

EXCHANGE RATE FORECASTING

A Thesis

submitted to

Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Shipra Kumar



Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2017

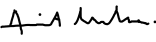
Supervisor: Amit Mitra

© Shipra Kumar 2017

All rights reserved

Certificate

This is to certify that this dissertation entitled EXCHANGE RATE FORECASTING towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Shipra Kumar at Indian Institute of Technology, Kanpur under the supervision of Amit Mitra, Professor, Department of Mathematics and Statistics , during the academic year 2016-2017.



Amit Mitra

Committee:
Amit Mitra
Uttara Naik-Nimbalkar

Dedicated to Dr. A. Raghuram,
Dept of Mathematics, IISER, Pune
and to my parents

Declaration

I hereby declare that the matter embodied in the report entitled EXCHANGE RATE FORECASTING are the results of the work carried out by me at the Department of Mathematics and Statistics , Indian Institute of Technology, Kanpur under the supervision of Amit Mitra and the same has not been submitted elsewhere for any other degree.


Shipra Kumar

Acknowledgments

This thesis would not have been possible without the guidance of Dr. Amit Mitra, Professor, IIT, Kanpur. It was his constant motivation and guidance that helped me during the whole process of diving into a completely new field. I am also thankful to Dr. Uttara Naik-Nimbalkar, who has always been there for me in an academic capacity ever since I have known her. Her constant words of motivation was what kept me going. Dr. A Raghuram, who has been more than a professor for me. A mentor, a friend, Dr. Raghuram has been a source of motivation on various levels. It was him who introduced me to mathematics, and for that I would always be grateful to him. Among others, I would like to extend heartfelt gratitude to all my professors at IISER, Pune, especially Dr. Chandrasheel Bhagawat and Dr. Kaneenika Sinha who not only were my academic mentors but also helped me grow personally.

This thesis would be incomplete if I do not mention my friends who have been an inexplicable part of these 5 years. First, I would like to thank my friends at IIT, Kanpur. They made IIT, Kanpur a home away from home for me. Shubham Karnawat, Anupreet Porwal, Sandeep Kumar, Kanupriya Agarwal, Shanu Vashishtha, Gaurav Doshi, Nilay Jain, Shailendra Singh, I thank you for the constant support and friendship. Deeksha Adil, Lipi Jain, Anirban Sharma, Prachi Atmasiddha, Subhadra Mokashe, Darshini Ravishankar, Prashali Chauhan, Kunal Mozumdar, Chris John, Papia Bera, Varun Prasad, Debarun Ghosh, Ajith Nair, Visakh Narayan, Dileep, you guys made me the person I am today. Thank you for all the support during the "Math years" and before.

Last but not the least, I would like to thank my mom, my dad and my brother for constantly being there.

Abstract

Forecasting of exchange rates between currencies is of utmost importance in the financial world because its implication on imports-exports, trading and world economy in general. Since exchange rates data is a sequential data, modelling was traditionally done by time series analysis. However, forecasting using even non-linear time series models like the ARCH and the GARCH model did not give better forecasts in comparison to the simplest of the models, the random walk or the AR(1) model. However, with a growing interest in artificial neural networks modelling since 1980s, they were also increasingly used for time series modelling. The neural network models gave a better forecast than the time series models used. Attempts were then made to combine the two modelling techniques to see if a hybrid model would perform better than either of the two models. With the advent of newer optimisation techniques like genetic algorithms in machine learning, these were incorporated as well to build new models. There was also an attempt to see how newer mathematical constructs like the fuzzy logic could be used for building an artificial neural networks (Jang, 1993). By 2006, Hinton had proposed a new probabilistic model for data modelling which was called the deep belief network. Time series modelling done using these models gave better forecasts in comparison to even the fixed geometry neural network models. In this thesis, I have attempted to study the theoretical basis behind such model and combine the forecasts of all the models used using information theoretic averaging. This is done to study whether an average of the forecasts gives a better forecast value than the individual models.

Contents

Abstract	xi
1 Literature Review	1
1.1 Introduction	1
1.2 Existing Models- A Review	2
1.3 Literature Review	2
1.4 Problem Statement	4
2 Machine Learning Models	5
2.1 Neural Networks	5
2.2 Genetic Algorithms and Neuro-Genetic Model	21
2.3 Neuro- Fuzzy Systems	33
2.4 Comparison with Random Walk Model	43
3 Combination Of Forecasts	45
3.1 Information Theoretic Averaging	45
3.2 Averaging Machine Learning Models	48

Chapter 1

Literature Review

1.1 Introduction

For any economy, exchange rate within countries is defined as the price of one unit of the currency of one country in terms of the currency of the other country. Exchange rate is a factor that directly impacts international trade, that is with respect to the exports and imports, day-to-day businesses and finance between two countries. Therefore, exchange rates are a key factor which feature in the international economy as a whole. Forecasting currency exchange rates then becomes an important problem in International Economics and Finance. This is a challenging problem in economics as mathematically modelling and forecasting the exchange rates becomes difficult due to uncertainty in international trade and capital flows. Classically, an exchange rate forecasting is considered to be a problem in time series analysis. However, there have also been numerous studies in recent times which combine classical mathematical models of time series with computational tools of machine learning. This is done to generate new models that improve the accuracy of forecasting these rates.

Exchange rate forecasting has always been a challenging area of research for applied statisticians and econometricians. Apart from working with large data sets that constitute exchange rates modelling, it may also happen that forecasting has to be done sometimes with incomplete data sets. This is a possible situation where a time series analysis may fall short and modern algorithms in machine learning may be required for modifications in modelling. Hybrid methodology of combining various models is another computational technique that seeks to combine properties of various models to improve the forecasting power. We start with a review of the existing literature.

1.2 Existing Models- A Review

Numerous studies show us that models based on time series models fail to deliver superior forecasts to the simplest of all models, the simple random walk or AR(1) model. Exchange rates exhibit significant non-linearities. Many studies on sequential data modelling have shown that these non-linearities, though also modelled by time-series models like the GARCH model, are better captured by Artificial Neural Network models. Artificial Neural Networks are data-driven and self-adaptive which make them highly desirable. This provides a major advantage when compared to the traditional approaches of modelling, which tend to become difficult to model with increasing complexity of the model. It has been shown that a neural network can approximate any continuous function to desired accuracy (Hornik,1991). Neural networks also provide us with the advantage that we need not specify the relationship between output and input beforehand. It has been shown through numerous models that ANN models are significantly better than existing statistical models in terms of forecasting ability.

Other than Neural Networks, there exist machine learning techniques like the Adaptive Neuro Fuzzy Inference System (ANFIS), Deep Learning and evolutionary methods like Genetic Algorithms which are used for sequential data forecasting. Apart from using these models in seclusion, the newer trend in time series forecasting is the use of hybrid models. Hybrid models are models which combine time series modelling and machine learning techniques mentioned to come up with better forecasts. Real world data is rarely purely linear or non-linear, thus a hybrid models helps capture complexities of the model better than a simple time series model or ANN model. Some of them are discussed in the next section.

1.3 Literature Review

Zhang(2001) [1]showed that a hybrid methodology that combines both ARIMA and ANN models indicate that the combined model can be an effective way to improve forecasting accuracy achieved by either of the models used separately. Autoregressive integrated moving average (ARIMA) is a linear model that is used extensively in time series forecasting. It is because they can represent several different types of time series, i.e autoregressive processes(AR), moving average processes (MA) and combined AR and MA (ARMA) processes. However, they are restricted by the assumed linearity of the model. Thus, the non-linearities that are present in the exchange rate data model cannot be captured. However, Artificial Neural Networks are proficient in capturing the underlying function of any given data set and hence can capture the non-linearity of a model satisfactorily. Additionally, no prior model is

required for further modelling as features are extracted and modelled from data itself. Since both models have been extensively used for forecasting, ARIMA models and ANNs have often been compared in terms of the superiority in forecasting performance. In this paper, for British Pound/USD exchange rate, a hybrid methodology that combines both ARIMA and ANN models is proposed. This is done by first deriving an ARIMA model from the data and then, modelling the residuals obtained by Neural Networks. Experimental results indicated that the combined model achieved a better forecasting rate than either of the models used separately.

Nag and Mitra (2002) [2] showed that use of a hybrid artificial intelligence method, based on neural network and genetic algorithm indicated superior performance compared to the traditional non-linear time series techniques and also fixed-geometry neural network models. Training a neural network based on standard training algorithms like the backpropagation algorithm tends to suffer from serious drawbacks. These include, but are not limited to, convergence to a local minima instead of a global minima and the search for optimal network architecture through manual experimentation and finetuning. This paper thus puts forward an improved model where parameters for network architecture are optimised using genetic algorithms rather than manual experimentation is used for exchange rate forecasting for the deutsche mark/US dollar, the Japanese yen/US dollar and the US dollar/British pound rates. Genetic algorithm is a search procedure which tries to find the optimal set of parameters in the search space for getting a optimal and improved neural network. The forecasts using the genetically optimised neural network is seen to give a better forecast in comparison to the the feedforward neural network.

