Forecasting chaotic systems with very low connectivity reservoir computers

Cite as: Chaos **29**, 123108 (2019); https://doi.org/10.1063/1.5120710
Submitted: 19 July 2019 . Accepted: 20 November 2019 . Published Online: 10 December 2019

Aaron Griffith, Andrew Pomerance, and Daniel J. Gauthier

Forecasting chaotic systems with very low connectivity reservoir computers

Cite as: Chaos **29**, 123108 (2019); doi: 10.1063/1.5120710
Submitted: 19 July 2019 · Accepted: 20 November 2019 ·
Published Online: 10 December 2019

Aaron Griffith,[1] Andrew Pomerance,[2] and Daniel J. Gauthier[1]

AFFILIATIONS

[1]Department of Physics, The Ohio State University, 191 W. Woodruff Ave., Columbus, Ohio 43210, USA
[2]Potomac Research LLC, 801 N. Pitt St. # 117, Alexandria, Virginia 22314, USA

Note: This paper is part of the Focus Issue, "When Machine Learning Meets Complex Systems: Networks, Chaos and Nonlinear Dynamics."

ABSTRACT

We explore the hyperparameter space of reservoir computers used for forecasting of the chaotic Lorenz '63 attractor with Bayesian optimization. We use a new measure of reservoir performance, designed to emphasize learning the global climate of the forecasted system rather than short-term prediction. We find that optimizing over this measure more quickly excludes reservoirs that fail to reproduce the climate. The results of optimization are surprising: the optimized parameters often specify a reservoir network with very low connectivity. Inspired by this observation, we explore reservoir designs with even simpler structure and find well-performing reservoirs that have zero spectral radius and no recurrence. These simple reservoirs provide counterexamples to widely used heuristics in the field and may be useful for hardware implementations of reservoir computers.

Reservoir computers (RCs) have seen wide use in forecasting physical systems, inferring unmeasured values in systems, and classification. The construction of a reservoir computer is often reduced to a handful of tunable parameters. Choosing the best parameters for the job at hand is a difficult task. We explored this parameter space on the forecasting task with Bayesian optimization using a new measure for reservoir performance that emphasizes global climate reproduction and avoids known problems with the usual measure. We find that even reservoir computers with a very simple construction still perform well at the task of system forecasting. These simple constructions break common rules for reservoir construction and may prove easier to implement in hardware than their more complex variants while still performing as well.

I. INTRODUCTION

A reservoir computer (RC) is a machine learning tool that has been used successfully for chaotic system forecasting[1] and hidden-variable observation.[2] The RC uses an internal or hidden artificial neural network known as a reservoir, which is a dynamic system that reacts over time to changes in its inputs. Since the RC is a dynamical system with a characteristic time scale, it is a good fit for solving problems where time and history are critical.

More recently, RCs were used to learn the *climate* of a chaotic system;[3,4] that is, it learns the long-term features of the system, such as the system's attractor. Reservoir computers have also been realized physically as networks of autonomous logic on an FPGA[5] or as optical feedback systems,[6] both of which can perform chaotic system forecasting at a very high rate.

A common issue that must be addressed in all of these implementations is designing the internal reservoir. Commonly, the reservoir is created as a network of interacting nodes with a random topology. Many types of topologies have been investigated, from Erdös-Rényi networks and small-world networks[4] to simpler cycle and line networks.[7] Optimizing the RC performance for a specific task is accomplished by adjusting some large-scale network properties, known as *hyperparameters*, while constraining others.

Choosing the correct hyperparameters is a difficult problem because the hyperparameter space can be large. There are a handful of known results for some parameters, such as setting the *spectral radius* ρ_r of the network near to unity and the need for recurrent network connections,[8,9] but the applicability of these results is narrow. In the absence of guiding rules, choosing the hyperparameters is done with costly optimization methods, such as grid search,[7] or

methods that only work on continuous parameters, such as gradient descent.[10]

The hyperparameter optimization problem has also been solved with Bayesian methods,[11,12] which are well suited to optimize over either discrete or continuous functions that are computationally intensive and potentially noisy. Hyperparameters optimized in this way can be surprising: the Bayesian algorithm can find an optimum set of parameters that defy common heuristics for choosing reservoir parameters, as we demonstrate below.

We use this Bayesian approach for optimizing RC hyperparameters for the task of replicating the climate of the chaotic Lorenz '63 attractor[13] the Rössler attractor,[14] and a chaotic double-scroll circuit.[15] We introduce a new measure of reservoir performance designed to emphasize global climate reproduction as opposed to focusing only on short-term forecasting. During optimization, we find that the optimizer often settled on hyperparameters that describe a reservoir network with extremely low connectivity, but which function as well as networks with higher connectivity. Inspired by this observation, we investigate even simpler reservoir topologies. We discover reservoirs that successfully replicate the climate despite having no recurrent connections and $\rho_r = 0$. Such simple network topologies may be easier to synthesize in physical RC realizations, where internal connections and recurrence have a hardware cost.

The rest of this paper is structured as follows: in Sec. II, we describe our RC construction at a high level. We describe the Lorenz '63 system, the Rössler system, and a double-scroll chaotic circuit in Sec. III, which we use as examples for the forecasting task. In Sec. IV, we detail the specifics of how our reservoir networks are constructed, introduce the five hyperparameters we consider, and describe the five network topologies we investigate. We also discuss how to choose these hyperparameters with Bayesian optimization, and how to train the resulting RC. Section V explains our process for evaluating the forecasting ability of these RCs. We discuss the short-term forecasting performance measure and its pitfalls and introduce our modified measure of performance. Section VI describes the results of our investigation, and finally, Sec. VII concludes with some ideas for applying these results in future research.

II. RESERVOIR COMPUTERS

At a high level, an RC is a method to transform one time-varying signal (the input to the RC) into another time-varying signal (the output of the RC), using the dynamics of an internal system called the *reservoir*.

