Chaotic Modeling and Simulation (CMSIM) 2: 163–173, 2020

Reservoir-computing machine-learning algorithms as observers of spatio-temporal chaos

Lucas Illing and Noah Shofer

Department of Physics, Reed College, Portland, Oregon, 27708, USA (E-mail: illing@reed.edu)

Abstract. We demonstrate that reservoir computing can be utilized as an observer not only of chaotic temporal dynamics, such as those produced by the Rössler system, but also for two-dimensional dynamics generated by an optoelectronic system. Our optoelectronic experimental system consists of a spatial-light modulator with self-feedback that generates complex two-dimensional spatio-temporal patterns. The observer successfully cross-predicts the dynamics at all spatial locations based on observed time series from a selected subset of locations. The observer consists of reservoir computing subnetworks that receive input and predict local regions in space only, making the proposed observer resource efficient.
Keywords: Optoelectronic Systems, Spatio-temporal Chaos, State Observer, Chaotic modeling, Recurrent Neural Networks, Reservoir Computing, Simulation, Chaotic simulation..

1 Introduction

A common problem in nonlinear dynamics is the observer problem; the reconstruction of all system variables from a smaller set of known variables. Typically, observers are used if some but not all state variables of a dynamical system of interest can be measured and a mathematical model is known; an observer using the mathematical model may then be used to reconstruct all dynamical state variables from the available time series. For nonlinear systems this task is generally nontrivial and a variety of estimation methods exist, including Kalman filters [1], synchronization schemes [2], a path integral formalism [3], and adaptive observers [4–7].

Alternatively, black-box machine learning techniques can be used as observers. The data that is necessary to train such a machine learning algorithm can either be generated by a mathematical model that captures the process of interest, if such a model is available, or training data can be experimental data, if data of all state variables is obtained through extensive measurements. Either way, after training, neither model nor complete experimental data is required for the reconstruction of dynamical variables from new observations.

Received: 30 March 2019 / Accepted: 15 January 2020

ISSN 2241-0503

Neural networks have been very successful in a variety of tasks, including tasks such as image classification and speech recognition [8]. One particular subset of neural network architectures, known as reservoir computing, is especially suited for time-series data generated by dynamical systems and has been used successfully for short-term prediction, attractor reconstruction, and observation of chaotic systems.

Reservoir computing was first proposed independently by Jaeger [9] as echo state networks and Maass *et al.* [10] as liquid state machines, before merging into a single field. Reservoir computing belongs to the class of recurrent neuronal networks, networks in which the internal nodes are connected recurrently instead of in layers, which makes such networks a high dimensional (driven) dynamical system [11]. The structure of the internal connections is chosen independent of the task and remains unaltered by the training process. For any given task, the reservoir is given an input time series and dynamically generates an output. The task-specific desired output is achieved during a training period by adjusting the post-processing function that maps the state of the internal nodes to the output variables. Typically, the training is done by linear regression. From a dynamical systems point of view, reservoir computers are driven systems that exhibit generalized synchronization between the reservoir internal dynamics and the dynamical system that generates the input data [12].

Recently, several authors have used reservoir computers as observers for nonlinear dynamical systems [13,14], such as the Lorenz system, as well as coupled-map-lattices with several variables per spatial location [15]. Nevertheless, observer inference of dynamical systems with two-dimensional spatial extent is still a challenge.

In this paper, we first explore what aspects of reservoir computing are necessary to successfully infer unobserved variables of a temporally chaotic system, the Rössler system, and then present a method that uses a group of reservoir computers to serve as a resource efficient observer of the two-dimensional coupled-map-lattice dynamics of an optoelectronic system.

2 Spatio-temporal dynamics generated by an optoelectronic system

We generate spatio-temporal dynamics with an experimental realization of a coupled map lattice. In our experiment, a liquid-crystal spatial light modulator is used to control, in a spatially resolved fashion, the polarization of a laser beam and thereby modulate the beam's irradiance. The coupling between different spatial locations across the laser beam is achieved through electronic feedback, in a fashion similar to previous work by Hagerstrom *et al.* [16].

