Project Report

Summer 2022

Application of Echo State Networks in Chaotic Time Series Prediction

Author:
Reetish Padhi

Submitted to:
Prof. Amit Apte

August 14, 2022

Table of Contents

1 Echo state Networks

Echo state network [1] is a type of Recurrent Neural Network, part of the reservoir computing framework. The basic idea behind the ESN is to project the input vector into a higher dimension space (reservoir state) and then train the weights using linear regression to map the higher dimension (reservoir state) state to the desired output. Since we are only performing a linear regression, the training time for such a network is relatively short, and it can produce good results with less data and without excessive hyperparameter optimization.

1.1 Architecture of ESN

An ESN is composed of three layers:

- Input Layer
- Reservoir
- Output Layer

The input layer is connected to the reservoir by the transformation matrix W^{in}, which projects the input vector onto the reservoir. The reservoir is made up of N nodes, and the adjacency matrix W describes the connections between them. The W^{out} transformation connects the Reservoir to the output layer. In some cases we also allow connections from the output layer to the reservoir by W^{back}. During the training process, only the weights of W^{out} are learned. The remaining weights are held constant.

1.2 Formalism

Let $U_i(t)$, $1 \leq i \leq K$ denote a K dimensional input series with $Y_j(t)$, $1 \leq j \leq L$ as the L dimensional output/target sequence we want to predict. Activations of the input units are $u(t) = (u_1(t), u_2(t) \ldots, u_K(t))$, of internal/reservoir units are $x(t) = (x_1(t), x_2(t) \ldots, x_N(t))$ and of output units are $y(t) = (y_1(t), y_2(t) \ldots, y_L(t))$. Real-valued connection weights are collected in a $N \times K$ weight matrix $\mathbf{W}^{\text{in}} = (w_{\text{ij}}^{\text{in}})$ for the input weights, in an $N \times N$ matrix $\mathbf{W} = (w_{\text{ij}})$ for the internal reservoir connections, in an $L \times (K + N)$ matrix $\mathbf{W}^{\text{out}} = (w_{\text{ij}}^{\text{out}})$

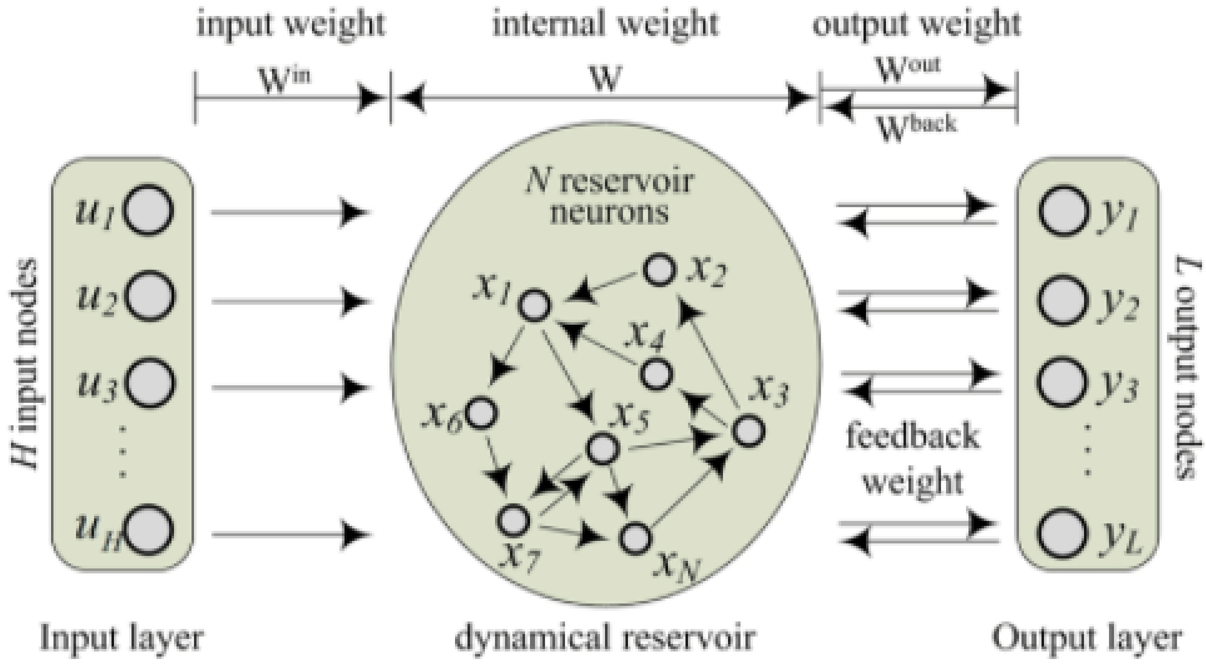
for the connections to the output units, and in a $N \times L$ matrix $\mathbf{W}^{\text{back}} = (w^{\text{back}}_{\text{ij}})$ for the connections that project back from the output to the internal units.

1.3 State Equations

The reservoir state is updated at every step according to:

$$x(t+1) = (1-\alpha)x(t) + \alpha f(\mathbf{W}^{\text{in}}u(t) + \mathbf{W}x(t) + \mathbf{W}^{\text{back}}y(t)) \tag{1}$$

where $0 \leq \alpha \leq 1$ is the leaking rate and f is the activation function.



1.4 Training equations

The training procedure involves finding optimum weights for $\mathbf{W}^{out}$ using ridge regression where β is the regularization coeff, $\mathbf{w}^{\text{out}}_i$ is the i^{th} column of $\mathbf{W}^{out}$ and $\|.\|$ is the L2 norm.

$$\mathbf{W}^{\text{out}} = \arg\min_{\mathbf{W}^{\text{out}}} \frac{1}{L}\sum_{i=1}^{L}(\sum_{t=1}^{T}\|y^{\text{pred}}_i(t) - y^{\text{target}}_i(t)\|^2_2 + \beta\|\mathbf{w}^{\text{out}}_i\|^2_2) \tag{2}$$

Once the weights are trained, future predictions can be made using:

$$y^{\text{pred}}(t) = \mathbf{W}^{\text{out}}(u(t), x(t))^{\intercal} \tag{3}$$

The value of $\mathbf{W}^{\text{out}}$ that minimises the quantity in (2) is given by:

$$\mathbf{W}^{\text{out}} = y^{\text{target}} X^{\intercal}(XX^{\intercal} + \beta I)^{-1} \tag{4}$$

where X is the $(N+H) \times T$ matrix obtained by concatenating all the input and reservoir states $(u_i(t), x_i(t))$ for each training point i $(0 \leq i < T)$,y^{target} is the corresponding $L \times T$ matrix such that the i^{th} columns are composed of $y_i(t)$ corresponding to $u_i(t)$. β corresponds to the regularization parameter.