Chao, Shen and Zhao (2011)[3], used deep belief network (DBN) to predict both British Pound/US dollar and Indian rupee/US dollar exchange rates and compare it to forecasting results obtained using feedforward neural network. Deep Belief Network is a generative neural network model and consists of many hidden layers. A DBN is composed of many units of a probabilistic model called a Restricted Boltzmann Machine or RBM. An RBM is a recurrent two layer neural network which assigns a probabilistic model to the incoming data using symmetrically weighted connections between the two layers. The first layer in an RBM corresponds to inputs (visible units v) and the second layer to the hidden units. This paper showed that a DBN model for forecasting exchange rates of British Pound/US dollar (GBP/USD) and Indian rupee/US dollar (INR/USD) achieved results were superior to that of forecasting using a feedforward neural network.

Khashei, Hejazi, Bijari (2006)[4] showed how ANNs and fuzzy regression are combined to over-

come the limitations posed by using incomplete data sets. Artificial Neural Networks usually require a large amount of data to make exchange rate predictions. Fuzzy forecasting methods ,on the other hand, are capable of making predictions even with incomplete data sets. However, it has been seen experimentally that their performance is not always satisfactory. Thus, this paper proposed a model which sought to combine ANN and fuzzy regression methods to get improved forecasting in cases where data sets were incomplete.

There exist other papers like Atsalakis and Valavanis(2009)[5] which elucidate how forecasting with hybrid models like ANFIS give us improved forecasts. Thus, different models have been experimented with to show us how different techniques in machine learning are used for forecasting exchange rates.

1.4 Problem Statement

The basic idea of the model combination in forecasting is to use each models unique feature to capture different patterns in the data. Theoretical and empirical findings suggest that combining different methods can be an effective and efficient way to improve forecasts. Thus, this project aims to look at methodology that is used to combine classical mathematical models of time series with modern tools, mainly **computational statistical and machine learning techniques** in detail. The next part of the project would be to look at different techniques of generating hybrid models and looking at a combination of forecasts obtained by using concepts in Bayesian statistics and Information theory. This would be done to see if a better forecast can be obtained by a combination of forecasts of different models. Thus, we begin by looking at what these machine learning techniques are and explain the basics of the techniques used in the models.

Chapter 2

Machine Learning Models

One of the most basic machine learning tools that are used in studying a time series data and their forecasting is neural networks. Artificial Neural Networks are extremely efficient in approximating non-linear functions to any desired accuracy. Papers by Refenes (1996), Weigend et al. (1992), Hann and Steurer (1996) observe that ANN models perform much better than linear time series models and even random walk for some exchange rate data. We thus begin by studying neural network in detail.[6, 7]

2.1 Neural Networks

A neural network is a machine learning tool which is used for processing large amount of data. A computational system which tries to mimic the human brain for processing information, it is a massive processor whose components are parallelly distributed. These components form what is called a "layer" in a neural network and each layer is composed of multiple "neurons" or nodes. It is at these nodes that information is processed and passed onto a neuron of the next layer using weighted connections. Thus, the three essential features of an artificial neural network (ANN) are the basic processing units known as neurons; the network architecture which describes how neurons in each layer are connected to each other and also the number of hidden layers between inputs and the outputs; and the training algorithm for the neural network that helps us find the optimal values of the parameters of the network architecture that is used performing a particular task.

Definition 1. *Neural Network* *A neural network is an interconnected group of neurons in a massively parallel distributed processor. It has a layer of nodes of inputs followed by a layer or multiple layers of hidden nodes and finally an output layer. It is an important*

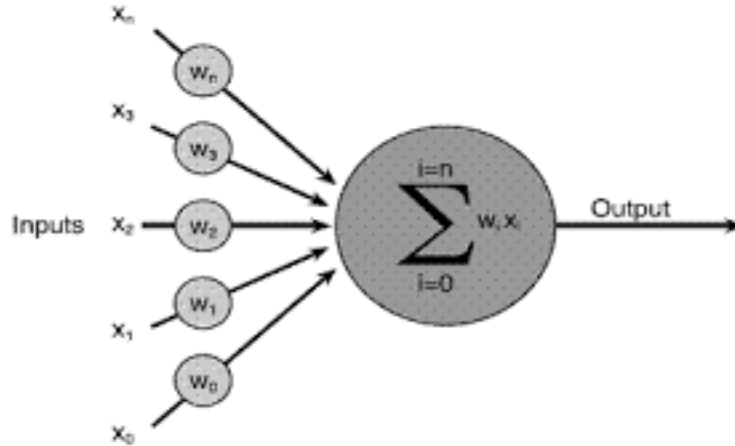


Figure 2.1: Structure of a Neuron

computational tool in machine learning and data sciences, used in tasks like approximating an unknown function, pattern recognition, signal processing and sequential data analysis.

Definition 2. Neurons *An artificial neuron ,the mathematical equivalent of biological neurons in the brain, are the basic units of an artificial neural network. An artificial neuron receives inputs from neurons of the previous layer and performs a weighted sum of the inputs it receives.*

Definition 3. Activation Function *An activation function is typically a monotonically increasing, differentiable function which is used to define the output of a neuron. An activation function takes the weighted sum of the inputs produced at a neuron and passes the resultant through the function to get an output of the neuron. The most commonly used activation functions include sigmoid function, hyperbolic tan function and the identity function*

Neural Networks have been increasingly used in statistical analysis. They provide network representation for statistical constructs like regression, methods of density estimation (parametric and non parametric). Many problems of modeling have both statistical and neural network inputs. Some neural networks have probabilistic elements (like Boltzmann machines) and there is an increasing effort to embed neural networks in general statistical framework. The subsequent sections will talk about the neural network architecture, the training algorithm and the results obtained.

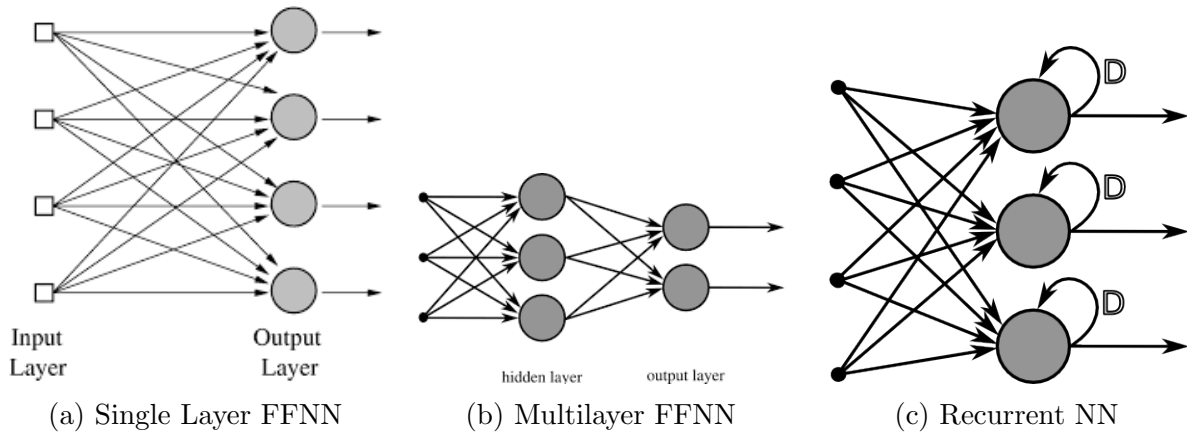


Figure 2.2: Types of Neural Networks

2.1.1 Neural Network Architecture

Neural networks can be single layer feedforward neural network, multilayer feedforward neural network or recurrent network.

- **Single Layer Feedforward Neural Network** The simplest feedforward neural network wherein connections between the units do not form a cycle. It consists of two layers a layer of input nodes which are fed directly to layer of output node via a series of weights.
- **MultiLayer Feedforward Neural Network** A feedforward neural network which has the presence of one or more hidden layers in addition to input and output layers.
- **Recurrent Neural Network** This type of network is where the output of a neuron affects part of the input given to the neuron because it consists of at least one feedback loop at that neuron.

2.1.2 Artificial Neuron : Structure and Function

As described at the beginning of the section, a neuron is the basic building block of any neural network. A neuron in itself is a computational unit which has three components.

