Machine Learning: Science and Technology

PAPER • OPEN ACCESS

Deep learning of chaos classification

MACHINE
LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
19 May 2020

REVISED
2 August 2020

ACCEPTED FOR PUBLICATION
9 September 2020

PUBLISHED
22 October 2020

PAPER

Deep learning of chaos classification

Woo Seok Lee[1] and Sergej Flach[1,2]

[1] Center for Theoretical Physics of Complex Systems, Institute for Basic Science, 34051 Daejeon, Republic of Korea
[2] Basic Science Program, Korea University of Science and Technology (UST), 34113 Daejeon, Republic of Korea

E-mail: wooseoklee06@gmail.com

Keywords: machine learning, chaotic dynamics, Chirikov standard map, standard map

Abstract

We train an artificial neural network which distinguishes chaotic and regular dynamics of the two-dimensional Chirikov standard map. We use finite length trajectories and compare the performance with traditional numerical methods which need to evaluate the Lyapunov exponent (LE). The neural network has superior performance for short periods with length down to 10 Lyapunov times on which the traditional LE computation is far from converging. We show the robustness of the neural network to varying control parameters, in particular we train with one set of control parameters, and successfully test in a complementary set. Furthermore, we use the neural network to successfully test the dynamics of discrete maps in different dimensions, e.g. the one-dimensional logistic map and a three-dimensional discrete version of the Lorenz system. Our results demonstrate that a convolutional neural network can be used as an excellent chaos indicator.

1. Introduction

Chaotic dynamics exists in many natural systems, such as heartbeat irregularities, weather and climate [1, 2]. Such dynamics can be studied through the analysis of proper mathematical models which generate non-linear dynamics and deterministic chaos. Chaotic and regular dynamics can co-exist in the phase space of low-dimensional systems [3]. To distinguish chaotic from regular dynamics, tangent dynamics is used to compute Lyapunov exponents (LEs) λ. In practice one integrates the tangent dynamics along a given trajectory and averages a finite time LE $\lambda(t)$. The averaging time T needed to reliably tell regular ($\lambda = 0$) from chaotic ($\lambda \neq 0$) trajectories apart is usually orders of magnitude larger than the Lyapunov time $T_\lambda \equiv 1/\lambda$.

Here, we introduce a machine learning approach that alleviates the problems of calculating LEs and can be used as a new chaos indicator. Machine learning has shown tremendous performance e.g. in pattern recognition [4, 5]. Machine learning approaches turned useful to solve partial differential equations and identify hidden physics models from experimental data [6–8]. Machine learning was used recently to predict future chaotic dynamics details from time series data without knowledge of the generating equations [9, 10]. In this paper, we introduce a machine learning way to use short time series data for telling chaos from regularity apart. We train a neural network using chaotic and regular trajectories from the Chirikov standard map. Our method has a success rate of 98% using trajectories with length $10T_\lambda$, while conventional methods need up to $10^4 T_\lambda$ to reach the same accuracy. The main reason for the small but finite failure rate of our machine learning method is due to sticky orbits. These orbits are chaotic, yet can mimic regular ones for long times due to trapping in fractal boundary phase space regions separating chaotic and regular dynamics. Our method is also surprisingly successful when trained with Standard Map data but tested on maps with different dimensions such as the logistic map ($d = 1$) and the Lorenz system ($d = 3$).

2. The Chirikov standard map

The Chirikov standard map is an area-preserving map in dimension $d = 2$ [11] also known as the kicked rotor [3]

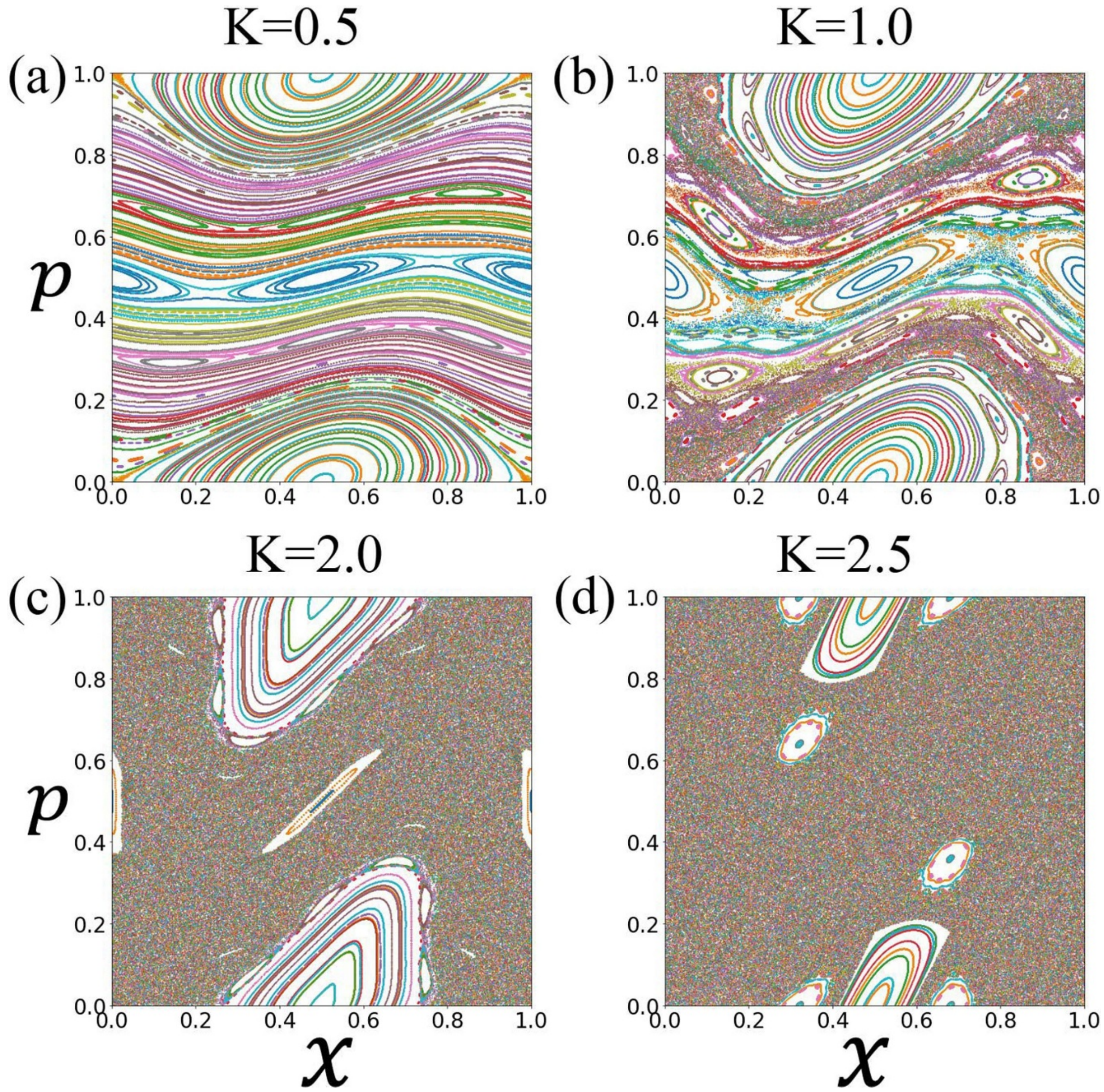

Figure 1. Examples of phase portraits (Poincare sections) of the standard map. (a) $K = 0.5$, (b) $K = 1.0$, (c) $K = 2.0$, (d) $K = 2.5$.

$$\begin{aligned} p_{n+1} &= p_n + \frac{K}{2\pi}\sin(2\pi x_n) \qquad \text{mod } 1 \,, \\ x_{n+1} &= x_n + p_{n+1} \qquad \text{mod } 1 \,. \end{aligned} \tag{1}$$

The kick strength K controls the degree of nonintegrability and chaos appearing in the dynamics generated by the map.