Let's say given $\{u(t)\}_{t=0}^{t_0}$ we want the ESN to generate the next t_1 datapoints $\{u(t)\}_{t=t_0+1}^{t_0+t_1}$. When we want the ESN to generate a sequence without input, we can simply take $u(t+1) = y(t)$ and continue this iteratively(provided $K = L$).

1.5 Echo State Property

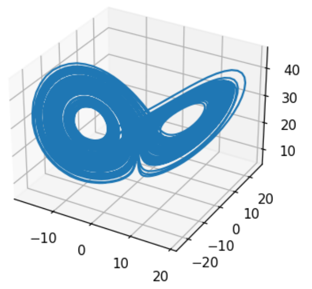
A network is said to have the echo state property if $\forall$ $x_i(t)$ $\exists$ e_i (echo function) such that if the network has been run for an indefinitely long time in the past then the current state can be written as $x_i(t) = e_i(u(t), u(t-1), u(t-2), \dots)$. A RNN which is driven by an external signal $u(t)$ has the echo state property if the activations $x(t)$ of the RNN neurons are systematic variations of the driver signal. ESP is satisfied in most practical cases when the maximum eigenvalue of $\mathbf{W}$ is less than 1. In a network with echo states the effect of initial conditions vanish with time ie effects due to choice of initial reservoir state die out after enough iterations.

1.6 Initialization

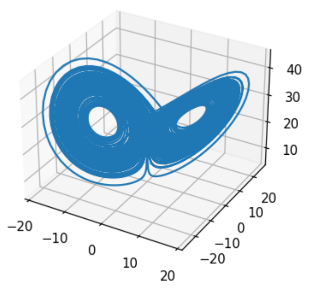
The weight matrices $\mathbf{W}^{\text{in}}$, $\mathbf{W}^{\text{back}}$ and $\mathbf{W}$ are initialised randomly and remain fixed throughout. The weights of $\mathbf{W}$ are usually picked from a $\mathcal{U}(-0.5, 0.5)$ distribution. We would want that $\mathbf{W}$ has a maximum eigenvalue (spectral radius) $\lambda_{\text{max}} \leq 1$ to ensure that the network has echo state property. We would also like $\mathbf{W}$ to be sparse (most weights set to 0) as it improves on computation time.The weights for $\mathbf{W}^{in}$ are picked from $\mathcal{U}(-a, a)$ distribution where a is the input scaling parameter. The leak rate α should be set to match the speed of dynamics of $y(t)$. Finally the spectral radius is set depending on the how much memory capacity we would like for the reservoir. A higher spectral radius would be preferred for slower dynamics.

2 Measures for Long-term Dynamics

Our goal is to train the ESN to be able to replicate the dynamics of Lorenz 63[2] or any chaotic system. We'd like to see how well the ESN predicts on its own after training it with hyperparameters(LR=0.3, SR=0.6, N=2000, inputscaling=0.5). The plots below show the predicted and actual trajectories:



(a) $ESNprediction$



(b) $TestCase$

We notice that the two plots are similar, but we'd like to be able to use some objective method to confirm that the ESN does, in fact, replicate the long-term dynamics of the Lorenz 63. To compare the plots generated by the ESN with those generated by the actual data, we need some measures that can characterise the system's overall long-term dynamics[3]. Some of these potential measures are as follows:

- Maximal Lyapunov Exponent
- Correlation Dimension
- Sample Entropy

3 Time Delay Embedding

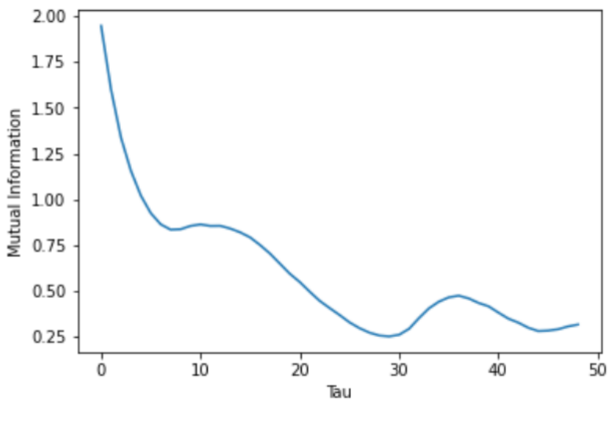
Before moving on to the statistical properties, we should mention attractor reconstruction using time delay embedding. The Maximal Lyapunov exponent, Correlation Dimension, and sample entropy are computed using time delayed embedding of one of the variables. Given an M-dimensional system, we do not always have observed values for all D variables, and using Takens' Embedding Theorem we can reconstruct the system's dynamics using the time delayed version of $x(t)$ ie $x(t), x(t-\tau), x(t-2\tau), \ldots, x(t-(D-1)\tau)$ where τ, D are to be chosen appropriately. This type of reconstruction preserves all properties that remain invariant under diffeomorphisms.

4 Picking appropriate values for τ using Mutual Information

The mutual information[4] between measurement a_i drawn from a set $A = \{a_i\}$ and measurement b_j drawn from a set $B = \{b_j\}$ is the amount learned by the measurement of a_i about the measurement of b_j In bits it is $\log_2 \frac{P_{AB}(a_i,b_j)}{P_A(a_i)P_B(b_j)}$, where $P_{AB}(a,b)$ is the joint probability density for measurements A and B resulting in values a and b. $P_A(a)$ and $P_B(b)$ are the individual probability densities for the measurements of A and of B. The average over all measurements of this information statistic, called the average mutual information between A measurements and B measurements, is:

$$I_{AB} = \sum_{a_i,b_j} P_{AB}(a_i,b_j) \log_2 \frac{P_{AB}(a_i,b_j)}{P_A(a_i)P_B(b_j)} \tag{5}$$

Since we are concerned with picking the ideal τ such that the mutual information carried by the vectors are minimal, we find the first minima of the I_{AB} vs τ plot, where $A = \{x(n)\}$ and $B = \{x(n+\tau)\}$.