- **Input weights and bias** A neuron has incoming inputs from the previous layer. Each of these inputs is assigned a weight at the neuron, which signifies the strength of each input. More specifically, an input x_j is one layer is connected to the k-th neuron in

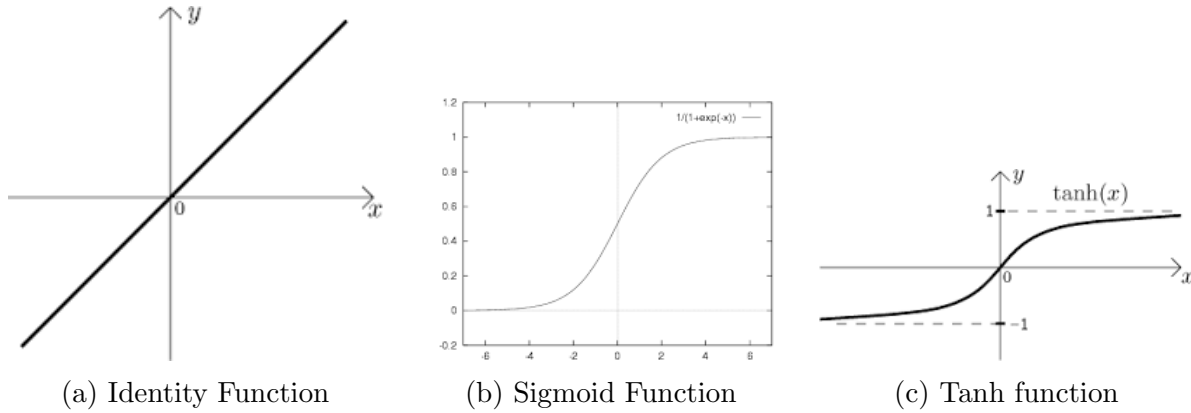


Figure 2.3: Types of Activation Functions

the next layer by w_{kj} . In addition to the inputs, an extra term $+1$ is sometimes added to the input layer which is multiplied by w_{k0} (i.e 0th input for the k -th neuron) and is known as the bias term (Figure 2.1).

- **Adder** Let x_i s represent the input which are multiplied by suitable weights w_i . Σ is the function that adds them up such that $v_i = \Sigma w_i x_i$. This Σ is known as the adder in the neuron and the resultant v_i is fed to the activation function which comes next to get the output.
- **Activation Function** These are monotonically increasing, differentiable functions which acts on the weighted sum of inputs v_i to get an output. Thus output $y_i = \phi(v_i)$, where ϕ is the activation function and v_i is the weighted sum as described above. These function include
 - Identity function: $\phi(v) = v, v \in (-\infty, \infty)$
 - Sigmoid function: $\phi(v) = \frac{1}{1+e^{-v}}, v \in (-\infty, \infty), \phi(v) \in [0, 1]$
 - Tanh function: $\phi(v) = \frac{e^{2v}-1}{e^{2v}+1}, v \in (-\infty, \infty), \phi(v) \in [-1, 1]$

2.1.3 Training Neural Network

Training a neural network includes optimising a cost function and updating the weights and biases as a result of that training. The cost function $\xi(e_i)$ is a function of errors $\mathbf{e}_i = \mathbf{d}_i - \mathbf{y}_i$ where d_i is the desired output and y_i is the output as defined above. Cost function are typically convex function, which makes them easier to optimise. Examples of cost functions are:

- $(e_i)^2$
- $|e_i|$

Since $y_i = \phi(v_i)$, where $v_i = \sum w_i x_i$ is a function of weights $w = [w_1, w_2, \dots, w_i, \dots, w_k]$, therefore the error and the cost function also become a function of w . Thus, the cost function can also be written as $\xi(w)$. In a single layer feedforward network, the most widely used algorithm for optimising the cost function is the Gradient Descent Algorithm. Gradient descent algorithm

Algorithm 1: Gradient Descent Algorithm

- 1 Initialise $w = w^{(0)}$.
- 2 Update w by moving along the gradient of the cost function $\xi(w)$

$$w(n+1) = w(n) - \eta \left. \frac{\partial \xi(w)}{\partial w} \right|_{w=w(n)}$$

where $\eta > 0$ is the learning rate.

- 3 Repeat until convergence.
-

is a method for updating weights w of the network by trying to minimise the cost function $\xi(w)$ at each point. In gradient descent, we take steps in a direction where the cost function is minimised and it is proportional to the negative of the gradient of the function at the current point. The proportionality constant is determined by η which determines how large or small the steps are to be taken. The value of η is important to the convergence of the algorithms. If η is very small, the algorithm will take a large time to converge to the optimal value of w . If η is large, the algorithms may become unstable and never reach the true optimal value as it will oscillate to large values of the cost function and might even diverge for critical values.

Now, $\Delta w(n) = w(n+1) - w(n) = -\eta \frac{\partial \xi(w)}{\partial w}$ Using Taylor's Expansion,

$$\xi(w(n+1)) \approx \xi(w(n)) - \eta \left\| \frac{\partial \xi(w)}{\partial w} \right\|^2$$

This shows that at every iteration, the cost decreases.

In a neural network of multiple hidden layers, the algorithm used for weight updates is the **Backpropagation Algorithm**. It consists of a forward pass and a backward pass. In the forward pass, input is applied and with fixed weights over the whole network, we get

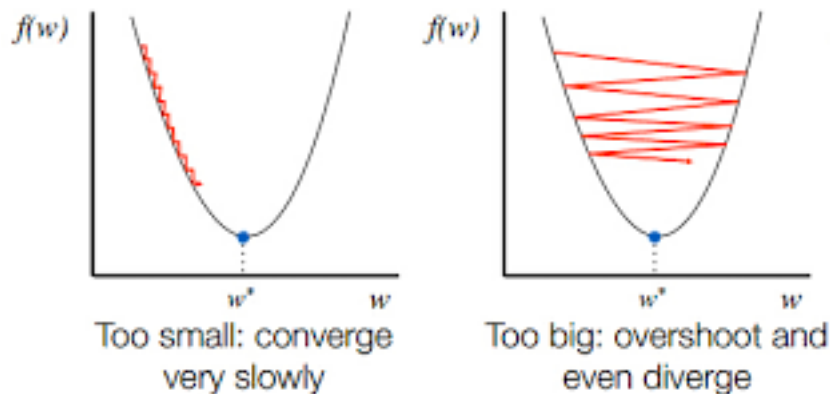


Figure 2.4: Learning trajectory for different rates of η

an output by propagating the effect of input layer by layer. In the backward pass, the error obtained because of the difference between the output of the network and the target value is propagated in a backward pass over the network which results in the weights of the network getting adjusted so as to minimise the error. The cost function used in the algorithm is invariably a convex function, which makes them easy to differentiate and derive the required quantities. We assume that the cost function used is $\xi(w(n)) = \frac{1}{2} \sum e_j^2(n)$ and hence, derive the algorithm.

While backpropagation is a steepest descent algorithm, the Marquardt-Levenberg algorithm is an approximation to Newton's method. It is also a modification to the Gauss-Newton method of optimization. [8]

We first define the functions for the gradient and Hessian of the cost function $\xi(w)$. The gradient is denoted by the function, g and it is calculated as follows for weights w_1, w_2, \dots, w_n .

$$g = \nabla \xi(w) = \left[\frac{\partial \xi}{\partial w_1}, \frac{\partial \xi}{\partial w_2}, \dots, \frac{\partial \xi}{\partial w_n} \right]^T$$

The Hessian H of a function is defined as:

$$H = \nabla^2 \xi(w) = \begin{pmatrix} \frac{\partial^2 \xi}{\partial w_1^2} & \frac{\partial^2 \xi}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 \xi}{\partial w_1 \partial w_n} \\ \frac{\partial^2 \xi}{\partial w_2 \partial w_1} & \frac{\partial^2 \xi}{\partial w_2^2} & \dots & \frac{\partial^2 \xi}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \xi}{\partial w_n \partial w_1} & \frac{\partial^2 \xi}{\partial w_n \partial w_2} & \dots & \frac{\partial^2 \xi}{\partial w_n^2} \end{pmatrix}$$

Algorithm 2: The Backpropagation Algorithm

- 1 The basic equations at j-th neuron in the outer layer

$$v_j(n) = \sum_{i=0}^m w_{ij}(n)x_i(n)$$

$$y_j(n) = \phi(v_j(n)) (\phi \text{ is the activation function})$$

$$e_j(n) = d_j(n) - y_j(n)$$

$$\xi(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \quad (C \text{ is output layer neurons})$$

- 2 The derivations and calculation of delta δ

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)y_i(n)\phi'(v_j(n))$$

Define

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

. This $\delta_j(n)$ is the local gradient for any neuron j.