Consider the case when $K = 0$. Equation (1) reduces to $p_{n+1} = p_n$ (mod 1) and $x_{n+1} = x_n + p_{n+1}$ (mod 1) which is integrable and every orbit resides on an invariant torus. The orbit can exhibit periodic or quasi-periodic behavior depending on the initial conditions (p_0, x_0). For small values of K e.g. $K = 0.5$ (figure 1(a)) most of these orbits persist, with tiny regions of chaotic dynamics appearing which are not visible on the presented plotting scales. At $K = K_c \approx 0.97$ the last invariant KAM tori are destroyed and a simply connected chaotic sea is formed which allows for unbounded momentum diffusion. For larger values of K the chaotic fraction grows confining regular dynamics to regular islands embedded in a chaotic sea (figure 1). Further increase of K leads to a flooding of the regular islands by the chaotic sea.

3. LEs and predictions

The LE characterizes the exponential rate of separation of a trajectory $\{p_n, x_n\}$ and its infinitesimal perturbation $\{\delta_n, \zeta_n\}$:

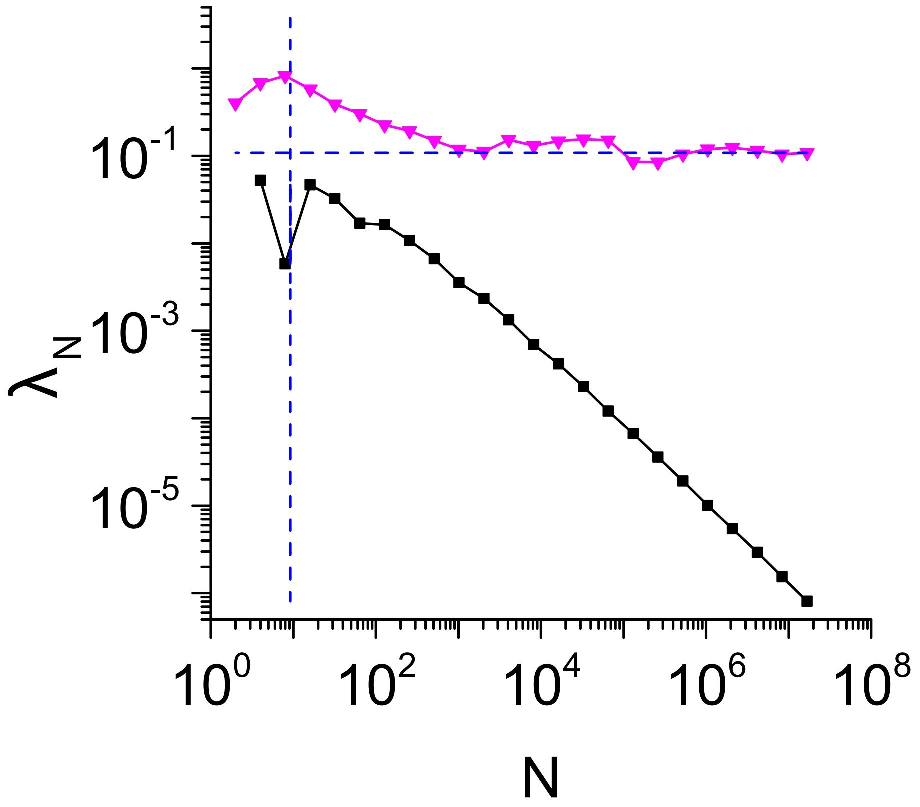

Figure 2. λ_N versus N for a chaotic (triangles) respectively regular (squares) trajectory with $K = 1.0$. The dashed horizontal line indicates the value of λ for the chaotic trajectory, and the dashed vertical one the corresponding value of T_λ.

$$
\begin{aligned}
p_{n+1} + \delta_{n+1} &= (p_n + \delta_n) + \frac{k}{2\pi}\sin(2\pi(x_n + \zeta_n)) \\
x_{n+1} + \zeta_{n+1} &= (x_n + \zeta_n) + (p_{n+1} + \delta_{n+1})
\end{aligned}
\tag{2}
$$

Linearizing (2) in the perturbation yields the tangent dynamics generated by the variational equations

$$
\begin{aligned}
\delta_{n+1} &= \delta_n + k\zeta_n \cos(2\pi x_n) \\
\zeta_{n+1} &= \zeta_n + \delta_{n+1}
\end{aligned}
\tag{3}
$$

For computational purposes δ and ζ can be rescaled after any time step without loss of generality, while keeping the rescaling factor. The LE λ for each trajectory is obtained from the time dependence of λ_N:

$$
\lambda_N = \frac{1}{N}\sum_{n=2}^{N} \ln\left(\frac{\sqrt{\delta_n^2 + \zeta_n^2}}{\sqrt{\delta_{n-1}^2 + \zeta_{n-1}^2}}\right) \; , \; \lambda = \lim_{N\to\infty} \lambda_N \, . \tag{4}
$$

The Lyapunov time is then defined as $T_\lambda \equiv 1/\lambda$. For the main chaotic sea it is a function of the control parameter K. A suitable fitting function yields $\lambda \approx \ln(0.7 + 0.42\ \mathrm{K})$ [12].

For a regular trajectory $\lambda_N \sim 1/N$ and $\lambda = 0$, at variance to a chaotic trajectory for which λ_N saturates at λ at a time $N \approx T_\lambda$. Technically this saturation, and the value of λ can be safely confirmed and read off only on time scales $N \approx 10^2..10^3 T_\lambda$, without becoming a quantifiable distinguisher of the two types of trajectories, see figure 2.

To quantify our statements, we run the standard map at $K = 2.5$ figure 1(d). We use a grid of 51 × 51 points which partitions the phase space $\{p, x\}$ into a square lattice. We use the corresponding 2601 initial conditions and generate trajectories. Each trajectory returns a function λ_N. We plot the resulting histogram for $N = 20$ and $N = 3 \cdot 10^5$ in figures 3(a) and (b), respectively. For $N \to \infty$ the histogram should show two bars only—one at $\lambda_N = 0$ (all regular trajectories) and one at $\lambda_N = \lambda$ (all chaotic trajectories). For finite N the distributions smoothen. Note that even negative values λ_N are generated due to fluctuations and finite averaging times. To tell chaotic from regular dynamics apart, we use the following protocol. We identify the

