Adding Filters to Improve Reservoir Computer Performance

T. L. Carroll
Code 6392, US Naval Research Lab
Washington, DC 20375 USA

Abstract

Reservoir computers are a type of neuromorphic computer that may be built as an analog system, potentially creating powerful computers that are small, light and consume little power. Typically a reservoir computer is built by connecting together a set of nonlinear nodes into a network; connecting the nonlinear nodes may be difficult or expensive, however. This work shows how a reservoir computer may be expanded by adding functions to its output. The particular functions described here are linear filters, but other functions are possible. The design and construction of linear filters is well known, and such filters may be easily implemented in hardware such as field programmable gate arrays (FPGA’s). The effect of adding filters on the reservoir computer performance is simulated for a signal fitting problem, a prediction problem and a signal classification problem.

Keywords: reservoir computer, machine learning

Email address: thomas.carroll@nrl.navy.mil (T. L. Carroll)
1. Introduction

A reservoir computer is a high dimensional dynamical system that may be used to do computation [1, 2]. The reservoir computer by itself will evolve to a stable fixed point; in use the reservoir computer is driven by an input signal $s(n)$. The reservoir computer is synchronized in the general sense to the input signal, meaning that the reservoir computer will follow the same trajectory every time it is driven with the same input signal (after an initial transient). To train a reservoir computer, a number of time series signals are extracted and used to do a linear fit to a training signal. The fit coefficients are the output of the training process. To use the reservoir computer for a computation, the same dynamical system is driven with a different input signal and a linear combination is made from the output signals using the coefficients found during the training process. An example of training and testing would be using the $x$ signal from the Lorenz chaotic system as the input signal and creating a linear combination of reservoir signals to fit the Lorenz $z$ signal [3]. In the computation stage, the reservoir would be driven by a signal $x'$ from the Lorenz system with different initial conditions, and the linear combination made from the training coefficients and the reservoir signals would be a close fit to the corresponding $z'$ signal.

Typically a reservoir computer is build by linking together a set of nonlinear
nodes in a directed network. A reservoir computer is similar to a recurrent neural network, but the connections between nodes do not change in a reservoir computer. As a result, reservoir computers may be built as analog systems. Examples of analog reservoir computers so far include photonic systems [4 5], analog circuits [6], mechanical systems [7] and field programmable gate arrays [8]. This analog approach means that reservoir computers can potentially be very fast, and yet consume little power, while being small and light.

One obstacle to building analog reservoir computers is that creating and connecting the analog nodes may be difficult, which is why some photonic systems [4 5] use one actual node and then use time multiplexing to create a set of virtual nodes by adding a delay loop to the laser system. This time multiplexing increases the number of nodes but slows the response time of the photonic reservoir computer. Other types of analog reservoir computers, such as those constructed from FPGAs [8] have nodes that are coupled directly, but in most systems the coupling of large numbers of nonlinear nodes into a reservoir computer network is still a research problem.

It has been shown that the number of linearly independent signals in a reservoir computer is an indicator of how well a reservoir computer can fit signals. In [9] this number was called the computational capacity, while in [10 11] it was referred to as covariance rank. The covariance rank of a reservoir computer is ultimately limited by the number of nodes in the network, but in some cases it is less than the number of nodes.

In this work I show that it is possible to increase the covariance rank (computational capacity) of a reservoir computer by adding filters to the reservoir computer output. There are many types of filters that could be used— to keep things simple, in this work I use linear finite impulse response (FIR) filters. It is known that infinite impulse response (IIR) filters can increase the fractal dimension of a signal [12], so FIR filters are used to avoid this complication. Filter design and implementation is a mature technology, and filters can be implemented in off the shelf devices such as FPGAs.

I begin by describing how adding functions to a reservoir computer may
increase the rank of the reservoir computer (Section 2). Section 3 then describes reservoir computers and how filters may be added to increase their rank. Section 4 shows how adding filters improves signal fitting, Section 5 describes the impact on prediction, and signal classification is discussed in Section 6.