- 3 **Weight update** for the connection between j-th neuron in the outer layer and i-th neuron in the layer before the outer layer (If the neural network has k layers and k-th layer is the outer layer, then the i-th neuron is in k-1-th layer)

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

.

$$\Delta w_{ji}(n) = \eta \delta_j(n)y_i(n). \quad (\eta \text{ is learning rate})$$

- 4 In case the j-th neuron is a neuron in one of the hidden layers,

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \xi(n)}{\partial v_j(n)} = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial \xi(n)}{\partial y_j(n)} \phi'(v_j(n)) \end{aligned}$$

Since, $\xi(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$ (k is the output node in set C of neurons in output layer)

$$\begin{aligned} \frac{\partial \xi(n)}{\partial y_j(n)} &= \sum_k e_k \frac{\partial \xi(n)}{\partial y_k(n)} \\ &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} = -\sum_k \delta_k(n)w_{kj}(n) \end{aligned}$$

. In addition, we also define the Jacobian matrix J of $e(i)$ where $e(i)$ is the error at time i .

$$J(m) = \begin{pmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \dots & \frac{\partial e_1}{\partial w_n} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \dots & \frac{\partial e_2}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_m}{\partial w_1} & \frac{\partial e_m}{\partial w_2} & \dots & \frac{\partial e_m}{\partial w_n} \end{pmatrix}$$

We know that in Newtons' Method, we minimise

$$\begin{aligned} \Delta\xi(w(n)) &= \xi(w(n+1)) - \xi(w(n)) \\ &\simeq g^T(n)\Delta w(n) + \frac{1}{2}\Delta w^T(n)H(n)\Delta w(n) \end{aligned}$$

By differentiating the equation with respect to $\Delta w(n)$ and equating it to 0, we get

$$\begin{aligned} g(n) + H(n)\Delta w(n) &= 0 \\ \Delta w(n) &= -H(n)^{-1}g(n) \end{aligned}$$

In the Gauss-Newton method, we assume $\xi(w) = \frac{1}{2} \sum e_j^2(w)$. Thus,

$$\begin{aligned} \nabla\xi(w) &= J^T(w)e(w) \\ \nabla^2\xi(w) &\approx J^T(w)J(w) \end{aligned}$$

. Using the equation from Newtons' method, we can easily see that

$$\Delta w = -(J^T(w)J(w))^{-1}J^T(w)e(w)$$

. However, for the calculation of Δw , we need to calculate $(J^T(w)J(w))^{-1}$. Thus, $J^T(w)J(w)$ should be an invertible matrix. We know that $J^T(w)J(w)$ is always a non- negative definite matrix. However, for it to be a non-singular matrix, it should have a row rank m , i.e. the m rows should be linearly independent. To ensure that this condition holds true, we apply a Marquardt-Levenberg modification to the Gauss-Newton method by adding a diagonal matrix δI to the term, where I is the identity matrix and δ is a small positive constant. Thus, the modified Gauss-Newton equation becomes

$$\Delta w = -(J^T(w)J(w) + \delta I)^{-1}J^T(w)e(w)$$

As, we have already seen standard backpropagation algorithm calculates terms like

$$\frac{\partial \xi}{\partial w_{ji}} = \frac{\partial \sum e_j^2(w)}{\partial w_{ji}}$$

. For the Marquardt-Levenberg algorithm, we need to calculate terms for the Jacobian matrix

$$\frac{\partial e_j(w)}{\partial w_{ji}}$$

. These terms for the Jacobian matrix can be calculated using the terms for the standard backpropagation algorithm with one modification at the final layer. In standard backpropagation, we calculate $\frac{\partial \xi}{\partial w_{ji}} = -e_j y_i \phi'(v_j)$ at the final layer. However, in Marquardt-Levenberg algorithm this is modified to $\frac{\partial e_j}{\partial w_{ji(n)}} = -y_i \phi'(v_j)$. The rest of the terms are then calculated using the backpropagation algorithm.

At each iteration, the weights are updated and the performance goal (i.e. sum of square of errors) is calculated. The algorithm is assumed to have converged when the norm of the gradient

$$\nabla \xi(w) = J^T(w)e(w)$$

is less than some predetermined value, or when the sum of squares has been reduced to some error goal.

In the Bayesian regularization method[9], we add a regularization term to the cost function $\xi(w) = \sum e_j^2(w)$. The new cost function that is to be optimised now becomes

$$\xi(w) = \beta \sum e_j^2(w) + \alpha \sum w_j^2$$

, where α and β are cost function parameters. If $\alpha \ll \beta$, then the training algorithm will drive the errors smaller. If $\alpha \gg \beta$, training will lead to weights of smaller magnitude at the expense of network errors, thus producing a smoother network response. In the Bayesian regularisation training algorithm, we assume a prior distribution on the weights $p(W|\alpha, M)$, where M is the model in consideration. When the data D is taken for training, the distribution of the weights is updated according to the Bayes' Rule:

$$p(w|D, \alpha, \beta, M) = \frac{p(D|w, \beta, M)p(w|\alpha, M)}{p(D|\alpha, \beta, M)}$$

$p(D|w, \beta, M)$ is the likelihood function of the data occurring given the weights and $p(D|\alpha, \beta, M)$

is the normalisation factor. Assuming

$$p(D|w, \beta, M) = \frac{1}{Z_D(\beta)} \exp(-\beta \sum e_j^2(w))$$

$$p(w|\alpha, M) = \frac{1}{Z_W(\alpha)} \exp(-\alpha \sum w_j^2)$$

. Here, $Z_D(\beta) = (\frac{\pi}{\beta})^{\frac{n}{2}}$ and $Z_W(\alpha) = (\frac{\pi}{\alpha})^{\frac{N}{2}}$, n is the number of data points and N is the number of network parameters. Combining all the equations, we get the posterior probability of the weights as

$$\begin{aligned} p(w|D, \alpha, \beta, M) &= \frac{\frac{1}{Z_D(\beta)Z_W(\alpha)} \exp(-(\beta \sum e_j^2(w) + \alpha \sum w_j^2))}{p(D|\alpha, \beta, M)} \\ &= \frac{1}{Z_F(\alpha, \beta)} \exp(-(\xi(w))) \end{aligned}$$

The optimal weights should maximise the posterior probability of the weights. From the equation above, we can see that the objective of maximising the posterior probability is equivalent to minimizing the regularized cost function.

For the optimisation of the regularisation parameters α and β , we have assumed a uniform prior $p(\alpha, \beta|M)$ on them. By Bayes Rules, we have :

$$p(\alpha, \beta|D, M) = \frac{p(D|\alpha, \beta, M)p(\alpha, \beta|M)}{p(D|M)}$$

Since the prior is uniform, the posterior probability is maximised if $p(D|\alpha, \beta, M)$ is maximised. From the equations above, we can see that

$$p(D|\alpha, \beta, M) = \frac{Z_F(\alpha, \beta)}{Z_D(\beta)Z_W(\alpha)}$$

The cost function is a convex function. Therefore, in a small area surrounding a minimum point, we can expand $\xi(w)$ around the minimum point of the posterior w^{MP} , where the gradient is zero. Solving for the normalization constant yields

$$Z_F \approx (2\pi)^{\frac{N}{2}} (\det((H^{MP})^{-1}))^{\frac{1}{2}} \exp(-\xi(w^{MP}))$$

where $H = \beta \nabla^2(\sum e_j^2) + \alpha \nabla^2(\sum w_j^2)$ is the Hessian matrix of the cost function. Placing this result into $p(D|\alpha, \beta, M) = \frac{Z_F(\alpha, \beta)}{Z_D(\beta)Z_W(\alpha)}$ and taking derivatives with respect to the log of the

equation and setting it to 0, we get:

$$\alpha^{MP} = \frac{\gamma}{2(w^{MP})^2} \text{ and } \beta^{MP} = \frac{n - \gamma}{2 \sum e_j^2(w^{MP})}$$

$\gamma = N - 2\alpha^{MP} \text{tr}(H^{MP})^{-1}$ in the above equation.

Algorithm 3: Bayesian Regularisation Training Algorithm

- 1 Initialize α , β and the weights for the network.
- 2 Using Levenberg-Marquardt algorithm, minimize the objective function

$$\xi(w) = \beta \sum e_j^2(w) + \alpha \sum w_j^2$$

- 3 Compute the $\gamma = N - 2\alpha^{MP} \text{tr}(H^{MP})^{-1}$ using of the Gauss-Newton approximation to the Hessian available in the Levenberg-Marquardt training algorithm:

$$H = \nabla^2 \xi(w) = 2\beta J^T J + 2\alpha I$$

. The Jacobian matrix as defined before is the Jacobian matrix of errors computed.

