A Reservoir Computer Training Methodology to Predict the Time Evolution of Arbitrary Points on Chaotic Attractors

DJ Passey [*] [†]

Abstract

The abstract goes here.

Reservoir computers, chaos, complex systems, prediction, machine learning.

1 Introduction

Due to their ability to replicate chaotic attractors, reservoir computers have seen a surge in popularity within the field of chaotic dynamical systems [**?**, **?**, **?**]. These machine learning models were able to capture the "climate" of attractors and maintained key dynamic properties such as Lyapunov exponents [**?**].

In the case of chaotic systems, reservoir computers are able to extrapolate the future trajectory of the training orbit. However, since the systems are chaotic, small differences between orbits increase exponentially with time, and all predictions eventually diverge. While the reservoir computers in the previously cited work can predict the future evolution of the training orbit, they cannot accurately predict the trajectory of an arbitrary point on the attractor.

Building on existing methods, we propose a training methodology that links reservoir initial conditions to learned system initial conditions and with it, successfully predict the evolution of arbitrary points on the attractor.

1.1 Reservoir Computers: An Accessible Machine Learning Model

The success of reservoir computers in reproducing chaotic attractors suggests that they are suitable for modeling, predicting and controlling complex systems. Unlike most state of the art machine learning models, reservoir computers are

[*]DJ. Passey is with the Department of Mathematics, University of North Carolina at Chapel Hill, Chapel Hill, NC, 27517 USA e-mail: djpassey@unc.edu

[†]Manuscript received November, 2, 2020

within reach for most scientists and engineers. Training a reservoir computer involves computing a numerical solution to an ODE and a single least squares (or Tikhanov regression) computation. It is possible to implement a simple reservoir computer in fifteen lines of code.

Modern reservoir computers are descendants of Echo State Networks and Liquid State Machines [**?**, **?**]. Each of these models makes use of a fixed internal network as a source of non-linearity. This "reservoir" network is never trained, instead training is focused entirely on a readout mapping that translates complicated network dynamics into useful signals. Reservoirs typically consist of complex networks, but a variety of physical systems such as circuits, optical arrays and in-vitro cell cultures were shown to successfully produce the signals needed for effective reservoir computation [**?**]. Famously, researchers once used a bucket of water as a reservoir, leveraging the non-linearity of the waves to project input into a higher dimensional embedding [**?**].

Currently, researchers are exploring chaining together princial component analysis, reservoir computers, neural nets, and support vector machines for better time series classification [**?**]. In addition to developing better architectures, there is work developing better reservoirs. The field of optics shows considerable promise due to low energy cost, intrinsic parallelism and speed [**?**].

1.2 Chaotic Systems

1.3 Prediction of Complex Systems

2 Standard Reservoir Computer Training

In the following section, we outline standard training methodology for reservoir computers.

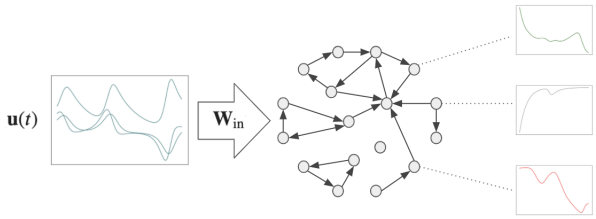


Fig. 1: **An untrained reservoir computer driven by an input signal**

Setup: We create a reservoir computer as follows. Let A be an $n_r \times n_r$ adjacency matrix where A_{ij} represents the weight of the connection from node j to node i. Let $\mathbf{r}(t)$ be an n_r-dimensional vector valued function of that represents the states of nodes in the network. If we desire to replicate the dynamics of $\mathbf{u}(t)$ (a n_s dimensional vector valued function), then we allow the reservoir nodes to

evolve according to:

$$\frac{d\mathbf{r}}{dt} = -\gamma\big[\mathbf{r}(t) + f\big(A\mathbf{r}(t) + \sigma W_{\text{in}}\mathbf{u}(t)\big)\big] \tag{1}$$

