nature machine intelligence

Article

https://doi.org/10.1038/s42256-022-00575-4

Data-driven discovery of intrinsic dynamics

In the format provided by the authors and unedited

CONTENTS

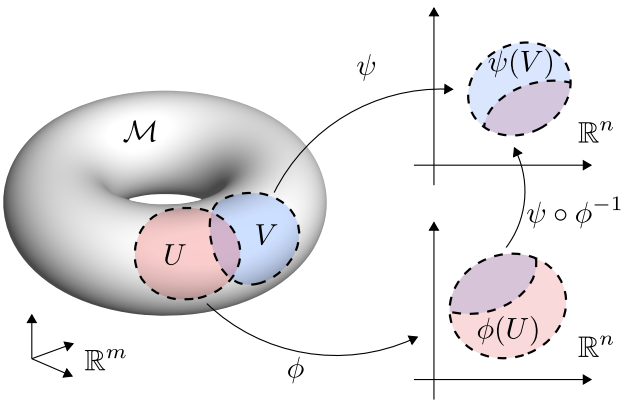

FIG. 1. Two charts, (U, ϕ) and (V, ψ), of an n-dimensional manifold, $\mathcal{M}$, embedded in $\mathbb{R}^m$, and a transition map, $\psi \circ \phi^{-1}$.

I. A PRIMER ON MANIFOLDS

The word "manifold" evokes an image of a curved surface embedded in a higher-dimensional Euclidean space; this is the relevant scenario for us, in which case our manifolds are, strictly speaking, submanifolds of $\mathbb{R}^m$. A manifold $\mathcal{M}$ is a set that is locally Euclidean, meaning that every point on the manifold has an open neighbourhood that can be mapped back and forth from and to a Euclidean space $\mathbb{R}^n$; the map and its inverse are continuous (i.e., the map is a homeomorphism), and $n \leq m$ is called the dimension of the manifold, which we called the intrinsic dimensionality earlier. Informally, one may think of breaking a manifold into overlapping patches such that each patch can be flattened. This representation is sketched in figure 1, and is formalized by the concept of charts.

A chart is a pair (U, ϕ), where the coordinate domain U is an open subset of the manifold, and the coordinate map ϕ is a homeomorphism mapping U to $\phi(U) \subseteq \mathbb{R}^n$. A point $p \in U$ is mapped to $\phi(p)$, called the local coordinates on U. In general, multiple charts are needed to cover the manifold, and each point on the manifold must belong to the coordinate domain of at least one chart. An atlas is a collection of charts whose coordinate domains cover the manifold, and forms a working representation of a manifold.

To switch between local coordinates, we use transition maps. Given two charts, (U, ϕ) and (V, ψ), whose coordinate domains intersect, the transition map from ϕ to ψ is given by $\psi \circ \phi^{-1} : \phi(U \cap V) \to \psi(U \cap V)$; it is also a homeomorphism. We refer the interested reader to [16] for more details.

A simple circle in the plane elucidates why charts are useful. A circle is a one-dimensional manifold, so we should be able to continuously map back and forth between the (x, y) coordinates on a circle and $\mathbb{R}$. That is, we should be able to find a one-dimensional latent representation of the circle. But this can only be done locally, not globally, since the circle closes on itself. Any attempt to produce a global one-dimensional latent representation of the circle will result in a discontinuity or a map that is not one-to-one. Instead, we must break the circle into at least two overlapping arcs, each of which can be mapped homeomorphically to $\mathbb{R}$. That is, we must use at least two charts.

Popular manifold learning methods, however, attempt to produce a global latent representation. Thus, they will be unable to reduce even the humble circle to one dimension and reconstruct the circle from the one-dimensional representation, as we demonstrate in Section II. This problem is due to the geometry of the circle; it is independent of the amount of training data, method, hyperparameters, etc., and is emblematic of the limitations of global manifold learning methods.

It is worth pointing out that methods that attempt to produce a global coordinate system, which we may think of as single-chart methods, are still able to achieve some degree of dimension reduction without loss of information, though generally not down to the intrinsic dimension. The strong Whitney embedding theorem states that any smooth real n-dimensional manifold can be smoothly embedded in $\mathbb{R}^{2n}$ [16, 31]. This means that a single-chart method can theoretically reduce the dimension to $2n$ in the worst case, compared to n for a multi-chart method. For example, a single-chart autoencoder would be able to reduce a circle in three-dimensional (x, y, z) space to two dimensions, but not to its intrinsic dimension of one.

We note that a limited set of work on manifold learning has recognized the concepts of charts and atlases [2, 22, 26]. Brand [2] was the first to explicitly mention charts, but uses them to form a global parameterization of a manifold, leading to the limitations of global manifold learning methods. More recent methods follow the formalism of charts and atlases more closely [22, 26], but would falter in the context of dynamics: Pitelis *et al.* [22] use linear approximations for the coordinate maps, which would lead to discontinuities when transitioning between charts; similarly, the method of Schonsheck *et al.* [26] may work, but it has not been designed or used for dynamics.

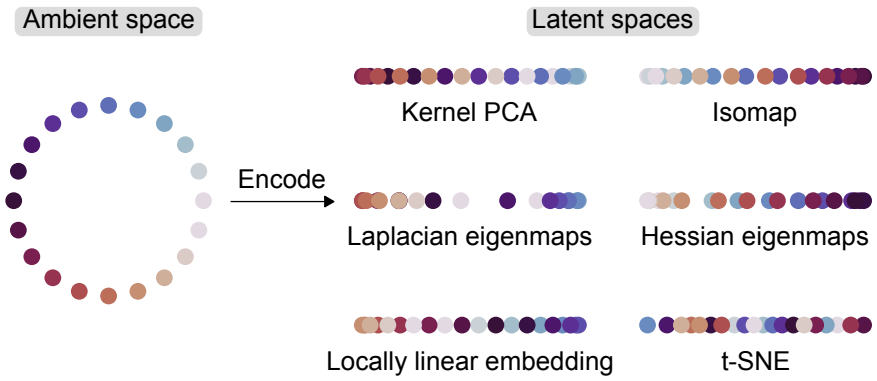

FIG. 2. One-dimensional latent representations of points on a circle produced by several manifold learning methods.

