Echo State Networks and their Applications to Chaotic Systems

Presented to the S. Daniel Abraham Honors Program
in Partial Fulfillment of the
Requirements for Completion of the Program

Stern College for Women
Yeshiva University
May 7, 2021

Nava Rosenblatt

Mentor: Marian Gidea, PhD.
Chair, Department of Mathematics

Abstract

Within the field of machine learning, Echo State Networks (ESN) are a type of neural networks, in which input signals are mapped into higher dimensional spaces, which are connected to outputs. ESNs are used in particular for sequential data, with applications to forecasting. The benefit of using an Echo State Network to predict data is that it is model-free, meaning that there does not need to be any prior knowledge about the data. This is in contrast to using a model-based approach, which requires understanding of the system. This paper will examine the application of ESNs to chaotic systems.

Introduction

Artificial intelligence

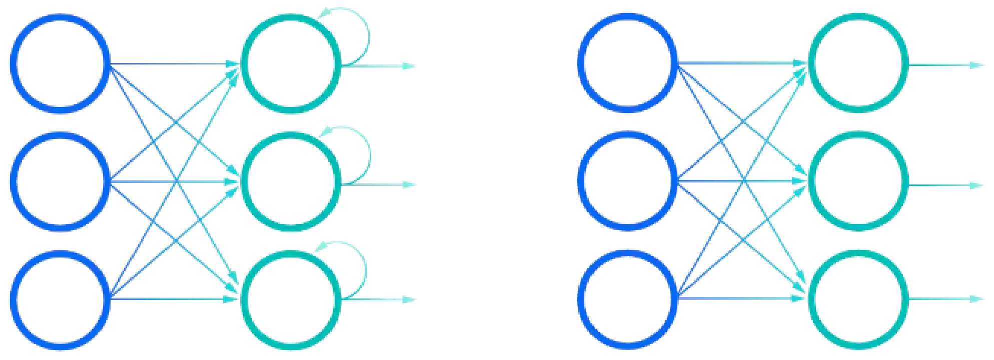
Artificial intelligence is the ability of a computer to perform tasks associated with intelligent beings. It refers to the task of developing systems which can carry out the same intellectual processes that humans do, such as the ability to reason, find meaning, generalize, or learn from the past [1]. John McCarthy, who is known as the father of artificial intelligence, defined it as "the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable. [2]"

Machine learning

Machine learning is an application of artificial intelligence. It gives a system the ability to learn and improve from experience without being programmed to do so. It involves feeding a computer program data in order for it to learn from the data itself, make sense of it, and find patterns in it. Machine learning uses statistics to find patterns in data [3]. There are different types of machine learning. In supervised machine learning, the training data that the computer learns from is labeled. This tells the machine exactly what patterns to look for. In unsupervised learning, the machine is fed unlabeled data, and its job is to find patterns [4]. Typically, machine learning works well on data that exhibits patterns, and the machine is able to detect these patterns. A more difficult task is using machine learning on data that is seemingly random and does not have any pattern. Chaotic systems are an example of this, and will be discussed at length later in this paper.

Neural networks

Neural networks are part of machine learning. They usually work on examples that have been manually labeled in advance. For example, an object recognition system could be fed labeled images of different objects, and the machine would find patterns in these images so that it could label new images with their correct object. Neural networks are modeled after the human brain, and contain neurons that are connected. Most neural networks today are “feed forward”, meaning that the data moves through them in one direction, from one layer to the next ones. Neural networks have multiple layers. The first layer is the input layer, in which the training data is fed. The first layer is connected to the second layer, also known as the hidden layer, which contains the neurons. Each neuron is connected to the others, and each connection has an associated weight. When the network is active, a neuron will receive data from each of its connections. This data is multiplied by its associated weight, which are all added together into a single number. If the number is below a certain threshold, nothing happens, but if it is above the threshold, the neuron will “fire”, sending a signal to all of its outgoing connections. During training of a neural network, the weights and thresholds are initially set to random values. The weights and thresholds are adjusted during the training period until they yield good results. There are algorithms that modify these weights and thresholds. As the data passes through the layers of the network, it is transformed, until it reaches the output layer, which is the third and final layer [5].


[8]

L: Recurrent neural network, R: feedforward neural network

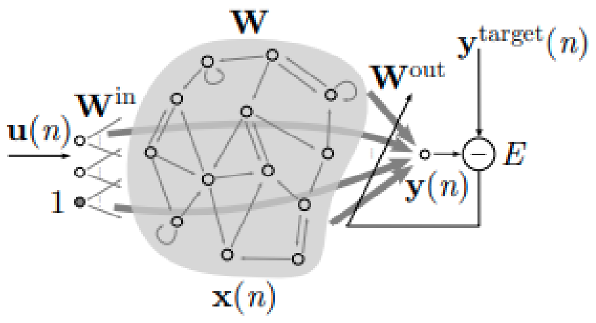
Recurrent neural networks

Unlike feed forward neural networks, where the data only travels forward, recurrent neural networks (RNNs) can have loops, where neurons fire, stimulating other neurons, and it is possible that some neurons which were previously stimulated will become stimulated again [6]. In other words, that previous output becomes input. The current state of the network at time n is a function of the weight matrix, new input data, and previous state of the network. This gives the network an aspect of “memory”, that previous information and inputs can affect current outputs. This means that the same input could give a different output, depending on previous input [7]. Unlike traditional neural networks, which assume that input and output are independent of each other, recurrent networks depend on previous information [8]. RNNs make use of sequential information, such as time series. Applications of RNNs include forecasting a time series, speech recognition, and automatic image capturing. The “memory” aspect of the RNN allows it to accomplish these tasks.

Echo State Networks

An application of RNNs is Echo State Networks (ESNs). ESNs avoid some of the difficulties that come with training recurrent neural networks. Because RNNs are difficult to train, many avoid them. RNNs are very powerful because they combine large dynamical memory with adaptable computational capabilities. With ESNs, the RNN (reservoir) is generated randomly and only the readout is trained. This is helpful because it reduces how much the model needs to be trained [12].

Training the ESN involves inputting the training data u(n) into the network. For each value of u(n), a corresponding vector x(n) of neural activation is generated. x(n) is dependent on the input weight matrix W^{in}, the neural connection matrix W, and x(n-1), the previous value of x(n). Once all the training data is fed, a matrix X of all the neural activation vectors x(n) is created. This matrix is used to find the output weight matrix, which is used to generate the y predicted values.



Method

Time series

The ESN is applied to machine learning tasks where the training input signal is a time series u(n), where n is time. n takes on the values 1, 2… T and is a discrete time, where T is the number of data points in the training set. u(n) is either a scalar or a vector in some dimension R^{N_u}. For one dimensional data, u(n) will be a scalar, and N_u=1. This represents a single variable with respect to time. An example of this is the price of a stock with respect to time. For multidimensional data, u(n) will be a vector, and N_u>1. This paper will focus on one dimensional data, with the goal of using ESNs to forecast sequential data.

Input weights

The input weight matrix W^{in} is generated as a random matrix, and remains fixed throughout the process. W^{in} has dimension N_x x (1+N_u), where N_x is the size of the network (the number of neurons), and N_u is the dimension of the input data. The entries of$W^{in}$are typically distributed in either in a symmetric uniform, discrete bi-valued, or normal distribution around zero. In the Mackey Glass example discussed later, W^{in} is uniformly distributed between -.5 and .5.

Hidden layer (the network of neurons/reservoir)

The matrix W is an N_x by N_x matrix, and stores the connections of the neurons. Each entry in the matrix holds a number which represents a connection between two neurons. W should be sparse, meaning that each neuron is connected to a small fixed number of other neurons, regardless of the size of the reservoir. Practically, this means that most of W's entries should be close to zero. The further the entry value is from zero, the more connected the neurons.

A sparse network was found to give much better results than a more dense network. If the number of connections is fixed, then as the network size increases, the computational cost will grow linearly instead of quadratically. This means that as the network increases in size, the time it takes to run it will only increase proportionally. W's elements are typically distributed the same as W^{in} (either symmetrical uniform, discrete bi-valued, or normal distribution around zero), but W is sparse while W^{in} is dense [12].