The optoelectronic system that generates the coupled-map-lattice dynamics is shown schematically in Fig. 1. A continuous-wave beam from a helium-neon laser is used as light source. After controlling the size and brightness of the beam, the polarization state of the light across the transverse beam profile is controlled in a spatially resolved fashion by a combination of a $\lambda/4$ wave plate and a spatial light modulator. The spatially resolved polarization state

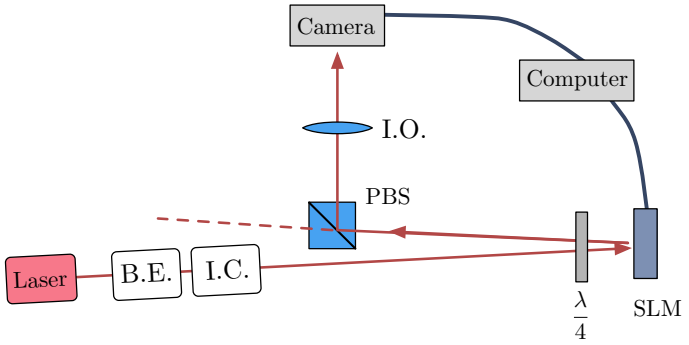

Fig. 1. Schematic of the optoelectronically experiment that generates lattice dynamics. The light source is a helium neon laser. The beam size and brightness are set by a beam expander (B.E.) and an irradiance controller (I.C.). The polarization is set by a combination of a quarter-wave plate ($\lambda/4$) that is oriented at 45° degree with respect to the linear polarization of the incoming light and a 16-bit Meadowlark Optics liquid-crystal spatial-light-modulator (SLM) with 512×512 pixels and a full 2π phase stroke. The beam then passes through a polarizing beam splitter (PBS) and imaging optics (I.O.) that images the front face of the spatial light modulator onto the imaging sensor of a Thorlabs DCC1545M monochrome camera. The image is read real-time by a computer and the computer sets the phase shift of the SLM pixels.

is then converted to a spatially resolved irradiance variation by a polarizing beam splitter that selects one linear polarization and passes it on to a camera that records the irradiance.

To create the two-dimensional coupled-map-lattice sites, the center region of both the spatial light modulator and the camera are partitioned into an 16×16 array of square regions. In our setup, each of the 16×16 lattice sites corresponds to a 10×10 pixel patch on the spatial light modulator, which is sharply imaged onto a corresponding 23×23 pixel patch on the camera. At each iteration, t, the spatial-light-modulator phase-shift of each pixel in each one of the 16×16 patches is set to some phase $\phi^t_{i,j}$ $(i, j = 1, 2, \ldots, 16)$ and the polarization optics then results in the following nonlinear relationship between this phase-shift and the irradiance $\mathcal{I}$ that is detected by the camera,

$$\mathcal{I}(\phi^t_{i,j}) = \frac{1}{2}\left(1 - \cos(\phi^t_{i,j})\right). \tag{1}$$

An example of an image of a 6×6 lattice, as seen by the camera, is shown in Fig. 2. The irradiance $\mathcal{I}(\phi^t_{i,j})$ is calculated from such an image by averaging the 0-255 grayscale values of the 23×23 camera pixels that comprise a single lattice site.

