AN ADAPTIVE RADIAL BASIS FUNCTION NEURAL NETWORK GLOWWORM SWARM OPTIMIZATION FOR TIME-SERIES FORECASTING

By

ISIMETO, ROSELYN OLUA

NOVEMBER, 2017

AN ADAPTIVE RADIAL BASIS FUNCTION NEURAL NETWORK GLOWWORM SWARM OPTIMIZATION FOR TIME-SERIES FORECASTING

By

ISIMETO, ROSELYN OLUA

Matriculation Number: 019074049

B.Sc. Agricultural Economics (1994), University of Nigeria

PGD Computer Science (2002), University of Lagos

M.Sc. Computer Science (2008), Université Charles de Gaulle, Lille, France

A thesis submitted to the School of Postgraduate Studies, University of Lagos, Akoka, Lagos, Nigeria, in partial fulfilment of the requirement for the award of the degree of Doctor of Philosophy (Ph.D.) in Computer Science.

NOVEMBER, 2017

AUTHOR'S STATEMENT

I hereby agree to give the University of Lagos Library, a non-exclusive, worldwide right to reproduce and distribute my thesis and abstract (hereinafter "the Work") in whole or in part, by any and all media of distribution, in its present form or style or in any form or style as it may be translated for the purpose of future preservation and accessibility provided that such translation does not change its content.

By the grant of non-exclusive rights to University of Lagos through the Library under this agreement, I understand that the rights of the University of Lagos are royalty free and that I am free to publish the Work in its present version or future versions elsewhere.

Warranties

I further agree as follows:

i. That I am the author of the Work and I hereby give the University of Lagos the right to make available the Work in the way described above after a three (3) year period of the award of my doctorate degree in compliance with the regulation established by the University of Lagos Senate.

ii. That the Work does not contain confidential information which should not be divulged to any third party without written consent.

iii. That I have exercised reasonable care to ensure that the Work is original and it does not to the best of my knowledge breach any Nigerian law or infringe any third party's copyright or other Intellectual Property Right.

iv. That to the extent that the Work contains material for which I do not hold copyright, I represent that I have obtained the unrestricted permission of the copyright holder to grant this license to the University of Lagos Library and that such third party material is clearly identified and acknowledged in the Work.

v. In the event of a subsequent dispute over the copyrights to material contained in the Work, I agree to indemnify and hold harmless the University of Lagos and all of its officers, employees and agents for any uses of the material authorised by this agreement.

vi. That the University of Lagos has no obligation whatsoever to take legal action on my behalf as the Depositor, in the event of breach of intellectual property rights, or any other right, in the material deposited.

_____	_____	_____
Author's Name	Signature/Date	Email
_____	_____	_____
Supervisor's Name	Signature/Date	Email
_____	_____	_____
Supervisor's Name	Signature/Date	Email
_____	_____	_____
Supervisor's Name	Signature/Date	Email

CERTIFICATION

School of Postgraduate Studies, University of Lagos.

This is to certify that the thesis entitled

AN ADAPTIVE RADIAL BASIS FUNCTION NEURAL NETWORK GLOWWORM SWARM OPTIMIZATION FOR TIME-SERIES FORECASTING

Submitted to the School of Postgraduate Studies, University of Lagos for the award of the degree of Doctor of Philosophy (Ph.D.) in Computer Science is a record of original research carried out by:

ISIMETO, ROSELYN OLUA

in the Department of Computer Sciences

Author's Name	Signature	Date
First Supervisor's Name	Signature	Date
Second Supervisor's Name	Signature	Date
Internal Examiner's Name	Signature	Date
First External Examiner's Name	Signature	Date
Second External Examiner's Name	Signature	Date

DEDICATION

In loving memory of Professor Adetokunbo Babatunde Sofoluwe. You left a vacuum difficult to fill. You are one of the world's most unassuming professors. A man humble extremely to the core. A man who will always encourage you when you have challenges by saying "don't worry, it will soon be over". Ever making me at ease and happy when I am around you. Your absence caused me a lot. Oh, you left too early, my Prof, my daddy, my friend!. Rest in Peace!!.

ACKNOWLEDGEMENTS

It has been a journey. A journey of my life. A journey that has been the most challenging, the most tasking, and the most time-consuming job I have ever undertaken in my life so far! A task which demands you to know this, to know that. A task that makes you connect to others and work as a team because you cannot do it all alone.

First of all, I thank the Almighty God for everything. From the start to the end of this research journey. I thank God for the wealth of knowledge gained from this study and for sparing my life all through my globe-trotting in the quest for knowledge. I also appreciate the University of Lagos and all the institutions that afforded me the opportunity to do this research.

The search for knowledge led me in 2012 to Maastricht University, Netherlands; where the Data Mining workshop I attended opened my eyes to practical data mining skills. Associate Professor Evgueni Smirnov, thank you for the invitation to attend the workshop and for all your advice to me after I presented my initial course of work when you asked all doctoral students to present their research work. In China, the Machine Learning Summer School workshop at Renmin University, which I attended in 2014, enriched further my knowledge and afforded me the opportunity to meet live, the experts I saw online in this field of research.

I owe unquantifiable gratitude to my supervisor, Professor C.O. Uwadia. A mentor extraordinaire! Thank you a trillion folds for all your support, patience, guidance and excellent advice through it all. Special heartfelt thanks go to you for your brilliant ideas and all you have done to see to the conclusion of this work. Thank you for all the days you spent tirelessly cleaning up my write-up. You had all the patience with me. Thank you, Prof. Uwadia. May Heavens bless your kind and patient spirit. I simply can never thank you enough!. Special gratitude goes to Dr . E. P. Fasina for his direction and shaping of this research. Your criticisms impacted this work in no small measure. The tremendous support of Dr. Yinka-Banjo is highly appreciated. I appreciate deeply your divine intervention in this work. Thank you so much, Dr. C.O. Yinka-Banjo. I must also appreciate Dr. A. O. Sennaike for his role towards the end of this work. God bless you all richly.

Indeed, the whole staff of our department of Computer Science must be appreciated. Dr. B. A. Sawyerr, Dr. V. Odumuyiwa, Dr. O. B. Okunoye, Dr. N. A. Azeez, Dr. A. U. Rufai, Mr. L. Ikuvwerha, Mrs. R.A. Ajetunmobi, Mrs. C. Ojiako, Mrs. D.T. Afolabi, Mr. S. E. Edagbami and Mr. O. O. Ajayi, I deeply appreciate you all for the spirit of love, unity and camaraderie in our department. I am happy being a part of you because without your love and encouragement, I would not be able to complete this study. You supported me through it all. I owe you all the success. You created an enabling environment for me to successfully complete this programme.

I also appreciate the support and contributions of the non-teaching staff of the department, Mrs. Toyin Aloku, Ms. Idowu, Mrs. Alayaki, Mrs. Bamgbelu, Mrs. Emeana, Mrs. Oluwamuhuru, Alhaja Gbolahan, Ms. Nonye Nbonu, Mr Isaiah Ayandele, Mr Imenvbore, Mr. Olaitan, Mr Adewunmi, Mr. Okhueleigbe, Mr. Olagunju and all other staff. I must specially appreciate you, Ms. Idowu for all the inconveniences you went through when you stayed severally with me till very late at night and even slept with me, at my office, to support me so l could meet the submission deadlines. Thank you so much.

Our indefatigable HOD, Dr. F.A. Oladeji, for all the tremendous progress achieved in our department during your tenue, thank you, ma. The effort of our P.G. Co-ordinator, Dr. A.P. Adewole (Associate Professor), is deeply acknowledged. Thanks for linking me to Dr. S.O.N. Agwuegbu (Associate Professor), former HOD, Department of Statistics, UNAAB. Dr. Agwuegbu, thanks for all the time you spent in teaching me and for travelling down to watch my seminar presentations. Engr. D. Alienyi, thank you for all your teachings as well. I must also mention the support of Dr. A.B Adeyemo and Dr. Osunade of Department of Computer Science, University of Ibadan.

To the founding pillars of our department, Emeritus Professor O. Abass, Professor H.O. D. Longe, Prof. J. O. A. Ayeni, thank you very much for your keen interest in this work and for your immense support and contributions. Long may you reign! I must not fail to appreciate our faculty members. To Prof. Adekunle, Dr. Adeoti, Dr. Akala, thank you for all the criticisms during my seminar presentations. To Prof. Ilori and Prof. Okafor, thank you for the motivations. I also acknowledge the faculty of science and the University of Lagos for the research grant awarded to me.

Lastly, my family has been immensely supportive. To my son, Imoaghene, thank you for being with me all through it all. You needed my attention to play with you, but I always begged you to allow me concentrate. You told me one day “when will you finish this Ph.D? Mummy, you are not intelligent. You started this Ph.D before I started secondary school. I have finished my secondary school. You are yet to finish”. I henceforth will have all the time for you. To my parents, Mr. & Mrs. J.K. Isimeto, and my siblings, thank you all for the love and support all these years. To my dearest friend, O’bryan, thank you for all your support and motivation.

To God be all the glory!!!

ABSTRACT

It is well noted that statistical approaches to forecasting of time series have been going on since the start of the twentieth century. Advances in the field of computing, motivated researchers to develop new models based on Machine Learning. The Artificial Neural Network models (ANN) are known to construct good and useful approximations for complex sequence dependencies variables. The past three decades have witnessed active research using a class of ANN, the Radial Basis Function Neural Networks, to forecast time series.Many techniques for forecasting time series using Radial Basis Function Neural Networks (RBFNN) have been proposed and developed in literature. The major challenges in RBFNN lie in the optimization of its full parameters: the number and location of cluster centres, the number of neurons in the hidden layer as well as the output weights. To address these challenges, this study adapted the Clustering Analysis based on Glowworm Swarm Optimization (CGSO) algorithm to obtain a modified Clustering Analysis based on Glowworm Swarm Optimization (CGSOm) algorithm for solving the clustering problem. Adaptation was achieved by incorporating a mechanism that determines the sensor range of the CGSO efficiently and automatically, modifying the glowworm initialization method, and introducing a function that measures the cluster error during the iteration phase. For the weight optimization, the Bioluminescence Swarm Optimization algorithm (BSO) was adopted, making it the first time it will be applied in training the weights of the RBFNN. Algorithm as well as software development, and graphical simulation in this work are implemented using functional programming paradigm. The algorithms implemented include the CGSO, CGSOm, BSO, Conjugate Gradient Descent (CGD), Gradient Descent (GD) and Particle Swarm Optimization algorithm (PSO). Using seven well known datasets in literature, the first set of results compared the effectiveness of the CGSOm with the following five well-known clustering algorithms: CGSO, K-means, average linking agglomerative Hierarchical Clustering (HC), Further First (FF), and Learning Vector Quantization(LVQ). Experimental results indicate that the CGSOm gave best entropy and purity values in four out of the seven datasets clustered (57%); CGSO gave best results in two datasets (28.5%); and HC gave best result in one dataset (14.5%). With respect to the weight training, stock price and currency exchange rate data were used to train the combinations of models developed (based on K-means, CGSO, CGSOm and GD, CGD, PSO, BSO). The results obtained from the training showed that the CGSOm-CGD RBFNN gave best forecasting accuracy by yielding lowest error values; followed by the CGSOm-BSO RBFNN that gave relatively similar error values. Hence, two new training methodologies for time series forecasting resulted from this study; they are the CGSOm-BSO RBFNN and the CGSOm-CGD RBFNN. Validation of the proposed approaches was done in comparison with other RBFNN models: Auto Regressive-Radial Basis Function tuned using Genetic Algorithm and Evolving Radial Basis Function Neural Network, using same data. The results obtained showed that CGSOm-BSO RBFNN and the CGSOm-CGD RBFNN yielded lowest error values.

Keywords: Radial Basis Function Neural Network, Time Series Forecasting, Swarm Intelligence, Clustering algorithms, Glowworm Swarm Optimization algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTS OFALGORITHMS

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

Time series forecasting problems are a difficult type of predictive modelling problem, since time series is a chronological sequence of observations on a particular variables(s). Unlike regression predictive modelling, time series adds the complexity of sequence dependence among the input variables, which are usually taken at regular intervals (days, months, years).

The goal of building a time series model is the same as the goal for other types of predictive models which is to create a model such that the error between the predicted value of the target variable and the actual value is as small as possible. The primary difference between time series models and other types of models is that the lag values of the target variables are used as predictor variables, whereas traditional models use other variables as predictors, and the concept of a lag value does not apply because the observations do not represent a chronological sequence.

Time series prediction using Artificial Intelligence (AI)/Machine Learning techniques has been ongoing in the last 30-40 years.Recent studies have shown a notable AI technique, the Artificial Neural Networks (ANN) can be constructed as a good and useful approximation for complex sequence dependencies variable(s). The search for new models of computing based on artificial neural networks is motivated by the quest to solve natural (intelligent) tasks by exploiting the developments in computer technology (Yegnanarayana,2010). Artificial Neural Network extracts relevant features from input data and perform pattern recognition tasks by learning from examples without explicitly stating the rules for performing the tasks.

Machine Learning is known as the domain of knowledge that entails programming computers to optimize a performance criterion using example data or past experience (Alpaydin, 2010;Han *et al*., 2012). The use of machine learning has spread rapidly throughout computer science and beyond and its application areas include websearch, recommender systems, fraud detection, robotics, medical diagnosis, and so on.

A class of ANN,the Radial Basis Function Neural Network (RBFNN) has been applied to solvevarious problems such as function approximation, modelingdynamic systems, time series prediction, pattern recognition, classification and system controls. For the past three

decades, there has been active research on using RBFNN to forecast time series data. This work joins this research effort.

A major challenge in RBFNN optimization is the difficulty in knowing the number and location of centres. From survey, most techniques used in clustering the centres require stating the numberor using trial and error. This is a limitation as it is practically impossible to know the number of clusters in a dataset, except there is a ground truth.

A bio-inspired swarm intelligence technique,the Clustering Analysis based onGlowworm Swarm Optimization (CGSO) algorithm was proposed by Aljarah and Ludwig (2013).It can automatically discover number of clusters and it has not yet been used to cluster the RBFNN centres. This work adapted the CGSO algorithm toobtain the CGSOm which was used in solving the clustering problem in RBFNN. Simulationresultsshow the effectiveness of the CGSOm over that of CGSO and other four standard clustering algorithms commonly used in the literature when tested on benchmark datasets.

With respect to optimizing the output weights of the RBFNN, tremendous achievements have been recorded with the Swam Intelligence (SI) techniques. One recent SI technique is the Bioluminescence Swarm Optimization (BSO) algorithm by Rossato de Oliveira *et al*.,(2011). BSO is attracted to global optimum; it converges more slowly and smoothly, avoiding getting trapped into local maxima compared to the Particle Swarm Optimization algorithm (PSO) that easily gets trapped into local maxima. Hence,BSO leads to more accurate, optimal results than the PSO. An account of the BSO outperforming the PSO was recorded by Rossato de Oliveira *et al*.,(2011). Due to these interesting characteristics of both the CGSO and BSO, this research focuses on using the adapted CGSO(the CGSOm) for clustering the centres and the BSO for training output weights of the RBFNN network. To the best of our knowledge, this is the first time BSO is being used to optimize the RBFNN model. The performance of this approach is compared with the performance of existing RBFNN variants for time series forecasting problems found in the literature.

1.2 Statement of the Problem

Various researchers have established the fact that the major challenge in RBFNN is optimization of its full parameters: number and location of cluster centres, the output weights along with the number of neurons in the hidden layer (Awad, 2010; Rivas *et al*.,2004). Broomhead and Lowe (1988) also emphasized the need for automatic mechanism to build the RBFNNs.

The learning process of the RBFNN involves two tasks which are clustering and weight optimization. Different clustering algorithms have been used by researchers to select optimal centre sets. Conventional clustering algorithms, such as the K-means, experience premature convergence and achieve local optimal solutions. The emergence of Swarm Intelligence (SI)clustering algorithms solved the problems of conventional clusteringtechniques (Handl and Meyer, 2007; Shifei*et al*., 2010). However, the limitations in these SIclustering approaches are that while in some, the number of clustersare fixedprior to starting the clustering process, in others, trial and error approach is used to get the number of cluster centres.

The emergence of the Clustering analysis Based on Glowworm Swarm Optimization (CGSO) algorithm by Aljarah and Ludwig (2013) has solved the limitations of earlier SI clustering algorithms. CGSO can automatically discover the clusters within a dataset without prior knowledge about the number of clusters. However, in CGSO a sensor range parameter which determined the number of clusters as well as the cluster quality was obtained experimentally by trial and error, thereby making the approach inefficient.

On weight optimization, different techniques have been developed with varying degrees of success. However, it was observed that the Bioluminescence Swarm Optimization(BSO)algorithm by Rossato de Oliveira *et al*., (2011) has not been used to optimize the RBFNN.This study therefore sets out to improve the CGSO by incorporating an automated mechanism that determines the sensor range efficiently, by modifying the glowworm initialization method and introducing a function that measures the cluster error during the iteration phase. It also seeks to adopt the BSO to optimize the weights of the RBFNN.

1.3 Aim and Objectives of Study

The overall aim of this research is to develop a new approach of optimizing the RBFNN parameters fully for any given time series forecasting problem.

The specific objectives of this research are to:

(1)efficiently determine the sensor range of the CGSO algorithm.

(2) automatically determine the optimal number of clusters in a dataset.

(3)develop a RBFNN model that adapts to the number of clusters in a dataset.

(4) optimize the RBFNN parameters fully

1.4 Scope and Delimitation of the Study

This study is focused on forecasting time series problems irrespective of application domain.However, for the proposed RBFNN model to perform optimally on time series problems, the datasets must be clusterable or have clustering tendency.Otherwise, the strength of the developed tool cannot be effectively demonstrated

1.5 Significance of the Study

As established, the major challenge in RBFNN is optimization of its full parameters: number and location of cluster centres, the output weights along with the number of neurons in the hidden layer (Awad, 2010; Rivas *et al*., 2004). Also, Broomhead and Lowe (1988) emphasized the need for automatic mechanism to build the RBFNNs.This work through the proposed models has been able to tackle these limitations. Thus, the research community will be able to use for instance, the CGSOm algorithm to cluster datasets in order to get its optimal number of clusters. This is because the CGSOm has a sensor range algorithm incorporated into it, whichhelps it to determine the sensor range automatically. The CGSOm will be of benefit to researchers in the data mining community who want to know the number and location of cluster centres (centroids) in a dataset.

The proposed CGSOm-BSO and CGSOm-CGD RBFNN models for time series forecasting yield good forecast precisions. These models fix the major challenges in RBFNN optimization be it the number and location of cluster centres,the number of neurons in the hidden layer, as well as the output weights. Additionally, the models serve as automatic mechanism to build the RBFNN and will be a useful tool to those faced with such task of building automatic RBFNNs. Indeed, these models are major contributions to the statistical and machine learning community and will be of benefit to those domains and sectors involved in time-series forecasting.

1.6 Definitions of Terms

Clustering:otherwise known as cluster analysis, is the process of partitioning a set of data objects (or observations) into subsets called clusters.

Entropy:is a metric that is a measure of the amount of disorder in a vector.Entropy values range from 0 (perfect clustering quality) to 1 (very poor clustering quality). Smaller values of entropy indicate less disorder in a clustering, which means a better clustering**.**

Optimization: is obtaining the best resultunder the given circumstances. It is the process of nding the best result in the form of minimizing or maximizing the benefit desired (profit function), expressed in the form of a function of decision variables under certain constraints and/or under given conditions (Raju, 2014).

Principal Components Analysis(PCA):is a statistical technique, designed to reduce the number of variables that need to be considered to a small number of indices, the principal components that are linear combinations of the original variables. Principal components analysis provides an objective way of finding indices of variations in data, so that the variations in the data can be accounted for as concisely as possible.

Purity:measures the percentage of the total number of objects (data points) that were classified correctly.Possible values of purity range from 0 (very poor clustering quality) to 1 (perfect clustering quality).

Rand Index:is a cluster quality evaluation measure that checks how close the resulting cluster is to the original cluster in terms of number of clusters and data points. It checks for the extent of agreement of thenumber of clusters as well as data points in the resulting cluster and the original cluster (the ground truth).Rand Index values range from 0 (very poor clustering quality) to 1 (perfect clustering quality).

Sensor range:isthe radius around a glowworm that determines its neighbourhood. All glowworms within the sensor range (perimeter) of a given glowworm are classified as its neighbours.

Time series:is a collection of observations made sequentially in time. A Time series is a sequence of vectors, $x(t)$, t=0,1,…, where t represents elapsed time. Time Series are ubiquitous as they occur in virtually most domains including medical, scientific, business, and entertainment.They exist in different data formats such as image data, video data, handwriting data, brain scan, and numeric data (Keogh, 2003).

1.7 List of Abbreviations

BSO	Bioluminescence Swarm Optimization algorithm
CGD	Conjugate Gradient Descent

CGSO **C**lustering Analysis based on**G**lowworm **S**warm **O**ptimization (CGSO) algorithm

CGSOm modified **C**lustering Analysisbased on **G**lowworm **S**warm **O**ptimization CGSOalgorithm

GSO Glowworm Swarm Optimization algorithm

PCA Principal Component Analysis

PSO Particle Swarm Optimization algorithm

RBFNN Radial Basis Function Neural Network model

1.8 Thesis Outline

The remaining chapters of this thesis are structured as follows:

Chapter two contains the literature review. The chapter presents the approaches to modelling time series,and introduces the Radial Basis Function Neural Network. Related work on approaches to full optimization of the RBFNN and their limitations are discussed. Also presented are limitations of related work on approaches to clustering as well as theories and concepts used in this study.

Chapter three presents the research methodology used to achieve each of the stated objectives. The objectives concerningclustering andweight optimization of the RBFNN model as well as the methods used to achieve these objectivesare well covered. Also presented are the proposed models: CGSOm-BSO and CGSOm-CGD RBFNN, as well as the techniques employed in developing these models for time series forecasting.

Chapter four presents the results and discussion on the simulation experiment.

Chapter five covers the summary of key findings, conclusion, contribution to knowledge and suggestion for future work.

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction

This chapter presents the approaches to modelling time series and introduces briefly the Radial Basis Function Neural Network. Related work on approaches to full optimization of the RBFNN and their limitations are discussed. Also presented are limitations of related work on approaches to clustering as well as theories and concepts used in this study.

2.1 Approaches to Modelling Time Series Data

Time series prediction is considered a modelling problem. The first step is establishing a mapping between the inputs and the outputs. Usually, the mapping is non-linear. After such mapping is done, future values are predicted based on past and current observations (Ortega*et al*., 2000).

Popular modelling techniques for time series analysis include statistical techniques and machine learning techniques.

2.1.1Statistical Techniquesinvolve use of statistical reasoning, analysis and modeling. These techniques make use of statistical probability and methods such as multiple correlation analysis, discriminant analysis and principal component analysis, factor analysis, regression methods, time series models (the Auto- Regressive Integrated Moving Average Process (ARIMA), Auto- Regressive (AR), Moving Average (MA), Vector Auto- Regressive (VAR) , Vector Auto- Regressive Moving Average (VARMA), etc. These techniques could be used for fitting models to univariate and multivariate data

2.1.2**Machine Learning Techniques**are programs that are able to improve their performance with experience. In other words, they are capable of learning. The domain of knowledge known as Machine Learning (Alpaydin, 2010) entails programming computers to optimize a performance criterion using example data or past experience. Machine Learning (Han *et al*., 2012) investigates how computers can learn or improve their performance based on data, and it is a fast growing discipline. Machine Learning systems (Domingo, 2012) automatically learn programs from data. The use of machine learning has spread rapidly throughout

computer science and beyond and its application areas include: web search, spam filters, recommender systems, ad placements, credit scoring, fraud detection, drug design, and many other applications. Machine learning also finds solutions to problems in vision, speech recognition, robotics, medical diagnosis and for time series forecasting problems.

Machine Learning techniquesincludethe Artificial Neural Networks, Support Vector Machine, Decision Tree, among a host of others.

2.1.2.1 Artificial Neural Network(ANN)is a system that is based on the biological neural network, such as the brain. The brain has about 100 billion neurons which communicate through electro-chemical signals. The neurons are connected through junctions called synapses, and each neuron receives thousands of connections with other neurons. The ANN attempts to recreate the computational mirror of the biological neural network. An ANN comprises of a network of artificial neurons called nodes. There are three types of neurons in an ANN: the input nodes, hidden nodes, and output nodes. The input nodes take in information from the environment, in form of predictor variables, which is numerically expressed as in $x_j \in$, j=1,...., d. Associated with each input is a connecting weights $w_j \in$, and the output, y, in the simplest case is a weighted sum of the input, as in Figure 1 (Alpaydin, 2010).

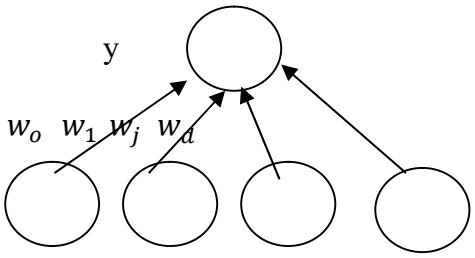


Figure 1: Artificial neural network architecture, the Simple Perceptron (Alpaydin, 2010)

The model for the Simple Perceptron is given as:

$$y = \sum_{j=1}^{d} w_j x_j + w_o \qquad (1)$$

Where w_o is the intercept to make the model more general, and it is modeled as the weight coming from an extra bias unit, x_o, which is always +1. The output of the perceptron could

be written as a dot product, y=W$^T x$, where W = [w_o , $w_{1,\ldots,} w_d$]^{T}and$x$ = [1,x_1, …, x_d]^{T}are augmented vectors to include the bias weight and input.

ArtificialNeural Networks provide a robust approach to approximating real-valued, discrete-valued, and vector valued target functions. ANNs are among the most effective learning methods currently known for certain types of problems such as learning to interpret noisy, complex real-world sensor data (Mitchell, 1997). A neural network is a two-stage regression or classification model. There exist variants of neural networks such as feed-forward, back-propagation, time delay, radial basis function, among others. ANN can be used for modeling univariate and multivariate data.

2.1.2.2 **Support Vector Machine (SVM)**is a machine learning algorithm that uses a linear hyperplane to create a classifier with a maximal margin. SVM is a powerful classification technique (Borgetto et al., 2012); and it is gaining much popularity in time series and regression prediction (Bankole and Ajila, 2013).

2.1.2.3 **Decision Tree Learning Method** is an efficient non-parametric method that can be used for both classification and regression. It is a method (Mitchell, 1997) for approximating discrete-valued target functions that is robust to noisy data, in which the learned function is represented by a decision tree. It has been successfully applied to a broad range of tasks from learning to medical cases to learning to access credit risk of loan applications. Variants of Decision tree algorithms exist.

This work focuses on modeling time series via the use of ANN. Precisely, this study focuses on the use of a machine learning technique, a variant of feed-forward neural networks, known as Radial basis function neural network (RBFNN). The RBFNN is strongly considered because of its ability for solving problems involving function approximation, prediction, pattern recognition, and modeling of dynamic systems and time series.

2.2Radial Basis Function Neural Network

A class of the ANN is the Radial Basis Function Neural Network (RBFNN)model.RBFNN is often referred to as model–free estimators as it can be used to approximate the desired outputs without requiring a mathematical description of how the output functionally depends on the inputs.As noted by Larsson and Fornberg(2005), the history of radial basis function approximations goes back to 1968, when multiquadric RBFs were first used to represent topographical surfaces given sets of sparse scattered measurements. Today, extensive

literature abounds on different aspects of RBF approximations. RBFs are used not only for interpolation or approximation of data sets but also as tools for solving differential equations.

Moody and Darken (1989) popularized RBF networks which have proven to be a useful neural network architecture.They are typically configured with a single hidden layer of units whose activation function is selected from a class of functions called radial basis functions or kernel functions. RBFNN has certain advantages over other types of neural networks such as better approximation capabilities, simple network structure and faster learning (Sneha, 2013). Though very similar to back propagation in many ways, radial basis function networks possess several advantages. They can be trained much more efficiently than the back propagation networks (Mitchell, 1997). The major difference between RBF networks and back propagation networks is the behavior of the single hidden layer. Rather than using the sigmoidal or S-shaped activation function as in back propagation, the hidden units in RBF networks use a Gaussian or some other basis kernel function. Radial basis function neural network (RBFNN)has received considerable applications in various problems such as function approximation, prediction, pattern recognition, and modeling of dynamic systems and time series.