We use an RC construct known as an *echo state network*,[8] which uses a network of nodes as the internal reservoir. Every node has inputs, drawn from other nodes in the reservoir or from the input to the RC, and every input has an associated weight. Each node also has an output, described by a differential equation. The output of each node in the network is fed into the *output layer* of the RC, which performs a linear operation of the node values to produce the output of the RC as a whole. This construction is summarized in Fig. 1.

A. Reservoir

The dynamics of the reservoir are described by

$$\dot{\mathbf{r}}(t) = -\gamma \mathbf{r}(t) + \gamma \tanh\left(W_r \mathbf{r}(t) + W_{\text{in}} \mathbf{u}(t)\right), \tag{1}$$

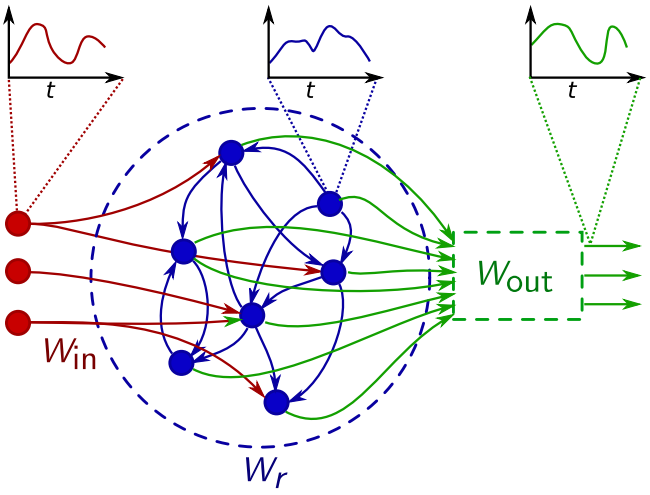

FIG. 1. High-level view of a reservoir computer. Each node may have three kinds of connections: connections to other nodes in the network (W_r, blue), connections to the overall input (W_{in}, red), or connections to the output (W_{out}, green). Note that the internal connections may contain cycles. When the RC is used to perform forecasting, the output on the right side is connected to the input on the left side, allowing the RC to run autonomously with no external input.

where each dimension of the vector $\mathbf{r}$ represents a single node in the network. Here, the function $\tanh(\ldots)$ operates component-wise over vectors: $\tanh(\mathbf{x})_i = \tanh(x_i)$.

For our study, we fix the dimension of the reservoir vector $\mathbf{r}$ at $N = 100$ nodes, and the dimension d of the input signal $\mathbf{u}(t)$ is $d = 3$. Therefore, W_r is an $N \times N$ matrix encoding connections between nodes in the network, and W_{in} is an $N \times d$ matrix encoding connections between the reservoir input $\mathbf{u}(t)$ and the nodes within the reservoir. The parameter γ defines a natural rate (inverse time scale) of the reservoir dynamics. The RC performance depends on the specific choice of γ, W_r, and W_{in}. This choice is discussed further in Sec. IV A.

B. Output layer

The output layer consists of a linear transformation of a function of node values,

$$\mathbf{y}(t) = W_{\text{out}} \tilde{\mathbf{r}}(t), \tag{2}$$

where $\tilde{\mathbf{r}}(t) = \mathbf{f}_{\text{out}}(\mathbf{r}(t))$. The function $\mathbf{f}_{\text{out}}$ is chosen ahead of time to break any unwanted symmetries in the reservoir system. If no such symmetries exist, $\tilde{\mathbf{r}}(t) = \mathbf{r}(t)$ suffices. W_{out} is chosen by *supervised training* of the RC. First, the reservoir structure in Eq. (1) is fixed. Then, the reservoir is fed an example input $\mathbf{u}(t)$ for which we know the desired output $\mathbf{y}_{\text{desired}}(t)$. This example input produces a reservoir response $\mathbf{r}(t)$ via Eq. (1). Then, we choose W_{out} to minimize the difference between $\mathbf{y}(t)$ and $\mathbf{y}_{\text{desired}}(t)$ to approximate

$$\mathbf{y}_{\text{desired}}(t) \approx W_{\text{out}} \tilde{\mathbf{r}}(t). \tag{3}$$

More details of how this approximation is performed can be found in Sec. IV C.

Once the reservoir computer is trained, Eqs. (1) and (2) describe the complete process to transform the RC's input $\mathbf{u}(t)$ into its output $\mathbf{y}(t)$.

C. Forecasting

To forecast a signal $\mathbf{u}(t)$ with an RC, we construct the RC as usual but train W_{out} to reproduce the reservoir input $\mathbf{u}(t)$: we set W_{out} to best approximate

$$\mathbf{u}(t) \approx W_{\text{out}}\tilde{\mathbf{r}}(t). \tag{4}$$

To begin forecasting, we replace the input to the RC with the output. That is, we replace $\mathbf{u}(t)$ with $W_{\text{out}}\tilde{\mathbf{r}}(t)$, and Eq. (1) with

$$\dot{\mathbf{r}}(t) = -\gamma\mathbf{r}(t) + \gamma \tanh\left(W_r\mathbf{r}(t) + W_{\text{in}}W_{\text{out}}\tilde{\mathbf{r}}(t)\right), \tag{5}$$

which no longer has a dependence on the input $\mathbf{u}(t)$ and runs autonomously. If W_{out} is chosen well, then $W_{\text{out}}\tilde{\mathbf{r}}(t)$ will approximate the original input $\mathbf{u}(t)$. These two signals can be compared to assess the quality of the forecast (see Sec. V).

III. EXAMPLE SYSTEMS

As examples for the forecasting task, we consider three chaotic systems: Lorenz '63, the Rössler system, and a double-scroll chaotic circuit. Because all three of these systems are three-dimensional, they can all be used as inputs to the same reservoir computer without modifying W_{in}.