where $\gamma > 0$ and $\sigma > 0$ are hyper parameters, f is an activation function and W_{in} is an $n_r \times n_s$ dimensional read-in matrix. In the system defined above, $A\mathbf{r}(t)$ represents recurrence, and governs how the nodes in the reservoir interact. The $W_{\text{in}}\mathbf{u}(t)$ term causes each reservoir node to receive a linear combination of each dimension of $\mathbf{u}(t)$. Since W_{in} is typically initialized randomly, each node will receive a random sum of the training data. The function f should be non-linear and should be bounded to prevent the system from blowing up. Good choices for f are tanh and sigmoid.

If $\mathbf{u}(t)$ for $0 \leq t \leq T$ is the training data, we proceed by discretizing the domain into $0 = t_0 < t_1 < ... < t_m = T$, choosing an initial condition $\mathbf{r}_0$ (the algorithm presented here will give a thorough treatment of initial conditions) and using a numerical solver to solve Equation **??** for $\mathbf{r}(t_i)$ $i \in \{0, ..., m\}$.

Learning Step: We now seek to approximate $\mathbf{u}(t)$ with a linear combination of the reservoir states $\mathbf{r}(t)$. This is equivalent to solving for the $n_s \times n_r$ matrix W_{out} that minimizes:

$$\frac{1}{m}\sum_{i=0}^{m} \|W_{\text{out}}\mathbf{r}(t_i) - \mathbf{u}(t_i)\|_2$$

This is often solved using a Tikhanov regression giving,

$$W_{\text{out}} = U^T R(R^T R - \alpha I)^{-1} \tag{2}$$

where α is the regularization parameter, R is a $m \times n_r$ matrix where the i^{th} row of R is equal to $\mathbf{r}(t_i)^T$ and U is an $m \times n_s$ matrix where the i^{th} row of U is $\mathbf{u}(t_i)^T$.

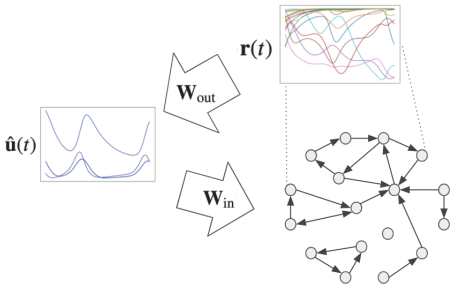


Fig. 2: **An trained reservoir approximating the training signal without any input**

Prediction: Once the readout is obtained, the reservoir computer is run as a stand-alone dynamical system to obtain a prediction. Since $\hat{\mathbf{u}}(t) = W_{\text{out}}\mathbf{r}(t)$

is an approximation to $\mathbf{u}(t)$ it is possible that for $t > T$, the approximation $\hat{\mathbf{u}}(t)$ will continue to predict $\mathbf{u}(t)$. Using the fact that $\mathbf{u}(t) \approx W_{\text{out}}\mathbf{r}(t)$ and substituting this into Equation **??** produces a new equation for the evolution of node states that is independent of $\mathbf{u}(t)$:

$$\frac{d\mathbf{r}}{dt} = -\gamma\big[\mathbf{r}(t) + f\big(A\mathbf{r}(t) + \sigma W_{\text{in}} W_{\text{out}}\mathbf{r}(t)\big)\big] \tag{3}$$

To predict $\mathbf{u}(t)$ for $t > T$, we need an initial condition $\mathbf{r}_0$, for the reservoir nodes. Conveniently, since $W_{\text{out}}\mathbf{r}(T) \approx \mathbf{u}(T)$, setting $\mathbf{r}_0 = \mathbf{r}(T)$ ensures that our reservoir node initial condition, $\mathbf{r}_0$ will map via W_{out} to the desired system initial condition, $\mathbf{u}_0$. With the selected $\mathbf{r}_0$ we numerically solve Equation **??** for $\mathbf{r}(t)$ when $t > T$ and apply the readout, W_{out}, to make a prediction: $\hat{\mathbf{u}}(t) = W_{\text{out}}\mathbf{r}(t)$.