Radial Basis Function Neural Networks typically have three layers: an input layer, a hidden layer with a non-linear radial basis functionand a linearoutput layer.The architecture of RBFNN is in Figure 2. The input layer consists of input signals which are propagated through the network. The transformation from the input space to the hidden unit space is nonlinear whereas the transformation from the hidden unit space to the output space is linear. Thus RBFNN produces a linear combination of non-linear basis functions where the dimension of input matches with the dimension of each radial centre. Each hidden unit or node is known as radial centre and each centre is representative of one or some of the input patterns. The hidden units in RBF networks use a Gaussian or some other basis kernel function. Each hidden unit acts as a locally tuned processor that computes a score for the match between the input vector and its connection centers. In effect, the basis units are highly specialized pattern detectors.

The weights connecting the basis units to the outputs are used to take linear combinations of the hidden units to produce the final classification or output. The output should ideally be equal to a desired output. The difference between the obtained and desired output is used to adjust or train the network parameters, so that the error is reduced. The network parameters consist of the hidden-to-output layer weights, and parameters associated to the hidden layer

functions represented by each hidden layer node. RBFNNs, similarly to all neural networks, are associated with a set of parameters that need to be adjusted in order for the neural network to "learn" the correct mapping between inputs and outputs. The set of parameters of a neural network is directly dependent on the neural network's architecture.

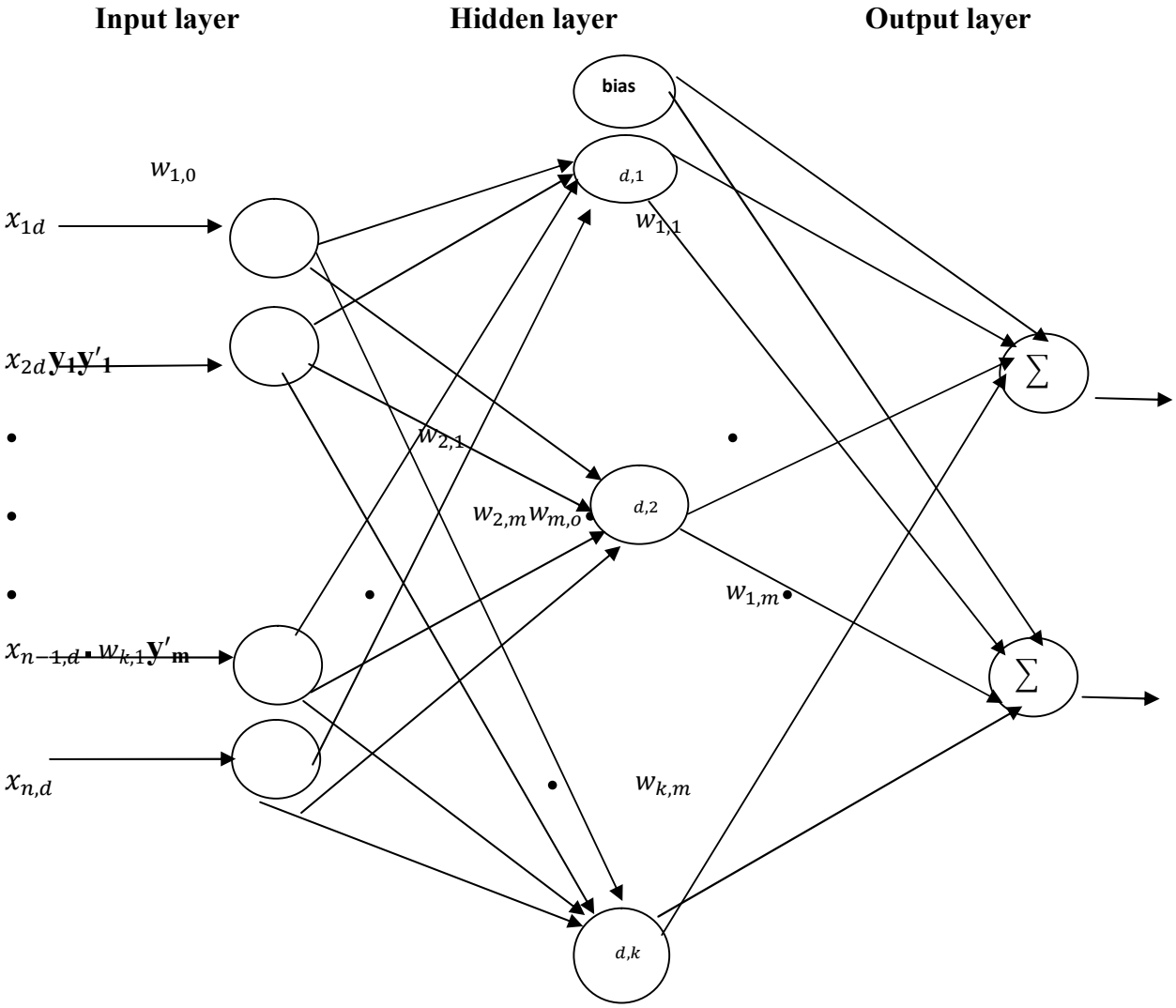


Figure 2: The Architecture of a basic RBFNN model (Alpaydin, 2010)

The RBFNN model is a function of the form: $y_m = w_0^m + \sum_{i=1}^{k} w_i^m \varphi(x)(2)$

where y_m is the value at output node m, w^m is a set of k number of weights used in mapping values from each hidden node to output node m (set of weights that minimize the sum of the

squared error, $\sum$); and w_0^m is a bias term. φ is the kernel function which the centroids are passed to, in order to compute the output from the hidden layer. x is the input vector.

2.3 The Learning Process in RBFNN

The two tasks in the learning process of RBFNN are clustering of the RBF cluster centres and optimization of the output weights.The learning process in RBFNN starts when a training set of input patterns (data) is presented to the RBF Neural Network, The input patterns are represented as points in the hyper-dimensional space. During the clustering process, the clustering algorithms adjust the centres, to get the optimal centre sets: mean and standard deviation, for the RBF network. After the clustering aspect is done with, the centroids are obtained and these are passed to the kernel function in order to compute the output from the hidden layer.Getting an output from hidden node requires the input vector, and the centroids and standard deviation at that node. The nodes in the hidden layer are a multidimensional vector. Each node in the hidden layer is defined by the values of centroids and the standard deviation that define its kernel function.

The second part is the weight optimization aspect. This entails adjusting the output weights via the use of optimization algorithms.The outputs from the hidden layer (kernel functions) are then passed as inputs to the output layer that then computes a weighted linear combination of these values from the hidden layer. At each output node, there is a calculated output, Y_{output}.If there is an error, that is, a difference between the calculated Y_{output} $(y'_{i,j})$ and the target output patterns Y_{target} $(y_{i,j})$, the weights are adjusted to reduce this error, until the desired error accuracy level is achieved. This adjustment process continues iteratively until one gets the weight matrix that enable us attain the desired error accuracy level.The resulting weights at this point are then used on data that is not used in the training data. That is, it is used on the validation data and test data.At the validation level, if the result is not satisfactory, one goes back to the beginning of the training and make adjustments as is deemed necessary.

It is known that the performance of a trained RBF network depends on certain parameters such as the number and location of Radial Basis Functions (RBF) centres, the output weights

along with the number of neurons in the hidden layer (Rivas *et al*., 2004; Awad, 2010). It has been established that the performance of RBF suffers degradation when the desired locations of the centres of the RBF are not suitable (Song *et al*., 2005). As long as the centres and the radii have been fixed, the weights of the links between the hidden and the outputs layers can be obtained.

For the past three decades, there has been active research work on using RBFNN to forecast time series data.The next section presents a review and critique of existing work related to this study.

2.4Related Work on Approaches to Full Optimization of RBFNN Model for Time Series Forecasting.

This section reviews existing work on optimizingall the parameters of the RBFNN in order to find their global optimum. Time series forecasting from diverse domains areconsidered.

Awad *et al*., (2009) proposed a method of optimizing the parameters of the RBFNN. Theauthors used well-known heuristics: the k-Nearest Neighbour technique (kNN) for the initialization of the radius of each RBF, Singular Value Decomposition (SVD) to directly optimize the weights. Finally, the Levenberg-Marquardt algorithm was used to fine-tune the obtained RBFNN.The constraint in this approach is that the number of clusters must be stated or determined apriori.

Rivas*et al*., (2002) used the evolutionary algorithm to optimize all the parameters related to the neural network architecture. A set of parameters to run the algorithm was found and tested against a set of different problems on time-series forecasting and function approximation. Results obtained were compared with those yielded by similar methods. The strength of this approach is that it can automatically determine the RBF centres.

Nekoukar and Beheshti (2009) presented a Local Linear Radial Basis Function Neural Network (LLRBFN). A modified Particle Swarm Optimization (PSO) with hunter particles was introduced for training the LLRBFN. The proposed methods have been applied for prediction of financial time-series and the result shows the feasibility and effectiveness. However, in this approach, the constraint is that thenumber of clusters must be known in advance.

Zang *et al*., (2008) adopted RBF neural network to model univariate and multivariable time series. The comparative analysis of the results from the forecast showed that multivariable

time series model has higher predictive accuracy for the landslide displacement than the univariatemodel. Thelimitation in this approach is that thenumber of clusters must be stated before clustering starts.

Shen*et al*., (2011) usedK-means clustering algorithm optimized by Artificial Fish Swarm Algorithm (AFSA) in the learning process of RBF. To verify the usefulness of algorithm, the authors compared the forecasting results of RBF optimized by AFSA, Genetic Algorithms (GA) and Particle Swarm O(PSO), as well as forecasting results of ARIMA, BP and Support Vector Machine (SVM). Of all the combinations considered in their paper, BIAS6+MA5+ASY4 was the optimum group with the least errors. The constraint in this approach is the number of clusters must be stated beforeclustering starts.

Isimeto *et al*., (2015) proposed an improved radial basis function neural network based on a convex cost function for rainfall forecasting. The network was trained by CGD and PSO algorithms. The proposed model predicts the occurrence of rainfall in a day with 72.68% accuracy, given weather information about the previous day.This approach figures out the RBF centres automatically.

Lin*and* Chen (2005) proposed a time-series forecasting model based on the radial basis function network (RBFN) and Self-Organizing Map (SOM). SOM was used to figure out the radial basis centres. The proposed model was examined using simulated time series data and actual groundwater head data. This approach figures out the RBF centres automatically.

Ko and Lee (2009) developed the radial basis function neural network (RBFNN) based on a Nonlinear Time-Varying Evolution Particle Swarm Optimization (NTVE-PSO) algorithm. When training RBFNNs, the NTVE-PSO method is adopted to determine the optimal structure of the RBFNN to predict time series, in which the NTVE-PSO algorithm is a dynamically adaptive optimization approach using the nonlinear time-varying evolutionary functions for adjusting inertia and acceleration coefficients. The proposed PSO method will expedite convergence toward the global optimum during the iterations. To compare the performance of the proposed NTVE-PSO method with existing PSO methods, the different practical load types of Taiwan power system (Taipower) were utilized for time series prediction of one-day ahead and five-days ahead. Simulation results illustrate that the proposed NTVE-PSO-RBFNN has better forecasting accuracy and computational efficiency for different electricity demands than the other PSO-RBFNNs.

Chao and Horng (2014) proposed a new algorithm called Firefly RBF network for training the radial basis function neural network, and this was tested on classification problems. Though a new approach for optimizing the RBFNN, their focus however, is on a classification task while our own focus is on a regression task.

Gan *et al.,* (2012) proposed a novel hybrid algorithm for selecting automatically the proper input variables, the number of hidden nodes of the radial basis function (RBF) network, as well as the optimization of the network parameters (weights, centers and widths) simultaneously. In the proposed algorithm, the inputs and the number of hidden nodes of the RBF network are represented by binary-coded strings and evolved by a genetic algorithm (GA). The performance of the presented hybrid approach is evaluated by several benchmark time series modeling and prediction problems. Experimental results show that the proposed approach produces parsimonious RBF networks, and obtains better modeling accuracy than some other algorithms. Optimization of the RBFNN parameters is a one-phase process as all the parameters were optimized simultaneously at once. This makes this approach to be more computationally expensive compared to our approach which entails a 2-stage optimization process. Also curse of dimensionality may occur as the number of parameters to be optimized at once are many.

Duand Zhang (2008) presented a new encoding scheme for training radial basis function (RBF) networks by genetic algorithms (GAs). In the proposed encoding scheme, both the architecture (numbers and selections of nodes and inputs) and the parameters (centres and widths) of the RBF networks are represented in one chromosome and evolved simultaneously by GAs in order that the selection of nodes and inputs can be achieved automatically. The performance and effectiveness of the presented approach are evaluated using two benchmark time series prediction examples and one practical application example, and are then compared with other existing methods. It is shown by the simulation tests that the developed evolving RBF networks are able to predict the time series accurately with the automatically selected nodes and inputs. Though a good approach, optimizing the whole parameters simultaneously is computationally more expensive compared to our approach. Also curse of dimensionality may occur as the number of parameters to be optimized at once are many.

Sheta and De Jong(2001) proposed an AutoRegressive Radial Basis Function (AR-RBF) model.GA was used to simultaneously optimize all of the RBF parameters so that an effective time-series model was designed. The model used for forecasting the exchange rates time series data, showed promising results.The limitation in this approach is that tuning the

RBF parameters to get the centre and weight was done via trial and error. Optimizing all the parameters at once is computationally expensive.

Rivas *et al*., (2004) followed up on Sheta and De Jong (2001). They proposed the evolvingRadial Basis Function (Ev-RBF) model. Theparameters of radial basis function neural networks (number of neurons, their respective centres and radii) were determined automatically using an evolutionaryalgorithm, the genetic algorithm. The weights were calculated using singular vector decomposition (SVD).Tested on currency exchange rates data, theresults obtained showed an improvement when compared with existing work of Sheta andDe Jong (2001).Our own work followed up on these authors' work. Thelimitation in Rivas *et al*.,'s approach is that it is more computationally expensive compared to our approach, as many parameters were optimized at once.Also curse of dimensionality may occur as the number of parameters to be optimized at once are many.Our own approach entails a 2-stage optimization process.

From existing work done so far, the limitations are summed up in the next section.

2.5 Summary of Limitations of Related Work on Approaches to Full Optimization of RBFNN Models for Time Series Forecasting.

A considerable amount of success has been achieved using the conventional and SI techniques to optimize fully the parameters of the RBFNN model for time series forecast problems. It was noted thatsome of the approaches proposed, for instance(Zhang *et al*., 2010; Armano and Farmani,2014) still depended on knowing the number of clusters before the clustering process starts. Other approaches by Sheta and De Jong (2001),Zhu(2009) used trial and errormethod to determine the number of clusters. Only a fewnumber of researchers (Rivas *et al*., 2004;Duand Zhang,2008;Gan *et al*., 2012) found their cluster centres automatically. However, these approaches are more computationally expensive compared to our approach as the whole parameters were optimized simultaneously.

From the foregoing, the main challenge in full optimization of the RBFNN parameters is in clustering involving optimal determination of the RBF centres. The next section focuses on the limitations of the conventional and current approaches to clustering with emphasis on the CGSO.

2.6 Limitations of related work on approaches to Clustering

Various clustering algorithms have been used by researchers to select optimal centre sets for the RBFNN. Most conventional clustering algorithms, including K-means, experience premature convergence and achieve local optimal solutions. They are not guaranteed to converge to the global optimum. It has been established that the performance of RBF suffers degradation when the desired locations of the centres of the RBF are not suitable (Song *et al*., 2005). Thisresults in local optimal network with low prediction precisions (Awad *et al*.,2009).

As a contribution to solving the clustering problem, Evolutionary Algorithms emerged. The Genetic Algorithm (GA) was used by (Awad, 2010) to cluster the RBF centres. The GA partially optimized the centres.The emergence of Swarm Intelligence techniques led to the development of the following methods which were used to solve the clustering problem:the kABC clustering algorithm proposed by Armano and Farmani(2014); the Artificial Bee Colony (ABC) algorithm by Zhang *et al*., (2010);and the Ant Colony Optimization algorithm by Shelokar *et al*.,(2004). However, the need to specify the number of clusters in advance remained a disadvantage. Ben-David (2014) noted that asserting when the optimal cluster centre sets have been achieved during the clustering process is a major challenge, as there is no ground truth with which to evaluate the resulting clusters.

This challenge was solved with the development of the Glowworm Swarm Optimization (GSO) algorithm meant for multimodal optimization (Krishnand and Ghose, 2005).Though GSO was first used by Huangand Zhou(2011) for cluster analysis,Aljarah and Ludwig(2013) adapted the GSO to obtainthe Clustering Analysis based onGlowworm Swarm Optimization (CGSO) algorithm. The CGSO does not require providing the number of clusters in advance as it can automatically discover the number. It alsosolves the problem of slow convergence. Survey shows that CGSO algorithm has not yet been used for RBFNN optimization(Karegowda and Prasad, 2013).

2.6.1 Limitationsof CGSO Algorithm

A critical review of Aljarah and Ludwig (2013) revealed that the CGSO algorithm suffers from the following limitations:

1) Method of Determination of the sensor range: CGSO has one parameter, the sensor range, r_s, which decides the number of clusters as well as the cluster quality.It was noted that the $r_s$was determined via preliminary experiments by trial and error. This approach is generally inefficient and lacks documentation.

2) Method of Initialization: CGSO initializes the population of glowworms byrandomly generating a collection of position vectors. This approach does not guaranteethat each glowworm covers a data instance.

2.7 Theories and Concepts used in this study

2.7.1 The Concept of Clustering

(i) Clustering:otherwise known as cluster analysis is the process of partitioning a set of data objects (or observations) into subsets called clusters. It aims at representing large datasets by a fewer number of prototypes or clusters. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in another cluster (Han *et al*., 2012; Abraham *et al*., 2008). Clustering is a challenging, dynamic field of research in data mining. It brings simplicity in modeling data and hence plays a major role in the process of knowledge discovery and data mining (Abraham *et al*., 2008). It is linked to unsupervised learning in machine learning. Its application area is not limited to only pattern recognition and web search, it could also be used as a standalone data mining tool to gain insight into data distribution. It could be used as a pre-processing step either, for other data mining algorithms (such as characterization, attribute subset selection, and classification), which would act on the detected clusters (Han *et al*., 2012); or as a pre-processing step before later stage of regression (Alpaydin, 2010).

Ben-David (2014) asserted that "there exists distressingly little theoretical understanding of clustering. In most practical clustering tasks, there is no clear ground truth to evaluate your solution by (in contrast with classification tasks, in which you can have a hold-out labeled set to evaluate the classifier against)".

Literature notes that there are many clustering algorithms in existence. As affirmed by (Han *et al*., 2012), providing a crisp categorization of clustering methods is difficult because these categories may overlap and at such, a method may have features from several categories. As highlighted by Alpaydin (2010), methods of clustering could be loosely split into two groups: the online method and Batch method. Yet some other authors Karayiannis andRandolph-Gips(2003); Ruiwang and Binwang(2002) noted that the strategies for selecting the RBF

centres could be classified as follows: (i) strategies selecting the RBF centers randomly from the training data, (ii) strategies employing unsupervised procedures for selecting the RBF centers, and (iii) strategies employing supervised procedures for selecting the RBF centers.

(ii) Online methods: For this method, the whole sample data are not available at hand during training. The instances are received one by one and then the model parameters are updated as soon as the instances are received. Examples of online methods include: (1) competitive methods which are neural network methods for online clustering. Included in this category are online k-means, two neural network extensions: Adaptive Resonance Theory (ART) and self organizing map (SOM).

(iii) Batch methods: For this method, the whole sample data are available at hand during training the clusters. These methods include: batch K-means, Expectation-maximization algorithm, Hierarchical clustering, andOrthogonal least square learning algorithm.

The advantages of online methods include (1) we do not need extra memory to store the whole training set; (2) updates at each step are simple and easy to implement, for example, in hardware; and (3) the input distribution may change in time and the model adapts itself to these changes automatically. In comparison with batch methods, one would have to collect a new sample and run the batch method from scratch over the new sample (Alpaydin, 2010).

2.7.2 Approaches to Clustering

Some approaches to clustering include the following:

2.7.2.1RandomSelection of Clustersis the simplest clustering technique that uses unsupervised method. This approach randomly selects a number of training examples as RBF centers.

This method has the advantage of being very fast, but the network will likely require an excessive number of centers. Once the center positions have been selected, the spread parameters σ_j can be estimated, for instance, from the average distance between neighboring centers. The limitation of this method is that selecting the Radial basis function centres is not guided by the mean square error objective function (Gutierrez-Osuna, 2014).

2.7.2.2 Partitioning Methodisconsidered as the simplest and most fundamental method of cluster analysis. This method organizes the objects of a set into several exclusive groups or clusters of spherical shape. They are distance-based, use the mean to represent each cluster. Formally, given a data set, D, of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions (k<=n), where each partition

represents a cluster. These clusters are formed to optimize an objective partitioning criterion such as a dissimilarity function based on distance, so that the objects within a cluster are "similar" to one another and "dissimilar" to objects in other clusters in terms of the data set attributes. They are effective for small to medium size datasets (Han et al., 2012). The most commonly used partitioning algorithm is the **k-means** which is discussed below.

K-means Algorithm is the most well known and commonly used partitioning method. It is an unsupervised technique used for clustering. That is, it is used to find groups in the data, where the groups are represented by their clusters, which are the typical representatives of the groups (Alpaydin, 2010). As affirmed by Han *et al*., (2012); K-means algorithm is a centroid-based partitioning technique that uses the centroid of a cluster, Ci. The centroid of a cluster is its centre point and this can be defined by the mean or medoid that is used to represent the cluster centre.

The quality of a cluster Ci as noted by Han *et al*.,(2012); Alpaydin (2010) can be measured by the within-cluster variation, which is the sum of squared error between all objects in Ci and the centroid ci; is defined by an objective function

$$E = \sum_{i=1}^{k} \sum_{p \varepsilon Ci} dist(p,ci)^2 \qquad (3)$$

where **E** is the sum of the squared error for all objects in the dataset; p is the point in space representing a given object; ci is the centroid of the cluster Ci (both p and ci are multidimensional). The difference between an object pϵCi and ci, the representative of the cluster, is measured by dist(p,ci), which is the Euclidean distance between two points.

In other words, for each object in each cluster, the distance from the object to its cluster centre is squared, and the distances are summed. This objective function tries to make the resulting k clusters as compact and as separate as possible (Alpaydin, 2010). In simple terms, this implies an objective function is used to assess the partitioning quality so that objects within a cluster are similar to one another but dissimilar to objects in other clusters, thus ensuring that the objective aims for high intracluster similarity and low intercluster similarity.

It is noted that optimizing the within-cluster variation is computationally challenging and a NP-hard problem, but k-means is commonly used to overcome the prohibitive computational cost.

The K-means algorithm is shown in Algorithm 1 (Han *et al*., 2012)

Algorithm 1: K-means Algorithm

Input:

K: the number of clusters

D: a data set containing n points

Method:

(1) arbitrarily choose k points from D as the initial cluster centres

(2) repeat

(3)(re)assign each point to the cluster to which the points is the most similar,
based on the mean value of the points in the cluster;

(4)Update the clusters means, that is, calculate the mean value of the points foreachcluster

(5) until no change

Output: A set of k clusters.

While the strengths of K-means include the following: K-means finds mutually exclusive clusters. It is distance-based and uses the mean to represent cluster centre. It is effective for small to medium –sized data sets. Its limitations are:K-means is not guaranteed to converge to the global optimum, hence it may experience premature convergence; and often it terminates at a local optimum. The need to specify the number of cluster, k, in advance can be seen as a disadvantage, as the k-means is sensitive to the initial number of centroids

2.7.2.3Hierarchical Clustering

This method works by grouping data objects into a hierarchy or "tree" of clusters. Grouping data in this form is useful for data summarization and visualization. As asserted by Alpaydin (2010), Hierarchical clustering are methods of clustering that only use similarities of instances, without any other requirement on the data; the aim is to find groups such that instances in a group are more similar to each other than instances in different groups. It has application in areas such as handwritten character recognition, studies of evolution, in applications that the data bear an underlying hierarchical structure that needs to be unrecovered, and so on(Han *et al.,* 2012; Frigui and Krishnapuram, 1999; Leung *et al*., 2000).

Han *et al*., (2012) notes that several orthogonal ways to categorize hierarchical clustering methods include algorithmic methods, probabilistic methods, and Bayesian methods. Algorithmic methods consider data objects as deterministic and compute clusters according to the deterministic distances between objects. Examples of these methods include: Agglomerative (bottom-up), divisive (top-down), and multiphase methods. Probabilistic methods use probabilistic models to capture and measure the quality of clusters by the fitness of models. An example is Probabilistic hierarchical clustering. Bayesian methods compute a distribution of possible clusterings. That is, conditional on the given data; they return a group of clustering structures and their probabilities, instead of a single deterministic clustering.

The strengths of hierarchical methods include firstly, the number of classes need not be speci ed a priori and secondly, they are independent of the initial conditions. It is possible to improve the clustering quality of hierarchical methods by integrating these methods with other clustering techniques, resulting in multiphase clustering. Examples of multiphase clustering methods are: Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) and Chameleon (Han *et al*., 2012). Its limitation is that this method can encounter difficulties regarding the selection of merge or split points. This issue is critical because once a group of objects is merged or split, the process at the next step will operate on the newly generated clusters. It will not undo what was done previously and will not perform object swapping between clusters, that is, it cannot correct erroneous merges or splits. Data-points assigned to a cluster cannot move to another cluster. Thus, merge or split decisions, if not well chosen, may lead to low-quality clusters. Moreover, the method do not scale well because each decision of merge or split needs to examine and evaluate many objects or clusters.

2.7.2.4 Density-Based Methods:these methods (Han et al., 2012) discover clusters that are of non spherical and arbitrary shape such as “S” shape and oval clusters. This method model clusters as dense regions in the data space, separated by sparse regions.
The strength of this method is that this method can identify convex regions, where noise or outliers are included in the clusters.

2.7.2.5Bio-Inspired Clustering Algorithms

Most conventional clustering algorithms experience premature convergence andachieve local optimal solutions. Research efforts in clustering to overcome this problem are well noted. Abraham *et al*., (2008) asserts that these days, data mining tasks require fast and accurate

partitioning of huge datasets, which may come with a variety of attributes or features. This fact imposes severe computational requirements on the relevant clustering techniques. A family of bio-inspired or nature-inspired algorithms, well-known as Swarm Intelligence (SI) techniques has recently emerged that meets these requirements and has successfully been applied to a number of real world clustering problems. Clustering with swarm-based algorithms has emerged as an alternative to more conventional clustering methods(Handl and Meyer, 2007; Shifei *et al*., 2010).Swarm Intelligence are known to imitate the natural social communities such as ant colonies, fish schools, bird flocks. The behavior of these communities is based on the receptor of the individual's interactions by communicating with each other to locate the food sources (Engelbrecht, 2007).They locate global solution for the given optimization problem.

One improvement to solving the problem of local minimum was proposed by Awad, *et al*., (2009); who came up with a new method based on K-means and local displacement process which locally minimizes the distortion within each cluster. Another achievement was made by Armano Farmani (2014) who proposed the kABC clustering algorithm, a combination of K–means and ABC algorithms. Their simulation results showed that kABC has more ability to search for the global optimum solutions, and more ability for passing local optimum, as it converges to optimal solution in most runs. The authors noted that to use the KABC, the number of clusters should be known a prior. This is a major limitation as it is not easy to know the number of clusters in advance.

To solve the problem of slow convergence and the problem of determining the number of clusters in advance, a new **C**lustering approach based on **G**lowworm **S**warm **O**ptimization algorithm (CGSO) was proposed by Aljarah and Ludwig in Aljarah and Ludwig (2013). The CGSO is a modification to the classical GSO proposed by Krishnanand and Ghose (2005), first used for optimizing multimodal functions. With this modified GSO, CGSO does not need the number of clusters to be provided in advance as it can automatically discover number of clusters in advance and it tackles slow convergence problem. A performance analysis of CGSO and other clustering algorithms was done by Aljarah and Ludwig (2013) and their experimental results on several real and artificial data sets with different characteristics showed that their proposed algorithm, CGSO is more efficient compared to other well-known clustering methods (such as k-means, hierarchical clustering).

