Declaration of Contribution

The research completed and presented within this document is declared to be completed by the author except where otherwise mentioned and/or cited. Much research was conducted by the author in MATLAB where the radial basis function neural network was implemented and code output was plotted. In the research results, one section on the Lorenz attractor analysis displays and discusses research conducted by the project partner Kyla Klein. With this exception, all research results were authentically produced solely by the author unless otherwise cited.

Artificial Neural Networks and the Prediction of Chaos

by Thomas J. Prince
researched in conjunction with Kyla Klein

Supervised by Charles Unsworth, The University of Auckland

Date of Publification: 30th October 2020

Abstract

Chaos and its field of study relates to deterministic dynamical systems with aperiodic and bounded trajectories that have sensitive dependence on initial conditions. These systems are globally stable, however, locally unstable. Similar nearby trajectories diverge exponentially from eachother but the differing orbits show fractality and self-similarity. Using chaos theory, one can analyse and quantify the complex dynamical behaveiour of nonlinear time-series.

The Lorenz equations model fluid convection. With the right parameter values, these differential equations produce a chaotic, non-linear, time-continuous, and time-invariant dynamical system in three dimensions. The equations are as follows:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z)$$
$$\frac{dz}{dt} = xy - \beta z$$

where σ, ρ, and β are parameters. Classically used chaotic producing parameters in the Lorenz equations are the values $\sigma = 10$, $\beta = \frac{8}{3}$, and $\rho = 28$.

A joint project aim was to develop chaos theory algorithms to analyse the Lorenz chaotic attractor, then employ the use of artificial neural networks to predict its future behaviour. This report focusses on artificial neural network models, in particular, the radial basis function neural network. Within this research, radial basis function neural networks have been implemented to predict future behaviour of the Lorenz dynamical system, firstly, by using a tapped delay line of a single variable's orbit and varying embedding delay and dimension. Similar networks were constructed to take a full three-dimensional Lorenz state input and predict successive states.

Research outcomes determined that a radial basis function neural network was able to be trained to fit a Lorenz variable's chaotic orbit so to successfully predict its dynamical behaviour from a beginning tapped delay line of given embedding dimension. Depending on parameters used in the network and which variable was being predicted, predictions of up to 8 units of time followed a computed orbit. Additionally, it was found that the x individual variable's information could be used to predict other variables and even reproduce the complete three-dimensional structure of the Lorenz system.

Using an entire Lorenz state as input provided significantly more information to the neural network which was reflected in its output and performance. An actual computed orbit could be followed arbitrarilly close by a predicted orbit for up to approximately 15 units of time in some simulations. Regardless of the fact that the prediction diverged from the computationally solved trajectory, high-level behaviour of the system's dynamics were observably retained in the prediction. Considering the implications of sensitive dependence on initial condition poses the question – could the predicted orbit not be *"correct"* to shadow some other orbit after it diverges from the numerically computed one?

In application to real-life signals, the performance of a radial basis function neural network on noiseless chaotic attractors shows promise. Networks appeared to be able to capture the overall high-level behaviour of the Lorenz chaotic attractor, such as the boundedness of orbits and their aperiodicity. Application of networks to signals such as the electroencephalogram would make for suiting subsequent research to that completed and delineated within this report.

Acknowlegement

The following document results from the conclusion of four years at the University of Auckland in my decided Engineering Science specialisation programme. First thanks must go to my parents and extended family, who have always been supportive of whichever educational or occupational pathway I decide to endeavour upon. Their support, both financial and emotional, has been invaluable. I would not be where I am today without them. Charles Unsworth, the supervisor of this honours project, must subsequently be acknowledged for the much appreciated and required project direction and knowledge imparted. He had our best interests at heart. Artificial neural networks and chaos theory are not trivial topics. Without his help, I would have certainly remained lost and clueless on the subject. Finally, to my project partner Kyla Klein who conducted her part of the research on Chaos Theory analysis. I was priviledged to have been able to work alongside such a competent individual. This past year has proven difficult, particularly due to the COVID-19 pandemic and its resulting impairment on structured in-person learning and induced additional stress. Regardless, Kyla and myself have withstood the worst and should be pleased with the finished final reports pertaining to this ENGSCI 700 honours project.

Contents

List of Figures

Abbreviations

ANM Artificial Neuron Model

ANN Artificial Neural Network

EEG Electroencephalogram

MLP Multilayer Perceptron

NARX Nonlinear Autoregressive Exogenous Model

RBF Radial Basis Function

RBFNN Radial Basis Function Neural Network

SDIC Sensitive Dependence on Initial Conditions

1 Introduction

The allocated project was Engineering Science Project 57: Chaos Theory Modelling & Artificial Neural Network Prediction of Time-Series from Lorenz Chaotic Attractors & Brain Signals. Two distinct roles divided the project, with the use of Artificial Neural Network (ANN) models being the primary focus of this research. Kyla Klein has undertaken work in the area of chaos theory model analysis. An ideal joint project outcome was decidedly defined so to analyse the Lorenz chaotic attractor using chaos theory algorithms, then predict its future behaviour using ANN models. In this report, research has been undertaken using the Radial Basis Function Neural Network (RBFNN). This network has demonstrated strong ability to predict and reconstruct time-series produced through computationally solving the Lorenz equations.

Sections that constitute this report include a literature review, research methodologies, research results, a discussion, summary, and conclusion. A detailed literature review on existing studies and literature relevant to this project was undertaken. It has the purpose of providing sufficient background knowledge so to grasp the context of the problem this research addresses. Research conducted involves use of the RBFNN to predict the future dynamics of the Lorenz system. Using two methods of prediction, the network proved successful in predicting the future dynamical behaviour of the Lorenz attractor.

The first method of prediction outlined within this research included the use of a tapped delay line of a single variable's orbit as the input to the RBFNN. Subsequently, the Lorenz dynamics were predicted using its three-dimensional state as input, where the network was trained to predict its successive state after a time dt had passed. Both methods proved successful and displayed impressive results.

2 Literature Review

This literature review enumerates some existing research in attempt to provide a sound foundation for the necessary relevant concepts pertaining to this report. The purpose of this review is to firstly survey existing studies and literature that are relevant to the project then describe them in a comprehensive manner. A referenced exploration of research conducted to date provides sufficient background knowledge within the area of research so to grasp the context of the problem, the issues the project intends to address, and hence the overall motivation behind this project.

2.1 Chaos Theory

The notion of chaos is an essential concept within this research. As stated by Priddy et al.: '*Since neural networks are data driven, the adage garbage in, garbage out is highly relevant to the task of building a neural network*' ([1]). Understanding fundamental chaos theory concepts hence supports the endeavour to properly collect, prepare, label, and code the data later extracted from the Lorenz equations into a high-functioning artificial neural network.

2.1.1 Literature Survey

Complexity Explorer's online course *Introduction to Dynamical Systems and Chaos* [2] begins with fundamental concepts assuming only a high-school equivalent level of mathematics, hence making the source accessible and intuitive. After describing the concept of functions, it delves deeper into the realm of dynamical systems and iterative processes where the notion of chaos is introduced and defined. Through this process, concepts such as orbits (or trajectories), phase spaces, fixed points and their stability, attractors, periodic behaviour, limit cycles, bifurcations, differential equations, and determinism are presented and explained. Finally, the three-dimensional continuous time Lorenz and Rössler attractors are introduced and discussed, as well as the concept of stretching and folding being the foundation on which chaotic behaviour depends on.

Kautz's book *Chaos: The Science of Predictable Random Motion* [3] delves deeper into the theory, providing further completeness to the overall picture. A multitude of examples are addressed and discussed which delineate chaotic properties and analytical methods. The book contains two chapters of significance with relevance to the research conduct. One

pertains to the butterfly effect; a popular alternative term to Sensitive Dependence on Initial Conditions (SDIC) coined by Edward Lorenz in his 1972 presentation to the American Association for the Advancement of Science [4]. The second discusses prediction and how return maps can show some insight into a dynamical systems forthcoming behaviour.

Edward Ott's book *Chaos in Dynamical Systems* [5] discusses further chaotic properties, observations, and analytical methods. One such mention is of delay coordinates which are later used as RBFNN inputs in this research.

The Lorenz attractor is that which is being predicted, and suitingly, Lorenz's book *The Essence of Chaos* was reviewed. The book introduces and describes chaotic concepts, with particular sections on SDIC described through the notion of a pinball machine and a trajectory down a ski slope.

2.1.2 Description of Research

The modern scientific term *deterministic chaos* refers to irregular and unpredictability of a deterministic dynamical system over time [6]. In the study of dynamical systems, such a system is considered chaotic if [2]:

1. its rule is deterministic,
2. its orbits are bounded,
3. its orbits are aperiodic,
4. it has sensitive dependence on initial conditions.

Determinism, simply implies the following of strict rules. The present state of a deterministic system is, in theory, fully defined by its initial or previous conditions [7]. It means there is no stochasticity or randomness [7].

Systems that vary in discrete steps are known as mappings [8]. The mathematical notion of describing deterministic mappings is called a difference equation [8]. For time-continuous systems, the motions ae known as flows, with the descriptive equation being a differential equation [8].

Chaotic orbits must have sensitive dependence on initial conditions. Simply stated: '*for any initial condition x_0, other initial conditions very near to it eventually end up far away*' ([2]). This means that trajectories that begin very near to eachother diverge and don't follow eachother indefinitely.

The concept of an attractor is such that there exists an attracting set in the phase space i.e. a non-zero bounded subset, which regions of initial conditions asymptote as time continues [5]. An attractor is strange if [2]:

1. nearby orbits get pulled into it i.e. it is stable,

2. motion on the attractor is chaotic.

In some experiments, not all components of a system's multi-dimensional state can be measured [5]. If only one component (i.e. one scalar function of the state vector) can be measured, delay coordinates can be taken which can make a surface of section revealing fractal structure of the whole dynamical system [5]. This notion is applied later in research where delay coordinates are used for prediction. Kyla's results of mutual information also brush on this concept, where a return map is of $x(t)$ against $x(t+01.5)$ is shown which resembles the fractal and aperiodic characteristics observed in the Lorenz attractor.

2.2 Artificial Neural Networks

2.2.1 Literature Survey

A detailed and comprehensible explanation of the biological neuron model (or nerve cell) is contained in Chapter 48 of Campbell Biology [9]. The textbook chapter expounds the nerve cell, action potentials, and the brain in a comprehensive manner, as well as expanding on the biological and chemical occurences at the cellular level.

Seo's 2014 thesis *An Artificial Muscle Neuron* [10] provides a well-researched introductory section on the biological neuron, then briefly discusses its mathematical abstraction into the Artificial Neuron Model (ANM). Principal components and concepts pertaining to an ANM and how it models a biological nerve cell are discussed.

McCulloch and Pitts' article *A Logical Calculus of the Ideas Immanent in Nervous Activity* [11] is considered a classical text in the field, where although the mathematics is obscurely notated, it laid a foundation for which many built upon with its influential mathematical model of the neuron.

Krenker's *Introduction to the Artificial Neural Networks* is a found within a collation [12] and provides a more recent approach to describing the concept of an artificial neural network. It contains a comprehensive

introductory section on the artificial neuron model.

Bishop's *Neural Networks for Pattern Recognition* [13] is a textbook which explains the ideas pertaining to ANNs clearly and consistently, predominantly using linear algebra, calculus, and probability theory. Relevant chapters to this research detail the perceptron, the Multilayer Perceptron (MLP), and the RBFNN. Additionally, the book contains much information that could be applied and drawn upon for further research such as parameter optimization and network generalization.

Priddy and Keller's book *Artificial Neural Networks: An Introduction* [1] covers almost all of that previously mentioned. It discusses neurons and the ANM, different learning methods, and data normalization, as well as providing a multitude of emperical examples from different neural networks and their solutions to problems.

2.2.2 Description of Research

ANNs such as the RBFNN and MLP have their foundational conceptualization in the biological neuron model [12]. Neurons (or nerve cells) receive and transmit signals to and from other cells [9], [10], [12]. The transmitting neuron is referred to as the presynaptic cell whilst the nerve, muscle, or gland cell which receives the signal is termed the postsynaptic cell [9]. Key components of the biological neuron (or spiking neuron) model, which are relevant to its abstraction into an artificial neuron model (ANM), include the cell body (or soma), axon, and dendrites [1], [9], [10]. A nerve cell's dendrites are like tree roots extending from the soma, reaching out to collect electrochemical information from the surrounding area so to induce stimulation of the cell body [9]. Axons belonging to other neurons can also stimulate the nerve cell through their connected synaptic terminals [9]. Every neuron has some threshold of voltage (or excitation) which must be overcome in order to fire [11] i.e. send an action potential electrochemical signal [9], [10]. These sent signals travel from the soma, along the neuron's axon, which then in turn stimulate other cells [1], [9]. Through the human brain's approximately 10^{11} neurons [10], practically all experience, thought, logic, and emotion is processed through these communicating cells called neurons [9]. A diagram of the biologial neuron is depicted below:

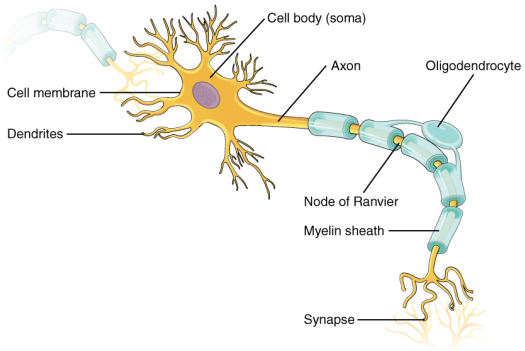


Figure 1: Parts of a Neuron. The major parts of the neuron are labeled on a multipolar neuron from the CNS ([14]).

An ANM is the derived functional model of the biological neuron [10], [12], and is the fundamental building block of any more complicated ANN [12]. Just as the biological nerve cell is the building block for our neural networks i.e. brain and nervous system. An artificial neuron receives an input from one or more sources, then produces an output [1], [10]. Each input into the neuron is weighted, then the sum of weighted inputs (and the neuron's bias) is passed into a transfer function which produces an output [1], [10], [12]. Some common transfer functions are sigmoid or step functions. In general, the output of an ANM is calculated as per below [1]:

$$y = f(\sum_{i=1}^{N} w_i x_i) \tag{1}$$

where y is the output, f is the activation function, x is the input vector of dimension N, and w is the vector of weights. The neuron's logical output provides information about the input through its output, and hence through adjusting the weights i.e. training the ANM, the artificial neuron can make sense of external inputs by providing a meaningful output. Validation and generalisation of ANMs are important once a neuron has been trained.

The ANM is the fundamental component of which the MLP comprises [1]. With a network implementation, however, multiple layers of neurons will exist [1], [15]. As with the ANM, training, validation, and generalisation must all occur in order to establish a plausible ANN [13], [15]. There are many different structures an ANN can take, with feed-forward being one such structure [12]. This is arguably the simplest structure and both the MLP and RBFNN is included in this category [13]. In this structure, there is no feed-back, information flows in one direction, from the input towards

the output.

