A reservoir computer is a complex nonlinear dynamical system that has been shown to be useful for solving certain problems, such as prediction of chaotic signals, speech recognition or control of robotic systems. Typically a reservoir computer is constructed by connecting a large number of nonlinear nodes in a network, driving the nodes with an input signal and using the node outputs to fit a training signal. In this work, we set up reservoirs where the edges (or connections) between all the network nodes are either +1 or 0, and proceed to alter the network structure by flipping some of these edges from +1 to -1. We use this simple network because it turns out to be easy to characterize; we may use the fraction of edges flipped as a measure of how much we have altered the network. In some cases, the network can be rearranged in a finite number of ways without changing its structure; these rearrangements are symmetries of the network, and the number of symmetries is also useful for characterizing the network. We find that changing the number of edges flipped in the network changes the rank of the covariance of a matrix consisting of the time series from the different nodes in the network, and speculate that this rank is important for understanding the reservoir computer performance.

I. INTRODUCTION

Reservoir computers were developed as a type of recurrent neural network by machine learning researchers [1, 2], but they may also be described using the language of dynamical systems. An advantage of reservoir computers over other machine learning techniques is that training a reservoir computer is fast and simple; the reservoir itself, a network of nonlinear nodes, is kept fixed, and the time series responses from the nodes are combined using a linear weighted sum, where the weights are varied to fit a training signal.

Reservoir computers have been shown to be useful for solving a number of problems, including reconstruction and prediction of chaotic attractors [3–7], recognizing speech, handwriting or other images [8] or controlling robotic systems [9]. One attractive feature of reservoir computers is that they may be implemented in a wide range of analog hardware, making them potentially very fast but with low power consumption. Examples of reservoir computers so far include photonic systems [10, 11], analog circuits [12], mechanical systems [13] and field programmable gate arrays [14].

One obstacle to understanding what reservoir computers can or can’t do is that there is only a limited amount of theory on how reservoir computers function. Much of the theoretical work hinges on understanding the tradeoff between nonlinearity in the reservoir computer nodes and memory [15, 16], where memory is described by the time dependent correlation between the reservoir computer input and output. Other work focuses on generalized synchronization [3, 17]. The previous work doesn’t describe how the choice of the network influences the reservoir computer, so in this work we study the effect of different networks on the reservoir computer.

Varying a parameter in a reservoir computer can lead to complex changes in the behavior, but it is possible to vary a parameter over a broad range and look for trends
in the behavior of an ensemble of similar reservoir computers. We proceed by setting up a network of edges that connect the reservoir computer nodes in which every edge is either 1 or 0. This type of network is easy to characterize. We vary the network structure by flipping network edges from +1 to -1. We will see that if we take two copies of the same initial network and flip the same number of edges but choose different edges to flip, the performance of the reservoir computer may be different. Therefore we choose many networks where the same number of elements are flipped and track trends in the behavior of the entire group of reservoir computers.

We will examine two different node types and two different input signals. The nodes are based on a nonlinear differential equation or a sigmoid function map, while the input signals will come from a Lorenz chaotic system or a nonlinear map acting on a random signal. The nonlinear mapping was chosen from a set of problems commonly used to test reservoir computers \[18\].

In section II, we will describe reservoir computers and show how they may be trained. Section III describes our choice of how to feed signals into the reservoir computer, while section IV described how the network adjacency matrix may be characterized. All elements of the adjacency matrix, which describes the edges between nodes, start as +1 or 0, and some of the +1 edges are flipped to -1. The values of the edges are then normalized so that the maximum of the absolute value of the real part of the set of eigenvalues of the adjacency matrix is 0.5. We may then characterize the adjacency matrix by the number of symmetries it contains or by the fraction of the elements flipped from +1 to -1. In section V we describe why the rank of the covariance of the reservoir matrix is useful for describing the reservoir computer.

II. RESERVOIR COMPUTERS

We used a reservoir computer to estimate one time series signal based on a different (but related) time series signal. Figure 1 is a block diagram of a reservoir computer. There is an input signal \(s(t)\) from which the goal is to extract information, and a training signal \(g(t)\) which is used to train the reservoir computer. In [6] for example, \(s(t)\) was the \(x\) signal from a Lorenz chaotic system, while \(g(t)\) was the Lorenz \(z\) signal. The reservoir computer was trained to estimate the \(z\) signal from the \(x\) signal.

There are no specific requirements on the nodes in a reservoir computer, other than when all nodes are connected into a network, the network should be stable; that is, it should settle into a stable fixed point. Commonly used nodes include hyperbolic tangent [12] or sigmoid functions [20], but in analog experiments the node nonlinearity is determined by the experimental system [10–14]. We select two different node types to decrease the chance that our results depend on the type of node used. One node type, the polynomial node, is chosen because a polynomial is a general way to represent a nonlinearity.

The parameters for the polynomial node are chosen so that network based on these nodes is stable. The second type of node is based on a sigmoid function. Sigmoid functions are common in neural network studies [21], although the form of our node is not the same as the most commonly used sigmoid function. The form of the nonlinearities in these two node types is different enough that our results should be general for different types of reservoir computers.