II. LIMITATIONS OF GLOBAL MANIFOLD LEARNING METHODS

As discussed in Section I, global manifold learning methods can produce discontinuities or coordinate maps that are not one-to-one. Here we demonstrate this effect for a suite of manifold learning methods: kernel principal component analysis (PCA) [25], Isomap [27], Laplacian eigenmaps [1], Hessian eigenmaps [7], locally linear embedding [23], and t-SNE [28]. We use the methods as implemented in the Scikit-learn library [21].

In figure 2, we show the one-dimensional latent representations of 20 evenly spaced points on the unit circle produced by applying the above manifold learning methods to the two-dimensional Cartesian coordinates of the points. In every case, points of dissimilar colours are mixed together. As discussed in the main text, this happens because the coordinate maps are not one-to-one, mapping different parts of the circle to the same point in the latent spaces. Intuitively, these manifold learning methods reduce the dimension of the data from two to one by essentially collapsing the circle to a dimension of one. Since the circle is a closed object, it collapses onto itself, leading to the observed effect.

Note that the effect shown in figure 2 is purely due to the geometry of the circle, independent of the values of the hyperparameters of each method (which we verified). We have chosen values for the hyperparameters that produce visually clear results. For kernel PCA, we used a radial basis function kernel with $\gamma = 10$. For Isomap, we used two neighbours. For locally linear embedding, we used two neighbours. For Laplacian eigenmaps, we used four neighbours. For Hessian eigenmaps, we used five neighbours. For t-SNE, we set perplexity equal to five. All other parameters were set to their default values. For meanings of these

parameters, we refer the reader to the documentation of the Scikit-learn library.

III. PSEUDOCODE

This section contains pseudocode describing how to learn an atlas of charts and the corresponding dynamics from data (Algorithm 1), and how to evolve an initial condition forward in time using the learned atlas and dynamics (Algorithm 2). The notation is the same as in the main text.

Algorithm 1 Train an atlas of charts and corresponding dynamics

Inputs: data $\{(x_i, x_i')\}_{i=1}^N$, number of charts k, encoding dimension n, number of nearest neighbours K, cluster expansion amount p

Create K-nearest neighbours graph for data $\{(x_i, x_i')\}_{i=1}^N$

Create k-means clustering of data $\{(x_i, x_i')\}_{i=1}^N$. Denote cluster centroids by $\{x_\alpha\}_{\alpha=1}^k$.

Expand each cluster p nodes along the K-nearest neighbours graph. Now some of the data belong to multiple clusters. ▷ Overlapping clusters implicitly define coordinate domains

for $\alpha = 1, \ldots, k$ **do**

    Train autoencoder with bottleneck dimension n on data $\{(x_i, x_i')\}_{i=1}^N$ that belong to cluster α. Denote the encoder by $\hat{\phi}_\alpha$ and the decoder by $\hat{\phi}_\alpha^{-1}$. ▷ Yields the coordinate map and its approximate inverse for chart α

    Train neural network to map from $\{\hat{\phi}_\alpha(x_i) \,\forall\, i \in \mathcal{I}_\alpha\}$ to $\{\hat{\phi}_\alpha(x_i') \,\forall\, i \in \mathcal{I}_\alpha\}$, where $\mathcal{I}_\alpha = \{i \in 1, \ldots, N \,|\, x_i, x_i' \in \text{cluster } \alpha\}$. Denote this map by $\hat{G}_\alpha$. ▷ $\hat{G}_\alpha$ gives the dynamics in chart α's local coordinates

end for

Algorithm 2 Evolve an initial condition forward in time

Inputs: initial condition x_0, number of steps T to evolve forward in time
$\alpha^* = \arg\min_\alpha \|x_0 - x_\alpha\|$ ▷ x_0 belongs to cluster α^*
$y = \hat{\phi}_{\alpha^*}(x_0)$ ▷ Map x_0 into chart α's local coordinates
for $j = 1, \ldots, T$ **do**
    $y \leftarrow \hat{G}_{\alpha^*}(y)$ ▷ Map forward one step in time
    $i^* = \arg\min_{i \in \mathcal{I}_{\alpha^*}} \|y - \hat{\phi}_{\alpha^*}(x_i)\|$ ▷ Find nearest training point in chart α^*
    if x_{i^*} is an interior point of cluster $\beta \neq \alpha^*$ **then**
        $y \leftarrow \hat{\phi}_\beta(\hat{\phi}_{\alpha^*}^{-1}(y))$ ▷ Transition from chart α^* to chart β
        $\alpha^* \leftarrow \beta$
    end if
end for

IV. ADDITIONAL DETAILS OF THE METHOD

A. Smooth transitions

For the learned dynamics to be accurate, we require smooth chart transitions. Let x be a point in the intersection of the coordinate domains of charts α and $\beta \neq \alpha$. Let ϕ_α and ϕ_β denote the coordinate maps that we learn, and let ϕ_α^{inv} and ϕ_β^{inv} denote their approximate inverses that we learn. In order for our model to have smooth dynamics across transitions between charts, we need $\phi_\alpha^{inv}(\phi_\alpha(x))$ and $\phi_\beta^{inv}(\phi_\beta(x))$ to be (nearly) equal for all such x. These are simply the reconstructions of x by the two charts from x's local coordinates. If they differ significantly, then there will be a discontinuity in the dynamics across transitions. Mathematically, we want to minimize

$$\|\phi_\alpha^{inv}(\phi_\alpha(x)) - \phi_\beta^{inv}(\phi_\beta(x))\| = \|[\phi_\alpha^{inv}(\phi_\alpha(x)) - x] - [\phi_\beta^{inv}(\phi_\beta(x)) - x]\|. \tag{1}$$

The first term in brackets is chart α's reconstruction error, and the second term is chart β's reconstruction error. By the triangle inequality,

$$\|[\phi_\alpha^{inv}(\phi_\alpha(x)) - x] - [\phi_\beta^{inv}(\phi_\beta(x)) - x]\| \leq \|\phi_\alpha^{inv}(\phi_\alpha(x)) - x\| + \|\phi_\beta^{inv}(\phi_\beta(x)) - x\|. \tag{2}$$

The difference between the two charts' reconstructions of x is upper-bounded by the sum of each chart's reconstruction error. We can indirectly minimize the difference in the re-

constructions by minimizing each chart's reconstruction error. This argument generalizes to multiple points and more than two charts.

