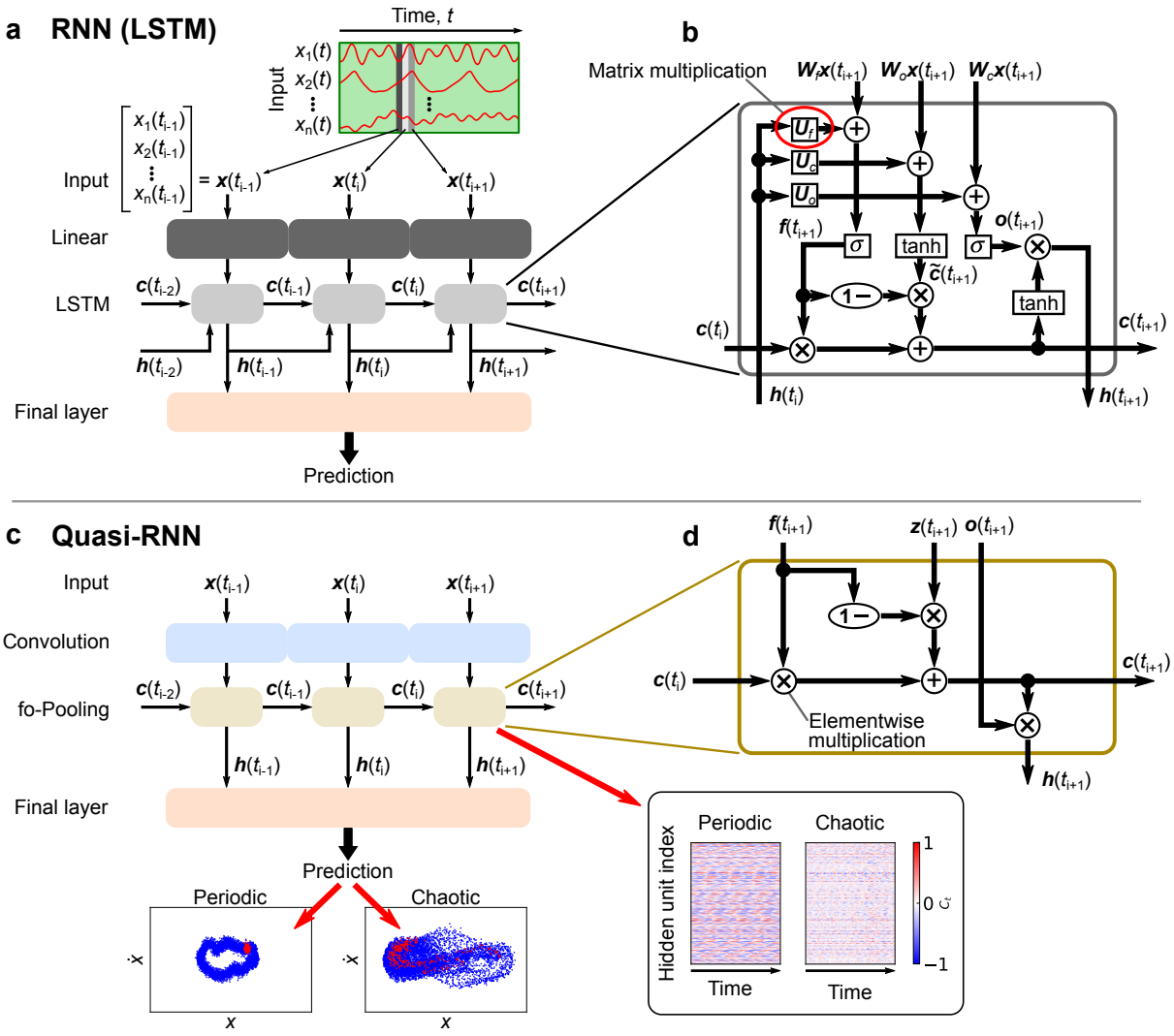
Supplementary Information: Data-driven prediction and analysis of chaotic origami dynamics

Hiromi Yasuda[1,2], Koshiro Yamaguchi[1], Yasuhiro Miyazawa[1], Richard Wiebe[3], Jordan R. Raney[2], and Jinkyu Yang[1]