Note: The first 0 of the auto-correlation function is not preferred since the value of τ came out to be a large number (250). For example, for the Lorenz 63 $x(t)$ series we have a minima at $\tau = 9$. This suggests we use $\tau_{ideal} = 9$ to obtain our delay embedding.

This allows to calculate topological properties of the original structure without complete knowledge of the original system. For example the delay embeding for the ESN predicted Lorenz 63 using x(t),y(t),z(t) are given below:

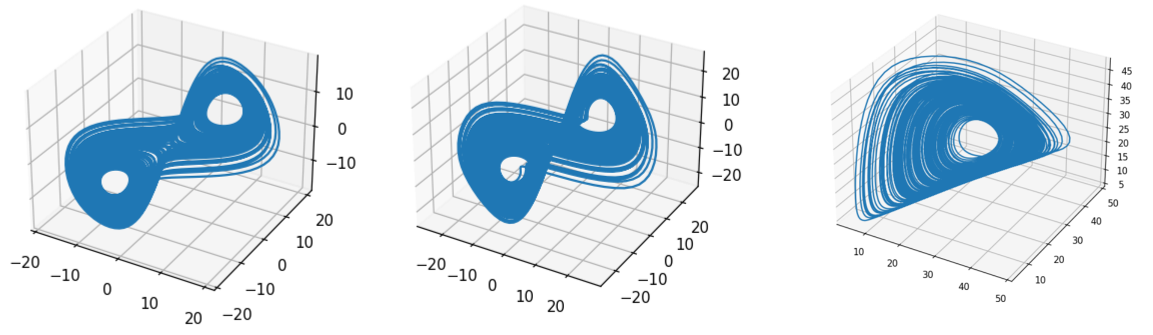


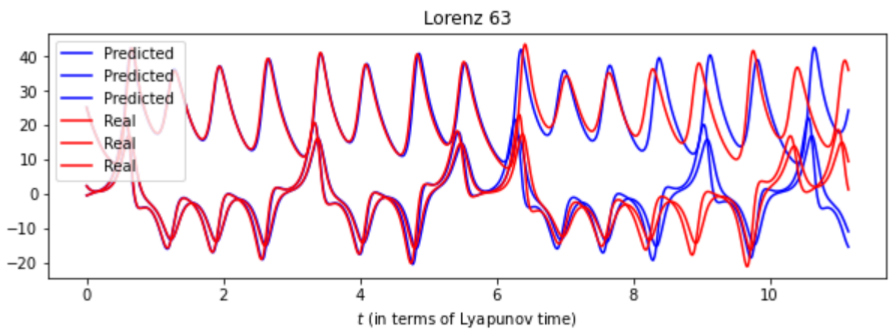
Figure 2: Delayed x(t), y(t), z(t)

5 Maximal Lyapunov Exponent

The Lyapunov exponent for a dynamical system describes the rate of separation of two infinitesimally close trajectories. Let $\vec{\delta z}$ denote the separation vector between two trajectories, the evolution of this vector is characterised by $\vec{\delta z}(t) = e^{\lambda t}\vec{\delta z}(0)$ where λ is the Lyapunov exponent. The rate of separation can vary depending on the orientation of the initial separation vector. As a result, there is a spectrum of Lyapunov exponents that is proportional to the dimensionality of the phase space. It is common to refer to the largest one as the maximal Lyapunov exponent (MLE). A positive MLE is usually taken as an indication that the system is chaotic. It should be noted that an arbitrary initial separation vector will typically contain some component in the direction associated with the MLE, and the effect of the other exponents will be obliterated over time due to the exponential growth rate.

5.1 Calculation of MLE

The theoretical value of $\lambda_{\max}$ is 0.9056 for the Lorenz 63($\sigma = 10, \rho = 28, \beta = 8/3$). Using the above algorithm we get $\lambda_{\max}^{\text{test}} = 0.88002$ and $\lambda_{\max}^{\text{pred}} = 0.98648$. Using this value we get the Lyapunov time as $1/\lambda_{\max} = 1.104 \approx 110$ steps, note that $\Delta t = 0.01$ is one timestep. The ESN is able to accurately predict upto 6 Lyapunov Times and stays close to the test trajectory until 8 Lyapunov Times before diverging.



6 Correlation dimension

The correlation dimension[5][6], also known as a type of fractal dimension in chaos theory, is a measure of the dimensionality of the space occupied by a set of random points. Correlation dimension has the advantage of being simple to calculate, less noisy when only a few points are available, and frequently agreeing with other dimension calculations.The correlation integral $C(r)$ is given by:

$$C(r) = \lim_{N \to \infty} \frac{g}{N^2} \tag{6}$$

where $g = \#(x, y)$ such that $d(x, y) < r$. As r is made small, we see that $C(r)$ depends on r as $C(r) \sim r^d$. This d is the correlation dimension. If the number of points is sufficiently large, and evenly distributed, a log-log graph of $C(r)$ versus r will yield an estimate of d.

6.1 Calculation of Correlation Dimension

The theoretical value for Correlation Dimension of the Lorenz 63 is 2.06. We have calculated the Correlation Dimension for Test data and the ESN predictions using the different values of embedding dimension. The error in prediction and test is less than 5% in all cases.

Emb Dim	Pred	Test
4	2.015	2.068
5	2.072	2.122
6	1.99	2.044
7	1.986	2.043
8	1.946	2.011
9	1.937	2.002

7 Sample Entropy

The sample entropy[7] of a time series is defined as the negative natural logarithm of the conditional probability that two sequences similar for N points remain similar at the next point, excluding self-matches. A lower sample entropy value thus corresponds to a higher probability and serves as a measure of complexity.