Finally, to implement lattice dynamics, the spatial-light-modulator phase-shift at iteration $t + 1$ is obtained from on the detected irradiances at time t. Lattice sites are coupled to their own state and those of neighbors by the

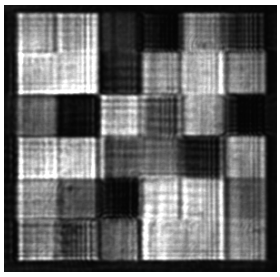

Fig. 2. An image of an 6×6 experimental lattice as seen by the camera in the laser cross-section. Each lattice site is randomly set to a phase shift between 0 and 2π.

update equation

$$\phi_{i,j}^{t+1} = 2\pi\alpha \left[\mathcal{I}(\phi_{i,j}^{t}) + \frac{\epsilon}{4R^2} \sum_{k,l=-R}^{R} \left[\mathcal{I}(\phi_{i+k,j+l}^{t}) - \mathcal{I}(\phi_{i,j}^{t}) \right] \right], \tag{2}$$

where α is an overall constant that controls the temporal dynamics of the uncoupled lattice sites, ϵ is the coupling strength, and R is the radius about the i, j^{th} site that are coupled to the i, j^{th} site. We use fixed boundary conditions, as these provided more interesting dynamics on a smaller lattice size.

A wide variety of dynamics were observed, depending on the chosen parameter values, including various steady state patterns and period-two patterns as well as highly complex dynamically evolving patterns that were without any discernible spatial correlations between neighboring lattice sites. The coupled-lattice dynamics used in this paper for the purpose of testing the local observer reservoir computers were created using parameter values of $\alpha = 0.5$, $\epsilon = 1$, and $R = 3$. The resulting dynamics was in an intermediate regime of complexity, in which there were clearly discernible spatial structures with length scales that extended across several lattice sites and that evolved in time. Snapshots of the resulting dynamics are depicted in Fig. 3.

The pattern-evolution continued over the entire experiment (tens of thousands of cycles), i.e. the dynamics was either non-transient or corresponded to an extremely long transient. We checked that the spatio-temporal evolution was non-periodic on a short timescale (tens of cycles). Visually it had all the hallmarks of spatio-temporal chaos. We expect such spatio-temporal chaos to be typical for this system because numeric investigations of the coupled map lattice Eq. (2) have found large parameter regions of chaos [16].

3 Reservoir Computing

3.1 The reservoir computing paradigm

Reservoir computing is a technique used to simplify the training and implementation of recurrent neural networks. The primary difference between reservoir computing and other recurrent neural networks is the training method. Instead of using error backpropagation to adjust the connection weights between

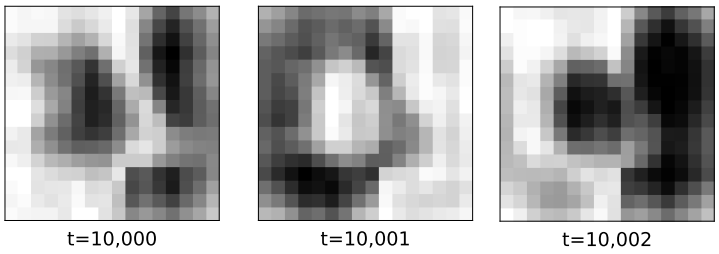

Fig. 3. Snapshots of spatio-temporal dynamics

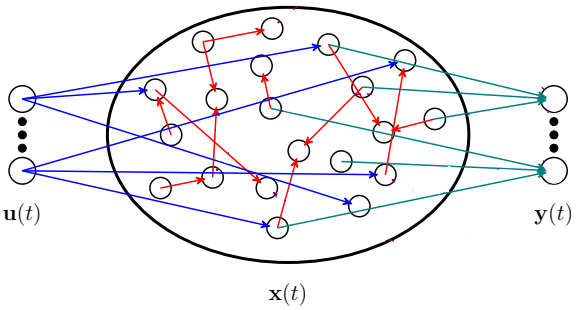

Fig. 4. General structure of reservoir computing: an input vector $\mathbf{u}(t) \in \mathbb{R}^K$ is mapped by a weight matrix $\mathbf{W}_{\text{in}}$ (represented by blue arrows) to the reservoir state vector $\mathbf{x}(t) \in \mathbb{R}^N$, which is, in turn, mapped to the output vector $\mathbf{y}(t) \in \mathbb{R}^L$ by the weight matrix $\mathbf{W}_{\text{out}}$ (represented by green arrows). Internal connections within the reservoir (red arrows) are specified by $\mathbf{W}$.

internal nodes, only the output weights are adjusted. Not only does this allow quick training of relatively large recurrent neural networks, but it can achieve a high degree of accuracy for many tasks [17].