2.1 Problems with Existing Training Techniques

Problems With Random Initialization: It is known that the prediction $\hat{\mathbf{u}}(t)$ for $t > T$ is not always close to $\mathbf{u}(t)$ [**?**]. The ability of the reservoir depends on the parameters γ, σ, A and W_{in}. In practice, random initialization of A and W_{in} adds some amount of variance to the predictive ability. Often, multiple reservoir computers are trained until one is found that adequately produces the desired dynamics. Statistical methods for verifying that a particular reservoir computer is "good" can be found in [**?**]. In addition, basic hyper parameter optimization can help reduce variance in prediction ability.

Since a sparse A matrix is preferable for computational efficiency, A can usually be interrupted as the adjacency matrix of a network. Because most machine learning models do rely on arbitrary network structures, reservoir computing provides a link between complex network theory and learning problems. How to construct the optimal network topology is still an open question and though multiple studies on this subject exist, results are mixed [**?**, **?**, **?**].

Problems With Initial Conditions: In order to produce reservoir node states from Equation **??**, it is necessary to choose an initial condition, $\mathbf{r}_0$ from which the nodes will evolve. When picking $\mathbf{r}_0$ for the first time, there need not be any connection to $\mathbf{u}(0)$, in fact, $\mathbf{r}_0$ can be initialized randomly. When a random initialization is used, it is common to give the reservoir nodes extra time to evolve and "eliminate transients" in the node states [**?**]. For example, researchers might use a random initial condition and solve Equation **??** for $t \in [-T, T]$, but only use $\mathbf{r}(t)$ and $\mathbf{u}(t)$ for $t \in [0, T]$ to solve for W_{out}.

As explained before, this method naturally lends itself to the prediction of $\mathbf{u}(t)$ for $t > T$ because $W_{\text{out}}\mathbf{r}(T) \approx \mathbf{u}(t)$. Thus setting the node initial condition to $\mathbf{r}(T)$ guarantees that the initial readout of the node states will be close to the desired value ($\mathbf{u}(T)$) and if the reservoir computer training was successful, $W_{\text{out}}\mathbf{r}(t) \approx \mathbf{u}(t)$ for $t > T$ with $\mathbf{r}(t)$ obtained by solving Equation **??**.

However, what if we want to predict the response of the learned system from an arbitrary initial condition $\mathbf{u}_0$? It is possible that if the reservoir computer was trained on appropriate data, it could extrapolate the evolution of the

learned system from an unseen initial condition. The challenge here, is choosing the appropriate $\mathbf{r}_0$. Current techniques in reservoir computing for chaotic dynamical offer little insight on this subject. As such, this work will describe a methodology for choosing appropriate initial conditions and training so that reservoir computers can accurately predict the evolution of the learned system from arbitrary initial conditions.

3 Proposed Training Methodology

In order to predict the orbit of the learned system from a given initial condition, we need some way of linking reservoir state to the state of the system.

While we've seen that $\mathbf{r}_0$ can be completely random and reservoir computers are still able to predict, this will not work in our case. The issue with a random initial condition is that it places reservoir nodes in an arbitrary location in the usually high dimensional reservoir space. This location in the reservoir space has no connection to the location in the system space.

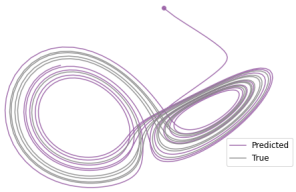


Fig. 3: Standard reservoir computer prediction of the Lorenz system from an arbitrary initial condition $\mathbf{u}_0$ on the attractor. The true evolution of $\mathbf{u}_0$ is pictured in gray. The reservoir computer prediction wanders through space for some time before settling on the attractor. Though the reservoir captures the "climate" of the Lorenz system it fails to predict the orbit of $\mathbf{u}_0$. (For this plot, the reservoir initial state was set to $W_{\text{out}}^{\dagger}\mathbf{u}_0$.)