A radial function has the characteristic that its response is dependent on its inputs distance from a central point [16]. From Charles' notes, a basis function is aptly described: '*A basis function is a shape (or continuous function) with the special mathematical property that a linear combination of the basis function can be used to represent any other continuous mathematical function*' ([16]). Such functions include fourier transforms which can fit any periodic function using a linear superposition of an infinite series of sinusoids [16]. The Gaussian function is an example of a radial basis function [13], [16]. Its response is dependent on its input's distance from its centre. A linear combination of Gaussian functions can be used to represent any other function [16]. The Gaussian function is of the following format:

$$y = ae^{-\frac{(x-c)^2}{2r^2}} \tag{2}$$

where a is the height of the function, c is the function's centre, x is its input, and r is its width.

The RBFNN is a type of feed-forward neural network which provides a smooth interpolation of a given input-output mapping [13]. Parameters for the fitting, given a number of input vectors (N) and their associated output targets, are the number of RBFNN centres (M) and the width parameter of each basis function (r) [13]. A matrix of Gaussian outputs which are dependent on each input vector's ($\mathbf{x}_i$) distance from each centre ($\mathbf{c}_j$) is computed [13], [16], normalized [16], then trained to best map the input vectors to their desired outputs through matrix inversion techniques [13]. These trained mappings can then be used for out of sample prediction.

3 Research Methodologies

Methods used and results sought after changed dynamically based on progress made, new perspectives or considerations, research direction advice received, and constant revisions and development of earlier work. For the sake of conciseness, only the most significant final research results and their respective methodologies of production are delineated.

An introduction to chaos follows a comprehensive succession of simulation based results that explain the essence of chaos then link back to research conducted by Kyla on chaotic analysis. Succeeding this, the method to be used for sampling the Lorenz attractor is determined and comments are made on SDIC and the shadowing lemma. RBFNN implementation is then delineated, showing multiple different approaches taken in predicting future Lorenz dynamical behaviour. These approaches include taking equidistant samples of a single variable's trajectory and predicting its successive points, as well as using more information from the Lorenz system for the neural network's input such as its full three-dimensional state and training to predict successive states.

3.1 Introduction to Chaos

Understanding the fundamentals of chaos is intrinsic to understanding the problem at hand. The one-dimensional iterative logistic map and its bifurcations are explored and explained, which help comprehension through emperical and observable chaotic concepts. The logistic map follows the following deterministic rule:

$$x_{n+1} = rx_n(1 - x_n) \tag{3}$$

where the initial condition $x_0 \in (0, 1)$ and the single parameter $r \in (0, 4)$. Depending on the parameter value chosen, it will be shown that the map can act as an attractor to a single point, periodically, or chaotically. The bifurcation diagram for the map will be produced and briefly discussed.

Results from Kyla's research were drawn upon and mentioned, which were used in deducing or affirming parameters of RBFNN parameters. Kyla's research results include the embedding dimension of the Lorenz attractor and the x variable's mutual information.

3.2 Sampling the Lorenz Attractor

Computers do not store numbers to infinite precision. Due to SDIC, following a specific chaotic orbit over any substantial period of time would require infinite preceision. Additionally, the Lorenz equations are time-continuous. In solving its differential equations computationally, we must discretize the system. The use of different time steps dt will produce discrepency between their predictions and the real orbit given the initial condition. The fact that time steps greater than 0 are used in solving the system creates error in the calculated orbit against the *"real"* trajectory, which become intrinsically significant due to their exponential divergence due to SDIC. Using differing computational methods in solving the equations will have the same effect and produce different solutions and hence diverge.

The 4^{th} order Runge-Kutta method was chosen to create a computational solution to the Lorenz differential equations. For its use in the RBFNN, a time step of 0.01 was used between samples. The method numerically solves the time-continuous differential equations as follows:

let $s(t) = s_t$ be the current state at a given time t i.e. $s_t = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

let h be the timestep for computationally solving the time-continuous differential equations, in our case $h = 0.01$

$$
\begin{aligned}
f(s_t) = \dot{s}(t) &= \frac{d}{dt}(s_t) = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix} \\
s_{t+dt} &= s_t + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \\
&\begin{cases} k_1 = f(s_t) \\ k_2 = f(s_t + \frac{h}{2}k_1) \\ k_3 = f(s_t + \frac{h}{2}k_2) \\ k_4 = f(s_t + h \cdot k_3) \end{cases}
\end{aligned} \tag{4}
$$

3.3 Input Data Normalization

Pre-processing of data is used in most signal processing applications. Though it is more of a neccissity for networks such as the MLP to normalize data for its activation functions, it is also advisable for other networks and even general scrutiny of data. Statistical normalization was chosen as the method of use. This follows the following rule [13]:

$$\hat{x}_i^n = \frac{x_i - \bar{x}_i}{\sigma_i} \tag{5}$$

where $\hat{x}_i^n$ is the normalized n^{th} input of the i^{th} variable, x_i^n is its unprocessed observed input, $\bar{x}_i$ is the i^{th} variable's mean, and σ_i is the i^{th} variable's standard deviation.

The mean and standard deviation were extracted from computationally produced Lorenz trajectories using the following [13]:

$$\begin{aligned} \bar{x}_i &= \frac{1}{N}\sum_{n=1}^{N} x_i^n \\ \sigma_i^2 &= \frac{1}{N-1}\sum_{n=1}^{N}(x_i^n - \bar{x}_i)^2 \end{aligned} \tag{6}$$

where N is the total number of taken samples.

3.4 The Radial Basis Function Neural Network

The RBFNN is a type of feed-forward neural network. A mapping from a n dimensional input vector $\mathbf{x}$ to a singular output y is depicted in the below diagraphic:

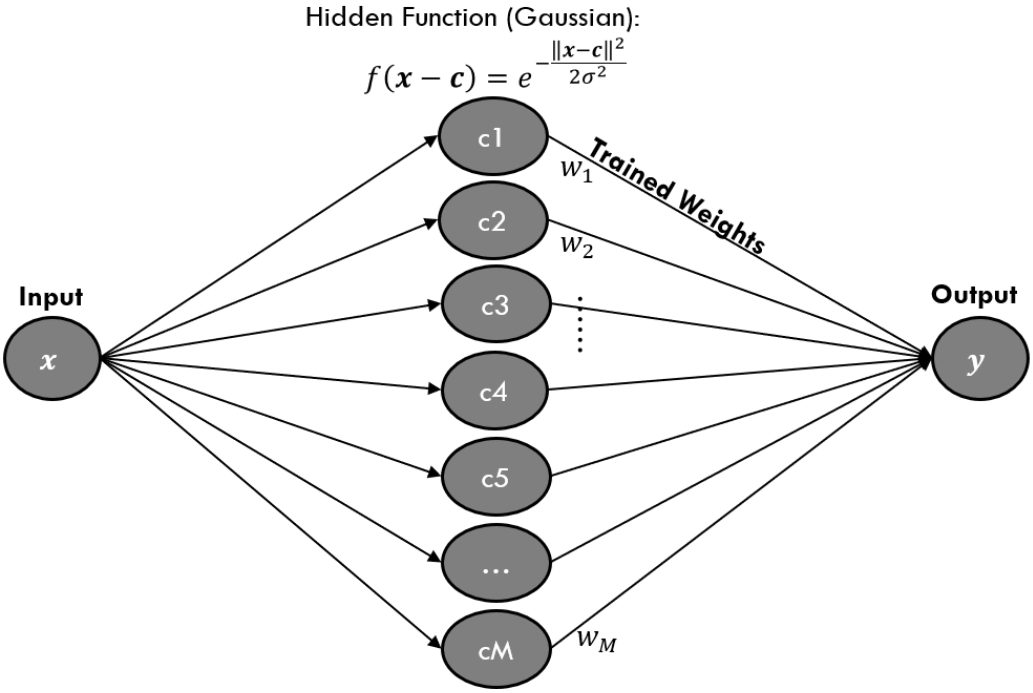


Figure 2: RBFNN Diagraphic

Radial Basis Function (RBF) parameters include its centre, and standard deviation (or width) [16]. An example of such a function is the Gaussian function which has the following shape: .

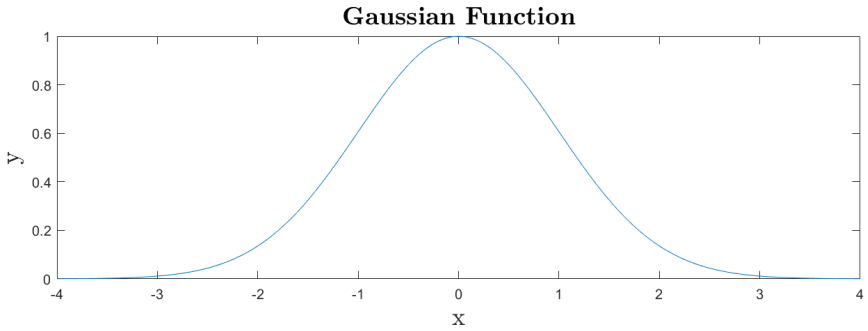


Figure 3: Gaussian Radial Function

The Gaussian function itself is of the following format [16]:

$$y = ae^{-\frac{(x-c)^2}{2r^2}} \tag{7}$$

where x is the input, c is the function's centre, and r is the function's width. For their use in implemented RBFNNs, centres are selected uniformally from within the training vectors.

A matrix of Gaussian kernels is formed by taking the Euclidean distance between each input vector and each centre and taking the Gaussian response [16]. The $\mathbf{\Phi}$ Gaussian matrix for training is determined using the following:

$$\mathbf{\Phi}_{ij} = e^{-\frac{\|\mathbf{x_i}-\mathbf{c_j}\|^2}{2r^2}} \tag{8}$$

where $\mathbf{x}_i$ is the i^{th} training input vector and $\mathbf{c}_j$ is the j^{th} centre.

Charles' notes [16] provide additional instruction to divide each row of the Gaussian matrix ($\mathbf{\Phi}$) by the sum of all the kernel outputs on that row. Such an action means that the summed outputs of all hidden functions for any input variable are equal to one [16]. The normalization action is notated below and was used in all implemented networks:

$$\mathbf{\Phi}_{ij}^{norm} = \frac{\mathbf{\Phi}_{ij}}{\sum_{j=1}^{M} \mathbf{\Phi}_{ij}} \tag{9}$$

Note that the pseudo inverse of a matrix $\mathbf{A}$ is defined [13], [16]:

$$\mathbf{A}^{+} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \tag{10}$$

In the case of the neural network, we desire the mapping [13]:

$$\mathbf{\Phi W} = \mathbf{T} \tag{11}$$

where $\mathbf{\Phi}$ is the matrix of normalized Gaussian outputs, $\mathbf{W}$ is the vector (or matrix) of weights, and $\mathbf{T}$ is the vector (or matrix) of targets (or desired outputs). The weights can be determined by manipulating the equations [13]:

$$\begin{aligned} \mathbf{\Phi}^T\mathbf{\Phi W} &= \mathbf{\Phi}^T\mathbf{T} \\ \mathbf{W} &= \mathbf{\Phi}^{+}\mathbf{T} \end{aligned} \tag{12}$$

which guarantees optimal least-squares mapping between the inputs and outputs using the Gaussian matrix [13].

Once weights have been determined (i.e. the network has been trained), a RBFNN prediction is determined by its input. The Gaussian matrix $\mathbf{\Phi}$ is recalculated to be a single row vector using equation:

$$\mathbf{\Phi}_j = e^{-\frac{\|\mathbf{x}-\mathbf{c_j}\|^2}{2r^2}} \tag{13}$$

where $\mathbf{x}$ is the provided input. The matrix is normalized, then multiplied by the weights to produce the networks output.

3.4.1 Single Variable Prediction

One cannot always measure all the components which give the state of the system [5]. An example of this may be the Electroencephalogram (EEG) signal, where electrodes are used to record electrical activity of the brain. Obviously, the brain's electrical activity is extremely complex and the electrodes hence only capture some information of what is actually occurring. To simulate this scenario, the data of a single variable's trajectory was used in prediction. The input to the RBFNN used that variable's delay coordinates. The figure below helps depict what is meant by this:

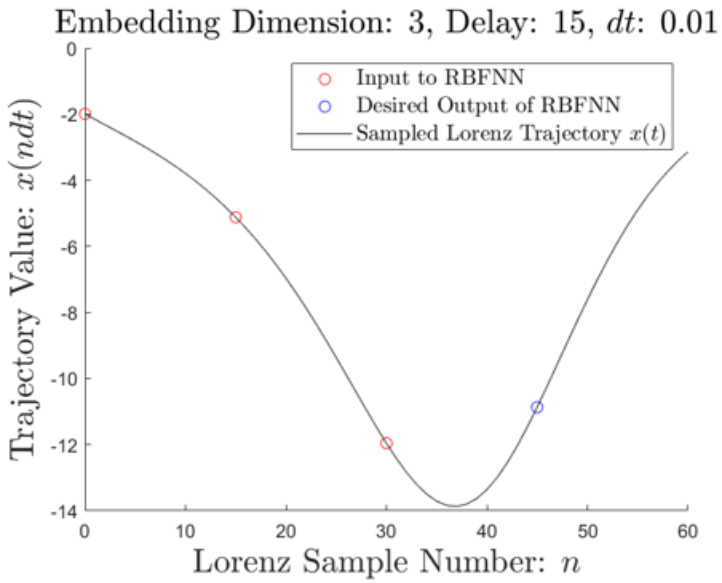


Figure 4: RBFNN Embedding Dimension Depiction

The black plotted line represents a computationally solved Lorenz x variable trajectory with 0.01 units of time between samples. Using an embedding delay of 15 means that every 15^{th} sample is used in the input vector. Combined with the embedding dimension of 3, the above figure depicts an example of an tapped delay line input with the Lorenz samples circled in red. The input is hence the vector $[x_0, x_{15}, x_{30}]$ of the computational sampled solution. The desired output of the input is x_{45}, as circled in blue. Training error is observed and used as a comparitive measure of how well the RBFNN fits the data. Error used was the absolute error between the trained predicted values and their associated desired outputs.

RBFNN training occurs on thousands of vectors such as the example above using a lengthy computed solution's trajectory. Once trained, a random starting tapped delay line (such as the one depicted and circled in red) of the Lorenz attractor is passed to the RBFNN. The prediction of the successive point is appended to the input vector and the first dimension of the input vector (i.e. x_0) is dropped. A new input vector is hence created and the RBFNN predicts off its own previous predictions.

For comparative purposes when varying embedding dimensions or delay, a proportionally consistent Gaussian radius was used which was dependent on the maximum Euclidean distance between any two centres (d_{ME}). The radius was shared between all centre's hidden functions and was determined by the following:

$$r = \frac{2d_{ME}}{\sqrt{2M}} \tag{14}$$

3.4.2 Prediction from Three-Dimensional Lorenz State

The radial basis function neural network is a versatile structure which can be adjusted by its programmer to map as many inputs to as many outputs as desired. For this implementation of the network, the information passed to the network was a Lorenz state, with its successive state after time dt being the desired output. Error for the network would now be measured by summing the absolute values of each predicted dimension's error:

$$E = \sum_{i=1}^{3} |\hat{x}_i - x_i| \tag{15}$$

where E is the error of a prediction, $\hat{x}_i$ denotes the predicted output for the i^{th} predicted variable, and x_i is the desired i^{th} output. The desired and

predicted outputs are intrinsically three-dimensional as are the Lorenz states.