By using the triangle inequality, we have decoupled the two charts. We can indirectly minimize the difference in the reconstructions while training our autoencoders separately simply by weighting training data that belong to multiple charts more heavily in each autoencoder's reconstruction loss function. Decoupling the charts allows for trivial parallelization of the training process.

For the results shown in this work, we have not weighted data that belong to multiple charts' coordinate domains more heavily in the autoencoder reconstruction losses. We experimented with doing so, but did not see appreciable effects until the weights were so large that the autoencoders' reconstructions of data belonging to only a single coordinate domain were poor. Nevertheless, this may be an important consideration in future work.

B. Time complexity

The time complexity to map a state forward one time step or to transition from one set of local coordinates to another is the time complexity of a forward pass of the associated neural network(s). This depends on the architectures of the neural networks, but a forward pass of a neural network is typically fast and would be necessary for a traditional single-chart model as well. The major expense comes from searching for a nearest neighbour. Using a k-d tree, the time complexity of a nearest neighbour search is $O(\log_2(N))$ for a constant dimension of the search space [8]. However, the performance degrades rapidly with the dimension of the search space due to the curse of dimensionality, approaching the $O(dN)$ time complexity of brute search in a d-dimensional space once $d \gtrsim 10$ [30]. Fortunately, we only perform a single nearest neighbour search in the high-dimensional ambient space (when assigning the initial condition to a chart), and all subsequent nearest neighbour searches are performed in the local coordinates of the charts. Moreover, if using k-means to construct the coordinate domains, we need only search over the cluster centroids when assigning the initial condition to a chart, drastically reducing the cost. More efficient data structures, such as the ball tree, can be used if the dimensionality is still high [18, 20].

C. Robustness to noise

The method is robust to moderate levels of noise due to the dimensionality reduction step, which filters noise [2]. Robustness to noise has been observed in other work on learning low-dimensional dynamical models directly from data [5, 29]. Even linear dimension reduction is widely used to filter noise [10]. One may wonder about the effects of noise on the overlap regions between charts. Even with noise, the representations are all smooth, so we can smoothly transition between charts. Since the dimension reduction step yields smooth representations, the downstream task of learning dynamics in the low-dimensional space is stable.

As the level of noise increases, accuracy decreases [5]. However, predictions of the state of a system will still be stable because our model enforces the structure of a manifold. That is, even though predictions may be inaccurate with high noise, they will still live on a low-dimensional manifold because we impose such structure.

V. COMPARISON TO SINGLE-CHART METHODS

As demonstrated in the main text and previous section, global—that is, single-chart—methods are unable to produce accurate models of intrinsic dimensionality. A contemporaneous work has proposed a method to learn dynamical models of intrinsic dimensionality [5]; it is a one-chart method. That work and ours both use the reaction-diffusion system as an example, so we compare the results of the two methods here. We thank the authors of [5] for making their code and data available online[1].

Using their code and data[2], we learned a two-dimensional model for the reaction-diffusion system (as done in the original work), as well as a one-dimensional model, only changing the dimension of the latent space. Note that the data consist of 128×128 image frames from a movie. We found that the two-dimensional model produced trajectories nearly indistinguishable from the ground truth test data, but the one-dimensional model was inaccurate, producing sharp jumps. We then used our multi-chart method—using the same architecture, training procedure, and training data as in [5], with three charts—to learn a one-dimensional model, and found it to produce trajectories nearly indistinguishable from the ground truth.

[1] https://github.com/BoyuanChen/neural-state-variables

[2] We used the second half of their data, by which time the system settles on a limit cycle.

The mean squared errors of the trajectories produced by the three models are shown in figure 3A. Supplementary Movie M1 compares all three models to the ground truth.

Pixel-wise errors better represent the difficulties that a one-chart one-dimensional model encounters. In figure 3B, we show the intensity of the green channel (normalized to be between zero and one) of one of the pixels during the first 100 time steps of a trajectory. The single-chart one-dimensional model is inaccurate, producing wild jumps, whereas our multi-chart one-dimensional model is accurate. Four consecutive frames produced by the ground truth dynamics and the single-chart one-dimensional model, occurring near the first sharp jump in figure 3B, are shown in figures 3C and D, respectively. For ease of comparison, we also show skeletal images below each frame, produced by only colouring pixels of a certain intensity. Pixels have been coloured red in regions that differ significantly from the ground truth results. The single-chart two-dimensional model and multi-chart one-dimensional model produce results that are nearly indistinguishable from the ground truth.

The work of [4] also uses the same example, producing a two-dimensional model. Their method is also a single-chart method. Clearly, multiple charts are generally needed to produce accurate dynamical models of intrinsic dimensionality. We believe our framework can be fruitfully combined with the works of [4] and [5] to produce even more accurate and interpretable models that are truly of intrinsic dimensionality.

VI. ADDITIONAL EXAMPLES

Here, we demonstrate the method on additional examples not included in the main text.

A. Periodic dynamics on a torus

Consider a particle moving along the surface of a torus. The particle travels at constant speeds in the poloidal and toroidal directions, three times as fast in the poloidal direction as in the toroidal direction, generating a periodic orbit. The data live on a one-dimensional submanifold of $\mathbb{R}^3$, shown in figure 4A.

We place the data into three clusters and follow our procedure to create an atlas of three charts and a local low-dimensional dynamical model for each chart (figure 4A–C). The resulting global dynamical model is excellent. In figure 4D, we show that the learned