The spectral radius $\rho(W)$ of the reservoir connection matrix W is a central parameter of the ESN. It is used to scale W to have a spectral radius less than one, which is necessary to maintain the echo state property. The echo state property must be satisfied in order for the reservoir to work properly. It requires that the state of the reservoir x(n) should be "uniquely defined by the fading history of the input u(n)." This means that the state of the reservoir x(n) shouldn't depend too much on the conditions from before the input. W is scaled by dividing it by its spectral radius, which gives a new matrix with a spectral radius of one [12]. In practice, sometimes W is scaled to have a spectral radius slightly more than one, such as in the Mackey-Glass example, where $\rho(W_{scaled})= 1.25$.

Spectral radius and eigenvalues

The spectral radius of a matrix is the maximal absolute eigenvalue of the matrix. Before understanding what the spectral radius is, it is first necessary to recall eigenvalues. The eigenvalue of a matrix is defined as follows: Given a matrix A, write the equation **Av**= λ**v,** where A is an n by n matrix, **v** is a non-zero n x 1 vector, and λ is a scalar. Any value of λwhich is a solution to the equation is known as an eigenvalue of the matrix **A**. The vector **v** is known as the eigenvector. To solve for λ and **v**, the equation can be written as

Av-λ**v**=0. Multiplying the second term by the identity matrix I gives

Av-λI**v**=0. Factoring out the vector v gives

(A-λI)**v**=0. If v is nonzero, then **(A**-λI) must be a singular matrix.

(**A**-λI)=0 will have a non-zero solution if the determinant of **A**-λI is zero (|**A**-λI|=0).

This is called the characteristic equation of A, and it is an nth order polynomial with n roots, which are the eigenvalues of A. For each eigenvalue, there is a corresponding eigenvector which makes the equation true [13]. The spectral radius (ρ(A)) of a matrix A is defined as

ρ(A) = max{|λ|, λ eigenvalue of A}. This means that the spectral radius of a matrix is the largest absolute value of the eigenvalues of that matrix [14].

Scaling W

The spectral radius is used to scale the width of the distribution of the elements in W. First the connection matrix W is initialized, then its spectral radius is calculated, and then W is divided by its spectral radius. This gives a new matrix $W_{scaled} = \frac{1}{\rho(W)}$*W. This newly scaled matrix will have a spectral radius of one, $\rho(W_{scaled}) = 1$ [12].

Reservoir neuron activation

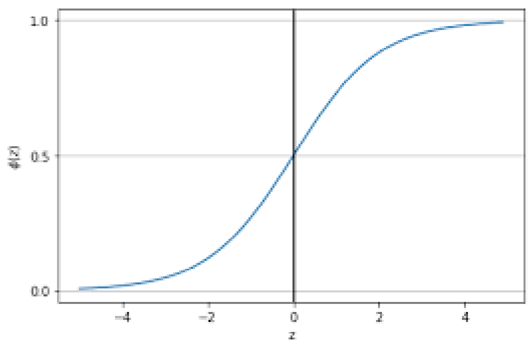
The reservoir activations of each neuron determines whether or not they will "fire" and activate their connected neurons. Because these neuron activation values should be within a specific interval, an activation function is applied to the vector which stores these activations, which ensures that the values will be within the prescribed range.

Activation functions

Different activation functions may be used, depending on the desired output.

Sigmoid function

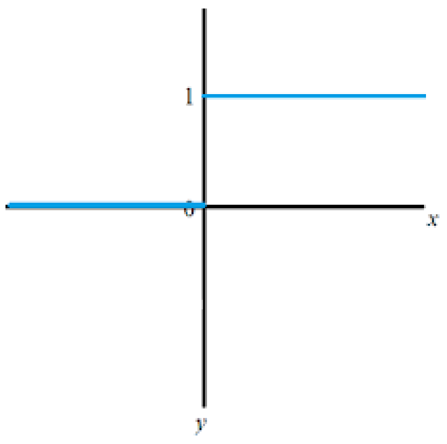
f(x) = $\frac{1}{1+e^{-x}}$



The sigmoid function is defined as f(x) = $\frac{1}{1+e^{-x}}$. Applying the sigmoid function to the activation matrix ensures that each entry is between zero and one. This function is often used in the context of probabilities, when the desired output is between zero and one.

Heaviside function

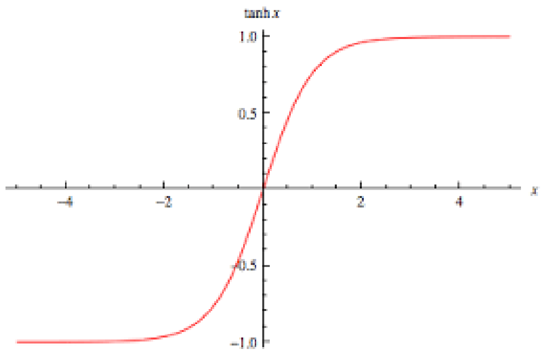
f(x) = {0 if x≤0, 1 if x>1



The heaviside function is defined as f(x) = {0 if x≤0, 1 if x>1. It is also known as a unit step function. For every input less than or equal to zero, the output will be zero. For every input greater than zero, the output will be one. A zero means that the neuron does not “fire,” and a one means that it does “fire.”

Hyperbolic tangent

tanh(x) = $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



The hyperbolic tangent function is defined as tanh(x) = $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. It generates values between -1 and 1.

Updating the reservoir

At each time n, the reservoir neuron activation vector x(n) gets updated. x(n) is a vector with N_x components, where each component corresponds to a neuron's activation at time n.

Reservoir update without leaking:

Without leaking, the x vector at time n is defined as

$x(n) = f(W^{in}[1; u(n)] + Wx(n-1))$

f is the activation function which is applied to each component of the vector that is being generated. In this paper, the hyperbolic tangent activation function is used.

W^{in} is the input weight matrix, with dimension N_x x (1+N_u), where N_x is the size of the network. (In case of one dimensional data when N_u=1, it is an N_x x 2 matrix). [1; u(n)] is an (N_u +1) x 1 column vector (in case of one dimensional data, when N_u=1, it is a 2 x 1 vector). The one appears by default in this vector and represents a constant term. Multiplying W^{in} with the vector [1; u(n)] gives an N_x x 1 vector. This left hand side of the sum represents the input data for the current u(n).

W, which stores the connections of the neurons, is an N_x x N_x matrix. x(n-1) describes the state of the neurons in the reservoir at time n-1. x(n-1) is an N_x x 1 vector. Multiplying W with x(n-1) gives an N_x x 1 vector. This right hand side of the sum represents the previous network state.

Reservoir update with leaking:

When the leaking term is included, the x vector at time n is defined as

$\bar{x}$(n) = tanh(W^{in}[1; u(n)] + Wx(n-1))

x(n)= (1-α)x(n-1) + α$\bar{x}$(n)

The second formula incorporates the leaking parameter, α, into the reservoir activation matrix. α is the speed at which the reservoir gets updated. Thus α can be thought of as the time interval between two consecutive time steps. Setting α is similar to resampling u(n) and $y^{target}(n)$ when the signals slow. α should be set to match the speed of u(n) and $y^{target}(n)$, but this can be difficult when the times of u(n) and $y^{target}(n)$ are very different. Therefore the parameter α is chosen based on trial and error [12]. Expanding $\bar{x}$ from the first formula into the second formula gives a single formula which incorporates the leaking rate:

x(n) = (1-α)x(n-1) + αtanh[W^{in}[1; u(n)] + Wx(n-1)]

X matrix

Once the network has been trained for n=1, 2… T, all the x(n)'s for each n will generate a matrix X $\in R^{(1+N_u+N_x)xT}$ = [1; U, X], which consists of all [1; u(n); x(n)] for n=1, 2… T. In other words, X is a matrix containing each of the vectors x(n) that the network produced for each u(n) that input into the network.