Once the RBFNN predicts a state, the predicted state can then be passed in as its input again. The network can perform this recursive action indefinitely to create a prediction as far as desired.

3.4.3 Robustness of RBFNN Predictions

Once trained, a method of testing the network's robustness was to observe its prediction performance from random starting trajectories on the Lorenz attractor. Ideally, the network would predict out of sample for similar temporal distances regardless of where on the attractor it started from.

4 Research Results

4.1 Introduction to Chaos

4.1.1 The Logistic Map

The iterative function known as the logistic map is determined by the following rule:

$$x_{n+1} = rx_n(1 - x_n),\ x_0 \in (0, 1) \tag{16}$$

The logistic map portrays varying attracting behaviour with values of its parameter (r) between 0 and 4. For values ranges [17]:

- $r \in (0, 1)$, single attracting point at zero
- $r \in (1, 3)$, single attracting point at $\frac{r-1}{r}$
- $r \in 3, 3.44949$, attracts to period 2
- $r \in 3.44949, 3.54409$, attracts to period 4
- As r increases from 3.54409, the attracting period doubles i.e. to 8, then 16, 32, 64, ..., until at 3.56995 where the behaviour is aperiodic.

So to picture what is occuring here, a bifurcation diagram is depicted below:

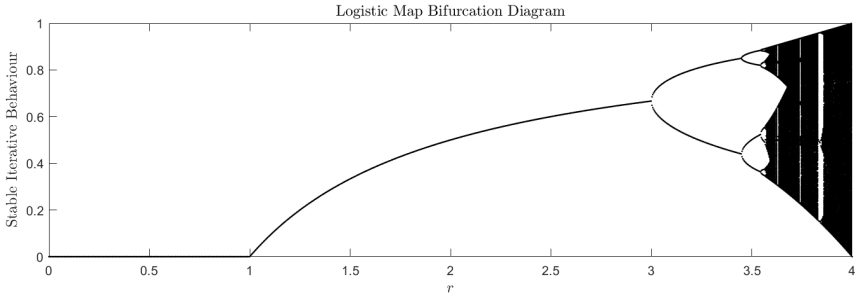


Figure 5: Bifurcation Diagram of the Logistic Equation

The bifurcation diagram depicts the attracting behaviour of the logistic map depending on its parameter value r. It was produced by running the logistic map for a 10000 iteration warmup from $x_0 = 0.5$, then vertically scatter plotting the following consecutive 1000 values in the orbit above its associated value of r. This process was implemented for one thousand

unique values of r equispaced between 0 and 4.

A 50 iteration with $x_0 = 0.5$ is shown below for 3 different values of r. Compare the observed behaviour to the bifurcation diagram:

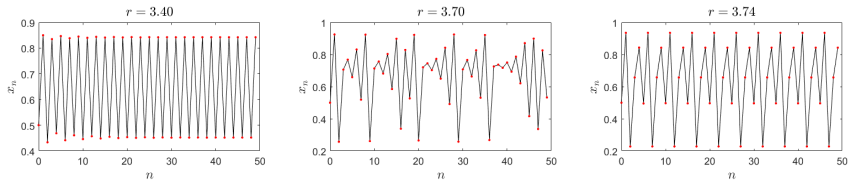


Figure 6: Varying Attracting Behaviour from the Logistic Map

It is observable that the bifurcation diagram and the above 3 plots are quite related. With the parameter value $r = 3.4$, the iterated values are attracted to an orbit of period two, consistantly alternating between the approximate values 0.45 and 0.85. In the bifurcation diagram, at $r = 3.4$ along the bottom axis, the corresponding values reinforce what is observed in the iterative plotted orbit. At $r = 3.7$, there appears to be no observable pattern in the plotted orbit. This is because there isn't. In the bifurcation diagram, the thousand scattered points appear to form a vertical line, roughly between 0.25 and 0.9, above the value $r = 3.7$. Finally, with the parameter $r = 3.74$, an attracting orbit of period 5 is observed. This periodic window is clearly observable in the bifurcation diagram also. A longer chaotic orbit of the logistic equation with $r = 3.7$ starting at $x_0 = 0.5$ is depicted below:

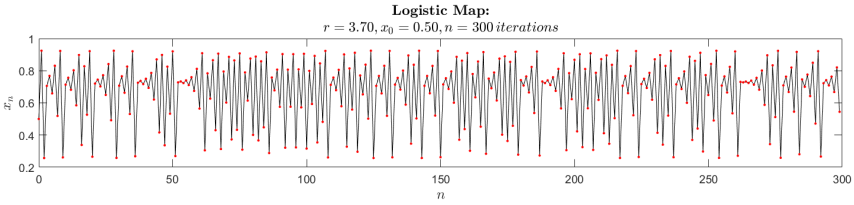


Figure 7: Chaotic Behaviour from the Logistic Map

Chaotic orbits must have sensitive dependence on initial conditions. Simply stated: '*for any initial condition x_0, other initial conditions very near to it eventually end up far away*' ([2]). This is depicted in the figure below, where two initial conditions $x_0 = 0.3$ and $x_0 = 0.3001$ had their orbits plotted together. One would expect, given the two initial conditions are so close, that they would follow relatively identical orbits; and they do, for a short time. By 20 iterations, however, the orbits bear no resemblence to eachother; there would be no reason to think they would have started so

close together. This poses a problem for prediction, regardless of whether you knew the deterministic rule that the dynamical system follows. You would need infinitely accurate measurements to predict the system long term. In the real world, measurements this accuracy are inherently meaningless due to noise, and measuring to this degree of accuracy is often impossible.

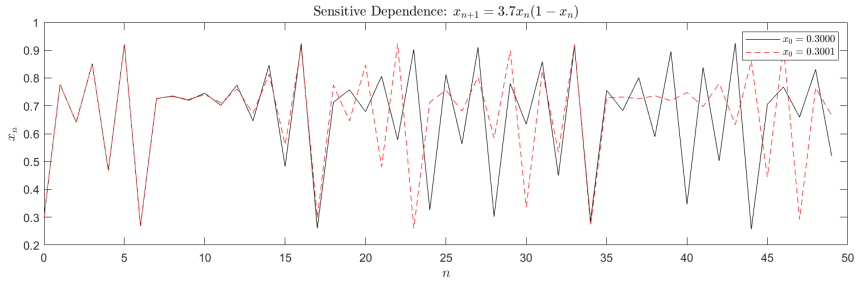


Figure 8: Logistic Sensitive Dependence on Initial Conditions

The orbit of the logistic map iterative equations with parameter $r = 3.7$ satisfies all requirements of chaos. Obviously, it was produced by a deterministic rule: $x_{n+1} = 3.7x_n(1 - x_n)$. The orbit is bounded, values produced in the orbit will never exceed 1 or become negative (in fact, for $r = 3.7$, they vary between roughly 0.25 and 0.9). The orbit is aperiodic, and hence never repeats itself or, by extension, it never returns to a value that was previously observed in the orbit. Additionally, the orbits have sensitive dependence on initial conditions as depicted earlier.

4.1.2 Kyla's results

Results relevant to this research included the embedding dimension of the Lorenz attractor, as well as the x variable's mutual information. Using the correlation integral applied to the Lorenz system, it was found that the embedding dimension of the Lorenz attractor was 3. This is found by analysing the linear region of the plotted lines and finding the dimension at which the lines become saturated i.e. the gradient of the linear region no longer changes.

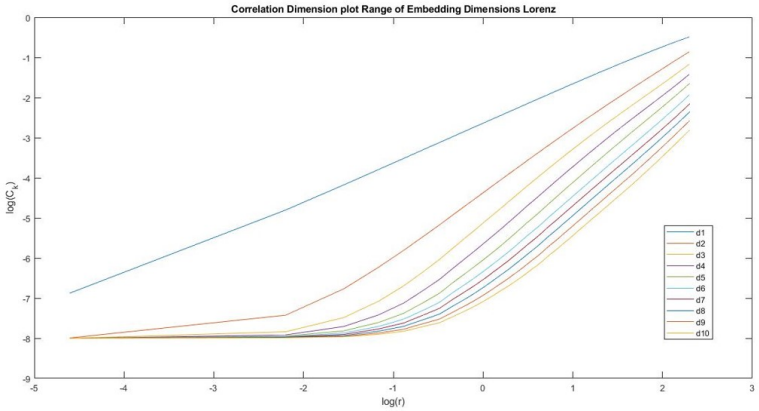


Figure 9: Correlation Dimension of Lorenz Attractor – from Kyla's Research

Kyla's research results showed that the x variable's first minima in the mutual information was at approximately 0.15 units of time delay. The below figure shows the minima in the curve, as well as the delay coordinate plot of $x(t)$ against $x(t + 0.15)$:

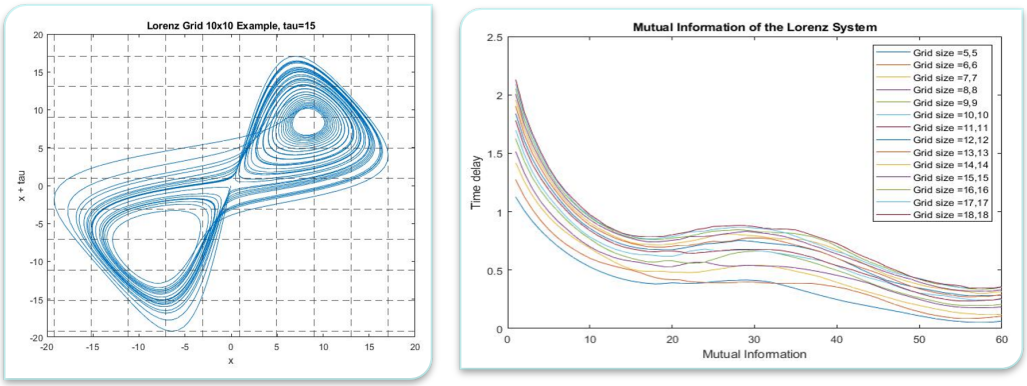


Figure 10: x Variable Lorenz Mutual Information – from Kyla's Research

Emperically, it is observable that the delay coordinate plot corresponds to chaotic behaviour similar to that observed in the Lorenz attractor. These results formed the hypothesis that when using a tapped delay line of the Lorenz x variable, a delay of 0.15 units of time and embedding dimension of 3 would be optimal for prediction.

4.2 Sampling the Lorenz Attractor

The question arisen is that if different methods produce significantly different orbits, who is to determine which orbit is *"correct"*. Fortunately, the shadowing lemma proves that a calculated orbit is arbitrarily close to some other alternative real orbit, making such a computationally produced orbit equally valuable and viable. This is a rigorously proved mathematical lemma [5]. The lemma states:

> *Although a numerically computed chaotic trajectory diverges exponentially from the true trajectory with the same initial coordinates, there exists a true (i.e. errorless) trajectory with a slightly different initial condition that stays near (shadows) the numerically computed one. Therefore, the fractal structure of chaotic trajectories seen in computer maps is real* ([5]).

As earlier stated, the Runge-Kutta 4^{th} order iterative method was chosen for the temperal discretization giving a numerical approximate solution to the Lorenz differential equations. Reasons for this is that it is a more accurate numerical approximation than Euler's method. Even though the shadowing lemma disregards the need for *"better"* numerical approximation due to divergence from a true trajectory and the shadowing of another, the Runge-Kutta 4^{th} method was chosen. This did result in the ability to increase the time-step to 0.05 if desired, without the numerically computed orbit diverging; a quality not shared with Euler's method on the Lorenz equations. Time-steps of this size were not used within RBFNN predictions, however, were used in finding the mean and standard deviation of all Lorenz variable's trajectories. A larger range of plausible times between samples could result in more room for parameter adjustment in determining best RBFNN parameters, however, this has not been explored in this research.

Demonstrating the rapid divergence of equivalently valid trajectories is shown in the below two figures:

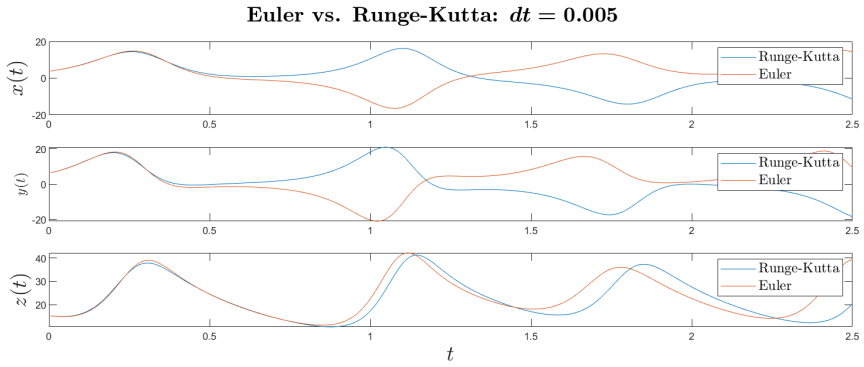


Figure 11: Euler's Method vs. Runge-Kutta Divergence

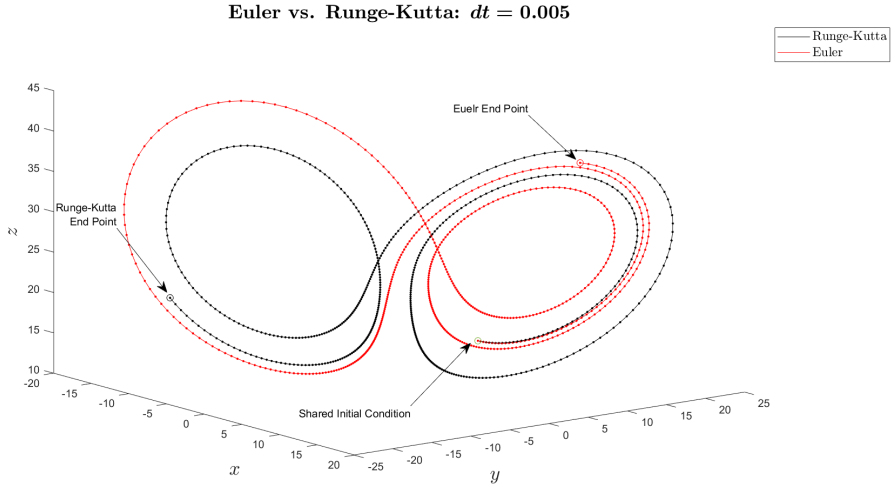


Figure 12: Euler's Method vs. Runge-Kutta 3D Divergence

Observably, the two equally valid solutions to Lorenz trajectories diverge almost instantaneously (after aproximately 0.5 units of time). This begs the question of whether temporal predicted distance following a Runge-Kutta computed orbit is a good measure of success for RBFNN performance.