The polynomial reservoir computer is described by

\[
\frac{dr_i(t)}{dt} = \lambda \left[ p_1 r_i(t) + p_2 r_i^2(t) + p_3 r_i^3(t) + \sum_{j=1}^{M} A_{ij} r_j(t) + w_i s(t) \right].
\]

Equation 1 was numerically integrated using a 4th order Runge-Kutta integration routine with a time step of 0.1. Before driving the reservoir, the mean was subtracted from the input signal \(s(t)\) and the input signal was normalized to have a standard deviation of 1.

The other reservoir computer used in this work is a map with nodes that implement a sigmoid function. The

\[
\frac{dr_i(t)}{dt} = \lambda \left[ p_1 r_i(t) + p_2 r_i^2(t) + p_3 r_i^3(t) + \sum_{j=1}^{M} A_{ij} r_j(t) + \tanh(w_i s(t)) \right].
\]

The form of the nonlinear relationship between the nodes is determined by the experimental system [10–14]. We select two different node types to decrease the chance that our results depend on the type of node used. One node type, the polynomial node, is chosen because a polynomial is a general way to represent a nonlinearity.

In section II, we will describe reservoir computers and show how they may be trained. Section III describes our choice of how to feed signals into the reservoir computer, while section IV described how the network adjacency matrix may be characterized. All elements of the adjacency matrix, which describes the edges between nodes, start as +1 or 0, and some of the +1 edges are flipped to -1. The values of the edges are then normalized so that the maximum of the absolute value of the real part of the set of eigenvalues of the adjacency matrix is 0.5. We may then characterize the adjacency matrix by the number of symmetries it contains or by the fraction of the elements flipped from +1 to -1. In section V we describe why the rank of the covariance of the reservoir matrix is useful for describing the reservoir computer.

II. RESERVOIR COMPUTERS

We used a reservoir computer to estimate one time series signal based on a different (but related) time series signal. Figure 1 is a block diagram of a reservoir computer. There is an input signal \(s(t)\) from which the goal is to extract information, and a training signal \(g(t)\) which is used to train the reservoir computer. In [6] for example, \(s(t)\) was the \(x\) signal from a Lorenz chaotic system, while \(g(t)\) was the Lorenz \(z\) signal. The reservoir computer was trained to estimate the \(z\) signal from the \(x\) signal.

There are no specific requirements on the nodes in a reservoir computer, other than when all nodes are connected into a network, the network should be stable; that is, it should settle into a stable fixed point. Commonly used nodes include hyperbolic tangent [12] or sigmoid functions [20], but in analog experiments the node nonlinearity is determined by the experimental system [10–14]. We select two different node types to decrease the chance that our results depend on the type of node used. One node type, the polynomial node, is chosen because a polynomial is a general way to represent a nonlinearity.

The parameters for the polynomial node are chosen so that network based on these nodes is stable. The second type of node is based on a sigmoid function. Sigmoid functions are common in neural network studies [21], although the form of our node is not the same as the most commonly used sigmoid function. The form of the nonlinearities in these two node types is different enough that our results should be general for different types of reservoir computers.

The polynomial reservoir computer is described by

\[
\frac{dr_i(t)}{dt} = \lambda \left[ p_1 r_i(t) + p_2 r_i^2(t) + p_3 r_i^3(t) + \sum_{j=1}^{M} A_{ij} r_j(t) + w_i s(t) \right].
\]
sigmoid node computer is described as

\[ r_i(n+1) = \alpha \left[ \frac{1}{1 + e^{-r_i(n)}} + \sum_{j=1}^{M} A_{ij} r_j(n) + w_i s(t) \right] \]  (2)

Again, \( s(t) \) was normalized to have a mean of 0 and a standard deviation of 1.

When the reservoir computer was driven with \( s(t) \), the first 2000 time steps were discarded as a transient. The next \( N = 10000 \) time steps from each node were combined in a \( N \times (M+1) \) matrix

\[ \Omega = \begin{bmatrix} r_1(1) & \ldots & r_M(1) & 1 \\ r_1(2) & r_M(2) & 1 \\ \vdots & \vdots & \vdots \\ r_1(N) & \ldots & r_M(N) & 1 \end{bmatrix} \]  (3)

The last column of \( \Omega \) was set to 1 to account for any constant offset in the fit. The training signal is fit by

\[ h(t) = \sum_{j=1}^{M} c_j r_j(t) \]  (4)

or

\[ h(t) = \Omega C \]  (5)

where \( h(t) = [h(1), h(2), \ldots, h(N)] \) is the fit to the training signal \( g(t) = [g(1), g(2), \ldots, g(N)] \) and \( C = [c_1, c_2, \ldots, c_N] \) is the coefficient vector.

The matrix \( \Omega \) is decomposed by a singular value decomposition

\[ \Omega = USV^T \]  (6)

where \( U \) is \( N \times (M+1) \), \( S \) is \( (M+1) \times (M+1) \) with non-negative real numbers on the diagonal and zeros elsewhere, and \( V \) is \( (M+1) \times (M+1) \).