To ensure that the results for all three systems are directly comparable, we rescale the temporal axis so that the maximum positive Lyapunov exponent matches that of the Lorenz system, $\lambda = 0.9056$. We also shift and rescale each component of the system to have zero mean and unit variance, to finally produce the true three-dimensional reservoir input $\mathbf{u}(t)$.

A. Lorenz '63

The Lorenz '63 chaotic system is described by

$$\begin{aligned} \dot{x} &= 10\left(y - x\right), \\ \dot{y} &= x\left(28 - z\right) - y, \\ \dot{z} &= xy - \frac{8}{3}z, \end{aligned} \tag{6}$$

with standard parameters.[13] The attractor of this system can be visualized easily in two dimensions by projecting the three-dimensional trajectory of the system onto a plane. We show the attractor in the x/z plane in Fig. 2.

Our goal is to train an RC by training on a segment of the Lorenz dynamics with Eq. (1), then perform prediction of the Lorenz system after that segment with Eq. (5). Because the Lorenz system is chaotic, forecasting must eventually fail. We choose to only perform prediction for windows of one Lyapunov period $1/\lambda = 1.104$ when evaluating the reservoir performance.

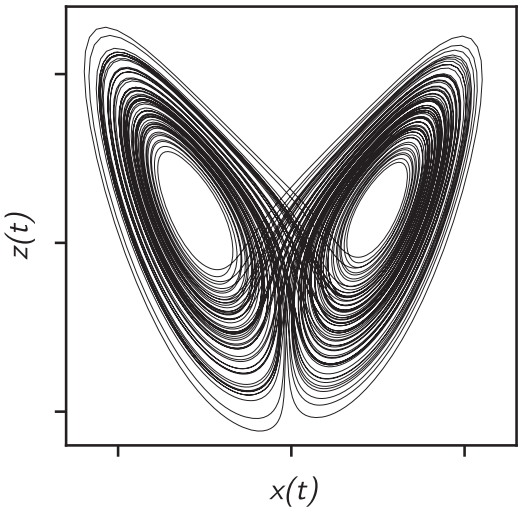

FIG. 2. The true Lorenz attractor in the x/z plane, produced by integrating Eq. (6).

B. Rössler

The Rössler system is described by

$$\begin{aligned} \dot{x} &= -y - z, \\ \dot{y} &= x + ay, \\ \dot{z} &= b + z(x - c), \end{aligned} \tag{7}$$

with standard parameters $a = 0.2$, $b = 0.2$, and $c = 5.7$.[14]

The z component of this system mostly stays near zero, with rare positive spikes. This makes prediction with an RC difficult. To make this component of the system more suitable for RC prediction, we use $\log z$ instead for both RC input and prediction output.

C. Double-scroll

The double-scroll chaotic circuit is described by the dimensionless equations,

$$\begin{aligned} \dot{V}_1 &= \frac{V_1}{R_1} - \frac{V_1 - V_2}{R_2} - 2I_r \sinh\left(\alpha(V_1 - V_2)\right), \\ \dot{V}_2 &= \frac{V_1 - V_2}{R_2} + 2I_r \sinh\left(\alpha(V_1 - V_2)\right) - I, \\ \dot{I} &= V_2 - R_4 I, \end{aligned} \tag{8}$$

with parameters $R_1 = 1.2$, $R_2 = 3.44$, $R_4 = 0.193$, $I_r = 2.25 \times 10^{-5}$, and $\alpha = 11.6$.[15]

IV. RESERVOIR CONSTRUCTION AND TRAINING

To build our reservoir computers, we need to build the internal network to use as the reservoir, create connections from the nodes to the overall input, and then train it to fix W_{out}. Once this is completed, the RC will be fully specified and able to perform forecasting.

A. Internal reservoir construction

There are many possible choices for generating the internal reservoir connections W_r and the input connections W_{in}. For W_{in}, we randomly connect each node to each RC input with probability σ. The weight for each connection is drawn randomly from a normal distribution with mean 0 and variance ρ_{in}^2. Together, σ and ρ_{in} are enough to generate a random instantiation of W_{in}.

For the internal connections W_r, we generate a random network where every node has a fixed in-degree k. For each node, we select k nodes in the network without replacement and use those as inputs to the current node. Each input is assigned a random weight drawn from a normal distribution with mean 0 and variance 1. This results in a connection matrix W_r' where each row has exactly k nonzero entries. Finally, we rescale the whole matrix,

$$W_r = \frac{\rho_r}{\text{SR}(W_r')} W_r', \tag{9}$$

where $\text{SR}(W_r')$ is the spectral radius, or maximum absolute eigenvalue, of the matrix W_r'. This scaling ensures that $\text{SR}(W_r) = \rho_r$. Together, k and ρ_r are enough to generate a random instantiation of W_r. We present an example of such a network in Fig. 3(a).

Therefore, to create a random instantiation of an RC suitable to begin the training process, we must set a value for five *hyperparameters*:

- γ, which sets the characteristic time scale of the reservoir;
- σ, which determines the probability a node is connected to a reservoir input;
- ρ_{in}, which sets the scale of input weights;
- k, the recurrent in-degree of the reservoir network; and
- ρ_r, the spectral radius of the reservoir network.

We select these parameters by searching a range of acceptable values selected to minimize the forecasting error using the Bayesian optimization procedure. The details of this can be found in Sec. IV B. However, during the optimization process, we discovered that the optimizer was often finding RCs with $k = 1$ that perform as well as RCs with a higher k. Such reservoirs have an interesting and simple network topology, thereby suggesting other simple topologies for comparison.

First, networks with $k = 1$ generated with our algorithm often have disconnected components. These components essentially act as RCs operating in parallel; we do not consider these further even though it is an interesting line of research.[16] We limit ourselves to only looking at reservoir networks with a single connected component.