Output

During the training phase, the output values Y^{target}(n) are known. Because the readout from an ESN is usually linear, the equation y(n) = W^{out}[1; u(n); x(n)] can be written in matrix notation as Y^{target}= W^{out}X, where X is [1; u(n); x(n)]. W^{out} is an N_yx (1+$N_u + N_x$) matrix, Y is the desired output (either a scalar or a vector) and N_y is the dimension. The goal of the training phase is to estimate the entries of the W^{out} matrix.

Training the network

T is the size of the training sample. Y is a matrix of all the y(n) vectors: Y= [y(1), y(2),… y(T)]. Each y(n) vector is dimension N_y, thus Y is dimension N_yx T. It is necessary to find W^{out} such that it gives the best approximate solution for each Y(n) = W^{out}X(n). This means W^{out} is chosen to optimize this equation for all values of n:

Y(1) = W^{out}X(1)
Y(2) = W^{out}X(2)
….
Y(T) = W^{out}X(T)

This is done by finding an approximate solution of the linear equations Y^{target}= W^{out}X for W^{out}. This is a system of N_y*T equations. It is an overdetermined system, so regression is used to find the best approximation. T should be much larger than 1+$N_u + N_x$.

Approximating W^{out}

A version of Mean Square Error (MSE) is used to measure the best approximation for W^{out}.

$$\text{MSE}=\frac{1}{N_y}\sum_{i=1}^{N_y}\sum_{n=1}^{T}[y_i(n) - y_i^{target}(n)]^2,$$

where T is the length of the training set, $y_i(n)$ is the values of the output obtained from Y= W^{out}X for W^{out} estimated, and $y_i^{target}(n)$ is the actual values of the output in the training data set. The goal is to find W^{out}that minimizes the MSE between $y_i(n)$ and $y_i^{target}(n)$. Linear regression can be used to solve this task.

Linear regression

Linear regression models the relationship between two variables by fitting the data points to a linear equation. The independent variable is written on the x axis, and the dependent variable on the y axis. This is often used to show correlation between two variables. The better the data points fit the line, the stronger the correlation. The line is modeled using the equation y=mx+b, where x is the independent variable, m is the slope, b is the y intercept, and y is the dependent variable. The method of least squares is the most common method for finding the best fitting regression line. For each possible line, the vertical distance between each point and the line is calculated, squared (to ensure a positive value), and then added all together into a single value. The line with the smallest value is chosen as the best fitting line [15].

For standard linear regression, multiply each side of $Y^{target}= W^{out}X$ by X^T:

$\rightarrow Y^{target}X^T= W^{out}XX^T$

Then multiply each side by $(XX^T)^{-1}$ (assuming XX^T is invertible):

$\rightarrow Y^{target}X^T(XX^T)^{-1} = W^{out}$

This will solve for W^{out}

Once W^{out} is approximated using the Y^{target} values, it is then used multiplied by the X matrix to generate the y predicted values (Y= W^{out}X).

Ridge regression

In practice, ridge regression is used to solve the system. It is more robust than standard linear regression. The equation$W^{out} = Y^{target}X^T(XX^T + \beta I)^{-1}$ is used, where β is a regularization coefficient and I is the identity matrix. This is similar to the equation used for linear regression except that it has the extra βI term [12]. When β=0 it becomes standard linear regression.

The main advantage of ridge regression is that it avoids overfitting the data. A model that is overfit will understand the details of the training data well, but it fails to generalize, so when it is tested with another dataset it performs poorly. Ridge regression gives a regression model that can generalize patterns, to work well on the training data as well as the testing data. The idea of regularization is to reduce the variability between two datasets by increasing the bias. By making the model perform a little worse during training, it can help it generalize and perform better on new data. Although ridge regression increases the error during training, it will make the model perform better during testing. Ridge regression takes the least square regression line, and it changes the slope of the line. This is regularization. Like method of least squares, ridge regression attempts to minimize the sum of square, but it adds another term, α*slope^2. This acts as a penalizing term which can change the effect of regularization. As α increases, the slope of the regression line decreases [16].

Ridge regression also helps to monitor the output weights inW^{out}. Large weights inW^{out} means that the model will amplify small differences of x(n). Because some networks use the output as the next input, this can be a problem. The βI part of the ridge regression addresses this. Ridge regression solves

$$\mathbf{W}^{\text{out}} = \underset{\mathbf{W}^{\text{out}}}{\arg\min} \frac{1}{N_{\text{y}}} \sum_{i=1}^{N_{\text{y}}} \left(\sum_{n=1}^{T} \left(y_i(n) - y_i^{\text{target}}(n) \right)^2 + \beta \left\| \mathbf{w}_i^{\text{out}} \right\|^2 \right)$$

and chooses parameters that give small output weights as well as small training error. The parameter β controls how much importance is given to each [12].

Testing the network

After the training is completed, W^{out} is computed and fixed. The first new value of u(n), u(T+1), is input into the network. The X vector is generated using the formula. The y predicted value is then generated using y= W^{out}[1; u(n); x(n)], where u(n) is u(T+1) and x(n) is x(T+1), and W^{out} is the weight matrix found earlier in the training phase. For one dimensional data, y is made into the new u, and this process is repeated for the remaining testing data. The output of this process is Y= y(n) for each value of n in the testing range of length S. These are the predicted values.

To test the prediction, Y^{target}= [y(T+1), y(T+2),...y(T+S)], the actual output values, are compared to the values of the Y vector generated by the network, by measuring the error. If it is successful, y(n) should approximately equal y^{target}(n) for some interval n=T+1, T+2,... The longer the interval, the better. If it is unsuccessful, the hyperparameters must be tweaked, and the network must be trained again.

Error measurement:

It is important to see how accurate the model was during the testing period. The actual y values associated with the testing data are known, and they are compared with the predicted y values using squared error. Squared error is calculated by:

SE= $\sum_{i=1}^{N_y} \sum_{i=1}^{S} [y_i(n) - y_i^{target}(n)]^2$, where $y_i(n)$ are the predicted values and $y_i^{target}(n)$ are the actual values, and S is the length of the testing data.

Often, mean squared error is taken instead. Mean square error takes the average of the square error, dividing the sum by the length of the testing data, S:

$$\text{MSE}=\frac{1}{N_y} \sum_{i=1}^{N_y} \sum_{n=1}^{S} [y_i(n) - y_i^{target}(n)]^2$$

Sometimes root mean square error is taken:

$$\text{RMSE}=\frac{1}{N_y} \sum_{i=1}^{N_y} \sqrt{\frac{1}{S} \sum_{n=1}^{S} [y_i(n) - y_i^{target}(n)]^2}$$

The mean squared error quantifies the distance between the target (actual) output and the output predicted by the network. When calculating mean squared error, the difference of the two outputs is calculated, and squared to ensure positive values and give more weight to larger values. These values are all added together, and divided by the number of data points to give the average difference. The smaller the mean squared error, the better the predicted output approximates the actual output [17].

Adjusting the parameters

In most machine learning settings, parameters must be selected manually, even if the parameters will later be learned. When tuning the parameters, one should change one at a time, so that any change that occurs as a result can be associated with that parameter. Once that parameter is set properly, then one should work to tune the next parameter. It is helpful to log the changes of optimizations [12].

It is also possible to automate parameter selection. ESNs have few parameters, so grid search is a good option. It can be implemented using nested loops and a high-level machine learning library. A courser search is done on larger parameter intervals to find the areas that give good results, and then a more detailed search is done on those areas to get more specific values [12].

In an ESN, the three main parameters that need to be chosen and optimized are the input scaling(s), the spectral radius, and the leaking rate(s). The other parameters, such as sparseness of the reservoir and weight distribution, can be set to reasonable default values. Sometimes the results can be optimized if a parameter is split, using different values for different cases. Using different values to scale the first column of W^{in} can be helpful, and using multiple values of α for different units can help with multi-timescale projects [12]. The most practical way to see the accuracy of a reservoir is to train the data and measure the error. If the model does not perform well, it is necessary to adjust the parameters and retrain the network.

A specific application of ESNs

Echo State Networks are reasonably successful at forecasting chaotic systems, which will be discussed in detail.