Since we know that W_{out} projects the reservoir space onto the system space, it is natural to consider the pseudoinverse, $W_{\text{out}}^{\dagger}$, and apply it to $\mathbf{u}_0$ to obtain an initial condition for the reservoir. Unfortunately, this method does not work. As illustrated by Figure **??**, setting the initial reservoir state in this way produces an initially transient orbit that eventually finds it's way to the attractor. It is remarkable that this orbit eventually does mimic the Lorenz system's behavior on attractor. This suggests that there is a manifold in the reservoir node space that corresponds to the Lorenz attractor, and it appears to be an attracting manifold. We can use this to our advantage.

Assuming that this attracting manifold really exists, we need to draw initial reservoir states from this manifold and also need them to be associated with the appropriate system initial condition $\mathbf{u}_0$. We can identify this manifold with the tools of nonlinear system theory.

3.1 Reservoir Node Fixed Points

Similar to work done here:

https://link.springer.com/article/10.1007/s12559-019-09634-2.

Beginning with the untrained reservoir system,

$$\frac{d\mathbf{r}}{dt} = -\gamma\big[\mathbf{r}(t) + f\big(A\mathbf{r}(t) + \sigma W_{\text{in}}\mathbf{u}(t)\big)\big]$$

we make one simplifying assumption and let $A = \rho I$ where I is the $n_r \times n_r$ identity, and $\rho > 0$ is the spectral radius. This choice removes recurrence in the reservoir and effectively decouples the system so that we can consider the evolution of reservoir nodes individually. This simplification is justified by studies showing that extremely sparse networks are more effective at learning, the benefit of recurrence is limited. With this simplification, we can study the dynamics of a single node independently with the system:

$$\frac{dr_i}{dt} = -\gamma\big[r_i(t) + f\big(\rho r_i(t) + \sigma \mathbf{w}_i^T\mathbf{u}(t)\big)\big] \tag{4}$$

Here $\mathbf{w}_i^T$ is the i^{th} row of W_{in}. We can solve for the fixed points of this system by setting $\frac{dr_i}{dt} = 0$. This produces,

$$-r_i = f\big(\rho r_i + \sigma \mathbf{w}_i^T\mathbf{u}(t)\big)$$

For a given time t_0, let $a = \sigma \mathbf{w}_i^T\mathbf{u}(t_0)$ we have fixed points where the line $y = -r_i$ intersects the function $f(\rho r_i + a)$. Commonly used functions for f include $\tanh(x), \sin(x)$ and sigmoid. Graphical analysis shows that for each of these functions, fixed points of the system will be located within $[-1, 1]$, (or more precisely, $[0, 1]$ for sigmoid).

3.2 Stability of Reservoir Node Fixed Points

Since the uncoupled system is one dimensional, we can compute the stability of the fixed points with,

$$\lambda = -\gamma\Big(1 + \rho f'(\rho r^\star + a)\Big) \tag{5}$$

where $r^\star$ is the fixed point. If $\lambda < 0$, the fixed point is stable and attracting and it is unstable and repelling if $\lambda > 0$.

Since γ is positive, we know that,

$$\lambda < 0 \text{ if } 1 + \rho f'(\rho r^\star + a) > 0.$$

Because $\tanh(x)$ and sigmoid have non-negative derivatives, and ρ is positive, $1 + \rho f'(\rho r^\star + a) > 0$ for any $r^\star$.