(1) The Basic Glowworm Swarm Optimization Algorithm

The classical Glowworm **S**warm Optimization (GSO) algorithm was rst presented byKrishnanand and Ghose in 2005(Krishnanand and Ghose, 2005) to model the collective behavior in robotics.The classical GSO locates multiple solutions having different or equal objective function values. This feature of GSO distinguishes it from other optimization techniques (that find one local or global solution).

In GSO, each glowworm uses a probabilistic mechanismto select a neighbour having higher luciferin value, and move towards it. Each glowworm carry its own luciferin value and has its own decision range. The luciferin value depends on the objective function value and glowworm position. A glowworm with a better position is brighter than others, has higher luciferin level value and is very close to one of the optimal solutions.All glowworms are attracted,and move to neighbour within their neighbourhood range, that glow brighter. Based on local information and interactions with selected neighbour, the swarm of glowworms move and divide themselves into disjoint subgroups that eventually converge to multiple local optima of a given multimodal function (Rossato de Oliveira *et al*., 2013; Aljarah and Ludwig, 2013).

GSO begins by populating a given search space with $n$glowworms of dimension d. Each glowworm is assigned a random position inside the search space. Initially,allglowwormsare assigned same, initial amount of luciferin l_0 and neighbourhood range decision r_0.The position $\overline{x}_\iota$ of each glowworm is evaluated by a fitness function $f(\overline{x}_\iota)$. During each iteration, the luciferin and position of each glowworm gets updated. GSO algorithm requires other parameters namely: step size (s), sensor range (r_s), luciferin decay constant (ρ), luciferin enhancement constant (γ), number of neighbors (n_t) and a constant value (β).

The GSO algorithmis shown in Algorithm 2(Rossato de Oliveira *et al*., 2013)

Algorithm 2: The GSO algorithm.

1: Set parameters: $n, l_0, r_0, \rho, \gamma, \beta, s, r_s, n_t$

2: Randomly generate the population of glowworms $\overline{x}_\iota$
3: **for** $i = 1$ **to** n **do**
4: Initialize luciferin $l_i(0) = l_0$
5: Initialize neighborhood range $r_d^i(0) = r_0$
6: **end for**
7: $t = 1$
8: **while** stop condition not met **do**
9: **for** each glowworm$i$**do** [update luciferin]
10: $l_i(t+1) = (1 \quad \rho).\, l_i(t) + \gamma.\, f(x_i(t))$
11: **end for**

12: **for** each glowworm$i$**do** [movement phase]
13: Find neighbors $N_i(t)$
14: **for** each glowworm$j \in N_i(t)$**do**
15: Compute probability $P_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)}$
16: **end for**
17: Select glowworm$j$ using P_{ij}
18: Update glowworm position with
$x_i(t+1) = x_i(t) + s\left[\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right]$
19: Update decision range:
20: $r_d^i(t+1) = min\left\{r_s, max\{0, r_d^i(t) + \beta.(n_t \quad |N_i(t)|)\}\right\}$
21: **end for**
22: $t = t + 1$
23: **end while**

(ii) Clustering Analysis Based on Glowworm Swarm Optimization algorithm

The classicalGSO can capture multiple peaks in multimodal functions; this advantage of GSO algorithm was exploited by (Aljarah and Ludwig, 2013) to produce the Clustering Analysis Based on Glowworm Swarm Optimization (CGSO) algorithm. That is, due to the GSO algorithm's ability for multimodal optimization, it was adapted to obtain the CGSO to solve the clustering problem.GSO locates multiple solutions while other optimization techniques find one local or global solution. In CGSO, the objective function is not defined by the user. In fact, it is an integral part of the algorithm and can significantly decide the cluster quality. According to the authors, the objective function is adjusted to locate multiple optimal centroids such that each centroid represents a sub-solution and the combination of these sub-solutions formulates the global solution for the clustering problem.

Applied to cluster datasets, the CGSO can find the number of clusters, as it does not need to be provided with the number of clusters in advance. It can automatically discover the number of clusters.

In CGSO, the clustering problem is formulated as a multimodal optimization problem to extract the centroids from a data set based on glowworms' movement. The CGSO partitions the given datasets into sets of clusters such that every glowworm in the swarm tries to cover larger numbers of data instances. Furthermore, eachglowworm basically gets attracted to glowworms that cover a larger amount of data instances.

The swarm ofglowworms consists of m glowworms, where each glowworm is represented by a vector, g_j, j= 1 . . . m. Each g_j has 5 parameters**:**luciferin level(L_j), fitness function F_j, d-

dimensional position vector(P_j), coverage set (Cr_j) which is the set of data instances covered byg_j, and intra-distance (IntraD_j) between the (Cr_j) set members and g_j position. The g_j needs to cover at least one data instance in its neighbourhood range.

The CGSO Algorithmconsists of four main phases: initialization phase, luciferin level update, glowworm movement, and candidate centroids set construction.

At the initialization phase, first an initial glowworm swarm of size m is created. For every glowworm g_j, a random position vector is generated within the search space within the bounds of the minimum and maximum values of the dataset (line 2). Using the initial luciferin level L_0, the luciferin levelL_j is initialized (line 4). The fitness function F_j is initialized to zero. The neighbourhood range r_s is set to an initial constant range r_o. Secondly, after initializing the swarm, the set of data instances Cr_jcovered byg_j is extracted from the dataset X (line 5), and the IntraD_j is calculated using equation (in line 6); where cr_{ji} is data instance i covered by g_j; $|cr_j|$ is number of data instances covered byg_j. The last step in the initialization phase entails the calculation of the swarm level fractions SSE and InterDist.

To initialize the SSE, the glowworms list that covered the highest number of data instances (the glowworms that have the maximum $|cr_j|$sizes) is extracted. These glowworms should be disjointed from each other. The extracted glowworm list is considered the initial set of the candidate centroid c. Next, the candidate centroid $c$is used to calculate the initial SSE (the Equation in line 9). The same candidate centroid c is used to calculate the InterDist (the Equation in line 10).

After the initialization phase, an iterative process takes place to find optimal glowworms that represent the clustering problem centroids. The result of each iteration is an updated swarm with updated candidate centroid set c. Firstly, the fitness function F is evaluated to assign new F_j values for each glowworm using the glowworm position and other information (line 13). Three fitness functions were proposed to evaluate the goodness of the glowworm. After the fitness functions evaluation for glowwormg_j, the luciferin level L_j is updated using (the Equation in line 18). Then, each glowwormg_j locates the neighbourhood group (line21-22), and the neighbour probability values are calculated using Equation in line 23; and using the roulette wheel selection method in line 25, the best neighbour is then found. Next, the glowworm is moved towards the best neigbour by updating its position vector using

Equationin line 26. After that,$|cr_j|$ and IntraDj are updated (using Equation in line 28) based on the new glowworm $g_j$positions.

Next is that the candidate centroid set c is reconstructed based on the highest fitness values (F_j) (from line 30), and not like the way they were extracted during the initialization phase, which was based on the highest number of data instances (the glowworms having the maximum $|cr_j|$). Then, the candidate centroid set c is used to calculate the new value for SSE which is calculated by the Equation in line 31. Also, the same candidate centroid set c is used to calculate InterDist which is calculated from the Equation in line 32. Using the new information, the fitness function is reevaluated in line 35. The iterative process continues until the size of the candidate centroid set c becomes less than a specific threshold (minimum number of centroids is stated), or the maximum number of iterations is achieved. The candidate centroid set c decreases throughout the iterative process.After the clustering process is completed, the candidate centroid set in line 39 is used to evaluate the clustering results.

The CGSO algorithm(Aljarah and Ludwig, 2013) is Algorithm 3

Algorithm 3: The CGSO algorithm

Input: a dataset X consisting of d data instances with $n$dimensions

Method:
//Initialization phase
1: Set parameters: $m,\ L_0,\ \rho,\ r_s, \gamma, s$
2: Generate the initial population of glowworms by randomly generating positions vectors within the search space.
3: **for** each glowworm g_j, where j = 1 **to** m **do**
4: Initialize luciferin level, $L_j(0) = L_0$
5: Extract set of data instances Cr_j covered by g_j
6: Calculate intra-distance $\boldsymbol{IntraD_j} = \sum_{i=1}^{|cr_j|} \| cr_{ji} \quad g_j \|$
7: **end for**
8: Extract the initial set of the candidate centroid,c. The initial centroid is a list of glowworms with maximum $|cr_j|$ sizes and also disjointed from one another.
9: Calculate $\boldsymbol{SSE} = \sum_{j=1}^{|C|} \sum_{i=1}^{|C_j|} \| c_j \quad g_i \|^2$
10: Calculate $\boldsymbol{InterDist} = \sum_{i=1}^{k} \sum_{j=i}^{k} \| c_i \quad c_j \|^2$
11: **for** each glowworm g_j, where j = 1 **to** m **do** {initializing fitness function}
12: Calculate fitness function
13: $\boldsymbol{F_j(g_j)} = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}\ Or\ \frac{interDist \times \frac{1}{n}|cr_j|}{\frac{intraD_j}{\max_j(intraD_j)}}\ Or\ \frac{interDist \times \frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}$
14: **end for**

15: $t = 1$
16: **while** stop condition not met **do**
17: **for** each glowworm g_j, where j = 1 **to** m **do** {update luciferin}
18: $L_j(t+1) = (1 \quad \rho).L_j(t) + \gamma.F_j(g_j)$
19: **end**
20: **for** each glowworm g_j, where j = 1 **to** m **do** {movement phase}
21: Find neighbors $N_J(t)$
22: **for** each glowworm $i \in N_j(t)$**do**
23: Compute probability $P_{ji}(t) = \frac{L_i(t) - L_j(t)}{\sum_{k \in N_j(t)} l_k(t) - l_j(t)}$
24: **end for**
25: Select glowworm j using P_{ji} by roulette wheel method
26: Update glowworm position with

$$g_j(t+1) = g_j(t) + s\left[\frac{g_i(t) \quad g_j(l)}{\|g_i(t) \quad g_j(t)\|}\right]$$

27: Extract set of data instances $\mathbf{Cr_j}$ covered by g_j
28: Calculate intra-distance $\boldsymbol{IntraD_j} = \sum_{i=1}^{|cr_j} \|cr_{ji} \quad g_j\|$
29: **end for**
30: Extract the set of the candidate centroids, c. The centroid is a list of glowworms with maximum fitness function and also disjointed from one another.
31: Calculate $\boldsymbol{SSE} = \sum_{j=1}^{|C|} \sum_{i=1}^{|C_j|} \|c_j \quad g_i\|^2$
32: Calculate $\boldsymbol{InterDist} = \sum_{i=1}^{k} \sum_{j=i}^{k} \|c_i \quad c_j\|^2$
33: **for** each glowworm g_j, where j = 1 **to** m **do** {fitness function update}
34: Calculate fitness function
35: $\boldsymbol{F_j(g_j)} = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}$ Or $\frac{interDist \times \frac{1}{n}|cr_j|}{\frac{intraD_j}{\max_j(intraD_j)}}$ Or $\frac{interDist \times \frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}$
36: **end for**
37: $t = t + 1$
38: **end while**
39: Return the set of candidate centroids, c
Output: A set of **k** clusters and **k** centroids

2.7.3Approaches to Training the Network weights

Training the network weightsinvolve the use of optimization algorithms to adjust the output weights of the RBFNN, in order to get the right set of weights that minimize the objective function. In the next sub-section, a brief survey of some training algorithms used in training RBFNNmodel is done. The conventional techniques and the bio-inspiredswarm intelligence optimization techniques will be looked at.

2.7.3.1 Conventional Techniques

(i)Gradient Descent Algorithm

Gradient Descent Algorithmis otherwise called the steepestdescent, or the method of steepest descent. It is a technique used to find a local minimum of a function. The Minimum of a function is found by following the slope of the function.

To apply this algorithm, one starts with an initial guess of the solution and takes the gradient of the function at that point. Then one steps the solution in the negative direction of the gradient and repeats the process. The algorithm eventually converges at the point where the gradient is zero (which corresponds to a local minimum). It is a first-order algorithm because it takes only the first derivative of the function.TheGradient Descent algorithmisshown in Algorithm 4

Algorithm 4: Gradient Descent algorithm

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step

Update

Until stopping criterion is satisfied

The limitation of this algorithm is that Gradient Descent finds the nearest minimum that can be a local minimum, and there is no guarantee of finding the global minimum unless the function has only one minimum. The use of good value for the step size is critical. If it is too small; the convergence may be too slow, and a large value may cause oscillations and even divergence (Alpaydin, 2010).

2.7.3.2 Swarm Intelligence Techniques

(i) Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic optimization technique based on movement and intelligence of swarm of birds. PSO was presented by Kennedy and Eberhart in 1995 in (kennnedy and Eberhart, 1995). It applies the concept of social interaction to problem solving. Each individual in the swarm, called a particle is treated as a point in space, and represents a potential solution. Each particle in the swarm as noted by Binitha and

Sathya(2012) represents a solution in a high-dimensional space with four vectors, its current position, best position found so far, the best position found by its neighborhood so far and its velocity and adjusts its position in the search space based on the best position reached by itself (pbest) and on the best position reached by its neighborhood (gbest) during the search process. In each iteration, each particle updates its position and velocity.

The simplicity of implementation, quick convergence, and few parameters have resulted in PSO gaining popularity. Many researchers have made modifications to the PSO. PSO has been used for problems across various applications such as image classification by Omran *et al*., (2002); pattern classification, biological system modeling, scheduling, signal processing, robotic application by Hardin *et al*., (2004); and for training neural networks by Engelbrecht (2007).

Originally, two PSO algorithms were developed which differ in the size of their neighbourhoods. These are the global best (gbest) PS0 and the local best (lbest) PSO. For the gbest PSO, the neighbourhood for each particle is the entire swarm while for the lbest, smaller neighbourhoods are defined for each particle. Basic PSO has been shown to outperform optimizers such as gradient descent, scaled conjugate descent and genetic algorithm. Although the PSO algorithm has been proven to be effective, its theoretical foundation is rather weak. The gbest PSO algorithm is faster but might converge to local optimum for some problems. The lbest PSO is a little bit slower but not easy to be trapped into local optimum (Engelbrecht, 2007).

A step by step overview of how the basic global best PSO algorithm is used in training RBFNN entails the following procedures: (1) initialize the particles, including the value assignment for hidden center vector, base width vector, weight of the network; 2) Calculate the fitness value (best solution) of each particle, and make the current position of the particle as the individual's maximum *pbest*, find out the particle of the minimum fitness value, and make it the initial global best, *gbest*; 3) Compare fitness value of the current particle with *pbest*, if the fitness value of the current one is smaller, then update *pbest* with the current fitness value; 4) For each particle, compare *pbest* with *gbest*, if *pbest* is better, then update *gbest*; 5) Update the speed and location of the particle . Repeat steps 4)-6), until the terminal condition is met, which is either to meet the maximum iterations or the error accuracy requirement; 7) Set *gbest* as the parameter of the RBF neural network (Shuai, 2013). The gbest PSO algorithm(Engelbrecht, 2007) isAlgorithm 5

Algorithm 5: gbest PSO algorithm

Create and initialize an n_x-dimensional swarm;

repeat

for each particle i =1,...,n_s do

//set the personal best position

if $f(x_i) < f(y_i)$ **then**

$y_i = x_i$;

end

//set the global best position

if $f(y_i) < f(\hat{y})$ then

$\hat{y} = y_i$;

end

end

for each particle i =1,...,n_s do

update the velocity using

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) \quad x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) \quad x_{ij}(t)]$$

update the position using

$x_i(t+1) = x_i(t) + v_i(t+1)$

end

until stopping condition is true;

(ii) Bioluminescence Swarm optimization

Bioluminescence Swarm optimization (BSO) is a new swarm-based evolutionary approach based on the bioluminescence behavior of fireflies. The BSO algorithm can be loosely seen as a hybrid between PSO and GSO, but with some unique features.The BSO uses two basic characteristics of the Glowworm Swarm Optimization (GSO) algorithm proposed by Krishnanand & Ghose, (2005); the luciferin attractant, and the stochastic neighbor selection. BSO goes further introducing new features such as: stochastic adaptive step sizing, global optimum attraction, leader movement, and mass extinction. BSO is hybridized with two local search techniques: local unimodal sampling and single-dimension perturbation. All these features make BSO a powerful algorithm for hard optimization problems (Rossato de Oliveira, *et al*., 2011).While the concept of global optimum does not exist in the GSO algorithm, every particle inBSO is attracted to the global optimum, like PSO.

As in GSO and many other algorithms, the rst step is initializing n particles in the d-dimensional search space. All particles, de ned by $\overline{x}_i$ =[$x_{i1,}$ $x_{i2,}$, …, x_{id}], are evaluated by a tness function $f(\overline{x}_i)$, i = 1,...,n. BSO uses luciferin-based attraction instead of tness-based attraction between the particles, as proposed by the GSO. This process is controlled by the parameters ρ and γ, which are the luciferin decay constant and the luciferin enhancement constant, respectively. It also uses stochastic step size, similarly to PSO, instead of xed step as in the GSO. This step size also varies for each particle, according to its luciferin value, and controlled by the cs parameter.

The BSO algorithm (Rossato de Oliveira, *et al*., 2011) is Algorithm 6

Algorithm 6: The BSO algorithm

1: Set parameters: $n, \rho, \gamma, s_o, c_g, c_s, lR, eT$
2: Randomly generate the bioluminescent particle population w_i
3: for $i = 1$ to n do
4: Initialize luciferin $l_i(0) = 0$
5: end for
6: Find the global best $g(t)$
7: $t = 1$
8: while stop condition not met do
9: for each particle i do {updateluciferin}
10: $l_i(t+1) = (1 \quad \rho).l_i(t) + \gamma.f(w_i(t))$
11: end for
12: for each glowworm i do {movement phase}
13: Find neighbors $N_i(t)$
14: for each particle j $\in N_i$ (t) do
15: Compute probability $P_{ij}(t) = \frac{L_j(t) - L_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)}$
16: end for
17: Select glowworm j using P_{ij}
18: Update particle step size with $s = s_0 + \frac{1}{1 + c_s.l_i(t)}$
19: Update glowworm position with

$$\mathrm{w_i(t+1)} = \mathrm{w}_i(t) + rand.s.\left[\frac{\mathrm{w}_j(t) - \mathrm{w}_i(t)}{\|\mathrm{w}_j(t) - \mathrm{w}_i(t)\|}\right] + c_g.rand.s.\left[\frac{g(t) - \mathrm{w}_i(t)}{\|g(t) - \mathrm{w}_i(t)\|}\right]$$

20: Find the global best $g(t)$

21: if t%lR= 0 then {LocalSearchProcedures}
22: Perform strong local search on $g(t)$
23: else
24: Perform weak local search on $g(t)$
25: end if
26: if iterations without a new $g(t) = eT$ then {MassExtinction}
27: Reinitialize all particles but $g(t)$
28: end if
29: end for
30: $t = t + 1$
31: end while

CHAPTER THREE

METHODOLOGY

This chapter presents the research methodology used to achieve each of the stated objectives in Section 1.3. Optimization of the parameters of the RBFNN model involves clustering and weight optimization processes. The objectives 1,2 and 3 concern clustering and the methods used to achieve these objectivesare covered in Sections3.1, 3.2, and 3.3respectively.Objective 4 pertains to weightoptimization of the RBFNN model andthe method to achieve this is covered in Section 3.4. This sub-section presents the RBFNN, the proposed models: CGSOm-BSO and CGSOm-CGD RBFNN, as well as the techniques employed in developing these models for time series forecasting.

To address the first objective of efficiently determining the sensor range of the CGSO algorithm,an automated mechanism in the form of an algorithm for determining the value of $r_s$was developed, and this is discussed in Section 3.1.Next, the glowworm initialization method was modified as explained in Section 3.1.1 and a function that measures the cluster error during the iteration phase (in Section 3.1.3) was introduced into the CGSO.These modifications to the CGSO resulted to the CGSOm algorithm presented (in Section 3.1.2).The CGSO algorithm exploited the advantages of GSO algorithm.

3.1 Efficient determination ofLocal Sensor Range (r_s) of the CGSO algorithm

Conventionally, r_s is determined via preliminary experiments by trial and error. This is the same approach adopted by Aljarah and Ludwig,(2013). This approach is generally inefficient

and lacks documentation. In this work, we propose an algorithm for determining the value of r_s.

The basic idea behind the algorithm is that n_r samples of r_s are generated between its limits; each value of r_s is then evaluated to produce $f(r_s)$. Then a quadratic function is fitted on r_s and $f(r_s)$ data. The value of r_s at the turning point of the quadratic function becomes the required local sensor range. Among the various functions tried while fitting the $f(r_s)$ vs. r_s curve, the quadratic function was used because it converged to a turning point which is the minimum point unlike exponential function which gave a minimum that tends to infinity with no visible turning point. Using complicated exponential functions yielded multiple turning points and this would be computationally expensive to use. The algorithmwith the modified initialization is given in Algorithm 7.

Algorithm 7: Algorithm for determination of value of sensor range, r_s

1: Set parameterm, r_s^{min}, n_r
2: Generate the initial population of glowworms by randomly selecting data instances from datasets and assigning to each glowworm.
3: Compute mean of the data set, $\bar{x}$
4: Compute $r_s^{max} = \frac{\sum_i^k \|\bar{x} - x_i\|}{k}$
5: Initialize $P, Q, R, S, T, U, V, W, Z = 0$
6: **for**$i = 1$**to** n_r**do**
7: $r_s(i) = r_s^{min} + (i \quad 1)\frac{r_s^{max} - r_s^{min} + 1}{n_r}$//Derived from Langrange scale interpolation
8: **for** each glowwormg_j (where j = 1 **to** m) **do**
9: Extract set of data instances Cr_j covered by g_j
10: **end for**
11: Extract the set of the candidate centroids, C
12: Compute $\boldsymbol{SMSE} = \sum_{j=1}^{|C|} \frac{\sum_{k=1}^{|Cr_j|} \|cr_{jk} - \mathrm{g_j}\|^2}{|Cr_j|}$
//$|\boldsymbol{Cr_j}|$ = $\boldsymbol{No of data instances covered by glowworm g_j; SMSE = Cluster error function}$
13: Compute $f(r_s) = SMSE \times |C| + \frac{1}{|C|}$//$|C|$ = No. of clusters;$f(r_s)$= error at each iteration $r_s(i)$;
14: $P = P + r_s^4(i)$
15: $Q = Q + r_s^3(i)$
16: $R = R + r_s^2(i)$
17: $S = S + r_s(i)$
18: $T = T + r_s^2(i) \times f(r_s)$
19: $U = U + r_s(i) \times f(r_s)$
20: $V = V + f(r_s)$
21: **end for**
22: Compute $W = \frac{S}{R}$
23: Compute $Z = \frac{n_r}{R}$

24: Compute $r_s = \quad 0.5\frac{(Q-PW)(V-TZ)-(U-TW)(R-PZ)}{(U-TW)(S-QZ)-(R-QW)(V-TZ)}$ //$\boldsymbol{r_s}$ =**computed sensor range**

3.1.1Initialization of Glowworm

CGSO initializes the population of glowworms by randomly generating a collection of position vectors. This approach does not guarantee that each glowworm covers a data instance. CGSOm implements a new approach that initializes the glowworms. The initial population of glowworms are generated by randomly selecting data instances from datasets. This approach guarantees that each glowworm is within the search space and covers at least a data instance.

3.1.2 The Modified CGSO(CGSOm)

CGSOm, a modified version of CGSO, is proposed in this work. Two key aspects of CGSO are modified – determination of the value of sensor range, r_s and the initialization of glowworms. Also, a function that measures cluster error quality during the iteration phase was introduced.

The CGSOm algorithm is Algorithm 8

Algorithm 8: The CGSOm algorithm

Input: a dataset X consisting of d data instances with $n$dimensions

Method:

//Initialization phase
1: Set parameters: $m,\ L_0,\ \rho, \gamma, s, r_s^{min}, n_r$
2: Generate the initial population of glowworms by randomly selecting data instances from datasets and assigning to each glowworm.
3: Determine r_s//**From Sensor Range DeterminationAlgorithm in Algorithm 3**
4: **for** each glowworm g_j, where j = 1 **to** m **do**
5: Initialize luciferin level, $L_j(0) = L_0$
6: Extract set of data instances Cr_j covered by g_j
7: Calculate intra-distance $\boldsymbol{IntraD_j} = \sum_{i=1}^{|cr_j|}\|cr_{ji} \quad g_j\|$
8: **end for**
9: Extract the initial set of the candidate centroid,c. The initial centroid is a list of glowworms with maximum $|cr_j|$ sizes and also disjointed from one another.
10: Calculate $\boldsymbol{SSE} = \sum_{j=1}^{|C|}\sum_{i=1}^{|C_j|}\|c_j \quad g_i\|^2$
11: Calculate $\boldsymbol{InterDist} = \sum_{i=1}^{k}\sum_{j=i}^{k}\|c_i \quad c_j\|^2$
12: **for** each glowwormg_j, where j = 1 **to** m **do** {initializing fitness function}
13: Calculate fitness function

14: $\boldsymbol{F_j(g_j)} = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}\ Or\ \frac{interDist \times \frac{1}{n}|cr_j|}{\frac{intraD_j}{\max_j(intraD_j)}}\ Or\ \frac{interDist \times \frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}$

15: **end for**

16: $t = 1$

17: **while** stop condition not met **do**

18: **for** each glowwormg_j, where j = 1 **to** m **do** {update luciferin}

19: $L_j(t+1) = (1 \quad \rho).L_j(t) + \gamma.F_j(g_j)$

20: **end**

21: **for** each glowwormg_j, where j = 1 **to** m **do** {movement phase}

22: Find neighbors $N_J(t)$

23: **for** each glowworm$i \in N_j(t)$**do**

24: Compute probability $P_{ji}(t) = \frac{L_i(t) - L_j(t)}{\sum_{k \in N_j(t)} l_k(t) - l_j(t)}$

25: **end for**

26: Select glowworm$j$ using P_{ji} by roulette wheel method

27: Update glowworm position with

$$g_j(t+1) = g_j(t) + s\left[\frac{g_i(t) \quad g_j(l)}{\|g_i(t) \quad g_j(t)\|}\right]$$

28: Extract set of data instances $\mathbf{Cr_j}$ covered by g_j

29: Calculate intra-distance $\boldsymbol{IntraD_j} = \sum_{i=1}^{|cr_j} \|cr_{ji} \quad g_j\|$

30: **end for**

31: Extract the set of the candidate centroids, c. The centroid is a list of glowworms with maximum fitness function and also disjointed from one another.

32: Calculate$\boldsymbol{SSE} = \sum_{j=1}^{|C|} \sum_{i=1}^{|c_j|} \|c_j \quad g_i\|^2$

33: Calculate $\boldsymbol{InterDist} = \sum_{i=1}^{k} \sum_{j=i}^{k} \|c_i \quad c_j\|^2$

34: **for** each glowwormg_j, where j = 1 **to** m **do** {fitness function update}

35: Calculate fitness function

36: $\boldsymbol{F_j(g_j)} = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}\ Or\ \frac{interDist \times \frac{1}{n}|cr_j|}{\frac{intraD_j}{\max_j(intraD_j)}}\ Or\ \frac{interDist \times \frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j(intraD_j)}}$

37: **end for**

38: $t = t + 1$

39: **end while**

40: Return the set of candidate centroids, c

Output: A set of k clusters and k centroids

3.1.3 Clustering Error Function

Unlike K-means, sum of squared errors (SSE) is not a true representation of error in CGSO or CGSOm since the number of clusters changes with iteration. In fact, SSE will tend to increase as the number of iteration increases. This is due to the fact that as the number of centroids declines, the intra-distance in each cluster increases which leads to an increase in SSE. In this work, to visualize how the CGSOm algorithm improves the cluster quality, a

function that measures the error at each iteration was formulated. The error function is a sum of the mean squared error (SMSE) multiplied by the number of clusters (Equation 4). The formulated error function generally decreases with iteration.

$$Error = |C| \times \sum_{j=1}^{|C|} \frac{\sum_{k=1}^{|Cr_j|} \left\| cr_{jk} \quad \mathrm{g_j} \right\|^2}{|Cr_j|} \tag{4}$$

3.1.3.1 Cluster Quality Evaluation Measures

To measureour clustering algorithm with data set with known truth, we use three different measures for the evaluation of the cluster quality: Entropy (Equation 5), Purity (Equation 6)(Zhao and Karypis, 2002)and Rand Index, RI (Rand, 1971;Vinh*et al*.,2009) (Equation 7).

$$Entropy = \sum_{j=1}^{k} \left(\frac{|C_j|}{n} \frac{1}{\log q} \sum_{i=1}^{q} \frac{|C_j \cap L_i|}{|C_j|} log\left(\frac{|C_j \cap L_i|}{|C_j|} \right) \right) \tag{5}$$

$$Purity = \frac{1}{n} \sum_{j=1}^{k} \max_i \left(|C_j \cap L_i| \right) \tag{6}$$

$$RI = \frac{a+b}{\binom{n}{2}} \tag{7}$$

where C_j is a set of data instance in cluster j;L_i is the true assignments of the data instances in cluster i;a is the number of pairs of data instance that are in the same set in C and in the same set in L;b is the number of pairs of data instance that are in different sets in C and in different sets in L;n is the number of data instances in the data set;k is the number of clusters that is generated from the clustering process; and q is the number of actual clusters in the dataset.