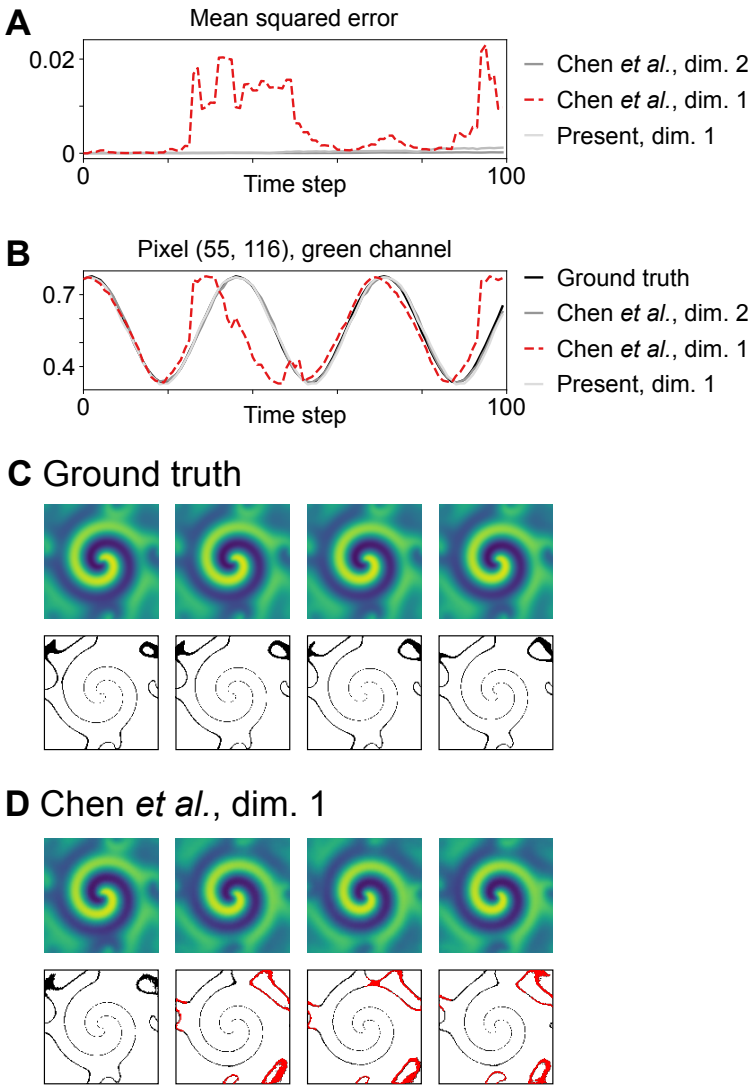

FIG. 3. Comparison between ground truth and various models for a reaction-diffusion system. (A) Mean squared error as a function of time. (B) Intensity of the green channel of pixel (55, 116). Only the multi-chart method proposed in the present work produces an accurate one-dimensional model. (C) Four consecutive snapshots from the ground-truth trajectory around 25 time steps from the initial condition. (D) Corresponding snapshots from a one-chart one-dimensional model using the method described in [5]. The skeletal pictures highlight features of the trajectory and are coloured red when differing significantly from the ground truth.

dynamics (coloured markers) match the true dynamics (grey curves) closely. Note especially the seamless transitions between charts.

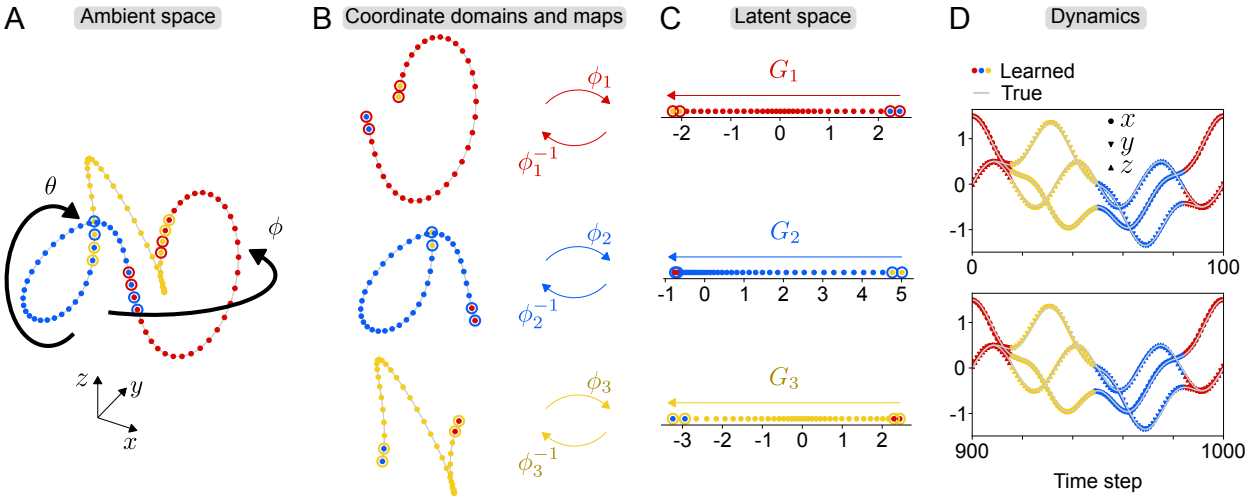

FIG. 4. (A) Data from a periodic orbit on the surface of a torus. (B) The learned coordinate domains. (C) The learned latent spaces/local coordinates. (D) Comparison between the learned dynamics and true dynamics. The first (top) and tenth (bottom) cycles of motion are well predicted.

B. Kuramoto-Sivashinsky beating dynamics

Our next example comes from the Kuramoto-Sivashinsky (K-S) partial differential equation,

$$u_t + uu_x + u_{xx} + \nu u_{xxxx} = 0, \tag{3}$$

for $0 \leq x \leq 2\pi$, with periodic boundary conditions. We set $\nu = \frac{16}{337}$ [24] and compute a numerical solution on a grid of 64 points. After transients have died out, we obtain a beating standing wave. The data live on a one-dimensional submanifold of $\mathbb{R}^{64}$, shown in figure 5A.

We place the data into three clusters and follow our procedure to create an atlas of three charts and a local low-dimensional dynamical model for each chart (figure 5A–C). Figure 5B shows the reconstruction of the data from their local coordinates, demonstrating excellent reconstruction.

In figure 5D, we show a trajectory that results from evolving an initial condition forward in time under our learned dynamical model. The transitions between charts are seamless and the trajectory is indistinguishable from the data. This example demonstrates that our method works in higher dimensions, producing a minimal one-dimensional dynamical model for nominally 64-dimensional data.