4.3 Input Data Normalization

By computationally solving the Lorenz equations with timestep 0.05 and recording 50 thousand samples, the mean and standard deviation over time were computed and extracted. A simulation results are plotted below:

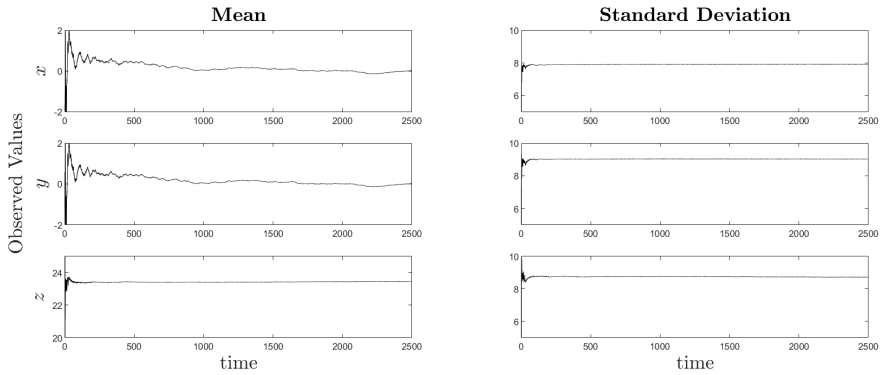


Figure 13: Lorenz Variable's Mean and Standard Deviation Over Time

The outcome of the above simulation resulted in the following:

$$\text{Mean: } \mu = \begin{bmatrix} 0.0212 \\ 0.0217 \\ 23.4414 \end{bmatrix} \quad \text{Standard Deviation: } \sigma = \begin{bmatrix} 7.9092 \\ 9.0305 \\ 8.7197 \end{bmatrix}$$

After running longer simulations and observing the found means and standard deviations, for use in RBFNNs implemented it was decided the following values would be used to statistically normalize the input:

$$\mu = \begin{bmatrix} 0 \\ 0 \\ 23.49 \end{bmatrix} \quad \sigma = \begin{bmatrix} 7.92 \\ 9.02 \\ 8.68 \end{bmatrix}$$

4.4 Radial Basis Function Neural Networks

4.4.1 Single Variable Prediction

x, y, and z: Using a tapped delay line as input, each variable was independently used for the RBFNN. In the case depicted below, an embedding dimension of 5 and delay of 1 was used for prediction. 250 centres were employed. Once trained, the RBFNN was tested from varying random starting orbits on the attractor and the below figure depicts an example of predictave performance observed:

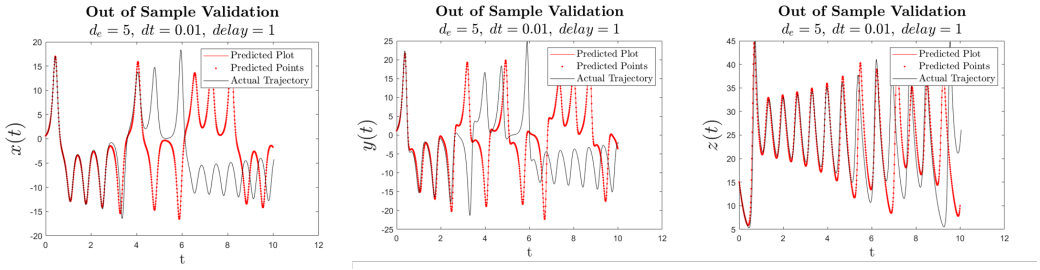


Figure 14: Predictive Distance of Individual Lorenz Variables

Both variables x and z showed potential for being predicted individually by the RBFNN. The dynamics of y appeard to be less predictable from its own trajectory, and perhaps this shouldn't be surprising. Remembering the governing differential equations, y is the only variable whose gradient is independent of its own state.

In-Training Measurements of Success: Training the RBFNN has proven successful as per the above outcomes. Whilst training the neural network, the error between each prediction and target from the mapping between input and outputs was recorded. The network was provided 250 centres, used a Gaussian radius of 0.4, embedding dimension 3, and delay 1. As observable in the below figure, the cumulative absolute error increases approximately linearly with respect to the number of training vectors:

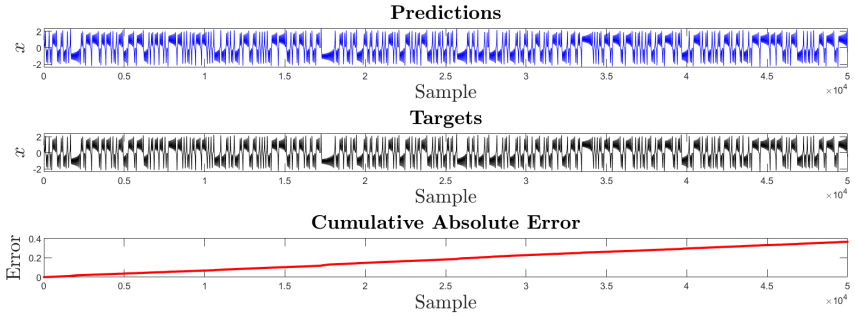


Figure 15: Cumulative Absolute Error within Training

The above figure shows the cumulative absolute error over all 50000 training vectors to equal to 0.3646, resulting in a mean error of $7.2918 \times 10^{-}6$. Regardless of this tiny error which suggests exceptional within sample training of predictions, the predicted trajectory diverges from the computed one after only approximately 2 units of time as per below:

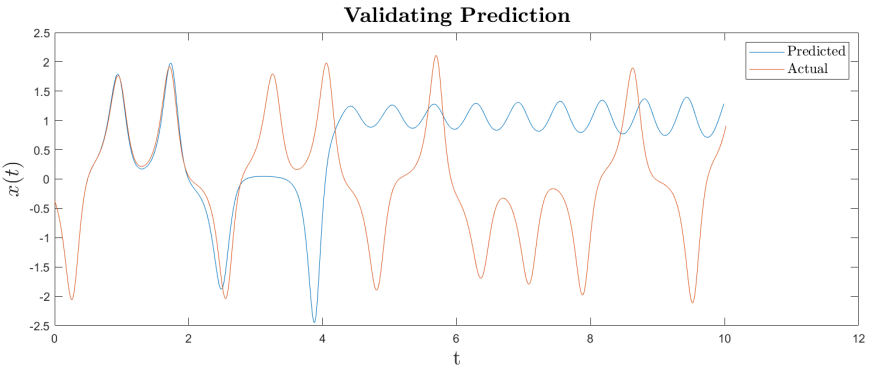


Figure 16: x Variable Prediction Example

Embedding Delay: Using an embedding dimension of 3, 250 centres, and a proportionately consistent Gaussian radius of $r = \frac{2d_{ME}}{\sqrt{2M}}$, embedding delay was adjusted using the x variable's tapped delay coordinates. At delay of 1, the predictions followed the actual orbit for almost 3 units of time, equating to 300 predictions. Increasing the delay to 5 resulted in further temporal prediction of approximately 5 units of time (100 predictions but almost twice as far temporally). Increasing the delay to 10 then to 15 both resulted in further temporal prediction, where at delay 15, the Lorenz x trajectory was followed by the predicted one for approximately 8 units of time. Increasing the delay further saw a turn-around in performance. At delay 20, the temporal prediction distance fell back to around 6 time units, and at delay of 30 the predicted orbit diverged completely from the computed one showing that as predictions are attempted further into the future, the prediction targets become increasingly unpredictable.

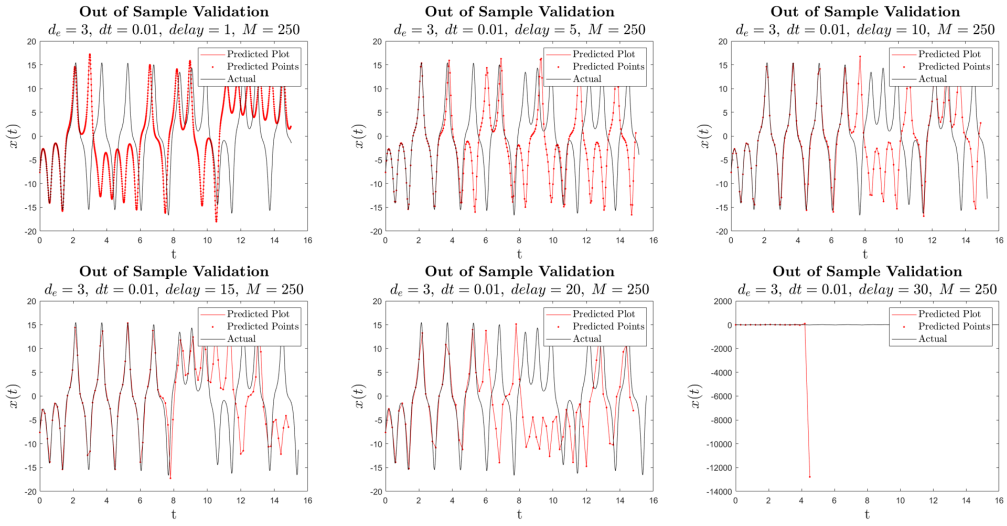


Figure 17: x Variable Prediction with Differing Delay

Embedding Dimension: The embedding dimension was adjusted from 3 to both 2 and 5 and difference in predictive ability emperically noted. Perhaps surprisingly, increasing the embedding dimension of the x delay coordinates did not appear to improve performance. Decreasing the delay from 3 to 2, however, did significantly decrease the RBFNNs ability to predict the x variable's trajectory.

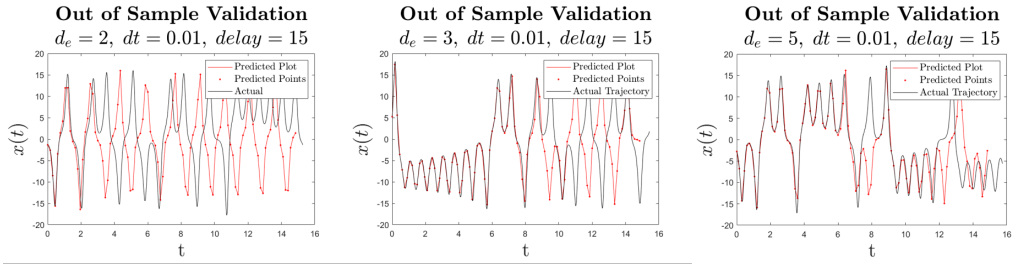


Figure 18: Embedding Dimension for x Variable Trajectory at Delay of 15

Number of Centres: Experimentation with the number of centres in the network and how it effects predictive performance was undertaken. Using an embedding dimension of 5 and delay of 1, the number of centres was increased so to observe difference in out of training sample performance. The Gaussian radius was kept proportionally constant at $\frac{2d_{ME}}{\sqrt{2M}}$. More centres consistently resulted in further predictive ability.

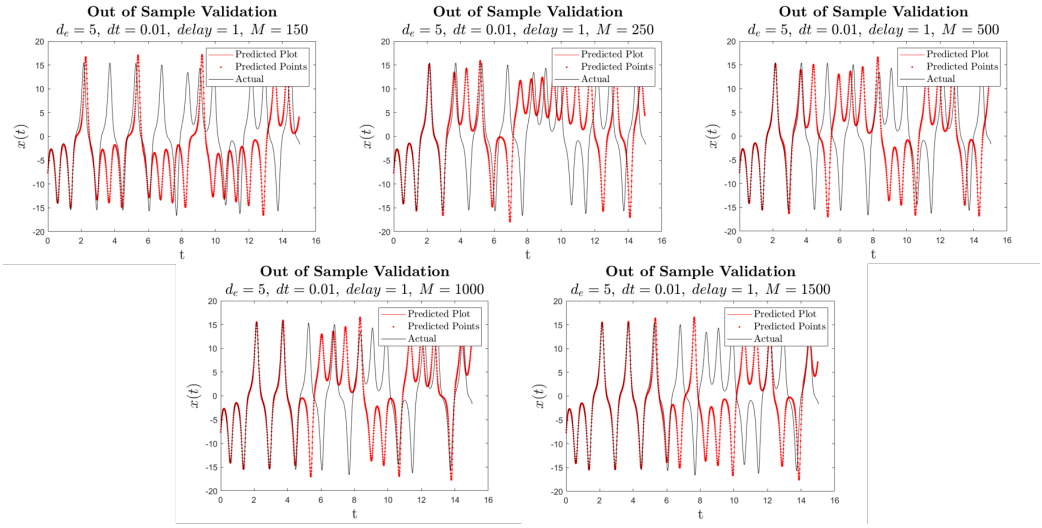


Figure 19: x Variable Prediction with Differing Number of Centres (M)

Lorenz Reconstruction: With the neural networks versatility to map any input dimension to any output dimension, an attempt to predict all Lorenz variable's from the sole x variables tapped delay line was made. It was found, that at embedding delay of 15, predictions were able to be made with fair accuracy. The following results were produced by a RBFNN with 250 centres and uniform Gaussian radius of $r = 0.4$ on statistcally normalized data input. The network was trained on 20000 vectors. Note that in this case of the following figure, the predicted and sampled trajectories have been plotted in their normalized state:

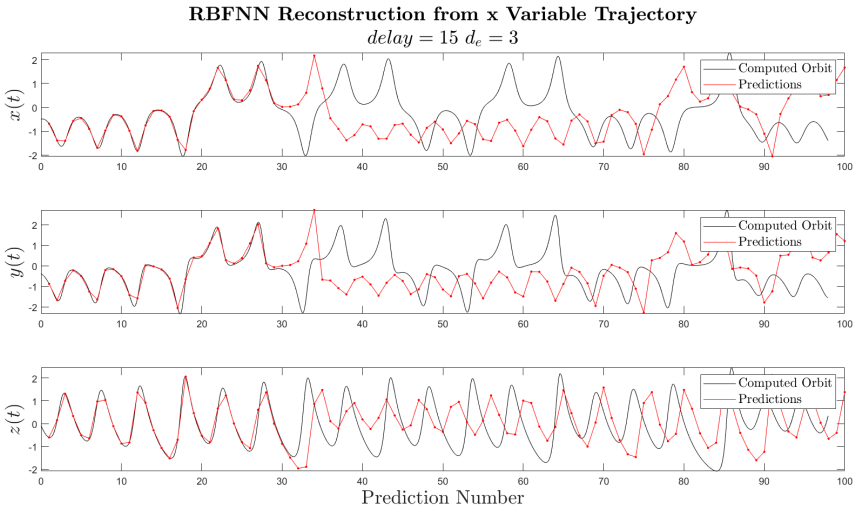


Figure 20: All Variable Predictions using Delay Coordinates of x

The real trajectory was followed by the predicted one for almost 30 predictions (4.5 units of time) from a single vector of 3 x delay coordinates with time delay 0.15.