2. Filters and Rank

Given a signal \( x(t) \) and some function \( f(x) \), one may create a basis of rank

2. Applying Gram-Schmidt orthogonalization,

\[
\begin{align*}
  u_1(t) &= \frac{x(t)}{\|x(t)\|} \\
  y(t) &= \frac{f(x)}{\|f(x)\|} \\
  z(t) &= y(t) - \langle u_1(t), y(t) \rangle u_1(t) \\
  u_2(t) &= \frac{z(t)}{\|z(t)\|}
\end{align*}
\]

where \( \langle \rangle \) indicates a dot product and \( \|x\| = \langle x, x \rangle \). The signals \( u_1(t) \) and \( u_2(t) \) form an orthonormal basis of rank 2, as long as \( f(x) \) is not just a multiple of \( x \). The Gram-Schmidt procedure may be repeated to create bases of higher rank using additional functions.

One type of function that can be used for \( f \) is a linear filter, in particular a finite impulse response (FIR) filter. An FIR filter is a linear system with no feedback. The design of FIR filters uses well established techniques, and implementing FIR filters in hardware is also well known; a field programable gate array (FPGA) may be used to implement these filters, for example. Because FIR filters have no feedback, stability is not a question. It is known that filters with feedback can increase the fractal dimension of a signal from a chaotic dynamical system [12], a complication that is avoided with FIR filters.

2.1. FIR filters

An FIR filter may be described by

\[
y(t) = \sum_{k=0}^{N_F} a_k x(t - k)
\]

where \( N_F \) is the filter order.
The particular type of FIR filter I used was a Bessel filter. The denominator of the transfer function an \( n' \)th order Bessel filter is the \( n' \)th order Bessel polynomial (the numerator is a constant).

I used Bessel filters with orders from 1 to 5. The Bessel filters were designated

\[ y_{i}^{\eta} (t) = \sum_{k=1}^{\eta} a_{k} \chi_{i} (t - k) \]  

where \( i \) indicates the node index from the reservoir computer and \( \eta \) is the filter order. The filter coefficients are given in Table 1.

### Table 1: Filter coefficients

| filter number | \( k = 1 \) | \( k = 2 \) | \( k = 3 \) | \( k = 4 \) | \( k = 5 \) |
|---------------|-------------|-------------|-------------|-------------|-------------|
| 1             | 1           | 0           | 0           | 0           | 0           |
| 2             | 1.7321      | 1           | 0           | 0           | 0           |
| 3             | 2.4329      | 2.4662      | 1           | 0           | 0           |
| 4             | 3.1239      | 4.3916      | 3.2011      | 1           | 0           |
| 5             | 3.8107      | 6.7767      | 6.8864      | 3.9363      | 1           |

3. Reservoir Computers

Figure 1 is a block diagram of a reservoir computer. There is an input signal \( s(n) \) from which the goal is to extract information, and a training signal \( g(n) \) which is used to train the reservoir computer.

A reservoir computer may be described by

\[ \chi_{i} (n + 1) = f [\chi_{i} (n)] + \sum_{j=1}^{M} A_{ij} \chi_{j} (n) + w_{i} s (t) \]  

where the reservoir computer variables are \( \chi_{i}(n), i = 1...M \) with \( M \) the number of nodes, \( A \) is an adjacency matrix that described how the different nodes in the network are connected to each other, \( W = [w_{1}, w_{2}, ...w_{M}] \) describes how the input signal \( s(n) \) is coupled into the different nodes, and \( f \) is a nonlinear function.

When the reservoir computer was driven with \( s(n) \), the first 1000 time steps were discarded as a transient. The next \( N = 10000 \) time steps from each node
Figure 1: Block diagram of a reservoir computer. We have an input signal $s(n)$ that we want to analyze, and a related training signal $g(n)$. In the training phase, the input signal $s(n)$ drives a fixed network of nonlinear nodes, and the time varying signals from the nodes are fit to the training signal $g(n)$ by a least squares fit. The coefficients are the result of the training phase. To use the reservoir computer for computation, a different signal $s'(n)$ is input to the reservoir computer. The time varying node signals that result from $s'(n)$ are multiplied by the coefficients from the training phase to produce the output signal $g'(n)$. Reproduced from Chaos 29, 083130 (2019) with the permission of AIP Publishing.