Therefore, for these two activation functions, fixed points are always attracting. This is not the case when $f(x) = \sin(x)$ since it's derivative, $f'(x) = \cos(x)$ can be negative. We know that $-1 \leq \cos(x) \leq 1$, so choosing $\rho < 1$ ensures that any fixed points are attracting. However, when $\rho > 1$, the fixed points may

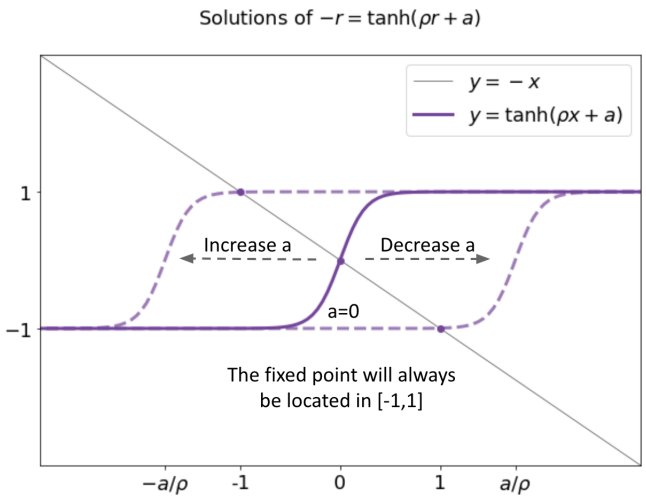


Fig. 4: **Attracting fixed point of a reservoir node with** tanh **nonlinearity.** In an untrained reservoir, the value of a varies with the state of the training signal $\mathbf{u}(t)$. Therefore, the reservoir node system will have a single attracting fixed point whose location changes according to the training signal. Thus, the reservoir node will "follow" the movement of this fixed point through space as $\mathbf{u}(t)$ evolves. However, no matter the value of $\mathbf{u}(t)$, this point will always lie in [-1, 1]. (Analysis of a sigmoidal activation function is similar.)

be repelling. As pictured in figure **??**, increasing ρ increases the frequency of sin which increases the number of fixed points. Because ρ is larger than one, some of the fixed points are attracting (in blue) and some are repelling (in red). Combining this with the changing value of a as $\mathbf{u}(t)$ changes creates potential for a complicated dynamical picture. If ρ is sufficiently large, the evolving reservoir node will face a landscape of repelling and attracting fixed points whose locations change with $\mathbf{u}(t)$.

3.3 A Note on Linearly Independent Node States

Since it is the goal of a reservoir computer to project it's node states, $\mathbf{r}(t)$ onto the training signal, $\mathbf{u}(t)$, we would presume that linearly independent $r_i(t)$ would lead to better reconstruction of the training signal.

A clear way to create nearly linear *dependent* $r_i(t)$ is to set $f(x) = \tanh(x)$, and use $\rho = 1$ with a large *sigma* and positive $u(t)$ and W_{in} with strictly positive entries. Then no matter the variation in $\mathbf{u}(t)$, if $a = \sigma \mathbf{w}_i^T \mathbf{u}(\mathbf{t})$ we will consistently have $a >> 0$. Thus, the fixed point of every reservoir node will consistently remain close to -1 with very little variation in response to $\mathbf{u}(t)$.

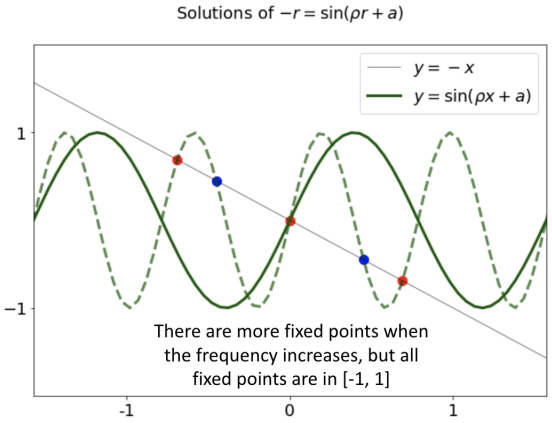


Fig. 5: **Fixed Points of a reservoir node With a** sin **non-linearity.** Fixed points in blue are attracting, fixed points in red are repelling. In the decoupled system, as the spectral radius increases, the number of fixed points will also increase. Unlike $\tanh(x)$ or a sigmoidal activation function, $sin(x)$ can produce more than one fixed point which may be attracting or repelling. Combining this with the fact that a changes as the training signal $\mathbf{u}(t)$ changes, creates a system of moving attracting and repelling fixed points.