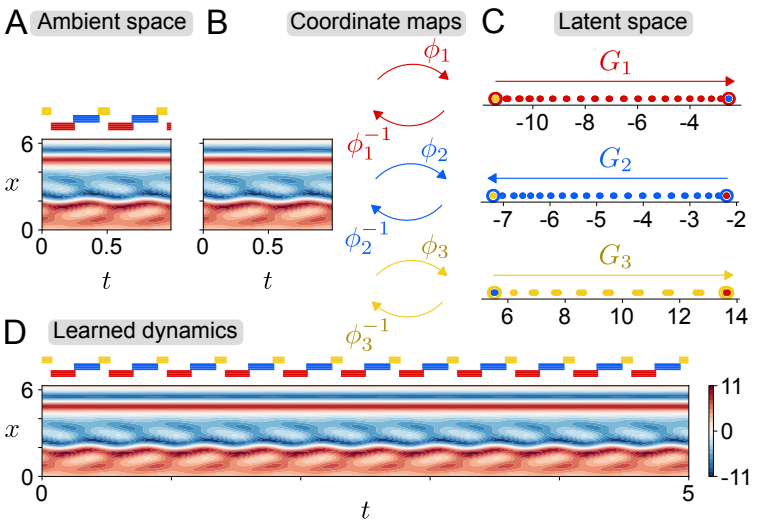

FIG. 5. (A) Space-time plot of data. The coloured bars show which learned coordinate domain the state is in at each instant in time. (B) Reconstruction of data from the learned latent spaces/local coordinates. (C) The learned latent spaces/local coordinates. (D) The learned dynamics.

C. Kuramoto-Sivashinsky beating travelling dynamics

Our next example also comes from the K-S equation, this time with $\nu = \frac{4}{87}$ [24]. After transients have died out, we obtain a beating travelling wave. The travelling period and beating period are incommensurable, making the orbit quasiperiodic. Note the separation in timescales: the two periods differ by a factor of nearly 200. This quasiperiodic orbit lives on a two-dimensional submanifold (a 2-torus) of $\mathbb{R}^{64}$, shown in figure 6A.

We separate the data into shape (figure 6B) and phase (figure 6C) variables (the shape variable's first Fourier mode's phase is zero) [3, 17]. Because of the translational equivariance of (3), the dynamics only depend on the shape variable, which lives on a one-dimensional submanifold of $\mathbb{R}^{64}$.

Our training data are shown in figure 7A. The data span a bit more than two beating periods, just over 1% of the travelling period. We place the data into three clusters and follow our procedure to create an atlas of three charts and a local low-dimensional dynamical model for each chart (figure 7A–C). Additionally, for each chart we train a neural network that takes the shape variable (in local coordinates) as input and predicts the change in phase over one time step, giving the phase dynamics [17].

In figure 7D, we show a trajectory that results from evolving an initial condition forward

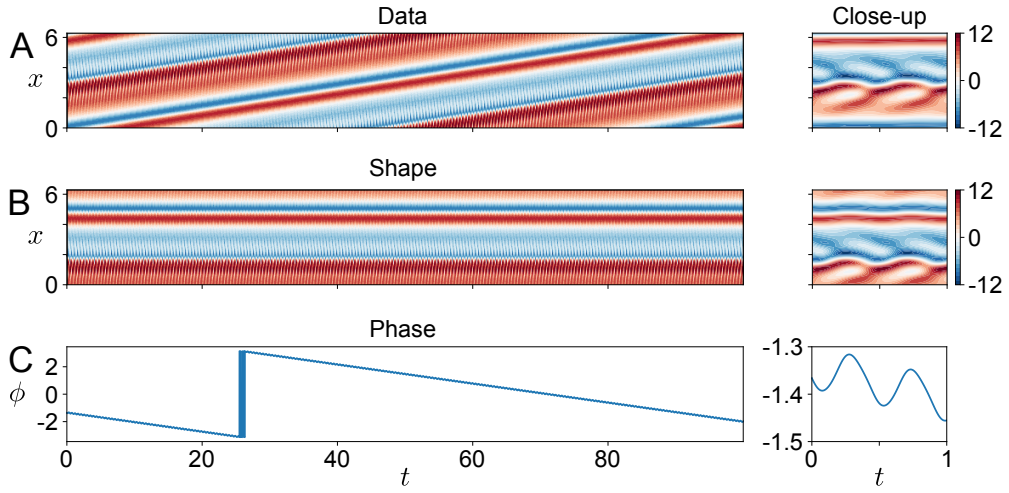

FIG. 6. The beating travelling wave data from the K-S system (A) are separated into shape (B) and phase (C) data. Plots on the right show close-up views of the first time unit of the dynamics.

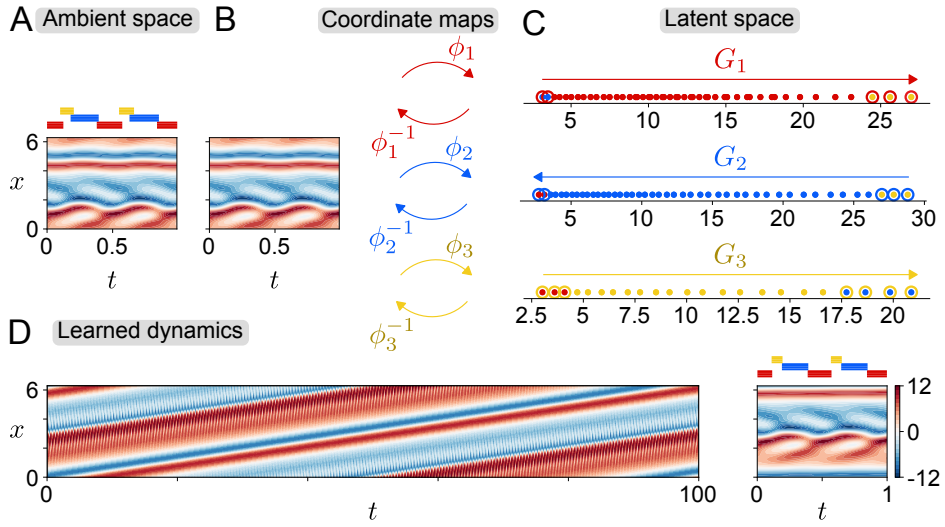

FIG. 7. Analogous to figure 5, but for a beating travelling wave. In D, we show the predicted dynamics starting from the same initial condition as in figure 6A.

in time under our learned dynamical model. Comparing figure 7D to figure 6A, our learned model again produces good results with seamless transitions between charts. The beating period of our model is 0.456 ± 0.001 time units, identical to the data, and the travelling period of our model is 94.88 ± 0.10 time units, compared to 90.46 ± 0.10 time units for the data. The discrepancy improves with increased training time of the neural networks.