As shown in Fig. 4, a reservoir computing architecture has three main layers: the input layer, the internal nodes that form the reservoir, and the output layer. The input is given by a vector $\mathbf{u}(t) \in \mathbb{R}^K$, which is mapped onto the internal reservoir nodes by an input weight matrix $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times K}$. The internal node dynamics is described by a state vector $\mathbf{x}(t) \in \mathbb{R}^N$ and the reservoir network structure is set by the connection matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$. The reservoir computing output vector is $\mathbf{y}(t) \in \mathbb{R}^L$ and is obtained from the state vector $\mathbf{x}(t)$ via the output weight matrix $\mathbf{W}_{\text{out}} \in \mathbb{R}^{L \times N}$.

The internal nodes each have a one-dimensional update equation with a nonlinear activation function, which we chose in accordance with previous work on reservoir computing [11,13,18–20] as:

$$\mathbf{x}(t+1) = \tanh\left(\mathbf{W}_{\text{in}}\,\mathbf{u}(t) + \mathbf{W}\,\mathbf{x}(t) + \xi\right), \tag{3}$$

where ξ is a constant bias term. The constant bias term serves to push the tanh activation function into the nonlinear or saturation regimes [13]. A small constant noise term ν, typically in the range of $\nu = 0.001 \to 0.01$, is often

added to the "input" during output weight training. The noise term helps to "immunize" the reservoir against unexpected changes in the input $\mathbf{u}(t)$, since it helps prevent overfitting to a specific dataset (see [9] for details). Although it is sometimes helpful to use both linear and nonlinear output functions, for this work only linear output is used. As such, the output equation is:

$$\mathbf{y}(t+1) = \mathbf{W}_{\text{out}}\,\mathbf{x}(t+1). \tag{4}$$

The three weight matrices are generated as follows:

We implement reservoir computing using python 3.6. Internal node connection weights $\mathbf{W}$ are generated using NetworkX's [21] `gnp_random_graph()` function. This function creates an $N \times N$ adjacency matrix with a specified density d_W. The adjacency matrix specifies whether two nodes are connected by assigning $\tilde{\mathbf{W}}_{i,j} = 1$ if the i^{th} and j^{th} nodes are connected and $\tilde{\mathbf{W}}_{i,j} = 0$ if they are not. However, since in general we want both positive and negative connection weights, we rescale each nonzero entry in $\tilde{\mathbf{W}}$ by multiplying it by a random number drawn from the uniform distribution $(-\sigma_W, \sigma_W)$, where $\sigma_W = 1$ in this work. All random matrix entries are generated using the Python Random library function `uniform()`, while p is generated using the Numpy library function `random.random()`. The matrix $\tilde{\mathbf{W}}$ is then rescaled to the user-specified spectral radius ρ by [18]

$$\mathbf{W} = \frac{\rho}{|\lambda_{\max}|}\tilde{\mathbf{W}}, \tag{5}$$

where $\tilde{\mathbf{W}}$ is the initial unscaled connection matrix, $\lambda_{\max}$ is the greatest magnitude eigenvalue of $\tilde{\mathbf{W}}$, and $\mathbf{W}$ is the rescaled connection matrix, which now has a largest eigenvalue of magnitude ρ.

To create $\mathbf{W}_{\text{in}}$, we first generate an all-zero matrix of the correct size and specify a density d_{in}. Then, for each i,j-entry in the connection matrix, a random number p is drawn. If $p < d_{\text{in}}$, a random number from $(-\sigma_{\text{in}}, \sigma_{\text{in}})$ is placed in the entry. Otherwise, the entry is left as 0. Usually $\sigma_{\text{in}} \in (0, 1)$, with different values of σ_{in} corresponding to more or less driving of the reservoir by the input.