The pseudo-inverse of \( \Omega \) is constructed as a Moore-Penrose pseudo-inverse \( ^\dagger \)

\[ \Omega_{inv} = VS'U^T \]  (7)

where \( S' \) is an \( (M+1) \times (M+1) \) diagonal matrix constructed from \( S \), where the diagonal element \( S'_{ii} = S_{ii}/(S_{ii}^2 + k^2) \), where \( k = 1 \times 10^{-5} \) is a small number used for ridge regression \( ^\dagger \) to prevent overfitting. There are some guidelines for choosing \( k \) \( ^\dagger \), but in this case \( k \) is chosen large enough to to keep the coefficients from becoming extremely large but small enough to keep the fitting error from becoming too large.

The fit coefficient vector is then found by

\[ C = \Omega_{inv} g(t) \]  (8)

The training error may be computed from

\[ \Delta_{RC} = \frac{\|\Omega C - g(t)\|}{\|g(t)\|} \]  (9)

where \( \| \| \) indicates a standard deviation. The training error is used as a measure of how well the training signal \( g(t) \) may be reconstructed from the input signal \( s(t) = [s(1), s(2), \ldots, s(N)] \). In this work, \( \Delta_{RC} \) will be used as a measure of how "good" the reservoir computer is. Lower values of \( \Delta_{RC} \) indicate a "better" reservoir computer.

III. THE INPUT COUPLING VECTOR \( W \)

The coupling vector \( W = [w_1, w_2, \ldots, w_M] \) describes how the input signal \( s(t) \) couples into each of the nodes. We want to look only at the effect of varying the coupling between nodes in the reservoir computer, so \( W \) is kept fixed. We have found that setting all the elements to +1 or -1 yields a larger reservoir computer training error than setting the odd elements of \( W \) to +1 and the even elements of \( W \) to -1, so the second method (odd=+1, even=-1) was used. This choice was arbitrary, and other choices of \( W \) could be made.

IV. CHARACTERIZING THE ADJACENCY MATRIX \( A \)

As described above, the reservoir contains 100 nodes, so the size of \( A \) is \( M \times M = 100 \times 100 \).

The diagonal elements of \( A \) are all 0. Initially, all the off diagonal elements (network edges) of \( A \) are set to +1 or 0. The initial network defined by \( A \) is fully connected; every network node is connected to at least one other network node. Different configurations of the network are created by flipping some of the edges between nodes from +1 to -1. The number of elements to be flipped, \( N_f \), is chosen and then the particular elements to be flipped are chosen randomly from all the elements that have the value +1 to give many realizations of the adjacency matrix for each value of \( N_f \). After the edges are flipped, the adjacency matrix is renormalized so that the absolute value of the largest real part of the matrix eigenvalues is 0.5.

Different networks with the same \( N_f \) will reveal a range of training errors for a fixed value of \( N_f \). For each \( N_f \) value, the network is initialized to have the same adjacency matrix \( A \), with all edges equal to +1 or 0. For each \( N_f \), 20 different sets of \( N_f \) edges from the nonzero network edges are randomly chosen to be flipped, and the training error is calculated for each of the 20 different versions of the network.

We choose to characterize the network by the fraction of the edges flipped from +1 to -1, or \( \varepsilon_f \). For some values of \( \varepsilon_f \), there may be ways to permute the network.
nodes and their attached edges that leave the network unchanged. If this type of permutation is possible, we say the network contains symmetries, and we use the number of symmetries in place of $\varepsilon_f$ to characterize the network.

A. Symmetry

Symmetries in networks can have a dramatic effect on the dynamics. Here we use the concept of symmetry from graph theory [25], where a symmetry is a permutation of the nodes of the network along with the edges attached to the nodes which leave the network unchanged. This is shown in Fig. 2. A simple 4-node network is shown in Fig. 2(a). The six symmetries are obvious. Along with the identity (no permutations), there are two rotations and three mirror symmetries.

Symmetries are easy to see with small networks, but with larger networks (7 or more nodes), the detection of symmetries is difficult and quickly becomes humanly impossible, as networks with more than 10 nodes can have millions or more symmetries (see Fig. 2(b)). There is however an algorithm [26] which can quickly determine the number of symmetries and give all possible permutation matrices from the matrix of connections. We use this algorithm here.

The reason symmetries can affect the dynamics of the network can be seen from the equations of motion. Let’s apply a symmetry permutation $P$ to the reservoir system. If $\mathbf{r} = (r_1, \ldots, r_M)$, then $P\mathbf{r} = (r_{\pi(1)}, \ldots, r_{\pi(M)})$, where $\pi(i)$ is a permutation of $(1, \ldots, M)$ into a different order. That is, $P$ moves the components of $\mathbf{r}$ around into a different ordering. Note that if $P$ is a symmetry of the network, then the network coupling matrix $A$ (think of an adjacency matrix or Laplacian, for instance) must remain unchanged under the action of $P$, thus $PAP^T = A$, recalling that $P^{-1} = P^T$. This means $A$ and $P$ commute: $PA = AP$.