It is worth re-iterating that the two periods of the quasiperiodic data differ by a factor of almost 200. Despite the large separation in timescales, our method produced a good

dynamical model. Moreover, we were able to take advantage of a symmetry present in the system to learn from data that spans a small fraction of one of the timescales while nevertheless capturing that timescale in the final dynamical model.

VII. DETAILS OF DATASETS AND MODELS

Here we provide details about the training datasets, neural network architectures, training procedures, and hyperparameters used for each of the examples. Neural networks were built and trained using Keras [6]. Unless otherwise indicated, all neural network layers are fully-connected, the mean squared error is used as the loss function, the default Adam optimizer is used for training [15], the default Glorot uniform initializer is used for weight initialization [12], and the batch size is the full number of training data points in the corresponding chart.

We did not attempt to optimize architectures nor hyperparameters, simply using ones that produced accurate results. To choose the dimension of the latent space, we either knew what the true dimension of the submanifold was (in which case we used that dimension), or we based our decision on reconstruction errors (i.e., figure 6E in the main text suggests that the latent dimension of the bursting data is 3 since the reconstruction error reaches a plateau there [17]; note that the latent dimension cannot be 2 since such a low dimension is not capable of producing the dynamics we observe). To choose the number of charts, we often knew the minimum number needed, and chose nearly that many. For the bursting example, we chose to use 6 charts based on an exploratory data analysis, where we saw that the state space structure of the data consists of two saddles connected by four heteroclinic orbits.

A. Examples from main text

1. Periodic dynamics on a circle

The training data are the Cartesian (x, y) coordinates of a particle as it moves counter-clockwise around the unit circle at a constant speed. There are 40 data points in total, uniformly spaced $\pi/20$ radians apart on the unit circle.

We use three charts (three clusters in k-means). The graph is built by connecting points to their two nearest neighbours. Clusters are expanded twice along the graph.

	Shape	Activations	Epochs
Encoders	2 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	1000
Decoders	1 : 4 : 16 : 32 : 32 : 2	elu : elu : elu : elu : linear	1000
Dynamics	1 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	500

TABLE I. Neural network parameters for periodic dynamics on a circle. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

The neural network parameters are listed in table I. The autoencoders are trained with a learning rate of 0.01, and the neural networks for dynamics are trained with a learning rate of 0.005.

2. Quasiperiodic dynamics on a torus

The training data are the Cartesian (x, y, z) coordinates of a particle as it moves around the surface of a torus. The surface of the torus is given by

$$x = (1 + 0.5\cos\theta)\cos\phi,\ y = (1 + 0.5\cos\theta)\sin\phi,\ z = 0.5\sin\theta, \tag{4}$$

where θ is the poloidal coordinate and ϕ is the toroidal coordinate. The particle moves at a constant rate in the poloidal and toroidal directions, $\sqrt{3}$ times as fast in the poloidal direction, generating a quasiperiodic orbit. There are 1000 data points in total, uniformly spaced $3\pi/100$ radians apart in the toroidal direction, covering 15 full periods of motion in the toroidal direction.

We use six charts (six clusters in k-means). The graph is built by connecting points to their five nearest neighbours. Clusters are expanded twice along the graph.

The neural network parameters are listed in table II. The autoencoders are trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.8 every 200 steps using a staircase function), and the neural networks for dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function).

For the learned model whose results are shown in the main text, we are confident that it indeed has quasiperiodic dynamics (we used the model to produce a trajectory that is

	Shape	Activations	Epochs
Encoders	3 : 32 : 64 : 32 : 16 : 4 : 2	elu : elu : elu : elu : elu : linear	2000
Decoders	2 : 4 : 16 : 32 : 64 : 32 : 3	elu : elu : elu : elu : elu : linear	2000
Dynamics	2 : 32 : 32 : 16 : 4 : 2	elu : elu : elu : elu : linear	2000

TABLE II. Neural network parameters for quasiperiodic dynamics on a torus. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

50,000 time steps long, finding no repeating pattern). However, the majority of the time we learn models that have periodic dynamics, with the period being on the order of the number of training data but varying fairly significantly (ten learned models with periodic dynamics had periods between 418 and 1892 time steps). We attribute this to the existence—in the space of weights for the neural networks for the dynamics—of Arnold tongues, leading to a mode locking phenomenon that often causes our learned model to have periodic dynamics. In short, if the dynamics of a nominally quasiperiodic system are perturbed, then the dynamics will be periodic in a finite region of parameter values; see [32] for more details.

3. Reaction-diffusion system

We compute a numerical solution to a lambda-omega reaction-diffusion system governed by

$$\begin{aligned} u_t &= [1-(u^2+v^2)]u + \beta(u^2+v^2)v + d_1(u_{xx}+u_{yy}), \\ v_t &= -\beta(u^2+v^2)u + [1-(u^2+v^2)]v + d_2(v_{xx}+v_{yy}), \end{aligned} \tag{5}$$

for $-10 \le x \le 10$, $-10 \le y \le 10$. The parameter values are $d_1 = d_2 = 0.1$ and $\beta = 1$, and we impose homogeneous Neumann boundary conditions on u and v. The domain is discretized into a uniform 101×101 grid, and the second-order derivatives from the diffusion terms are approximated by second-order centered finite differences. Time integration is performed using MATLAB's `ode45` function with default settings [19]. The initial condition is

$$\begin{aligned} u(x,y,0) &= \tanh\left[\sqrt{x^2+y^2}\cos\left(\angle(x+iy) - \sqrt{x^2+y^2}\right)\right], \\ v(x,y,0) &= \tanh\left[\sqrt{x^2+y^2}\sin\left(\angle(x+iy) - \sqrt{x^2+y^2}\right)\right]. \end{aligned} \tag{6}$$

Once transients decay, we obtain a spiral wave and collect our data. The training data

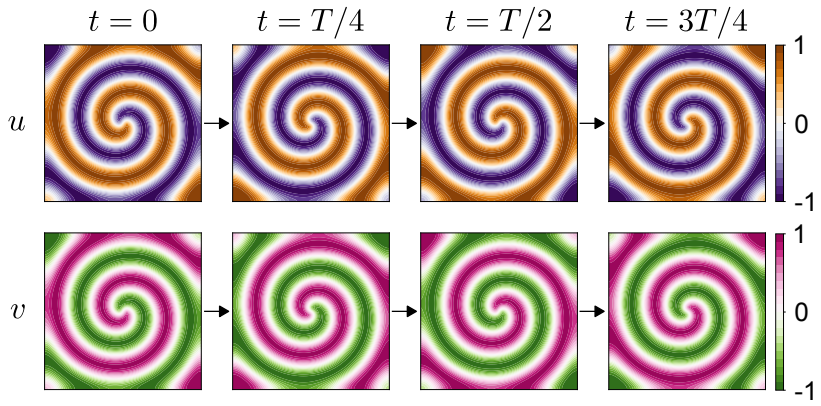

FIG. 8. Four snapshots showing one period T of the spiral wave.