Entropy values range from 0 (perfect clustering quality) to 1 (very poor clustering quality). Possible values of purity range from 0 (very poor clustering quality) to 1 (perfect clustering quality).

The rand index is a cluster quality evaluation measure that checks how close the resulting cluster is to the original cluster in terms of number of clusters and data points. It checks for the extent of agreement of thenumber of clusters as well as data points in the resulting cluster and the original cluster (the ground truth). Rand index results,though not appliedin the CGSO, are included in this work to provide more information about the cluster quality. Its value is between 0 and 1 and it is interpreted in a similar way as purity.

3.2 Automatic determination of the optimal number of clusters in a dataset.

To addressthis objective, the CGSOm was used to cluster the datasets/ RBF centres. This produced the clusters automatically and optimally due to the sensor range mechanism incorporated into the CGSOm. Toshow that theproposed CGSOm produces the optimal number of clusters in a dataset, the modified glowworm initialization method was replaced with the glowworm initialization method of CGSO and the performances of both resulting clustering algorithms in clustering datasets were compared.

3.3 Development of a RBFNN model that adapts to the number of clusters in a dataset.

This objective is achieved after the determination of the optimal number of clusters in a dataset. Once the number of clusters is determined, the topology of the network is reconfigured. As a rule in neural networks, the number of clusters is equal to the number of the neurons in the hidden layer which determines the topology of the network.

3.4 Optimization of the RBFNN parameters fully

To address this objective, the following were achieved:A machine learning platform was developed. The platform offers an approach allowing the user to explore and run simulation experiments of combinations of training techniques considered in this work. Next, training the RBFNN model was done with existing techniques and the BSO algorithm, after which theperformances of thetechniques were evaluated.The results from this experimentation produced two novel RBFNN training methodologies: CGSOm-CGD-RBFNN and the CGSOm-BSO-RBFNN models which were able to train the RBFNN model parameters fully and optimally.

The CGSOm-CGD-RBFNN and the CGSOm-BSO-RBFNN models are presented in Section 3.4.2.The processes involved in modelling multivariate time series data using the 2 models are discussed in Section 3.4.3.First, an introduction of the RBFNN model and the process of trainingthe network are outlined in Section 3.4.1.Furthermore, a description of how the software used in this work was developed is in Section 3.4.4.

3.4.1 The Basic RBFNN Model

Every RBFNN architecture has three layers and each layer is made up of one or more nodes. Figure 2 shows the architecture of a basic RBFNN. The input layer contains the feature nodes,$matrix\ X$ expressed as:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ & & \dots & \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Where n represents the feature size and d the size of the data set.

The only hidden layer in RBFNN consists of k nodes. Each node in the hidden layer encapsulates two data: cluster centroidμ_k and its standard deviation σ_k. The cluster centroids are usually determined by unsupervised learning algorithm. A typical example of such algorithm is K-means. K-means algorithm clusters hyper-dimensional points into k units by minimizing the sum of squared errors between each point and its centroid. K-means suffers one limitation – the value of k must be specified. One way to tackle this problem is to try-out different k valuesand eventually select the one which resulted in the least error (Isimeto *et al*., 2015). Alternatively, a swarm-based clustering approach, the CGSO can be used to find the value of k automatically.

The clustering process yields $k$centroids positioned at each hidden node. The matrix representation of the centroids is:

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ & & \dots & \\ c_{k1} & c_{k2} & \dots & c_{kn} \end{bmatrix}$$

The centroids are critical in computing the outputs from each hidden node. This is done via kernel functions. The most commonly used are:

I. Gaussian function:

$$\varphi(x) = \mathrm{e}^{-\gamma\|x-c\|^2} \quad (8)$$

Where$\|x \quad c\|$ represents Euclidean distance andγ could be a constant or a function of σ

II. Multi-Quadric function:

$$\varphi(x) = \sqrt{\|x \quad c\|^2 + \gamma^2} \quad (9)$$

III. Inverse Multi-Quadric function:

$$\varphi(x) = \frac{1}{\sqrt{\|x-c\|^2+\gamma^2}} \quad (10)$$

IV. Thin-plate spline function:

$\varphi(x) = \|x \quad c\|^2 \log(\|x \quad c\|)$ (11)

V. Cubic function:

$\varphi(x) = \|x \quad c\|^3$ (12)

VI. Linear:

$\varphi(x) = \|x \quad c\|$ (13)

The most widely used among these kernels is Gaussian function. The matrix representation of the hidden layer output is given as:

$$H = \begin{matrix} 1 & 1 & \ldots & 1 \\ 11 & 12 & \cdots & 1d \\ 21 & 22 & \cdots & 2d \\ & & \ldots & \\ k1 & k2 & \cdots & kd \end{matrix}$$

One or more nodes constitute the output layer.Values at each of the M output nodes are computed by:

$y_m = w_0^m + \sum_{i=1}^{K} w_i^m \varphi(x)$ (14)

where y_m is the value at output node m, w^m is a set of K number of weights used in mapping hidden nodes values to output node m and $w_0^m$is a bias term, x is the input vector.

The matrix representation of the weights is given as:

$$W = \begin{bmatrix} w_{10} & w_{11} & w_{12} & \ldots & w_{1k} \\ w_{20} & w_{21} & w_{22} & \ldots & w_{2k} \\ & & & \ldots & \\ w_{m0} & w_{m1} & w_{m2} & \ldots & w_{mk} \end{bmatrix}$$

By matrix multiplication of W and H, the matrix representation of the output nodes can be derived:

$$Y = \ W \times H = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1d} \\ y_{21} & y_{22} & \cdots & y_{2d} \\ & & \ldots & \\ y_{m1} & y_{m2} & \cdots & y_{md} \end{bmatrix}$$

Getting the right set of weights that minimizes the prediction error requires solving an optimization problem.The evaluation or cost function (Equation 15) is represented as:

$$E = \frac{1}{d} \sum_{i=1}^{d} \sum_{m=1}^{M} \left(y_{mi}^{obs} \quad y_{mi}\right)^2 + \frac{\lambda}{2d} \sum_{k=1}^{K} \sum_{m=1}^{M} w_i \quad (15)$$

Wherey_{mi}^{obs} is the target at output node m for sample data$i$; and λ is the regularization term for controlling over-fitting.

The optimization algorithms commonly used in previous work include Gradient Descent, Conjugate Gradient Descent and PSO. In this work, however, a novel optimization algorithm

known as BSO was used and its performance compared with existing techniques that have been used to optimize the RBFNN.

3.4.2 Proposed CGSOm-BSO and CGSOm-CGD RBFNN Models

The CGSOm-BSO RBFNN Model is a RBFNN variant. It is a RBFNN model whose centroids are determined by the proposed CGSOm and the weights are determined by BSO. Figure3 is a schematic diagram of the network flow.

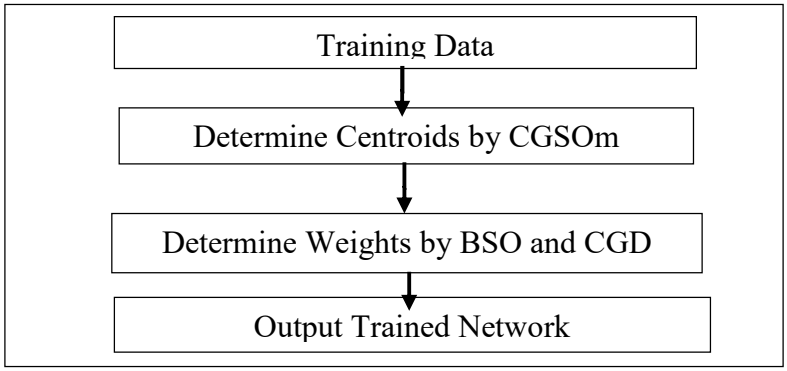


Figure 3: The Flow of the proposed models

3.4.3 Procedures for Modelling CGSOm-BSO and CGD RBFNN Model

The specific steps for training and validating our proposed model is outlined. It covers data collection, data pre-processing, data partitioning, feature extraction and parametertuning.

3.4.3.1Data Collection

Two sets of data are used. The data for case 1 is extracted from General Electric Company's daily historical stock prices dataat(YAHOO! FINANCE, 2009).The data used consists of two variables – openstock price data and closestock price data. It is made up of 2,000 data samples.For case 2 experiment, currency exchange rate data at (PACIFIC Exchange Rate Service, 1996)was used. The currency exchange rate data employed in the investigation of the forecasting problem composed of real data representing the weekly averaged exchange rates between British pound and US dollar during the period from 31 December 1979 to 26 December 1983.

3.4.3.2 Data Pre-processing

The range of values of data are mostly widely varied. This could lead to poor performance by clustering and optimization functions because the variable with large values will dominate the result of the objective function. Data pre-processing is required to prevent this kind of

problem.The data collected were pre-processed by standardization of the data for each variable. The formula for standardization (Equation 16) is given as:

$$v_{new} = \frac{v \quad \bar{v}}{\sigma} (16)$$

Where $\bar{v}$ and σ are the means and standard deviation respectively.

This process ensures that the new values in each variable has a zero-mean and unit-variance.

3.4.3.3 Data Partitioning

Every data set used in this work is partitioned into training set (70%), validation set (20%) and test set (10%).

3.4.3.4 Feature Extraction

For time series data, the features for each input vector consists of data samples from the past l days. l stands for lag. For example, if lag happens to be 2, the first input vector would consist of data points for the first two days unrolled into a row vector. One major challenge is determining the right value of lag. To solve this problem various values of lag were tested and the right value selected for each data set.

3.4.3.5 Parameter Tuning

The proposed model has several parameters that require tuning. These parameters include CGSOm parameters, regularization term and BSO parameters. In this work, the parameters were tuned by going through a training-validation cycle until the results are satisfactory and can be generalized.

3.4.4 The Software Development

A Matlab-based software application was built specifically for this work. It is important to note that this application does not depend on Matlab in-built neural network toolbox. The clustering and training algorithms were written from scratch. This choice was made because the work requires having full control of every variable that affects the successful implementation of RBFNN; also, the novel algorithms used in this work do not exist in Matlab toolbox.

The application has graphical user interface (GUI) and was built in Matlab Graphical User Interface Development Environment (GUIDE). The application was written using functional programming paradigm. It has custom-built Matlab functions. Figure 4shows a snapshot of the application interface.

The features of the application include data importation from excel and automatic data pre-processing. It allows users to change options including training algorithm, clustering algorithms, RB function, lag, and use of PCA. The effect of PCA, number of hidden units, regularization term, kernel function and many others can be diagnosed. Also, the core components of the trained network can be saved and used in building a commercial or open-source application.

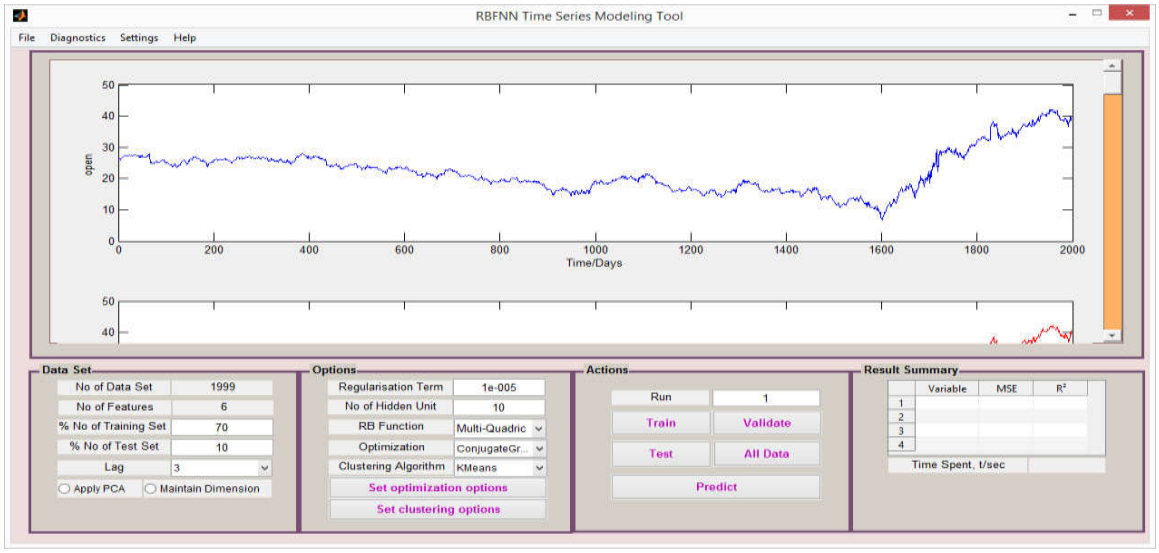


Figure 4: The Application Interface

CHAPTER FOUR

EXPERIMENTAL RESULTS AND DISCUSSION

Two experiments were conducted in this work. First, the experiment on the clustering aspect of this researchis presented in Sections 4.1 to 4.3. This is to demonstrate the effectiveness of CGSOm over CGSO and other four well-known clustering algorithms. Secondly, the experiment for training the RBFNN model follows in Sections 4.4 to 4.7. Validation of the proposed approaches is presented in Section 4.8.

4.1 Experimental Results and Discussions of Effectiveness of CGSOm

This section demonstrates the effectiveness of CGSOm in data clustering.Each experiment is run on a PC with 6GB of RAM and 3 Intel cores (1.90 GHz each).

4.1.1 Test Data

Seven data sets were collected and pre-processed by rescaling each feature to value between 0 and 1. The first five data sets are real data obtained from UCI Machine Learning

Repository (Lichman, 2013), while the last two are artificial data sets from (ELKI, n.d.) data repository.Table 1 summarizes the properties of each data set.

Table 1: Summary of the data sets

Data Set	Record	Featur	Cluster	Type	Source
Iris	150	4	2	Real	UCI
Balance	625	4	3	Real	UCI
Seed	210	7	3	Real	UCI
Ecoli	327	7	5	Real	UCI
Glass	214	9	6	Real	UCI
Mouse	490	2	3	Artificial	ELKI
VaryDensity	150	2	3	Artificial	ELKI

4.1.2 Parameter Settings

Table 2 summarizes the constant parameters used in all experiments.

Table 2: The CGSOm constant parameters

Parameter	Value
L_0	5.0
ρ	0.4
γ	0.6
$population\ of\ glowworms, m$	1000
$minimum\ r_s$	0.0001
$number\ of\ r_s to\ sample, n_r$	30
$maximum\ number\ of\ iteration$	200

4.1.3 Efficient determination of Local Sensor Range (r_s) of the CGSO

There is need to show that the sensor range determinationalgorithm (in Section 3.1, algorithm 7) efficiently determines the sensor range when compared to the trial and error technique in the CGSO. This was achieved by evaluating the efficiency of the CGSOm in clustering data and comparing the results with that of CGSO as well as other clustering techniques. A summary of theresults of experimental simulations showing the effectiveness of the CGSOmalgorithmin data clusteringwas compared against those of the CGSOand other four standard, well-known clustering techniques commonly used in the literature as benchmarks. Evaluation of theperformances of these clustering algorithms were based on cluster quality measures of entropy, purity and rand index. The cluster quality results from these are in Tables4, 5 and 6.

First, the average sensor range of each data set using the sensor range (r_s) algorithm is firstcomputed. Table 3summarizes the computed mean sensor ranges and standard deviation of each data. From the table,it can be observed that the standard deviations are very small with respect to the mean. This shows that the values for all the runs do not differ much.

Table 3: Computed mean sensor range for each data set for 50 runs

Data Set	**Sensor Range Mean**	**Sensor Range Standard Deviation**
Iris	0.3040	0.0314
Balance	0.4441	0.0050
Seed	0.2311	0.0138
Ecoli	0.3812	0.0160
Glass	0.4100	0.0444
Mouse	0.1734	0.0012
VaryDensity	0.2045	0.0465

A comparison of the clustering qualityof the CGSOm with the CGSO as well as with thefour well-known clustering methods, in terms of entropy and purity, are shown in Tables 4 and 5 respectively. The clustering quality (the mean and the standard deviation of entropy and purity results) for each data is determined for 50 runs and for each fitness function. Best results are placed in square brackets. In each case, the underlined are the highest purity and lowest entropy values. The four well-known clustering algorithms commonly used in the literature are K-means clustering (Macqueen, 1967), average linking agglomerative Hierarchical Clustering, HC (Zhao and Karypis, 2002), Further First, FF (Hochbaum and Shmoys, 1985), and Learning Vector Quantization, LVQ (Kohonen, 2003). Rand Index result is contained in Table6which is not reported in CGSO.

Table 4: Entropy Results

Data Set	**CGSOm**			**CGSO**			**K Means**	**HC**	**FF**	**LVQ**
	F1	***F2***	***F3***	***F1***	***F2***	***F3***				
Iris	0.382 ± 0.091 [0.136]	0.408 ± 0.052 [0.195]	0.405 ± 0.062 [0.136]	0.209	0.241	0.233	0.264	0.230	0.307	0.790
Balance	0.499 ± 0.021 [0.460]	0.498 ± 0.020 [0.453]	0.497 ± 0.019 [0.465]	0.622	0.690	0.669	0.701	0.739	0.654	0.753
Seed	0.327 ± 0.032 [0.276]	0.323 ± 0.028 [0.276]	0.329 ± 0.031 [0.265]	0.302	0.317	0.305	0.327	0.298	0.537	0.577
Ecoli	0.249 ± 0.011 [0.231]	0.250 ± 0.015 [0.224]	0.253 ± 0.015 [0.226]	0.325	0.342	0.324	0.307	0.522	0.611	0.579

Glass	0.337± 0.035 [0.270]	0.328 ± 0.036 [0.269]	0.332 ± 0.033 [0.244]	0.543	0.569	0.568	0.567	0.662	0.646	0.754
Mouse	0.130 ± 0.019 [0.096]	0.135 ± 0.023 [0.096]	0.138 ± 0.021 [0.085]	0.299	0.302	0.304	0.319	0.165	0.351	0.262
VaryDensity	0.252 ± 0.153 [0.030]	0.243 ± 0.150 [0.116]	0.238 ± 0.159 [0.000]	0.141	0.141	0.138	0.145	0.421	0.466	0.728

Table 5:Purity Results

Data Set	CGSOm			CGSO			K Means	HC	FF	LVQ
	F1	*F2*	*F3*	*F1*	*F2*	*F3*				
Iris	0.708 ± 0.096 [0.960]	0.681 ± 0.058 [0.913]	0.683 ± 0.065 [0.960]	0.919	0.903	0.909	0.887	0.877	0.860	0.507
Balance	0.762 ± 0.017 [0.794]	0.765 ± 0.016 [0.811]	0.765 ± 0.014 [0.786]	0.726	0.685	0.694	0.659	0.632	0.653	0.619
Seed	0.892 ± 0.013 [0.910]	0.894 ± 0.013 [0.910]	0.891 ± 0.014 [0.914]	0.900	0.889	0.897	0.876	0.895	0.667	0.667
Ecoli	0.846 ± 0.006 [0.857]	0.845 ± 0.007 [0.857]	0.845 ± 0.007 [0.857]	0.792	0.779	0.789	0.774	0.654	0.599	0.654
Glass	0.730 ± 0.068 [0.808]	0.741 ± 0.056 [0.804]	0.732 ± 0.061 [0.822]	0.541	0.533	0.529	0.542	0.463	0.481	0.411
Mouse	0.955 ± 0.009 [0.969]	0.953 ± 0.012 [0.969]	0.952 ± 0.010 [0.971]	0.837	0.834	0.833	0.827	0.910	0.800	0.843
VaryDensity	0.859 ± 0.134 [0.993]	0.867 ± 0.132 [0.960]	0.868 ± 0.134 [1.000]	0.956	0.956	0.957	0.953	0.667	0.667	0.567

Table 6: Rand Index Results

Data Set	CGSOm		
	F1	*F2*	*F3*
Iris	0.798 ± 0.051 [0.950]	0.783 ± 0.028 [0.899]	0.785 ± 0.035 [0.950]
Balance	0.593 ± 0.004 [0.599]	0.593 ± 0.004 [0.602]	0.593 ± 0.004 [0.600]
Seed	0.831 ± 0.014 [0.854]	0.834 ± 0.015 [0.878]	0.831 ± 0.014 [0.859]
Ecoli	0.901 ± 0.006 [0.909]	0.900 ± 0.007 [0.909]	0.900 ± 0.007 [0.909]
Glass	0.770 ± 0.036 [0.841]	0.774 ± 0.035 [0.817]	0.771 ± 0.032 [0.827]
Mouse	0.717 ± 0.008 [0.741]	0.720 ± 0.009 [0.737]	0.721 ± 0.010 [0.743]
VaryDensity	0.820 ± 0.083 [0.919]	0.824 ± 0.080 [0.911]	0.826 ± 0.084 [0.934]

In Table 4, where a smaller entropy implies a better result, theentropy resultsshowthat CGSOmperforms better than CGSO and other clustering algorithms in most data sets which are Balance, Ecoli, Glass and Mouse, with an average entropy of 0.497, 0.249, 0.328 and 0.130 respectively. Although for the Mouse data, it is noted that HC algorithm obtained an entropy of 0.165, however our CGSOm obtained a much better entropy of 0.130 thus

outperforming the HC.Also, it is noted that for Ecoli data, K-means gave entropy of 0.307, while CGSOm performed better with entropy of 0.249. For VaryDensity and Iris data, CGSO-F3 and CGSO-F1 gave best entropy values of 0.138 and 0.209respectively. However, for the Seed data, HC obtained the best entropy.In Summary, CGSOm gave best results in 4 out of the 7 datasets clustered (57%); CGSO gave best results in 2 datasets(28.5%); and HC gave best result in only 1 dataset (14.5%).A pictorial representation of this result is presented in Figure 5.

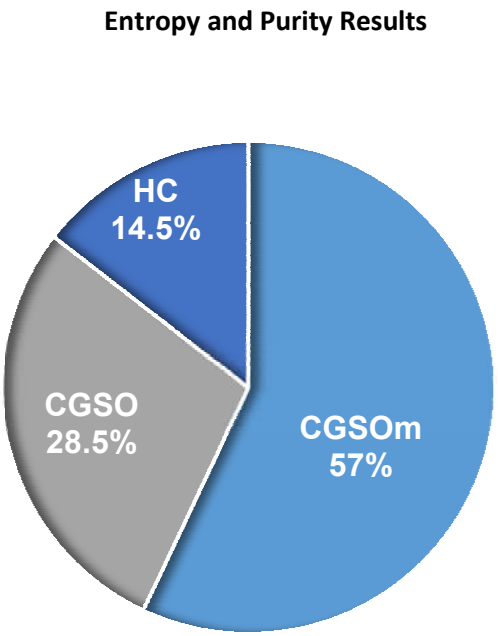


Figure 5: Comparing clustering quality of CGSOm with other Clustering techniques

In Table 5, where a higher purity implies a better result, the purity resultsshowthat CGSOm performs better than CGSO and other clustering methods in the data sets: Balance, Ecoli, Glass and Mouse, with an average purity of 0.765, 0.846, 0.741 and 0.955 respectively. However, for the Iris and VaryDensity data, CGSO-F1 and CGSO-F3 gave best purity values of 0.919 and 0.957 respectively, while for the Seed data, CGSO and CGSOm have similar values.In summary, again CGSOm gave best results in 4 out of the 7 datasets clustered(57%); CGSO gave best results in 2 datasets (28.5%); and HC gave best result in only 1(14.5%). Figure 5 also serves as the pictorial representation of this result.

Though the Mouse data has the best entropy and purity, it does not have the best rand index. This shows that entropy and purity are not sufficient indicators of cluster quality. In fact, it is practically possible to have 0 entropy and 1 purity and not have 1 rand index. Rand index results are presented in Table 6.The datasets: Mouse, Balance, Ecoli, Glass, Iris, Seed and VarDensity have a rand index of 0.721, 0.593, 0.901, 0.774, 0.798, 0.834 and 0.826 respectively. It can be deduced that theCGSOm produces clusters that agree to a large degree with the ground truth since it gave rand index values of 0.70 and above in 6 out of the 7 cases considered (86%). This result is presented graphically in Figures 6 and 7.