After the transient period, every reservoir node will move to -1 and stay there. This creates linearly dependent node states which would decrease the signal reconstruction accuracy.

This un-example illustrate the importance of having fixed points in the reservoir system that move with $\mathbf{u}(t)$. The fixed points will not move as well when a is large relative to ρ. Clearly, a $sin(x)$ activation function with a large ρ will create more linearly independent node states, however, it is possible that in the mix of attracting and repelling fixed points, the nodes might lose clear correlation to the training signal. Additionally, this mix of attracting and repelling fixed points can case the untrained system to become chaotic, even in the absence of input. The reservoir system:

$$\frac{d\mathbf{r}}{dt} = -(\mathbf{r} + \sin(A\mathbf{r}))$$

which we note is the is the untrained reservoir from Equation **??** with $\gamma = 1, \sigma = 0$, is chaotic when

$$A = 3\pi \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

. The attractor of this system is pictured in Figure **??**.

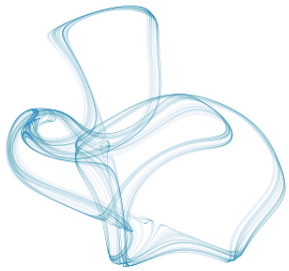


Fig. 6: **Example of reservoir nodes behaving chaotically in the absence of input with a sinusoidal activation function.** Due to the mix of attracting and repelling fixed points in a reservoir computer with a $\sin(x)$ activation function, it is not hard to create a system of reservoir nodes that behave chaotically, even without input from the training signal.

Since this reservoir behaves chaotically, even without input from the training signal, it raises questions about its ability to learn. This kind of dynamic environment could make prediction more difficult, since the reservoir itself would not behave predictably. It is hard to know without further experimentation if a $sin(x)$ activation function would help or hurt a reservoir computers ability to learn.

4 Algorithm

4.1 Fixed Point Initial Condition

Based on this analysis, we will focus the remainder of the paper on tanh and sigmoidal activation functions. Bases on our fixed point analysis, we see that to eliminate transient orbits in the reservoir , the initial reservoir condition should be set to a fixed point of the reservoir system. We can compute the fixed point corresponding tp $\mathbf{u}(t) = \mathbf{u}_0$ by solving for a solution, $\mathbf{r}^\star$ to

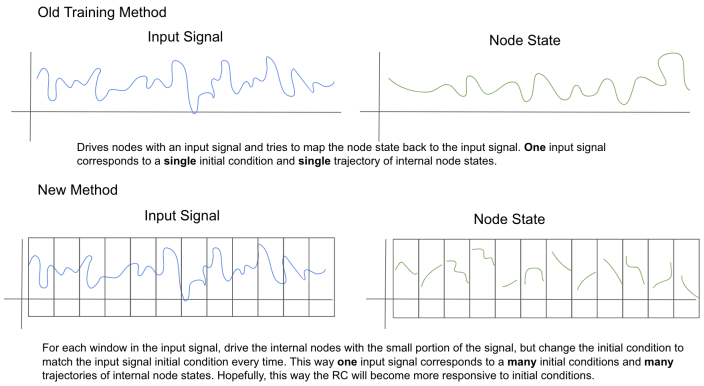
$$0 = -\gamma\big[\mathbf{r}(t) + f\big(A\mathbf{r}(t) + \sigma W_{\text{in}}\mathbf{u}_0\big)\big]$$

.