Chaotic Systems

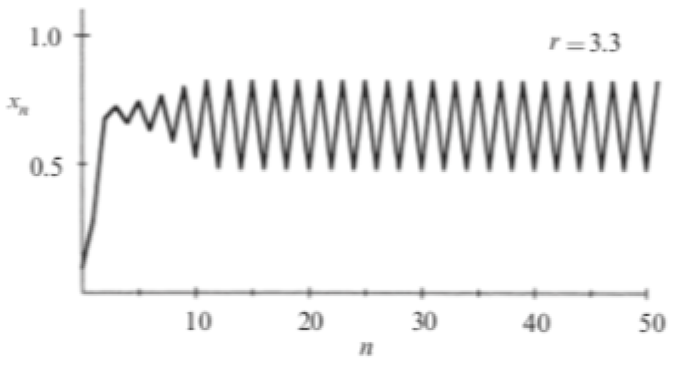
In chaotic systems, states have apparently random behaviors, but they are actually governed by simple rules and initial conditions. Contrary to how it is used colloquially, chaotic behavior is deterministic, not random. This means that given the same initial conditions, a system will perform the same results over and over, even if there is not a pattern. Chaotic systems are also sensitive to initial inputs, meaning that small changes in initial inputs can cause very large changes in the outcome of the system, especially as time goes on. This is known as sensitive dependence on initial conditions [9].

At the 139th meeting of the American Association for the Advancement of Science, Edward Lorenz, a mathematician and meteorologist, posed the question, “Does the flap of a butterfly’s wings in Brazil set off a tornado in Texas?” Lorenz was demonstrating what came to be known as the “butterfly effect,” that complex dynamical systems such as weather could exhibit unpredictable behavior because of small changes in the initial conditions that could have large and divergent effects on the system's outcomes. Because the systems are sensitive to the smallest changes, their outcomes are unpredictable. Lorenz found that although these systems are deterministic, human measurement of them is imprecise, and therefore we often get it wrong. He found that the cause and effect aspect in nature is too complex to be able to predict fully [10].

Logistic Map

The logistic map is the simplest model of a chaotic system. It is defined as $x_{n+1}= rx_n(1-x_n)$, where each term is r times the previous term, multiplied by one minus the previous term, and r is a parameter. x_0 is the initial condition, and plugging it into the formula yields x_1, which when plugged back into the formula yields x_2… There are intervals for r in which the dynamics of x_n are periodic, and intervals that they are chaotic. When r is between 3.57 and 4, the model exhibits chaotic behavior [9].

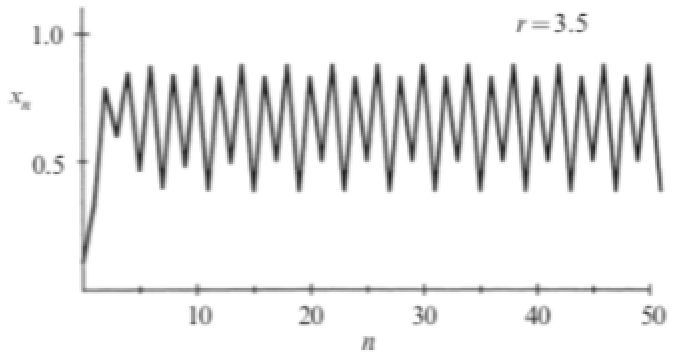
r=3.3: period-2 cycle



[9]

When r=3.3, x_n settles into an oscillation with a period of two. This means the pattern repeats every two time steps. The data has a pattern, which means it does not exhibit chaotic behavior.

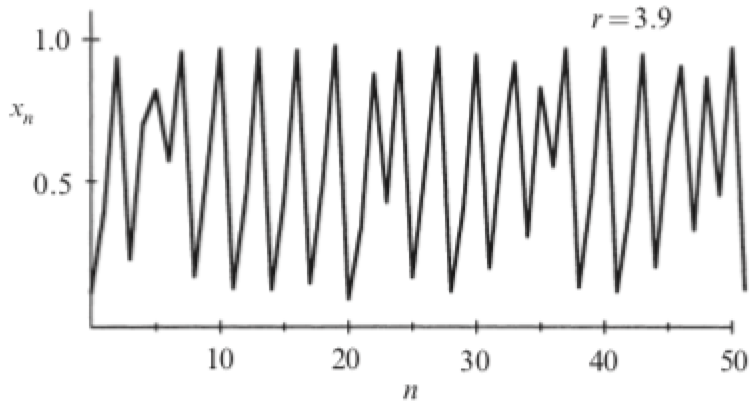
r=3.5: period-4 cycle



[9]

When r= 3.5, x_n settles into an oscillation with a period of four. This is also not chaotic behavior.

r=3.9: aperiodic behavior



[9]

As r approaches 3.569946 (known as r_∞), the period approaches infinity. This means that once r increases beyond 3.57, x_n will not have repeating cycles, and the system will exhibit chaotic behavior. For r=3.9, the system exhibits chaotic behavior, and is aperiodic [9], which means that the shape of the graph is irregular and does not have a pattern that repeats. For r> r_∞, the sequence has some periodic windows for certain values of r, interspersed between mostly chaotic data.

Mackey-Glass

Another example of a chaotic system is the Mackey-Glass equation, which is a nonlinear time-delay differential equation.

Mackey-Glass equation:

$$\frac{dx}{dt} = \beta\frac{x_\tau}{1+x_\tau^{\,n}} - \gamma x, \qquad \gamma, \beta, n > 0$$

τ= delay, x_τ= x(t-τ)

x_τ represents the value of the variable x at time t-τ, where τ is the delay. γ, β , n are parameters. For some values of the parameters, the system exhibits a sequence of period doubling bifurcations and chaotic behavior.

Physiologically, this equation models the concentration of circulating blood cells in an individual. x is the concentration of circulating blood cells, and β_0 and n are constants which describe the dependence of the production of these blood cells as a function of x_τ. It is a feedback system, in which the concentration of blood cells in the body determines the rate that the blood cells are produced in the future. Feedback systems often have a lag time between when the value of the variable is sensed, and the system's response, which τ represents here. If x_τ were very small but greater than zero ($0<x_\tau<<1$), then the person would be very sick and not able to make these blood cells, and if x_τ were much larger than one ($1<<x_\tau$), the person would have too

many blood cells circulating, which would also slow down the production rate. The production rate is maximized for some intermediate value of x_τ.

Mackey and Glass suggested that physiological disorders (called dynamical diseases) occur when there are qualitative changes in the dynamics of the system. Changes in the parameters, which might be caused by disease or environment, cause bifurcation in the equation dynamics. Bifurcations being associated with abnormal dynamics of disease has applications in many fields of medicine such as hematology, cardiology, and neurology [11].

Lorenz System

Another example of a chaotic system is the Lorenz system. Edward Lorenz derived these equations in 1963 as a simplified model of convection rolls, or movement of air, in the atmosphere.

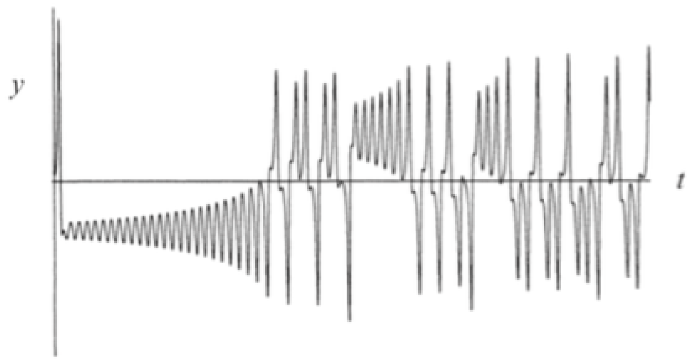
The Lorenz equations:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = rx - y - xz$$
$$\frac{dz}{dt} = xy - bz \qquad [9]$$

σ, r, b >0 are parameters. σ is the Prandtl number, r is the Rayleigh number, and b does not have a name. The system only has two nonlinear terms, the xz and xy. The Lorenz equations are symmetric, meaning that using (x, y) and (-x, -y) in the equations will give the same output. The Lorenz system is also dissipative, meaning volumes in the space contract under the flow. Like the waterwheel, the Lorenz system has two types of fixed points. The origin (x*, y*, z*)= (0, 0, 0) is a fixed point for all values of parameters. For r>1, there is a pair of symmetric fixed points $x^* = y^* = \pm\sqrt{b(r-1)}$, z*= r-1. Lorenz referred to these points as C^+ and C^- [9].