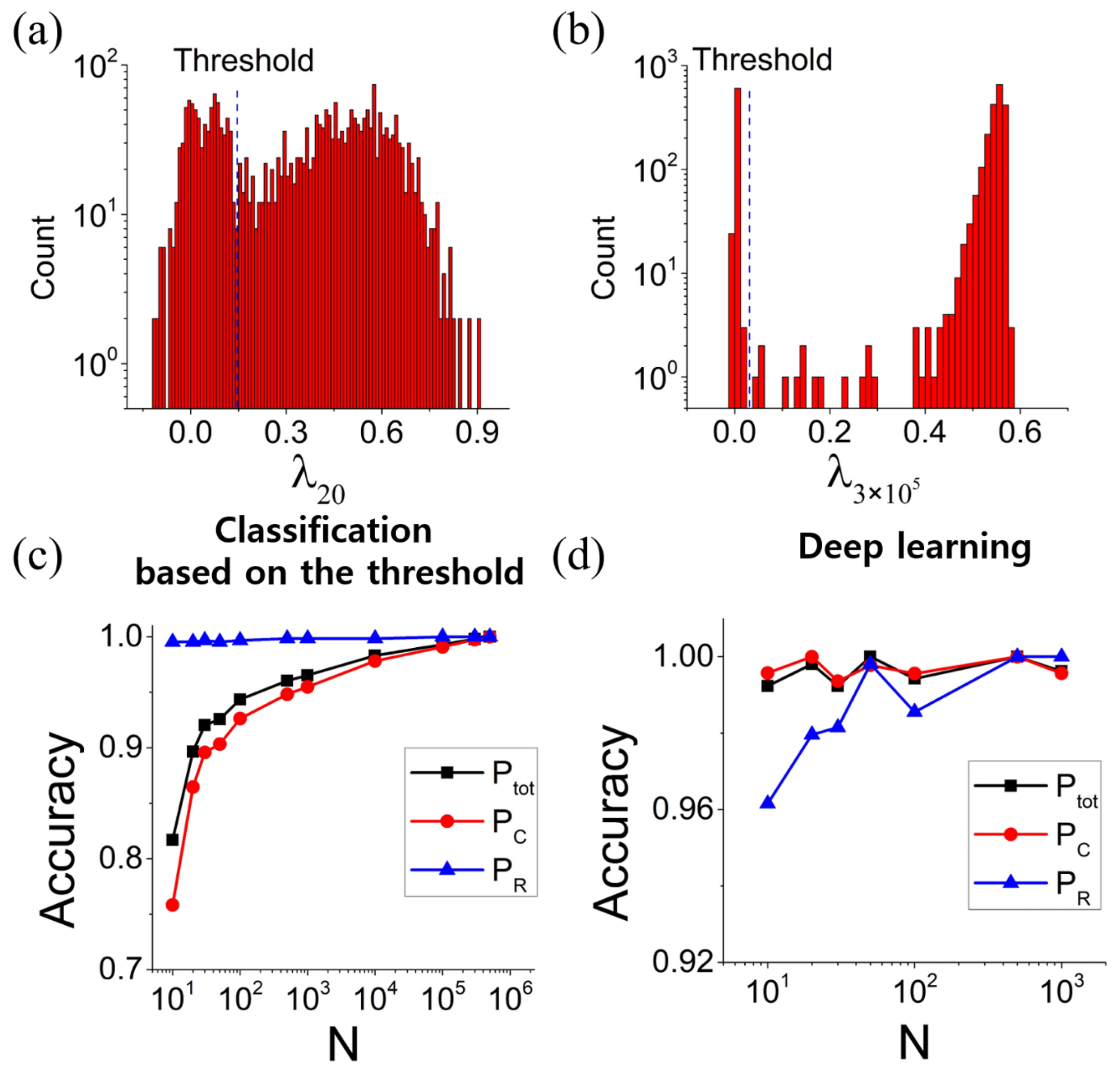

Figure 3. Performance comparison of a Lyapunov exponent based method and a deep learning method to distinguish chaotic and regular trajectories for $K = 2.5$ and $\lambda \approx 0.56$. (a) Histogram of $\lambda_{N=20}$. the dashed vertical line indicates the location of the threshold (see text for details). (b) Same as (a) but $N = 3 \times 10^5$. (c) The success rates P_R, P_C and P_{tot} as a function of N for the Lyapunov exponent based method (see text for details). (d) Same as in (c) but for the deep learning based method. The network was trained for $K = 2.5$ and 2081 trajectories. The remaining 520 trajectories are used for testing. N in (d) represents the trajectory length used for network training and test. $K_{\min} = K_{\max} = 2.5$, $M_{tr} = 2081$, $M_{tt} = 520$, $N_K \equiv N$.

two largest peaks in each histogram, and identify the threshold dividing dynamics into regular and chaotic as the deepest minimum between them (in case of a degeneracy, the one with the smallest value of λ_N). The location of the threshold is shown for $N = 20$ and $N = 3 \cdot 10^5$ in figures 3(a) and (b), respectively. We then assign a chaos respectively regular label to each trajectory. This label can fluctuate as a function of time for any given trajectory. We use the division for the largest simulation time $N = 3 \cdot 10^5$ as a reference ('true') label for all trajectories. The success rate in predicting the correct regular P_R or chaotic P_C label is defined by the ratio of the correctly predicted labels within each subgroup of identical true labels. Likewise the success rate of predicting any label correctly is denoted by P_{tot}. The results are plotted versus time N in figure 3(c). While regular labels are predicted with high accuracy, chaotic ones are reaching 98% at only $N \approx 10^3 T_\lambda$.

The low success rate P_C is therefore also lowering the total success rate P_{tot}.

4. Neural networks and predictions

The input data of an artificial neural network consisting of only fully connected layers are limited to a one-dimensional (array) form [13]. Fully connected layers connect all the inputs from one layer to every activation unit of the next layer. The standard map generates sequences embedded in two dimensions. In order to learn data embedded in dimensions two or larger, the data must be flattened, and spatial information can get lost. A convolutional neural network (CNN) is known to learn while maintaining spatial informations of images [14]. A CNN is usually configured with convolution and pooling layers. The former

Table 1. CNN performance. For each K value, 2601 different initial values $(p_{0,i}, x_{0,j})$ were selected as $(p_{0,i} = (i-1)\frac{1}{50}, x_{0,j} = (j-1)\frac{1}{50},\ (i,j \in \mathbb{Z},\ 1 \leq i,j \leq 51,\ ))$. Other parameters are listed in the main text.

KN_K	20	18	16	14	12	10	2	1
	P_C/P_R	P_C/P_R	P_C/P_R	P_C/P_R	P_C/P_R	P_C/P_R	P_C/P_R	P_C/P_R
3.0	0.99/0.99	0.93/0.98	0.95/0.98	0.92/0.98	0.97/0.96	0.83/0.95	0.89/0.97	0.78/1.0
3.1	0.90/0.98	0.94/0.96	0.96/0.96	0.93/0.96	0.90/0.96	0.83/0.91	0.90/0.93	0.79/1.0
3.2	0.93/0.95	0.94/0.97	0.96/0.97	0.93/0.97	0.97/0.94	0.85/0.91	0.90/0.92	0.79/1.0
3.3	0.97/0.99	0.93/0.99	0.95/0.99	0.93/0.99	0.94/0.96	0.85/0.93	0.89/0.98	0.77/1.0
3.4	0.94/0.99	0.89/0.97	0.94/0.96	0.92/0.98	0.93/0.97	0.82/0.93	0.88/0.98	0.76/1.0
3.5	0.93/0.94	0.93/0.93	0.96/0.88	0.92/0.99	0.92/0.91	0.83/0.92	0.87/0.94	0.76/1.0

employ convolutional integrals with input data and filters to produce output feature maps. An additional activation function turns the network non-linear. At the end of the convolution layers a pooling layer is added which performs value extraction in a given pooling region. Through multiple convolution layers and pooling layers, the network can improve its prediction features.

Finally, a fully connected layer generates classified output data. For binary classification, the last layer consists of one node. Its output value is either zero or one. We refer the reader to appendix A for further technical details of the CNN we use.

4.1. The standard map

The input of the neural network is a time series (p_n, x_n) from equation (1). The trajectory (p_n, x_n) shows regular or chaotic behavior depending on the initial values (p_0, x_0). Each of the trajectories is assigned a class label based on the Lyapunov time: Class R corresponds to a non-chaotic trajectories while C corresponds to a chaotic trajectories. We remind that the phase space is discretized into $51 \times 51 = 2601$ grid points. The training and testing is quantified with a set of parameters: (i) $K_{\min}$ and $K_{\max}$ denote the range of training values of K on an equidistant grid with M_K values; (ii) M_{tr} is the number of training trajectories per K value; (iii) N_K is the training trajectory length; (iv) M_{tt} is the number of test trajectories per K value.