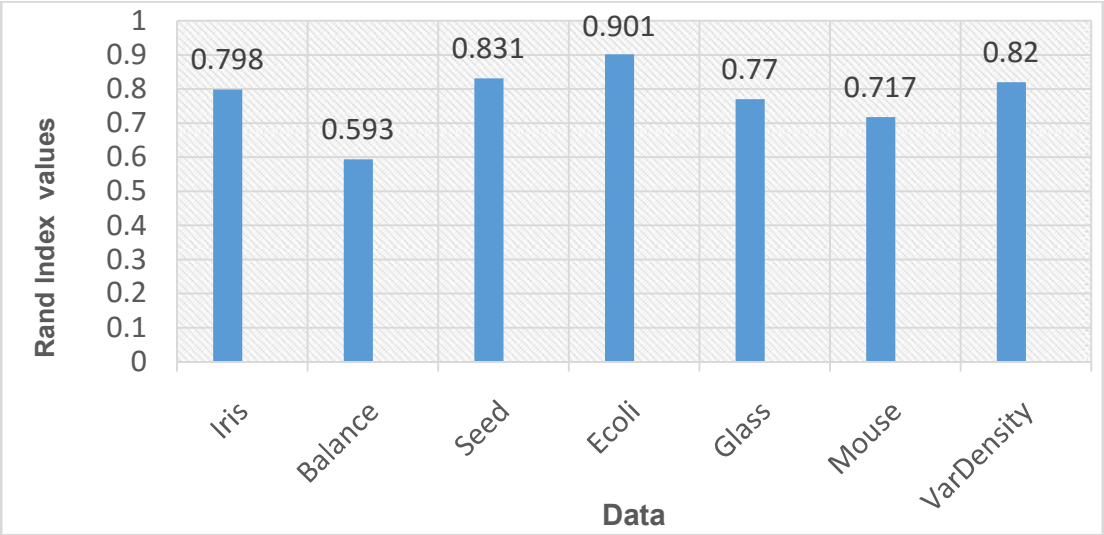


Figure 6: Rand Index Resultof the CGSOm

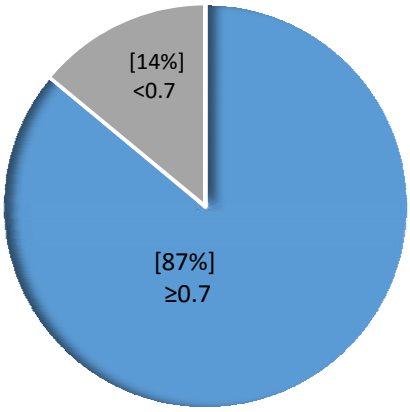
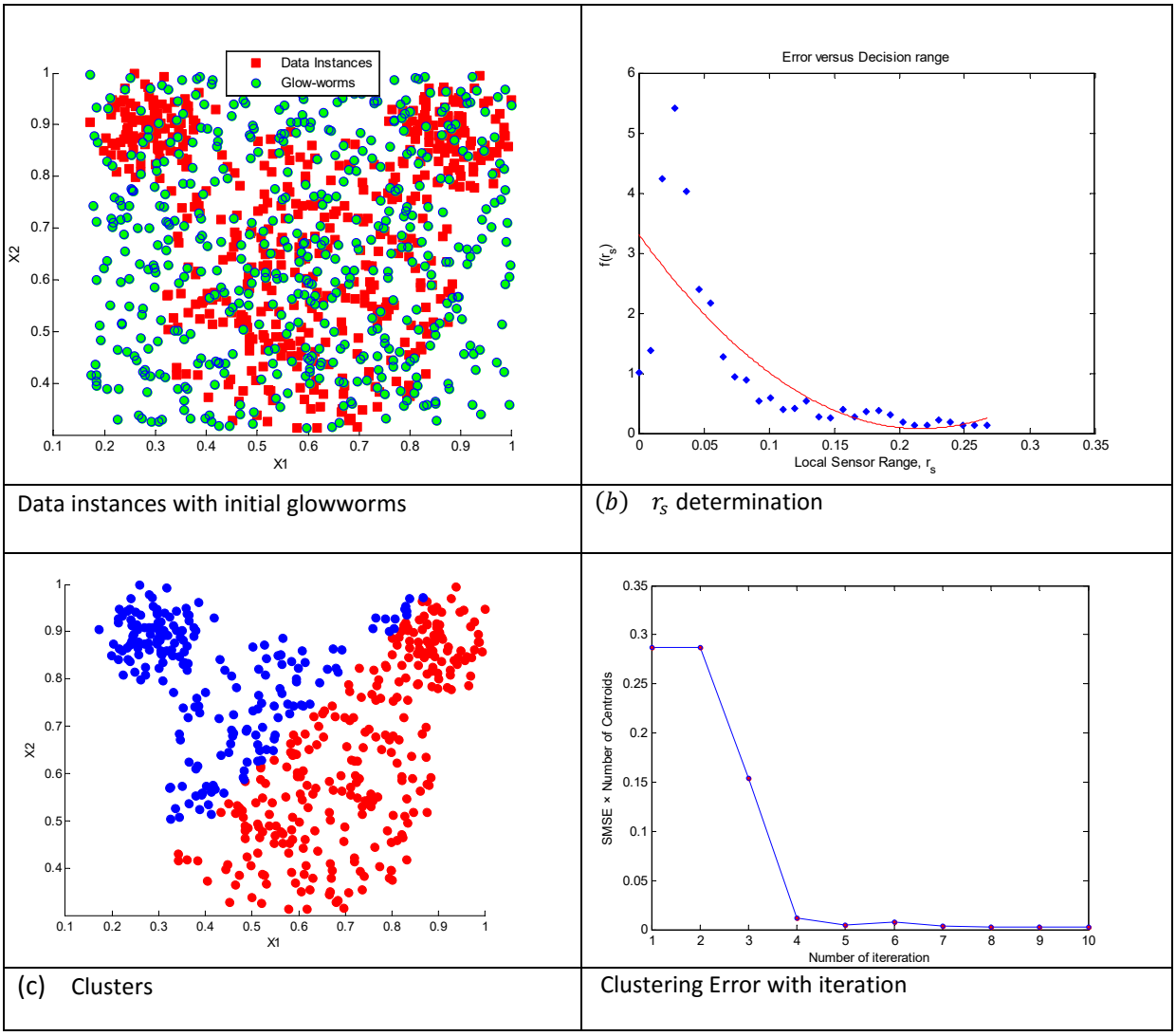


Figure 7: Showing Agreement of CGSOm result with the ground truth

4.2Automatic determination of the optimal number of clusters in a dataset.

This section shows that the method of initializing the glowworm affects the clustering results.The result of comparing the two glowworm initialization methods described is presented in Figures8b and Figure 9b;

To evaluate the effectiveness of the modified initialization of the glowworm, the glowworms initialization technique in CGSOm was compared with thatin CGSO, using Mouse data set. Comparing Figures 8b and 9b for the CGSO and CGSOm respectively, it can be observed thatthe CGSOm based on the modified glowworminitializationmethod produced a better curve for determining sensor range,r_s than that obtained using the initializationmethod ofCGSO. The decision range from Figure 8b (based on CGSO glowworminitialization method) is 0.218 whereas that of Figure 9b (based on CGSOmglowworm initialization method) is 0.173. The number of clusters for the CGSO is 2; while that for the CGSOm is 3, which is the same number of clusters (ground truth) in the original mouse data. This proves that the way the glowworms are initialized plays a vital role in determining sensor range, r_s correctly and consequently determines the number of clusters found.



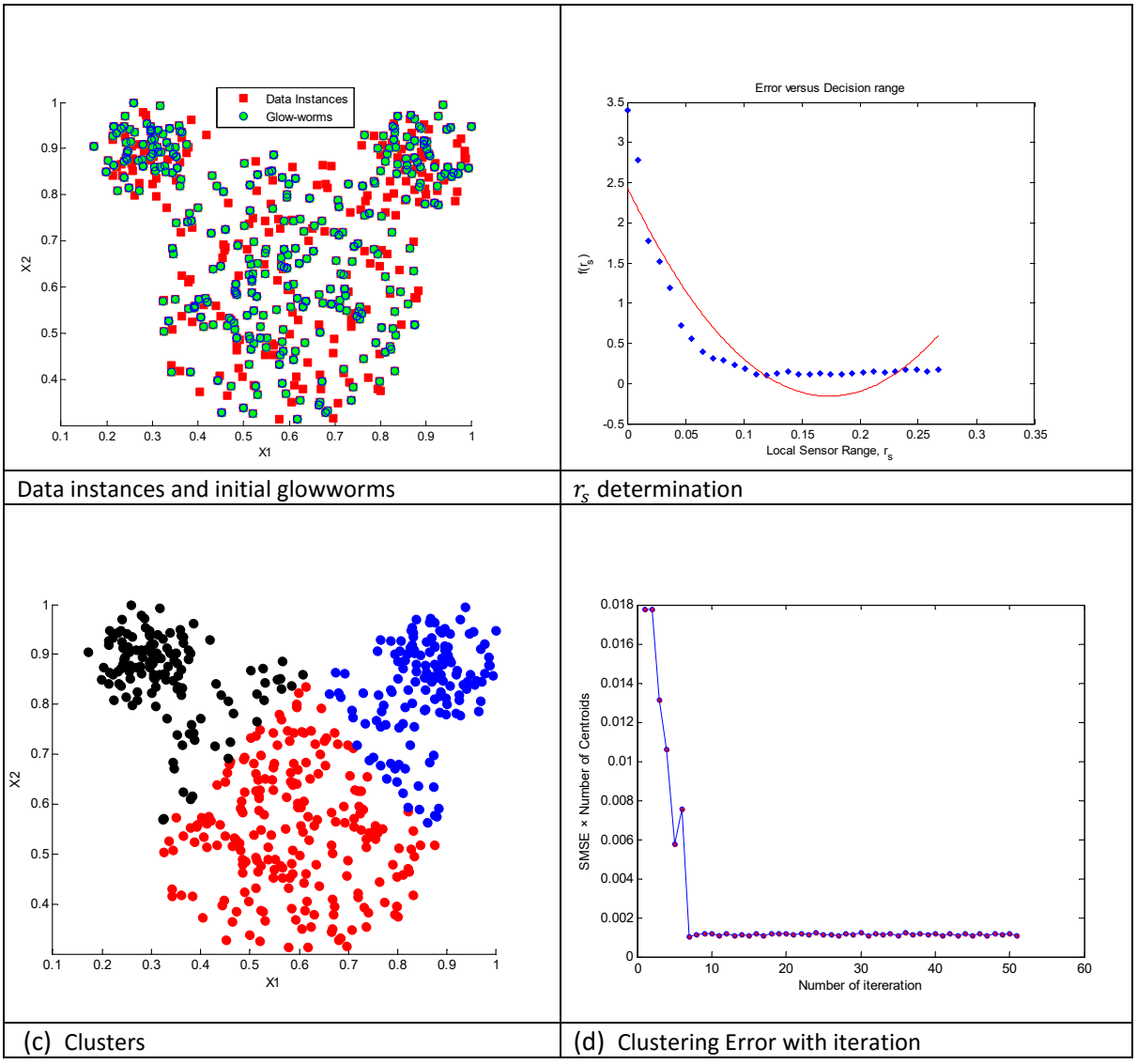
Data instances with initial glowworms

(b) r_s determination

(c) Clusters

Clustering Error with iteration

Figure 8: Clustering result for Mouse data set using original CGSO initialization.



Data instances and initial glowworms | r_s determination

(c) Clusters | (d) Clustering Error with iteration

Figure9: Clustering result for Mouse data set using modified CGSOm initialization

4.3 Development of a RBFNN model that adapts to the number of clusters in a dataset

The number of clusters in a datasetdetermines the topology of the network, since theoretically the number of clusters is equal to the number of the neurons in the hidden layer and the topology of the network. Hence, for any given clusterable datasets, as soon as the CGSOm gets the number of clusters, the RBFNN adapts its architecture accordingly based on the number of clusters. For instance, if the number of clusters obtained is k, the network adapts its architecture to have same k number of neurons in the hidden layer. This adaptive architecture of the RBFNN is shown in figure 2 in chapter 2.

In this work, CGSO has been improved by developing an algorithm for determining sensor range and by modifying the glowworm initialization. It was shown that the modified initialization phase improves the performance of the algorithm that determines the sensor range. It was also demonstrated that the computed sensor range in CGSOm leads to better cluster quality for most data sets when compared with the results of CGSO andthose of other four well-known clustering algorithms.

4.4Experimental Results and Discussion on RBFNN Weight Optimization

Thefollowing sections present the experimental results and discussion on RBFNN training and testing

4.5Optimizing the RBFNN parameters fully

Presented in this section are the experimental results of the training methodologies of existing RBFNN models compared withthe proposed training approach. Results from an empirical study are presented to show how the proposed approacheswere realized. To train the RBFNN models, two sets of experiments were conducted; these involved the use of two case-study problems namely: Stock Price Forecasting problemin Section 4.6and Currency Exchange Rate Forecasting problem in Section 4.6.In each case, several variants of RBFNN models were developed.

TheRBFNN models implemented are as listed in Table 9.The performance of the RBFNN models were determined and compared by the mean of the Mean Squared Error (MMSE) and R^2values of 10 independent runs.All results were obtained by training the model with 70% of the data.

Details of the results/findings are discussed in Section 4.6.3and results shown in Tables 9, 10 (for case 1: Stock Price problem) as well as in Section 4.7.3with results shown in Tables 13, 14 (for case 2: the Currency Exchange problem).

The results from the study achieved theobjective (4) ofoptimizing the RBFNN parameters fully. This study produced two new training methodologies for optimizing the RBFNN parameters fully.These are: the CGSOm-CGD RBFNN model and the CGSOm-BSO RBFNN model. These are new and optimal RBFNN models for time series forecasting problems.

4.6 Case 1: Stock Price Forecasting problem

In this case,RBFNN models weredeveloped using stock price data extracted from General Electric Company's daily historical stock prices data. The dataemployed in the investigation of the forecasting problem composed of data attributes "Open price data" and "Close price data" and 2000 instances of this data. The time series plot of how the data evolves over time is in Figure10.

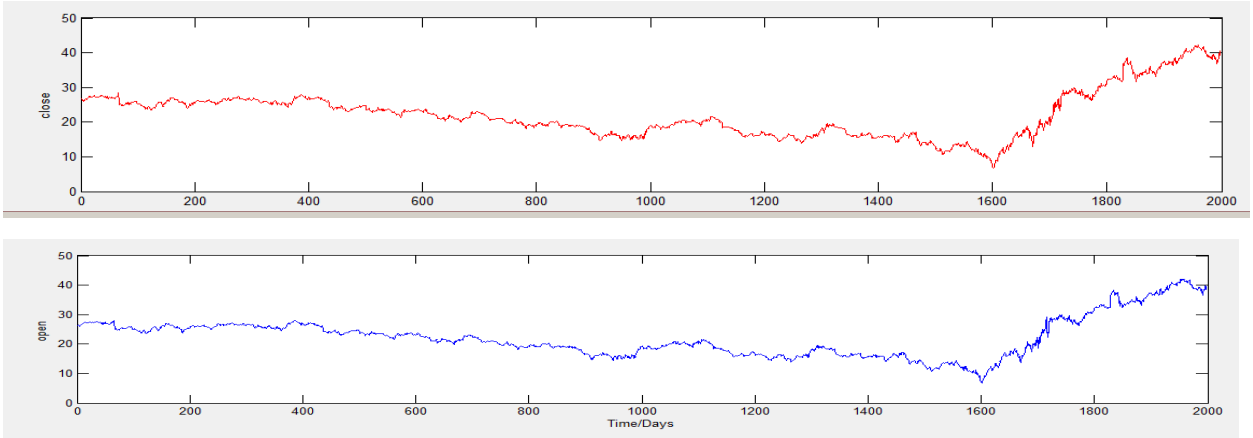


Figure 10: Time series plot of Stock Price data

4.6.1 Parameter Settings

The parameter settings for the CGSO and CGSOm algorithms as well as the parameter settings for BSO algorithm used for Stock Price Forecasting problem are in Tables 7 and 8

Table 7: Parameters settings for CGSO and CGSOm algorithm used for stock price forecasting problem

S/no	Parameter	Value
1	glowworms population	70
2	Initial Luciferin,L0	4
3	Decision range, r0	1.2
4	luciferin decay constant,rho	0.2
5	luciferin enhancement constant,y	0.2
6	constant value,B	0.5
7	step size,s	0.2
8	number of neighbours, nt	4
9	maxIter	70

Table 8: Parameters settings for BSO algorithm used for stock price forecasting problem

S/no	Parameter	Value
1	glowworms population	700
2	Initial Luciferin,L0	4
3	luciferin decay constant,rho	0.2
4	luciferin enhancement constant,y	0.2
5	step size, s	0.7
6	global best attraction constant, cg	0.03
7	maxIter	500
8	adaptive step sizing control, cs	3
9	mass extinction control, eT	70
10	strong local search control, IR	10

4.6.2 Plots of Optimized RBFNN Models

Figures 11 and 12showthe time series plot and regression plot of actual and predicted stock pricerespectivelyusing CGSO-BSOtrained RBFNN. Figures13 and 14displaythe time series plot and regression plot of actual and predicted stock price respectively using CGSOm-BSO trained RBFNN. Figures 15 and 16showthe time series plotand regression plot of actual and predicted stock price respectively using the PCA-CGSOm-BSO RBFNN.

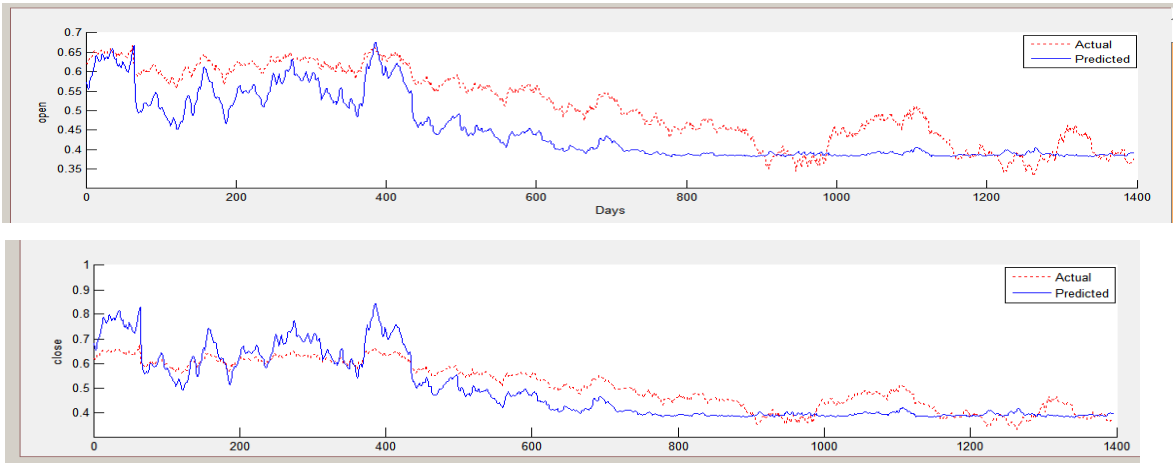


Figure 11: Time series plot of actual and predicted stock price using CGSO-BSO trained RBFNN

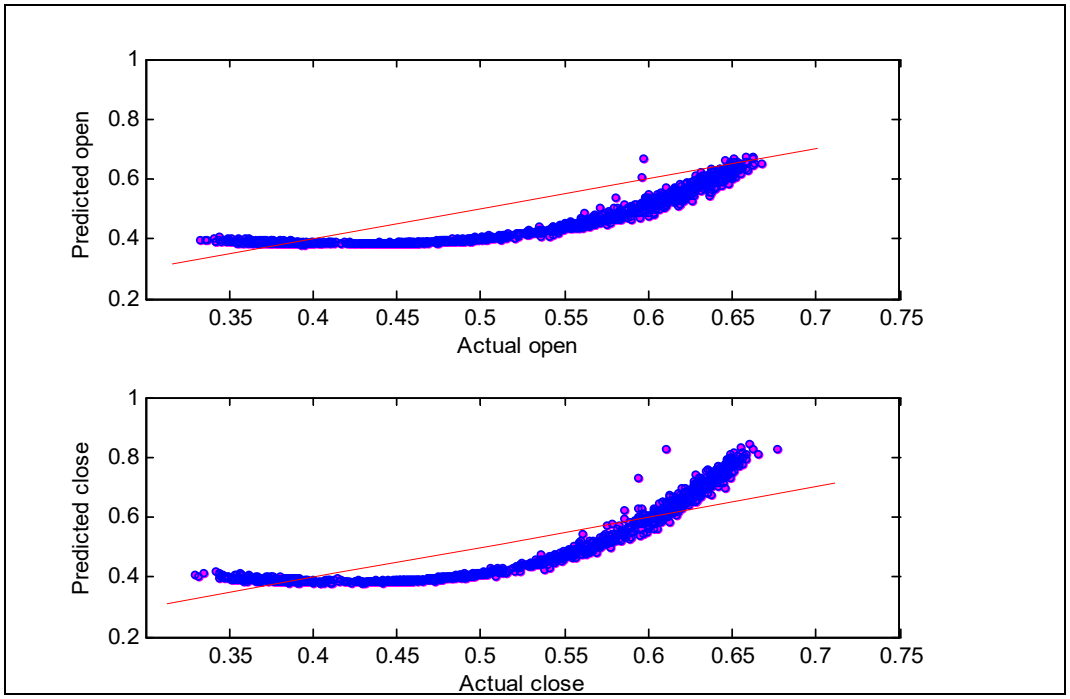


Figure 12: Regression plot of actual and predicted stock price from CGSO-BSO trained RBFNN

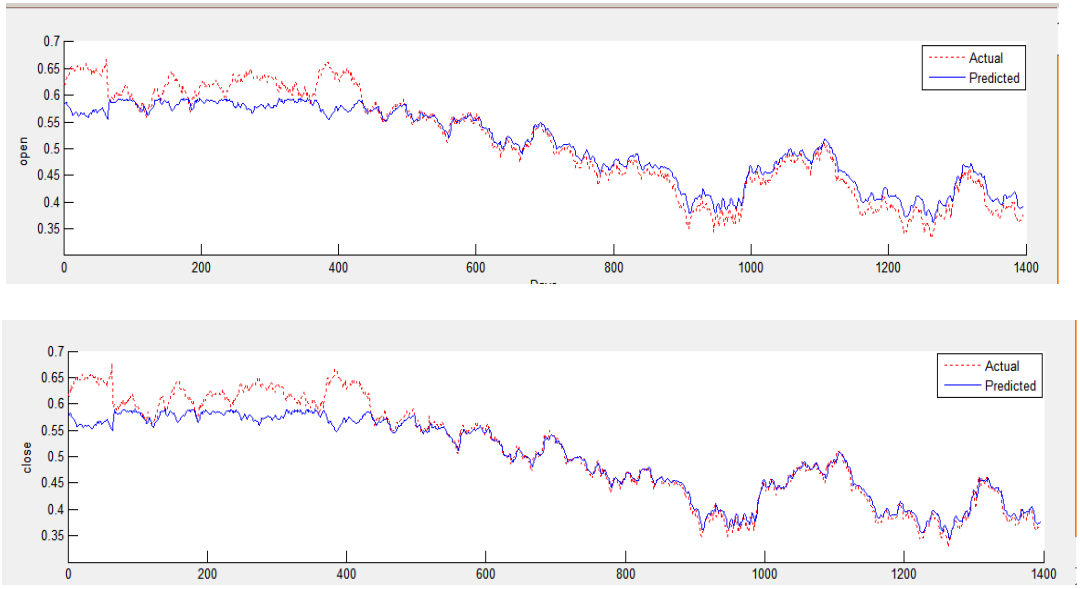


Figure 13: Time series plot of Actual and Predicted stock price from CGSOm-BSO trained RBFNN

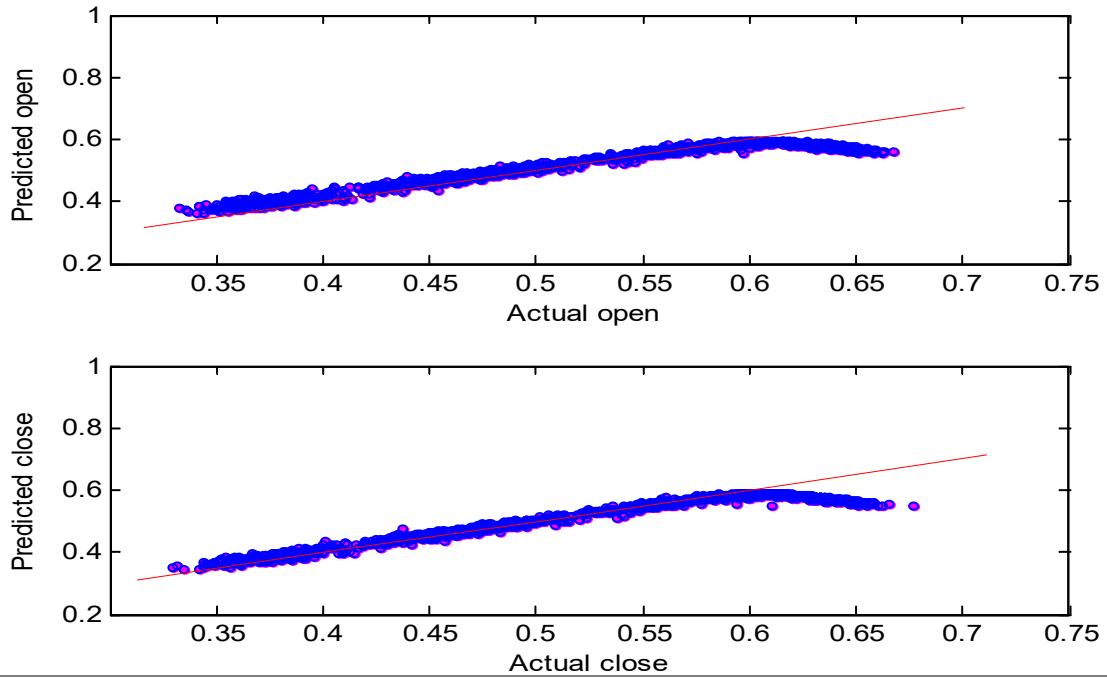


Figure 14: Regression plot of actual and predicted stock price using CGSOm-BSO RBFNN

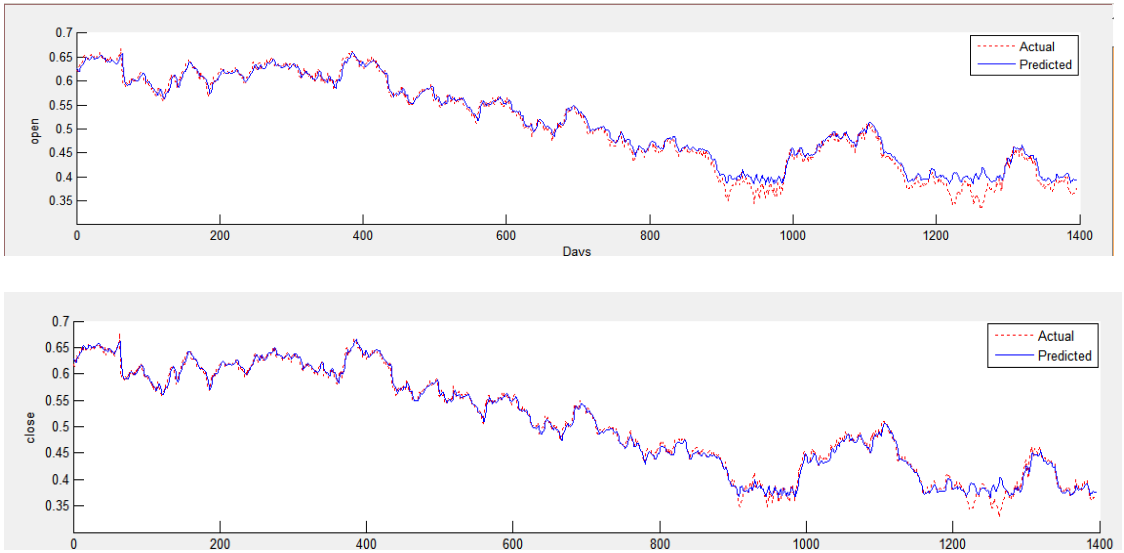


Figure 15: Time series plot of actual and predicted stock price trained by PCA-CGSOm-BSO RBFNN

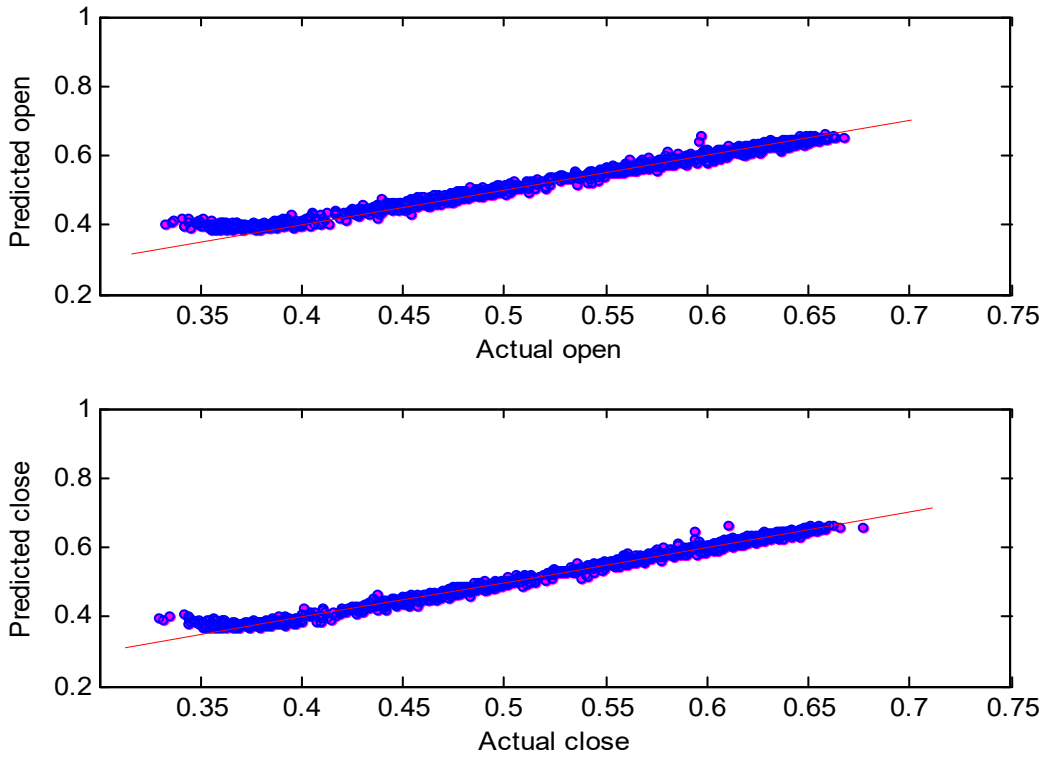


Figure 16: Regression plot of actual and predicted stock price trained by PCA-CGSOm-BSO RBFNN

4.6.3Comparative Analysis

This sub-section presents the results of the variants of RBFNN models optimized by different techniques and developed using stock price data.The mean of the mean squared error (MMSE) of 10 runs and the standard deviations for each RBFNN model variant are computed.