There is a special value of r, $r_H = \frac{\sigma(\sigma+b+3)}{\sigma-b-1}$, which makes the fixed points become unstable. The trajectory is stable for $1 < r < r_H$. Lorenz studied the specific case where σ=10, b= $\frac{8}{3}$, r=28. Because $r_H = 24.74$ in this case, he knew the system would act chaotically (because $r > r_H$ and therefore it is unstable). He integrated from (0, 1, 0) and got the following results:

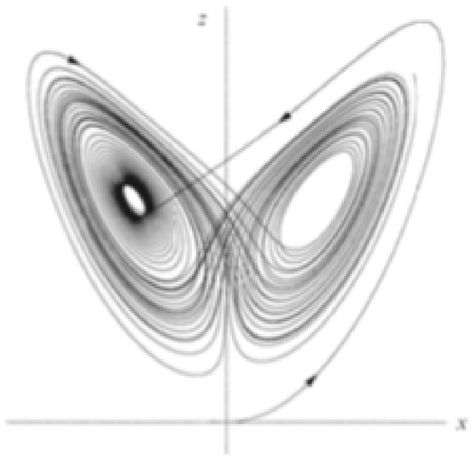
The Lorenz System for σ=10, b=$\frac{8}{3}$, r=28, plotting y(t)



[9]

After an initial time period, the solution develops into an irregular oscillation that continues as $t \rightarrow \infty$ but never repeats. The motion is aperiodic. When x(t) is plotted against z(t), a butterfly shape appears:

The Lorenz System forσ=10, b=$\frac{8}{3}$, r=28; x(t) plotted against z(t)

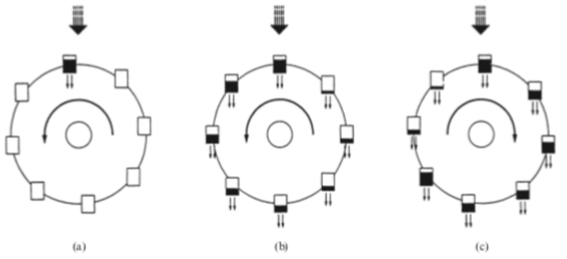


[9]

The two points within the loops of the wings are C^{+} and C^{-}. The motion has sensitive dependence on initial conditions, meaning two trajectories with similar initial conditions will still diverge quickly and have completely different futures. The Lorenz system with these parameters exhibits chaotic behavior. Although the trajectories appear to cross, in three-dimensional space they do not cross. Lorenz explained that there is an “infinite complex of surfaces” where the surfaces appear to merge. This set of points therefore has zero volume but infinite surface area, which is known as a fractal [9].

A practical illustration of the Lorenz equations in a mechanical system is the mechanical waterwheel, invented by Willem Malkus and Lou Howard at MIT in the 1970s. Water is poured from the top, and fills the cups. The water needs to flow fast enough to allow the top cups to fill up enough to overcome friction and cause the wheel to turn. When the top cup fills and becomes heavy, the wheel will turn. The rotation of the wheel is equally likely in either direction, and depends on the initial conditions. If the flow rate is increased, the steady rotation will become destabilized, and the motion will become chaotic. The wheel will rotate in one direction, then some of the cups will become too heavy and the wheel will not have enough inertia to get them over the top, so the wheel will slow and may change direction. The wheel will continue to change direction erratically [9].

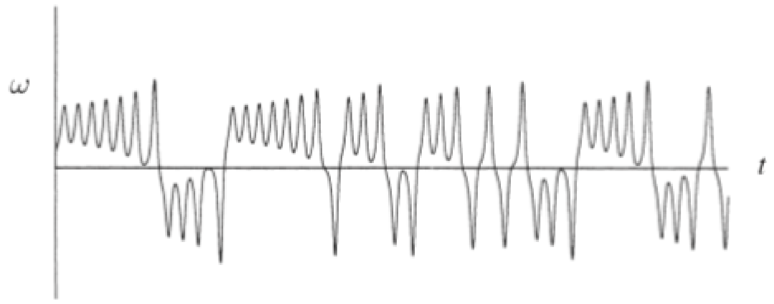
Waterwheel with water poured steadily from the top
The wheel will rotate and may change direction erratically



[9]

The wheel's angular velocity ω(t) can be graphed as shown below:

The angular velocity ω(t)



[9]

The following equations model this chaotic waterwheel:

$$\dot{a}_1 = \omega b_1 - K a_1$$
$$\dot{b}_1 = -\omega a_1 - K b_1 + q_1$$
$$\dot{\omega} = (-\nu\omega + \pi g r a_1)/I$$

[9]

Setting each derivative to zero gives

$a_1 = \omega b_1/\mathrm{K}$ (*)
$\omega a_1 = q_1 - \mathrm{K}b_1$ (**)
$a_1 = \mathrm{v}\omega/\pi\mathrm{gr}$ (***) [9]

Substituting a_1 from (*) into (**) and solving for b_1 gives $b_1 = \frac{Kq_1}{\omega^2+K^2}$

Equating (*) and (***) gives $\omega b_1/\mathrm{K} = \mathrm{v}\omega/\pi\mathrm{gr}$. Either $\omega=0$, or $b_1 = \mathrm{Kv}/\pi\mathrm{gr}$.

These give two fixed points: If $\omega=0$, then $a_1=0$ and $b_1 = q_1/\mathrm{K}$. This gives a fixed point $(a_1^*, b_1^*, \omega^*) = (0, q_1/\mathrm{K}, 0)$, which corresponds to the wheel's state of no rotation. If $\omega \neq 0$, then $b_1 = \frac{Kq_1}{\omega^2+K^2} = \mathrm{Kv}/\pi\mathrm{gr}$. We know that $\mathrm{K} \neq 0$, thus we can divide both sides by K and get $\frac{q_1}{\omega^2+k^2} = \mathrm{v}/\pi\mathrm{gr}$. Solving for ω^2 gives $\omega^2 = \frac{\pi g r q_1}{v} - K^2$. If the right-hand side of the equation is positive, then there are two solutions, $\pm\omega^*$, which correspond to steady rotation in either direction. These solutions exist if and only if $\frac{\pi g r q_1}{K^2 v} > 1$. $\frac{\pi g r q_1}{K^2 v}$ is the Rayleigh number, the same as in the Lorenz equations, and is a ratio of gravity and inflow over leakage and damping. Thus there will only be steady rotation if this number is large enough [9].

Using ESNs to forecast chaotic data

Now that I have explained how echo state networks are used to predict data, I will show an implementation in Python, using ESN on chaotic data. Below is the code for an ESN run with Mackey-Glass data.

The Python packages must be loaded, and variables for the sizes of the data initialized. Here, the training length and test length are both set to 2000, and the initial length to 100. The initial length of data is discarded. It accounts for the initial period of the chaotic data when it is settling into the model.

```
# -*- coding: utf-8 -*-
"""
A minimalistic Echo State Networks demo with Mackey-Glass
(delay 17) data
in "plain" scientific Python.
from https://mantas.info/code/simple_esn/
(c) 2012-2020 Mantas LukoÅ¡eviÄ ius
Distributed under MIT license
https://opensource.org/licenses/MIT
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy import linalg
# numpy.linalg is also an option for even fewer dependencies

# load the data
trainLen = 2000
testLen = 2000
initLen = 100
data = np.loadtxt('MackeyGlass_t17.txt')

# plot some of it
plt.figure(10).clear()
plt.plot(data[:1000])
plt.title('A sample of data')
```