To quantify the CNN performance, we assign a discrete label to each of the initial phase space points—C respectively R based on the LE method with trajectory length $N = 3 \cdot 10^5$. This way we separate all phase space points into two sets—C and R, each containing A_C and A_R points. We then run the CNN prediction on trajectories of length $N = 20$ which start from each of the gridded phase space points. We compute the accuracy quantifying probabilities

$$P_C = \frac{B_C}{A_C},\ P_R = \frac{B_R}{A_R},\ P_{tot} = \frac{B_C + B_R}{A_C + A_R} \tag{5}$$

where B_C and B_R are the numbers of trajectories predicted by the CNN to be chaotic respectively regular within each of the true sets A_C and A_R. Thus strictly $B_C \leq A_C$ and $B_R \leq A_R$.

Figure 3(d) compares the CNN performance to the standard Lyapunov base one. Accuracies of 98% and more are reached by the CNN for trajectory length $N_K \geq 30$. Similar accuracies need trajectory length $N \approx 10^4$ and more when using standard Lyapunov testing. Figure 4 shows the CNN performance with $N_K = 10$ in the phase space of the standard map. We observe that most of the failures correspond to chaotic trajectories starting in the fractal border region close to regular islands. These trajectories can be trapped for long times in the border region, with trapping time distributions exhibiting power law tails [15].

To quantify the performance of the CNN, we first vary the N_K from 1 to 20 (table 1). The network is trained with chaotic and regular trajectories for $K_{\min} = 1.0$, $K_{\max} = 2.0$, $M_K = 11$, and $1 \leq N_K \leq 20$ and the network performance is evaluated for $3 \leq K \leq 3.5$ and $M_K = 6$. The CNN requires that the length of test trajectories is always kept equal to the length of the training trajectories. Note that the Lyapunov time $T_\lambda \approx 2$ for the test values of K. The CNN shows improvement of the accuracy with increasing N_K. While the performance fluctuates with varying K, it shows excellent results for N_K values and clearly outperforms the LE based method.

We then further test the CNN performance for untrained K values by varying the training K range and other relevant training parameters in figure 5. The network shows better performance on untrained K values when trained with a set of different K values. As expected, smaller numbers of training K values yield poorer accuracy due to overtraining. With increasing training range of K values and ranges the network improves its chaos region predictions for untrained K values.

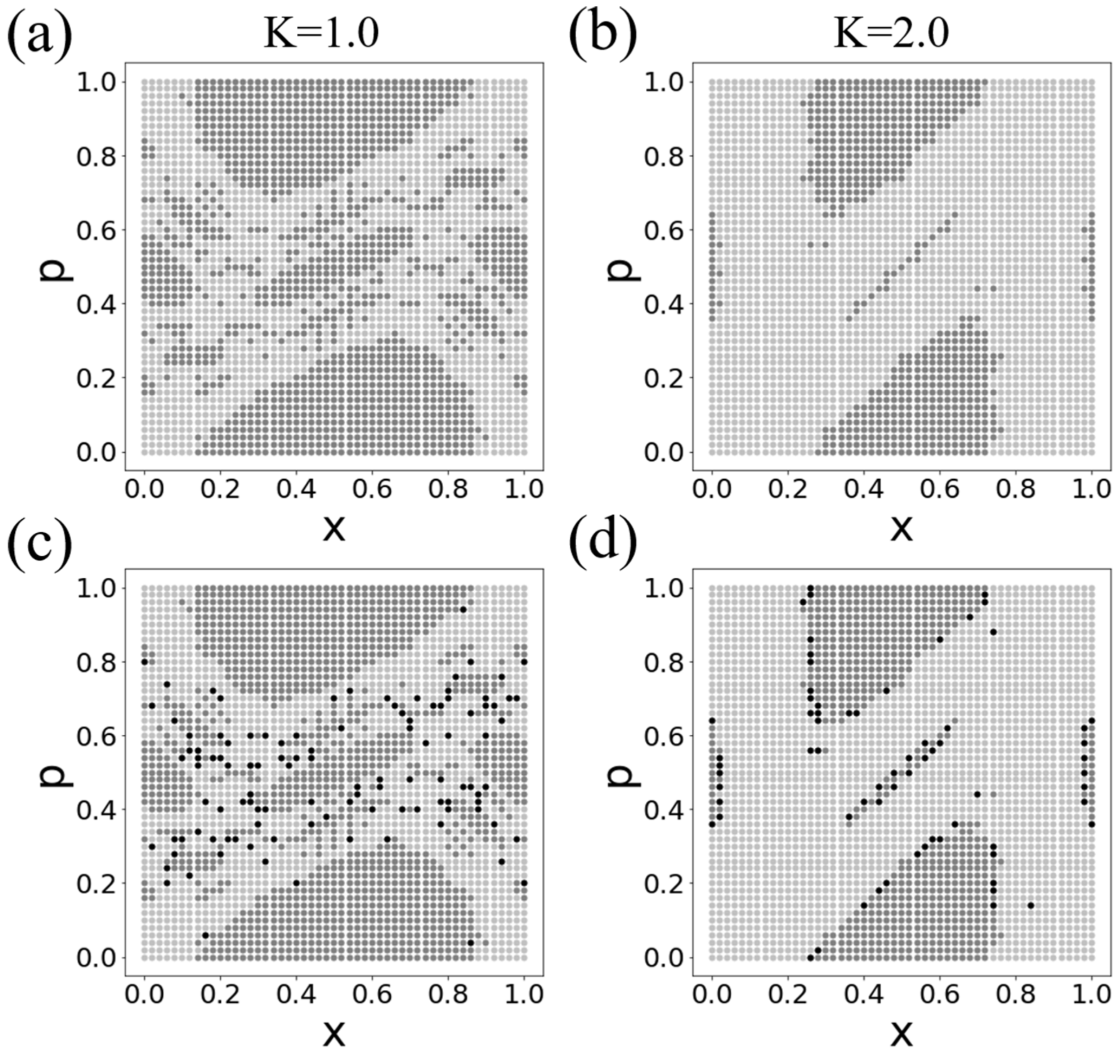

Figure 4. Chaos classification in the standard map. The Lyapunov exponent classification with trajectory length $N = 3\cdot10^5$ is used as a reference classifier for $K = 1$ (a) and $K = 2$ (b). The CNN test results are shown for $K = 1$ (c) and $K = 2$ (d). Open circles—regular, gray circles—chaotic. Black circles show the error locations of the CNN prediction. The CNN parameters are $K_{\min} = 1.0$, $K_{\max} = 2.0$, $M_K = 11$, $M_{tr} = 2081$, $M_{tt} = 520$, $N_K = 10$.

4.2. Training with the standard map, testing the logistic map

We proceed with testing how the CNN trained with standard map data performs in predicting chaos for other maps. We choose the logistic map as a simple one-dimensional chaotic test bed. The logistic map is written as $x_{n+1} = rx_n(1-x_n)$. The parameter r controls the crossover from regular to chaotic dynamics, which happens at $r_c\approx 3.56995$. We choose the initial condition $x_0 = 0.4$ and iterate over 40 steps. After that we record the next 110 x-values and plot them for each value of r in figures 6(a) and (b). We observe the well-known periodic orbits, the period doubling bifurcations, and the route to chaos. The corresponding LE is plotted in figure 6(c) as a function of r and shows that periodic orbits stay regular (negative LE) while chaotic attractors show positive LEs.