Because this fixed point depends on $\mathbf{u}_0$, it associates the learned system with the internal system. Since the state of the reservoir at this point is stationary, there is no transience in the system. As $\mathbf{u}(t)$ changes with time, the location of this fixed point will move. Because we have excluded $\sin(x)$ from our analysis, the fixed point must be attracting. Thus, as it moves, the reservoir node states will follow it, synchronizing their behavior with $\mathbf{u}(t)$ immediately.

4.2 Window Training

Now that we have chosen a natural initial condition, we turn our focus to training the reservoir computer to learn the correspondence between learned system



Tab. 1: Results of testing the proposed algorithm on three chaotic systems.

System:	Rossler		Thomas		Lorenz	
Training Method:	Standard	Proposed	Standard	Proposed	Standard	Proposed
Predict Training Signal	**59.5**	38.6	20.9	**51.2**	**6.0**	3.97
Predict Random Initial	0.0	**20.7**	0.44	**31.9**	0.01	**2.21**
System Norm	2.6	**1.2**	4.02	**3.9**	**0.16**	0.18
Lyapunov Exp. Approx.	0.081	0.07	0.048	0.044	0.88	0.87

initial conditions and reservoir initial conditions. We do this, by exposing the reservoir computer to many initial conditions instead of just one. This is accomplished by breaking the training signal into overlapping windows and resetting the initial condition for each window.

An innovation from [**?**], to break up Tikanov regression computation into batches simplifies this process:

Typically, when training a reservoir, we take an initial condition $\mathbf{u}_0$, and, using an ODE solver, generate an array $(m \times n_d)$ of samples, U from the training system. Next, we set the reservoir initial condition with $\mathbf{r}_0$ and use and ODE solver to obtain R, the $(m \times n_r)$ array of reservoir node states.

The training step invloves solving $||W_{\text{out}}R - U||_2$ for minimizer W_{out}. Tikanov regression with regularization parameter α gives the solution

$$W_{\text{out}} = (R^T R - \alpha I)^{-1} R^T U$$

When n is large the cost of storing R in memory can be prohibitive. This problem is addressed by computing $R^T R$ and $R^T U$ in batches. We write

$$R = [R_1 \; R_2 \; R_3 \; \cdots \; R_k]^T$$

where each Ri is an $(m_i \times n_r)$ array and $\sum_i^k m_i = m$.

Then

$$R^T R = \sum_i^k R_i^T R_i$$

If we break up U into

$$U = [U_1 \; U_2 \; \cdots \; U_k]^T$$

where each U_i has dimension $(m_i \times n_d)$ we can compute $R^T U = \sum_i^k R_i^T U_i$. This allows us to compute W_{out} in batches. If the resulting matrices, $R^T R$ and $R^T U$ are saved, W_{out} can be updated with additional data.

When solving for W_{out}, we point out that there is no requirement that R_i corresponds with R_j when $j \neq i$. That is, whereas before, we assumed that R was a continuous stream of node states, and W_{out} projects R onto U, in actuality W_{out} attempts to send the rows of R to the associated rows in U. Therefore, the matrix U may contain orbits from multiple different initial conditions as long as the corresponding rows of R are the response of the reservoir computer to those orbits and their corresponding initial conditions.

Therefore we can break up the training signal into overlapping time windows and drive the reservoir computer with each window separately. For each window, we reset the reservoir internal initial condition to correspond with the first input condition of the particular time window. After this we can map the driven internal states back on to the concatenation of time windows. This allows us to train a reservoir computer to replicate an orbit starting from multiple places along the orbit. The number and size of time windows is a hyper parameter that can be tuned.