- 4 Compute α and β .
 - 5 Repeat till convergence.
-

Training of a neural network can happen in two modes:

1. Sequential Mode- In this mode, the update of weights is done after presentation of *each* training example. For example, the weight of the neural network is updated after (x_1, y_1) is presented. The updated network now is trained on the training example (x_2, y_2) and the weight is updated again. This goes on till all the training examples are presented to the network. Presentation of all the training example once to the network is defined as an epoch.
2. Batch mode -In this mode, the update of weights is done after presentation of *all* the training examples.

Sequential mode of training requires less computational storage than the batch mode of training. The use of pattern by pattern weight update makes the algorithm stochastic in nature. Thus, there it is less likely to get trapped in a local minimum. However, sequential training because of its stochastic nature makes it difficult to establish theoretical convergence conditions for the algorithm. This can easily be done when batch mode of training is used. Batch training accurately calculates gradient vectors at each point, thereby convergence to

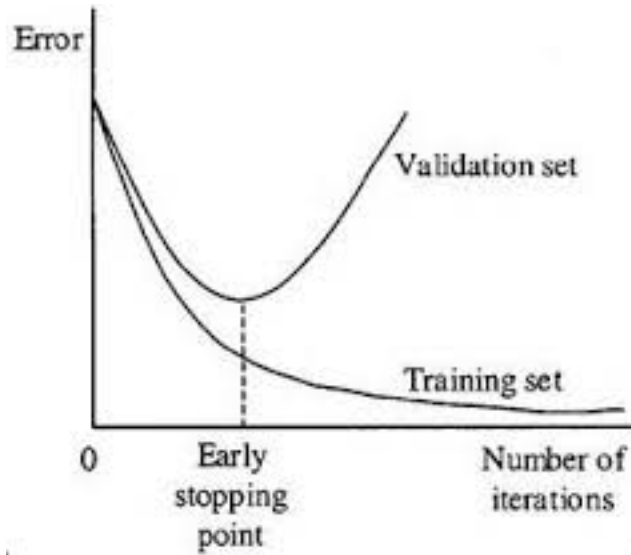


Figure 2.5: Error plot for Validation Set

a local minimum is guaranteed under simple conditions.

For checking the overfitting of the model, we use a part of data as validation data. After a period of training the network, we fix all the weights and the bias and compute the output for the validation set. The errors for the validation set are then calculated. The training is then resumed for another period and the process is repeated. Typically, the model gives a higher error on the validation set than the training set. This means that the mean square error, which is the performance goal of the network, keeps on decreasing with each epoch. Sometimes, however, as the neural network starts to overfit the training data, the error on the validation starts increasing. It is at this point that we stop the training of the neural network.

2.1.4 Results

The codes for the neural network were written in MATLAB and the simulations were run for different architectures of the neural network. In the first part, we used a feedforward neural network with one hidden layer. Activation functions used were tanh on the first layer and purelin or identity on the second layer. The training used was the Bayesian Regularization algorithm. We assumed that the model was $y(t) = f(y(t-1), y(t-2), \dots, y(t-d))$. The d was differed. The data was exchange rate data on US dollar/Euro from 2011 to present day. The data was normalised by using the formula $\frac{data_point - Min}{Max - Min}$. It was divided into three sets: 70 % for the training data, 15% for the validation data and remaining 15% for the test data.

The constant μ was kept at 0.005 and was changed at the rate of 0.1 for improving the fit. The maximum epochs for training was kept to be at 1000. An "epoch" describes the number of times the algorithm sees the ENTIRE data set. So each time the algorithm has seen all samples in the dataset, an epoch has completed. The training done was online training and in each epoch, all the training examples were presented and resulted in weights update of the network. The performance of the neural networks was measured as the mean square error or MSE. The training was carried to the full 1000 epochs if the performance goal, i.e. the difference in MSE between each epoch = 10^{-7} was not reached. Otherwise, the training was terminated at the epoch that the performance goal was reached. In the next part of the simulation, the number of neurons in the hidden layer was differed keeping the value of the d same.

Number of inputs	Number of hidden nodes	Training Performance	Validation Performance	Test Performance
4	10	2.080140e-04	2.765050e-04	9.939453e-04
5	10	2.301353e-04	2.988231e-04	2.468123e-03
6	10	2.797359e-04	2.787228e-04	2.257220e-03
8	10	2.184029e-04	2.921051e-04	2.237289e-03
10	10	2.069532e-04	2.771385e-04	9.960684e-04
12	10	2.060887e-04	2.770555e-04	8.663912e-04
15	10	2.062721e-04	2.770395e-04	1.090613e-03

Table 2.1: Performance of neural network keeping the number of hidden units same

Number of inputs	Number of hidden nodes	Training Performance	Validation Performance	Test Performance
12	10	2.155171e-04	2.832236e-04	3.106627e-03
12	15	2.973352e-04	3.008010e-04	2.038894e-03
12	20	2.054431e-04	2.778482e-04	7.293144e-04
12	24	2.061975e-04	2.774234e-04	6.658699e-04
12	30	2.156659e-04	2.766033e-04	6.727146e-04

Table 2.2: Performance of neural network keeping the number of input units same

Thus, we select a neural network of inputs 12 and hidden nodes 24.