We use two training methods. The first one trains the network only with the p_n data sequence from the standard map in equation (1). We coin that trained network 1D. In that case we used a sequence of $N_K = 20$ consequtive logistic map iteration data $x_n, \ldots, x_{n+19}$ as test data.

The second training method is the original CNN one discussed above for the standard map, coined here 2D. The network input trained by the standard map has the shape of an array with two columns each of length $N_K = 20$ (see appendix A). We write $N_K = 20$ consecutive logistic map iteration data into the first column, and the next $N_K = 20$ ones into the second column to obtain a test input vector.

As shown in figure 6, the network mainly generates errors at the boundary between the chaos and regular regions similar to the standard map. For $2.5 \leq r \leq 4.0$ the accuracy is 84% for 2D network and 90% for the 1D network.

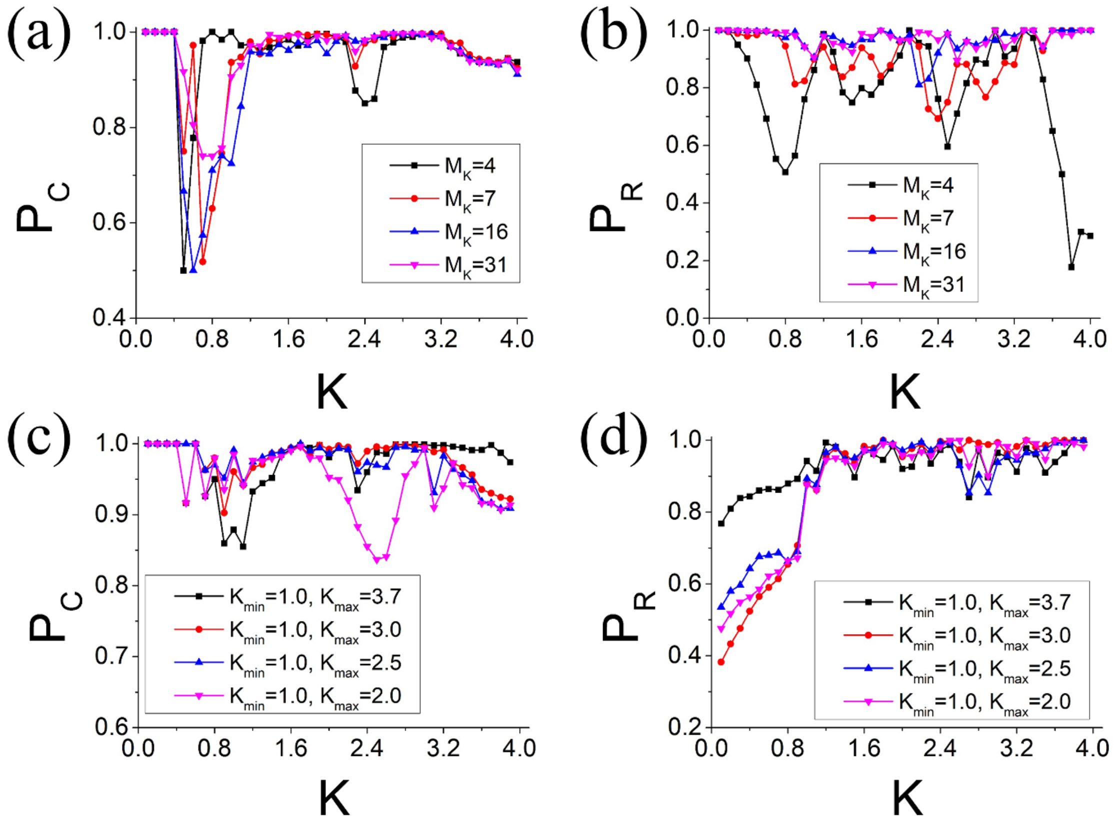

Figure 5. Network performance versus K for different trained K value numbers and ranges. (a), (b) Varying the number of K values used for network training in a fixed interval with equidistant spacing ($K_{\min} = 0.1,\ K_{\max} = 3.1$ $M_{tt} = 2601, N_K = 20$). (black square) $M_K = 4$. (red circle) $M_K = 7$. (blue triangle) $M_K = 16$. (magenta inverted triangle) $M_K = 31$. (c), (d) Varying the interval of trained K values. The range of K values used in network learning are (black square) $K_{\min} = 1.0,\ K_{\max} = 3.7,\ M_K = 28$, (red circle) $K_{\min} = 1.0,\ K_{\max} = 3.0,\ M_K = 21$, (blue triangle) $K_{\min} = 1.0,\ K_{\max} = 2.5,\ M_K = 16$, (magenta inverted triangle) $K_{\min} = 1.0,\ K_{\max} = 2.0,\ M_K = 11$. The length of the input trajectories are 20.

4.3. Training with the standard map, testing the Lorenz system

Next we test the Lorenz system—a set of three coupled non-linear ordinary differential equations. We discretize time to arrive at a three-dimensional map:

$$\begin{aligned} X_{n+1} &= X_n + \sigma\Delta(Y_n - X_n), \\ Y_{n+1} &= Y_n + \rho\Delta X_n - \Delta X_n Z_n - \Delta Z_n, \\ Z_{n+1} &= Z_n + \Delta X_n Y_n - \beta\Delta Z_n. \end{aligned} \tag{6}$$

We use a CNN trained on the two-dimensional standard map. The parameters $\sigma = 10$ and $\beta = \frac{8}{3}$. The time step $\Delta = 0.001$ measures the discretized time interval length when replacing the original differential equations with a map.

The chaos parameter $0 \leq \rho \leq 39.8$ was varied in steps of 0.2. Because the network is trained with 2D data (standard map), the prediction is performed by selecting only two dimensions in the 3D Lorenz system $((X_n, Y_n), (X_n, Z_n), (Y_n, Z_n))$. As figure 7(a) shows, using trajectories obtained from equation (6) directly as a network input classifies most of them as chaotic. We think this happens because the trajectory data of the standard map used for training are bounded between 0 and 1, but the trajectories from Lorenz system are not. Input values that exceed these boundaries cause nodes in the network to be active regardless of the input characteristics. Therefore we normalize the input data from the Lorenz system. This leads to a drastic increase of accuracy as shown in figure 7(b). We also tested the outcome when selecting only one dimension in the Lorenz system for the input vector. We find a strong reduction of the accuracy. We therefore conclude that the training and testing data are yielding best performance when for both the minimum of the two dimensions (training map, testing map) is chosen.