In Tables 9and 10, the summary of results for average of 10 runs of Open stock price and Closestock price data are shown respectively, for the RBFNN models considered.Experimental results show that for the Open price data, the CGSO-BSO trained RBFNN taken as control in this study, yields a MMSE of $4.01x10^{-2}$, the CGSOm-BSOtrained RBFNN yields a MMSE of $2.77\ x10^{-2}$. The CGSOm-CGDtrained RBFNN yields MMSE of $7.8\ x\ 10^{-3}$, while the PCA-CGSOm-CGD RBFNN yields MMSE value of $7.9x10^{-3}$ indicating that the PCA does not have much influence on the result of the CGSOm-CGDtrained RBFNN, as both have almost same error values.It is noted that the BSO variant slightly outperforms the CGSOm-PSO RBFNN with an error of $5.4x10^{-2.}$ This is an advantage for the BSO.

Table 9: Comparative Summary of Results for average of 10 Simulation Runs for Open stock price

Model	Open			
	Training		Test	
	MMSE $\pm$ SD	R^2	MMSE $\pm$ SD	R^2
CGSO-BSO trained RBFNN =CONTROL	$4.01 \times 10^{-2} \pm 0.006$	0.9606	$2.74 \times 10^{-2} \pm 0.0003$	0.9732
CGSOm-BSO trained RBFNN	$2.77 \times 10^{-2} \pm 0.0004$	0.9733	$1.80 \times 10^{-2} \pm 0.0002$	0.9837
PCA-CGSOm-BSO trained RBFNN	$2.04 \times 10^{-2} \pm 0.0022$	0.9616	$4.19 \times 10^{-2} \pm 0.0002$	0.9745
Other RBFNN Variants :				
CGSOm-CGD-RBFNN	$7.8 \times 10^{-3} \pm 0.0003$	0.9924	$7.8 \times 10^{-3} \pm 0.0002$	0.9932
PCA-CGSOm-CGD-RBFNN	7.9×10^{-3}	0.9926	7.6×10^{-3}	0.9899
CGSOm-GD-RBFNN	1.64×10^{-1}	0.8505	1.60×10^{-1}	0.8701
CGSOm-PSO-RBFNN	5.4×10^{-2}	0.9602	4.4×10^{-2}	0.9592
CGSO-CGD-RBFNN	3.0×10^{-1}	0.9505	2.7×10^{-1}	0.9612
Kmeans-GD-RBFNN	164×10^{-2}	0.1011	1.02×10^{-2}	0.3011
Kmeans-CGD-RBFNN	9.77×10^{-3}	0.9888	5.60×10^{-3}	0.9688
Kmeans-PSO-RBFNN	1.4×10^{-3}	0.8631	1.0×10^{-2}	0.8721

For the results ofClose price data (in table 10), CGSO-BSO trained RBFNN gives MMSE of $3.89x10^{-2}$, CGSOm-BSORBFNN gives MMSE of $2.71x10^{-2.}$. Also, the effect of PCA is not felt in theClose stock data as the CGSOm-CGD RBFNN and the PCA-CGSOm-CGD RBFNN yieldsimilar MMSE values.Further, the R^2 values are all on the high side for these models for both the Open and Close data indicating that all the models fit the data well. The standard deviations are very small with respect to the mean indicating that the values for all the runs do not differ much.

Table 10: Comparative Summary of Results for average of 10 Simulation Runs for Close stock price

Model	Close			
	Training		Test	
	MMSE $\pm$ SD	R^2	MMSE $\pm$ SD	R^2
CGSO-BSO trained RBFNN =CONTROL	$3.89 \times 10^{-2} \pm 0.028$	0.9616	$2.51 \times 10^{-2} \pm 0.0022$	0.9740
CGSOm-BSO trained RBFNN	$2.71 \times 10^{-2} \pm 0.0005$	0.9790	$1.30 \times 10^{-2} \pm 0.0004$	0.9842
PCA-CGSOm-BSO trained RBFNN	$1.56 \times 10^{-3} \pm 0.0018$	0.9630	$3.2 \times 10^{-2} \pm 0.0011$	0.9742
Other RBFNN Variants :				
CGSOm-CGD-RBFNN	$5.7 \times 10^{-3} \pm 0.0005$	0.9945	$4.4 \times 10^{-3} \pm 0.0002$	0.9948
PCA-CGSOm-CGD-RBFNN	$5.2 \times 10^{-3} \pm 0.0018$	0.9950	$8.2 \times 10^{-3} \pm 0.0012$	0.9895
CGSOm-GD-RBFNN	1.85×10^{-2}	0.8701	1.1×10^{-2}	0.8901
CGSOm-PSO-RBFNN	5.71×10^{-2}	0.9589	3.57×10^{-2}	0.9692
CGSO-CGD-RBFNN	3.5×10^{-1}	0.9521	1.71×10^{-1}	0.9712
Kmeans-GD-RBFNN	3.5×10^{-3}	0.6567	1.5×10^{-2}	0.6767
Kmeans-CGD-RBFNN	7.99×10^{-3}	0.9908	5.20×10^{-3}	0.9421
Kmeans-PSO-RBFNN	1.25×10^{-3}	0.9859	1.0×10^{-2}	0.9590

Next, the test data is used on the models to determine their predictive accuracy on unseen data.For the Open stock Price data, the CGSO-BSO RBFNN gives MMSE= 2.74×10^{-2} with a R^2 value of 0.9732. The CGSOm-BSO-RBFNN model yields MMSE = 1.8×10^{-2} and R^2 =0.9837. Also, it was observed that the CGSOm-CGD RBFNN yields 7.8×10^{-3},while PCA-CGSOm-CGD RBFNN yields MMSE of 7.9×10^{-3}.This again indicates that the PCA does not have much influence on the result of the CGSOm-CGDtrained RBFNN.A similar trend is observed for Close price data. This shows that the RBFNN models are generalizing well for unseen data.

Taylor (2006) noted that traditionally, it is accepted that the best forecast model is that with the smallest overall error measurement value using the test data. In order words, the predictive accuracy of a model can be measured by the mean squared error on the test set. This is the conventional method used by most researchers in validating their predictive models.

Using test dataset, the CGSOm-CGD RBFNN and CGSOm-BSO RBFNN seem tohavelower error values compared to the CGSOm-PSO RBFNN. However, the CGD variant has a slightly lower error with an error value of $7.8x10^{-3}$ compared to the BSO variant with error of $1.8\ x10^{-2}$. The BSO slightly outperformsthe PSO variant having an error of $4.4\ x\ 10^{-2}$.

4.7Case 2: Currency Exchange Rate Forecasting problem

In this case, currency exchange rate data was used to develop the RBFNN models.The currency exchange rate datafrom (PACIFIC Exchange Rate Service, 1996)was used and it is composed of real data representing the weekly averaged exchange rates between British pound and US dollar during the period going from 31 December 1979 to 26 December 1983. The time series plot of the data is in Figure 17.

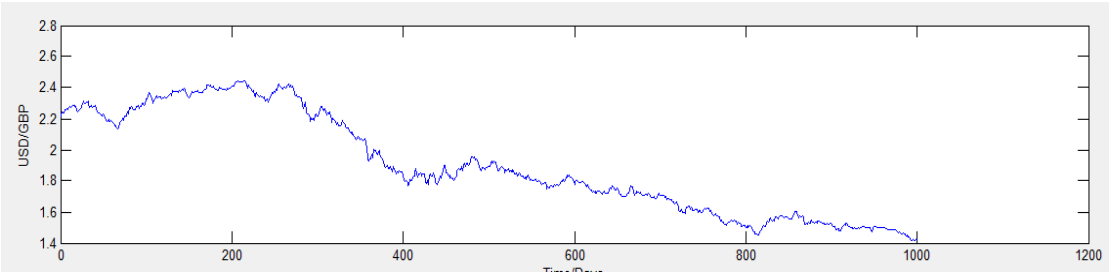


Figure 17: Time series plot of the Currency Exchangedata

4.7.1Parameter Settings

he parameter settings for the CGSO and CGSOm algorithm as well as for the BSO algorithm used for Currency Exchange Rate Forecasting problem are in Tables 11 and 12.

Table 11: Parameters settings for CGSO and CGSOm algorithm used for Currency

Exchange Rate Forecasting problem

S/n	Parameter	Value
1	glowworms population	3500
2	Initial Luciferin,L0	1
3	Decision range, r0	0.2
4	luciferin decay constant,rho	0.3
5	luciferin enhancement constant, ɣ	0.05
6	constant value, B	0.04
7	step size, s	0.3
8	sensor range,r_s	0.25
9	number of neighbours, nt	4
10	maxIter	1800

Table 12: Parameters settings for BSO algorithm For Currency Exchange Rate Forecastingproblem

S/no	Parameter	Value
1	glowworms population	2000
2	Initial Luciferin,L0	1
3	Decision range, r0	0.3
4	luciferin decay constant,rho	0.3
5	luciferin enhancement constant,ɣ	0.05
6	constant value,B	0.04
7	step size,s =	0.4
8	sensor range,rs	0.25
9	global best attraction constant, cg	0.3
10	maxIter	1000
11	adaptive step sizing control, cs	0.2
12	mass extinction control, eT	30
13	strong local search control, IR	10

4.7.2Plots of Optimized RBFNN Models

Figures 18 and 19show the time series plot and regression plot of actual and predicted currency exchange rate respectivelyusing CGSOm-BSO trained RBFNN. Figures 20 and 21display the time series plot and regression plot of actual and predicted currency exchange rate respectively using CGSO-BSO trained RBFNN.

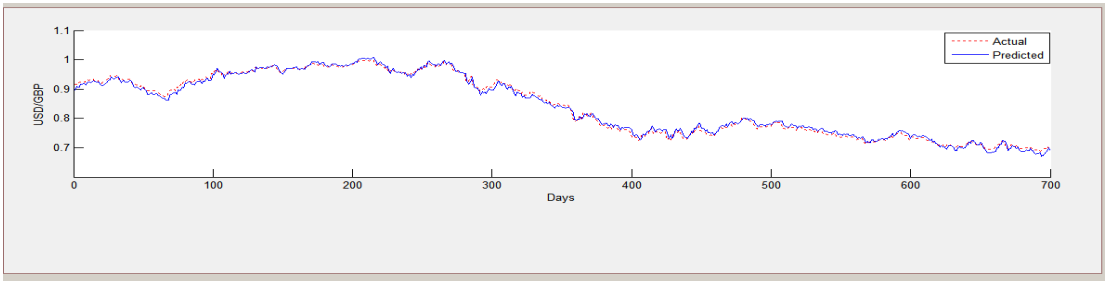


Figure 18: Time series plot of actual values vs. predicted currency exchange rate using CGSOm-BSO trained RBFNN

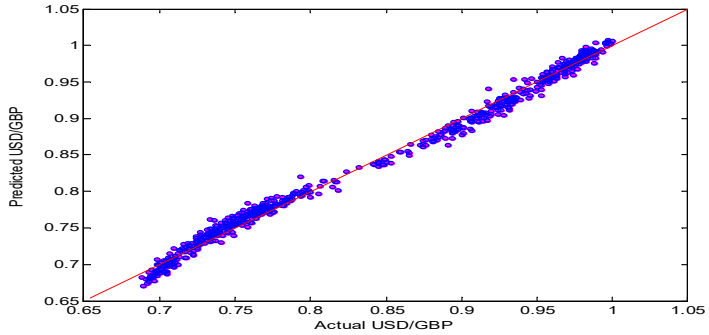
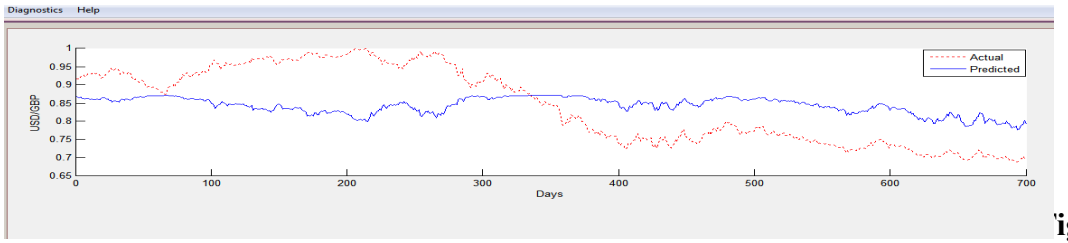


Figure 19:Regression plot of actual and predicted currency exchange rate using CGSOm-BSO trained RBFNN



igure 20: Time series plot of actual values vs. predicted currency exchange rate using CGSO-BSO trained RBFNN.

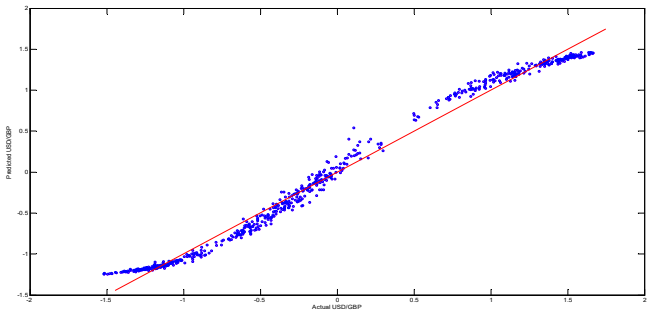


Figure 21: Regression plot of actual and predicted currency exchange rate using CGSO-BSO trained RBFNN

4.7.3Comparative Analysis

This sub-section presents the results of RBFNN models optimized by different techniques and developed using currency exchange rate data. The mean of the mean squared error (MMSE) of 10 runs and the standard deviations for each RBFNN model variant were computed.

In Table13, the summary of results for average of the RBFNN models considered is presented. Experimental results show that usingtest data,the CGSO-BSO trained RBFNN yields a MMSE of $8.3x10^{-3}$with a R^2 value of 0.9219. The CGSOm-BSOtrained RBFNN yields a MMSE of $5.6x10^{-5}$ with a R^2 value of 0.9977. The CGSOm-CGDtrained RBFNN yields MMSE of $4.9x10^{-5}$with a R^2 value of 0.9977, while the PCA-CGSOm-CGD RBFNN yields MMSE value of $7.6x10^{-3}$ indicating that the PCA does not have a positive influence on the result of the CGSOm-CGDtrained RBFNN as the error values increases. Also noted is that the R^2 values are high for these models indicating that all the models fit the data well.

Table 13: Comparative Performance of RBFNN variants for average of 10 simulation runs

RBFNN VARIANTS	**MMSE**	**R^2**
CGSO-BSO-RBFNN. = CONTROL	$8.311x\ 10^{-3}$	0.9219
CGSOm-BSO-RBFNN	$5.64\ x\ 10^{-5}$	0.9977
PCA-CGSOm-BSO-RBFNN	$7.06\ x\ 10^{-5}$	0.9969
CGSO-PSO-RBFNN	$1.4442\ x\ 10^{-4}$	0.9933
CGSOm-PSO-RBFNN	$5.4183\ x\ 10^{-5}$	0.9975
CGSO-CGD-RBFNN	$1.5158\ x\ 10^{-4}$	0.9930
CGSOm-CGD-RBFNN	$4.9362\ x\ 10^{-5}$	0.9977
PCA- CGSOm-CGD-RBFNN	$7.6\ x\ 10^{-3}$	0.9807
CGSOm-GD-RBFNN	$6.5\ x10^{-3}$	0.9658
Kmeans-GD-RBFNN	$8.8x10^{-3}$	0.9902
Kmeans-CGD-RBFNN	$3.6602\ x\ 10^{-5}$	0.9983
Kmeans-PSO-RBFNN	$1.4700\ x\ 10^{-4}$	0.9932

The CGSOm-BSO RBFNN has MMSE of $5.6\ x10^{-5}$; CGSOm-CGD RBFNN hasMMSE of $4.9\ x10^{-5}$; the CGSOm-PSO-RBFNN has MMSE of $5.4x10^{-5}$; all three have similar error values.

4.8Validation of approach

This section presents a benchmarkingof the approaches from this study against existing RBFNN models. The models considered are Evolving Radial basis Function Neural Network

(EvRBF)(Rivas *et al*., 2004); the Auto Regressive-RBF tuned using GA (Sheta and De Jong, 2001), the-Auto Regressive model tuned using Least Square Estimate (LSE). These models were chosen because they were designed for the same application domain as this study, which is time series forecasting. The RBFNN models developed in this study were trained using same dataset that the afore-mentioned 3 models applied.

Table14: Comparative Performance of RBFNN variantsbased on proposedand Existing approaches.

Dataset	**AR-RBF tuned using-LSE**	**AR-RBF tuned using-GA**	**EvRBF**	**CGSOm-BSO-RBFNN**	**CGSOm-CGD-RBFNN**
Training (MSE)	7.2601×10^{-4}	5.1407×10^{-4}	3×10^{-4}	5.64×10^{-5}	4.9×10^{-5}
Full dataset (MSE)	12.001×10^{-4}	8.7220×10^{-4}	6×10^{-4}	8.03×10^{-5}	7.1×10^{-5}

Simulations were done for 10 runs as initiated in EvRBF(Rivas *et al*.,2004). Table 14 gives the MSE computed over the training and full data sets for all the models. The results indicate lower error values for CGSOm-BSO-RBFNN andCGSOm-CGD-RBFNN when compared to the ones obtained from the other 3 benchmark models. For example, using the training datasets, the AR-RBF tuned using LSE gave MSE=7.2601×10^{-4}; the AR-RBF tuned usingGA gave MSE = 5.1407×10^{-4};EvRBFgave MSE = 3×10^{-4}, the CGSOm-BSO-RBFNN yielded MSE = 5.64×10^{-5} andCGSOm-CGD-RBFNNyieldedMSE=4.9×10^{-5}.These results validate the modelsdeveloped in this study for optimizing the RBFNN parameters.

CHAPTER FIVE

SUMMARY OF FINDINGS, CONCLUSION,CONTRIBUTIONS TO KNOWLEDGE AND FURTHER WORK

5.1Summary of Findings

Based on the discussions of results in Chapter four, Table15 shows the summary of findings

Table 15: Summary of Findings

Objectives	**Findings/Results**
For objective 1: To efficiently determine the sensor range of the CGSO algorithm.	The CGSOm algorithm was developed. The CGSOmcomputes efficiently the sensor range (r_s) automatically, modified the glowworm initialization method and introduced a function that measures the cluster error during the iteration phase. Results showed the effectiveness of the CGSOm against the CGSO, and existing standard clustering techniques used as benchmarks. Using cluster quality evaluation measures of Entropy, Purity and Rand Index values; CGSOm gave best entropy and purity values in four of the seven datasets clustered (57%), CGSO gave best results in two datasets (28.5%), and HC gave best result in one dataset(14.5%).Additionally, it was noted that CGSOm produced clusters that agree to a large degree with the ground truth since it gave rand index value of 0.70 and above in six of the the seven datasets considered.
For objective 2: To automatically determine the optimal number of clusters in a dataset.	It was observed that the way the glowworms are initialized plays a vital role in determining sensor range, r_s correctly and consequently determines the number of clusters found. Hence, the glowworm

	initialization method of CGSOm was compared with that of CGSO. It was observed that the method based on CGSOm produced a better curve for determining sensor range,r_s than that obtained using the initializationmethod of CGSO.This was observed from the several clustering experiments carried out.For instance, for the mouse data, the sensor range (based on CGSO glowworm initialization method) is 0.218; whereas that (based on CGSOm glowworm initialization method) is 0.173. The number of clusters for the CGSO is 2; while that for the CGSOm is 3, which is the same number of clusters (ground truth) in the original mouse data. This proves that the way the glowworms are initialized plays a vital role in determining sensor range,r_s correctly and consequently determines the number of clusters found. This CGSOm glowworm initialization method contributes to the performance of the sensor range determination algorithm.
For objective 3: To develop a RBFNN model that adapts to the number of clusters in a dataset.	As soon as the number of clusters is determined, the topology of the network adapts to this number, resulting in an neural network with an adaptive architecture.

For objective 4: To optimize the RBFNN parameters fully	Two new training methodologies for optimizing the RBFNN parameters fully resulted from this work, yielding the CGSOm-CGD RBFNN and CGSOm-BSO RBFNN models. These are new and optimal RBFNN models for time series forecasting problems. Comparing the performance of these models with existing models, results obtained showed that the CGSOm-CGD RBFNN and CGSOm-BSO RBFNN gave betterforecasting accuracy by yielding lowest error values.

5.2Conclusion

The overall goal of this work is to optimize fully the RBFNN models in such a manner that the limitations of existing models are addressed.The clustering aspect of theRBFNN learning process was improved upon; in this case,an improved version of CGSO,the CGSOm,was proposed. The CGSOm solves the challenge in RBFNN optimization as it finds the number of clusters in an efficient manner and fixes the configuration of the network in an adaptable manner.The CGSO was improved upon by incorporating an algorithm for determining the sensor range automatically, modifying the glowworm initializationmethod, and introducing a function that measures cluster error during the iteration phase. It was shown that the modified initialization phase improves the performance of the algorithm that determines the sensor range. It was also demonstrated that the computed sensor range in CGSOm leads to better cluster quality for most data sets when compared with other existing clustering techniques.

optimizing the weights, the fact that the BSO based RBFNN model competes favourably with the standard CGD based RBFNN, thereby leading to development of two new RBFNN training methodologies-CGSOm-BSO and CGSOm-CGD.The newmodels determine automatically the optimal number of RBF centres in a given problem, the number of hidden neuronsin the network and the configuring of the network.This result yields high forecastaccuracy.

5.3Contributions to Knowledge

This study made the following contributions to knowledge:

(1) A new clustering algorithm, CGSOm was developed. It incorporates an efficient

mechanism for determining the sensor range (sensor range determination algorithm), in place of the existing trial and error method.It includes a function that measures the cluster error during the iteration phase, and an improved glowworm initialization method that assists in obtaining the optimal number and quality of clusters.

(2) This work derived two new, efficient and adaptive techniques, namely the CGSOm-CGD-RBFNN and CGSOm-BSO-RBFNN models for training the Radial basis Function Neural Network. These models are major contributions to the statistical and machine learning community and will be of benefit to all those domains and sectors involved in time-series forecasting.

(3) The efficient mechanism for determining the sensor range, the sensor range determination algorithm labeled as “Algorithm 7” was created from this research work.This algorithm helps the CGSOm to determine the number of clusters in an efficient manner.

5.4 Further Work

(1) The fitness function significantly affects the cluster quality. Further work can be done as regards finding a more effective and efficient fitness function.

2) In this work, the CGSOm has been applied in time-series forecasting. Its effectiveness could be investigated in big data mining.

REFERENCES

m, A. Das, S. Roy, S. (2008) Swarm Intelligence Algorithms for Data Clustering, In Maimon**,** O. Ro
. (Eds.),*Soft Computing for Knowledge Discovery and Data Mining*, Springer, US, 279-313.

, I. and Ludwig, S. (2013) A New Clustering Approach based on Glowworm Swarm Optimization, *2013IEEE Congress on Evolutionary Computation, Cancun, 2642-2649,* doi: 10.1109/CEC.2013.6557888

in, E. (2010) *Introduction to Machine Learning*, 2nd Edition, Massachusetts: The MIT Press, Cambridge, Boston, USA.

ɔ, G. and Farmani, M. R. (2014) Clustering Analysis with Combination of Artificial Bee Colony Algorithm and k-Means Technique, *International Journal of Computer Theory and Engineering*, ***6***(2), 141-145.

M. Pomares, H. Rojas, I. Salameh, O. and Hamdon, M. (2009) Prediction of Time Series Using RBF Neural Networks: A New Approach of Clustering, *The International Arab Journal of Information Technology*, ***6***(2), 138-144.

M. (2010) Optimization RBFNNs Parameters using Genetic Algorithms: Applied on Function Approximation, *International Journal of Computer Science and Security (IJCSS),***4** (3), 295 – 307.

e, A. and Ajila, S.A. (2013) Cloud Client Prediction Models for Cloud Resource Provisioning in a Multitier Web Application Environment, *7th IEEE International Symposium on service-Oriented System Engineering (IEEESOSE 2013),* San Francisco Bay, USA

ıvid, S. (2014) Theoretical Foundations of Clustering – Few Results, Many Challenges*, Tutorial Machine Learning Summer School 2014 Workshop*, Remnin University, Beijing, China.

, S. and Sathya, S.S. (2012) A Survey of Bio inspired Optimization Algorithms,*International Journal of Soft Computing and Engineering (IJSCE),* Vol.**2** (2), 2231-2307.

o, D. Maurer, M. Da-Costa, G. Pierson, J. and Brandic, I. (2012) Energy-Efficient andSLA-Aware Management of IaaS clouds, *3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (E-Energy),* Madrid, Spain.

head, D.S and Lowe, D. (1988) Multivariate Functional Interpolation and Adaptive Networks, *Complex Systems*, ***2***, 321-355.

-F. and Horng M. (2014) Firefly algorithm for training the radial basis function network in ultrasonic supraspinatus image classification, *Computer Modelling & New Technologies,* **18**(3), 77-83.

go, P. (2012) A Few Useful Things to Know about Machine Learning, *Communications of the ACM*, ***55***(10), 78-87.

and Zhang, N. (2008) Time series prediction using evolving radial basis function networkswith new encoding scheme, *Journal of Neurocomputing*,**71**(7-9), 1388-1400.

(n.d.) **E**nvironment for Deve**L**oping **K**DD-Applications Supported by **I**ndex-Structures, Retrieved from: http://elki.dbs.ifi.lmu.de/wiki/DataSets [Accessed on 20th March, 2016]

recht, A., (2007) *Computational Intelligence, An Introduction,* 2nd Ed, England: John Wiley and Sons Ltd, England, Great Britain.

H and Krishnapuram, R. (1999) A Robust Competitive Clustering Algorithm with Applications in Computer Vision, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **21** (5), 450-465.

l. Peng, H. and Dong, X. (2012) A Hybrid Algorithm to optimize RBF Network Architecture and Parameters for Nonlinear Time Series Prediction, *Applied Mathematical Modelling,**36***(7), 2911–2919.

ez-Osuna, R. (2014) CSCE 666 Pattern Analysis, Texas A&M University, Retrieved from: *http://research.cs.tamu.edu/prism/lectures/pr/pr_l19.pdf* [Accessed on 25th May,2014].
Kamber, M. and Pei, J. (2012) *Data Mining: Concepts and Techniques*. 3rd edition. CA, USA: Morgan Kaufmann Publishers, CA, USA.

Handl, J. and Meyer, B. (2007) Ant-Based and Swarm-Based Clustering, *Swarm Intell.**1***, 95-113.

, T. Cui, X. Ragade, R. K. Graham, J. H. and Elmaghraby, A. S. (2004) A Modified Particle Swarm Algorithm for Robotic Mapping of Hazardous Environments, *The 2004 WorldAutomation Congress,* SEVILLE, Spain.

ım D. S. and Shmoys, D. B. (1985) A Best Possible Heuristic for the k-Analysis, *Journal of* Problem, *Mathematics of Operations Research*, ***10***(2), 180–184.

Z and Zhou, Y. (2011) Using Glowworm Swarm Optimization Algorithm for ClusteringAnalysis, *Journal of Convergence Information Technology, **6***(29*),* DOI: 10.4156/jcit

R. Fasina, E.P. Alienyi, C.D. and Uwadia, C.O. (2015) Predicting the Occurrence of Rainfall using Improved Radial Basis Function Neural Network, *Journal of the Computer Science and its Applications: An International Journal of the Nigerian Computer Society,**22***(2), 66-72.
ınnis, Nicolaos B., and Randolph-Gips, Mary M. (2003) The Construction and Training of Reformulated Radial Basis Function Neural Networks, *IEEE Transactions on Neural*

Networks, **4**, 835-844.

wda, A. G. and Prasad, M. (2013) A Survey of Applications of Glowworm Swarm Optimization Algorithm, *International Journal of Computer Applications (0975 – 8887) International Conference on Computing and information Technology (IC2IT-2013), 39-42*

ly, J. and Eberhart, R. (1995) Particle Swarm Optimization, *Proceedings of IEEE International Conference on Neural Networks*, 1942–1948.

E. (2003) Data mining and machine learning in time series databases, *Tutorial ECML/PKDD-2003: Fourteenth European Conference on Machine Learning and Seventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, Cavtat-Dubrovnik, Croatia.