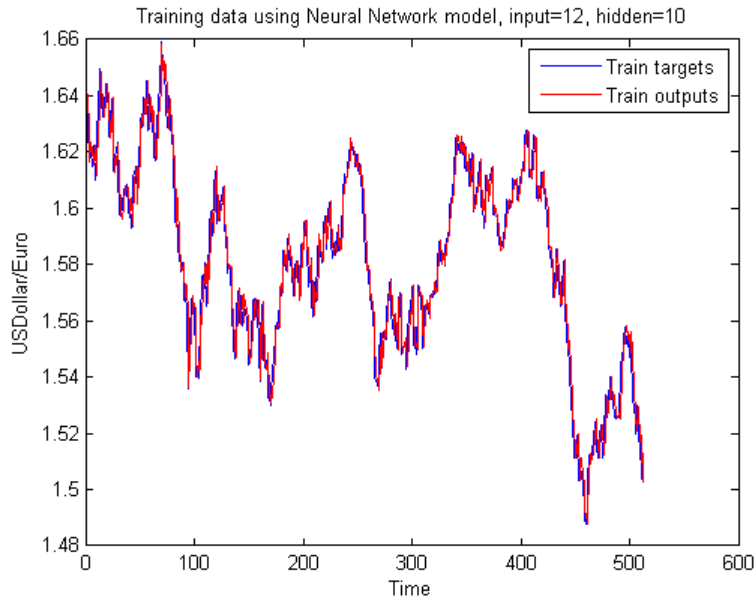


Figure 2.6: Training Data of Neural Network with input=12,hidden=10

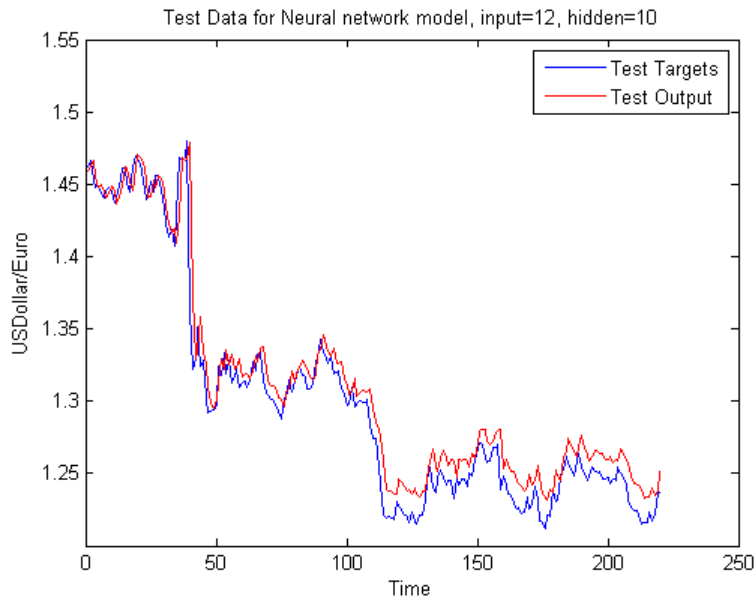


Figure 2.7: Test Data of Neural Network with input=12,hidden=10

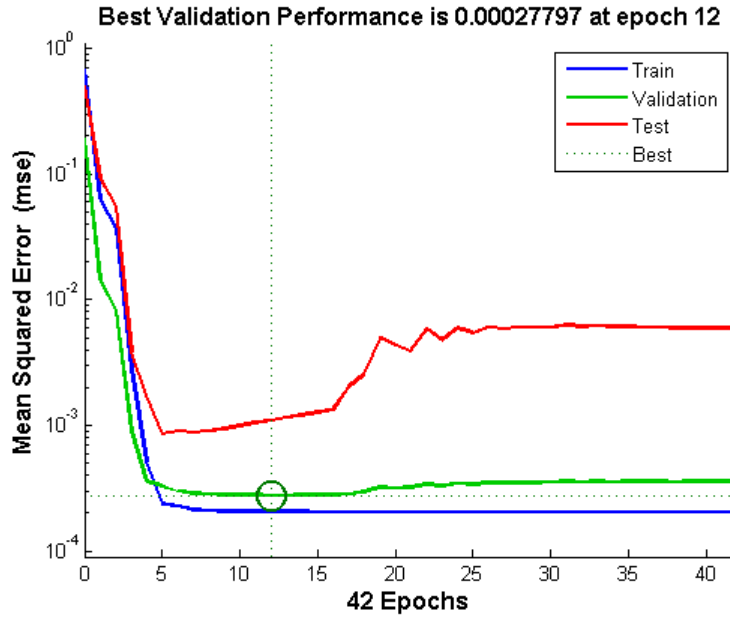


Figure 2.8: Performance of Neural Network with input=12, hidden=10

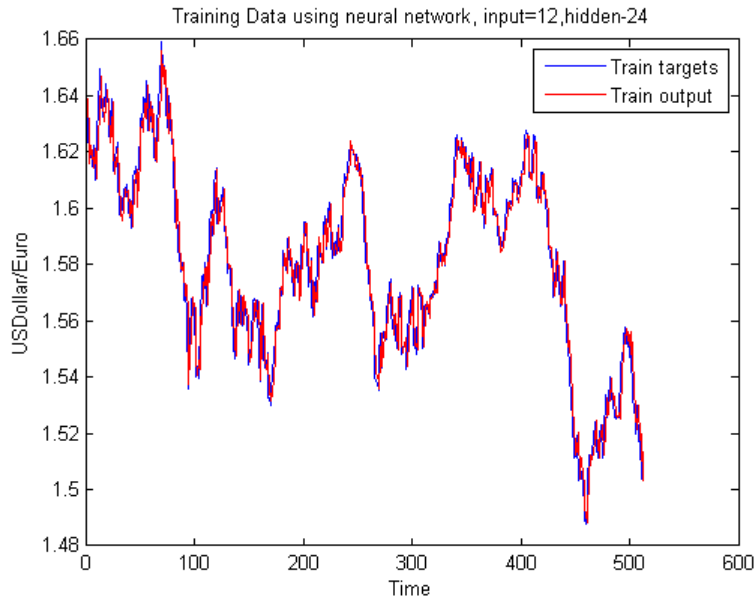


Figure 2.9: Training Data of Neural Network with input=12,hidden=24



Figure 2.10: Test Data of Neural Network with input=12,hidden=24

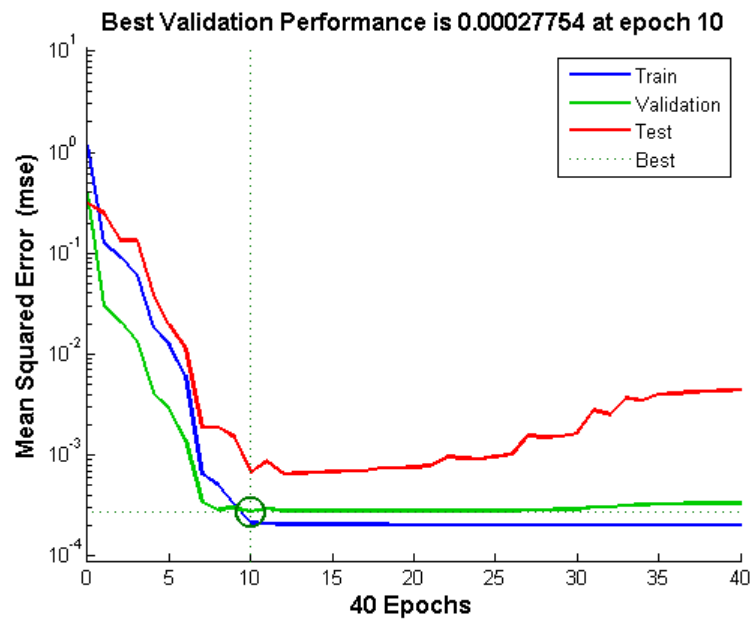


Figure 2.11: Performance of Neural Network with input=12, hidden=24

2.2 Genetic Algorithms and Neuro-Genetic Model

Genetic Algorithms are a search mechanism for optimum solution in the search space using techniques borrowed from concepts in natural genetics and selection. It requires the conversion of the space over which we need to find the optimised solution into a string, most common of which is using the binary $\{0, 1\}$ for integer solutions. In case of real or complex solutions, the space can also constitute vectors of the required solution. For example if we have to optimise $f(w_1, w_2)$, then the search space will constitute a vector of w_1 and w_2 over which the algorithm will be applied. In this thesis, we will use the terms "strings" or "vector" interchangeably.

The algorithm leads to finding of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm. Genetic algorithms have been characterised by robustness as they are not limited by assumptions concerning continuity, existence of derivatives, uni-modality and other auxiliary information, unlike the optimisation techniques used in neural networks. These algorithms perform an effective search for better string structures using only require payoff values i.e. value of fitness function associated with individual strings. This characteristic makes a GA a more canonical method than many search algorithms. A genetic algorithm is also different from more traditional optimisation approaches in the way that it works with a population of strings at a time rather than a single string.

Definition 4. Genetic Algorithms *A genetic algorithm (GA) is an optimisation method based on concepts of natural selection and genetics for solving constrained and unconstrained problems. This algorithm works with a population of solution vector or strings at each iteration and selects the "fittest" vector to be propagated to the next population also also produce new vectors for the next generation. Over successive generations, the algorithms tends to converge towards an optimal solution.[10]*

Definition 5. Fitness function *Fitness function is the function which is to be optimised and is used in genetic algorithms as a tool for finding the value of the variable over which the function is maximum in the given population.[10]*

A simple genetic algorithm is composed of three operators:

1. Selection- It is a process in which individual strings are copied according to their objective function values or the fitness function. This function, as defined above, is the measure of profit, utility, or goodness that we want to maximize. Strings with a higher value have a higher probability of reproducing.

2. Crossover- It is the process where members of the newly reproduced strings in the mating pool are mated at random. For each pair of strings, an integer position k along the string is randomly selected where k is anywhere between the 1st position on the string and one less than last position on the string, that is the $(l-1)$ th position for a string of length l . New strings are created by swapping all characters between positions $k + 1$ and l between the two strings.
3. Mutation-It is a process of extremely small probability that leads to random alteration of the value of a string position. In binary coding, this simply means changing a 1 to a 0 and vice-versa.

2.2.1 Genetic Algorithms: A simple example

A simple example of how genetic algorithm works will be illustrated with an example of finding the maximum value of the function $f(x) = x^3$ over the set of integers $x \in [1, 64]$.

Aim: To find x such that $f(x)$ is maximum.

Solution: Through calculus, we already know the maximum of the function will exist at $x = 64$. Since the optimisation has to be done over a set of integers, we first convert the set of integers into binary numbers. These will create a set of vectors $[(0,0,0,0,0,0,0),(0,0,0,0,0,0,1),(0,0,0,0,0,1,1),\dots,(1,0,0,0,0,0,0)]$. We select a random population of 4 vectors, say:

$$(0,0,1,0,1,0,0)= 20;$$

$$(0,1,1,0,0,0,0)=48;$$

$$(0,0,0,1,0,0,1)=9;$$

$$(0,0,1,0,0,0,0)=16$$

Calculating $f(x) = x^3$, we get the fitness functions as The probability of each vector get-

Vector	Value	Fitness function	Probability
(0,0,1,0,1,0,0)	20	8000	0.0648
(0,1,1,0,0,0,0)	48	110592	0.896
(0,0,0,1,0,0,1)	9	729	0.0059
(0,0,1,0,0,0,0)	16	4096	0.0331
Total =123417			1

Table 2.3: Fitness values for the first population

ting selected is calculated by $p_i = \frac{f_i}{\Sigma f_i}$, where Σf_i is the value of the total fitness of all the vectors in the population. Since $(0,1,1,0,0,0,0)=48$ has a very high fitness function, it

has a very high probability(=0.896) of getting selected and passed onto the next generation. Once a vector has been selected for reproduction, an exact replica of the string is made. This vector is then entered into a mating pool for further genetic operator action for the creation of a next population. Thus, highly fit vectors have a higher number of offspring in the succeeding generation. Suppose the vectors selected to be considered for reproduction are $(0,0,1,0,1,0,0)= 20$, $(0,1,1,0,0,0,0)=48$, $(0,0,1,0,0,0,0)=16$. We randomly mate $(0,0,1,0,1,0,0)= 20$, $(0,1,1,0,0,0,0)=48$ by crossing them over at position 2. Thus, the new strings created are $(0,0,1,0,0,0,0)= 16$ and $(0,1,1,0,1,0,0)=56$. Thus, the new randomly selected population is

$(0,1,1,0,0,0,0)=48$;

$(0,0,1,0,0,0,0)= 16$;

$(0,1,1,0,1,0,0)=56$;

$(0,0,1,0,1,0,0)= 20$.