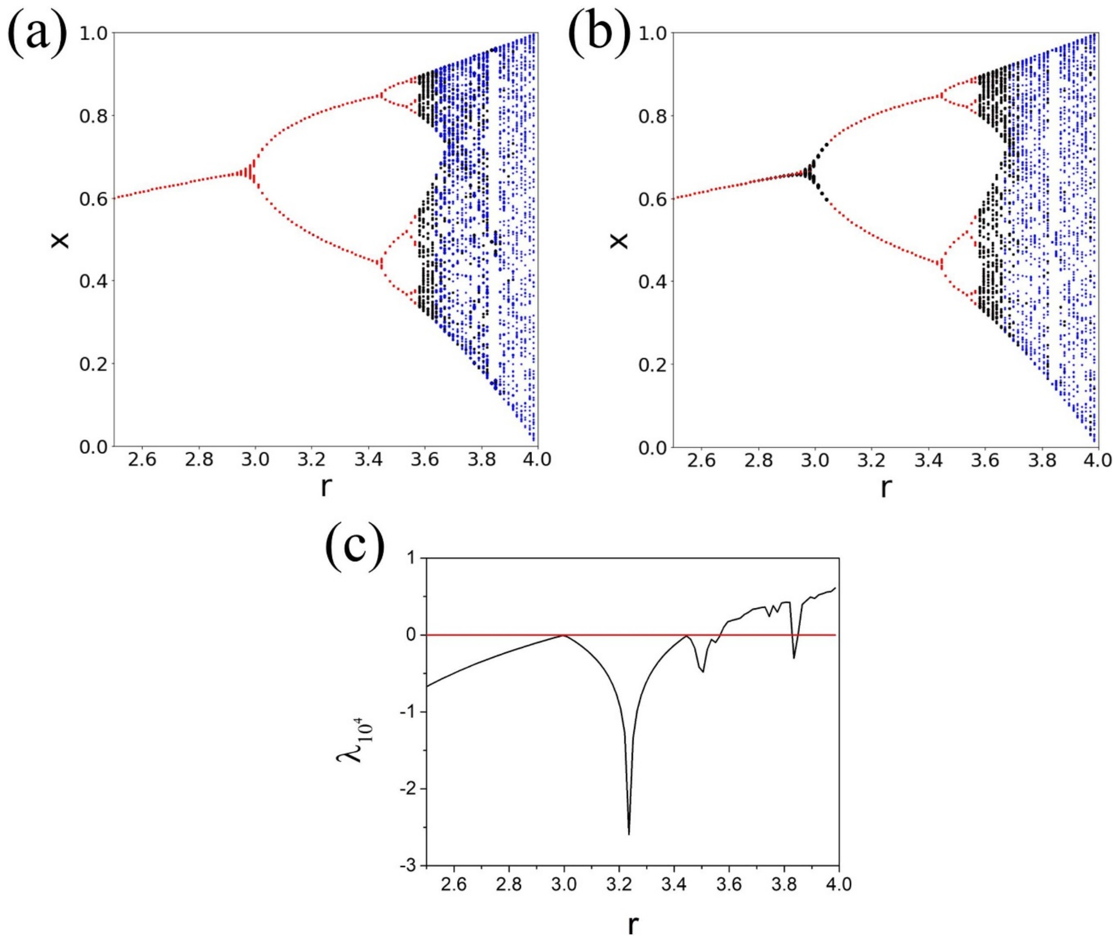

Figure 6. The result of predictions for the logistic map with a network trained from the standard map. The blue and red dots in bifurcation diagram are the cases where the network correctly predicts chaotic and regular attractors, respectively. The black dots show where the prediction fails. The network is trained with $K_{\min} = 1.0,\ K_{\max} = 2.0,\ M_K = 11,\ M_{tr} = 2081,\ M_{tt} = 520,$ and $N_K = 20$. (a) Test results for the 2D training (see text for details). (b) Test results for the 1D training (see text for details). (c) Lyapunov exponents for the logistic map. The red line marks $\lambda = 0$.

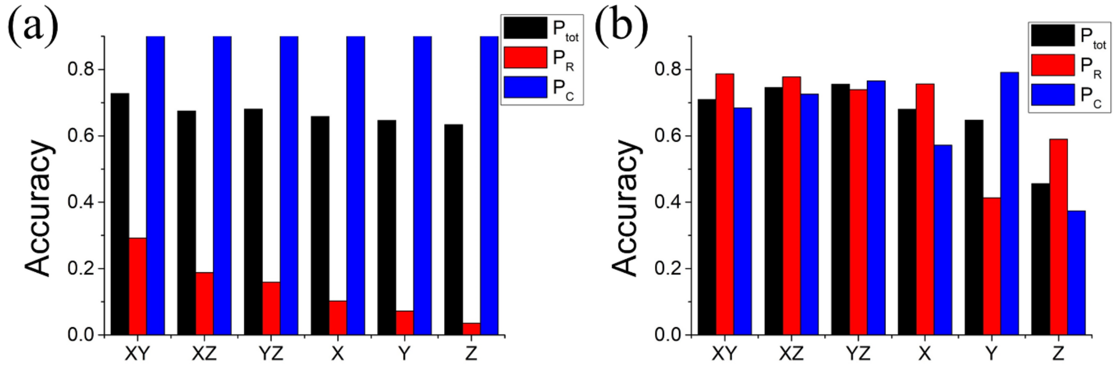

Figure 7. The result of predictions for the Lorenz system with a network trained from the standard map. The XY, XZ, YZ bars represent the dimensions of the Lorenz system used as input to the network trained with (p, x) data from the standard map. The training conditions are $N_K = 20$, $K_{\min} = 1.0$, $K_{\max} = 2.0$, and $M_K = 11$. The X, Y, Z bars represent the single dimensions of the Lorenz system used as input to the network trained with p data only from the standard map. (a) Accuracy without normalizing the trajectories of the Lorenz system. (b) Accuracy when normalizing trajectories of the Lorenz system.

5. Conclusion

We trained CNNs with time series data from the two-dimensional standard map. As a result, the network can classify unknown short trajectory sequences into chaotic or regular with high accuracy. To reach accuracies of up to 98% we need trajectory segments with length less than 5–10 Lyapunov times. Similar accuracies need 100–1000 longer segments when using traditional classifiers based on measuring LEs. The main cause of errors is due to fractal phase space structures at the boundaries between chaotic and regular dynamics. Trajectories launched in these regions yield sticky trajectories which can mimic regular ones for long times, only to escape at even larger times into the chaotic sea. We also used a network trained with two-dimensional standard map data to classify chaotic and regular dynamics in one- and three-dimensional maps. Surprisingly high accuracy is reached when the training data are projected into one dimension for predictions on the one-dimensional logistic map, and when to-be-predicted data from the three-dimensional Lorenz system are projected onto two dimensions. We conclude that accuracy is optimized when the minimum of the two dimensions (training map, testing map) is chosen for both training and testing.

Acknowledgments

This work was supported by the Institute for Basic Science, Project Code IBS-R024-D1. SF thanks Konstantin Kladko for discussions during a visit to IBS, which led to the main idea of machine learning based chaos testing, and Natalia Khotkevych for early attempts to figure a realization pathway.

Data availability

The data that support the findings of this study are available upon reasonable request from the authors.

Appendix A. Details of the neural network structure

The neural network model we use to analyze the chaotic pattern is the convolutional neural network (CNN) [14] with a fully connected (FC) network [16, 17]. The required non-linear response of the system is provided by the rectified linear unit (ReLU) [17, 18]. For supervised learning, we use a cross entropy as loss function [17]. Figure A1 shows one of the CNN structures used here. In the figure, 1024 filters in the first layer scan the input data independently, then yield 1024 feature maps which are used as the input data for the second layer after applying the activation function ReLU. After processing through all convolutional layers, we rearranged the pixels of the last feature maps into one-dimensional data for the fully connected layers. In the last layer we set the desired output: 1 is a chaotic, and 0 is a regular trajectory.