If a $k = 1$ network only has a single connected component, then it must also contain only a single directed cycle. This limits how recurrence can occur inside the network compared to higher-k networks. Every node in a $k = 1$ network is either part of this cycle or part of a directed tree branching off from this cycle, as depicted in Fig. 3(b). Inspired by the high performance of this simple structure, we also investigate $k = 1$ networks when the single cycle is cut at an arbitrary point. This turns the entire network into a tree, as in Fig. 3(c).

Finally, we also investigate reservoir networks that consist entirely of a cycle or ring with identical weights with no attached tree structure, depicted in Fig. 3(d), and networks with a single line of nodes (a cycle that has been cut), in Fig. 3(e). These are also known as *simple cycle reservoirs* and *delay line reservoirs*, respectively.[7]

In total, there are five topologies we investigate

(a) general construction with unrestrained k;
(b) $k = 1$ with a single cycle;
(c) $k = 1$ with a cut cycle;
(d) single cycle, or *simple cycle reservoir*; and
(e) single line, or *delay line reservoir*.

Both the $k = 1$ cut cycle networks (c) and line networks (e) are rescaled to have a fixed ρ_r before the cycle is cut. However, after the cycle is cut, they both have $\rho_r = 0$.

B. Bayesian optimization

The choice of hyperparameters that best fits this problem is difficult to identify. Grid search[7] and gradient descent[10] have been used previously. However, these algorithms struggle with either non-continuous parameters or noisy results. Because W_r and W_{in} are determined randomly, our optimization algorithm should be able to handle noise. We use Bayesian optimization,[11,12] as implemented by the `skopt` Python package.[17] Bayesian optimization deals well with both noise and integer parameters like k, is more efficient than grid search,[12] and works well with minimal tuning.

For each topology, the Bayesian algorithm repeatedly generates a set of hyperparameters to test within the ranges listed in Table I. Larger ranges require a longer optimization time. We selected these

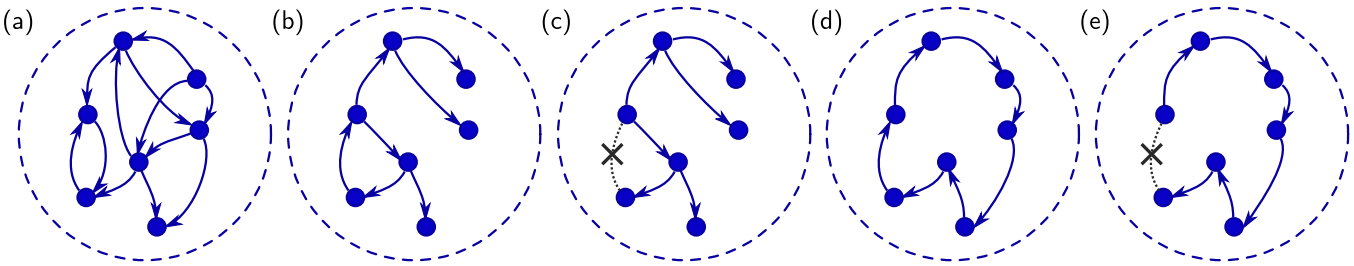

FIG. 3. The five reservoir topologies tested. Only internal reservoir connections are pictured. Connections to the reservoir computer input, or to the output layer (as in Fig. 1) are not shown. (a) A general, fixed in-degree network, here pictured with $N = 7$ and $k = 2$. (b) A $k = 1$ network with a single connected component. (c) A $k = 1$ network with the single cycle cut at an arbitrary point. (d) A *simple cycle reservoir*. (e) A *delay line reservoir*.

TABLE I. Range of hyperparameters searched using Bayesian optimization.

Parameter	Min	Max
γ	7–11	
σ	0.1–1.0	
ρ_{in}	0.3–1.5	
k	1–5	
ρ_r	0.3–1.5	

ranges to include the values that existing heuristics would choose and to allow exploration of the space without a prohibitively long runtime. However, exploring outside these ranges is valuable. Here, we focus on the connectivity k, but expanding the search range for the other parameters may also produce interesting results.

At each iteration of the algorithm, the optimizer constructs a single random reservoir with the chosen hyperparameters, trains it according to the procedure described in Sec. IV C, and measures its performance with the metric ε described in Sec. V. From this measurement, it chooses a new set of hyperparameters to test that may be closer to the optimal values. We limit the number of iterations of this algorithm to test a maximum of 100 reservoir realizations before returning an optimized reservoir. In order to estimate the variance in the performance of reservoirs optimized by this method, we repeat this process 20 times.

C. Training

To train the RC, we integrate Eq. (1) coupled with the chosen input [Eqs. (6) to (8)] via the hybrid Runge-Kutta 5(4)[18] method from $t = 0$ to 300 with a fixed time step of $\Delta t = 0.01$ and divide this interval into three ranges:

- $t = 0-100$: a transient, which is discarded;
- $t = 100-200$: the training period; and
- $t = 200-300$: the testing period.

We use the transient period to ensure that the later times are not dependent on the specific initial conditions. We divide the rest into a training period, used only during training, and a testing period, used later only to evaluate the RC performance.

This integration produces a solution for $\mathbf{r}(t)$. However, when the reservoir is combined with the Lorenz system, it has a symmetry that can confuse prediction.[3] Before integration, we break this symmetry by setting $\mathbf{f}_{\text{out}}$ so that

$$\tilde{r}_i(t) = \begin{cases} r_i(t) & \text{if } i \le N/2, \\ r_i(t)^2 & \text{if } i > N/2. \end{cases} \tag{10}$$

For consistency across our three example input systems, this is done for every reservoir we construct, even if the input system we eventually use is not the Lorenz system. We then find a W_{out} to minimize

$$\sum_{t=100}^{200} |\mathbf{u}(t) - W_{\text{out}}\,\tilde{\mathbf{r}}(t)|^2 + \alpha ||W_{\text{out}}||^2, \tag{11}$$

where the sum is understood to be over time steps Δt apart. Now that W_{out} is determined, the RC is trained.