were combined in a $N \times (M + 1)$ matrix

$$
\Omega = \begin{bmatrix}
\chi_1 (1) & \ldots & \chi_M (1) & 1 \\
\chi_1 (2) & \chi_M (2) & 1 \\
\vdots & \vdots & \vdots \\
\chi_1 (N) & \ldots & \chi_M (N) & 1
\end{bmatrix}
$$

(5)

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

$$
h(n) = \Omega \mathbf{C}
$$

(6)

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

The pseudo-inverse of $\Omega$ is constructed as a Moore-Penrose pseudo-inverse

$$
\Omega_{inv} = V \Sigma' U^T
$$

(7)

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

$$\mathbf{C} = \Omega_{n \times g(n)}$$  \hspace{1cm} (8)

The training error may be computed from

$$\Delta_{RC} = \frac{\text{std} [\mathbf{C} - \mathbf{g}(n)]}{\text{std} [\mathbf{g}(n)]}$$  \hspace{1cm} (9)

where std[ ] indicates a standard deviation.

To learn new information, we use the reservoir computer in the testing configuration. As an example, suppose the input signal $s(n)$ was an $x$ signal from the Lorenz system, and the training signal $g(n)$ was the corresponding $z$ signal. Fitting the Lorenz $z$ signal trains the reservoir computer to reproduce the Lorenz $z$ signal from the Lorenz $x$ signal.

We may now use as an input signal $s'(n)$ the Lorenz signal $x'$, which comes from the Lorenz system with different initial conditions. We want to get the corresponding $z'$ signal. The matrix of signals from the reservoir is now $\Omega'$. The coefficient vector $\mathbf{C}$ is the same vector we found in the training stage. The testing error is

$$\Delta_{tx} = \frac{\text{std} [\mathbf{C} - z']}{\text{std} [z']}$$  \hspace{1cm} (10)

3.1. Adding Filters

Each node output $\chi_i(t)$ was passed through between one and five filters, with filter coefficients defined in Table 1. The filter outputs were $y^\eta_i(t)$, found as

$$y^\eta_i(t) = \sum_{k=1}^{\eta} a_k \chi_i(t - k)$$  \hspace{1cm} (11)

where $i$ indicates the node index from the reservoir computer and $\eta$ is the filter order.
The filter outputs were arranged in a matrix similar in form to $\Omega$

$$
\Lambda = \begin{bmatrix}
  y_1^1 (1) & y_2^1 (1) & \ldots & y_1^2 (1) & \ldots & y_{MF}^N (1) & 1 \\
  y_1^1 (2) & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  y_1^1 (N) & y_1^2 (N) & \ldots & y_2^N (N) & \ldots & y_{MF}^N (N) & 1 
\end{bmatrix}
$$

(12)

The last column is set to 1 to allow for a constant offset in the fit. For $N$ time series points, $M$ nodes and a filter order of $NF$, the size of $\Lambda$ is $N \times (M \times NF + 1)$.

Fitting proceeds as with the reservoir only, but using the full $\Lambda$ matrix instead; the fit coefficients are found as

$$
C_F = \Lambda_{\text{inv}} g(t)
$$

(13)

where $g(t)$ is the same training signal as was used for the reservoir. The training and testing errors are calculated in the same manner as for the reservoir, substituting $\Lambda$ for $\Omega$.

### 3.2. Node Types

Two different node types are used in this work. The first node type is the leaky tanh [16],

$$
\chi_i (n + 1) = \alpha \chi_i (n) + (1 - \alpha) \tanh \left( \sum_{j=1}^{M} A_{ij} \chi_j (n) + w_i s(t) + 1 \right)
$$

(14)

with the parameter $\alpha$ set to 0.35. For the leaky tanh map reservoir computer, half of the elements of the adjacency matrix $A$ were chosen randomly and set to random numbers drawn from a uniform random distribution between -1 and 1. The diagonal elements of $A$ were then set to zero. The entire adjacency matrix was then renormalized to have a spectral radius of 0.5.