We immediately see that in the next generation we have a better vector $(0,1,1,0,1,0,0)=56$ which is close to the true value at which we get the optimal solution $x = 64$.

As we keep iterating it over the next few populations, the algorithm will converge to a point x in the search space which is very close to the true value. In the next section, we will discuss why the genetic algorithms work and give us the optimal values and the Fundamental Theorem of Genetic Algorithms, also known as the Schemata Theorem.

2.2.2 Genetic Algorithms: The Schemata Theorem

Definition 6. *A schema (Holland, 1968, 1975) is a similarity template describing a subset of strings with similarities at certain string positions.*

A schema H is a way of determining how similar to vectors are in the search space. We will elucidate this using the binary numbers as an example as has been done in the previous section. A schema on the vector of binary numbers can be defined as a vector formed on the set $\{1,0,*\}$, which is an extension of the set $\{1,0\}$ on which the binary vectors are formed. The $*$ can represent either 1 or 0. For example, a schema $(1,*,1,0,0)$ can refer to either vector of $(1,1,1,0,0)$ or $(1,0,1,0,0)$. Thus, the schema represents the fact that the above vectors $(1,1,1,0,0)$ and $(1,0,1,0,0)$ are similar.

Definition 7. Order of a schema *The order of a schema H , denoted by $o(H)$ is defined as the number of fixed positions present in the template. For example, the order of the schema $(0,1,1,*,*,1,*,*)$ is 4.*

Definition 8. Length of a Schema *The length of a schema H , denoted by $\delta(H)$ is defined as the distance between the first and last specific position in the vector. For example, the schema $(0,1,1,*,*,1,*,*)$ has defining length $\delta = 5$ because the last specific position is 6 and the first specific position is 1.*

Schemas are generally known as the building blocks of a population in genetic algorithms. Schemas with their property of attaching a similarity template to vectors help us divide the vectors in a population into groups of their schema types. This helps us in understanding the underlying mechanism of how the genetic algorithm work on the schemas in the population, thereby giving us the mechanism of how GAs work on the whole population. The net effect of the genetic operators like selection and crossover on the whole can be easily seen by their action on the schemas present in the population. Some schemas contain more definite information than others. For example, the schema $(0,1,1,*,1,*,*)$ tells us more about the structure of the possible vectors of the form than the schema $(0,*,*,*,*,*)$. Combinatorially, there are 3^l schemas that can be formed for a binary vector of length l . Furthermore, for n vectors in a population, there are at most $n \cdot 2^l$ schemata contained in the population because each string represents 2^l possible schemas. We will now look at how genetic algorithms work on the schemas present in the population. This will be done by studying the net effect of genetic operators like selection, crossover and mutation on a particular schema H in the population.

The first operator that we will look at is the selection operator. Given time step t , suppose there are m vectors of a particular schema H that is present in the population $P(t)$. Thus for the time t , we write $m = m(H, t)$. We already know that the basic premise of a vector getting selected for reproduction in a genetic algorithm is its fitness according to the fitness function assigned to the problem. Thus, in a population, $P(t)$, a vector P_i gets selected with probability $p_i = \frac{f_i}{\Sigma f_i}$, where Σf_i is the value of the total fitness of all the vectors in the population. Based on fitness values, we select another generation of n individuals for time step $t + 1$. Thus, the number of vectors that are the representation of schema H in the population $P(t + 1)$ is denoted as $m(H, t + 1)$ and this quantity is calculated using the total fitness values of all the vectors that represented schema H in the population $P(t)$. Thus,

$$\begin{aligned} m(H, t + 1) &= m(H, t) \cdot n \cdot \frac{f(H)}{\Sigma f_i} \\ &= m(H, t) \frac{f(H)}{\hat{f}} \end{aligned}$$

where $f(H)$ is the average fitness of the vectors of schema type H at time t and \hat{f} is the average fitness of the entire population. This shows us that the schemas with average fitness that is greater than the average population fitness have an increasing representation in the next generation of population that is formed. Also, schemas with average fitness lesser than the average population fitness tend to have lesser representation over the generations which may even cause them to die out completely.

The second operator in consideration is the crossover operator of genetic operator. Crossover is an operator which randomly selects a position in the length of the vector and then crosses it over with another vector at that position to create two new vectors. Crossover in a schema has the effect of retaining it for the next generation or destroying it in the generation in consideration itself. This is because if the crossover site is a number more than the length of the schema $\delta(H)$, the schema is retained. Otherwise, it is destroyed. For example, for the schema $(0,1,1,*,1,*,*)$, a crossover at position 3 would break the schema $(0,1,1,|*,1,*,*)$ whereas a crossover at position 5 will retain the schema $(0,1,1,*,1,|*,*)$. Therefore, a schema with length $\delta(H)$ and the vector of length l has the choice of having $l - 1$ crossover sites. Thus, the schema in the process of crossover is destroyed by probability $p_d = \frac{\delta(H)}{l-1}$. The probability of survival of the schema would then be

$$p_s = 1 - \frac{\delta(H)}{l-1}$$

. If the crossover has a random probability p_c assigned to it, then

$$p_s \geq 1 - p_c \cdot \frac{\delta(H)}{l-1}$$

. We now combine this with the results we had for the selection operator by assuming that the two operators are independent. The number of vectors that follow schema H in the time step $t + 1$ can now be calculated as

$$m(H, t + 1) \geq m(H, t) \cdot \frac{f(H)}{\hat{f}} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{l-1}\right)$$

As we discussed that selection of a schema depends on its average fitness in comparison to the average population fitness, crossover depends on the length of the schema. Schemas with short lengths have a higher chance of survival. Thus, schemas with short lengths and average fitness above the average fitness of the population are sampled at increasing rates with each generation.

The last operator that we will discuss is mutation. Mutation is an operator that alters the value of the vector at a position, i.e. by changing 0 to 1 at a certain position and vice versa. For a schema to survive, all the specified positions must not be altered. Therefore if the probability of mutation is p_m , the schema survives with probability $(1 - p_m)^{o(H)}$, where $o(H)$ is the order of the schema as defined above (assuming mutation at each position is independent). Since, the probability of mutation is generally taken to be an extremely small value, the probability of survival of schema can be approximated by $1 - o(H) \cdot p_m$.

We now derive the **Fundamental Theorem of Genetic Algorithms**. This algorithm gives us the number of vectors of a particular schema that can be expected in the next generation or at time step $t+1$ when the genetic operators of selection, crossover and mutation are applied. Assuming each of the operators are independent and ignoring the small cross-product terms, we get

$$\begin{aligned} m(H, t+1) &\geq m(H, t) \cdot \frac{f(H)}{\hat{f}} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{l-1}\right) \cdot (1 - o(H) \cdot p_m) \\ &\geq m(H, t) \cdot \frac{f(H)}{\hat{f}} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{l-1} - o(H) \cdot p_m\right) \end{aligned}$$

The Fundamental theorem then states that short, low-order, above-average schemata receive increasingly represented in subsequent generations.

2.2.3 Neuro-Genetic Model: Algorithms and Training

For the neuro-genetic model for forecasting, two different approaches were used. In the first part, the genetic algorithm was applied to optimise the set of weights used in the neural network while keeping the architecture fixed. The architecture was determined by the results of the previous section on neural network models. The second part involved applying genetic algorithm on the architecture of the neural network and then training the neural network, i.e. the number of inputs and the number of neurons in the hidden layer. The number of inputs considered was on a set of integers from [1,20] and the number of neurons in the hidden layer was [1,30]. The simulations were run on MATLAB and the data that was used was US Dollar/Euro exchange rate data from 2011 to present day. The performance of each neural network was measured by mean square error or MSE and the fitness function was taken to be $f(x) = \frac{1}{MSE}$ (fitter the neural network, lesser the mean square error).