The equations of motion used in this paper are of the form (see Eqs. 1 and 2),

$$\frac{dr}{dt} = \mathbf{F}(\mathbf{r}) + A\mathbf{H}(\mathbf{r}) + \mathbf{w} s(t), \quad (10)$$

where $\mathbf{F}(\mathbf{r})$ is the node vector field, $\mathbf{H}(\mathbf{r})$ is the coupling function, $A$ is the coupling matrix, and $\mathbf{w} s(t)$ is the weighted driving term. If the weights for the drive term are invariant under $P$, i.e., $PW = W$ and we apply a symmetry $P$ to Eq. (10) and recall that $A$ and $P$ commute and the functions $\mathbf{F}$ and $\mathbf{H}$ are the same for all nodes, we get,

$$\frac{dP\mathbf{r}}{dt} = \mathbf{F}(P\mathbf{r}) + A\mathbf{H}(P\mathbf{r}) + \mathbf{w} s(t) \quad (11)$$

In other words, the permuted nodes $P\mathbf{r}$ have the same equation of motion as the original nodes. The consequence is that if the subsets of nodes that are permuted are started in synchronized state $(r_{\pi(i)}(t_0) = r_i(t_0))$, they will remain synchronized. This dynamic is called \textit{flow invariance}. The state where symmetry-related nodes synchronize is called \textit{cluster synchronization}, where the nodes related by symmetry permutations are synchronized among themselves, but are not synchronized with nodes in other clusters [25, 27, 28].

If this synchronized state is stable, then it is possible that the system will evolve into it. The dimension of the network will then be lower since multiple nodes will follow identical trajectories. Note that even if the symmetries are approximate, i.e. the components of $A$ and/or the node dynamics $\mathbf{F}$, and/or the weights $\mathbf{w}$ vary only slightly from a symmetric case there can still be approximate synchronization [29] where the trajectories of nodes related by symmetry permutations tend to closely follow an average trajectory. This still results in a reduction of dimension and complexity.

V. COVARIANCE RANK OF $\Omega$

We may also characterize the matrix of reservoir computer signals $\Omega$. The individual columns of $\Omega$ will be used as a basis to fit the training signal $g(t)$. The columns of $\Omega$ may be correlated with each other, so we would like to know the number of uncorrelated columns in $\Omega$.

Principle component analysis [30] states that the eigenvectors of the covariance matrix of $\Omega$, $\Theta = \Omega^T \Omega$, form an uncorrelated basis set. The rank of the covariance matrix tells us the number of uncorrelated vectors. Therefore, we will use the rank of the covariance matrix of $\Omega$,

$$\Gamma = \text{rank } (\Omega^T \Omega) \quad (12)$$

to characterize the reservoir matrix $\Omega$. We calculate the

FIG. 2: (a) Simple network with rotation and mirror symmetries. (b) A more complex network where the symmetries are not obvious.
rank using the MATLAB rank() function.

VI. INPUT SIGNALS

The first system we used to generate input and training signals is the Lorenz system [31]

\[
\begin{align*}
\frac{dx}{dt} &= c_1 y - c_1 x \\
\frac{dy}{dt} &= x (c_2 - z) - y \\
\frac{dz}{dt} &= xy - c_3 z
\end{align*}
\] (13)

with \(c_1=10\), \(c_2=28\), and \(c_3=8/3\). The equations were numerically integrated with a time step of \(t_s = 0.02\).

The second system is a nonlinear map acting on a random signal, taken from [18]

\[
\begin{align*}
x(k) &= \text{random}[0, 0.5] \\
y(k+1) &= 0.3y(k) + 0.05y^2(k) + 1.5x^2(k) + 0.1
\end{align*}
\] (14)

This system is commonly used as a test of the ability of a reservoir computer to fit a signal.

VII. SIMULATIONS: FLIPPING NETWORK EDGES FROM +1 TO -1

Because the reservoir is nonlinear, changing the adjacency matrix can have a complicated effect on the training error. In order to get good statistics, each time the number of network edges \(N_f\) to be flipped was chosen, 20 different adjacency matrices were generated with \(N_f\) randomly flipped edges. The graphs below show all 20 values of the training error for each number of edges flipped.

We begin with a \(100 \times 100\) adjacency matrix with 9800 of the network edges equal to +1. All of the diagonal elements are 0. If the number of nonzero network edges is \(N_f\), then the fraction of edges flipped is \(\varepsilon_f = N_f/N_1\). For some values of \(\varepsilon_f\), the network contains symmetries (see section IV A). For networks that contain symmetries, the network will be characterized by the number of symmetries, \(\zeta_s\). When no edges were flipped, the network contained 9.2678 \times 10^{31} symmetries, calculated using the methods from [26]. For networks that contain only one symmetry, the identity, the network will be characterized by the fraction of elements flipped, \(\varepsilon_f\).