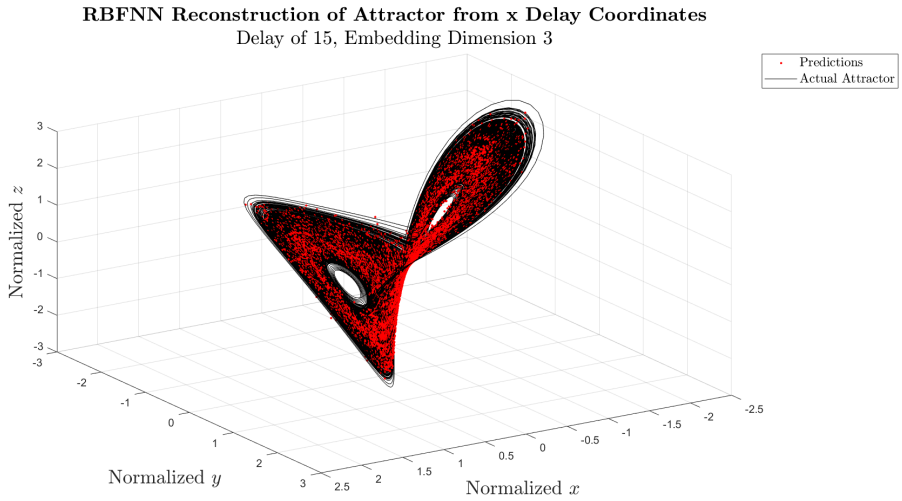


Figure 21: RBFNN Reconstruction of Lorenz Attractor using Delay Coordinates of x

4.4.2 Prediction from Three-Dimensional Lorenz State

Training of the network proved that the RBFNN can fit the Lorenz attractor exceptionally well. In the below figure, an example of training is depicted

using 20000 training vectors (i.e. Lorenz states) and 500 centres. The centres are circled in red on the three-dimensional plot of the predictions. An accumulated absolute error of just

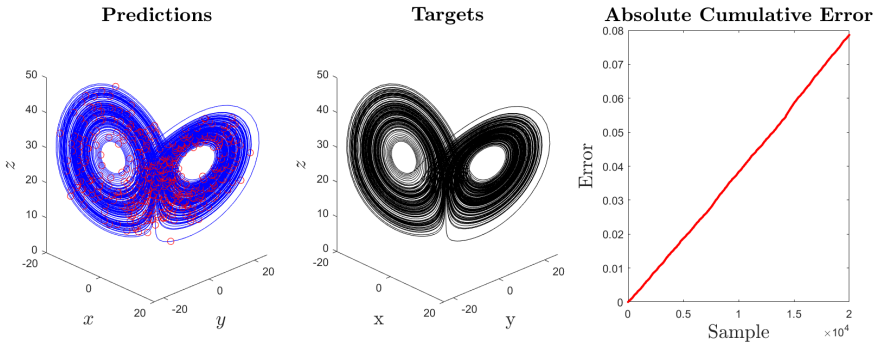


Figure 22: Lorenz Attractor in Phase Space from Identical Initial Condition

The following figure was produced by a RBFNN taking a three-dimensional Lorenz input state to predict successive states. This example used a delay of 1 and dt of 0.01 with 250 centres and a specified Gaussian radius of 4.

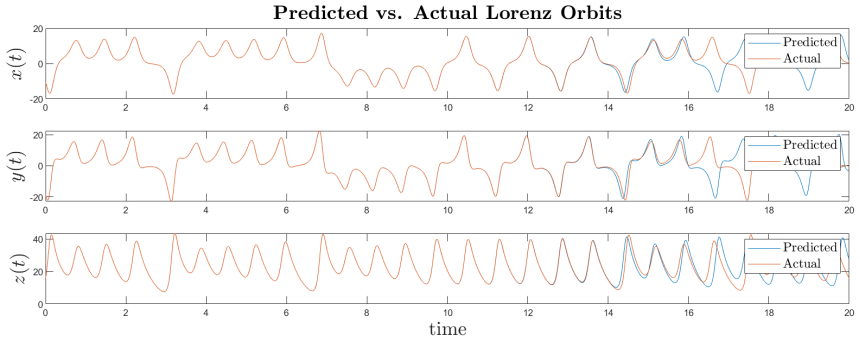


Figure 23: Lorenz Three-Dimensional Trajectory Prediction from a Single Random Starting State

The length of accurate temporal prediction is high. The RBFNN's predictions follow the actual computed orbit for almost 15 units of time, equating to 1500 predictions of successive state. A plot depicting the orbits diverging in three dimensions is shown below, where 500 consecutive actual orbit samples and predicted trajectories are plotted in state space adjacently, starting from the corresponding 1000^{th} prediction from the figure above:

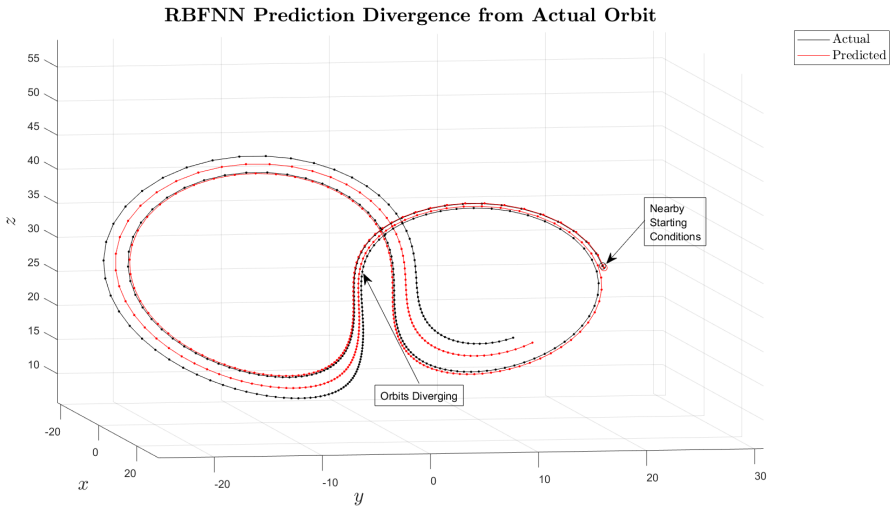


Figure 24: Visualization of Diverging RBFNN Predicted and Computationally Solved Trajectories

Regardless of diverging from the actual computed solution, the RBFNN predicted orbits appear to have captured the high-level dynamical behaviour of Lorenz variable trajectories. This statement is further reinforced by plotting the predictions in three-dimensional state space. The figure shown below is a RBFNN reconstruction of the Lorenz attractor of 100000 states:

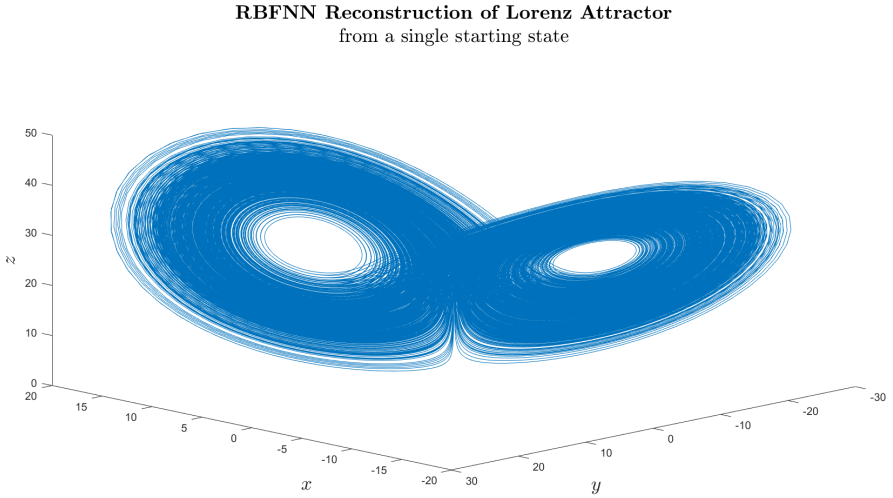


Figure 25: RBFNN Reconstruction of Lorenz Attractor from a Single Starting State

The reconstruction by the neural network could easily be mistakenly

identified as one produced authentically through solving the Lorenz differential equations. From the same initial condition that was provided to the RBFNN, the actual computed trajectory in phase space is depicted below:

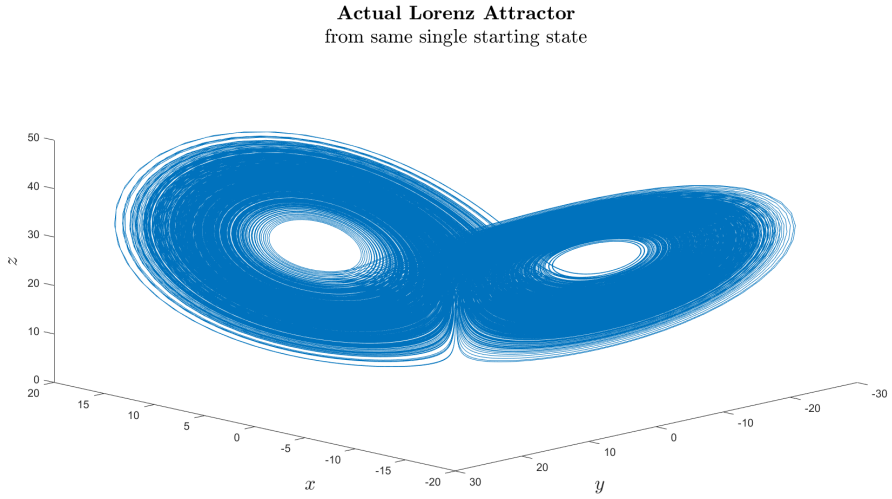


Figure 26: Lorenz Attractor in Phase Space from Identical Initial Condition

Both the real and reconstructed attractors are visually comparable, demonstrating that the neural network really has captured the dynamics of a Lorenz chaotic attractor exceptionally well. RBFNNs are intrinsically deterministic and with being trained to predict Lorenz states, appears to have captured the aperiodicity and boundedness of orbits. To demonstrate that periodicity is not occuring, the final RBFNN output after 100000 predictions was circled in red and has been zoomed in on below:

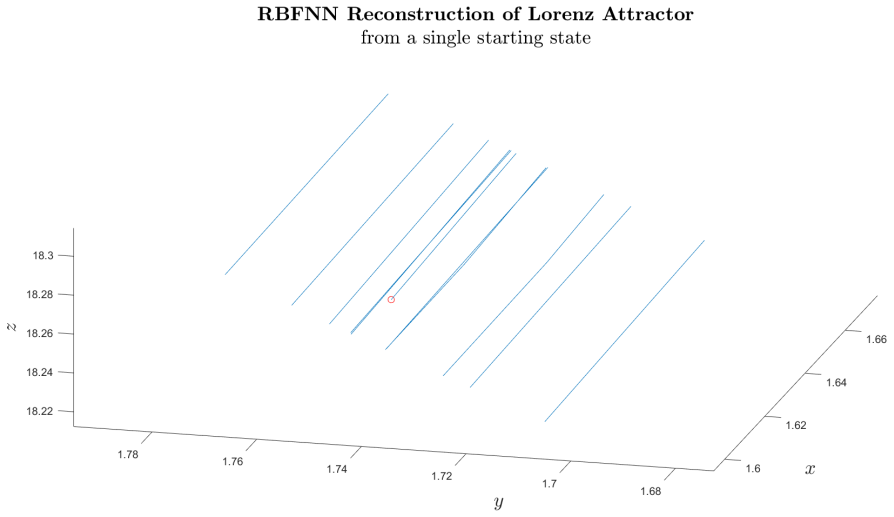


Figure 27: Apparent Aperiodicity of the RBFNN Reconstruction of a Lorenz Attractor

The fact that the final point is not one which has previously been predicted shows that a periodic limit cycle has not been converged upon by the network after 100000 predictions.

5 Discussion

5.1 Conducted Research

From the research conducted, it is clear that the RBFNN approximates and predicts Lorenz chaotic dynamic behaviour exceptionally well. Chaotic dynamics, although notoriably random, appears predictable in the short term as shown where a RBFNN is trained to predict the Runge-Kutta 4^{th} order solution of the differential equations. Additionally, it was shown that using an embedding delay of 15 and dimension 3, the x variable's trajectory contained information that could reconstruct the three-dimensional trajectories of an entire Lorenz attractor using RBFNN mappings.

Using a three-dimensional Lorenz state as input, the orbit of almost 15 units of time could be followed by the RBFNN. As earlier mentioned, however, different equally valid solutions (such as that produced by Euler's method and Runge-Kutta) diverged over less than half a time unit. Could this imply that the networks implemented have been overfitted to a single method of computational solution?

5.2 Further Research Potential

This research scarcely scratches the surface of neural networks ability to predict chaotic time series. Much further research within the field could be undertaken and some ideas for such endeavours have been delineated.

5.2.1 Parameter Optimization

Both Lorenz sampling methods and the radial basis function neural network have a number of parameters which can be adjusted including the computational solution method used, time between samples (dt), number of centres (M), number of training vectors (N), Gaussian radius (r), and embedding delay d_e to name a few. Not only are these parameters adjustable, but changing one individually has influence on the what is optimal or even viable for other parameters. Many ideal parameter values are interdependent. Within the research conducted, ad hoc methods were predominantly used heuristically to find parameter values that appeared to work well.

The Gaussian radii used in the hidden functions of the radial basis function neural network is arguably the most important parameter. As

mentioned above, ad hoc adjustment and emperical comparison of results was often employed as the method for determining parameter values within this research. For comparitive purposes of some outputs, the radius was chosen to be consistently dependent on the maximum Euclidean distance between centres (d_{ME}) and the number of centres used (M). This provided some sense of conviction that outputs were being fairly compared, however, is this actually the case? One of the examples where a consistent radius across centres was applied is where embedding delay of the x variable was explored. Maximum distance between centres (d_{ME}) would change due to embedding delay (d_e); and hence the radius used in the Gaussians would change proportionally. Whether or not this was a fair comparative technique is at question. Ideally, each comparison of delay should be made using optimal corresponding parameters for best out of sample performance. Bishop mentions the option of giving each basis function its own width σ_j whose value can be determined during training [13].

Temporally equidistantly spaced centres extracted from the training sample were used in RBFNN implementation throughout this research. These provide no guarantee of being well-placed or effective centres, however, with a higher quantity the likelihood is increased. Other techniques for determining centres could be to create or seclect them randomly, or to manually choose suspected good centres. No research within this report considered centre selection, however, speculation into whether the Lorenz attractor's fixed points could be good centres, or whether certain spaces require more or fewer centres for accurate prediction could be an interesting consideration.

5.2.2 Training Trajectories

The training of networks implemented occured using a Lorenz trajectory of user determined length from an established point on the attractor. All training and validation orbits were computed using Runge-Kutta 4^{th} order solution methods. These orbits, depending on length, capture much of the dynamical behaviour, however, miss areas very close to the two fixed points in the centres of the three-dimensional orbits. Exploration of multiple user determined training trajectories such as those starting close to, and diverging from, the fixed points may provide additional information to the neural network necessary for higher completeness of captured Lorenz dynamics.