Let us say we have a series $X = \{x_0, x_1, \ldots, x_n\}$ where x_i corresponds to the value of the series X at time $t_0 + i\Delta t$ where t_0 is the value of X at x_0. Consider a template vector of size m $X_m(i) = \{x_i, x_{i+1}, \ldots, x_{i+m-1}\}$ and let $d(X_m(i), X_m(j))$ be row-wise Chebyshev distance function. $d_{chebyshev}(x, y) := \underset{i}{\text{argmax}} \|x_i - y_i\|$

We define the sample entropy as:

$$\text{SampEn} = -\log \frac{\#\text{Pairs } (X_m(i), X_m(j)) \text{ such that } d(X_{m+1}(i), X_{m+1}(j) < r}{\#\text{Pairs } (X_m(i), X_m(j)) \text{ such that } d(X_m(i), X_m(j) < r} \tag{7}$$

Thus, SampEn is always greater than 0. Entropy is typically used to measure the randomness of a series. More entropy is associated with more uncertainty.

7.1 Calculation of Sample Entropy

The ESN predictions and test data had a similar sample entropy of 0.09285 and 0.09408 respectively.

8 Lorenz 96 predictions

In order to quantify the prediction capability of the ESN we define the prediction horizon (P_r) for a given tolerance r as the minimum $t > 0$ (where $t = 0$ is the first prediction made by the ESN) such that the mean squared error defined as mse $= \frac{1}{n} \sum_{i=1}^{n} (y_{\text{test}} - y_{\text{pred}})^2$ exceeds r.

N=5,F=8

These parameters are known to give chaotic solutions and the MLE of the test series is $\lambda_{max} = 0.93$. This means the Lyapunov time is $\frac{1}{\lambda_{max}} = 1.07(107\Delta t)$. The reason for selecting these parameters is that the MLE is very close to that of the Lorenz 63. As a result, it is a simple method to compare how the network performs for higher dimensional predictions without introducing increased chaotic behaviour.

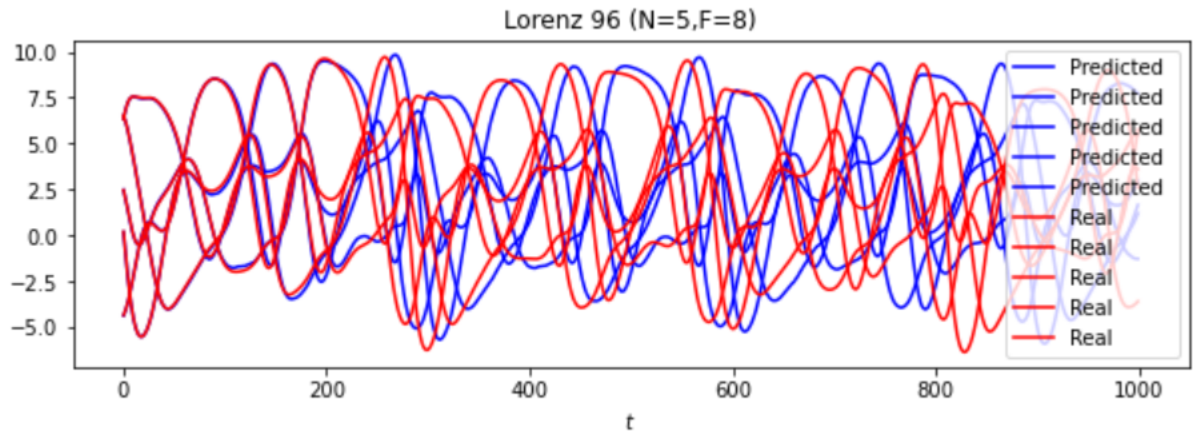


Figure 3: Predictions of the ESN (2000 units) ($\Delta t = 0.01$)

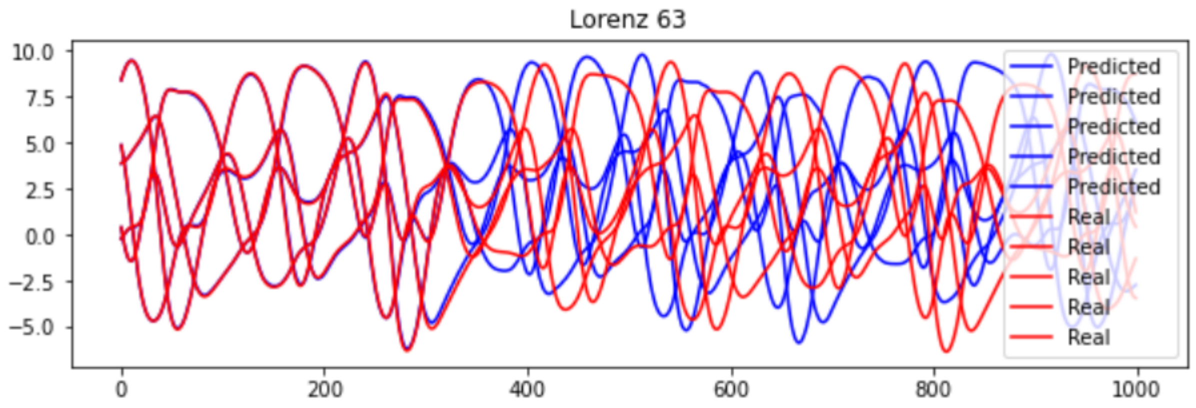


Figure 4: Predictions of the ESN (3000 units) ($\Delta t = 0.01$)

Find below the prediction horizon of the network with different values of tolerance and reservoir size. All the other hyperparameters have been kept the same.

	Tolerance(r)			
Size of Reservoir	0.1	0.3	0.6	1.0
500	100	120	130	190
1000	130	160	230	260
2000	230	250	270	280
3000	380	390	410	470

Table 1: Variation of Prediction Horizon with Tolerance and size of the reservoir

Below is the evolution of errors of 4 reservoirs with different sizes keeping other parameters same.