Equation (11) is known as Tikhonov regularization or ridge regression. The ridge parameter α could be included among the hyperparameters to optimize. However, unlike the other hyperparameters, modifying α does not require reintegration and can be optimized with simpler methods. We select an α from among 10^{-5} to 10^5 by leave-one-out cross-validation. This also reduces the number of dimensions the Bayesian algorithm must work with.

V. FORECASTING AND EVALUATION

To evaluate the performance of the trained RC, we use it to perform autonomous forecasting using Eq. (5).

The most common method for evaluating an RC forecast is to choose a time t_1 to begin forecasting and then compare the forecast to the true system.[9] Usually, t_1 is chosen to be directly after the training period, which is $t_1 = 200$ in our procedure. Then, we initialize the reservoir state $\mathbf{r}(t_1)$ to the value found earlier during training and integrate Eq. (5) for one Lyapunov time, between $t = t_1$ and $t = t_1 + 1/\lambda$. This produces a reservoir forecast $W_{\text{out}}\,\tilde{\mathbf{r}}(t)$ during these times, which we compare to the true system $\mathbf{u}(t)$ to produce a root-mean-squared error (RMSE),

$$\varepsilon_1 = \left(\Delta t \lambda \sum_{t=t_1}^{t_1+1/\lambda} |\mathbf{u}(t) - W_{\text{out}}\tilde{\mathbf{r}}(t)|^2 \right)^{1/2}. \tag{12}$$

Note that ε_1 is normalized because we construct the input signal $\mathbf{u}(t)$ with unit variance.

This method of evaluating forecasting ability is flawed for our purposes. Previous results have shown that a low ε_1 is not a good indicator of whether a reservoir computer has learned the climate of a system[3,4] and we also observe the same effect here. Figure 4 depicts two common ways for an RC to fail to replicate the true Lorenz

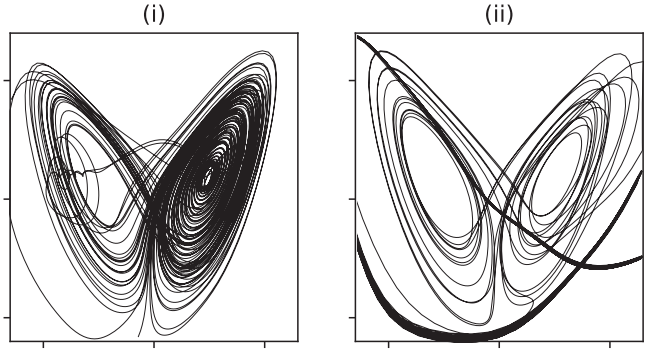

FIG. 4. Two examples of reservoir computers that fail to reproduce the Lorenz climate, but produce a low ε_1. Compare with the true attractor in Fig. 2. Reservoir (i) fails to learn the attractor, overemphasizing the right lobe in which t_1 resides. Reservoir (ii) matches the attractor well early on, but as the prediction lengthens, it falls into a periodic orbit (thick black line). Both (i) and (ii) show a promising low ε_1, but the averaged ε measure more accurately captures their failure to learn the Lorenz attractor.

attractor, shown in Fig. 2. However, both produce a good short-term forecast near t_1. In particular, reservoir (ii) in Fig. 4 has a lower ε_1 score than any of the optimized reservoirs despite its obvious failure to learn the Lorenz attractor.

This problem is exacerbated in an optimization setting because searching for reservoirs with the lowest ε_1 risks producing reservoirs that are only good at forecasts near t_1, but otherwise perform poorly in reproducing the climate. It can also waste time, as the optimization algorithm explores areas of the parameter space it believes perform well, but actually do not.

Finally, the choice of t_1 can dramatically affect ε_1. For example, the Lorenz system has an unstable saddle point at the origin, and trajectories that approach this point may end up in either the left or right lobe of the attractor in Fig. 2. If t_1 happens to lie near this point, then even very small prediction errors can be magnified. For example, if the reservoir predicts a trajectory into the left lobe, while the true system goes to the right, the ε_1 measure might be very high even though the reservoir is well-trained.

To combat these problems, we instead evaluate a short-term forecast at 50 times t_i, evenly spaced within our testing period between $t = 200 - 300$. For each t_i, we perform the evaluation method as described above, producing 50 error measures ε_i. Because each t_i is drawn from the testing period, we only evaluate the reservoir on data it has not yet seen: no information about the input or reservoir system at t_i is used to construct W_{out}.

We then combine these errors into a single overall error

$$\varepsilon = \left(\frac{1}{50} \sum_{i=1}^{50} \varepsilon_i^2 \right)^{1/2} \tag{13}$$

that represents the average ability of the reservoir computer to forecast accurately at any point on the input system attractor.

The parameter ε is our figure of merit that the Bayesian algorithm is trying to minimize. By combining short-term forecasting errors from many points on the input attractor, this metric emphasizes learning the global dynamics of the system as opposed to short-term forecasting over a single temporal segment. Using this method, we see that the Bayesian optimization algorithm works more effectively, as it no longer gets trapped in valleys of low ε_1 that otherwise fail to reproduce the attractor. This evaluation method is summarized in Fig. 5.