The question of whether the network may have been overtrained to fit the

Runge-Kutta 4^{th} order solution of a trajectory arises. Training on multiple trajectories produced using differing solution methods may provide higher generalization ability to the neural network. After all, differing trajectories due to solution method used from equivalent initial conditions are both equally valid as proven by the shadowing lemma. Additionally, comparing predicted orbits to a single method of trajectory computation could be problematic. Perhaps for validation of the network, the predictions could be compared to multiple trajectories from comparable initial conditions rather than just a single computed orbit. Ultimately, what is most important, is that the high-level dynamics of the system is captured. The use of how far a neural network predicted orbit can follow a computed one may be an unwise measure of performance.

Finally, training of the neural networks within this research was undertaken on noiseless computationally solved Lorenz signals. An introduction of noise may change the performance of the RBFNNs predictive ability. A better generalization of the network may ensue.

5.2.3 Differing Neural Networks

The RBFNN was used within this research for the prediction of Lorenz trajectories and was found to perform extraordinarilly well. A quantitative and qualitative comparison between its performance and that of other neural networks, such as the MLP or a recurrent network such as a Nonlinear Autoregressive Exogenous Model (NARX), could form the foundation for futher research and provide further insight into optimal methods for predicting nonlinear time-series.

5.2.4 Alternative Chaotic Attractors

The RBFNN proved to perform exceptionally well in predicting Lorenz dynamics, however, is similar predictave ability universal between other chaotic systems? Dynamical systems such as the Rösler attractor, logistic map, and Henon map also display chaotic behaviour. The use of a RBFNN or other networks on these systems provide additional further research options.

5.2.5 Application to Real Observed Signals

Real signals such as EEG appear chaotic in nature. Using EEG signal orbits and using a RBFNN to predict their future dynamics would provide scope for much interesting research.

6 Conclusion

The radial basis function neural network proved to be an excellent method for function approximation and prediction of the future dynamical behaviour of chaotic attractors. Long-term behaviour of a chaotic orbits could not be predicted accurately to follow a specific computed orbit, which is unsurprising due to the intrinsic chaotic characteristic of sensitive dependence of initial conditions (or the butterfly effect). No matter how well-trained or resilient a predition method is, application to chaotic trajectories will always diverge exponentially.

Not only does divergence occur between very similar trajectories from slight difference in initial conditions, it was also observed that equally valid trajectories solved using differing computational methods diverged almost instantaneously from eachother, regardless of the fact that they started from the exact same initial condition. This leads to the concern that temporal length of prediction matching a computed orbit may not be a good method of network validation. Instead, the observation of aperiodicity and emperical continuation of observed high-level dynamics may be a preferable indicator of successful neural network chaotic prediction.

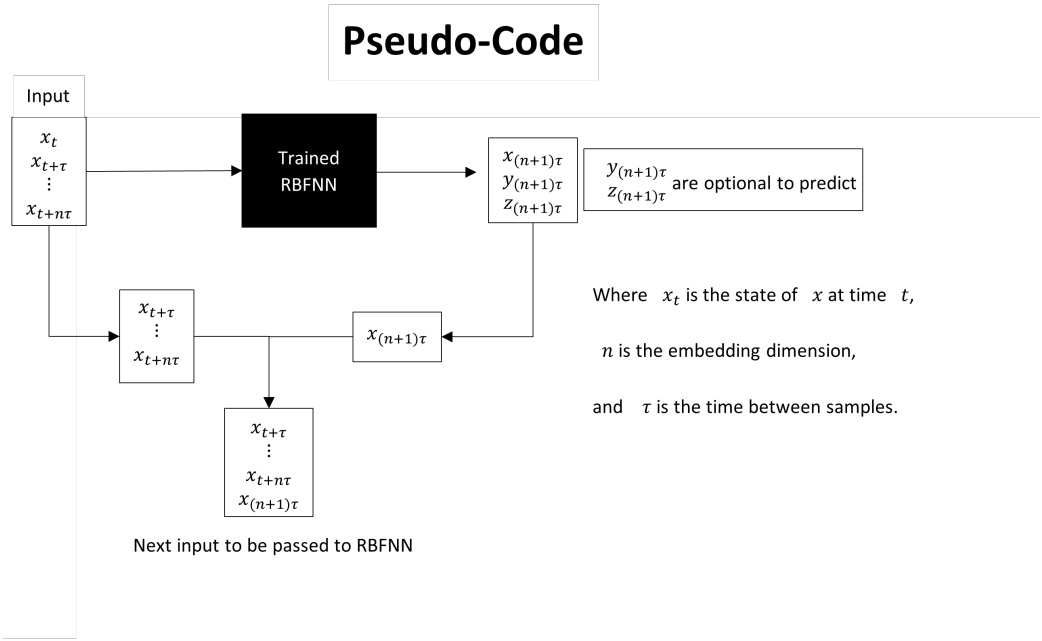
The first network implemented involved prediction from a single variable, namely, the x variable of the Lorenz system using a tapped delay line. Prediction of its own trajectory as well as the dynamics and position of other variable's trajectories, were captured by the neural network using only the x variable's delay coordinates in embedding dimension 3 and with a delay of 0.15 units of time.

Finally, prediction of successive states from a current state using a radial basis function neural network performed exceptionally. The network was able to follow the flow of a computed trajectory for almost 15 units of a time from a single starting state. Reconstruction of the attractor in phase space from using the network's predictions visually resembled the actual attractor almost exactly. Additionally, the neural network appeared to also capture the aperiodicity, a characteristic shared with that of a real chaotic attractor.

Appendices

Appendix A: Single Variable Prediction

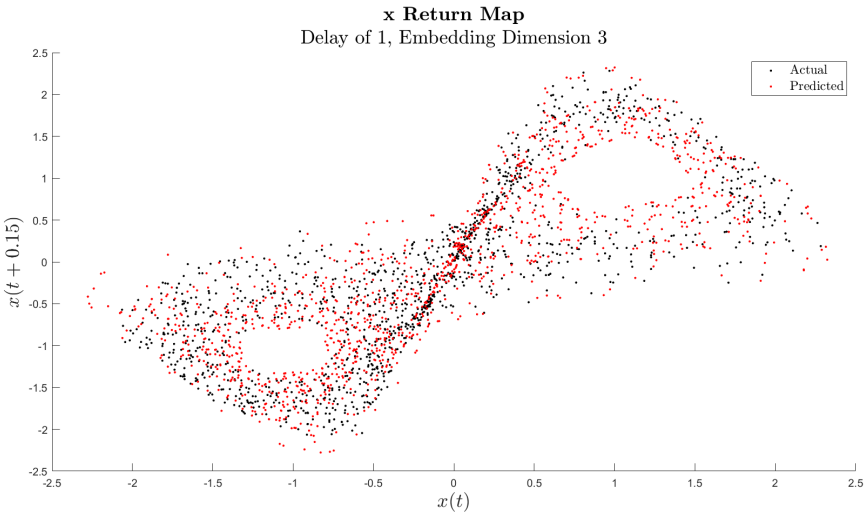
A.1: Pseudo Code



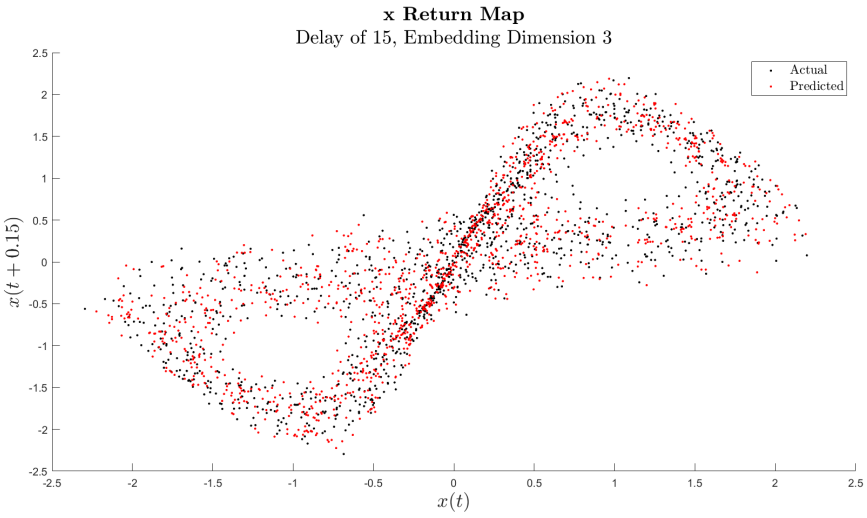
Pseudo Code for RBFNN Single Variable Prediction

A.2: x Return Map with Varying Delay

With both delay 1 and 15, it appears that the dynamics were captured for the x variable's trajectory. As per outcomes delineated within the report, however, the temporal prediction length was greater with delay 15.



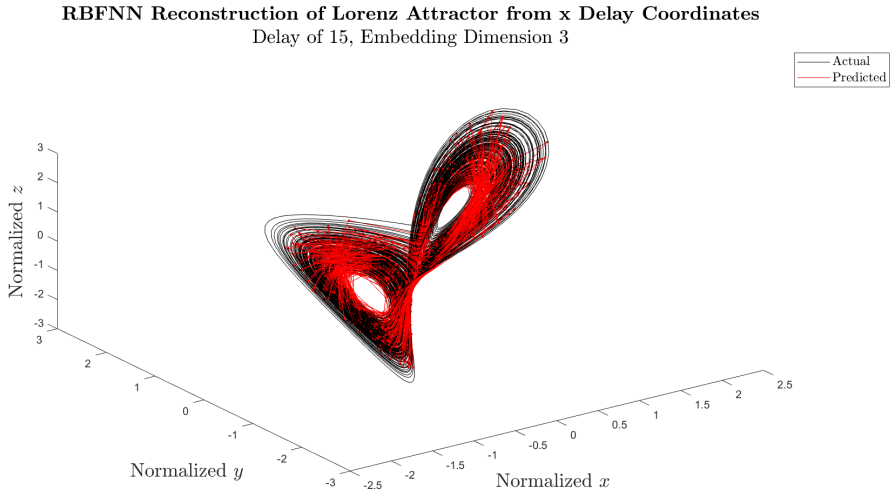
x Return Map with Delay 1



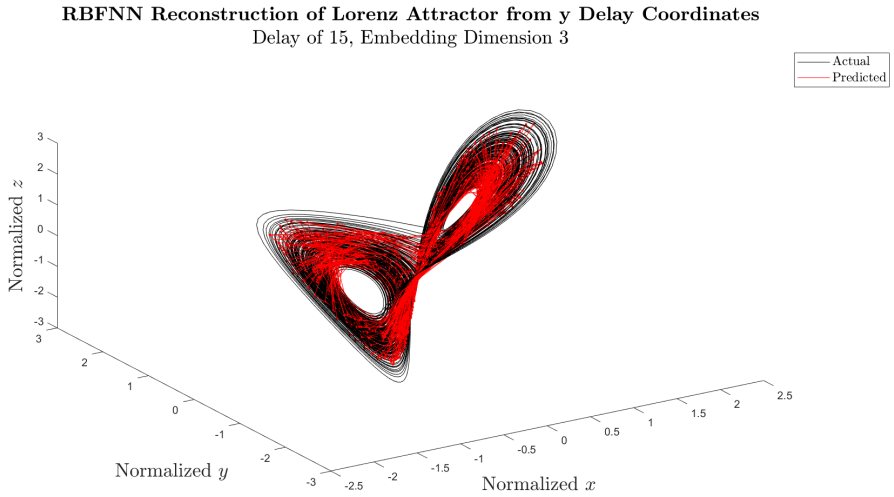
x Return Map with Delay 15

A.3: 3D Lorenz Prediction from Differing Variable Delay Coordinates

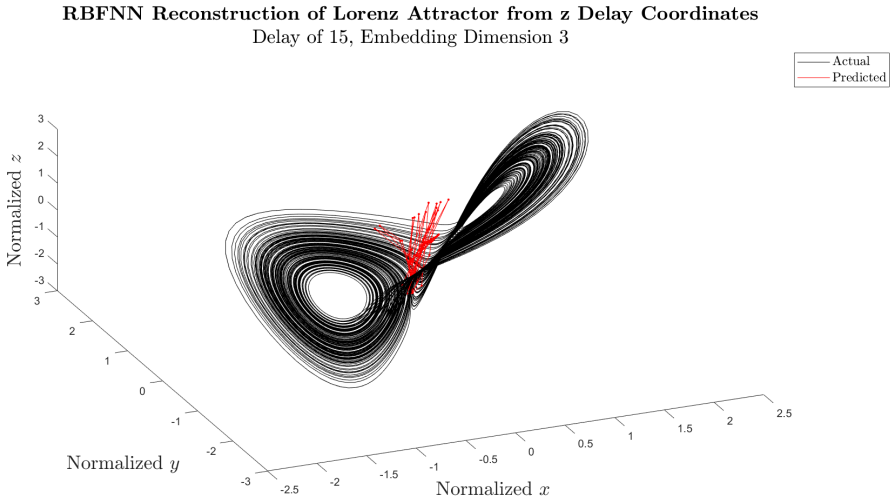
At delay 15 i.e. 0.15 time units, it was found that both the x and y variables with an embedding dimension of 3 were able to reconstruct the high-level dynamics of a Lorenz attractor.