The first step is to load the data and plot it. The next step is to generate the ESN's reservoir. The leaking rate α (here denoted as a) is set to .3. Because W^{in} and W are generated randomly, a random seed is created before their initialization. Then W^{in} and W are initialized. Then W is normalized by computing its spectral radius and dividing W by its spectral radius.

```
# generate the ESN reservoir
inSize = outSize = 1
resSize = 1000
a = 0.3 # leaking rate
np.random.seed(42)
Win = (np.random.rand(resSize,1+inSize) - 0.5) * 1
W = np.random.rand(resSize,resSize) - 0.5
# normalizing and setting spectral radius (correct, slow):
print('Computing spectral radius...')
rhoW = max(abs(linalg.eig(W)[0]))
print('done.')
W *= 1.25 / rhoW
```

An empty matrix X is initialized, and then the network is run with the data to generate each x(n), using the expanded formula, $x(n) = (1-\alpha)x(n-1) + \alpha \tanh[W^{in}[1; u(n)] + Wx(n-1)]$, which incorporates the leaking rate as discussed previously. Each value for x(n) with its corresponding time series output u(n) is loaded into the matrix X.

```
# allocated memory for the design (collected states) matrix
X = np.zeros((1+inSize+resSize,trainLen-initLen))
# set the corresponding target matrix directly
Yt = data[None,initLen+1:trainLen+1]

# run the reservoir with the data and collect X
x = np.zeros((resSize,1))
for t in range(trainLen):
    u = data[t]
    x = (1-a)*x + a*np.tanh( np.dot( Win, np.vstack((1,u)) ) +
np.dot( W, x ) )
    if t >= initLen:
        X[:,t-initLen] = np.vstack((1,u,x))[:,0]
```

Once X is filled, ridge regression is used to find the best values for W^{out}. β, the regularization coefficient, is denoted as ‘reg’.

```
# train the output by ridge regression
reg = 1e-8  # regularization coefficient
# direct equations from texts:
#X_T = X.T
#Wout = np.dot( np.dot(Yt,X_T), linalg.inv( np.dot(X,X_T) + \
#    reg*np.eye(1+inSize+resSize) ) )
# using scipy.linalg.solve:
Wout = linalg.solve( np.dot(X,X.T) +
reg*np.eye(1+inSize+resSize),
    np.dot(X,Yt.T) ).T
```

Now W^{out} is used to generate the predicted y values for the test data, together with the new input values. The matrix Y is filled with each value of y. In generative mode, the y value predicted is used as the u for the next value of t. In predictive mode, the next value of t uses the corresponding data from the dataset.

```
# run the trained ESN in a generative mode. no need to
initialize here,
# because x is initialized with training data and we continue
from there.
Y = np.zeros((outSize,testLen))
u = data[trainLen]
for t in range(testLen):
    x = (1-a)*x + a*np.tanh( np.dot( Win, np.vstack((1,u)) ) +
np.dot( W, x ) )
    y = np.dot( Wout, np.vstack((1,u,x)) )
    Y[:,t] = y
    # generative mode:
    u = y
    ## this would be a predictive mode:
    #u = data[trainLen+t+1]
```

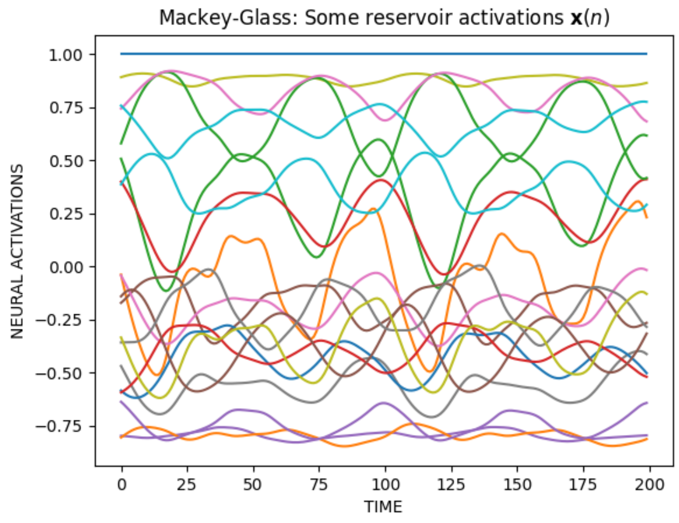
Then the mean squared error (MSE) is computed to quantify the accuracy of the predictions for y.

```
# compute MSE for the first errorLen time steps
errorLen = 500
mse = sum( np.square( data[trainLen+1:trainLen+errorLen+1] -
    Y[0,0:errorLen] ) ) / errorLen
print('MSE = ' + str( mse ))
```

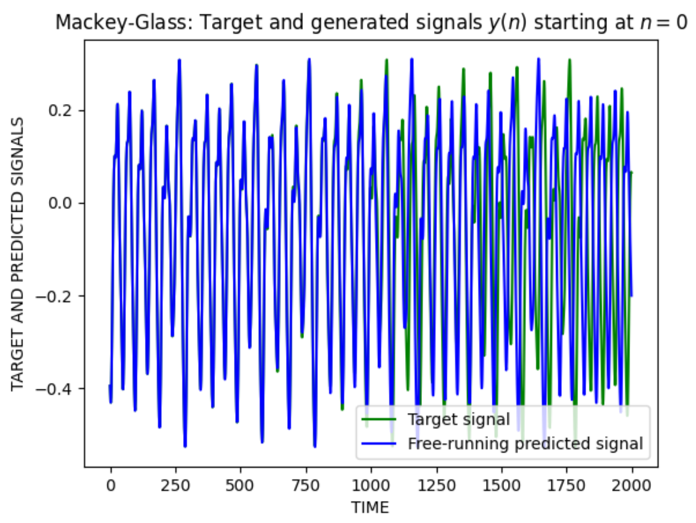
Lastly, the results are plotted.

```
# plot some signals
plt.figure(1).clear()
plt.plot( data[trainLen+1:trainLen+testLen+1], 'g' )
plt.plot( Y.T, 'b' )
plt.title('Mackey-Glass: Target and generated signals $y(n)$
starting at $n=0$')
plt.legend(['Target signal', 'Free-running predicted signal'])
plt.xlabel('TIME')
plt.ylabel('TARGET AND PREDICTED SIGNALS')

plt.figure(2).clear()
plt.plot( X[0:20,0:200].T )
plt.title(r'Mackey-Glass: Some reservoir activations
$\mathbf{x}(n)$')
plt.xlabel('TIME')
plt.ylabel('NEURAL ACTIVATIONS')

plt.figure(3).clear()
plt.bar( np.arange(1+inSize+resSize), Wout[0].T )
plt.title(r'Mackey-Glass: Output weights $\mathbf{W}^{out}$')
plt.xlabel('NEURONS')
plt.ylabel('NEURAL WEIGHTS')

plt.show()
```

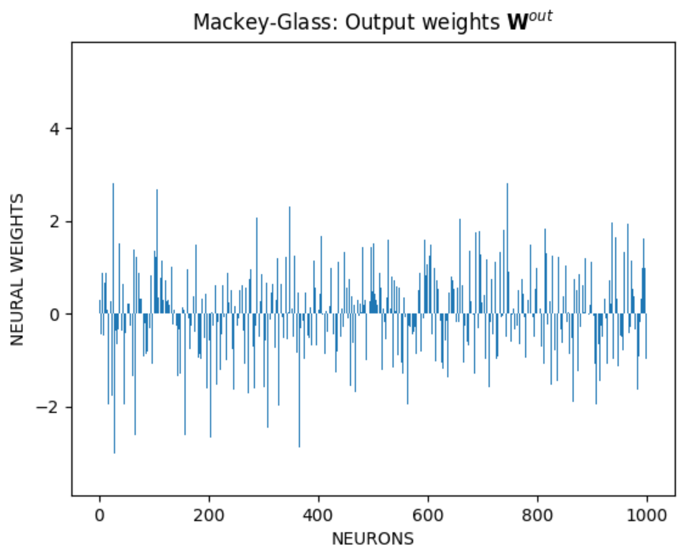
The following image depicts the activations of 20 neurons in the reservoir from time zero to 200. The x axis represents the time n, and the y axis represents the activation. At n=0, the neuron is at its initial state of activation. The more the line moves and squiggles, the more active the neuron is. As shown here, in this example almost all of the neurons are very active. The top line, which is straight, represents a neuron that has not been activated at all. The fact that most of the neurons are active indicates that the model performed well.