A. Training Error vs. Number of Symmetries

1. Polynomial Nodes

These simulations used a reservoir computer with the polynomial nodes (eq. 1) and a time constant of \(\lambda = 2\). Figure 3 shows the log of training error \(\Delta_{RC}\) as a function of the log of the number of symmetries \(\zeta_s\) for a reservoir computer with polynomial nodes when the input signal \(s(t)\) is the Lorenz \(x\) signal and the training signal \(g(t)\) is the Lorenz \(z\) signal. A linear fit to the data yields slope of \(0.0052 \pm 4.32 \times 10^{-5}\) and a vertical axis intercept of \(-1.15 \pm 8.87 \times 10^{-4}\) and a \(\chi^2\) of 1.36.

Figure 4 shows the training error for the reservoir computer with polynomial nodes when the input signal \(s(t)\) is the nonlinear map signal \(x(k)\) (eq. 14) and the training signal \(g(t)\) is the nonlinear map signal \(y(k)\). The red line is a fit to the data with a slope of \(6 \times 10^{-4} \pm 5.9 \times 10^{-6}\) and an intercept of \(-1.13 \pm 1.23 \times 10^{-4}\), and a \(\chi^2\) value of 0.026.

Comparing figures 3 and 4 in both cases the training error \(\Delta_{RC}\) increases as the number of symmetries \(\zeta_s\) increases, so that when the are more ways that the individual nodes can be re-arranged without changing the network structure, the training error is larger. The polynomial nodes are more sensitive to the number of symmetries when the input signal \(s(t)\) is the Lorenz \(x\) signal than when the input signal is the random \(x(k)\) signal from eq. (14); the slope of the line in figure 3 is 0.0052, while the slope in figure 4 is 0.006.
2. Sigmoid Nodes

The reservoir computer with sigmoid nodes (eq. 2) follows the same trends as the reservoir computer with polynomial nodes, but some of the details are different. The constant $\alpha$ in eq. 2 was set to 1.0.

FIG. 5: Reservoir computer training error $\Delta_{RC}$ vs. number of symmetries $\zeta_s$ in the network for a reservoir computer with sigmoid nodes when input signal $s(t)$ is the Lorenz $x$ signal and the training signal $g(t)$ is the Lorenz $z$ signal. The red line is a linear fit with a slope of $0.0014 \pm 4.9 \times 10^{-5}$ and a vertical axis intercept of $-1.76 \pm 1.4 \times 10^{-3}$, with a $\chi^2$ of 0.2.

When the input signal $s(t)$ comes from the Lorenz system or the nonlinear map of eq. (14), the log-log plot of the training error $\Delta_{RC}$ vs. number of symmetries $\zeta_s$ is not linear (figures 5 and 6). The training error is smaller when the input signal comes from the nonlinear map than when it comes from the Lorenz system, but the slopes of the log-log plots are similar; a slope of 0.001 for the Lorenz system and 0.0014 for the nonlinear map system. These slopes derive from an attempt to fit a line to data that is not linear, so they are only crude estimates. Figure 5 does show that the reservoir computer training error is larger when attempting to fit the Lorenz $z$ signal from the Lorenz $x$ signal than when fitting the output signal $y(k)$ from the random input signal in the system of eq. (14). The logarithms of the training error in figure 5 range from -0.9 to -0.8, while in figure 6 these values go from -1.9 to -1.7.

Figure 5 shows that increasing the number of symmetries $\zeta_s$ decreases the covariance rank $\Gamma$ of the reservoir. The reservoir with sigmoid nodes produces a higher covariance rank than the reservoir with polynomial nodes, and the random input signal from the nonlinear map produces a higher covariance rank than the Lorenz $x$ signal, although the preceding section shows that increasing rank only lowers the reservoir computer training error if the node type and input signal are kept fixed.

B. Training Error vs. Fraction of Edges Flipped

If more than 100 edges were flipped in the network used in the previous section, then only one symmetry was present, the identity. For larger numbers of flipped edges, we plot the training error $\Delta_{RC}$ as a function of the fraction of edges flipped from +1 to -1, $\varepsilon_f$.

1. Polynomial and Linear Nodes

In this section we consider two types of nodes. We study the polynomial nodes of eq. (1) and a reservoir computer with linear nodes. The linear nodes are also described by eq. (1), but with parameters $p_1 = -3$, $p_2 = 0$ and $p_3 = 0$. The difference in the training error $\Delta_{RC}$ will reveal how the reservoir computer performance depends on nonlinearity.

Figure 6 shows the training error $\Delta_{RC}$ vs. the fraction of edges flipped $\varepsilon_f$ for a reservoir computer using 100 nodes when the nodes were described by the polynomial of eq. (1). The input signal $s(t)$ was the Lorenz $x$ signal, while the training signal $g(t)$ was the Lorenz $z$ signal. The figure also shows the training error when the nodes were linear, that is $p_1 = -3$, $p_2 = 0$ and $p_3 = 0$. As a larger fraction of edges in the network are flipped from +1 to -1, the training error plotted in figure 6 de-
reservoir does a better job of fitting the Lorenz reservoir variables, which may be why the polynomial variables span a much lower dimension than the polynomial with a maximum rank of 6. The linear reservoir variables increase only slightly, \( \Omega \) increases with the fraction of edges flipped, while the rank \( \Gamma \) of the covariance of the reservoir matrix does not decrease. Nonlinearity is necessary for the reservoir computer to fit the Lorenz \( z \) signal when the input signal was the Lorenz \( x \) signal.