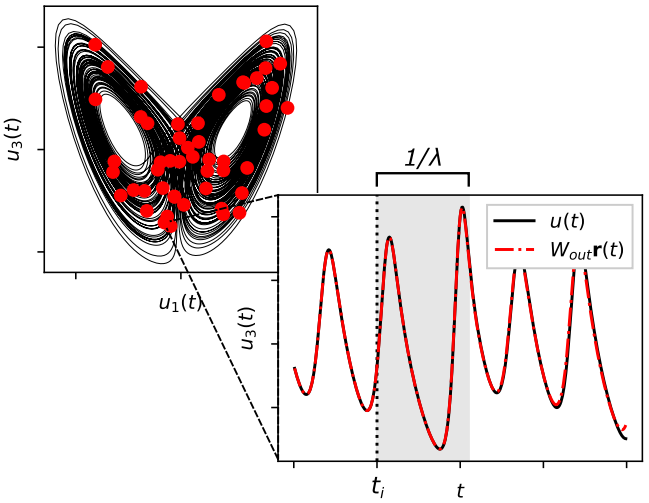

FIG. 5. Summary of our forecasting evaluation method. We calculate errors ε_i at times t_i, marked on the attractor as red dots, and combine them into a single error ε. Before each t_i (dotted vertical line), the reservoir is integrated with Eq. (1) and listening to the input. After t_i, the reservoir is integrated with Eq. (5) and runs autonomously. The reservoir's prediction (dotted red line) must eventually diverge from the true system (solid black line). We calculate ε_i only during a single Lyapunov period after forecasting begins, marked here in grey.

VI. RESULTS

We run all five reservoir topologies through 100 iterations of the Bayesian algorithm using the Lorenz system as input and record the best-performance RC for each topology according to the metric ε. These reservoirs, and the hyperparameters that generated them, are reported in Table II. We estimate the errors on these with the standard deviation of ε after repeating the optimization process 20 times.

When optimized, all reservoir topologies perform well. In particular, the simpler topologies all perform almost as well as the general-k topology. They often lie within one, or at most two standard deviations from topology (a). This is despite the fact that topologies (c) and (e) both have $\rho_r = 0$ and no recurrent connections within the network. The other topologies have $\rho_r \ll 1$. Previous work has already demonstrated that reservoirs with low or zero spectral radius can still function.[3,7] These results act as additional counterexamples to the heuristic that reservoir computers should have $\rho_r \approx 1$.[9]

However, these best-observed reservoirs are not representative of a typical RC. We use the hyperparameters to guide the random input connections and connections within the reservoir, and so even once the hyperparameters are fixed, constructing the reservoir is a random process. Not all reservoirs with a fixed topology and hyperparameters will perform the same.

To explore this variation, we generate and evaluate 200 RCs of each topology on the Lorenz system, using the optimized hyperparameters in Table II. For all five topologies, as the measured ε increases, the quality of the reproduced attractor decreases gradually.

TABLE II. Best reservoir computers of each topology, after 100 iterations of the Bayesian optimization algorithm using the Lorenz system as input. The hyperparameters chosen by the algorithm are shown on the right. The simpler topologies (b)–(e) all perform nearly as well as the general topology (a).

		Lorenz				
	Topology	ε	γ	σ	ρ_{in}	ρ_r
(a)	Any k[a]	0.022 ± 0.004	7.7	0.81	0.37	0.41
(b)	$k = 1$ with cycle	0.024 ± 0.005	10.9	0.44	0.30	0.30
(c)	$k = 1$ no cycle	0.028 ± 0.005	7.2	0.78	0.30	0.30[b]
(d)	Cycle	0.023 ± 0.008	7.9	0.17	0.58	0.30
(e)	Line	0.024 ± 0.003	10.6	0.79	0.30	0.45[b]

[a] After optimization, $k = 3$.
[b] ρ_r measured before cycle is cut. Afterwards, $\rho_r = 0$.

On manual inspection of the attractors, the reason for this decrease can be divided into three qualitative regions. For $\varepsilon < 0.3$, the RCs reproduce the Lorenz attractor consistently. Failures still rarely occur, but they always reproduce part of the attractor before falling into a fixed point or periodic orbit. In this region, small differences between the true attractor and the reproduced attractor contribute more to ε than outright failure. For $0.3 < \varepsilon < 1.0$, RCs always fail to reproduce the attractor, though they will still always reproduce a portion of it before failing. Examples of these failures are provided in Fig. 4. Above $\varepsilon > 1.0$, these failures become catastrophic, and no longer resemble the Lorenz attractor at all. A more quantitative description of these regions is one line of possible future research.

Though the optimized best-performing reservoirs of each topology show very little performance difference, the differences between them become more apparent when we compare the probability distribution of ε for each topology. These distributions are shown in Fig. 6.

A well-performing reservoir with arbitrary k (a) is a much more likely outcome than a well-performing reservoir with a single cycle (d). However, the performance of arbitrary k reservoirs (a) is very similar to that of treelike reservoirs (c). Though (c) has a longer tail on the high end, the simpler structure of the reservoir may be appealing for hardware RCs.

The distribution of performance for each topology can be a deciding factor if reservoir construction and evaluation is expensive, as it might be in hardware. In software, though, we find the best-performing reservoirs in Table II after only 100 trials. A hardware design can still benefit from the simpler topologies (b)–(e) despite their very wide performance distributions if the creation of the evaluation of the design can be automated to test many candidate reservoirs, as on an FPGA.[5]

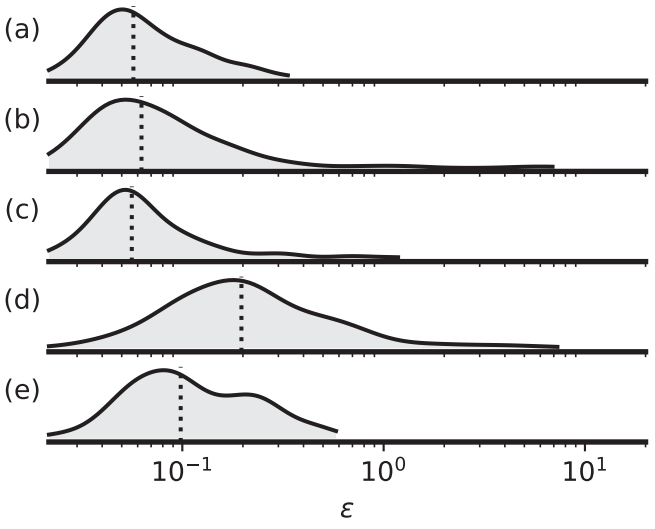

FIG. 6. Performance for each reservoir topology, evaluated for the Lorenz system at the hyperparameters listed in Table II. Each is visualized as a Gaussian kernel density estimation in $\log_{10} \varepsilon$ with a bandwidth of 0.35,[19] which can be interpreted as a probability distribution. Using a narrower bandwidth does not reveal additional features. A vertical line marks the median. Note that topologies (b)–(e) have very long tails and can produce reservoirs that perform very poorly in comparison to (a). However, all five have the capability to produce well-performing reservoirs.