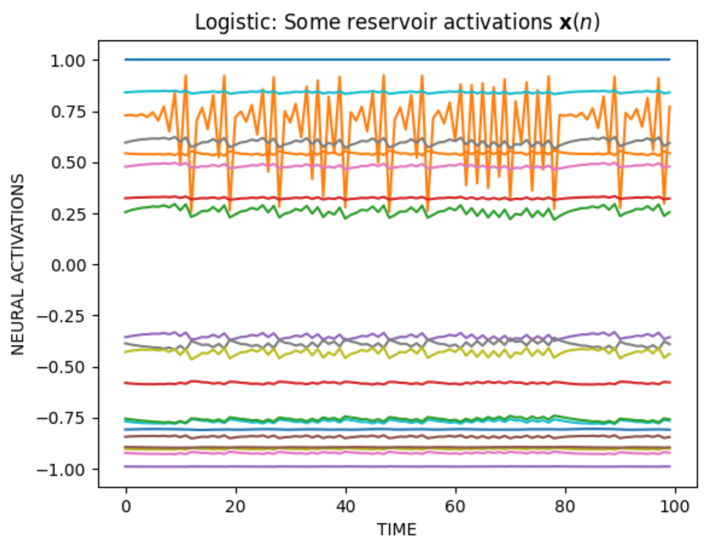
The following image shows the target signal and predicted signal graphed together. The model predicts very well for the first half of the data until n=1000, with the blue and green lines overlapping. The MSE from zero to 500 is 1.0246029601644007e-06, a very small number, indicating a good prediction. The MSE from zero to 1000 is 0.00021410850858493203. The MSE of the whole trial, from zero to 2000, is 0.04389133334038875. This demonstrates that the model does well for short-term predictions, but gets less accurate for longer predictions. This makes sense given that the system is chaotic; small deviations of the prediction from the target data will grow over time and will cause them to diverge.



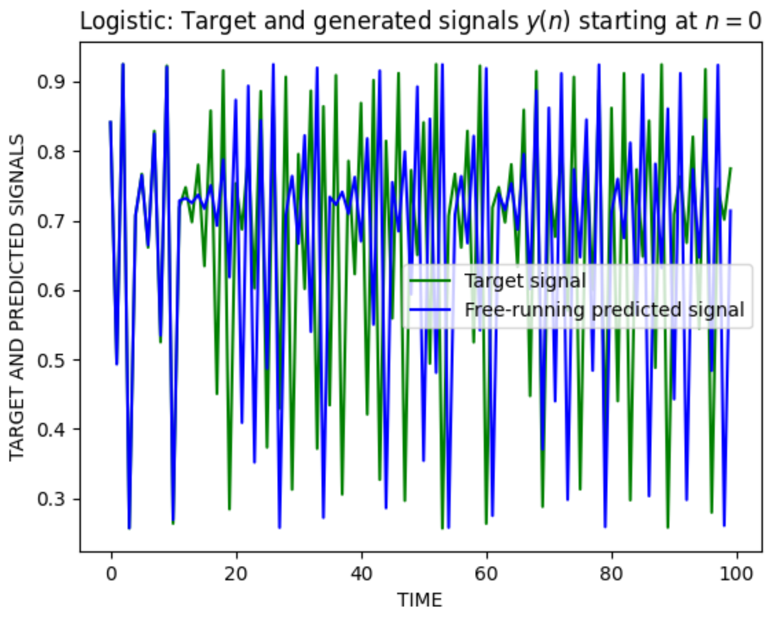
The following image depicts the output weights for W^{out}. The x axis corresponds to the 1000 neurons that make up the neural network, and the y axis represents the neurons' weights. W^{out} is found during the training phase, using ridge regression on the X vector and Y^{target}.



Mantas Lukosevicius implemented this method on the Mackey-Glass data, and he was able to optimize the parameters to achieve these great results. Using these same parameters on the other chaotic data sets does not yield as good results. Running Lukosevicius's code with the Logistic model data does not perform as well, because the parameters are not optimized for this specific model. Below are the reservoir activations from this run.



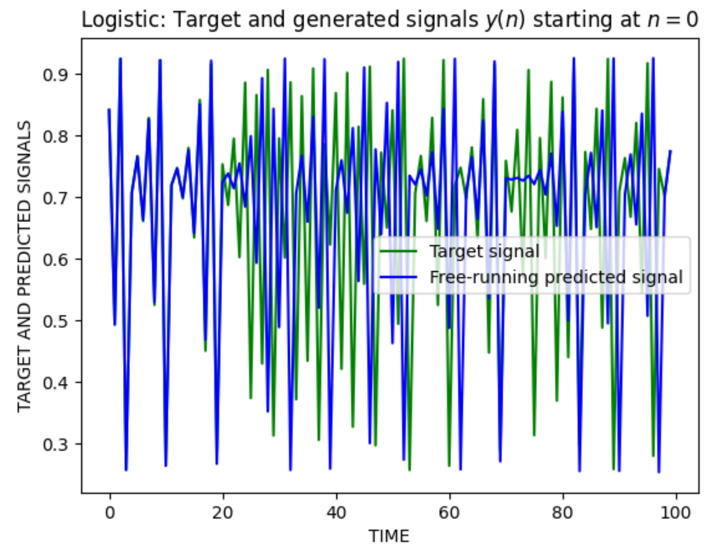
As depicted above, most of the neurons are not very activated. Only the neuron activity depicted by the orange line is really active. The remainder of the lines are close to straight, which represents low neuron activity. This can help explain why it does not yield as satisfactory results.



Depicted above is the target and predicted signals graphed together. The prediction is very good from zero to ten, where the target and predicted signals overlap, but afterwards the two signals diverge. The MSE from zero to 10 is 1.3276776401990342e-05, while the MSE for the whole range of prediction until n=100 is 0.07570337842957092, much larger than the previous one. Once again, this demonstrates that the model predicts well up to 10 time points into the future, but then diverges from the target.

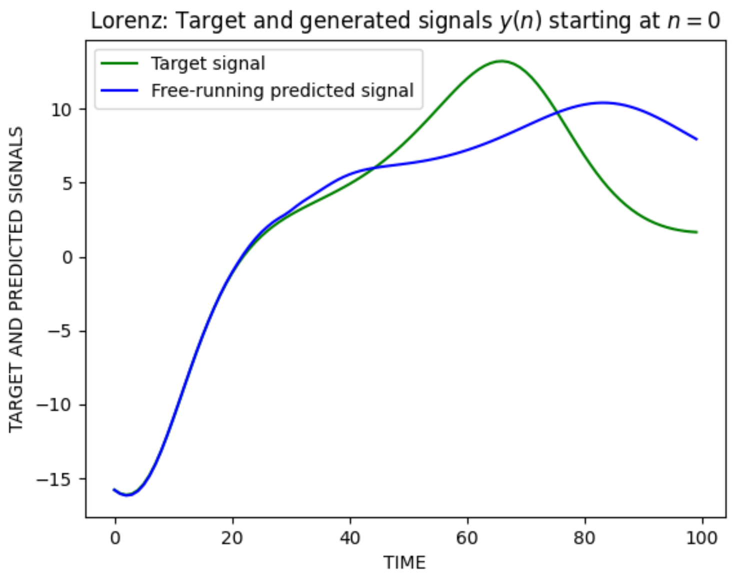
Changing α from .3 to .2 makes the MSE decrease from 0.010397477217171144 to 4.1885438911715894e-05, which is a large improvement. The prediction looks as follows:

α = .2



This shows a better prediction for α = .2 than the prediction that α = .3 gave. This demonstrates that adjusting the parameters can help improve performance of the prediction.