A possible explanation for why flipping more edges in the network from +1 to -1 reduces the training error \( \Delta_{RC} \) is shown in figure 9, which shows the covariance rank \( \Gamma \) (defined in eq. 12) of the reservoir variables \( R(t) \) as a function of fraction of edges flipped, \( \varepsilon_f \).

When the nodes are polynomial nodes, figure 9 shows that the training error \( \Delta_{RC} \) did not decrease. Nonlinearity is necessary for the reservoir computer to fit the Lorenz \( z \) signal when the input signal was the Lorenz \( x \) signal.

FIG. 8: Blue circles are the reservoir computer training error \( \Delta_{RC} \) vs. fraction of edges \( \varepsilon_f \) in the network flipped from +1 to -1 for a reservoir computer with polynomial nodes when input signal \( s(t) \) is the Lorenz \( x \) variable and the training signal \( g(t) \) is the Lorenz \( z \) signal. The green squares are for the same system when the reservoir computer nodes are linear, that is in eq. (1) \( p_1 = -3, p_2 = 0 \) and \( p_3 = 0 \).

FIG. 9: Blue circles are the rank \( \Gamma \) of the covariance of the reservoir computer matrix \( \Omega \) vs. fraction of edges flipped \( \varepsilon_f \), when the nodes are polynomial nodes. The input signal \( s(t) \) was the Lorenz \( x \) variable, while the training signal \( g(t) \) was the Lorenz \( z \) signal. The green squares are the covariance rank when the nodes were linear.

When the nodes are polynomial nodes, figure 9 shows that the rank \( \Gamma \) of the covariance of the reservoir matrix \( \Omega \) increases with the fraction of edges flipped, while the rank when the nodes are linear increases only slightly, with a maximum rank of 6. The linear reservoir variables span a much lower dimension than the polynomial reservoir variables, which may be why the polynomial reservoir does a better job of fitting the Lorenz \( z(t) \) signal in figure 8.

When the input signal \( s(t) \) for the reservoir comes from the random \( x(k) \) signal from eq. (14) and the training signal is the \( y(k) \) signal, figure 10 shows the training error \( \Delta_{RC} \) as a function of fraction of edges \( \varepsilon_f \) in the network flipped from +1 to -1, for both polynomial nodes and linear nodes.

FIG. 10: Blue circles are the reservoir computer training error \( \Delta_{RC} \) vs. fraction of edges \( \varepsilon_f \) in the network flipped from +1 to -1 for a reservoir computer with polynomial nodes when input signal \( s(t) \) was the random \( x(k) \) signal from eq. (14) and the training signal is the \( y(k) \) signal. The green squares are for the same system when the reservoir computer nodes are linear, that is in eq. (1) \( p_1 = -3, p_2 = 0 \) and \( p_3 = 0 \).

Figure 11 shows the covariance rank \( \Gamma \) as a function of fraction of edges flipped, \( \varepsilon_f \) when the reservoir input signal is the random \( x(k) \) signal from eq. (14) and the training signal is the \( y(k) \) signal.

FIG. 11: Blue circles are the covariance rank \( \Gamma \) of the matrix of the reservoir computer variables \( \Omega \) as defined in eq. (12), when the nodes are polynomial nodes. The input signal \( s(t) \) was the random \( x(k) \) signal from eq. (14) and the training signal is the \( y(k) \) signal. The green squares are the covariance rank when the nodes were linear. The fraction of edges flipped from +1 to -1 in the network is \( \varepsilon_f \).

Similar to when the polynomial nodes were driven by the Lorenz system, the training error \( \Delta_{RC} \) decreases as the fraction of edges flipped \( \varepsilon_f \) increases for polynomial nodes, but not for linear nodes for the nonlinear map system. Figure 11 shows that once again, the covariance rank \( \Gamma \) increases with the fraction of edges flipped for polynomial nodes, but increases only slightly for the linear nodes. The rank actually saturates at 100 for the polynomial nodes. The signal \( x(k) \) is a random signal, so it makes sense that the reservoir \( R(t) \) would have a higher covariance rank when driven with the infinite dimensional random signal than when driven by the finite dimensional Lorenz signal.
2. Sigmoid Nodes

The training error $\Delta_{RC}$ as a function of fraction of edges flipped $\varepsilon_f$ when the input signal $s(t)$ for the reservoir with sigmoid nodes was the Lorenz $x$ signal and the training signal $g(t)$ was the Lorenz $z$ signal is shown in figure 12.

As with the polynomial nodes, the reservoir computer training error in figure 12 decreases as the fraction of edges flipped increases. Figure 13 shows the covariance rank $\Gamma$ as a function of fraction of edges flipped.