TABLE III. Result of optimizing reservoirs with the Bayesian algorithm over 100 iterations, on both the Rössler system and the double-scroll system. All five topologies can perform equally well at both systems.

	Topology	Double scroll ε	Rössler ε
(a)	Any k	0.029 ± 0.006	0.017 ± 0.005
(b)	$k = 1$ with cycle	0.033 ± 0.007	0.020 ± 0.007
(c)	$k = 1$ no cycle	0.033 ± 0.008	0.018 ± 0.006
(d)	Cycle	0.033 ± 0.007	0.018 ± 0.006
(e)	Line	0.037 ± 0.01	0.019 ± 0.015

There may also be a benefit in software. The simpler topologies are represented by weight matrices W_r in very simple forms. Topology (c) can always be represented as a strictly lower-diagonal matrix, and (d)–(e) can be represented with nonzero entries only directly below the main diagonal. Software simulations can take advantage of this structure to speed up the integration of Eq. (1).

To explore whether these topologies remain equally effective at tasks beyond forecasting the Lorenz system, we run all five reservoir topologies through 100 iterations of the Bayesian algorithm for both the Rössler and the double-scroll systems. As with the Lorenz system, we estimate the errors on these results by repeating the process 20 times each. These results are reported in Table III.

The results for the double-scroll and Rössler systems agree well with those for Lorenz. All five topologies optimize reliably with the Bayesian algorithm, and all perform similarly when optimized. Optimizing a reservoir to reproduce either system will almost always work after only 100 iterations.

One advantage to RCs is that a single reservoir can be reused on many different tasks by retraining W_{out}. To evaluate whether this is possible with these optimized reservoirs, we take the 20 reservoirs optimized for the Lorenz system and retrain W_{out} for each to instead reproduce the double-scroll circuit system. Every other part of the RC is left unchanged. We then evaluate how accurate this prediction is using the metric ε. These results are summarized in Table IV.

In general, these reservoirs perform poorly on this new task. However, there is extremely high variation. Even though they were optimized to perform Lorenz forecasting, many of these reservoirs

TABLE IV. Result of reusing reservoirs optimized for Lorenz prediction to perform double-scroll prediction. The minimum error encountered across all 20 reservoirs of each topology is reported as ε_{min}.

	Topology	Double scroll ε	Double scroll ε_{min}
(a)	Any k	0.43 ± 1.2	0.028
(b)	$k = 1$ with cycle	0.30 ± 0.5	0.048
(c)	$k = 1$ no cycle	0.37 ± 0.8	0.032
(d)	Cycle	0.17 ± 0.2	0.056
(e)	Line	0.22 ± 0.3	0.058

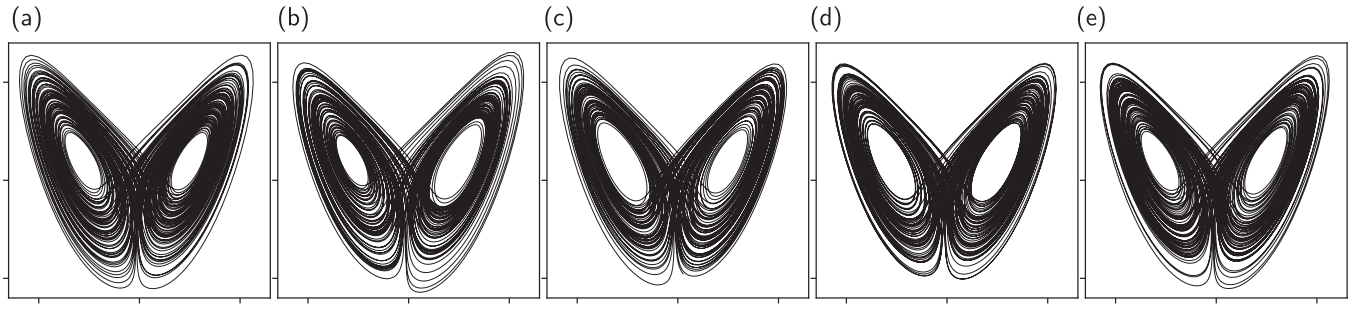

FIG. 7. Lorenz attractor plots in the x/z plane for long-term free-running predictions for each reservoir topology in Table II. Compare with true Lorenz attractor in Fig. 2, and the failed reservoirs in Fig. 4.

are still able to reproduce the double-scroll attractor. Moreover, the best performers in each category approach the performance of reservoirs optimized specifically for the double-scroll system. This indicates that it is possible to find a single reservoir in any of these topologies that works well on more than one system. The Bayesian optimization algorithm may be able to find these reservoirs more reliably if the metric ε is modified to reward RCs that perform well on many systems.

Finally, for each topology, we produced a free-running prediction of the Lorenz system for 100 time units using the best-performing RC. We use these predictions to produce an attractor as shown in Fig. 7. All five optimized RCs reproduce the Lorenz attractor well. Though comparing these plots by eye is not quantitative, we find them qualitatively useful: an RC that fails to reproduce the Lorenz attractor by eye is unlikely to have a low ε compared to the one that does.