	Shape	Activations	Epochs
Encoders	20402 : 128 : 64 : 16 : 8 : 1	elu : elu : elu : elu : linear	2000
Decoders	1 : 8 : 16 : 64: 128 : 20402	elu : elu : elu : elu : linear	2000
Dynamics	1 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	2000

TABLE III. Neural network parameters for spiral wave dynamics of the reaction-diffusion system. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

are the values of the fields u and v at all grid points; our dataset consists of 200 such data points, uniformly spaced 0.05 time units apart, covering a bit more than one period of the spiral wave. One period of the spiral wave is shown in figure 8.

We use three charts (three clusters in k-means). The graph is built by connecting points to their four nearest neighbours. Clusters are expanded once along the graph.

The neural network parameters are listed in table III. The autoencoders are trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.8 every 200 steps using a staircase function), and the neural networks for dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function).

4. Kuramoto-Sivashinsky bursting dynamics

We compute a numerical solution to the Kuramoto-Sivashinsky equation using the scheme described in Section VII B 2. We set $\nu = \frac{16}{71}$; once transients decay, we obtain bursting dynamics and collect our data. The training data are the values of the field u at 64 points spaced equally in x; our dataset consists of 6565 such data points, uniformly spaced 0.05 time units apart, covering 16 bursts.

The data described above are used to train the autoencoders. The neural networks for dynamics are trained using off-attractor data. The reason for using off-attractor data to learn the dynamics is that the attractor is very thin, making it difficult to learn accurate dynamics in the three-dimensional latent spaces. When only using the original dataset to learn the dynamics, we found that the learned dynamics often included a stable fixed point that was off the attractor (in a region where there was no data to learn from). To create the dynamics dataset, we took every third data point of the autoencoder dataset (2189 in total), perturbed them, and used the resulting data as initial conditions for new simulations. The perturbations are given by

$$a_1 \cos 2x + a_2 \cos x + a_3 \sin x, \tag{7}$$

where the a_i are randomly sampled from a uniform distribution spanning $[-0.05, 0.05]$. The new simulations were all run for 1.5 time units, and only the last time unit of data was stored. In the end, our dynamics dataset consists of 2189 slightly-off-attractor trajectories, each spanning one time unit and consisting of 21 data points uniformly spaced 0.05 time units apart. The new dynamics dataset was partitioned and transformed to the latent spaces based on the previously trained charts before training the neural networks for the dynamics.

We use six charts (six clusters in k-means). The graph is built by connecting points to their four nearest neighbours. Clusters are expanded twice along the graph.

The neural network parameters are listed in table IV. For the autoencoders, we use the method described in [17] with $\alpha = 1$ (cf. eq. 4 in that work). The autoencoders are trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.8 every 200 steps using a staircase function), and the neural networks for dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function).

	Shape	Activations	Epochs
Encoders	64 : 128 : 64 : 16 : 8 : 3	elu : elu : elu : elu : linear	1000
Decoders	3 : 8 : 16 : 64 : 128 : 64	elu : elu : elu : elu : linear	1000
Shape dynamics	3 : 16 : 64 : 64 : 16 : 3	elu : elu : elu : elu : linear	1000

TABLE IV. Neural network parameters for bursting dynamics of the Kuramoto-Sivashinsky equation. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

In this example, we add an additional step before learning the dynamics. In each chart, we take the training data in their local coordinates, subtract the mean, and perform principal components analysis (PCA), changing the local coordinates to their coordinates in the PCA basis (without any dimension reduction). We then normalize each PCA coordinate by the data's standard deviation along the coordinate. This set of steps is similar to whitening. We found this type of normalization to be helpful in learning accurate dynamics. The rationale for normalizing the local coordinates before learning the dynamics is that the saddle points have sharp cusps leading into them, and we thought normalizing the local coordinates would make the cusps less sharp and therefore make the sensitive dynamics easier to learn.

Even with the above modifications, many of our learned models display periodic dynamics. We have found models that have one stable limit cycle (essentially consisting of two of the heteroclinic orbits), two stable limit cycles (each consisting of an independent pair of the heteroclinic orbits), and one longer stable limit cycle (consisting of all four heteroclinic orbits), in addition to models with non-periodic dynamics. We attribute this to the presence—in the space of weights for the neural networks for the dynamics—of a gluing bifurcation [9, 11, 13]. In short, perturbations to the dynamics can "glue" the heteroclinic orbits together, forming periodic orbits of various degrees of complexity. It may be possible to avoid the gluing bifurcation by enforcing some of the symmetries present in the K-S system [14], but we do not pursue this avenue here. The main text describes results from a model that displays qualitatively correct non-periodic dynamics.

	Shape	Activations	Epochs
Encoders	3 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	1000
Decoders	1 : 4 : 16 : 32 : 32 : 3	elu : elu : elu : elu : linear	1000
Dynamics	1 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	1000

TABLE V. Neural network parameters for periodic dynamics on a torus. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

B. Additional examples from Supplementary Information

1. Periodic dynamics on a torus

The training data are the Cartesian (x, y, z) coordinates of a particle as it moves around the surface of a torus. The surface of the torus is given by (4). The particle moves at a constant rate in the poloidal and toroidal directions, three times as fast in the poloidal direction, generating a periodic orbit. There are 100 data points in total, uniformly spaced $\pi/50$ radians apart in the toroidal direction, covering one full period of the motion.

We use three charts (three clusters in k-means). The graph is built by connecting points to their two nearest neighbours. Clusters are expanded twice along the graph.

The neural network parameters are listed in table V. The autoencoders are trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function), and the neural networks for dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function).