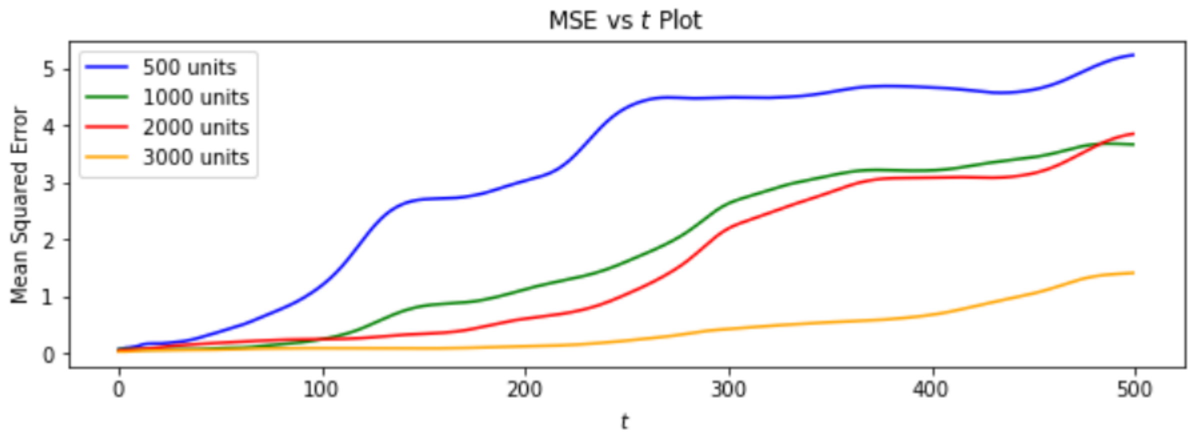


Figure 5: Evolution of MSE over time

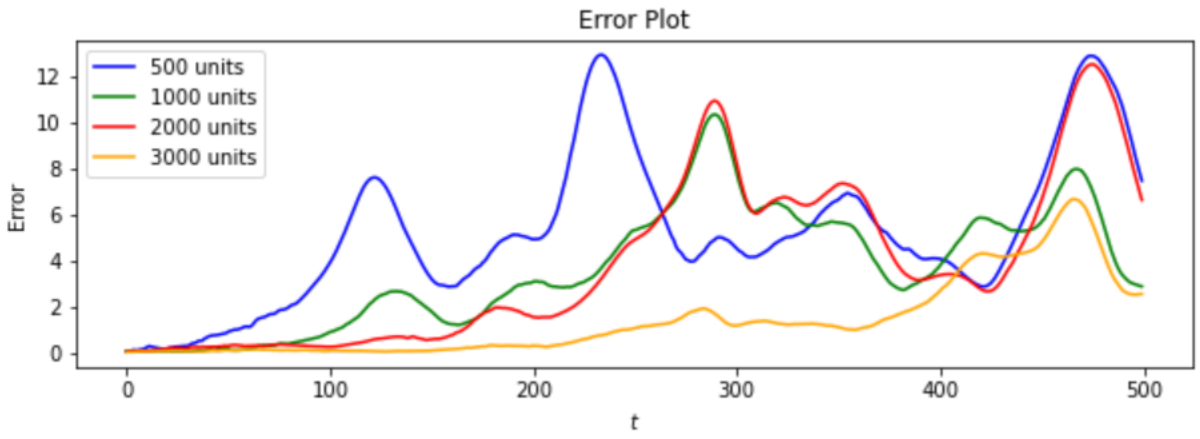


Figure 6: Distance between predicted and test trajectories over time

From the plots it seems to appear that the error follows an exponential growth. To illustrate this, we take the log of both plots and use linear regression to obtain a linear fit for MSE and distance over time. The slope and R^2 of the best linear fit for various reservoir sizes are shown in the table below. For interpolation only the error before time $t < t_0$ is considered where t_0 is the prediction horizon for tolerance $r = 0.1$. The plots below suggests that the error grows the slowest for larger reservoir and the error growth is exponential.

MSE

Size of Reservoir	Slope of line	R^2
500	0.025	0.99
1000	0.017	0.92
2000	0.009	0.94
3000	0.007	0.94

Distance

Size of Reservoir	Slope of line	R^2
500	0.0362	0.97
1000	0.0315	0.95
2000	0.0133	0.92
3000	0.0104	0.87

N=5, F=10

We change the forcing term because we want to study how the network responds to more chaotic situations without having to cope with more dimensions. Chaotic solutions are likewise provided by N=5 and F=10, and the MLE of the test series is $\lambda_{max} = 2.58$. This means the Lyapunov time is $\frac{1}{\lambda_{max}} = 0.387(38.7\Delta t)$. The ESN is trained using a sequence of 300000 time steps and 3000 units. The prediction horizon $P_{0.6}$ is 260 time steps $\approx$ 6.71 lyapunov times.

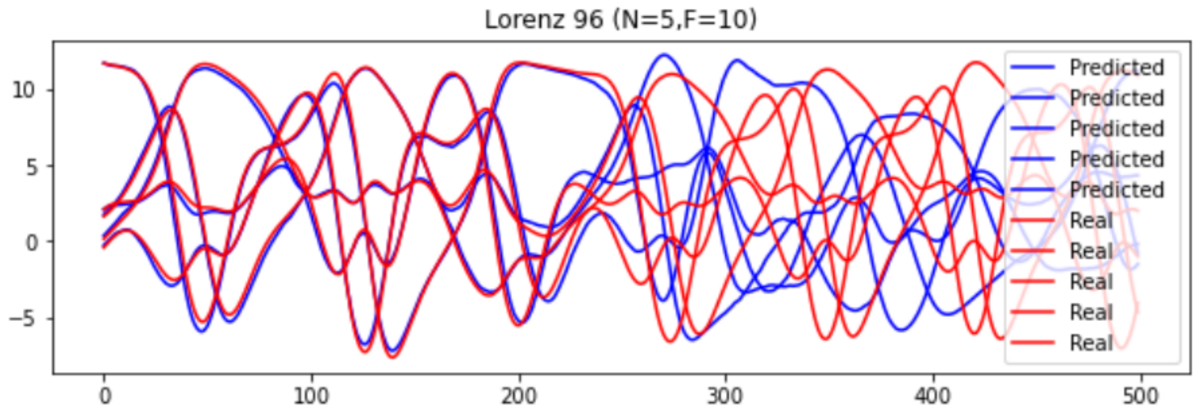


Figure 7: ESN predictions with 3000 units($\Delta t = 0.01$)

8.1 Dependance on length of training data

Now we would also like to train the ESN and see how it performs as a function of the length of the training sequence. The data for the prediction horizon of a 1000 unit ESN trained on several Lorenz 96(N=5,F=8) training sequences are shown in the following table. The percentage of data used to train the network is referred to as the "train length" in this case.