The output weight matrix $\mathbf{W}_{\text{out}}$ is the only matrix that is input dependent. It is determined during training, which is achieved using a method called ridge regression. When using this training method, the reservoir is provided with an input vector $\mathbf{u}$ for a training phase of τ time steps. Also given is a corresponding output target $\mathbf{y}_{\text{target}}$. The output matrix $\mathbf{W}_{\text{out}}$ is then given by [11]:

$$\mathbf{W}_{\text{out}} = \mathbf{T}^T\mathbf{M}\left(\mathbf{M}^T\mathbf{M} + \beta\mathbb{I}\right)^{-1} \tag{6}$$

where β is the ridge regression parameter, $\mathbb{I}$ is the $N \times N$ identity matrix, $\mathbf{M} \in \mathbb{R}^{\tau \times N}$ is the reservoir internal state collecting matrix, which consists of state vectors $\mathbf{x}$ from τ time steps, and $\mathbf{T} \in \mathbb{R}^{\tau \times L}$ is the teacher matrix, which consists of the target outputs.

3.2 Observer of numerical data: Temporal chaos

To start, we first test the performance of reservoir computing as a black-box observer of nonlinear dynamical systems on the example of the chaotic Rössler

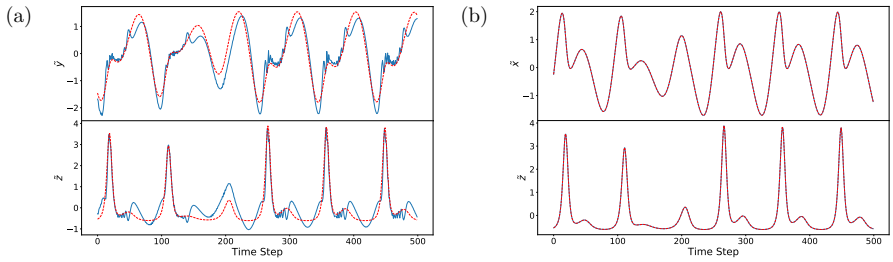

Fig. 5. Output generated, after training, by a reservoir computing structure with purely linear activation functions in its reservoir. The solid blue line is the reservoir computing output, while the dashed red line is the actual value that is obtained by numeric integration of the Rössler equation. (a) Variables $\tilde{y}$ and $\tilde{z}$ are inferred from input $\tilde{x}$. (b) Variables $\tilde{x}$ and $\tilde{z}$ are inferred from input $\tilde{y}$. Note that the solid blue line is under the dashed red line, but the lines are nearly indistinguishable due to the excellent reservoir computing performance.

system

$$\dot{x} = -y - z \tag{7a}$$
$$\dot{y} = x + a\,y \tag{7b}$$
$$\dot{z} = b + z(x - c) \tag{7c}$$

with parameter values of $a = 0.5, b = 2.0$, and $c = 4.0$.

We numerically integrate Eq. (7) and provide one of the three variables as input. The other two variables are used, during the training phase, as the target outputs that the reservoir computing structure is trained to reproduce. We then continue to integrate, drive the system with one of the three variables and test how close the reservoir-computing-generated output is to the two unobserved variables. This is done 100 times for each variable, to minimize the effect of any individual reservoir characteristics and to obtain a variance estimate.

To facilitate mutual comparison between the three variables, we use unit scaled versions of x, y, and z, which we denote as $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$. For example, $\tilde{x}$ is defined by

$$\tilde{x}(t) = \frac{x(t) - \langle x(t) \rangle}{\sqrt{\langle [x(t) - \langle x(t) \rangle]^2 \rangle}}, \tag{8}$$

where $\langle \cdot \rangle$ denotes the time average.