N. Lee, C-M. (2009) Time series prediction using RBF neural networks with anonlinear time-varying evolution PSO algorithm, *Journal Neurocomputing,* **73** (1-3), 449-460.

en, T. (2003) *Learning Vector Quantization: The Handbook of Brain Theory and Neural Networks*, 2nd Edition, Massachusetts: The MIT Press, MA, USA, 631-634

anand, K. and Ghose, D. (2005) Detection of Multiple Source Locations using a Glowworm Metaphor with Applications to Collective Robotics, *Proceedings of the IEEE Swarm Intelligence Symposium,* CA, USA, 84 – 91.

ı, E. and Fornberg, B. (2005) Theoretical and Computational Aspects of Multivariate Interpolation with increasingly Flat Radial Basis Functions, *Computers & Mathematics with Applications,***49**(1), 103–130.

Y. Zhang, J. and Xu, Z. (2000) Clustering by Space-Space Filtering, *IEEE Transactions on Pattern Analysis and Machine Intelligence,***22** (12), 1396-1410.

ın, M. (2013) UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, CA, USA.

-F. and Chen, L.-H. (2005) Time Series Forecasting by Combining the Radial Basis Function Network and the Self-Organizing Map, *Hydrological Processes*, ***19***(10), 1925–1937.

een, J. (1967) Some Methods for Classification and Analysis of Multivariate Observations, in *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.* 1, University of California Press, CA, USA, 281–297.

ll, T.M. (1997) *Machine Learning,*Massachusetts: The MIT Press, Cambridge, Boston, USA.

, E. and Darken, C. (1989) Fast Learning in Networks of Locally Tuned Processing Units, *Computer Journal Neural Computation*, ***2***(1), 281-294.

car,V. and Beheshti, M.T. (2009) A Local Linear Radial Basis Function Neural Network for Financial Time-Series Forecasting, *Appl Intell (*2010), Springer US, ***33***(3), 352-356.

, M. Salman, A. and Engelbrecht, A. P. (2002) Image classification using particle swarm optimization, Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, (SEAL 2002),370-374.

J. González, M. Salmeron, A. Prieto, A. Pomares, H. Ros, A. and Rojas, I. (2000) A New Radial Basis Function Networks Structure: Application to Time Series Prediction, *Proceedings of IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN-2000),* Italy, 445-449.

IC Exchange Rate Service (1996) Sauder School of Business, University of British Columbia, Retrieved from: http://pacific.commerce.ubc.ca/xr/data.html [Accessed on 30th March, 2016]

I.V.S. (2014) *Optimization Methods for Engineers*, Delhi, India: PHI Learning Private Limited, Delhi, India.

W.M. (1971) Objective criteria for the evaluation of clustering methods, *Journal of the American Statistical Association. American Statistical Association*,***66*** (336), 846–850.

V.M. Castillo,P.A. Merelo,J.J. (2002) Evolved RBF Networks for Time-Series Forecasting and Function Approximation. In: Guervós, J.J.M. Adamidis P. Beyer HG. Schwefel HP. Fernández-Villacañas JL. (eds) Parallel Problem Solving from Nature — PPSN VII. PPSN 2002. Lecture Notes in Computer Science, 2439. Springer, Berlin, Heidelberg

V.M. Merelo, J.J. Castillo, P.A. Arenas, M.G. and Castellano, J.G. (2004) Evolving RBF Neural Networks for Time-Series Forecasting with EvRBF, *Information Sciences*, **165**, 207–220.

› de Oliveira, D. Parpinelli, R.S. and Lopes, H.S. (2011) Bioluminescent Swarm Optimization Algorithm, *Evolutionary Algorithms*, Prof. Eisuke Kita (Ed.), InTech, ISBN: 978-953-307-171-8

ıg, H. and Binwang, H. (2002) A New Algorithm of Selection the Radial BasisFunction Networks Center, *Proceedings of the First International Conference on Machine Learning and Cybernetics,* Beijing, 1801-1804.

ar, P. Jayaraman, V. and Kulkarni, B. (2004) An ant colony approach for clustering, *Analytica Chimica Acta*, ***509***(2), 187 – 195.

/. Guo, X. Wu C. Wu, D. (2011) Forecasting stock indices using Radial Basis Function Neural Networks optimized by Artificial Fish Swarm algorithm, *Knowl.-Based Syst **24,**378-385.*
A.F. and De Jong, K. (2001) Time-series Forecasting using GA-tuned Radial Basis Functions, *Information Sciences,* Elsevier, ***133***(3), 221-228(8),

D. Li, X. Hang, X. Liwen, Z. (2010) Research and Progress of Cluster Algorithms based on Granular Computing , *JDCTA,**4***(5), 96-104.

P. M. and Bapat, A. U. (2013) Clustering Algorithms for Radial Basis Function Neural Network, *ITSI Transactions on Electrical and Electronics Engineering* (*ITSI TEEE),**1***(1), 113-116.

S. Zhigang, Y. Chen, X (2005) A Novel Radial Basis Function Neural Network for Approximation, *International Journal of Information Technology*, ***11***(9), 110-118.

B.J. (2006) Methods and Procedures for the Verification and Validation of Artificial Neural Networks, Springer USA.

N.X. Epps, J. and Bailey, J. (2009), Information Theoretic Measures for Clustering Comparison: Is a Correction for Chance Necessary?, *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning, ACM*, 1073–1080.

O! FINANCE (2009) [Online]: http://finance.yahoo.com/quote/GE/history?ltr=1 , [Accessed on 27th March, 2016]

ıarayana, B. (2010) *Artificial Neural Networks,* New Delhi: PHI Learning Private Limited, New Delhi, India.

Y. and Karypis, G. (2002) Evaluation of hierarchical clustering algorithms for document datasets," in *Proceedings of the eleventh CIKM '02*, NY, USA, 515–524.

Y. Yan, E. Li, C. and Li, Y. (2008) Application of Multivariable Time Series Based on RBF Neural Network in Prediction of Landslide Displacement, *The 9th International Conference for Young Computer Scientists, ICYCS 2008,* 2707 – 2712.

C. Ouyang, D. and Ning, J. (2010) An Artificial Bee Colony Approach for Clustering, *Expert Systems with Applications*, ***37***, 4761–4767.

. (2009) Nonlinear Time Series Prediction by Using RBF Network, *Advances in Neural Networks – ISNN 2009Lecture Notes in Computer Science*, ***5551***, 901-908.