The second node type was a model for the laser experiment of [4]. This system is described by

$$
\varepsilon \dot{x} (s) + x (s) = \beta \sin^2 [\mu x (s - 1) + \rho u_I (s - 1) + \phi]
$$

(15)

where $s$ is a normalized time.
This system may be modeled by the map

$$x(t + \tau_s) = \sum_{j=1}^{N} \beta H(j) \sin^2 \left[ \mu x(t - j) + W_s \left( \left\lfloor \frac{t}{M} \right\rfloor \right) + \phi \right]$$  \hspace{1cm} (16)$$

where $\lfloor \cdot \rfloor$ is the floor function and $M$ is the number of nodes. The floor function means that the value of the input signal is sampled once every $M$ time steps of the map. The variable $\beta$ was set at 0.5, $\mu = 0.1$ and $\phi = 0$. The signal $H(j)$ is the impulse response of the low pass filter in eq. (15). The low pass filter was a first order filter with a time constant of $\tau_R = 1.5 \times 10^{-6}$. The elements of the vector $W$ are drawn from a uniform random distribution between +1 and -1.

The time step in the map of eq. (16) was $t_s = 7.5 \times 10^{-8} \text{ s}$. The number of nodes $M$ was varied.

The map signal $x(t)$ was rearranged into a matrix $\Omega$ to create the set of reservoir computer nodes.

$$t = 1, 2 \ldots N$$

$$i = (t \mod M) + 1$$

$$j = \left\lfloor \frac{t}{M} \right\rfloor + 1$$

$$\Omega_{i,j} = x(t)$$  \hspace{1cm} (17)$$

3.3. Covariance Rank

The individual columns of the reservoir matrix $\Omega$ or the filter matrix $\Lambda$ may be used as a basis to fit the training signal $g(n)$. Among other things, the fit will depend on the number of orthogonal columns in $\Omega$.

Principal component analysis [17] 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 $\Omega$,

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

$$\Gamma (\Lambda) = \text{rank} (\Lambda^T \Lambda)$$  \hspace{1cm} (18)$$

to characterize the reservoir matrix $\Omega$ or the filter matrix $\Lambda$. We calculate the rank using the MATLAB rank() function. The maximum covariance rank is
equal to the number of nodes, $M$ or the number of nodes times the number of filters. The covariance rank defined here corresponds to the capacity defined in [9]. In [10, 11], higher covariance rank was associated with lower testing error.

4. Signal Fitting

The first example of adding filters to a reservoir computer will be fitting the $z$ signal from the Lorenz chaotic system based on the $x$ signal. The Lorenz system [18] is described by

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

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

4.1. Leaky Tanh Nodes

Figure 2 shows the testing error $\Delta_{tx}(\Omega)$ and the covariance rank $\Gamma(\Omega)$ as a function of the number of nodes $M$ for the leaky tanh nodes when the input signal is the Lorenz $x$ signal and the testing signal is the Lorenz $z$ signal. The notation $(\Omega)$ is used to indicate that the testing error and the covariance rank are calculated from the reservoir signal matrix $\Omega$ defined in eq. 5.

Figure 2 shows a typical result, that the covariance rank increases and the testing error decreases as the number of reservoir computer nodes goes up. The covariance rank in figure 2 is equal to the number of nodes for all values of $M$.

Figures 3 and 4 show that adding FIR filters as described in Section 2.1 can increase the covariance rank and decrease the testing error for the leaky tanh nodes. Figure 3 shows the ratio of the covariance rank calculated from the filter matrix $\Lambda$ of eq. (12) to the covariance rank calculated from the reservoir matrix $\Omega$. When only one filter is added the filter is a first order FIR filter, which amounts to the identity, so the rank is the same as for the reservoir computer without filters. When up to three filters are added, the covariance
rank of the filter matrix $\Lambda$ increases roughly as the number of added filters, but for more than three filters the rank increase is not as great for larger reservoir computers, with more than about 30 nodes. The MATLAB rank function uses a lower threshold to determine how many of the singular values of the covariance matrix are nonzero, but for large filter matrices, numerical effects mean that some of the eigenvalues will be below the threshold, so the measured increase in rank is less than the number of filters.