We use a two-dimensional convolutional filter. The data comprise multiple spatial channels and each channel gives time-series data. The relation between the 2D input vector a and the convolution layer output vector z is

$$z^{(\ell)}_{i,j,m} = \sum_{q=1}^{F_{row}} \sum_{p=1}^{F_{col}} w^{(\ell)}_{p,q,m} a^{(\ell-1)}_{(i+p-1),(j+q-1)} + b^{(1)}_{m}. \tag{A1}$$

After calculating z, the non-linear activation function (we used ReLU) is used to obtain the value of a of the next layer:

$$a^{(\ell)}_{i,j,m} = \mathrm{ReLU}(z^{(\ell)}_{i,j,m}), \tag{A2}$$

where i, j are input element indices, p, q are filter element indices, m is the filter index (e.g. $m = 1024$ means that 1024 filters of size $F_{\text{row}} \times F_{\text{col}}$ were used), and ℓ is the layer index (the indices of input and first layer are $\ell = 0$ and $\ell = 1$, respectively). The weight $w^{(\ell)}_{p,q,m}$ is the (p, q)th element of mth filter. The bias b_m is a constant. The filter size is $F_{\text{row}} \times F_{\text{col}}$, where we chose $F_{\text{row}} = 2$ and $F_{\text{col}} = 1$. To apply the filter to all the elements of the input, the boundary is filled with zeroes to match the size of the input, which is called zero padding [19]. The 2D input through the convolution layer has a dimension of (i, j, m) due to the number of filters m in the convolution layer. Accordingly, the input / output relationship of the next layer is as follows:

$$z^{(\ell)}_{i,j,n} = \sum_{n=1}^{M_\ell} \sum_{q=1}^{F_{row}} \sum_{p=1}^{F_{col}} w^{(\ell)}_{p,q,n} a^{(\ell-1)}_{(i+p-1),(j+q-1),m} + b^{(\ell)}_{n}, \tag{A3}$$

where M_ℓ is the number of filters between the ℓth and $(\ell - 1)$th layers, ℓ has a range of 1 to L. After convolution, it processes through the activation function as shown in equation (A2).

As shown in figure A1, there is a pooling layer at the end of the convolution layers. This flattens the convolutional output by finding the maximum according to the filter dimension of the input (equation (A4)).

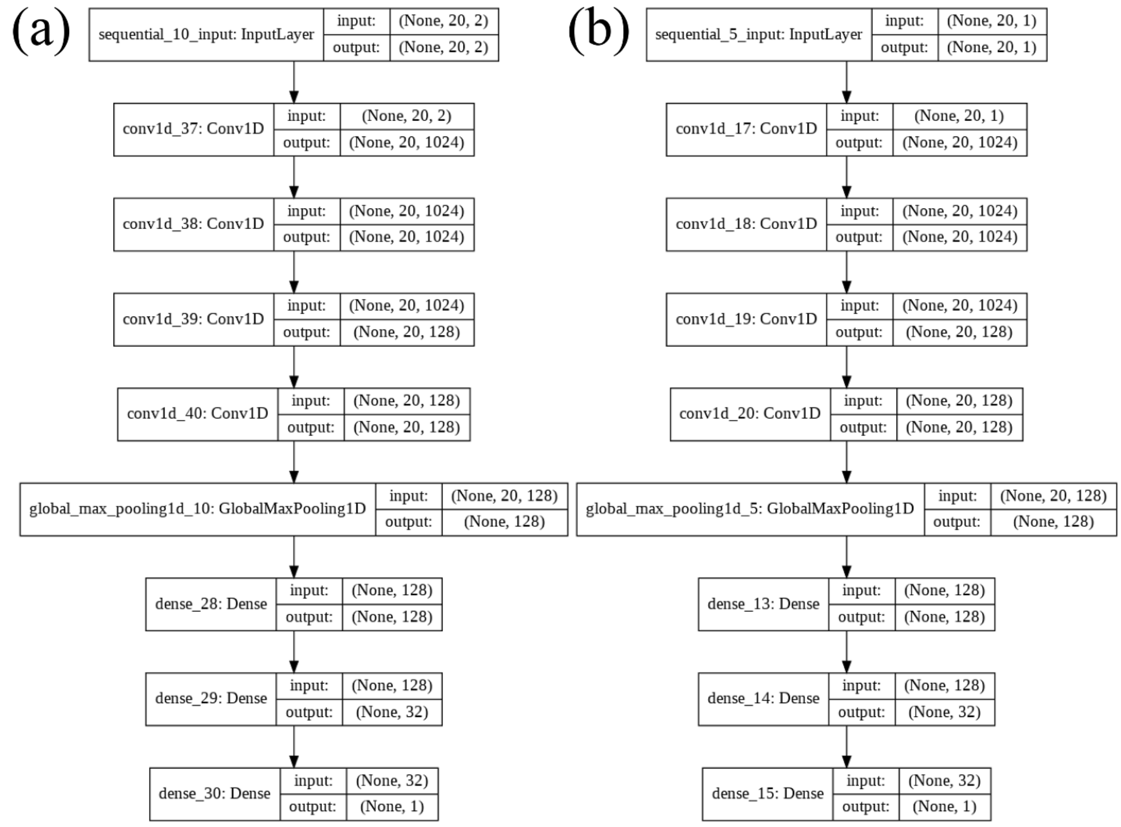
(a)
sequential_10_input: InputLayer | input: (None, 20, 2) | output: (None, 20, 2)
conv1d_37: Conv1D | input: (None, 20, 2) | output: (None, 20, 1024)
conv1d_38: Conv1D | input: (None, 20, 1024) | output: (None, 20, 1024)
conv1d_39: Conv1D | input: (None, 20, 1024) | output: (None, 20, 128)
conv1d_40: Conv1D | input: (None, 20, 128) | output: (None, 20, 128)
global_max_pooling1d_10: GlobalMaxPooling1D | input: (None, 20, 128) | output: (None, 128)
dense_28: Dense | input: (None, 128) | output: (None, 128)
dense_29: Dense | input: (None, 128) | output: (None, 32)
dense_30: Dense | input: (None, 32) | output: (None, 1)
(b)
sequential_5_input: InputLayer | input: (None, 20, 1) | output: (None, 20, 1)
conv1d_17: Conv1D | input: (None, 20, 1) | output: (None, 20, 1024)
conv1d_18: Conv1D | input: (None, 20, 1024) | output: (None, 20, 1024)
conv1d_19: Conv1D | input: (None, 20, 1024) | output: (None, 20, 128)
conv1d_20: Conv1D | input: (None, 20, 128) | output: (None, 20, 128)
global_max_pooling1d_5: GlobalMaxPooling1D | input: (None, 20, 128) | output: (None, 128)
dense_13: Dense | input: (None, 128) | output: (None, 128)
dense_14: Dense | input: (None, 128) | output: (None, 32)
dense_15: Dense | input: (None, 32) | output: (None, 1)

Figure A1. Network architecture for chaos classification, consisting of four convolution layers, and three fully-connected layers. The filter size and output size are provided for each layer. Network structure for (a) 2D and (b) 1D shape training.

$$a_m^{(\ell)} = \max(a_{i,j,m}^{(\ell)}). \tag{A4}$$

The reason for flattening the output is to use the convolutional output as the input of the fully connected layer:

$$z_i^{(\ell)} = \sum_{j=1}^{K} w_{j,i}^{(\ell-1)} a_j^{(\ell-1)}, \tag{A5}$$

where $w_{i,j}$ is a weighted connection between the jth component of $(\ell-1)$th layer and the ith component of (ℓ)th layer and K is the number of nodes in the $(\ell-1)$th layer. As in equation (A6), the activation function uses ReLU:

$$a_i^{(\ell)} = \mathrm{ReLU}(z_i^{(\ell)}). \tag{A6}$$

The output value of the network, when obtained in this way, is different from the desired output because it is obtained from the unfitted w value. w updates in the direction of reducing this difference and we call this difference the loss or cost. In this work, we selected the cross entropy as the loss function, defined as

$$C = -\sum_{k=1}^{N_L} (a_k^{\mathrm{true}} \log(a_k^{(L)}) + (1 - a_k^{\mathrm{true}}) \log(1 - a_k^{(L)})), \tag{A7}$$

where N_L is the number of nodes in the output layer and a_k^{true} is the desired output at the kth node. The cross-entropy loss function can reflect the degree of error to weight updates better than the mean square error loss function (MSE, MSE$= \frac{1}{N_L} \sum_{k=1}^{N_L} (a_k^{\mathrm{true}} - a_k^{(L)})^2$) because of the log term and is known as a cost function suitable for classification problems [20]. Similar to the energy minimization problem in physics, supervised learning minimizes a cost function C at the output layer.