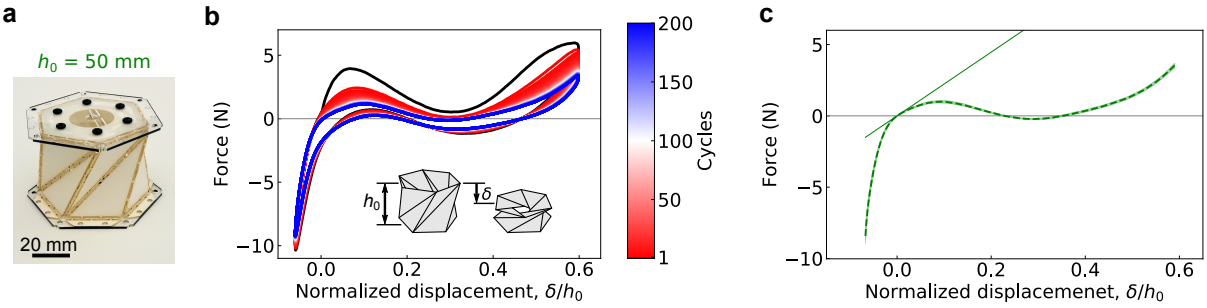
[1]*Department of Aeronautics & Astronautics, University of Washington, Seattle, WA 98195-2400, USA*

[2]*Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104, USA*

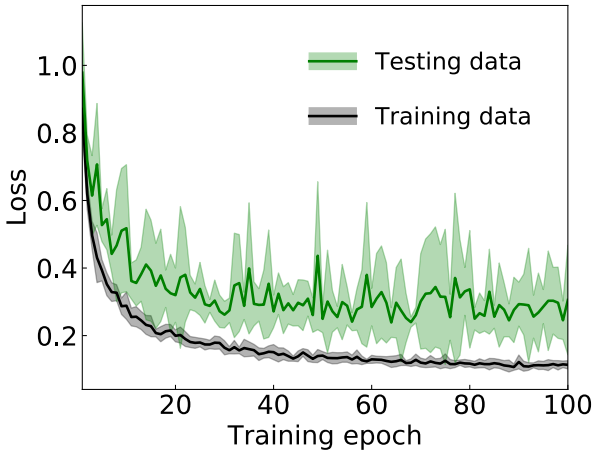
[3]*Department of Civil and Environmental Engineering, University of Washington, Seattle, WA 98195-2400*