Figure 4 shows the ratio of the testing error calculated from the filter matrix $\Lambda$ of eq. (12) to the testing error calculated from the reservoir matrix $\Omega$ of eq. (5). When a single filter is used, it is a first order Bessel filter, and Table 1 shows that the first order Bessel filter is an identity, so the testing error for one added filter is the same as the testing error for the reservoir computer by itself. Otherwise, as the number of added filters and the number of nodes increases, the testing error when using the filter matrix $\Lambda$ to fit the Lorenz $z$ signal decreases relative to the error using the reservoir computer alone.

4.2. Laser system

The leaky tanh nodes of the previous section are usually implemented on a digital computer, in which case there is no advantage to adding filters. The
Figure 3: Ratio of covariance rank $\Gamma(\Lambda)$ when $N_f$ filters are added after the reservoir to the testing error found using only the reservoir, $\Gamma(\Omega)$. The number of filters used is $N_f$ while the number of nodes is $M$. The reservoir computer is based on the leaky tanh nodes of eq. [14].
Figure 4: Ratio of testing error $\Delta_{tx}(\Lambda)$ when $N_f$ filters are used after the reservoir to the testing error found using only the reservoir, $\Delta_{tx}(\Omega)$. The number of filters used is $N_f$ while the number of nodes is $M$. The reservoir computer is based on the leaky tanh nodes of eq. [14].
laser system of [4] was built as an actual experiment. The performance of the reservoir computer could be improved by adding more virtual nodes, but adding virtual nodes slowed the response time, so there is a more obvious advantage to adding filters to this system. The map of eq. (16) was used to simulate the laser reservoir computer.

Figure 5 shows the testing error and covariance rank for the laser simulation of eq. (16). For this system, the covariance rank is less than the number of nodes. Instead of an adjacency matrix, the coupling between nodes in the laser system comes from the low pass filter that is part of the delay loop. The result of this coupling is that the laser reservoir computer has a ring network, which is a high symmetry network, resulting in a lower covariance rank.

Figure 6 shows the ratio of the covariance rank calculated from the filter matrix $\Lambda$ of eq. (12) to the covariance rank calculated from the reservoir matrix $\Omega$ of eq. (5) for the laser reservoir computer simulation. As in figures 3 and 4 when only a single filter is present, it is equivalent to the identify.

For one to three added filters, the increase in rank for the laser system model is proportional to the number of added filters. For less than 50 nodes in the reservoir computer, this trend is still true for up to five added filters. Comparing
Figure 6: Ratio of covariance rank $\Gamma(\Lambda)$ when $N_f$ filters are used after the reservoir to the testing error found using only the reservoir, $\Gamma(\Omega)$. The number of filters used is $N_f$ while the number of nodes is $M$. The reservoir computer was modeled by the laser system of eq. (16).

Figure 6 to figure 3, added filters were more effective for increasing the rank of the laser system reservoir computer, probably because the rank of this system was lower to begin with.

The improvement in testing error when the laser reservoir computer is followed by a set of filters is shown in figure 7. This plot is similar to figure 4 for the leaky tanh nodes: the greatest improvement in testing error comes for a small number of nodes, in this case about 10 nodes and five filters.

This section on signal fitting shows that augmenting a reservoir computer can improve performance, but the improvement is larger for reservoir computers with smaller numbers of nodes. This is not necessarily bad; if it is easy to build the reservoir computer with large numbers of nodes, then performance improvement is not as useful.
Figure 7: Ratio of testing error $\Delta_{tx}(\Lambda)$ when $N_f$ filters are used after the reservoir to the testing error found using only the reservoir, $\Delta_{tx}(\Omega)$. The number of filters used is $N_f$ while the number of nodes is $M$. The reservoir computer was modeled by the laser system of eq. (16).