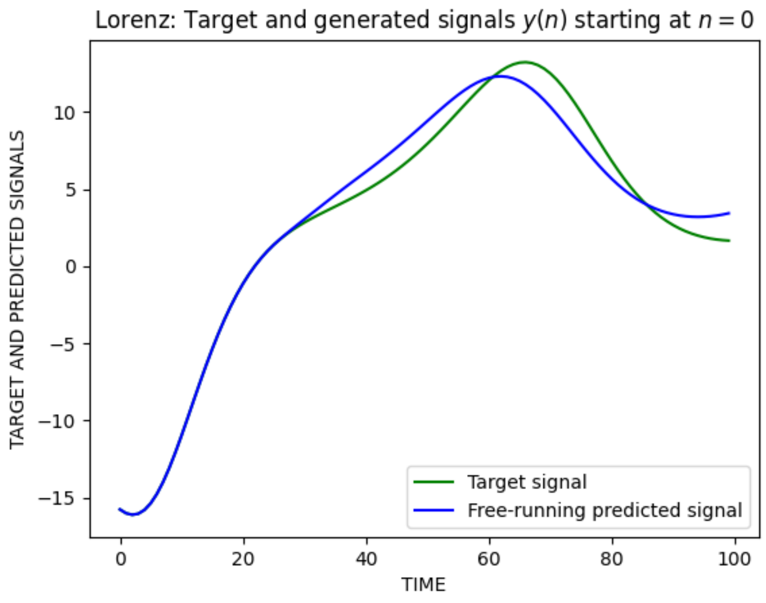
When the Lorenz data is input into the ESN, the prediction looks as follows:



The MSE from zero to 100 is 12.255376483296505, a large number. The MSE from zero to 20 is 0.0012579830546665888, indicating a more accurate short-term prediction than long-term.

Increasing the reservoir size from 100 to 1000 improves the prediction results:

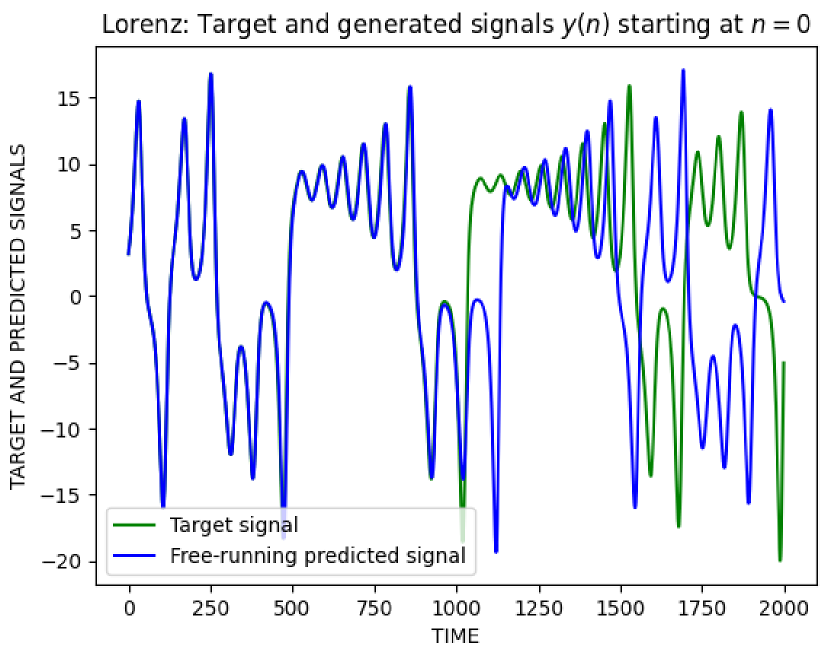
Reservoir size=1000



The MSE from zero to 100 is 1.1398741526878147, and the MSE from zero to 20 is 2.2988206596351826e-05. Both of these values are an improvement from the smaller reservoir size. Once again, adjusting the parameters helps improve the accuracy of the prediction.

Adjusting the parameters further significantly improves the results. Some of these adjustments include changing the training length from 1000 to 5000, test length from 100 to 2000, changing the leaking rate from 3 to .16, normalizing the spectral radius of W from 1.25 to 1.2, and changing the regularization coefficient βfrom 1e-8 to 1.56e-5.

training length= 5000; test length= 2000; α= .16; ρ(W)= 1.2; β= 1.56e-5



MSE from zero to 20: 2.4017663892153266e-08

MSE from zero to 100: 6.628954668444104e-05

MSE from zero to 1000: 0.3604076445313357

MSE from zero to 2000: 66.26048291081491

This forecast is a significant improvement. The prediction's accuracy is improved for a much longer length of time. This illustrates how the model can perform very well when the parameters are adjusted correctly.

Conclusion

Echo State Networks are an important application of reservoir computing. This thesis has demonstrated their ability of making accurate short-term predictions for chaotic data. In general, a benefit of ESNs as opposed to other applications of RNNs is that because the input weights and network connection weights are assigned randomly, only the output weights need to be trained. This helps with computability and speed. This has wide applications, specifically with regard to forecasting chaotic time series.

Acknowledgements

I would like to thank Professor Gidea for all his help and guidance with researching and writing this thesis. He went above and beyond his duties as a mentor by not only assisting me with research, but also clarifying many of the topics and answering all of my questions. I would also like to express my gratitude to the entire Departments of Mathematics and Computer Science for making my academic experience at Stern College for Women both educational and enjoyable.

References:

[1] Copeland, B.J. “Artificial Intelligence.” *Encyclopædia Britannica*, Encyclopædia Britannica, Inc., www.britannica.com/technology/artificial-intelligence.

[2] Kersting, Kristian. “Machine Learning and Artificial Intelligence: Two Fellow Travelers on the Quest for Intelligent Behavior in Machines.” *Frontiers*, Frontiers, 24 Oct. 2018, www.frontiersin.org/articles/10.3389/fdata.2018.00006/full.

[3] “What Is Machine Learning? A Definition - Expert System.” *Expert.ai*, 6 May 2020, www.expert.ai/blog/machine-learning-definition/.

[4] Hao, Karen. “What Is Machine Learning?” *MIT Technology Review*, MIT Technology Review, 5 Apr. 2021, www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-a nother-flowchart/.

[5] Hardesty, Larry. “Explained: Neural Networks.” *MIT News | Massachusetts Institute of Technology*, 15 Apr. 2017, news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[6] Nielsen, Michael A. “Neural Networks and Deep Learning.” *Neural Networks and Deep Learning*, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/chap1.html.

[7] Amidi , Afshine, and Shervine Amidi. “Recurrent Neural Networks Cheatsheet Star.” *CS 230 - Recurrent Neural Networks Cheatsheet*, stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[8] “What Are Recurrent Neural Networks?” *IBM*, 14 Sept. 2020, www.ibm.com/cloud/learn/recurrent-neural-networks.

[9] Strogatz, Steven. *Nonlinear Dynamics and Chaos: with Applications to Physics, Biology, Chemistry, and Engineering*. Second ed., CRC Press, Taylor Et Francis Group, 2018.

[10] Vernon, Jamie L. “Understanding the Butterfly Effect.” *American Scientist*, 12 June 2017, www.americanscientist.org/article/understanding-the-butterfly-effect.

[11] Glass, Leon, and Michael Mackey. “Mackey-Glass Equation.” *Scholarpedia*, 2010, www.scholarpedia.org/article/Mackey-Glass_equation.

[12] Lukosevicius, Mantas. “A Practical Guide to Applying Echo State Networks.” *Neural Networks: Tricks of the Trade, Reloaded*, Springer, 2012.

[13] Cheever, Erik. “Eigenvalues and Eigenvectors.” Linear Physical Systems Analysis, lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html.

[14] “Lecture 6: Matrix Norms and Spectral Radii - Drexel University.” *Drexel University College of Arts and Sciences*, www.math.drexel.edu/~foucart/TeachingFiles/F12/M504Lect6.pdf.

[15] “Linear Regression.” *Yale University Department of Statistics and Data Science*, www.stat.yale.edu/Courses/1997-98/101/linreg.htm.

[16] T., Bex. “Intro to Regularization With Ridge And Lasso Regression with Sklearn.” *Medium*, Towards Data Science, 28 Feb. 2021, towardsdatascience.com/intro-to-regularization-with-ridge-and-lasso-regression-with-sklearn-edcf4c117b7a.

[17] Glen, Stephanie. “Mean Squared Error: Definition and Example.” *Statistics How To*, www.statisticshowto.com/mean-squared-error/.