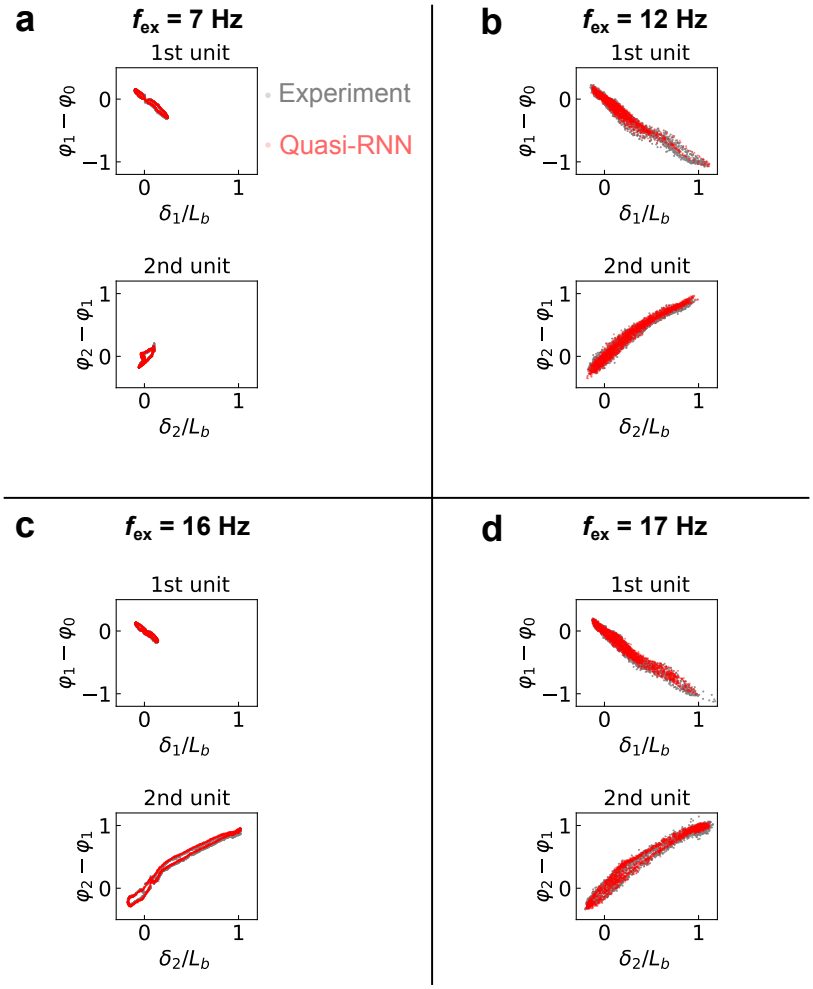
Supplementary Figure 1: **Schematic illustration of the comparison between a recurrent neural network (RNN), specifically a long short term memory (LSTM), and quasi-recurrent neural network (QRNN).** (**a–b**) Block diagram of a RNN/LSTM shows the conceptual computation structure operating on the discrete time series data (see **a**). LSTM cells in the hidden layer produce the hidden state vectors $c(t_i)$ and $h(t_i)$ (see Eq. (2)), and this operation depends on both $c(t_{i-1})$ and $h(t_{i-1})$ from the previous time step. Internal calculations of a LSTM cell consists of matrix multiplications (see matrices U_f, U_c, and U_o for forget gate (f), candidate (c, or new candidate value $\tilde{c}$), and output gate (o), respectively; e.g., U_f denoted by the red circle) and element wise multiplications (see **b**). (**c–d**) Block diagram of a QRNN shows the computation structure composed of convolution and pooling layers (**c**). Pooling parts in which recurrent relation is included produce $c(t_i)$ and $h(t_i)$ based the forget gate (f), candidate vector (z), and output gate (o). However this operation depends only on $c(t_{i-1})$ from the previous time step instead of using both $c(t_{i-1})$ and $h(t_{i-1})$ from the previous time step. Within each pooling layer, the calculations consist of only element-wise multiplications without direct interactions between different hidden units which allows both prediction (Lower left insets show phase planes for periodic and chaotic cases. Red dots represent Poincaré map) and visualization of hidden states readily without additional layers or components in the system (**d**).



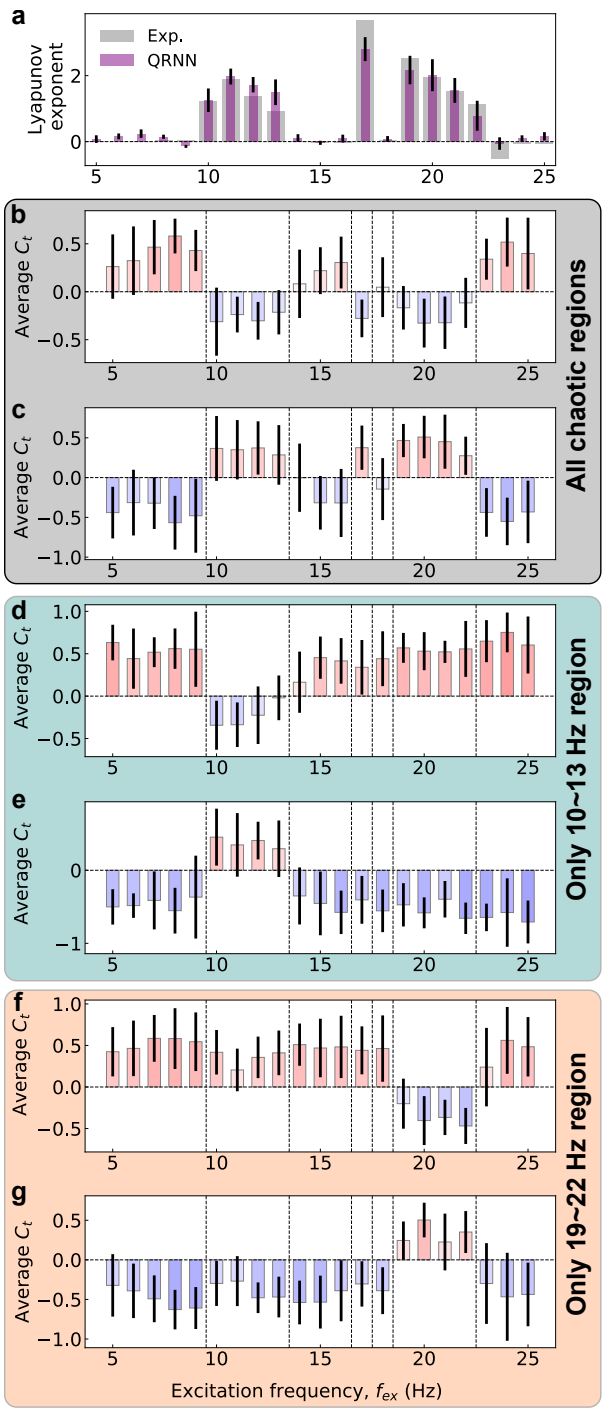
Supplementary Figure 2: **Cyclic loading tests on the triangulated cylindrical origami (TCO) unit cell.** (**a**) Actual paper prototype with the design parameters $(h_0, \theta_0, R) = (50 \text{ mm}, 70^\circ, 36 \text{ mm})$ (see Fig. 1c-d in the main text), (**b**) cyclic loading test result, and (**c**) the loading curve from the last (200 th) cycle. In (**c**), the dashed line and colored thin area indicate the mean value and standard deviation calculated from 6 different samples, respectively. By using the loading curve from the last cycle as shown in (**b**), we obtain the slope, i.e., stiffness value, at the initial state by applying the linear fitting, which is denoted by the slanted solid line in (**c**).