Reservoir computers have successfully been used as observers for the Rössler nonlinear system [12] and our results confirm this finding. The trained reservoir computers do act as observers and do successfully cross-predict variables that are not given as input.

To gain further insight into properties of the reservoir computing approach that leads to successful black-box observers, we investigated the tanh activation function in Eq. (3). This sigmoidal activation function of the reservoir nodes is both nonlinear and saturates for large inputs; both properties are believed

Table 1. Root mean square error (RMSE) of the reservoir-computing-generated unobserved variables compared to the true output for the Rössler system.

Activation Function	$\tilde{x}$ RMSE	$\tilde{y}$ RMSE	$\tilde{z}$ RMSE
Quadratic	input	0.004 ± 0.002	0.002 ± 0.001
Linear	input	0.307 ± 0.003	0.342 ± 0.004
Quadratic	0.003 ± 0.002	input	0.003 ± 0.001
Linear	$(2.0 \pm 1.4) \times 10^{-4}$	input	$(8.4 \pm 0.2) \times 10^{-4}$

to be important. The linearity of this activation function for small input signals is thought to be important for memory, and there is a trade-off between nonlinearity and memory [22]. To test which of these aspects is essential, we replace the tanh function by the following second-order Taylor expansion,

$$\tanh(1 + x) \approx 0.76159 + 0.41997\, x - 0.31985\, x^2. \tag{9}$$

Keeping terms to quadratic order yields an activation function that retains both a linear and nonlinear part but has no saturation. Dropping the quadratic term yields a purely linear activation function that has neither nonlinearity nor saturation.

Performing observation tests, as described above, for a reservoir of 250 nodes, we find that saturation is essential when the reservoir-computing-structure is driven by the $\tilde{z}$-variable of the Rössler system. For our implementation of reservoir computing, the output was found to be unstable for both the linear and quadratic reservoir in the sense that the output grows without bound.

In contrast, bounded output is obtained when the reservoir computing algorithm is driven by the $\tilde{x}$ or $\tilde{y}$ variables of the Rössler system (see Fig. 5). The root mean square error is used as a measure of the deviation of the reservoir output from the truth, i.e. the values obtained by direct integration of Eq. (7).

As seen in Tab. 1, if $\tilde{x}$ is used to drive the reservoir, the quadratic nonlinear activation function performs well in cross-predicting both the $\tilde{y}$ and $\tilde{z}$ variables, whereas the linear activation function has an error that is two order of magnitude greater, indicating that the nonlinearity is important but saturation is not a necessary ingredient.

More interestingly, when $\tilde{y}$ is used as input, the linear activation function outperforms the quadratic nonlinear reservoir in inferring both $\tilde{x}$ and $\tilde{z}$. This indicates that memory alone is sufficient and neither nonlinearity nor saturation is required. There apparently exists some linear mapping from $\tilde{y}$ to the unmeasured variables.

3.3 Observer of experimental data: Spatio-temporal chaos and local modeling

The observation task becomes much more difficult when one goes beyond purely temporal chaos, such as in the Rössler system, and considers spatio-temporal dynamics. Indeed, the authors are not aware of any work where reservoir

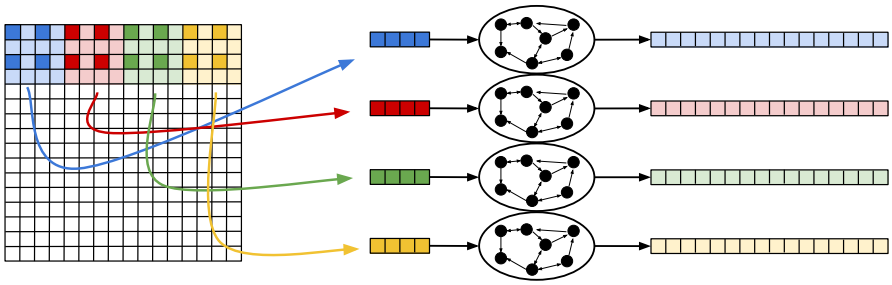

Fig. 6. A representation of the local observer scheme. Each local observer receives 4 (or 2) observed nodes from each 4×4 block of lattice sites. Note that the actual observer scheme uses 16 local observers. Only 4 are shown here for clarity.