Figure 14 shows the training error $\Delta_{RC}$ vs. the fraction of network edges $\varepsilon_f$ flipped from $+1$ to $-1$ when the reservoir computer with sigmoid nodes is driven by the random $x(k)$ signal from eq. 14, while the training signal $g(t)$ was the Lorenz $z$ signal. Note unlike all the previous examples, the training error actually goes up as the fraction flipped increases past 0.4.

Figure 15 points to a cause for the increase in training error seen in figure 14. As the fraction of edges flipped in the network increases past $\varepsilon_f = 0.25$, the covariance rank $\Gamma$ saturates at 100. Because there are 100 nodes, no further increase in covariance rank is possible. In all of the previous simulations in this paper, an increase in covariance rank was associated with a decrease in training error. Increasing covariance indicates an increase in the complexity of the reservoir signals $r_i(t)$; increasing the complexity of these signals beyond the point where they are sufficient to saturate the covariance rank may lead to worse fits to the training signal.

VIII. CHANGING SPARSITY OF $A$

In the previous sections, 98% of the edges in the network were nonzero. We may ask if the level of sparsity, which we will define as $\rho$, affects our results. If there are few connections between nodes, nonlinear effects occurring in a small number of nodes may dominate the training error of the reservoir, while such effects may be averaged out when there are many connections between nodes.

For the previous sections, the sparsity of the adjacency
matrix $A$ was $\rho = 0.98$. We will calculate the training error $\Delta_{RC}$ for adjacency matrices with sparsities from $\rho = 0.01$ to $\rho = 0.5$. The number of edges flipped from +1 to -1 is fixed at $N_f = 50$. When $\rho$ was 0.98, the adjacency matrix was fully connected; every node had an input from at least one other node as well as the input signal $s(t)$. For low levels of the sparsity, the adjacency matrix may not be fully connected; some nodes may not have an input from any other node.

For the combination of the Lorenz and sigmoid nodes and random input signal from the nonlinear map system, the training error $\Delta_{RC}$ doesn’t show any trends for $\rho > 0.3$, while below $\rho = 0.3$ the rank is constant, so the rank will have no effect on the training error.

Finally, we may plot the covariance rank as a function of the fraction of edges flipped, $\varepsilon_f$. This plot is shown in figure 19.
Figure 19 echos figures 9, 11, 13 and 15. The covariance matrix rank $\Gamma$ increases as the fraction of edges flipped increases. Figure 19 only extends to $\varepsilon_f = 0.2$ because the fraction of nodes of the reservoir that were not connected to any other nodes becomes large for $\varepsilon_f > 0.2$. The fraction of edges flipped becomes large because a fixed number of edges are flipped ($N_f = 50$) while the number of nonzero edges decreases.

IX. RANDOM ADJACENCY MATRICES

The fraction of nodes flipped, $\varepsilon_f$, is a useful similarity measure when the structure of the adjacency matrix remains the same but different numbers of edges are flipped from $+1$ to $-1$. Is $\varepsilon_f$ a useful statistic when the structure of the adjacency matrix varies? We answered this question by creating random adjacency matrices whose entries were all $+1$ or 0 and flipping random fractions of the $+1$ edges to $-1$. The sparsity (fraction of nonzero edges) of the networks ranged from 20% to 100%, while the fraction of edges flipped $\varepsilon_f$ ranged from 0 to 1. Figure 20 shows the reservoir computer training error $\Delta_{RC}$ as a function of $\varepsilon_f$ for these random adjacency matrices.

For three of the combinations of the node type and input signal in figure 20, the reservoir computer training error $\Delta_{RC}$ drops as $\varepsilon_f$ increases from 0 to 0.5, and then increases as $\varepsilon_f$ goes from 0.5 to 1. If $\varepsilon_f$ is greater than 0.5, the adjacency matrix $A$ contains the same amount of variation as for $\varepsilon_f = 1 - \varepsilon_f$.

The combination of the nonlinear map signal with sigmoid nodes shows the opposite trend of the other combinations in figure 20. The reservoir computer training error $\Delta_{RC}$ increases as $\varepsilon_f$ increases from 0, with a maximum near $\varepsilon_f = 0.6$. Figure 21 reveals a possible explanation for these differences.

Figure 21 shows that for three of the combinations of node type and input signal, the covariance rank $\Gamma$ of the reservoir computer matrix $\Omega$ increases as $\varepsilon_f$ increases from 0, with a maximum that is symmetric about $\varepsilon_f = 0.5$. For the combination of the random input signal from the nonlinear map and the sigmoid nodes, the covariance rank is saturated for most values of $\varepsilon_f$. Increasing covariance rank caused by an increase in the complexity of the reservoir signals $r_i(t)$, but once the covariance rank is saturated, further increases in the complexity of the reservoir signals may lead to worse fits to the training signal $g(t)$.

The conclusion from this section is that $\varepsilon_f$, the fraction of entries of the adjacency matrix $A$ that are less than 0, is a useful difference measure for $A$ if the following conditions are true: the entries of $A$ are all either $\pm \beta$ or 0, where $\beta$ is a constant, and the covariance rank $\Gamma$ of the reservoir matrix $\Omega$ is less than $M$, where $M$ is the number of nodes in the reservoir.