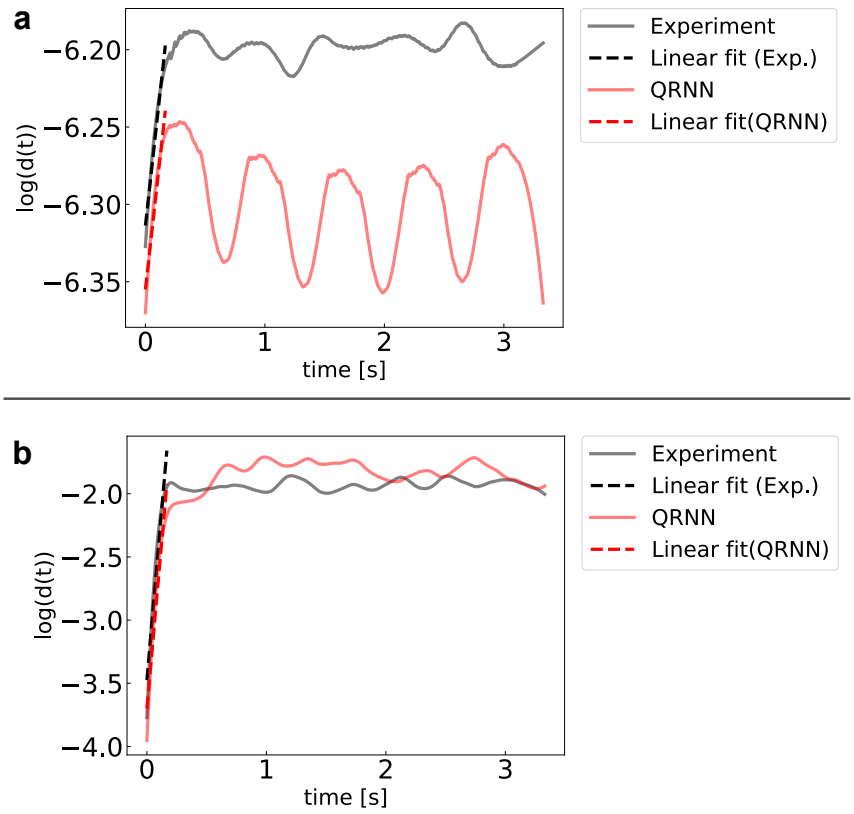
Supplementary Figure 3: **Training result.** Training loss calculated from training and testing data sets over 100 training epochs is shown. The solid lines and shaded areas indicate the mean and standard deviation from 10 different training runs, respectively.