5 Results

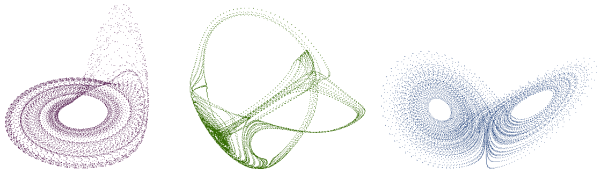


Fig. 7: **Chaotic systems used to test reservoir prediction from random initial points on the attractor.** In order from left to right, the Rossler attractor, Thomas' cyclically symmetric attractor, the Lorenz attractor.

To test the algorithm outlined above, 3 chaotic systems were selected: The Lorenz system, the Rossler system and Thomas' cyclically symmetric system [**?**, **?**, **?**]. For each system, hyper parameters were optimized for the *standard training algorithm* via a random search. These same hyper parameters were used to test the proposed new algorithm.

For 200 different random reservoir computers, the following metrics were tested. First, the length of time that the reservoir computer could predict the

training orbit with absolute error less that 0.2. Second, a random initial condition on the attractor was selected and the length of time that the reservoir computer could predict the evolution of this random initial condition with absolute error less than 0.2 was recorded.

Solution Error We measured how well the orbit predicted by the reservoir solved the given system equations. Our approach here was to interpret a reservoir computer prediction as a continuous function $\hat{\mathbf{u}}(t)$. Since the reservoir computer is trying to learn a system of the form $\mathbf{u}' = F(\mathbf{u}, t)$, a successful prediction, $\hat{\mathbf{u}}(t)$, should be a solution the the equation above. That is,

$$\hat{\mathbf{u}}' = F(\hat{\mathbf{u}}, t)$$

if $\hat{\mathbf{u}}(t)$ is a perfect prediction. We can therefore measure the degree that $\hat{\mathbf{u}}(t)$ is not a solution with the expression,

$$\|F(\hat{\mathbf{u}}(t), t) - \hat{\mathbf{u}}'(t)\|$$

Because computation of $\hat{\mathbf{u}}(t)$ in practice results in the discrete values, $\hat{\mathbf{u}}(t_i)$ for equally spaced $t_1, t_2, ...t_m$, we do not have an explicit expression for $\hat{\mathbf{u}}'$. We work around this by approximating $\hat{\mathbf{u}}'$ with a centered finite difference scheme:

$$\hat{\mathbf{u}}'(t_i) \approx \frac{\hat{\mathbf{u}}(t_{i+1}) - \hat{\mathbf{u}}(t_{i-1})}{2h}$$

where $h = t_i - t_{i-1}$ for any $1 \leq i \leq m$.

Thus, for a particular time, t_i we can approximate the difference $\|F(\hat{\mathbf{u}}(t_i), t) - \hat{\mathbf{u}}'(t_i)\|$ with,

$$\|F(\hat{\mathbf{u}}(t_i), t) - \frac{\hat{\mathbf{u}}(t_{i+1}) - \hat{\mathbf{u}}(t_{i-1})}{2h}\|$$

How well this predicted orbit fit the system equations was also measured (This measurement needs an explanation and some development). Finally, the largest lyapunov exponent of the reservoir computer was estimated. The mean these measurements taken from 200 experiments are summarized in Table **??**.

In **??**, we see that the proposed algorithm achieves the prediction of a random initial condition while standard training techniques do not. It is interesting to note that the proposed algorithm does not always predict the trajectory of the training signal as well as standard training techniques. This may be due to the small window size preventing the reservoir computer from learning long term dynamics.

6 Conclusion

In conclusion, using the tools of dynamical systems theory, we were able to derive an algorithm capable of training a reservoir computer to predict the evolution of a chaotic system from arbitrary initial conditions.

Tab. 2: Hyper Parameters

Parameter	Rossler	Thomas	Lorenz
γ	5.63	12.6	19.1
Reservoir mean degree	0.21	2.2	2.0
Tikhanov α	2×10^{-7}	4×10^{-4}	6×10^{-7}
σ	0.078	1.5	0.063
Spectral radius of A	14.6	12.0	8.472