APPENDIXA

```
function varargout = Predictor(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Predictor_OpeningFcn, ...
                   'gui_OutputFcn',  @Predictor_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
handles.output = hObject;
handles.VarNames = {};
%set(handles.summaryTable,'data',[handles.tableVar,{'';'';'';''},{'';'';''
;''}])
handles.noVar = 4;
handles.featureSize = 12;
handles.noSamples = [];
handles.data = [];

handles.rbfNames = {'Gaussian', 'Multi-Quadric', 'Inverse Multi-
Quadric','Thin Plate Spline','Cubic', 'Linear'};
handles.rbfcns =
{'gaussRBF','multiQuadRBF','invMultiQuadRBF','thinPlateSplineRBF','cubicRB
F','linearRBF'};
set(handles.rbFunctions, 'string', handles.rbfNames)

optData, optNames, optH] = ExtractOptFcn('optsettings.txt');
handles.optfcnsString = optNames;
handles.optfcnsData = optData;
handles.optfcnsHandle = optH;
set(handles.optMenu,'string', optNames);
set(handles.optMenu,'string',handles.optfcnsString)

% Extracting the clustering function from file
[clusterData, clusterNames, clusterH] =
ExtractOptFcn('clustersettings.txt');
handles.clusterData = clusterData;
handles.clusterNames = clusterNames;
handles.clusterHandles = clusterH;
set(handles.clustFunctions,'string',handles.clusterNames)

handles.actualOptFcn = {};
handles.timeAgo = 3;
handles.currentRBF = handles.rbfcns{2};

handles.lambda = 0.00001;
handles.noHidden = 10;
handles.fracTrainSet = 0.7;
handles.fracTestSet = 0.1;
handles.fracValSet = 1 - handles.fracTestSet - handles.fracTrainSet;

% Default values
defaultVar.dataPartition = [0.7,0.2];
defaultVar.timeAgo = 3;
defaultVar.numHidden = 10;
defaultVar.lambda = 0.000001;
defaultVar.rbf = 1;
defaultVar.pcaDim = 0;
defaultVar.optimFcn = 1;
defaultVar.clusterFcn = 1;

handles.defaultVar = defaultVar;

handles.plotHandles = [];
handles.plotHolderInitPos = get(handles.primaryPlotHolder,'position');
set(handles.regTerm,'string',num2str(handles.lambda));

handles.fsMethod = 1;
```

```
handles.dataMins = [];
handles.dataMaxs = [];
handles.dataMeans = [];
handles.dataSDs = [];

handles.allX = [];
handles.allY = [];
handles.allXRaw = [];
handles.allYRaw = [];

handles.maintainDim = 0;
handles.PC = [];

%plotdata = {(1:10)',(1:10)','r--','X','Y';(1:10)',(1:10)','go-
','X','Y';(1:10)',(1:10)','b*-','X','Y';(1:10)',sin(1:10)','k*-','X','Y'};
%plotHandler(plotdata,handles);

guidata(hObject, handles);

% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Predictor_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --------------------------------------------------------------------
function aboutMenu_Callback(hObject, eventdata, handles)

aboutHdles.figure1 =
figure('parent',0,'menubar','none','units','characters',...
'toolbar','auto','handlevisibility','callback','color',[0.831  0.816
0.784],...
'position',[102.8   15.23077  130.8   39.53846],...
'visible','on','windowstyle','normal','name','About','resize','off',...
'windowstyle','normal');

aboutHdles.uipanel1 =
uipanel('parent',aboutHdles.figure1,'fontsize',[8],...
'units','characters','fontweight','normal','foregroundcolor',[0  0  0],...
'fontangle','normal','backgroundcolor',[1       1       1],...
'BorderType','beveledout','position',[9.8      6         110.4
25],...
'visible','on','title','','BorderWidth',[3],'TitlePosition','lefttop');

aboutHdles.text1 =
uicontrol('parent',aboutHdles.uipanel1,'style','text','fontsize',[13],...
'units','characters','fontweight','bold','foregroundcolor',[0  0  1],...
'fontangle','normal','backgroundcolor',[1           1         1],...
'horizontalalignment','center','position',[4.6     0.8           100
23],...
```

```
'visible','on','string',{'  ','This program is designed for a PhD Thesis
titled:','  ',...
'A Framework for the Automatic Generation of an Optimal Radial Basis
Function Neural Network For Time Series Data Forecasting Problems ','
',...
'Author''s Name: Roselyn Isimeto',' No: ','Department: Computer
Science','Institution: University of Lagos','Email: '},...
'enable','on','tooltipstring','');

fid = fopen(filename, 'r');
optNames = {};
optH = {};
optData = [];
handlePattern = 'opt';
counter = 0;
currentTag = '';
currentData = struct();
currentOptions = struct();
currentOptionType = '';
currentSetField = {};
nameTag = 'name:';
funcTag = 'function:';
optionTypeTag = 'option_type:';
optionTag = 'options:';
allTags = {nameTag, funcTag, optionTypeTag, optionTag};
while true
    aline = fgetl(fid);
    if (aline==-1)
        break;
    end
    aline = strtrim(aline);
    if isempty(aline)
        continue;
    end
    if any(strcmpi(aline, allTags))
        currentTag = aline;
        if strcmpi(aline, nameTag)
            currentData = struct();
            currentOptions = struct();
            currentOptionType = '';
            currentSetField = {};
            counter = counter + 1;
        end
        continue;
    end

    if strcmpi(currentTag,nameTag)
        optNames = [optNames,{aline}];
        anOptH = [handlePattern,num2str(counter)];
        optH = [optH,{anOptH}];
        currentData.name = aline;
        currentData.options = currentOptions;
    elseif strcmpi(currentTag, optionTypeTag)
        currentOptionType = aline;
        if ~strcmpi(currentOptionType,'general')
            currentOptions = eval(aline);
        end
    elseif strcmpi(currentTag, funcTag)
        currentData.function = aline;
    elseif strcmpi(currentTag, optionTag)
        [field,val] = strtok(aline,'=');
```

```
        [val,notin] = strtok(val,'=');
        if ~isempty(str2num(val))
            val = str2num(val);
        end
        currentOptions.(field) = val;
        currentData.options = currentOptions;
        currentSetField = [currentSetField, {field}];
        currentData.setfield = currentSetField;
    end

    optData.(anOptH) = currentData;

end
fclose(fid);

% --------------------------------------------------------------------
function lcMenu_Callback(hObject, eventdata, handles)



function [dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf,
optimFcn, clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles)
dataPartition = handles.defaultVar.dataPartition;
timeAgo = handles.defaultVar.timeAgo;
numHidden = handles.defaultVar.numHidden;
lambda = handles.defaultVar.lambda;
rbf = handles.rbfcns{handles.defaultVar.rbf};
pcaDim = handles.defaultVar.pcaDim;
[optimFcn, optimOptions] = GetOptimFcn(handles.defaultVar.optimFcn,
handles);
[clusterFcn, clusterOptions] =
GetClusterFcn(handles.defaultVar.clusterFcn, handles);



function [clusterFcn, clusterOptions] = GetClusterFcn(pos, handles)
% The clustering function
clusterData = handles.clusterData;
clusterHandle = handles.clusterHandles;
clusterFcnObj = clusterData.(clusterHandle{pos});
clusterFcn = clusterFcnObj.function;
clusterOptions = clusterFcnObj.options;

function [optimFcn, optimOptions] = GetOptimFcn(pos, handles)
% The optimisation function
optData = handles.optfcnsData; % Optimisation data holder
optH = handles.optfcnsHandle;  % A list of handles to optimization object
curOptObj = optData.(optH{pos});  % The current optimization function
position
optimFcn = curOptObj.function;
optimOptions = curOptObj.options;

function hlMenu_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);
inputdata = inputdlg('Provide number of hidden nodes');
if isempty(inputdata)
    return
end
xdata =  str2num(inputdata{1});
```

```
sz = length(xdata);
if isempty(xdata)
    return
end
numHidden = num2cell(xdata);
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 4, sz,
xdata);

% --------------------------------------------------------------------
function regMenu_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);
inputdata = inputdlg('Provide range of regularisation term');
if isempty(inputdata)
    return
end
xdata =  str2num(inputdata{1});
sz = length(xdata);
if isempty(xdata)
    return
end
lambda = num2cell(xdata);
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 5, sz,
xdata);

function UpdateInterface(theState, handles)

hdlesToEnable =
[handles.trainBtn,handles.validateBtn,handles.testBtn,handles.predictBtn,
handles.allData];
sz = length(hdlesToEnable);

for h = hdlesToEnable;
    set(h, 'enable', theState)
end
if strcmpi(theState,'on')
    oppState = 'off';
else
    oppState = 'on';
end
set(handles.pleaseWait,'visible',oppState); % The "Please Wait" shows up
drawnow


% --- Executes on button press in trainBtn.
function trainBtn_Callback(hObject, eventdata, handles)
try
    tic                % The timer ticks
    UpdateInterface('off', handles);

    noRun = str2double(get(handles.runEditBox,'string'));
    wbar = waitbar(0,char('Please wait...',['Run 0/', num2str(noRun)]));
    lambda = str2double(get(handles.regTerm,'string'));  % The
regularisation term is fetched from its control
    numHidden = str2double(get(handles.numHiddenLayer,'string')); % The
number of hidden layer is fetched from its control
    trainR = str2num(get(handles.numTrainSet,'string'))/100;
    testR = str2num(get(handles.numTestSet,'string'))/100;
```

```
    valR = 1 - trainR - testR;
    handles = ShuffleData(handles);
    allY = handles.allY;
    allX = handles.allX;
    tempAllX = allX;
    %ReverseFeatureScaling(allY, handles);
    handles.pcaUsed = 0;
    if get(handles.applyPCARadBtn,'value') % Should pca be applied
        noComp = str2num(get(handles.pcaDimTextBox,'string')); % Get the
new feature size or dimension
        if isempty(noComp)
            set(handles.pleaseWait,'visible','off')
            errordlg('The PCA number of dimension is invalid','Error
message','modal');
            return
        end
        [allX, PC] = pca(allX,noComp);
        handles.PC = PC(:,1:noComp);
        handles.pcaUsed = 1;
    end

    actualSamSize = handles.actualSamSize;
    noTrain = floor(trainR*actualSamSize);
    noTest = floor(testR*actualSamSize);
    noVal = actualSamSize - noTrain - noTest;
    handles.trainR = trainR;
    handles.testR = testR;
    handles.valR = valR;
    handles.lambda = lambda;
    handles.noHidden = numHidden;
    handles.noTVT = [noTrain,noVal,noTest];
    trainSetY = allY(1:noTrain,:)
    trainSetX = allX(1:noTrain,:)
    valSetY = allY(noTrain+(1:noVal),:);
    valSetX = allX(noTrain+(1:noVal),:);
    testSetY = allY(noTrain+noVal+(1:noTest),:);
    testSetX = allX(noTrain+noVal+(1:noTest),:);

    handles.maintainX = tempAllX(1:noTrain,:);

    % The clustering function
    clusterData = handles.clusterData;
    clusterHandle = handles.clusterHandles;
    clusterFcnObj =
clusterData.(clusterHandle{get(handles.clustFunctions,'value')});
    clusterFcn = clusterFcnObj.function;
    clusterOptions = clusterFcnObj.options;

    % The optimisation function
    optData = handles.optfcnsData;
    optH = handles.optfcnsHandle;
    curOptObj = optData.(optH{get(handles.optMenu,'value')});
curOptimFcn = curOptObj.function;
    options = curOptObj.options;

    set(handles.numHiddenLayer, 'enable', 'off')
    meanR = 0;
    meanMSE = 0;
    ishandle(wbar)
    mseList = zeros(noRun, size(allY,2));
```

```
    for ii = 1:noRun;

        if ishandle(wbar)
            waitbar(ii/noRun, wbar,char('Please wait...',['Run
',num2str(ii),'/', num2str(noRun)]));
        end
        param =
RBFTrainingAlgorithm(trainSetX,trainSetY,numHidden,lambda,handles.currentR
BF, curOptimFcn, options, clusterFcn, clusterOptions, handles, true);
        if handles.maintainDim
            yPredicted = predict(handles.maintainX, param);
        else
            yPredicted = predict(trainSetX, param);
        end
        [CofTrain,allMSE] = computeStatistics(trainSetY,yPredicted);
        meanR = meanR + CofTrain;
        meanMSE = meanMSE + allMSE;
        mseList(ii,:) = allMSE(:)';
    end
    if handles.maintainDim
        trainSetX = tempAllX(1:noTrain,:);
        valSetX = tempAllX(noTrain+(1:noVal),:);
        testSetX = tempAllX(noTrain+noVal+(1:noTest),:);
    end
    meanR = meanR/noRun;
    meanMSE = meanMSE/noRun;

    save('mseSaved','mseList');

    plotdata = makePlotData(ReverseFeatureScaling(trainSetY,
handles),ReverseFeatureScaling(yPredicted, handles), handles);
    handles.plotHandles = plotHandler(plotdata,handles);
    plotRegression(trainSetY,yPredicted,handles)
    handles.trainSetY = trainSetY;
    handles.trainSetX = trainSetX;
    handles.valSetY = valSetY;
    handles.valSetX = valSetX;
    handles.testSetY = testSetY;
    handles.testSetX = testSetX;
    handles.param = param;
        delete(wbar);
    end
    tm = toc;

set(handles.summaryTable,'data',[handles.VarNames',num2cell(meanMSE),num2c
ell(meanR)])
    set(handles.timeResult,'string',num2str(tm))
    UpdateInterface('on', handles);
    guidata(hObject,handles);
catch  error
    UpdateInterface('on', handles);
    errordlg(char('An error occurred while training the network','Please
check the input parameters and try again', ...
        ['Error Details: ',error.message]),'Error message','modal');
end

function plotdat = makePlotData(allY,allPY,handles)
sz = size(allPY,2);
nox = size(allPY,1);
xrange = repmat((1:nox)',1,1);
```

```
plotdat{sz,6} = [];
if isempty(allY)
    xrange = repmat((1:nox)',1,1);
    for k = 1:sz;
        plotdat(k,1:6) =
{xrange,allPY(:,k),'r:','Days',handles.VarNames{k},char('Actual','Predicte
d')};
    end
else
    xrange = repmat((1:nox)',1,2);
    for k = 1:sz;
        plotdat(k,1:6) =
{xrange,[allY(:,k),allPY(:,k)],{'r:','b'},'Days',handles.VarNames{k},char(
'Actual','Predicted')};
    end
end

function makePlot(allY,allPY,handles)
axeHdles =
[handles.rainPlot,handles.temPlot,handles.humPlot,handles.windPlot];
ylab = handles.tableVar;
nox = size(allY,1);
for k = 1:4;
    axes(axeHdles(k));
    pl1 = plot((1:nox)',allY(:,k),'r:','linewidth',1);
    hold on
    pl2 = plot((1:nox)',allPY(:,k),'b','linewidth',1);
    legend([pl1,pl2],char('Actual','Predicted'));
    ylabel(ylab{k});
    axis tight
    if k<4
        set(gca,'xtick',[])
    end
    hold off
end
function plotRegression(y,yp,handles)
lab = handles.VarNames;
sz = handles.noVar;
figure
for k = 1:sz;
    subplot(ceil(sqrt(sz)),sz/ceil(sqrt(sz)),k)

plot(y(:,k),yp(:,k),'mo','markerfacecolor','m','markeredgecolor','b','mark
ersize',3)
    hold on

plot(linspace(0.95*min(y(:,k)),1.05*max(y(:,k)),5),linspace(0.95*min(y(:,k
)),1.05*max(y(:,k)),5),'r')
    xlabel(['Actual ',lab{k}])
    ylabel(['Predicted ',lab{k}])
    hold off
end

% --- Executes on button press in testBtn.
function testBtn_Callback(hObject, eventdata, handles)
try
    tic
    UpdateInterface('off', handles);

    testSetY = handles.testSetY;
    testSetX = handles.testSetX;
```

```
    param = handles.param;
    yPredicted = predict(testSetX,param);
    [CofTrain,allMSE] = computeStatistics(testSetY,yPredicted);
    plotdata = makePlotData(ReverseFeatureScaling(testSetY, handles),
ReverseFeatureScaling(yPredicted, handles),handles);
    handles.plotHandles = plotHandler(plotdata,handles);
    plotRegression(testSetY,yPredicted,handles)

    tm = toc;

set(handles.summaryTable,'data',[handles.VarNames',num2cell(allMSE),num2ce
ll(CofTrain)])
    set(handles.timeResult,'string',num2str(tm))
    UpdateInterface('on', handles);
    guidata(hObject,handles);
catch  error
    disp(error.message)
    UpdateInterface('on', handles);
    errordlg(char('An error occurred while testing the network','Please
check the input parameters and try again'),'Error message','modal');
end


% --- Executes on button press in predictBtn.
function predictBtn_Callback(hObject, eventdata, handles)
try
    tic
    UpdateInterface('off', handles);
    noFeatures = handles.featureSize;
    lag = handles.timeAgo;
    allY = handles.allYRaw;
    allX = handles.allXRaw;

    param = handles.param;

    inputDay = inputdlg('How many days away from now would you like to
predict?','Day to predict');

    if ~isempty(inputDay)
        inputDay = str2double(inputDay{1});
        for k = 1:inputDay;
            hldY = allY((end-lag+1):end,:);
            hldX = hldY(:)';
            yPred = predict(hldX,param);
            allY(end+1,:) = yPred;
        end
        yPredicted = allY((end-inputDay+1):end,:);
        yPredicted = ReverseFeatureScaling(yPredicted, handles);
        plotdata = makePlotData('', yPredicted, handles);
        handles.plotHandles = plotHandler(plotdata,handles);
    end
    tm = toc;
    %
set(handles.summaryTable,'data',[handles.VarNames,num2cell(allMSE),num2cel
l(CofTrain)])
    set(handles.timeResult,'string',num2str(tm))
    UpdateInterface('on', handles);
    guidata(hObject,handles);
catch  error
```

```
    disp(error.message)
    UpdateInterface('on', handles);
    errordlg(char('An error occurred while testing the network','Please
check the input parameters and try again'),'Error message','modal');
end


function numTrainSet_Callback(hObject, eventdata, handles)
% hObject    handle to numTrainSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numTrainSet as text
%        str2double(get(hObject,'String')) returns contents of numTrainSet
as a double


% --- Executes during object creation, after setting all properties.
function numTrainSet_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numTrainSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function numTestSet_Callback(hObject, eventdata, handles)
% hObject    handle to numTestSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numTestSet as text
%        str2double(get(hObject,'String')) returns contents of numTestSet
as a double


function numTestSet_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numTestSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function regTerm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to regTerm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function numHiddenLayer_Callback(hObject, eventdata, handles)
% hObject    handle to numHiddenLayer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numHiddenLayer as text
%        str2double(get(hObject,'String')) returns contents of
numHiddenLayer as a double


% --- Executes during object creation, after setting all properties.
function numHiddenLayer_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numHiddenLayer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function gaMax_Callback(hObject, eventdata, handles)
% hObject    handle to gaMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gaMax as text
%        str2double(get(hObject,'String')) returns contents of gaMax as a
double


% --- Executes during object creation, after setting all properties.
function gaMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gaMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function rbFunctions_Callback(hObject, eventdata, handles)
handles.currentRBF = handles.rbfcns{get(hObject,'value')};
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function rbFunctions_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rbFunctions (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in optMenu.
function optMenu_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function optMenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to optMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on key press with focus on numTrainSet and none of its
controls.
function numTrainSet_KeyPressFcn(hObject, eventdata, handles)

% --- Executes on key press with focus on numTestSet and none of its
controls.
function numTestSet_KeyPressFcn(hObject, eventdata, handles)

function popSize_Callback(hObject, eventdata, handles)
% hObject    handle to popSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of popSize as text
%        str2double(get(hObject,'String')) returns contents of popSize as
a double
```

```
% --- Executes during object creation, after setting all properties.
function popSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in timeAgoMenu.
function timeAgoMenu_Callback(hObject, eventdata, handles)
val = get(hObject,'Value');
allStr = get(hObject,'string');
handles.timeAgo = str2double(allStr{val});
featureSize = (handles.timeAgo)*(handles.noVar);
handles.featureSize = featureSize;
if ~isempty(handles.data)
    [allY,allX] = makeFullDataSet(handles.data,handles.timeAgo);
    handles.allY = allY;
    handles.allX = allX;
    handles.actualSamSize = size(allY,1);
end

set(handles.numFeatures,'string',num2str(featureSize));
guidata(hObject,handles);

% Hints: contents = cellstr(get(hObject,'String')) returns timeAgoMenu
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
timeAgoMenu



% --- Executes during object creation, after setting all properties.
function timeAgoMenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to timeAgoMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --------------------------------------------------------------------



% --------------------------------------------------------------------
function popsizeMenu_Callback(hObject, eventdata, handles)
```

```
% --------------------------------------------------------------------
function generationMenu_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function lagMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);
inputdata = inputdlg('Provide range of time lag');
if isempty(inputdata)
    return
end
xdata =  str2num(inputdata{1});
if isempty(xdata)
    return
end
timeAgo = num2cell(xdata);
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 2,
length(xdata), xdata);

%%============================ Helper Functions  ========================

function [filename,ext] = ImportPath
    [flname,flpath,findex] = uigetfile({'*.xls;*.xlsx*','Excel
Files(*.xls,*xlsx)'},'Import Data');
    if findex ~= 0 & flname ~= 0
        filename = fullfile(flpath,flname);
        [a,b,ext] = fileparts(filename);
    else
        filename = 0;
        ext = 0;
    end
function [dat,dataname] = ImportFunction
    [filename,ext] = ImportPath;
    dat = {};
    dataname = {};
    if ischar(filename) & ischar(ext)
        [data,dataname] = xlsread(filename);
        dat = data;
    end

function ret = EuclidDistance(pos1,pos2)
ret = sqrt(sum((pos1-pos2).^2,2));

function SetNoHiddenLayer(no_hidden,handles)
set(handles.numHiddenLayer,'string',num2str(no_hidden), 'enable', 'on');
drawnow

function param =
RBFTrainingAlgorithm(X,y,no_hidden,lambda,rbfcn,optfcn,options,clusterfcn,
clusterOptions, handles, setHidden)

output_layer_size  = size(y,2);
[theMeans,theSDs] = eval(clusterfcn)  % This clusters the data in X
%%theMeans  to see centroids
%% Takes care of redimensioning the centroids to the original dimension of
the input space
if handles.maintainDim
   pc = handles.PC;         % the reduced principal component (PC)
```

```
   theMeans = theMeans*pinv(pc);  % calulating the centroid from the
reduced centroid and PC
   X = handles.maintainX;     % X switches to the original input space
end


no_hidden = length(theSDs);
if setHidden
    SetNoHiddenLayer(no_hidden,handles);
end
hValues = eval([rbfcn,'(X,theMeans,theSDs)']);
initial_Theta = randInitializeWeights(no_hidden, output_layer_size);
initial_params = initial_Theta(:);
no_param = length(initial_params);

costFunction = @(p) nnCostFunction(p,no_hidden,output_layer_size, hValues,
y, lambda);
[nn_params, cost] = eval(optfcn)

Theta = reshape(nn_params, no_hidden+1,output_layer_size);
param = {theMeans,theSDs,Theta,rbfcn};

function [J,grad] = nnCostFunction(nn_params, no_hidden, out_layer_size,
X, y, lambda)

Theta = reshape(nn_params, (no_hidden + 1),out_layer_size);

% Setup some useful variables
m = size(X, 1);
J = 0;
X = [ones(m,1),X];
z = X*Theta;
sqError = (z-y)';
J = sum(sum((z - y).^2))/(2*m) +
lambda/(2*m)*(sum(sum(Theta(2:end,:).^2)));

grad = 1/m*(sqError*X)';
grad(2:end,:) = grad(2:end,:) + lambda/m*Theta(2:end,:);
grad = grad(:);
%
=========================================================================


function yPredicted = predict(X,params)
theMeans = params{1};
theSDs = params{2};
theta = params{3};
rbfcn = params{4};
hValues = eval([rbfcn,'(X,theMeans,theSDs)']);
m = size(hValues,1);
hValues = [ones(m,1),hValues];
yPredicted = hValues*theta;

function W = randInitializeWeights(L_in, L_out)
epsilon = sqrt(6)/sqrt(L_in + L_out);
W = 2*epsilon*rand(1 + L_in,L_out) - epsilon;

function [allCof,allMSE] = computeStatistics(y,ypred)
m = size(y,1);
mse = sum((y-ypred).^2)/m;
```

```
r = corrcoef(y,ypred);
sz = size(y,2);
allCof = zeros(sz,1);
for k = 1:sz;
    R = corrcoef(y(:,k),ypred(:,k));
    %allCof(k) = R(2);
    allCof(k) = R(2)^2;
end
res = ypred - y;
allMSE = mean(res.^2)';

function dataMenu_Callback(hObject, eventdata, handles)

try
    [rawData,dataNames] = ImportFunction;
    if isempty(rawData)
        return
    end
    UpdateInterface('off', handles);
    drawnow;               handles.VarNames = dataNames;
    theMin = min(rawData);
    handles.noVar = nocl;
    plotdata{nocl,5} = [];
    noSamples = size(rawData,1);
    xrange = (1:noSamples)';
    styleList = {'b','r','g','m','c','k','y'};
    stySz = length(styleList);
    dataMeans = mean(rawData);
    dataSDs = std(rawData);
    for k = 1:nocl;
        spos = mod(k-1,stySz) + 1;
        plotdata(k,:) =
{xrange,rawData(:,k),styleList{spos},'Time/Days',dataNames{k}};
        if handles.fsMethod == 1
            rawData(:,k) = (rawData(:,k)-dataMeans(k))/dataSDs(k);
        elseif handles.fsMethod == 2
            rawData(:,k) = rawData(:,k)/theMax(k);
        else
            rawData(:,k) = (rawData(:,k) - theMin(k))/(theMax(k) -
theMin(k));
        end
    end

    handles.plotHandles = plotHandler(plotdata,handles);
    handles.data = rawData;
handles.dataMeans = dataMeans;
    handles.dataSDs = dataSDs;
handles.dataMins = theMin;
handles.dataMaxs  = theMax;
    handles.noSamples = noSamples;
    featureSize = handles.noVar*handles.timeAgo;
    [allY,allX] = makeFullDataSet(rawData,handles.timeAgo);
    handles.allXRaw = allX;
    handles.allYRaw = allY;

    noSam = size(allY,1);
    rangePos = randperm(noSam);
    allX = allX(rangePos,:);
    allY = allY(rangePos,:);
```

```
    mnX = min(max(allX));
    mzX = max(max(allX));
    %disp(['Min X: ', num2str(mnX), ', Max X: ', num2str(mzX)])
    handles.allY = allY;
    handles.allX = allX;
    handles.actualSamSize = size(allY,1);
    handles.featureSize = featureSize;
set(handles.numFeatures,'string',num2str(handles.featureSize));
set(handles.numDataSet ,'string',num2str(noSamples));
    UpdateInterface('on', handles);
catch er

    disp(er.message)
    UpdateInterface('on', handles);
    errordlg(char('An error occurred while importing data','Please check
the data you want to import'),'Error message','modal');
end
guidata(hObject,handles);

function [retY,retX] = makeFullDataSet(data,timeAgo)
% Inputs/Argument:
% data - the raw data
% timeAgo - also known as the lag
% nofactor - feature size
% Returns:
% retY - the output space
% retX - the input space

rw = size(data,1);
cl = size(data,2);

nofactor = cl*timeAgo;

if rw <= timeAgo
    retY = [];
    retX = [];
    return
end
sz = rw-timeAgo;
retY = data((timeAgo+1):rw,:);
retX = zeros(sz,nofactor);
for k = 1:sz;
    hld = data(k-1+(1:timeAgo),:);
    retX(k,:) = hld(:)';
end

function ret = gaussRBF(X,mn,sd)
mnSz = size(mn,1);
samSz = size(X,1);
ret = zeros(samSz,mnSz);

% Recompute SD
sd = computeCentroidWidth(mn);

for k = 1:mnSz;
    amean = mn(k,:);
    thedist = exp(-
(EuclidDistance(repmat(amean,samSz,1),X)).^2/(2*sd(k)^2));
    ret(:,k) = thedist;
end
```

```
function ret = multiQuadRBF(X,mn,sd)
mnSz = size(mn,1);
samSz = size(X,1);
ret = zeros(samSz,mnSz);

sd = computeCentroidWidth(mn);

for k = 1:mnSz;
    amean = mn(k,:);
    thedist = (EuclidDistance(repmat(amean,samSz,1),X).^2 + sd(k)^2).^0.5;
    ret(:,k) = thedist;
end

function ret = invMultiQuadRBF(X,mn,sd)
mnSz = size(mn,1);
samSz = size(X,1);
ret = zeros(samSz,mnSz);
% Recompute SD
sd = computeCentroidWidth(mn);

for k = 1:mnSz;
    amean = mn(k,:);
    thedist = (EuclidDistance(repmat(amean,samSz,1),X).^2 + sd(k)^2).^-
0.5;
    ret(:,k) = thedist;
end

function ret = thinPlateSplineRBF(X,mn,varargin)
mnSz = size(mn,1);
samSz = size(X,1);
ret = zeros(samSz,mnSz);
for k = 1:mnSz;
    amean = mn(k,:);
    eudist = EuclidDistance(repmat(amean,samSz,1),X);
    thedist = eudist.^2.*log(eudist);
    ret(:,k) = thedist;
end

function ret = cubicRBF(X,mn,varargin)
mnSz = size(mn,1);
samSz = size(X,1);
ret = zeros(samSz,mnSz);
for k = 1:mnSz;
    amean = mn(k,:);
    eudist = EuclidDistance(repmat(amean,samSz,1),X);
    thedist = eudist.^3;
    ret(:,k) = thedist;
end

function ret = linearRBF(X,mn,varargin)
mnSz = size(mn,1);
samSz = size(X,1);
ret = zeros(samSz,mnSz);
for k = 1:mnSz;
    amean = mn(k,:);
    eudist = EuclidDistance(repmat(amean,samSz,1),X);
    thedist = eudist;
    ret(:,k) = thedist;
end
```

```
function ret = computeCentroidWidth(X)
rw = size(X,1);
dm = 0;
for k = 1:(rw-1);
    arow = X(k,:);
    n = rw-k;
    dm = max(dm, max(EuclidDistance(repmat(arow,n,1), X(k+1:end,:))));
end
ret = dm/sqrt(2*rw);
ret = repmat(ret,1,rw);

function validateBtn_Callback(hObject, eventdata, handles)
try
    tic
    UpdateInterface('off', handles);  %just wait

    valSetY = handles.valSetY;  % extracting
    valSetX = handles.valSetX;

    param = handles.param;
    yPredicted = predict(valSetX,param);
    [CofTrain,allMSE] = computeStatistics(valSetY,yPredicted);
    plotdata = makePlotData(ReverseFeatureScaling(valSetY,
handles),ReverseFeatureScaling(yPredicted, handles), handles);
    handles.plotHandles = plotHandler(plotdata,handles);
    plotRegression(valSetY,yPredicted,handles)

    tm = toc;

set(handles.summaryTable,'data',[handles.VarNames',num2cell(allMSE),num2ce
ll(CofTrain)])
    set(handles.timeResult,'string',num2str(tm))
    UpdateInterface('on', handles);
    guidata(hObject,handles);
catch  error
    disp(error.message)
    UpdateInterface('on', handles);
    errordlg(char('An error occurred while testing the network','Please
check the input parameters and try again'),'Error message','modal');
end


% --- Executes on button press in allData.
function allData_Callback(hObject, eventdata, handles)
try
    tic
    UpdateInterface('off', handles);

    allY = handles.allY;
    allX = handles.allX;

    if handles.pcaUsed == 1
        allX = allX*(handles.PC);
    end

    param = handles.param;
    yPredicted = predict(allX,param);
    [CofTrain,allMSE] = computeStatistics(allY,yPredicted);
```

```
    plotdata = makePlotData(ReverseFeatureScaling(allY, handles),
ReverseFeatureScaling(yPredicted, handles),handles);
    handles.plotHandles = plotHandler(plotdata,handles);
    plotRegression(allY, yPredicted,handles)

    tm = toc;

set(handles.summaryTable,'data',[handles.VarNames',num2cell(allMSE),num2ce
ll(CofTrain)])
    set(handles.timeResult,'string',num2str(tm))
    UpdateInterface('on', handles);
    guidata(hObject,handles);
catch  error
    disp(error.message)
    UpdateInterface('on', handles);
    errordlg(char('An error occurred while evaluating the error of all the
data','Please check the input parameters and try again'),'Error
message','modal');
end

function clustFunctions_Callback(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function plotSlider_Callback(hObject, eventdata, handles)
val = get(hObject,'value');
mx = get(hObject,'max');
intPos = handles.plotHolderInitPos;
pos = get(handles.primaryPlotHolder,'position');
pos(2) = intPos(2) - val;
set(handles.primaryPlotHolder,'position',pos)

if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function [plHdles] = plotHandler(plotdata,handles)
sz = size(plotdata,1);
plHdles = zeros(1,sz);
graphWidth = 210.4;
graphHeight = 15.6;
graphLeft = 13.4;
graphSpacing = 6;
startTop = 1.65;
lastPlotHandles = handles.plotHandles;
holderPos = handles.plotHolderInitPos;
totalHeight = max([sz*graphHeight + (sz-1)*graphSpacing +
2*startTop,holderPos(4)]);
extension = max([totalHeight - holderPos(4),0]);
if extension > get(handles.plotSlider,'min')
    set(handles.plotSlider,'max',extension,'value',extension,
'visible','on')
else
    set(handles.plotSlider,'visible','off')
end
curTop = holderPos(4) - totalHeight;
```

```
holderPos(2) = curTop;
holderPos(4) = totalHeight;
set(handles.primaryPlotHolder,'position',holderPos);
if ~isempty(lastPlotHandles)
    delete(lastPlotHandles);
end

newTop = startTop;
for k = 1:sz;
    pos = [graphLeft,holderPos(4)-newTop-
graphHeight,graphWidth,graphHeight];

    newTop = newTop + graphHeight + graphSpacing;
    x = plotdata{k,1};
    y = plotdata{k,2};
    styl = plotdata{k,3};
    xlab = plotdata{k,4};
    ylab = plotdata{k,5};
    plHdles(k) =
axes('parent',handles.primaryPlotHolder,'units',get(handles.primaryPlotHol
der,'units'),'position',pos);
    axes(plHdles(k));
    noInnerPlot = size(x,2);
    if noInnerPlot>1
        legendStr = plotdata{k,6};
        allPlots = zeros(1,noInnerPlot);
        hold on
        for r = 1:noInnerPlot;
            xx = x(:,r);
            yy = y(:,r);
            allPlots(r) = plot(xx,yy,styl{r});
        end
        legend(allPlots,legendStr)
        hold off
    else
        plot(x,y,styl)
    end
    ylabel(ylab)
    xlabel(xlab)
end

function Fx = getFcn(objfcn,Xs)
    n = size(Xs,1);
    Fx = ones(n,1);
    for k = 1:n;
        Fx(k) = objfcn(Xs(k,:));
    end




function applyPCARadBtn_Callback(hObject, eventdata, handles)
if get(hObject,'value') == 1
    set(handles.pcaDimLabel,'visible','on');
    set(handles.pcaDimTextBox,'visible','on');
else
    set(handles.pcaDimLabel,'visible','off');
    set(handles.pcaDimTextBox,'visible','off');
end
guidata(hObject,handles);
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function setOptimOptBtn_Callback(hObject, eventdata, handles)
handles.optfcnsData = SetOptionHandler(handles.optMenu,
handles.optfcnsData, handles.optfcnsHandle, handles);
guidata(hObject, handles);

function setCGSOOptBtn_Callback(hObject, eventdata, handles)
handles.clusterData = SetOptionHandler(handles.clustFunctions,
handles.clusterData, handles.clusterHandles, handles);
guidata(hObject, handles);

function data = SetOptionHandler(popMenuHandle, data, objHs, handles)
optData = data;
optH = objHs;
curOptObj = optData.(optH{get(popMenuHandle,'value')});
options = curOptObj.options;
setfields = curOptObj.setfield;
sz = length(setfields);
defaultValues{sz} = '';
variableType = zeros(1,sz);
for k = 1:sz;
    op = options.(setfields{k});
    if (isnumeric(op))
        op = num2str(op);
        variableType(k) = 1;
    elseif ~ischar(op)
        op = func2str(op);
        variableType(k) = 2;
    else
        variableType(k) = 3;
    end
    defaultValues{k} = op;
end

newOptions = inputdlg(setfields,'Set Options',1,defaultValues);
if isempty(newOptions)
    return
end

for k = 1:sz;
    op = newOptions{k};
    typ = variableType(k);
    if typ == 1
        options.(setfields{k}) = str2num(op);
    elseif typ == 2
        options.(setfields{k}) = str2func(op);
    else
        options.(setfields{k}) = op;
    end
end
curOptObj.options = options;
optData.(optH{get(popMenuHandle,'value')}) = curOptObj;

data = optData;
```

```
function RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden,
lambda, rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles,
variableNumber, variableSize, xdata, varargin)

try
     try
        matlabpool open
     catch
        matlabpool close
        matlabpool open
     end

     %wbH = waitbar(0,'0% Done');

     xlab = '';
     ylab = 'MSE';
     trainMSE = zeros(1,variableSize);
     valMSE = zeros(1,variableSize);
     testMSE = zeros(1,variableSize);
     trainR = zeros(1,variableSize);
     valR = zeros(1,variableSize);
     testR = zeros(1,variableSize);
     counter = 0;
     if variableNumber == 1
         xlab = 'Data Partition';
         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition{k}, timeAgo,
pcaDim, numHidden, lambda, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
         end
     elseif variableNumber == 2
         xlab = 'Time Lag';
         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo{k},
pcaDim, numHidden, lambda, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
         end
     elseif variableNumber == 3
         xlab = 'PCA Dimension';
         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo,
pcaDim{k}, numHidden, lambda, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
```

```
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
         end
     elseif variableNumber == 4
         xlab = 'Number of hidden nodes';
         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden{k}, lambda, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
      elseif variableNumber == 5
         xlab = 'Regularisation Term';
         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda{k}, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
      elseif variableNumber == 6
         xlab = 'Radial Basis function';

         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda, rbf{k}, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
      elseif variableNumber == 7
         xlab = 'Optimization function';

         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda, rbf, optimFcn{k}, clusterFcn, optimOptions{k},
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
```

```
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
     elseif variableNumber == 8
         xlab = 'Clustering Function';

         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda, rbf, optimFcn, clusterFcn{k}, optimOptions,
clusterOptions{k}, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
     elseif variableNumber == 9
         xlab = varargin{1};

         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda, rbf, optimFcn, clusterFcn, optimOptions{k},
clusterOptions, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
     elseif variableNumber == 10
         xlab = varargin{1};

         parfor k = 1:variableSize;
            [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions{k}, handles);
            trainMSE(k) = theMSE(1);
            valMSE(k) = theMSE(2);
            testMSE(k) = theMSE(3);
            trainR(k) = theR(1);
            valR(k) = theR(2);
            testR(k) = theR(3);
            %waitbar(k/variableSize,sprintf('%12.9f',
100*k/variableSize));
         end
     end
     figure;
     if iscell(xdata)
         xrange = 1:length(xdata);
         pl = bar(xrange,[trainMSE(:),valMSE(:),testMSE(:)]);
         set(get(pl(1),'parent'),'xticklabel',xdata)
         xlabel(xlab)
         ylabel('MSE');
```

```
        figure;
        plR = bar(xrange,[trainR(:),valR(:),testR(:)]);
        set(get(plR(1),'parent'),'xticklabel',xdata)
        xlabel(xlab)
        ylabel('R')

    else
        pl1 = plot(xdata, trainMSE, 'r', 'linewidth', 2);
        hold on
        pl2 = plot(xdata, valMSE, 'b', 'linewidth', 2);
        pl3 = plot(xdata, testMSE, 'm', 'linewidth', 2);
        pl = [pl1, pl2, pl3];
        xlabel(xlab)
        ylabel('MSE')
        hold off
        figure;
        pl1R = plot(xdata, trainR, 'r', 'linewidth', 2);
        hold on
        pl2R = plot(xdata, valR, 'b', 'linewidth', 2);
        pl3R = plot(xdata, testR, 'm', 'linewidth', 2);
        plR = [pl1R, pl2R, pl3R];
        xlabel(xlab)
        ylabel('R')
        hold off
    end
    legend(pl, char('Training','Validation','Test'))
    legend(plR, char('Training','Validation','Test'))

    %delete(wbH)
    matlabpool close
catch e
    disp(e.message)
    %delete(wbH)
    matlabpool close
end


function [theMSE, theR] = DiagnoseOnce(dataPartition, timeAgo, pcaDim,
numHidden, lambda, rbf, optimFcn, clusterFcn, optimOptions,
clusterOptions, handles)

    trainR = dataPartition(1);
    valR = dataPartition(2);
    testR = 1 - trainR - valR;
    [allY,allX] = makeFullDataSet(handles.data, timeAgo);
    tempAllX = allX;

    if pcaDim > 1
        [allX,PC] = pca(allX,pcaDim);
        handles.PC = PC(:,1:pcaDim);
    end

    actualSamSize = size(allX,1);
    noTrain = floor(trainR*actualSamSize);
    noTest = floor(testR*actualSamSize);
    noVal = actualSamSize - noTrain - noTest;

    trainSetY = allY(1:noTrain,:);
    trainSetX = allX(1:noTrain,:);
    valSetY = allY(noTrain+(1:noVal),:);
```

```
    valSetX = allX(noTrain+(1:noVal),:);
    testSetY = allY(noTrain+noVal+(1:noTest),:);
    testSetX = allX(noTrain+noVal+(1:noTest),:);

    handles.maintainX = tempAllX(1:noTrain,:);

    param = RBFTrainingAlgorithm(trainSetX, trainSetY, numHidden,lambda,
rbf, optimFcn, optimOptions, clusterFcn, clusterOptions, handles, false);
% This clusters and trains the weight. It returns the trained weights
among other parameters

    if handles.maintainDim
        trainSetX = tempAllX(1:noTrain,:);           % Here is training
set input space
        valSetX = tempAllX(noTrain+(1:noVal),:);     % Here is validation
set input space
        testSetX = tempAllX(noTrain+noVal+(1:noTest),:);    % Here is test
set output space
    end

    % Training set accuracy
    yPredicted = predict(trainSetX,param);  % The Y predicted
    [trainCof,trainMSE] = computeStatistics(trainSetY,yPredicted);  %
Computing MSE and R value

    % Validation set accuracy
    yPredicted = predict(valSetX,param);  % The Y predicted
    [valCof,valMSE] = computeStatistics(valSetY,yPredicted);  % Computing
MSE and R value

    % Test set accuracy
    yPredicted = predict(testSetX,param);  % The Y predicted
    [testCof, testMSE] = computeStatistics(testSetY,yPredicted);  %
Computing MSE and R value

    theR = [trainCof, valCof, testCof];
    theMSE = [trainMSE, valMSE, testMSE];


function OptimMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);

xdata =  handles.optfcnsString;
sz = length(xdata);
optimFcn = {};
optimOptions = {};

for k = 1:sz;
    [optim, options] = GetOptimFcn(k, handles);
    optimFcn{k} = optim;
    optimOptions{k} = options;
end

RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 7, sz,
xdata);
```

```
function ClusterMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);

xdata =  handles.clusterNames;
sz = length(xdata);
clusterFcn = {};
clusterOptions = {};

for k = 1:sz;
    [clusterF, options] = GetClusterFcn(1, handles);
    clusterFcn{k} = clusterF;
    clusterOptions{k} = options;
end

RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 8, sz,
xdata);



function PCAMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);
inputdata = inputdlg('Provide range of PCA dimension');
if isempty(inputdata)
    return
end
xdata =  str2num(inputdata{1});
if isempty(xdata)
    return
end
pcaDim = num2cell(xdata);
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 3,
length(xdata), xdata);

function RBFunctionMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);

xdata =  handles.rbfNames;
sz = length(xdata);
rbf = {};

for k = 1:sz;
    rbf{k} = handles.rbfcns{k};
end
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 6, sz,
xdata);

function OptimOptionsMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);

[optimIndex, ok] = listdlg('ListString',
handles.optfcnsString,'SelectionMode',...
```

```
    'single','Name', 'Optimisation', 'PromptString', 'Select an
optimisation algorithm');
if ok == 0
    return
end
[optimFcn, options] = GetOptimFcn(optimIndex, handles);
optFields = fieldnames(options);
[optionIndex, ok] = listdlg('ListString', optFields,'SelectionMode',...
    'single','Name', 'Optimisation Options', 'PromptString', 'Select an
option');
if ok == 0
    return
end
optionName = optFields{optionIndex};
inputdata = inputdlg(['Provide range of for ', optionName]);
if isempty(inputdata)
    return
end
xdata =  str2num(inputdata{1});
sz = length(xdata);
if isempty(xdata)
    return
end
optimOptions = {};

for k = 1:sz;
    options.(optionName) = xdata(k);
    optimOptions{k} = options;
end
xlab = [optionName,' for ', handles.optfcnsString{optimIndex}];
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 9, sz,
xdata, xlab);


function ClusteringMenuBar_Callback(hObject, eventdata, handles)
[dataPartition, timeAgo, pcaDim, numHidden, lambda, rbf, optimFcn,
clusterFcn, optimOptions, clusterOptions] = DefaultVar(handles);

[clusterIndex, ok] = listdlg('ListString',
handles.clusterNames,'SelectionMode',...
    'single','Name', 'Clustering Algorithm', 'PromptString', 'Select a
clustering algorithm');
if ok == 0
    return
end
[clusterFcn, options] = GetClusterFcn(clusterIndex, handles);
optFields = fieldnames(options);
[optionIndex, ok] = listdlg('ListString', optFields,'SelectionMode',...
    'single','Name', 'Clustering Options', 'PromptString', 'Select an
option');
if ok == 0
    return
end
optionName = optFields{optionIndex};
inputdata = inputdlg(['Provide range of for ', optionName]);
if isempty(inputdata)
    return
end
xdata =  str2num(inputdata{1});
```

```
sz = length(xdata);
if isempty(xdata)
    return
end
clusterOptions = {};

for k = 1:sz;
    options.(optionName) = xdata(k);
    clusterOptions{k} = options;
end
xlab = [optionName,' for ', handles.clusterNames{clusterIndex}];
RunDiagnoseInParallel(dataPartition, timeAgo, pcaDim, numHidden, lambda,
rbf, optimFcn, clusterFcn, optimOptions, clusterOptions, handles, 10, sz,
xdata, xlab);

function DefaultMenuBar_Callback(hObject, eventdata, handles)
defaultVar = handles.defaultVar;
theFields = fieldnames(defaultVar);
sz = length(theFields);
lastAns = {};
for k = 1:sz;
    op = theFields{k};
    lastAns{k} = num2str(defaultVar.(op));
end
newDefault = inputdlg(theFields,'Set Default Values',1, lastAns);
if isempty(newDefault)
    return
end

for k = 1:sz;
    op = newDefault{k};
    defaultVar.(theFields{k}) = str2num(op);
end

handles.defaultVar = defaultVar;
guidata(hObject, handles);

function MNMenu_Callback(hObject, eventdata, handles)
st = get(hObject, 'checked');
menuH = [handles.MNMenu, handles.RescalingMenu, handles.MinMaxMenu];
set(menuH, 'checked', 'off');
if (strcmpi(st,'on'))
    set(hObject, 'checked','off');
else
    set(hObject, 'checked','on');
end
handles.fsMethod = 1;
guidata(hObject,handles);

function RescalingMenu_Callback(hObject, eventdata, handles)
st = get(hObject, 'checked');
menuH = [handles.MNMenu, handles.RescalingMenu, handles.MinMaxMenu];
set(menuH, 'checked', 'off');
if (strcmpi(st,'on'))
    set(hObject, 'checked','off');
else
    set(hObject, 'checked','on');
end
handles.fsMethod = 2;
guidata(hObject,handles);
```

```
function MinMaxMenu_Callback(hObject, eventdata, handles)
st = get(hObject, 'checked');
menuH = [handles.MNMenu, handles.RescalingMenu, handles.MinMaxMenu];
set(menuH, 'checked', 'off');
if (strcmpi(st,'on'))
    set(hObject, 'checked','off');
else
    set(hObject, 'checked','on');
end
handles.fsMethod = 3;
guidata(hObject,handles);

function result = ReverseFeatureScaling(rescaledValues, handles)
rw  = size(rescaledValues,1);
if handles.fsMethod == 1
    result = rescaledValues.*repmat(handles.dataSDs, rw,1) +
repmat(handles.dataMeans, rw,1);
elseif handles.fsMethod == 2
    result = rescaledValues.*repmat(handles.dataMaxs, rw,1);
else
    theMin = repmat(handles.dataMins, rw,1);
    theMax = repmat(handles.dataMaxs, rw,1);
    result = rescaledValues.*(theMax - theMin) + theMin;
end

function hdles = ShuffleData(handles)

allY = handles.allY;
allX = handles.allX;
noSam = size(allY,1);
rangePos = randperm(noSam);
allX = allX(rangePos,:);
allY = allY(rangePos,:);
handles.allX = allX;
handles.allY = allY;

hdles = handles;


function mDimRad_Callback(hObject, eventdata, handles)
handles.maintainDim = get(hObject,'value');

guidata(hObject, handles);




function runEditBox_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

APPENDIX B

Publications from this Study

Isimeto, R. Fasina, E.P. Alienyi, C.D. and Uwadia, C.O. (2015) Predicting the Occurrence of Rainfall using Improved Radial Basis Function Neural Network, *Journal of the Computer Science and its Applications: An International Journal of the Nigerian Computer Society,* 22(2), 66-72

Isimeto, R., Yinka-Banjo, C., Alienyi, C.D., and Uwadia, C.O. (2017)An Enhanced Clustering Analysis Based on Glowworm Swarm Optimization, *Proceedings of the 4th International conference on Soft Computing and Machine Intelligence* (ISCMI2017), Mauritius.