VII. CONCLUSION

We find that Bayesian optimization of RC hyperparameters is a useful tool for creating high-performance reservoirs quickly. We also find that allowing the optimizer to explore areas of the parameter space that are typically excluded in other optimization studies can lead to interesting and effective reservoir designs such as those presented here.

For this procedure to be effective, we find that evaluating the RC performance at many points along the attractor and averaging, rather than at a single point, encourages the optimization algorithm to find reservoirs that reproduce the Lorenz climate. Using this evaluation method helps direct the optimizer away from reservoirs that perform good short-term forecasting at only one point on the Lorenz attractor.

One surprising outcome of our optimization procedure is finding reservoirs that perform well even with no recurrent connections and $\rho_r = 0$. Though some reservoirs of this kind have been explored previously and shown to work,[3,7] the heuristics remain common in reservoir design. We present additional concrete examples that provide evidence these heuristics are not unbreakable rules.

In greater detail, we find reservoirs with very low internal connectivity that perform at least as well as their higher-connectivity counterparts. A reservoir with only a single internal cycle, or even no cycle at all, can perform as well as those with many recurrent cycles. These simpler topologies manifest as simpler weight matrices, which can result in faster integration in software. In a hardware environment where connections between nodes have a cost, or recurrence is difficult to implement, these network topologies may also have a direct benefit.

Though the best of these low-connectivity reservoirs perform as well as the more complicated reservoirs, they tend to perform worse on average. However, searching for the best-performing instance of these reservoirs can be done in few trials and may be feasible for hardware reservoirs that can be constructed and evaluated in an automated way.

We have discovered many interesting lines of future research. First, we can evaluate the new metric ε by comparing the output of the reservoir predictions to the true system attractor using a new metric for attractor overlap.[20] This overlap metric can also be used to quantify the qualitative observations of different failure modes in regions of our ε metric. Second, there are known results that prove that a linear network architecture with time-independent nodes, the discrete-time NARX networks, can simulate fully-connected networks.[21] There may be a similar proof for RCs, which might explain why we see no difference in the best-performing reservoirs in each topology. Third, our optimization procedure finds the best network weights W_r for a given task. In many ways, this is a similar task to training a traditional recurrent neural network. We are interested in comparing this method to those used for networks other than RCs.

Our results show that these very low-connectivity reservoirs perform well in the narrow context of software-based, chaotic system forecasting. Future work will explore whether these results hold for other reservoir computing tasks such as classification, and whether it is possible to find reservoirs that perform well on a variety of tasks simultaneously by modifying the metric ε to encourage generalization. We also intend to explore whether these results hold in hardware reservoirs and if the simpler reservoir designs allow for more efficient and faster operating hardware RCs.

ACKNOWLEDGMENTS

We thank our reviewers for their comments and for their suggestion to use $\log z$ for prediction in the Rössler system. We gratefully acknowledge the financial support of the U.S. Army Research Office (U.S. ARO, Grant No. W911NF-12-1-0099), Defense Advanced

Research Projects Agency (DARPA, Award No. HR00111890044), and a Network Science seed grant from the The Ohio State University College of Arts & Sciences.

REFERENCES

[1]H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," Science **304**, 78–80 (2004).
[2]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).
[3]J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, "Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data," Chaos **27**, 121102 (2017).
[4]A. Haluszczynski and C. Räth, "Good and bad predictions: Assessing and improving the replication of chaotic attractors by means of reservoir computing," Chaos **29**, 103143 (2019).
[5]D. Canaday, A. Griffith, and D. J. Gauthier, "Rapid time series prediction with a hardware-based reservoir computer," Chaos **28**, 123119 (2018).
[6]P. Antonik, M. Hermans, F. Duport, M. Haelterman, and S. Massar, "Towards pattern generation and chaotic series prediction with photonic reservoir computers," in *Proceedings of SPIE* (SPIE, 2016), Vol. 9732, p. 97320B.
[7]A. Rodan and P. Tiňo, "Minimum complexity echo state network," IEEE Trans. Neural Netw. **22**, 131–144 (2011).
[8]H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," German National Research Center for Information Technology GMD Technical Report, Bonn, Vol. 148, p. 13, 2001.
[9]M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*, 2nd ed., edited by G. Montavon, G. B. Orr, and K.-R. Müller (Springer, Berlin, 2012), pp. 659–686.
[10]H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," Neural Netw. **20**, 335–352 (2007).
[11]J. Yperman and T. Becker, "Bayesian optimization of hyper-parameters in reservoir computing" CoRR (2016), arXiv:1611.05193 [cs.LG].
[12]J. R. Maat, N. Gianniotis, and P. Protopapas, "Efficient optimization of echo state networks for time series datasets," in *International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2018).
[13]E. N. Lorenz, "Deterministic nonperiodic flow," J. Atmos. Sci. **20**, 130–141 (1963).
[14]O. E. Rössler, "An equation for continuous chaos," Phys. Lett. A **57**, 397–398 (1976).
[15]D. J. Gauthier and J. C. Bienfang, "Intermittent loss of synchronization in coupled chaotic oscillators: Toward a new criterion for high-quality synchronization," Phys. Rev. Lett. **77**, 1751–1754 (1996).
[16]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**, 024102 (2018).
[17]The scikit-optimize contributors, "scikit-optimize/scikit-optimize: v0.5.2" (2018).
[18]J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," J. Comput. Appl. Math. **6**, 19–26 (1980).
[19]D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, 1st ed. (Wiley, Berlin, 1992).
[20]R. Ishar, E. Kaiser, M. Morzyński, D. Fernex, R. Semaan, M. Albers, P. S. Meysonnat, W. Schröder, and B. R. Noack, "Metric for attractor overlap," J. Fluid Mech. **874**, 720–755 (2019).
[21]H. T. Siegelmann, B. G. Horne, and C. L. Giles, "Computational capabilities of recurrent NARX neural networks," IEEE Trans. Syst. Man Cybernet. B **27**, 208–215 (1997).