computing is used to cross-predict spatio-temporally chaotic dynamics of unobserved spatial sites from measurements at a few selected sites in systems of spatial dimensions larger one. We, also, where unsuccessful with this observation task for the optoelectronic coupled-map-lattice dynamics of our experimental system when using a single reservoir. We speculate that this may be due to the unfavorable scaling of the necessary reservoir size with the dimension and complexity of the dynamical system that one tries to observe.

Taking inspiration from convolutional neural networks, we, therefore, use several independent reservoir computing structures that act in tandem as the observer of the spatio-temporal lattice dynamics. Each reservoir computing structure is given 4 (or 2) coupled-map-lattice sites as an input and produces as output the dynamics of a 4×4 patch of coupled-map-lattice sites. That is, of the 16 lattice sites in a 4×4 patch the dynamics of 12 (or 14) sites is completely unknown and has to be inferred, while the dynamics of 4 (or 2) "observer" sites is known from measurements. The inferred sites are spatially adjacent to the "observer" sites, which means that each independent reservoir computing structure acts as a local observer. Since the coupled-map-lattice is only locally connected, this strategy is presumably resource efficient because the relevant dynamics for inference of a given site is less likely to be overwhelmed by irrelevant dynamics of distant sites. Additionally, reducing the number of variables that have to be inferred means that the number of internal nodes in the reservoir can be chosen much lower, again increasing efficiency.

A cartoon diagram of the local observer reservoir setup is shown in Fig. 6. The coupled map lattice used has 16×16 nodes, so we use 16 individual reservoirs in tandem as our complete observer. Each reservoir computer is instantiated independently, meaning each has a unique set of matrices $\mathbf{W}$, $\mathbf{W}_{\text{in}}$, and, after training, $\mathbf{W}_{\text{out}}$.

To test the effectiveness the local observers, iterations of the coupled-lattice-map experiment are collected, the first 2000 of which are discarded to get rid of transients and the next 18000 coupled-lattice-map iterations are used as the training dataset. Each individual local-observer reservoir-computing-structure is supplied with the dynamics of the appropriate subset of input sites and output sites. The training returns $\mathbf{W}_{\text{out}}$ for each local observer.

Table 2. Network parameters for each local RC.

Parameter	Value
Reservoir size (N)	1000
Spectral radius (ρ)	1.1
Reservoir density (d_W)	0.1
Input density (d_{in})	1.0
Input radius (σ_{in})	0.95
Bias (ξ)	1
Training length (τ)	18,000
Regularization parameter (β)	10^{-2}

After training, which is done off-line, the coupled-lattice-map experiment is run with the last iteration of the training dynamics as the first iteration of the data run. At each iteration, the state of the "observer" site is passed to the respective local observer, and the trained local observers infer the dynamics of the unobserved lattice sites.

For the 4-observer per block configuration, we found a root mean squared error of 0.067 ± 0.006, while for the 2-observer per block configuration, we found a root mean squared error of 0.109 ± 0.007. Both errors were calculated from 500 iterations of lattice dynamics, and averaged over 10 runs. Each run was trained using the same data, and the lattice dynamics used, evolved independently for each run from the last iteration of the training dataset. As expected, the performance is better for the 4-observer per block configuration, but in either configuration unobserved spatiotemporally complex dynamics are inferred with good accuracy.

4 Conclusion

This paper details a novel reservoir computing method by which relatively small reservoirs can be used in assembly to infer unobserved dynamics of a two-dimensional spatio-temporally chaotic coupled-map-lattice. Depending on the number of supplied observer nodes, the inference of the dynamics is more or less robust. An ongoing direction of research is to quantify the degree of model sufficiency and develop a sound statistical procedure that can be used to test for unmodeled dynamics.