Train length	$P_{0.1}$	$P_{0.3}$	$P_{0.6}$	P_1
0.1	100	140	180	250
0.2	300	340	390	410
0.25	210	250	270	280
0.3	120	270	280	350
0.35	60	280	310	340
0.4	130	270	290	350
0.5	40	150	180	240
0.6	160	360	410	450
0.7	140	210	290	310
0.8	110	200	240	250
0.9	120	190	220	250

The table above shows that smaller train sequences have better prediction horizons than bigger train sequences. This shows that using smaller datasets for training the ESN would be preferable because larger training data doesn't seem to provide any improvement in performance.

Using just 0.3 of the total data $\approx$ 300000 data points and 5000 units we got a prediction horizon P_1 of 10.65 lyapunov times (1140 timesteps).

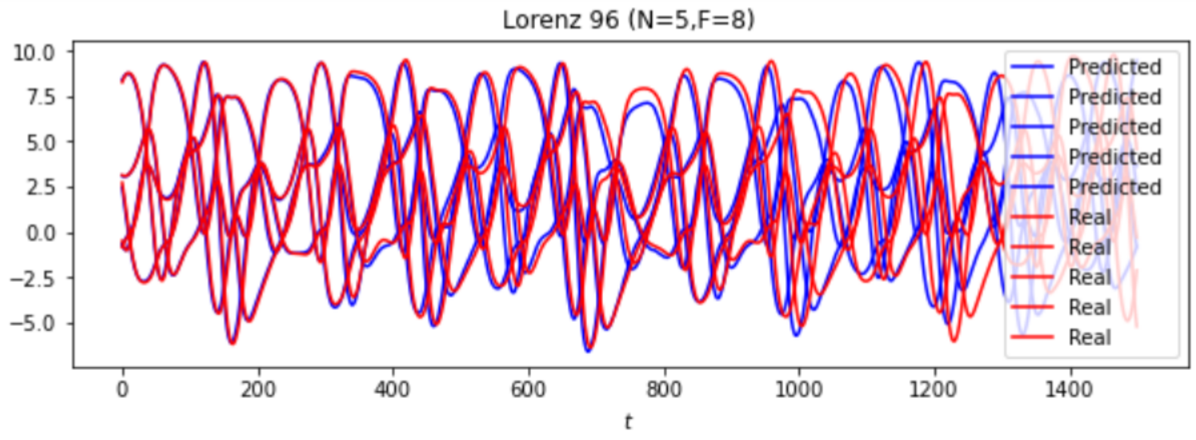


Figure 8: ESN predictions with 5000 units($\Delta t = 0.01$)

8.2 Effect of Noise in Data

We also want to evaluate how well the ESN performs after being trained on noisy data. This helps us understand if the ESN can filter through the noise and pick up the useful information from the training data. In order to evaluate the performance of the network, we add a random noise term to the training set of Lorenz 96 ($N = 5, F = 8$) and use the prediction horizon measure. We have added a normal distribution noise term with mean μ and standard deviation σ^2. The reservoir parameters are kept same (2000 units), training on 100000 time steps.

μ	σ^2	$P_{0.1}$	$P_{0.3}$	$P_{0.6}$	P_1
0	0	460	510	630	860
0	0.1	300	460	500	510
0	0.25	270	360	500	550
0	0.5	140	280	320	330
0	0.75	210	260	440	500
0	1	180	380	460	500
0	10	-	-	-	-

9 Time Complexity

Let us try to compute the time complexity of training the ESN. The training process involves 2 steps - Updating the reservoir state after each input and ridge regression to compute W^{out}. We need to compute the time complexity both of the steps separately.

Before that we need to define some terms. $x(t), u(t), y(t)$ represents the state vector(n dimensional) of the reservoir, (h dimensional) input vector and target output vector (b dimensional) at time t respectively. $\mathbf{W}^{\text{in}}$ is the $n \times h$ input matrix, $\mathbf{W}$ is the $n \times n$ reservoir matrix $\mathbf{W}^{\text{out}}$ is the $b \times (h+n)$ output matrix, X is the $(h+n) \times T$ matrix obtained by concatenating all the input and reservoir states $(u(t), x(t))$.

9.1 Updating Reservoir State

The reservoir state is updated at every step according to:

$$x(t+1) = (1-\alpha)x(t) + \alpha f(\mathbf{W}^{\text{in}}u(t) + \mathbf{W}x(t)) \tag{8}$$

where $0 \leq \alpha \leq 1$ is the leaking rate and f is the activation function(in this case ReLU).

- $\mathbf{W}^{\text{in}}u(t) + \mathbf{W}x(t)$
  Multiplying the matrices and adding the vectors takes time $\mathcal{O}(n(2h-1)+n(2n-1)+n)$ ($\mathbf{W}^{\text{in}}u(t)$ term and $\mathbf{W}x(t)$ term and adding the matrices respectively)
- Applying ReLU to the vector from previous step
  The ReLU function acts on each element of $W = \mathbf{W}^{\text{in}}u(t) + \mathbf{W}x(t)$ and thus takes time $\mathcal{O}(n)$
- Multiplying $f(\mathbf{W}^{\text{in}}u(t) + \mathbf{W}x(t))$ by α, multiplying x(t) by $(1-\alpha)$ and adding the resulting vector takes time $\mathcal{O}(n)$.