5. Prediction

Predicting the future time evolution of a signal given its past time evolution is a variation on fitting signals. In this case, if the input signal is $s(n)$, the training signal is $g(n) = s(n + \tau)$, where $\tau$ represents some number of time steps into the future. A useful time scale for predicting the Lorenz $x$ signal is the Lyapunov time, or the reciprocal of the largest Lyapunov exponent. For the parameters in eq. (19), the largest Lyapunov exponent is $0.9/s$, so the Lyapunov time is $T_L = 1.1$ s. For this section, the reservoir computers will be driven with the Lorenz $x$ signal and predict the $x$ signal $0.25T_L$ s into the future. At an integration time step of $0.02$ s, this amounts to $13$ points into the future. The error in prediction is calculated by a process analogous to the testing error in eq. (10), but to avoid confusion the prediction error will be called $\Delta_P$. 
Figure 8 shows the error for predicting the future of the Lorenz $x$ signal 0.25 Lyapunov times into the future, or $\Delta P$, as a function of the number of nodes $M$ for a reservoir computer using the leaky tanh nodes (no filters).

Figure 9 shows the ratio of the prediction error using a reservoir computer augmented with filters to the prediction error using the reservoir computer only.

Figure 9 shows that adding linear filters to the leaky tanh reservoir computer can lower the error in predicting the Lorenz $x$ signal. For reservoir computers up to about 60 nodes, increasing the number of nodes or filters reduces the prediction error. Prediction is closely related to signal fitting, and figure 9 for prediction error does resemble figure 4 which showed the error in fitting the Lorenz $z$ signal.

The prediction error for the laser system model is plotted in figure 10.

Adding filters to the reservoir computer modeled on the laser system can improve prediction, as shown in figure 11. Figure 11 echoes figure 9 in that the prediction error tends to get smaller as both the number of reservoir computer nodes and the number of filters increases.

6. Classification

The reservoir computers from the previous sections will be used to determine if adding filters to a reservoir computer can improve the ability to classify a set of signals. The signals in this case are the $x$ component of the 19 Sprott chaotic systems. Each of the Sprott systems was numerically integrated with a time step of 0.5.

Some of the attractors for the Sprott systems have small basins of attraction, so rather than set random initial conditions to create different realizations of each Sprott system, a long time series of the $x$ signal was generated for each of the Sprott systems. The test and training signals were taken from different sections of this long time series. The reservoir computers used to classify the Sprott signals each had $M = 100$ nodes.

The reservoir computers were trained with 100 training examples each. For each training example, the reservoir computer was first driven by a 1000 point
signal to eliminate transients, after which the next 1000 output points from each node were used to fit the training signal. Fit coefficients were found using both the reservoir matrix $\Omega$ and the filter matrix $\Lambda$. For each of the Sprott systems, the fit coefficient vectors were given by $c(j, k), j = 1 \ldots 100, k = 1 \ldots 19$, where $j$ indicated the $j$'th section of the $x$ signal and $k$ indicated the particular Sprott system. Each coefficient vector had $M$ components: $c(j, k) = [c_1(j, k), c_2(j, k), \ldots c_M(j, k)]$, where $M$, the number of nodes, was 100. For each of the Sprott systems a reference coefficient vector was defined as the mean of the coefficient vectors:

$$C(k) = \frac{1}{100} \sum_{j=1}^{100} c(j, k).$$

(20)

The set of $C(k) = [C_1(k), C_2(k), \ldots C_M(k)], k = 1 \ldots 19$ coefficients formed a reference library.

To identify the Sprott systems, the reservoir computers were again driven with a 1000 point time series of the $x$ signal from each of the Sprott systems to eliminate transients. The next 1000 points were saved in the reservoir computer matrix $\Omega$ or the filter matrix $\Lambda$. Once again, for each section a set of fit coefficients $c(j, l), j = 1 \ldots 1000, l = 1 \ldots 19$ was found, for both the reservoir computer matrix and the filter matrix.

Each time a coefficient vector was found, it was compared to the reference library according to

$$\Psi_j(l, k) = \sqrt{\sum_{i=1}^{M} (c_i(j, l) - C_j(k))^2}$$

(21)

where $i = 1 \ldots M$ indicated the components (or node numbers) of the coefficient vectors. The difference $\Psi_j(l, k)$ was computed for all 19 reference coefficient vectors $C_k$, and the value of $k$ that gave the minimum of $\Psi_j(l, k)$ was identified as the Sprott system that generated the coefficient vector $c(i, l)$.