Supplementary Figure 4: **Deformation of each triangulated cylindrical origami (TCO) unit cell.** Trajectories of the first and second TCO unit cells are plotted as a function of axial deformation (δ_i/L_b) and rotational angle ($\varphi_i - \varphi_{i-1}$) where $i = 1, 2$. The excitation frequency (f_{ex}) is (**a**) 7 Hz, (**b**) 12 Hz, (**c**) 16 Hz, and (**d**) 17 Hz. In the plots, the grey (red) markers indicate the experimentally measured (predicted) data points.



Supplementary Figure 5: **Average neuron activation of the hidden units in the final third hidden layer over 10 training runs.** (**a**) Lyupnov exponent values for different frequency cases are calculated from the experiments (grey) and predictions from the QRNN (purple). The error bars indicate the standard deviations. We extract the activation of the hidden unit cells which exhibit various activation patterns such as negative/positive activation for (**b-c**) three chaotic regions, (**d-e**) only the first chaotic regime (10-13 Hz), and (**f-g**) only the third region (19-22 Hz). The error bars are calculated based on the results of 10 different training. We selected six different hidden units from each trained QRNN to create panels **(b-g)**. The difference between panels **b** and **c** (or **d** and **e**, or **f** and **g**) is the different hidden units. Note that we use different sets of 6 neurons from 10 training runs, then calculate the average values and error bars over these 10 runs.



Supplementary Figure 6: **Divergence from the nearest neighbor.** The logarithm of the divergence $d(t)$ from the nearest neighbor of the trajectories of unit 1 as a function of time for external frequency (**a**) 6 Hz and (**b**) 20 Hz. These are the examples of the calculation of the Lyapunov exponent from the (**a**) periodic and (**b**) chaotic responses. The values of the inclination of dashed lines correspond to the values of the Lyapunov exponent. Each result from the data set of experiments and QRNN output is a single run.

Supplementary Table I: **Forcing frequencies and time windows.** Forcing frequency denotes the excitation frequency applied to the base of TCO units. Time windows were used to decide the region of the linear fit (dashed lines) in Supplementary Figure 6. They were chosen for each forcing frequency.

Forcing frequency f_{input} [Hz]	Time window $\Delta t[s]$	Forcing frequency f_{input} [Hz]	Time window $\Delta t[s]$
5	0.6	16	0.27
6	0.5	17	0.18
7	0.43	18	0.17
8	0.38	19	0.16
9	0.33	20	0.15
10	0.3	21	0.14
11	0.27	22	0.14
12	0.25	23	0.13
13	0.23	24	0.13
14	0.21	25	0.12
15	0.2		

Supplementary Note 1: Quasi-recurrent neural network

The QRNN processes time-series input data and computes a sequence of output via convolution components and pooling components [16] (see Supplementary Figure 1b). Let input data be $\mathbf{X} = [\mathbf{x}(t_1), \mathbf{x}(t_2), \cdots, \mathbf{x}(t_T)]$ where $\mathbf{x}\,(t_i) = [x_1(t_i), x_2(t_i), \cdots x_n(t_i)]^T$ $(i = 1, 2, \ldots, T)$, we calculate a sequence $\mathbf{Z} \in \mathbb{R}^{T \times m}$ composed of candidate vectors ($\mathbf{z}(t_i)$), and $\mathbf{F}$ and $\mathbf{O}$ composed of $\mathbf{f}(t_i)$ and $\mathbf{o}(t_i)$ respectively for the gates required for the pooling components. Here, m is a number of hidden units in a layer. The computation of the convolutional layer is expressed by

$$\begin{aligned} \mathbf{Z} &= \tanh\left(\mathbf{W}_Z * \mathbf{X}\right) \\ \mathbf{F} &= \sigma\left(\mathbf{W}_F * \mathbf{X}\right) \\ \mathbf{O} &= \sigma\left(\mathbf{W}_O * \mathbf{X}\right) \end{aligned} \tag{1}$$

where $\mathbf{W}_z$, $\mathbf{W}_f$, and $\mathbf{W}_o \in \mathbb{R}^{k \times n \times m}$ indicate convolutional filter banks with filter width of k, and these are optimized during training. Also, $\sigma\left(\boldsymbol{.}\right)$ is a sigmoid function, and $*$ denotes a masked convolution along the time sequence. For the pooling components, we utilize the *fo*-pooling [16] as follows:

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t \\ \mathbf{h}_t &= \mathbf{o}_t \odot \mathbf{c}_t \end{aligned} \tag{2}$$

where $\odot$ is elementwise multiplication. Please note that unlike standard RNNs, the calculation of the hidden state ($\mathbf{c}_t$) of the QRNN depends only on the hidden state at previous time step ($\mathbf{c}_{t-1}$) instead of using both $c(t_{i-1})$ and $h(t_{i-1})$ from the previous time step, and the *fo*-pooling layer of the QRNN computes only elementwise multiplication, which can avoid mixing different channel information due to matrix multiplication with a dense matrix. These convolution component and pooling part constitute a single QRNN hidden layer, and like standard convolutional neural networks, multiple hidden layers can enhance the ability to express complex behaviors.

Supplementary Note 2: Lyapunov exponent

Perhaps the most important characteristic of chaos is its extreme sensitivity to initial conditions [1, 2]. This can be quantified by measuring the Lyapunov exponents of dynamical systems. These quantify the average rate of local exponential divergence of initially close state-space trajectories. The appearance of one or more Positive Lyapunov exponents indicates almost-sure

instability in the dynamics describing separation distance between nearest neighbors, and can be associated with chaotic response. In this manuscript, we adopt Rosenstein's method [35] to obtain the largest (most positive, and most indicative of chaos) Lyapunov exponent of the system. This method has been previously applied to the experimental results of a nonlinear mechanical oscillator [3] and has shown its effectiveness alongside the FFT-based analysis. In Rosenstein's method, first, a phase portrait is first reconstructed using time-delay embedding. Thereafter, nearest neighbors for each data point are found. The divergence rate of the pairs of nearest neighbors is then tracked, and the average of the logarithm of the divergence rate is plotted. The slope of the initial linear part of the resulting plot corresponds to the largest Lyapunov exponent of the system. This algorithm was applied to the actual data of the first and the second unit from the experiments and the predictions via QRNN. Because the lateral and rotational displacements are coupled for each unit, only the largest Lyapunov exponents of the former were calculated.

The dataset size is $N = 800$, and the embedding dimension is $m = 5$. The reconstruction delay is 1/4 of the forcing period, where it is measured by f_{cam}/f_{input}. Here, $f_{cam} = 240$ Hz is the framerate of the cameras for the testing setup, and f_{input} is the excitation frequency for each test. The 1/4 period delay commonly typically works well for harmonically forced systems. One of the most challenging aspects of Rosenstein's algorithm is selecting the time windows over which to calculate the Lyapunov exponents. This is because as separation distance between nearest neighbors grows, nonlinear terms begin to affect the response and 'pollute' the local (linearized) exponential dynamics. In scenarios where the governing equations are known, this can be addressed via renormalization. In a data-driven approach, however, this is more difficult as the time horizon at which nonlinear effects occur changes depending on the initial separation of neighbors, separation direction in state space, and also the Lyapunov exponent itself. To address this challenge, the plots herein were calculated using a sliding sampling window to sample data and to average the largest Lyapunov exponents obtained for each calculation. The time windows we chose are three times larger than the forcing frequency, and Supplementary Table. I shows the values for each forcing frequency. We take multiple linear fits starting from 0 to 0.2 second with those time windows. For example, in 25 Hz case, the first sample is between 0 and 0.12 second, and the last sample is between 0.2 and 0.32 second. Then, we average the results and determine the largest Lyapunov exponent. Supplementary Figure 6 shows one example of linear fits for 6 Hz and 20 Hz case. The largest Lyapunov exponents for each excitation frequency is shown in Fig.4f in the main article.

Supplementary References

[1] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering* (Westview Press, Boulder, 2015), second edition

[2] S. Wiggins, *Introduction to applied nonlinear dynamical systems and chaos*, no. 2 in Texts in applied mathematics (Springer, New York, 2003), 2nd

[3] R. Wiebe and L. N. Virgin, Chaos **22**, 013136 (2012).

[16] J. Bradbury, S. Merity, C. Xiong, and R. Socher, in *International Conference on Learning Representations (ICLR 2017)* (2017), pp. 1–11, 1611.01576.

[35] M. T. Rosenstein, J. J. Collins, and C. J. De Luca, Physica D: Nonlinear Phenomena **65**, 117 (1993).