References

1. E. Ott, B. R. Hunt, I. Szunyogh, A. V. Zimin, E. J. Kostelich, M. Corazza, E. Kalnay, and D. J. Patil, A local ensemble Kalman filter for atmospheric data assimilation, *Tellus A* **56**, 415–428 (2004).
2. D. R. Creveling, P. E. Gill, and H. D. Abarbanel, State and parameter estimation in nonlinear systems as an optimal tracking problem, *Phys. Lett. A* **372**, 2640–2644 (2008).
3. J. C. Quinn and H. Abarbanel, State and parameter estimation using Monte Carlo evaluation of path integrals, *Q. J. R. Meteorol. Soc.* **136**, 1855 - 1867 (2010).

4. Q. H. Zhang, Adaptive observer for multiple-input-multiple-output (MIMO) linear time-varying systems, *IEEE T. Automat. Contr.* **47**, 525–529 (2002).
5. B. R. Andrievskii, V. O. Nikiforov, and A. L. Fradkov, Adaptive observer-based synchronization of the nonlinear nonpassifiable systems, *Automat. Rem. Contr.* **68**, 1186–1200 (2007).
6. D. C. Yu and U. Parlitz, Estimating parameters by autosynchronization with dynamics restrictions, *Phys. Rev. E* **77**, 066221-7 (2008).
7. L. Illing, A. M. Saunders, and D. Hahs, Multi-parameter identification from scalar time series generated by a Malkus-Lorenz water wheel, *Chaos* **22**, 013127 (2012).
8. Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature* **521**, 436 (2015).
9. H. Jaeger, The "Echo State" Approach to Analysing and Training Recurrent Neural Networks, (2001).
10. W. Maass, T. Natschläger, and H. Markram, Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations, *Neural Comp.* **14**, 2531–2560 (2002).
11. M. Lukoševičius and H. Jaeger, Reservoir computing approaches to recurrent neural network training, *Comput. Sci. Rev.* **3**, 127–149 (2009).
12. Z. Lu, B. R. Hunt, and E. Ott, Attractor reconstruction by machine learning, *Chaos* **28**, 061104 (2018).
13. Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, Reservoir observers: Model-free inference of unmeasured variables in chaotic systems, *Chaos* **27**, 041102 (2017).
14. J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach, *Phys. Rev. Lett.* **120** (2018).
15. R. S. Zimmermann and U. Parlitz, Observing spatio-temporal dynamics of excitable media using reservoir computing, *Chaos* **28**, 043118 (2018).
16. A. M. Hagerstrom, T. E. Murphy, R. Roy, P. Hövel, I. Omelchenko, and E. Schöll, Experimental observation of chimeras in coupled-map lattices, *Nature Physics* **8**, 658–661 (2012).
17. A. Goudarzi and C. Teuscher, in *Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication* (ACM Press, 2016), pp. 1–6.
18. H. Jaeger, A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach, (2002).
19. M. Lukoševičius, in *Neural Networks: Tricks of the Trade*, edited by G. Montavon, G. Orr, and K.-R. Müller (Springer-Verlag, Berlin Heidelberg, 2012), pp. 659–686, 2nd ed.
20. H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, Optimization and applications of echo state networks with leaky- integrator neurons, *Neural Networks* **20**, 335–352 (2007).
21. A. A. Hagsberg, D. A. Schult, and P. J. Swart, in *Proceedings of the 7th Python in Science Conference*, edited by G. Varoquaux, E. Vaught, and J. Millman (Pasadena, CA USA, 2008), pp. 11–15.
22. M. Inubushi and K. Yoshimura, Reservoir Computing Beyond Memory-Nonlinearity Trade-off, *Scientific Reports* **7** (2017).