Table B1. Performance for different deep learning classifiers. The networks are trained with chaotic and regular trajectories for $K_{\min} = 1.0,\ K_{\max} = 2.0,\ M_K = 11,\ M_{tr} = 2601$ and $N_K\ =\ 20$. The network performances are evaluated for $K_{\min} = 3.0,\ K_{\max} = 3.5,\ M_K = 6,\ M_{tt} = 2601$ and $N_K\ =\ 20$. For each K value, 2601 different initial values $(p_{0,i}, x_{0,j})$ were selected as $(p_{0,i} = (i-1)\frac{1}{50}, x_{0,j} = (j-1)\frac{1}{50},\ (i,j \in \mathbb{Z},\ 1 \leq i,j \leq 51,\ ))$.

Classifiers	P_{tot}	P_C	P_R
FCN	0.89	0.88	0.91
SimpleRNN	0.94	0.96	0.92
GRU	0.95	0.96	0.93
LSTM	0.94	0.96	0.93
CNN	0.96	0.94	0.97

Training the neural network means finding optimized parameters $w^{(\ell)}_{pqm}$ and $b^{(l)}_n$ that minimize C. We use an Adaptive Moment Estimation (Adam) algorithm for the optimization of learning [21]. As far as we know, the choice of optimization algorithm has little impact on the network performance, but is instead mainly related to the speed of learning. It is known that an Adam algorithm can find fitting variables faster than stochastic gradient descent methods, RMSprop and AdaDelta [21].

Appendix B. Network comparison

In this section, we compare three different network architectures. First, we consider a fully connected neural network (FCN) and then compare it to other machine learning methods. The FCN is chosen because it is the most basic structure of deep learning classifiers. We also consider a recurrent neural network (RNN) [22]. The RNNs are usually used to deal with temporal dynamic behavior. We consider supervised classification with labels indicating chaos or regularity, where the input of the neural networks is fixed at $N_K\ =\ 20$, and the output is one node for the corresponding label for training. The neural networks presented in this paper end with a sigmoid layer.

The first type of network considered in this section is a fully connected network, which consists of multiple fully connected layers and each layer has a non-linear activation function. We use eight hidden layers with ReLU (Rectified Linear Unit) activation and the number of hidden neurons in each layer is [256, 256, 512, 512, 512, 256, 128, 64]. The network is trained with the Adaptive Moment Estimation (Adam) algorithm [21].

Recurrent neural networks (RNN) are neural networks for processing sequential data. RNN uses the current input as well as any previously processed input. This is possible with a loop structure between the RNN input and the output. Each node in a given layer is connected with a directed connection to the current layer. Because of this, the RNN is expected to have a function of memory. The sequence itself has information, and recurrent networks use this information through the loop structure. We use three type of RNNs: simpleRNN [22], LSTM [22, 23], and GRU [24]. Three RNN cells(layers) were used, each with 200 hidden neurons. After the RNN cells, three fully connected layers are connected with the size of 200, 100 and 32, respectively. It is known that recurrent networks perform well for sequential data, but at least in our data sets there was no significant difference between using CNN and RNN.

ORCID iD

Woo Seok Lee https://orcid.org/0000-0002-9381-1253

References

[1] Skinner J E, Goldberger A L, Mayer-Kress G and Ideker R E 1990 Chaos in the heart - implications for clinical cardiology *Nat. Biotechnol.* **8** 1018–24

[2] Slingo J and Palmer T 2011 Uncertainty in weather and climate prediction *Phil. Trans. R. Soc.* A **369** 4751–67

[3] Ott E 2002 *Chaos in Dynamical Systems* 2nd edn (Cambridge: Cambridge University Press)

[4] Dodge S and Karam L 2017 A study and comparison of human and deep learning recognition performance under visual distortions *2017 26th International Conference on Computer Communication and Networks (ICCCN)* United States Institute of Electrical and Electronics Engineers Inc pp 1–7

[5] Mohammed Al-Saffar A A, Tao H and Talab M A 2017 Review of deep convolution neural network in image classification *2017 Int. Conf. on Radar, Antenna, Microwave, Electronics and Telecommunications (ICRAMET)* United States Institute of Electrical and Electronics Engineers Inc pp 26–31

[6] Rudy S H, Brunton S L, Proctor J L and Nathan Kutz J 2017 Data-driven discovery of partial differential equations *Sci. Adv.* **3** 4

[7] Han J, Jentzen A and Weinan E 2018 Solving high-dimensional partial differential equations using deep learning *Proc. Natl. Acad. Sci.* vol 115 pp 8505–10

[8] Raissi M and Karniadakis G E 2018 Hidden physics models: Machine learning of nonlinear partial differential equations *J. Comput. Phys.* **357** 125–41

[9] Agrawal S, Barrington L, Bromberg C, Burge J, Gazen C and Hickey J 2019 Machine learning for precipitation nowcasting from radar images arXiv:1912.12132

[10] Pathak J, Hunt B, Girvan M, Zhixin L and Ott E 2018 Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach *Phys. Rev. Lett.* **120** 024102

[11] Lichtenberg A J and Lieberman M A 2013 *Regular and Chaotic Dynamics* vol 38 (New York: Springer)

[12] Harsoula M, Karamanos K and Contopoulos G 2019 Characteristic times in the standard map *Phys. Rev.* E **99** 032203

[13] Ramsundar B and Zadeh R B 2018 *Tensorflow for Deep Learning: From Linear Regression to Reinforcement Learning* (Sebastopol: O eilly Media)

[14] LeCun Y, Haffner P, Bottou L and Bengio Y 1999 *Object Recognition With Gradient-Based Learning* (Berlin, Heidelberg: Springer Berlin Heidelberg) pp 319–45

[15] Zaslavsky G M 1998 *Physics of Chaos in Hamiltonian Systems* (London: Imperial College Press)

[16] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks *Advances in Neural Information Processing Systems 25* eds Pereira F, Burges C J C, Bottou L and Weinberger K Q (New York: Curran Associates, Inc.) pp 1097–105

[17] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* Adaptive Computation and Machine Learning series (Cambridge: MIT Press)

[18] Nair V and Hinton G E 2010 Rectified linear units improve restricted Boltzmann machines *Proc. of the 27th International Conference on Machine Learning (ICML-10)* Madison Omnipress pp 807–14

[19] Dumoulin V and Visin F 2016 A guide to convolution arithmetic for deep learning arXiv: 1603.07285

[20] Nielsen M A 2015 *Neural Networks and Deep Learning* (San Francisco: Determination Press)

[21] Kingma D P and Jimmy B 2015 Adam: A method for stochastic optimization eds Bengio Y and LeCun Y *3rd Int. Conf. on Learning Representations, ICLR 2015 San Diego, CA, USA, May 7- 9, 2015, Conf. Track Proc.*

[22] Sherstinsky A 2020 Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network *Physica* D **404** 132306

[23] Gers F A, Schmidhuber J and Cummins F 1999 Learning to forget: Continual prediction with LSTM *Neural Comput.* **12** 2451–71

[24] Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H and Bengio Y 2014 Learning phrase representations using RNN encoder–decoder for statistical machine translation *Proc. of the 2014th Conf. on Empirical Methods in Natural Language Processing (EMNLP)* Doha Association for Computational Linguistics pp 1724–34 October