The probability of making an error when identifying from which of the Sprott systems an $x$ signal originated is shown in figure 12. Each time a signal from a Sprott system was compared to the reference library, if the value of $k$ that gave a minimum of $\Psi_j(l, k)$ did not correspond to the Sprott system that generated
the signal, an error was recorded. The probability of error $P_E$ was the total number of errors divided by the total number of comparisons.

Figure 12 shows that adding just two filters (really just one filter, since one of the filters is the identity) to the leaky tanh reservoir computer improves the ability to identify the Sprott systems if the reservoir computer has only two nodes, but not if it has more than two nodes. Adding five filters improves the classification of signals if the reservoir computer has less than eight nodes. Once the reservoir has eight nodes, adding more nodes or adding filters gives little extra benefit.

Adding filters to a reservoir computer showed a greater advantage when fitting signals (Section 4) than when classifying signals. In fitting signals the reservoir time series acts as a basis, so the covariance rank of the reservoir output is important. Adding filters increases this rank. Classifying signals may not depend as much on how well the reservoir computer fits the signals; what is more important is that the set of fit coefficients are sufficiently different for different inputs. Adding filters to a reservoir computer does create more coefficients, but the filters are linear, so the extra coefficients may not be useful in distinguishing the different Sprott signals.

Figure 13 shows the probability of error in identifying the Sprott systems using a reservoir computer based on the laser system model of eq. (16).

Figure 13 for the laser model reservoir computer shows that adding two or five filters to this type of reservoir computer does reduce classification error when the reservoir has less than six nodes. Once the reservoir computer has six or more nodes, adding filters does not show any advantage for classification.

Adding filters to a very small reservoir computer can increase the dimension of the coefficient vector, but once the coefficient vector has enough dimensions, adding additional filters does not lower the classification error. Still, if creating a reservoir with many coupled nodes is difficult or expensive, adding linear filters to a small reservoir computer can be useful.
7. Summary

Reservoir computers should show the greatest advantage over other types of computing when they are built as analog systems, but building these systems may be difficult or expensive. Creating individual nonlinear nodes may be difficult, but the largest cost in building analog reservoir computers may be in connecting the nodes in a network. The work in this paper demonstrates that reservoir computers may be expanded if the nonlinear network is followed by a set of linear filters. The design and implementation of linear filters is well known, so there should be little cost for adding filters to the reservoir computer.

This work showed that adding filters improved the performance of two types of reservoir computer for signal fitting, for prediction and for classification. Improvements in classifying signals were largest for small reservoir computers, but small reservoir computers are where the improvement is most needed.

Reservoir computers may be expanded using other types of functions besides linear FIR filters; the linear filters were used here because they are simple to design and characterize. It is even possible to use nonlinear functions; one early paper, for example, connected linear nodes into a network and followed the linear network with nonlinear output functions [20].

This work was supported by the Naval Research Laboratory’s Basic Research Program.

References

[1] 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, vol. 148, pp. 34, 1. 2001

[2] T. Natschlaeger, W. Maass and H. Markram, ”The ”Liquid Computer”: A Novel Strategy for Real-Time Computing on Time Series”, Special Issue on Foundations of Information Processing of TELEMATIK, vol. 8, pp. 39-43, 1. 2002
[3] 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: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, pp. 041102, 4. 2017

[4] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso and I. Fischer, "Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing", *Optics Express*, vol. 20, pp. 3241-3249, 3. 2012

[5] G. Van der Sande, D. Brunner and M. C. Soriano, "Advances in photonic reservoir computing", *Nanophotonics*, vol. 6, pp. 561-576, 3. 2017

[6] F. Schurmann, K. Meier and J. Schemmel "Edge of Chaos Computation in Mixed-Mode VLSI - A Hard Liquid," in *Proc. Advances in Neural Information Processing Systems 17* Vancouver,British Columbia, Canada, 2004 pp. 1201-1208