x Delay Coordinates



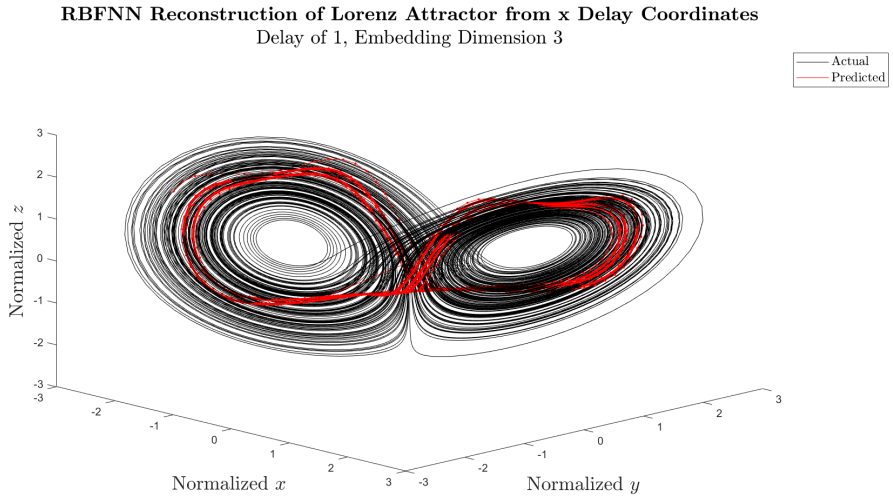
y Delay Coordinates



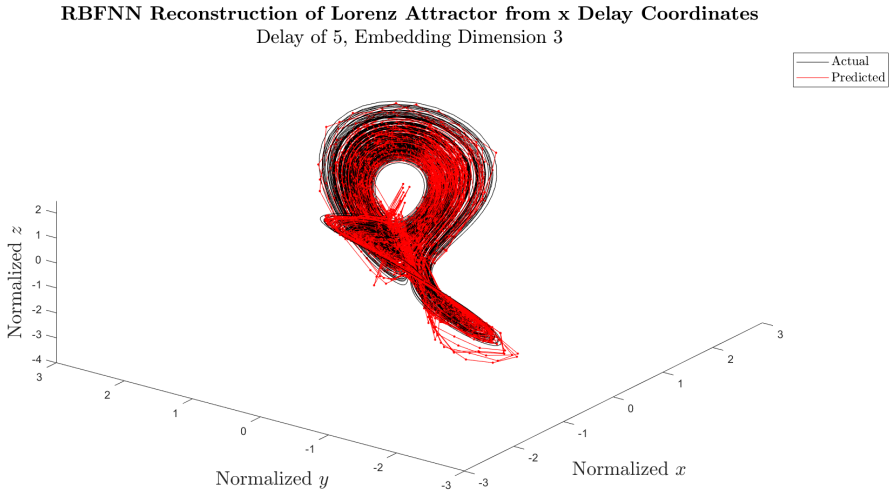
z Delay Coordinates

A.4: 3D Lorenz Prediction from x Trajectory – an Exploration of Delay

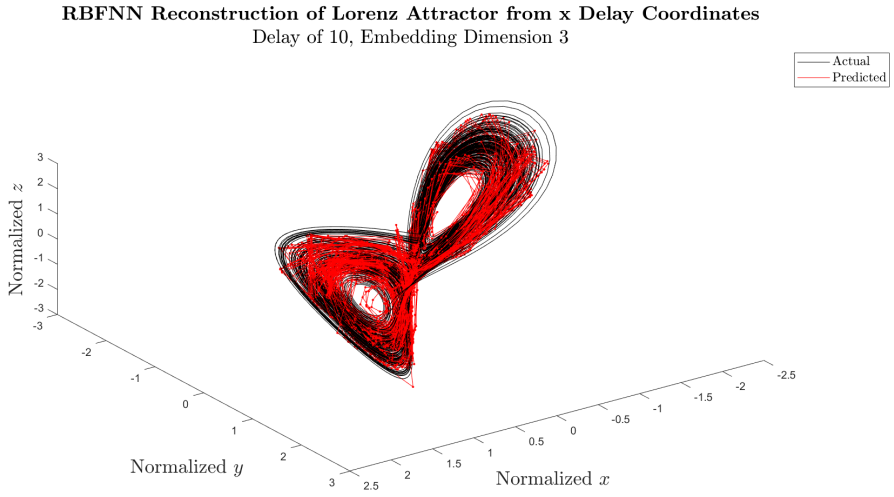
An embedding dimension of 3 was used for the following results with varying embedding delays. The RBFNN had 250 centres. These results further reaffirm the use of time delay 0.15 units in the x trajectory in opening up the attractor and making predictions. The delay of 15 at $dt = 0.01$ appears to be most important and required when predicting external variable trajectories from the x variable's tapped delay line.



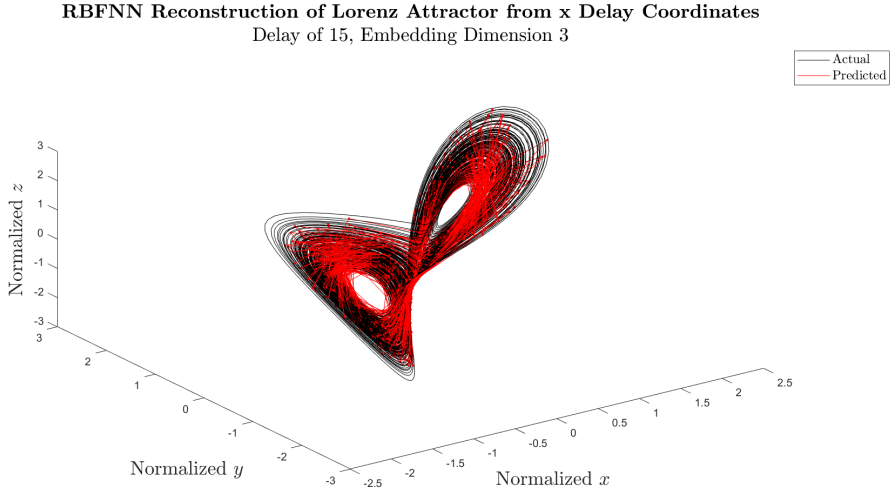
Embedding Delay 1



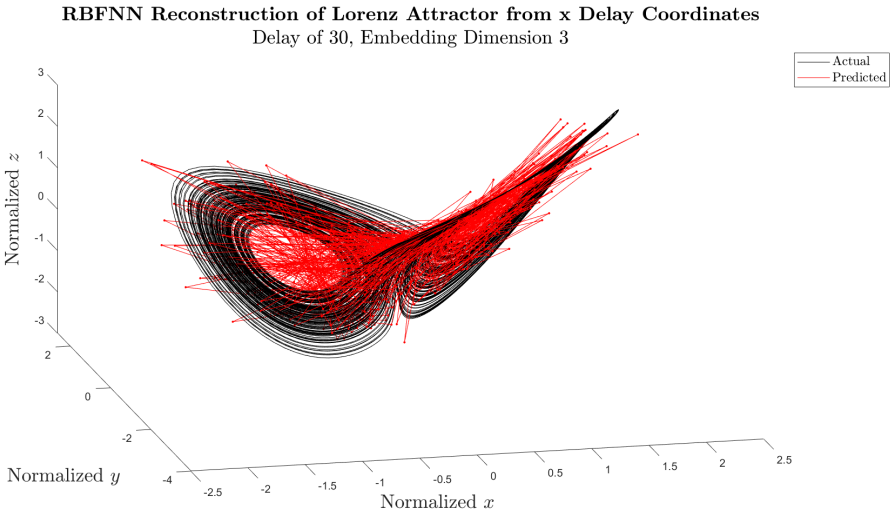
Embedding Delay 5



Embedding Delay 10



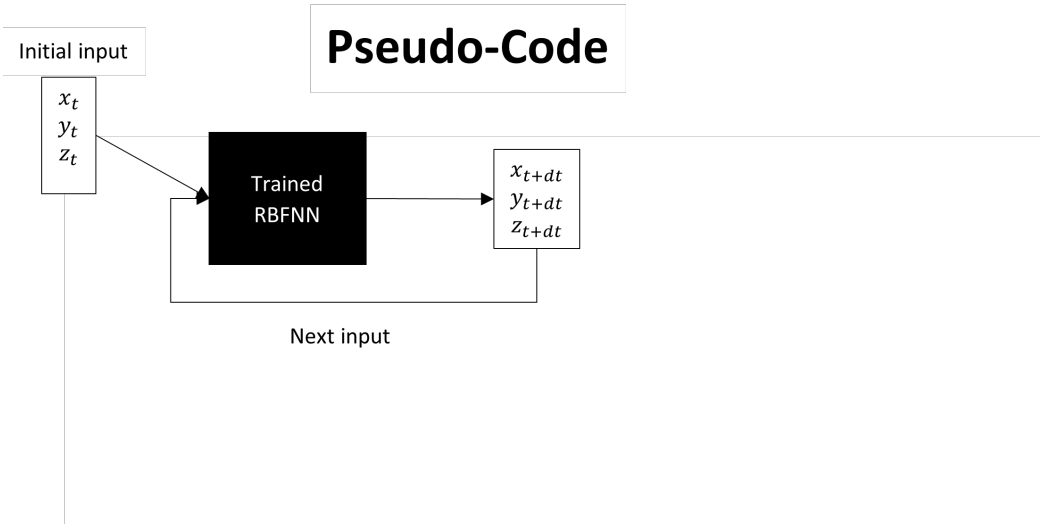
Embedding Delay 15



Embedding Delay 30

Appendix B: Lorenz State Prediction

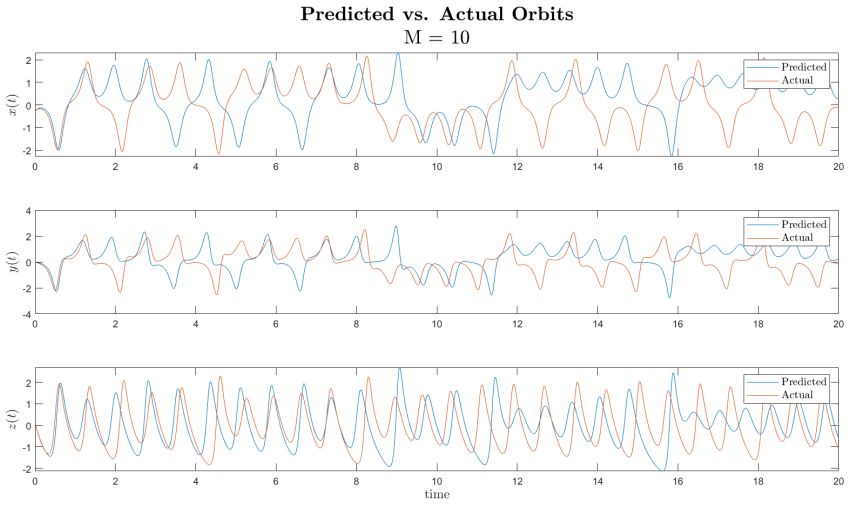
B.1: Pseudo Code



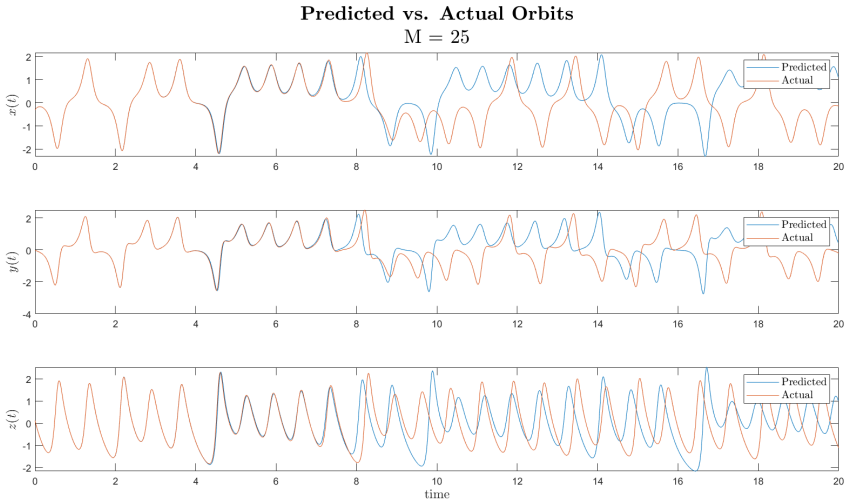
Pseudo Code for RBFNN Lorenz State Prediction

B.2: Varying Number of Centres

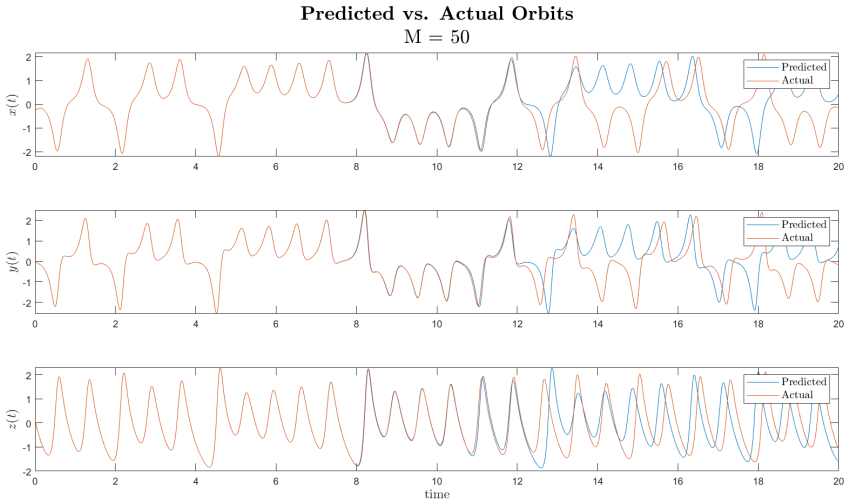
Although it was found within the report that additional centres helped tremendously for prediction of a single variable off it's own tapped delay line, the same was not observed using three-dimensional state prediction. Regardless, the attractor and overall dynamical behaviour of the Lorenz attractor was captured and the divergence is to be expected due to SDIC. The results show that as few as 100 centres can accurately follow a computed trajectory for approximately 12 units of time. Use of as few as 10 centres, however, showed that the overall dynamical behaviour of the Lorenz attractor can still be captured with very little information. These results used a RBFNN with consistently proportionate Gaussian radius of $r = \frac{10 d_{ME} d_e}{\sqrt{2M}}$ which was trained on 20000 training vectors.