This updation takes place for every input. Thus finally time $T_1 = \mathcal{O}(T(n+n+(n(2n-1+2h-1))+n)) = \mathcal{O}(Tn(n+h))$

9.2 Computing W^{out}

W^{out} is computed such that $\frac{1}{b}\sum_{i=1}^{b}(\sum_{t=1}^{T}\|y_i^{\text{pred}}(t) - y_i^{\text{target}}(t)\|_2^2 + \beta\|\mathbf{w}_i^{\text{out}}\|_2^2)$ is minimised over all training data. The value of W^{out} that minimises this error function is given by:

$$W^{\text{out}} = y^{\text{target}}X^T(XX^T + \beta I)^{-1} \tag{9}$$

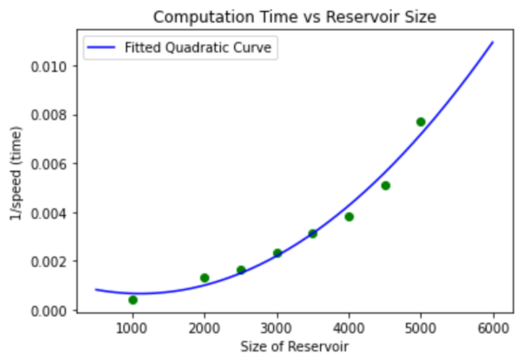
- $XX^T + \beta I$ takes time $= \mathcal{O}((h+n)(2T-1)(h+n) + (h+n)^2)$
- Calculating Inverse of the previous step takes time $\mathcal{O}((h+n)^3)$
- Computing $y^{\text{target}}X^T$ takes time $\mathcal{O}(b(2T-1)(h+n))$
- Multiplying $y^{\text{target}}X^T$ and $(XX^T + \beta I)^{-1}$ takes time $\mathcal{O}(b(2(h+n)-1)(h+n))$

Finally time $T_2 = \mathcal{O}(T(h+n)^2 + (h+n)^3 + b(2T-1)(h+n) + b(2(h+n)-1)(h+n)) = \mathcal{O}((T(h+n)^2)$

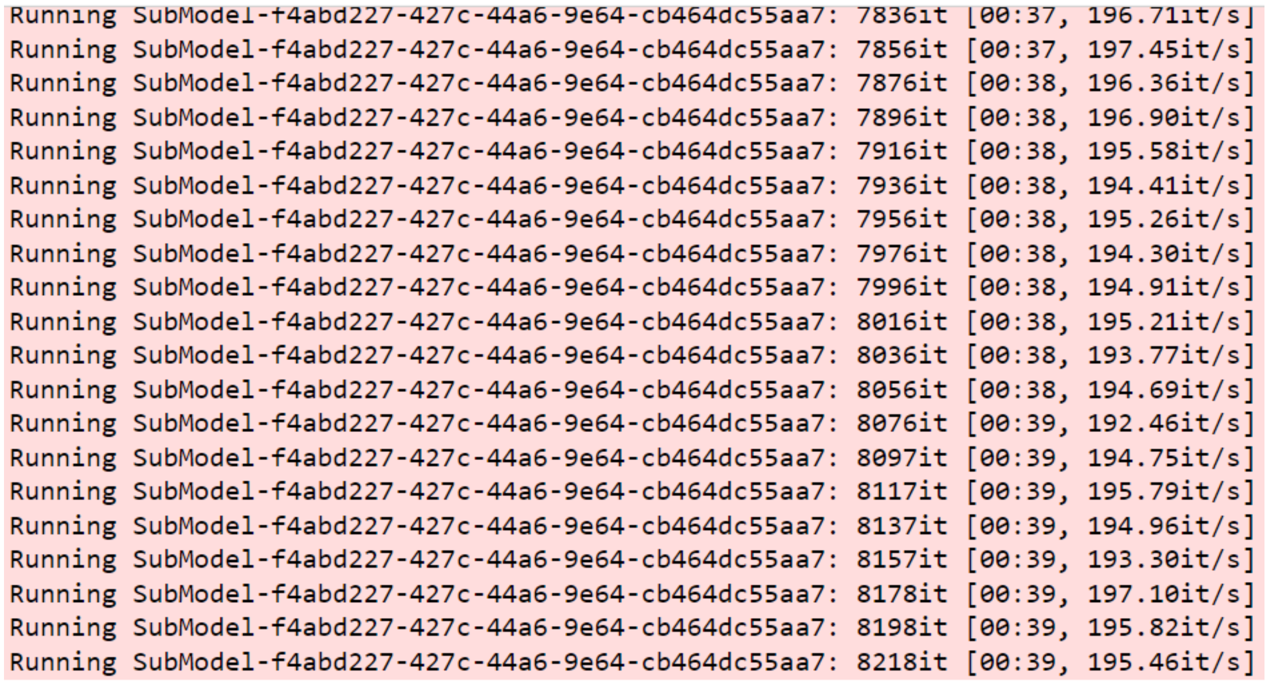
Total time complexity(training) $= T_1 + T_2 = \mathcal{O}(Tn^2)$. In other words the computation time for training the ESN increases linearly with length of the training data and increases quadratically with size of the reservoir. Note: We have used $u, b << n << T$ to get the final time complexity.

We plot the time taken to update the state of the reservoir as a function of the size of the reservoir. We have the average 'speed' (number of updates per second) whose inverse gives us a measure of the time elapsed per iteration. According to our theoretical value we should get a quadratic fit ie the time should increase quadratically with size of reservoir and that is exactly what we notice below.

```
{'r_squared': 0.9776245865723951}
           2
4.299e-10 x  - 9.552e-07 x + 0.001198
```



Next we also verify that the computation time grows linearly with size of the training dataset T. The 'speed' of the state updation in iterations/s is more or less constant which verifies that the computation time grows linearly with T.



```
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7836it [00:37, 196.71it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7856it [00:37, 197.45it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7876it [00:38, 196.36it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7896it [00:38, 196.90it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7916it [00:38, 195.58it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7936it [00:38, 194.41it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7956it [00:38, 195.26it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7976it [00:38, 194.30it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 7996it [00:38, 194.91it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8016it [00:38, 195.21it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8036it [00:38, 193.77it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8056it [00:38, 194.69it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8076it [00:39, 192.46it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8097it [00:39, 194.75it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8117it [00:39, 195.79it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8137it [00:39, 194.96it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8157it [00:39, 193.30it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8178it [00:39, 197.10it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8198it [00:39, 195.82it/s]
Running SubModel-f4abd227-427c-44a6-9e64-cb464dc55aa7: 8218it [00:39, 195.46it/s]
```

Figure 9: Runtime information of the ESN

To summarise, the computation time for training:

- Grows quadratically (n^2) with reservoir size n
- Grows linearly with length of the training data T
- Independent of length of input and output vectors u, b respectively.

