, *Antisymmetricrnn: A dynamical system view on recurrent neural networks*, in 7th International Conference on Learning Representations, ICLR 2019, 2019.

[6] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural ordinary differential equations*, in Advances in Neural Information Processing Systems, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., vol. 31, 2018, pp. 6572–6583.

[7] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, *The ucr time series classification archive*, 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[8] E. Fanioudakis, M. Geismar, and I. Potamitis, *Mosquito wingbeat analysis and classification using deep learning*, in 2018 26th European Signal Processing Conference (EUSIPCO), IEEE, 2018, https://doi.org/10.23919/eusipco.2018.8553542.

[9] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, *Deep learning for time series classification: a review*, Data Mining and Knowledge Discovery, 33 (2019), pp. 917–963, https://doi.org/10.1007/s10618-019-00619-1.

[10] C. Finlay, J. Jacobsen, L. Nurbekyan, and A. M. Oberman, *How to train your neural ODE*, in Proceedings of the International Conference on Machine Learning, 2020, https://arxiv.org/abs/2002.02798, https://arxiv.org/abs/2002.02798.

[11] K. Funahashi and Y. Nakamura, *Approximation of dynamical systems by continuous time recurrent neural networks*, Neural Networks, 6 (1993), pp. 801–806.

[12] A. Ghosh, H. S. Behl, E. Dupont, P. H. S. Torr, and V. Namboodiri, *STEER : Simple temporal regularization for neural odes*, in Advances in Neural Information Processing Systems, vol. 33, 2020.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[14] E. Haber and L. Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2017), p. 014004, https://doi.org/10.1088/1361-6420/aa9a90.

[15] M. Habiba and B. A. Pearlmutter, *Neural ordinary differential equation based recurrent neural network model*, CoRR, abs/2005.09807 (2020), https://arxiv.org/abs/2005.09807, https://arxiv.org/abs/2005.09807.

[16] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation, 9 (1997), pp. 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.

[17] J. J. Hopfield, *Neurons with graded response have collective computational properties like those of two-state neurons*, Proceedings of the National Academy of Sciences, 81 (1984), pp. 3088–3092, https://doi.org/10.1073/pnas.81.10.3088.

[18] I. D. Jordan, P. A. Sokół, and I. M. Park, *Gated recurrent units viewed through the lens of continuous time dynamical systems*, CoRR, abs/1906.01005 (2019), http://arxiv.org/abs/1906.01005, https://arxiv.org/abs/1906.01005.

[19] I. Karatzas and S. E. Shreve, *Brownian Motion and Stochastic Calculus*, Springer, 1991.

[20] P. Kidger, J. Morrill, J. Foster, and T. J. Lyons, *Neural controlled differential equations for irregular time series*, in Advances in Neural Information Processing Systems, vol. 33, 2020.

[21] T. Laurent and J. von Brecht, *A recurrent neural network without chaos*, in 5th International Conference on Learning Representations, ICLR 2017, 2017.

[22] S. D. Lawley, *Blowup from randomly switching between stable boundary conditions for the heat equation*, Communications in Mathematical Sciences, 16 (2018), pp. 1133–1156, https://doi.org/10.4310/cms.2018.v16.n4.a9, https://doi.org/10.4310%2Fcms.2018.v16.n4.a9.

[23] K. Ott, P. Katiyar, P. Hennig, and M. Tiemann, *When are neural ODE solutions proper odes?*, CoRR, abs/2007.15386 (2020), https://arxiv.org/abs/2007.15386, https://arxiv.org/abs/2007.15386.

[24] R. Pascanu, Ç. Gülçehre, K. Cho, and Y. Bengio, *How to construct deep recurrent neural networks*, in 2nd International Conference on Learning Representations, ICLR 2014, Y. Bengio and Y. LeCun, eds., 2014.

[25] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*, in Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, eds., vol. 28, Atlanta, Georgia, USA, 2013, PMLR, pp. 1310–1318, http://proceedings.mlr.press/v28/pascanu13.html.

[26] M. M. Poulton, *Neural networks as an intelligence amplification tool: A review of applications*, Geophysics, 67 (2002), pp. 979–993.

[27] A. Quaglino, M. Gallieri, J. Masci, and J. Koutník, *SNODE: spectral discretization of neural odes for system identification*, in 8th International Conference on Learning Representations, ICLR 2020, 2020.

[28] Y. Rubanova, R. T. Q. Chen, and D. K. Duvenaud, *Latent ordinary differential equations for irregularly-sampled time series*, in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, eds., vol. 32, 2019, pp. 5320–5330.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-propagating Errors*, Nature, 323 (1986), pp. 533–536.

[30] T. C. Sideris, *Ordinary Differential Equations and Dynamical Systems*, Springer, 2013, https://doi.org/10.2991/978-94-6239-021-8.

[31] H. Siegelmann and E. Sontag, *Turing computability with neural nets*, Applied Mathematics Letters, 4 (1991), pp. 77–80.

[32] H. T. Siegelmann and E. D. Sontag, *On the computational power of neural nets*, Journal of Computer and Systems Sciences, 50 (1995), pp. 132–150.

[33] E. B. M. Tamil, N. H. Kamarudin, R. Salleh, and A. M. Tamil, *A review on feature extraction & classification techniques for biosignal processing (part i: Electrocardiogram)*, in IFMBE Proceedings, Springer Berlin Heidelberg, 2008, pp. 107–112, https://doi.org/10.1007/978-3-540-69139-6_31.

[34] E. M. Tamil, N. S. Bashar, M. Y. I. Idris, and A. M. Tamil, *A review on feature extraction & classification techniques for biosignal processing (part III: Electromyogram)*, in IFMBE Proceedings, Springer Berlin Heidelberg, 2008, pp. 117–121, https://doi.org/10.1007/978-3-540-69139-6_33, https://doi.org/10.1007%2F978-3-540-69139-6_33.

[35] E. M. Tamil, H. M. Radzi, M. Y. I. Idris, and A. M. Tamil, *A review on feature extraction & classification techniques for biosignal processing (part II: Electroencephalography)*, in IFMBE Proceedings, Springer Berlin Heidelberg, 2008, pp. 113–116, https://doi.org/10.1007/978-3-540-69139-6_32.

[36] G. Tzanetakis and P. Cook, *Musical genre classification of audio signals*, IEEE Transactions on Speech and Audio Processing, 10 (2002), pp. 293–302, https://doi.org/10.1109/tsa.2002.800560.

[37] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, *Convolutional neural networks for human activity recognition using mobile sensors*, in 6th International Conference on Mobile Computing, Applications and Services, 2014, pp. 197–205, https://doi.org/10.4108/icst.mobicase.2014.257786.

[38] L. Zhang and H. Schaeffer, *On the convergence of the SINDy algorithm*, Multiscale Modeling & Simulation, 17 (2019), pp. 948–972, https://doi.org/10.1137/18m1189828.

This manuscript is for review purposes only.