2. Kuramoto-Sivashinsky beating dynamics

We compute a numerical solution to the Kuramoto-Sivashinsky equation,

$$u_t + uu_x + u_{xx} + \nu u_{xxxx} = 0, \tag{8}$$

for $0 \leq x \leq 2\pi$ with periodic boundary conditions using a pseudo-spectral method with 64 Fourier modes. Time integration is performed using Crank-Nicolson for the linear terms and two-step Adams-Bashforth for the nonlinear term. The first step of the time integration is

	Shape	Activations	Epochs
Encoders	64 : 128 : 64 : 16 : 8 : 1	elu : elu : elu : elu : linear	2000
Decoders	1 : 8 : 16 : 64: 128 : 64	elu : elu : elu : elu : linear	2000
Dynamics	1 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	2000

TABLE VI. Neural network parameters for beating dynamics of the Kuramoto-Sivashinsky equation. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

performed using the fourth-order Runge-Kutta method. The time step used is 10^{-4}. The initial condition is $u(x,0) = -\sin x + 2\cos 2x + 3\cos 3x - 4\sin 4x$ [24].

We set $\nu = \frac{16}{337}$; once transients decay, we obtain a beating standing wave and collect our data. The training data are the values of the field u at 64 points spaced equally in x; our dataset consists of 100 such data points, uniformly spaced 0.01 time units apart, covering a bit more than two periods of the beating wave.

We use three charts (three clusters in k-means). The graph is built by connecting points to their four nearest neighbours. Clusters are expanded once along the graph.

The neural network parameters are listed in table VI. The autoencoders are trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.8 every 200 steps using a staircase function), and the neural networks for dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function).

3. Kuramoto-Sivashinsky beating travelling dynamics

We compute a numerical solution to the Kuramoto-Sivashinsky equation as before. We set $\nu = \frac{4}{87}$; once transients decay, we obtain a beating travelling wave and collect our data. The training data are the values of the field u at 64 points spaced equally in x; our dataset consists of 100 such data points, uniformly spaced 0.01 time units apart, covering a bit more than two beating periods and 1% of the travelling period. Separation of the data into shape and phase variables is described in the main text.

We use three charts (three clusters in k-means). The graph is built by connecting points to their four nearest neighbours. Clusters are expanded twice along the graph.

	Shape	Activations	Epochs
Encoders	64 : 128 : 64 : 16 : 8 : 1	elu : elu : elu : elu : linear	2000
Decoders	1 : 8 : 16 : 64 : 128 : 64	elu : elu : elu : elu : linear	2000
Shape dynamics	1 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	2000
Phase dynamics	1 : 32 : 32 : 16 : 4 : 1	elu : elu : elu : elu : linear	2000

TABLE VII. Neural network parameters for beating travelling dynamics of the Kuramoto-Sivashinsky equation. "Shape" indicates the dimension of each layer, "activation" indicates the activation functions between layers, and "epochs" indicates the number of epochs used for training.

The neural network parameters are listed in table VII. The autoencoders are trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function), the neural networks for shape dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function), and the neural networks for phase dynamics are also trained with an exponential decay learning rate schedule (initial learning rate 0.01, decay rate 0.9 every 200 steps using a staircase function).

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[2] M. Brand. Charting a manifold. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 985–992. MIT Press, 2003.

[3] N. B. Budanur, P. Cvitanović, R. L. Davidchack, and E. Siminos. Reduction of SO(2) symmetry for spatially extended dynamical systems. *Physical Review Letters*, 114(8):084102, 2015.

[4] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[5] B. Chen, K. Huang, S. Raghupathi, I. Chandratreya, Q. Du, and H. Lipson. Automated discovery of fundamental variables hidden in experimental data. *Nature Computational Science*, 2(7):433–442, 2022.

[6] F. Chollet et al. Keras. `https://keras.io`, 2015.

[7] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[8] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[9] J.-M. Gambaudo, P. Glendinning, and C. Tresser. Collage de cycles et suites de Farey. *Comptes rendus des scéances de l'Académie des sciences, Série 1, Mathématique*, 299(14):711–714, 1984.

[10] M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014.

[11] P. Glendinning. Global bifurcations in flows. In T. Bedford and J. Swift, editors, *New Directions in Dynamical Systems*, volume 127 of *London Mathematical Society Lecture Note Series*, pages 120–149. Cambridge University Press, Cambridge, 1988.

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[13] M. D. Graham, U. Middya, and D. Luss. Pulses and global bifurcations in a nonlocal reaction-diffusion system. *Physical Review E*, 48(4):2917, 1993.

[14] I. G. Kevrekidis, B. Nicolaenko, and J. C. Scovel. Back in the saddle again: a computer assisted study of the Kuramoto–Sivashinsky equation. *SIAM Journal on Applied Mathematics*, 50(3):760–790, 1990.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] J. M. Lee. *Introduction to Smooth Manifolds*. Springer, 2013.

[17] A. J. Linot and M. D. Graham. Deep learning to discover and predict dynamics on an inertial manifold. *Physical Review E*, 101(6):062209, 2020.

[18] T.. Liu, A. W. Moore, A. Gray, and C. Cardie. New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research*, 7(6), 2006.

[19] MATLAB. *version 9.10.0 (R2021a)*. The MathWorks Inc., Natick, Massachusetts, 2021.

[20] S. M. Omohundro. Five balltree construction algorithms. Technical report, International

Computer Science Institute Berkeley, 1989.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] N. Pitelis, C. Russell, and L. Agapito. Learning a manifold as an atlas. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1642–1649, 2013.

[23] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[24] C. W. Rowley and J. E. Marsden. Reconstruction equations and the Karhunen–Loève expansion for systems with symmetry. *Physica D: Nonlinear Phenomena*, 142(1-2):1–19, 2000.

[25] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[26] S. Schonsheck, J. Chen, and R. Lai. Chart auto-encoders for manifold structured data. *arXiv preprint arXiv:1912.10094*, 2019.

[27] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[28] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

[29] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti. Visual interaction networks: Learning a physics simulator from video. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[30] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24th International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205. Morgan Kaufmann, 1998.

[31] H. Whitney. The self-intersections of a smooth n-manifold in $2n$-space. *Annals of Mathematics*, pages 220–246, 1944.

[32] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Springer-Verlag

New York, 2 edition, 2003.