A Algorithm to compute Largest Lyapunov exponent

- First, the dynamics of the data are reconstructed using a delay embedding method with a lag, such that each value x_i of the data is mapped to the vector $X_i = (x_i, x_{i+\tau}, x_{i+2\tau}, \ldots, x_{i+(m-1)\tau})$.
- For each such vector X_i, we find the closest neighbor X_j using the euclidean distance. We know that as we follow the trajectories from X_i and X_j in time in a chaotic system the distances between X_{i+t} and X_{j+t} denoted as $d_i(t)$ will increase according to a power law $d_i(t) = ce^{\lambda t}$ where λ is a good approximation of the highest Lyapunov exponent, because the exponential expansion along the axis associated with this exponent will quickly dominate the expansion or contraction along other axes.
- Taking log on both sides we get, $\log(d_i(t)) = \log(c) + \lambda t$. This gives a set of lines (one for each index i) whose slope is an approximation of lambda. We therefore extract the mean log trajectory $d'(k)$ by taking the mean of $\log(d_i(k))$ over all orbit vectors X_i. We then fit a straight line to the plot of $d'(k)$ versus k. The slope of the line gives the desired parameter λ[8].

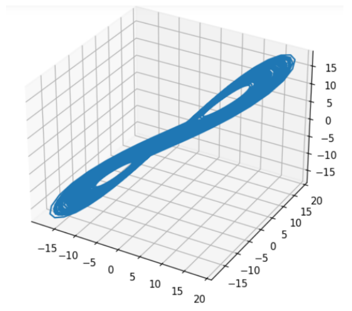
B Sample Entropy as a measure of randomness

We tried calculating SampEn ($m = 4$) for a series $f(x) = x + p\epsilon, x \in 1, 2, \ldots, 10000$ where p is a scaling parameter taken from the set $\{0, 100, 10000\}$. ϵ is the noise term and is taken from a Normal distribution ($\mu = 0, \sigma = 100$).

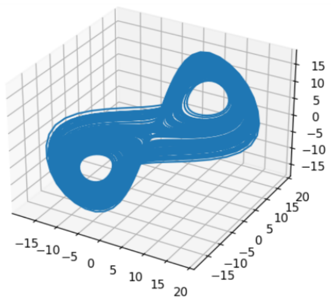
- p=0 $\implies$ the series is noise free and sampEn takes a value 0.
- p=100 $\implies$ there is 'intermediate' noise and sampEn ≈ 0.056 takes a non zero but small value.
- p=10000 $\implies$ there is mostly noise in the series and sampEn ≈ 2.209 takes a high value.

C Time Delay Embedding with τ

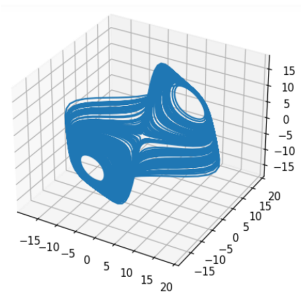
Find below the the reconstructed attractor using different values of τ.



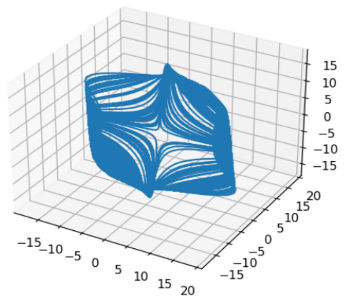
(a) $\tau = 1$



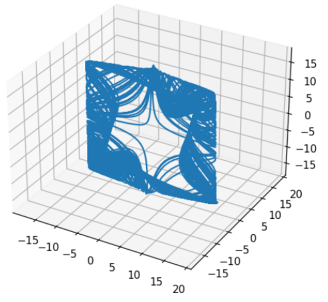
(b) $\tau = 5$



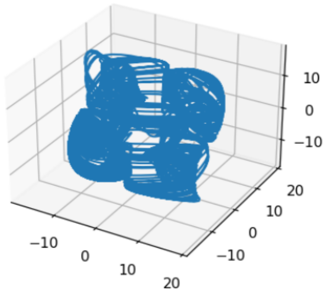
(c) $\tau = 9$



(d) $\tau = 12$



(e) $\tau = 16$



(f) $\tau = 28$

D Code Availability

ReservoirPy[9] was used for creating the Network. Nolds[10] was used to compute some of the statistical properties of the time series. The code for this project will be available here eventually.

References

[1] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

[2] Jaideep Pathak, Zhixin Lu, Brian R Hunt, Michelle Girvan, and Edward Ott. Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(12):121102, 2017.

[3] Zonghua Liu. Chaotic time series analysis. *Mathematical Problems in Engineering*, 2010, 2010.

[4] Henry Abarbanel. *Analysis of observed chaotic data*. Springer Science & Business Media, 2012.

[5] Peter Grassberger and Itamar Procaccia. Characterization of strange attractors. *Physical review letters*, 50(5):346, 1983.

[6] Peter Grassberger. Grassberger-procaccia algorithm. *Scholarpedia*, 2(5):3043, 2007.

[7] Joshua S Richman and J Randall Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*, 278(6):H2039–H2049, 2000.

[8] Michael T Rosenstein, James J Collins, and Carlo J De Luca. A practical method for calculating largest lyapunov exponents from small data sets. *Physica D: Nonlinear Phenomena*, 65(1-2):117–134, 1993.

[9] Nathan Trouvain, Luca Pedrelli, Thanh Trung Dinh, and Xavier Hinaut. ReservoirPy: An efficient and user-friendly library to design echo state networks. In *Artificial Neural Networks and Machine Learning – ICANN 2020*, pages 494–505. Springer International Publishing, 2020.

[10] Christopher Schölzel. Nonlinear measures for dynamical systems, June 2019.