[7] G. Dion, S. Mejiauri and J. Sylvestre, "Reservoir computing with a single delay-coupled non-linear mechanical oscillator", *Journal of Applied Physics*, vol. 124, pp. 152132, 15. 2018

[8] D. Canaday, A. Griffith and D. J. Gauthier, "Rapid time series prediction with a hardware-based reservoir computer", *Chaos*, vol. 28, pp. 123119, 12. 2018

[9] J. Dambre, D. Verstraeten, B. Schrauwen and S. Massar, "Information Processing Capacity of Dynamical Systems", *Scientific Reports*, vol. 2, pp. 514, 2012

[10] T. L. Carroll and L. M. Pecora, "Network structure effects in reservoir computers", *Chaos*, vol. 29, pp. 083130, 8. 2019

[11] T. L. Carroll, "Dimension of reservoir computers", *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 30, pp. 013102, 1. 2020

21
[12] R. Badii, G. Broggi, B. Derighetti, M. Ravani, S. Ciliberto, A. Politi and M. A. Rubio, "Dimension increase in filtered chaotic signals", *Phys Rev Lett*, vol. 60, pp. 979-982, 11. 1988

[13] R. Penrose, "A generalized inverse for matrices", *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, pp. 406-413, 3. 1955

[14] A. N. Tikhonov, "On the stability of inverse problems", *Comptes rendus de l’Académie des sciences de l’URSS*, vol. 39, pp. 176-179, 1943

[15] G. H. Golub, M. Heath and G. Wahba, "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter", *Technometrics*, vol. 21, pp. 215-223, 2. 1979

[16] H. Jaeger, Lukoševičius, D. Popovici and U. Siewert, "Optimization and applications of echo state networks with leaky- integrator neurons", *Neural Networks*, vol. 20, pp. 335-352, 3. 2007

[17] I. T. Jolliffe *Principal component analysis*, New York: London: Springer 2011

[18] E. N. Lorenz, "Deterministic non-periodic flow", *Journal of Atmospheric Science*, vol. 20, pp. 130-141, 2. 1963

[19] J. C. Sprott, "Some Simple Chaotic Flows", *Physical Review E*, vol. 50, pp. R647-R650, 2. 1994

[20] S. Boyd and L. Chua, "Fading memory and the problem of approximating nonlinear operators with Volterra series", *IEEE Transactions on Circuits and Systems*, vol. 32, pp. 1150-1161, 11. 1985
Figure 8: $\Delta P(\Omega)$ is the error for predicting the future of the Lorenz $x$ signal 0.25 Lyapunov times into the future, plotted versus the number of nodes $M$. This prediction is for the reservoir computer only, using leaky tanh nodes.
Figure 9: Prediction error ratio $\Delta P(\Lambda)/\Delta P(\Omega)$ for predicting the future of the Lorenz $x$ signal when the leaky tanh reservoir computer is augmented with up to five filters. The number of nodes is $M$, while $N_f$ is the number of filters.
Figure 10: $\Delta P(\Omega)$ is the error for predicting the future of the Lorenz $z$ signal $0.25$ Lyapunov times into the future, plotted versus the number of nodes $M$. This prediction is for the reservoir computer only, using nodes modeled on the laser system (eq. 16.25).
Figure 11: Prediction error ratio $\Delta \rho(\Lambda)/\Delta \rho(\Omega)$ for predicting the future of the Lorenz x signal when the reservoir computer based on the laser system model is augmented with up to five filters. The number of nodes is $M$, while $N_f$ is the number of filters.
Figure 12: Probability of error $P_E$ in identifying the source of an $x$ signal from one of the 19 Sprott systems as a function of the number of nodes $M$ in the leaky tanh reservoir computer.
Figure 13: Probability of error $P_E$ in identifying the source of an $x$ signal from one of the 19 Sprott systems as a function of the number of nodes $M$ in the reservoir computer based on the laser system model of eq. (16).

![Graph showing $P_E$ vs. $M$ for different numbers of filters: reservoir only, 2 filters, and 5 filters.](image-url)