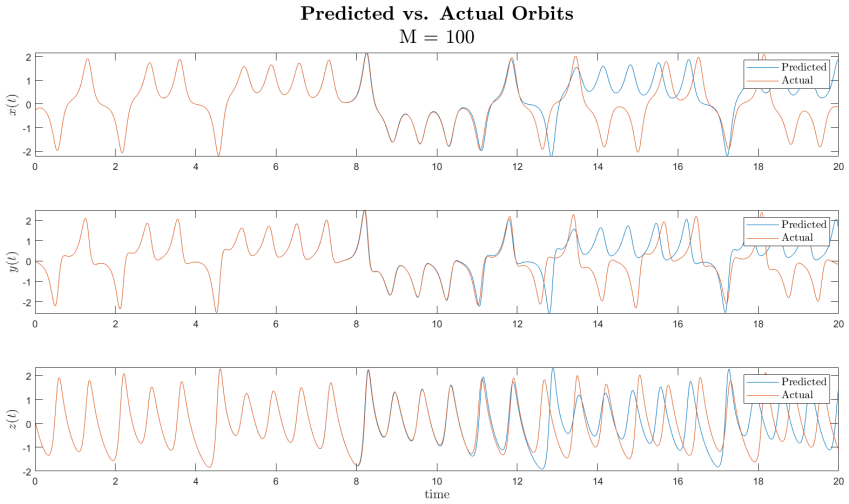
Prediction with 10 centres



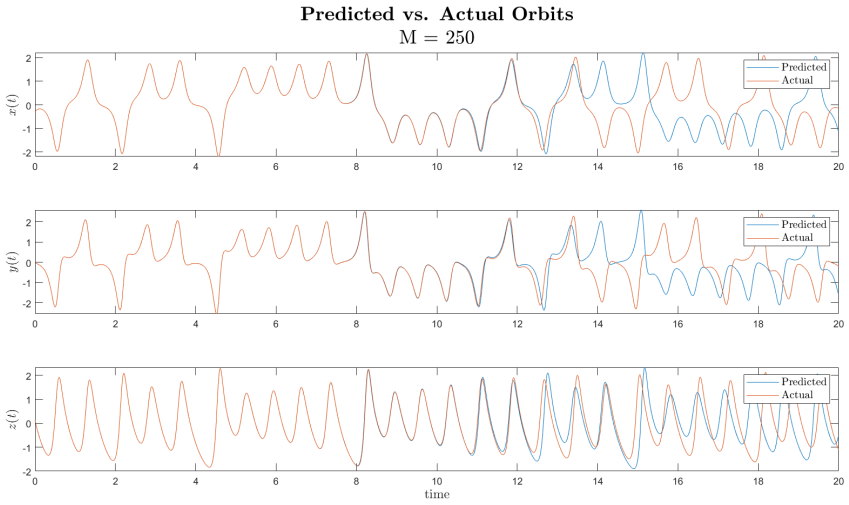
Prediction with 25 centres



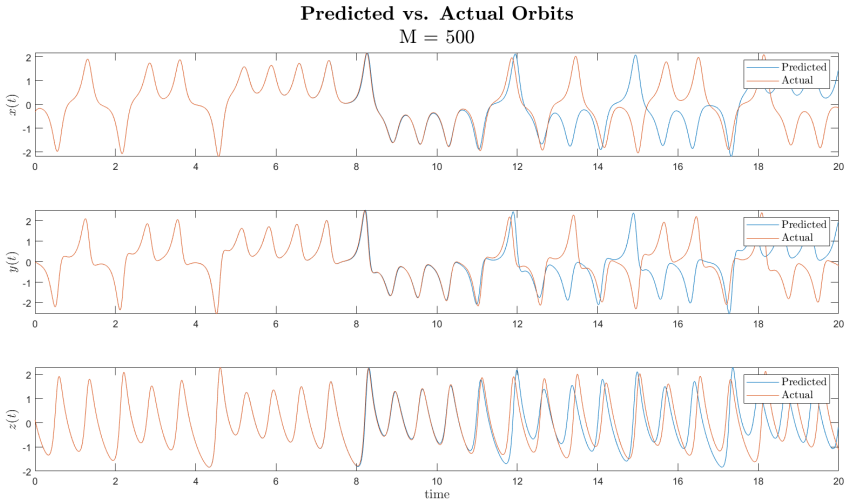
Prediction with 50 centres



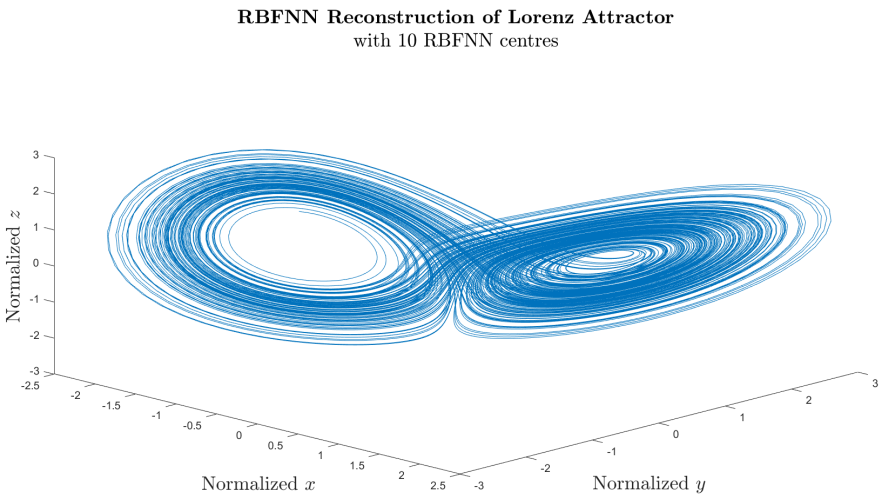
Prediction with 100 centres



Prediction with 250 centres



Prediction with 500 centres



RBFNN Reconstruction of Lorenz Attractor using 10 Centres

Appendix C: Matlab Code

C.1: Lorenz Prediction

```
%{

Centres are chosen to be equally spaced

% Author: Thomas Prince.

%}

clear
clc

% ------------------------------------------------------------------------
% Input these values:
% ------------------------------------------------------------------------

% Integer number of points to skip before sampling. Note that these are
% skipped with at a spacing of dt and that delay does not affect this.
nskip = 2000;
% Integer number of vectors to train on.
ntrain = 20000;
% Start position for the warmup of the Lorenz sampling.
rng(1);
startpos = [rand; rand; rand];
% Time between samples.
dt = 0.01;
% Lorenz parameters [sigma, beta, rho]. Classically [10, 8/3, 28].
pars = [10, 8/3, 28];
% Variable(s) to use as input for prediction. [x y z] => [1 2 3].
var = [1];
% Variable(s) to try and predict. [x y z] => [1 2 3]. Note that all
% input variables must also be predicted variables due to the nature of the
% network using successive predictions for further prediction.
pred = [1];
% Integer embedding dimension. The number of consecutive samples used
% for predicting an equidistant subsequent point in the orbit.
de = 3;
% Integer embedding delay. How many sampled points between predictions.
delay = 1;
% Integer number of centres. Note that there must be more training vectors
% than centres due to the fact that centres are chosen to be equidistantly
% spaced training vectors. If it is larger, than duplicate centres will
% inadvertently be created.
M = 250;
% Number of RBFNN predictions to validate on.
validate = round(1200/delay);
% Specified Gaussian radius. If r = 0, then the radius will be determined
% in the algorithm. r must otherwise be greater than 0.
r = 0;
% Binary for z-score normalization of input/output data.
normalize = 1;
% Integer number of validation tests from a random starting orbit on the
% Lorenz attractor
rtest = 3;
```

```
% ---------------------------------------------------------------------
% Samples the Lorenz equations
% ---------------------------------------------------------------------

warmup = lorenz(delay*(nskip+de), startpos, dt, pars);
nextstart = lorenz(1, warmup(:,nskip*delay), dt, pars);
trainsample = lorenz(delay*(ntrain+de), nextstart, dt, pars);

mu = [0; 0; 23.49];
sd = [7.92; 9.02; 8.68];

if normalize
    trainsample = (trainsample - mu)./sd;
end

% ---------------------------------------------------------------------
% Trains the RBFNN
% ---------------------------------------------------------------------

trainvecs = zeros(ntrain, length(var)*de);
traintarg = zeros(ntrain, length(pred));

for i = 1:ntrain
    temp = trainsample(var, delay*(i+(0:de-1)));
    trainvecs(i, :) = temp(:)';
    temp = trainsample(pred, (i+de)*delay);
    traintarg(i, :) = temp(:)';
end

centres = trainvecs(floor(linspace(1,ntrain,M)), :);

if r == 0
    d = zeros(M);
    for i = 1:M
        for j = i+1:M
            d(i,j) = sqrt(sum((centres(i,:) - centres(j,:)).^2));
        end
    end
    d = d + d';
    a = mean(mean(d,2));
    dme = max(max(d));
    r = 10*de*dme/(delay*sqrt(2*M));
end

phi = zeros(ntrain, M);
for i = 1:ntrain
    for j = 1:M
        phi(i,j) = exp(-sum((trainvecs(i,:) - centres(j,:)).^2)/(2*r^2));
    end
end

sums = sum(phi,2);
w = pinv(phi)*traintarg;
```

```
trainpred = phi*w;
error = abs(traintarg - trainpred);
meanerror = mean(error);
cumerror = cumsum(error);

% ---------------------------------------------------------------------
% Validating the network from the end of the training sample
% ---------------------------------------------------------------------

nextstart = trainsample(:, delay*ntrain);
valsample = lorenz(delay*(validate+de), nextstart, dt, pars);
if normalize
    valsample = (valsample-mu)./sd;
end
valvecs = zeros(validate, length(var)*de);
temp = valsample(var, 1+(0:de-1)*delay);
valvecs(1,:) = temp(:)';
phi = zeros(1,M);
preds = zeros(validate, 3);
predict = zeros(validate, length(pred));
for i = 2:validate+1
    for j = 1:M
        phi(j) = exp(-sum((valvecs(i-1,:) - centres(j,:)).^2)/(2*r^2));
    end
    predict(i-1,:) = phi*w;
    preds(i-1,pred) = predict(i-1,:);
    valvecs(i,:) = ...
        [valvecs(i-1,length(var)+1:length(var)*de),...
         preds(i-1,var)]';
end

% ---------------------------------------------------------------------
% Validating the network from random starting points
% ---------------------------------------------------------------------

randsample = zeros(3,delay*(validate+de));
randpredict = zeros(validate, length(pred), rtest);
for test = 1:rtest
    warmup = lorenz(delay*(nskip+de), [rand;rand;rand], dt, pars);
    nextstart = warmup(:, delay*nskip);
    temp = lorenz(delay*(validate+de), nextstart, dt, pars);
    randsample(:,:,test) = temp;
    if normalize
        randsample(:,:,test) = (randsample(:,:,test)-mu)./sd;
    end
    randvecs = zeros(validate, length(var)*de);
    temp = randsample(var,1+(0:de-1)*delay,test);
    randvecs(1,:) = temp(:);
    phi = zeros(1,M);
    preds = zeros(validate, 3);
    for i = 2:validate
        for j = 1:M
            phi(j) = exp(-sum((randvecs(i-1,:) - centres(j,:)).^2)/(2*r^2));
        end
```

```
        randpredict(i-1,:,test) = phi*w;
        preds(i-1,pred) = randpredict(i-1,:,test);
        randvecs(i,:) = [randvecs(i-1,length(var)+1:length(var)*de),...
            preds(i-1,var)]';
    end
end
```

C.2: Logistic Map

```
% Research results pertaining to the Logistic Map
% Author: Thomas Prince

clear
clc

r = 3.7;
x_0 = 0.5;
n = 300;
x = zeros(1,n);
x(1) = x_0;
for i = 2:n
    x(i) = r*x(i-1)*(1-x(i-1));
end

plot(0:n-1,x,'k')
hold on
scatter(0:n-1,x,'r.')
title(["\textbf{Logistic Map:}", sprintf("$r=%0.2f,x_0=%0.2f,n=%d\\,iterations$",↙
r,...
    x_0, n)], 'fontsize', 12, 'Interpreter','latex')
xlabel("$n$",'fontsize', 12, 'Interpreter','latex')
ylabel("$x_n$", 'fontsize', 12, 'Interpreter','latex')

clear
clc

r = [3.4,3.7,3.74]';
x_0 = 0.5;
n = 50;
x = zeros(length(r),n);
x(:,1) = x_0;
for i = 2:n
    x(:,i) = r.*x(:,i-1).*(1-x(:,i-1));
end

figure()
for i = 1:length(r)
    subplot(1,length(r),i)
    plot(0:n-1,x(i,:),'k',0:n-1,x(i,:),'r.')
    title(sprintf("$r=%0.2f$", r(i)), 'fontsize', 12, 'Interpreter','latex')
    xlabel("$n$",'fontsize', 12, 'Interpreter','latex')
    ylabel("$x_n$", 'fontsize', 12, 'Interpreter','latex')
end

clear
clc

r = 3.7;
x_0 = [0.3, 0.3001]';
n = 50;
x = zeros(length(x_0),n);
x(:,1) = x_0;
```

```
for i = 2:n
    x(:,i) = r.*x(:,i-1).*(1-x(:,i-1));
end
figure()
plot(0:n-1,x(1,:),'k',0:n-1,x(2,:),'--r')
legend(sprintf('$x_0=%0.4f$',x_0(1)), sprintf('$x_0=%0.4f$',x_0(2)),...
    'Interpreter','latex')
title(sprintf("Sensitive Dependence: $x_{n+1}=%0.1fx_n(1-x_n)$", ...
    r), 'fontsize', 12,'Interpreter','latex')
xlabel("$n$", 'fontsize', 12, 'Interpreter','latex')
ylabel("$x_n$", 'fontsize', 12, 'Interpreter','latex')
```

References

[1] K. L. Priddy and P. E. Keller, *Artificial Neural Networks: An Introduction*. Bellingham, Washington: SPIE Press – The International Society for Optical Engineering, 2005, ISBN: 0-8194-5987-8.

[2] P. Mellodge and D. Feldman. (2020). 'Complexity Explorer: Introduction to Dynamical Systems and Chaos.' Accessed: September, 2020, Santa Fe Institute, [Online]. Available: `https://www.complexityexplorer.org/courses/105-introduction-to-dynamical-systems-and-chaos`.

[3] R. Kautz, *Chaos: The Science of Predictable and Random Motion*. New York: Oxford University Press Inc., 2011, ISBN: 978-0-19-959458-0.

[4] C. Rouvas-Nicolis and G. Nicolis, 'Butterfly effect,' *Scholarpedia*, vol. 4, no. 5, p. 1720, 2009, revision #137268. DOI: `10.4249/scholarpedia.1720`.

[5] E. Ott, *Chaos in Dynamical Systems*, 2nd ed. Cambridge, United Kingdom: Cambridge Univerity Press, 2002, ISBN: 0 521 01084 5.

[6] C. Sparrow, *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*. New York: Springer-Verlag, 1982, ISBN: 978-1-4612-5767-7.

[7] V. G. Ivancevic and T. T. Ivancevic, *High-Dimensional Chaotic and Attractor Systems: A Comprehensive Introduction*. Springer, 2007, ISBN: 1-4020-5456-4.

[8] E. N. Lorenz, *The Essence of Chaos*. UCL Press Limited, 1995, ISBN: 0-295-97270-X.

[9] J. B. Reece, S. A. Wasserman, N. Meyers, P. V. Minorsky, L. A. Urry, R. B. Jackson, M. L. Cain and B. N. Cooke, 'Neurons, Synapses, and Signalling,' in *Campbell Biology Australian and New Zealand Version*, 10th ed. Melbourne, VIC: Pearson Australia Group Pty Ltd, 2015, ch. 48, ISBN: 9781486007042.

[10] C. P. H. Seo, 'An Artificial Muscle Neuron,' M.S. thesis, The University of Auckland, 2014.

[11] W. S. McCulloch and W. Pitts, 'A Logical Calculus of the Ideas Immanent in Nervous Activity,' *Bulletin of Mathematical Biology*, vol. 52, pp. 99–115, 1990.

[12] A. Krenker, 'Introduction to the Artificial Neural Networks,' in *Artificial Neural Networks – Methodological Advances and Biomedical Applications*, K. Suzuki, Ed. Croatia: InTech, 2011, ch. 1.

[13] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford Uniersity Press Inc., 1995.

[14] OpenStax. (2016). 'Anatomy & physiology.' Accessed: October, 2020, [Online]. Available: `http://cnx.org/contents/14fb4ad7-39a1-4eee-ab6e-3ef2482e3e22@8.24`.

[15] A. Kempa-Liehr, *ENGSCI712 Computational Techniques for Signal Processing*, University Lectures, 2020.

[16] C. Unsworth, 'The Radial Basis Function Neural Network (RBFNN),' in *ENGSCI712 Computational Techniques for Signal Processing*, 2020.

[17] (2020). 'Logistic map.' Accessed: October, 2020, [Online]. Available: `https://en.wikipedia.org/wiki/Logistic_map`.