X. CONCLUSIONS

We have simulated reservoir computers where the edges between the reservoir computer nodes were all $+1$ or 0, and then changed the reservoir computer network by flipping some of these edges from $+1$ to $-1$. We have done this for different combinations of node type and input signals.

If a small fraction of the edges of the adjacency matrix were flipped from $+1$ to $-1$, the number of symmetries in the adjacency matrix could function as a similarity measure for the adjacency matrix. Adjacency matrices that had more symmetries led to reservoirs that had a lower covariance rank and larger errors in fitting a training signal.

When the only symmetry in the adjacency matrix was the identity, a different measure of the variation in the adjacency matrix was necessary. Because the adjacency matrices used in this work were simple, we could use the fraction $\varepsilon_f$ of elements flipped from $+1$ to $-1$ as a measure of this variation. Increasing $\varepsilon_f$ increased the rank of the covariance of the adjacency matrix, which in most cases led to a smaller error in fitting a training signal.
We did not investigate the effect of the network statistics on the behavior of the reservoir computer. Networks whose connections are not all ±1 will require different measures of diversity. Nonrandom networks may also behave differently, and different methods for coupling nodes together may give different results.

11

XI. REFERENCES

[1] H. Jaeger, German National Research Center for Information Technology GMD Technical Report 148, 34 (2001).
[2] T. Natschläger, W. Maass, and H. Markram, Special Issue on Foundations of Information Processing of TELEMATIK 8, 39 (2002).
[3] Z. Lu, B. R. Hunt, and E. Ott, Chaos: An Interdisciplinary Journal of Nonlinear Science 28, 061104 (2018).
[4] R. S. Zimmermann and U. Parlitz, Chaos: An Interdisciplinary Journal of Nonlinear Science 28, 043118 (2018).
[5] P. Antonik, M. Gulina, J. Pauwels, and S. Massar, Physical Review E 98, 012215 (2018).
[6] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, Chaos: An Interdisciplinary Journal of Nonlinear Science 27, 041102 (2017).
[7] H. Jaeger and H. Haas, Science 304, 78 (2004).
[8] A. Jalalvand, K. Demuynck, W. D. Neve, and J.-P. Martins, Neurocomputing 277, 237 (2018).
[9] M. Lukoševičius, H. Jaeger, and B. Schrauwen, KI - Künstliche Intelligenz 26, 365 (2012).
[10] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso, and I. Fischer, Optics Express 20, 3241 (2012).
[11] G. V. der Sande, D. Brunner, and M. C. Soriano, Nanophotonics 6, 561 (2017).
[12] F. Schurmann, K. Meier, and J. Schmuel, in Advances in Neural Information Processing Systems 17 (MIT Press, 2004), pp. 1201–1208.
[13] G. Dion, S. Mejjouri, and J. Sylvestre, Journal of Applied Physics 124, 152132 (2018).
[14] D. Canaday, A. Griffith, and D. J. Gauthier, Chaos: An Interdisciplinary Journal of Nonlinear Science 28, 123119 (2018).
[15] M. Imbush and K. Yoshimura, Scientific Reports 7, 10199 (2017).
[16] S. Marzen, Physical Review E 96, 032308 (2017).
[17] T. Lymburn, A. Khor, T. Stemler, D. C. Corra, M. Small, and T. Jngling, Chaos: An Interdisciplinary Journal of Nonlinear Science 29, 023118 (2019).
[18] A. Rodan and P. Tino, IEEE Transactions on Neural Networks 22, 131 (2011).
[19] G. Manjunath and H. Jaeger, Neural Computation 25, 671 (2013).
[20] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, in The 2010 International Joint Conference on Neural Networks (IJCNN) (IEEE, 2010), pp. 1–8.
[21] F. Hadaeghi and H. Jaeger, Neurocomputing (2019).
[22] R. Penrose, Mathematical Proceedings of the Cambridge Philosophical Society 51, 406 (1955).
[23] A. N. Tikhonov, Comptes Rendus De L Academie Des Sciences De L Urss 39, 176 (1943).
[24] G. H. Golub, M. Heath, and G. Wahba, Technometrics 21, 215 (1979).
[25] M. Golubitsky, I. Stewart, and D. Schaeffer, Singularities and groups in bifurcation theory, vol. II (Springer-Verlag, New York, NY, 1985).
[26] W. Stein, SAGE: Software for Algebra and Geometry Experimentation (http://www.sagemath.org/sage/, http://sage.scipy.org/, 2013).
[27] L. M. Pecora, F. Sorrentino, A. M. Hagerstrom, T. E. Murphy, and R. Roy, Nature communications 5 (2014).
[28] Y. Cho, T. Nishikawa, and A. Motter, Phys. Rev. Lett. 119, 084101 (2017).
[29] F. Sorrentino and L. Pecora, CHAOS 26, 094823 (2016).
[30] I. T. Jolliffe, Principal component analysis (Springer, 2011).
[31] E. N. Lorenz, Journal of Atmospheric Science 20, 130 (1963).