Algorithm 4: Genetic Algorithm applied on weight vector

- 1 Convert all the set of weights used in training of network into a single vector.
 - 2 Initialise a population of 50 vectors of weights.
 - 3 For each vector, initialise the weight of the neural network with the vector and train the neural network.
 - 4 Calculate the performance of each neural network and then calculate the fitness values of each of the neural network.
 - 5 Based on the fitness values, select the weight vectors with highest fitness and pass it onto the next generation as *elite* children. The pSelect was a random value generated between $[0.75,1]$.
 - 6 Apply other operators of six- point crossover (crossover at 6 positions) and mutation and create crossover and mutation children.
 - 7 Repeat till 100 iterations. For each generation, note down the set of weights that has given the least mean square error upon training.
 - 8 Select the set of weights which gives the least MSE over the generations and train your neural network using those set of weights.
-

Algorithm 5: Genetic Algorithm applied on network architecture

- 1 Convert the set of integers [1,20] and [11,30] into binary vectors.
 - 2 Initialise the binary vectors of [1,20] and [11,30] into the initial population for number of input nodes and number of hidden nodes respectively .
 - 3 Let V_I be the vector for the input in the population and V_H be the vector for the hidden node.
 - 4 For each vector V_I and V_H , initialise the number of inputs and the number of hidden nodes of the neural network .
 - 5 Calculate the performance of each neural network and then calculate the fitness values of each of the neural network.
 - 6 Based on the fitness values, select the vector V_I and V_H with highest fitness and pass it to the next generation as *elite* children. The pSelect was a random value generated between [0.75,1].
 - 7 Apply other operators of three- point crossover (crossover at 3 positions) and mutation on the vectors V_I and V_H in the population and create crossover and mutation children
 - 8 Repeat till 100 iterations. For each generation, note down the number of inputs that has given the least mean square error upon training.
 - 9 Select the number of input and hidden nodes which gives the least MSE over the generations and train your neural network using the same number of inputs and hidden nodes.
-

2.2.4 Results

For a neural network of 12 inputs and 24 hidden nodes, we applied the genetic algorithm as described in the algorithm 1 above. The algorithm gave the minimum MSE at 59th iteration for the 14th weighth vector.

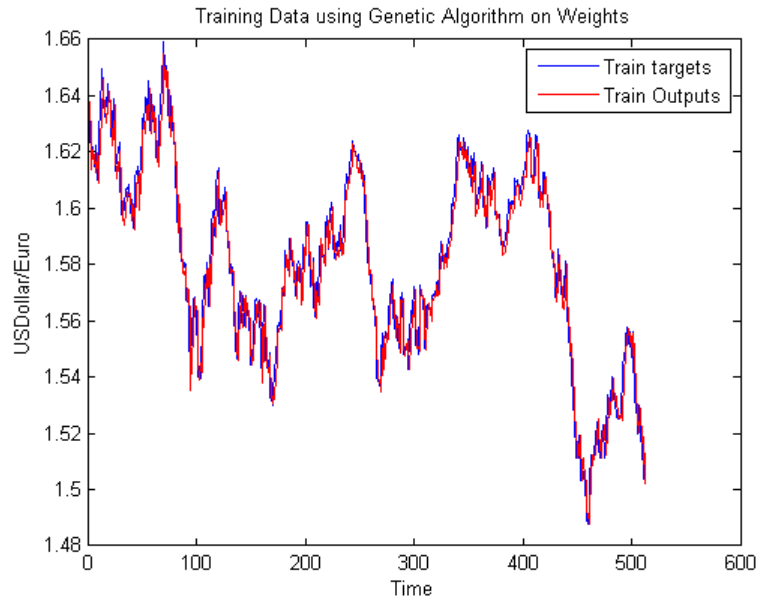


Figure 2.12: Training Data of Network with Genetic algorithm on Weights

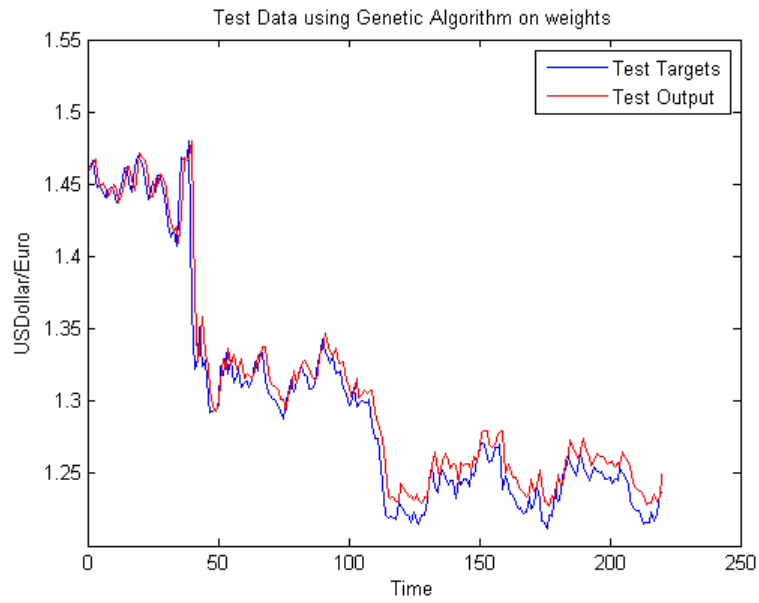


Figure 2.13: Test Data of Network with Genetic Algorithm on Weights

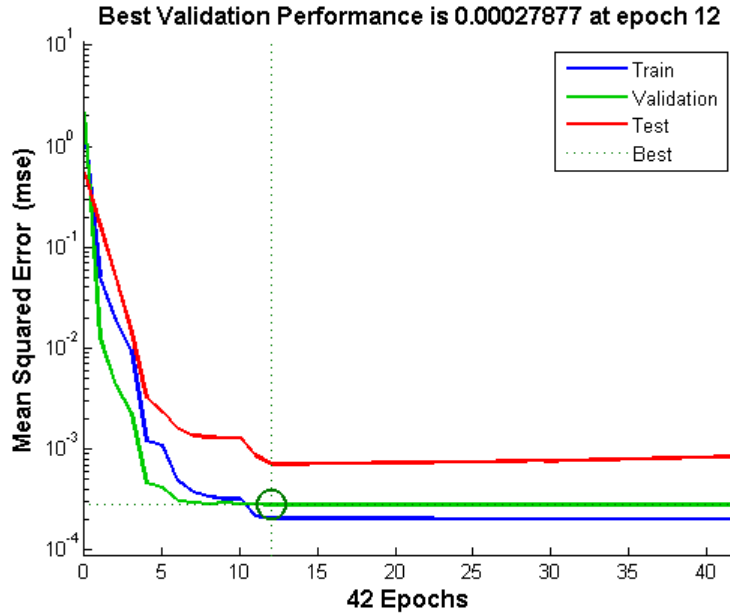


Figure 2.14: Performance of Network with Genetic Algorithm on Weights

A part of table that shows the training performance and test performance for a part of population is shown as follows:

Population number	Mean MSE of population	Test Performance
51	8.275065e-04	6.418594e-04
52	8.654775e-04	6.226621e-04
53	1.052213e-03	6.456736e-04
54	8.428374e-04	6.211825e-04
55	8.354304e-04	6.363530e-04
56	7.883199e-04	6.368228e-04
57	8.200971e-04	6.296454e-04
58	1.835029e-03	6.291646e-04
59	7.907987e-04	5.987798e-04
60	8.716574e-04	6.302475e-04

Table 2.4: Mean MSE and Test Performance of Populations of weights

In the second part, genetic algorithm was applied on the architecture of the neural net-

work, i.e the number of input units and the number of hidden nodes. The minimum MSE was found at the 9th iteration for input units=14 and hidden nodes=28.

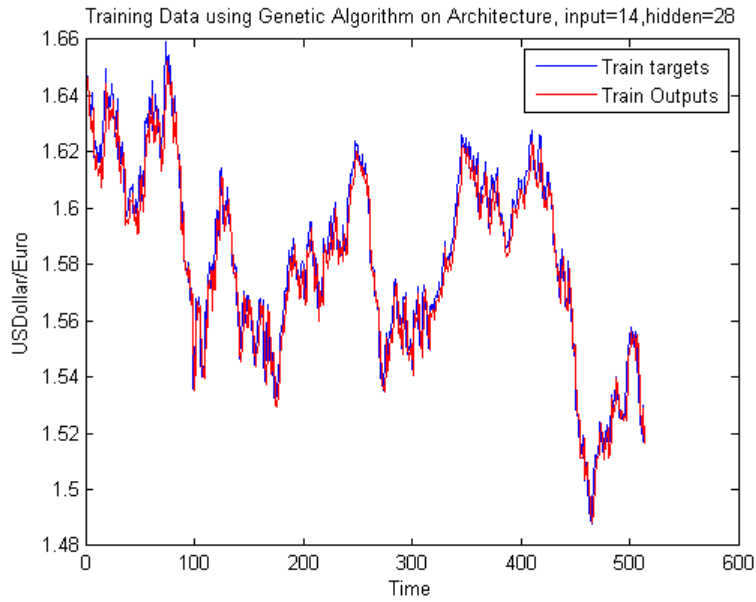


Figure 2.15: Training Data of Network with Genetic algorithm on Architecture

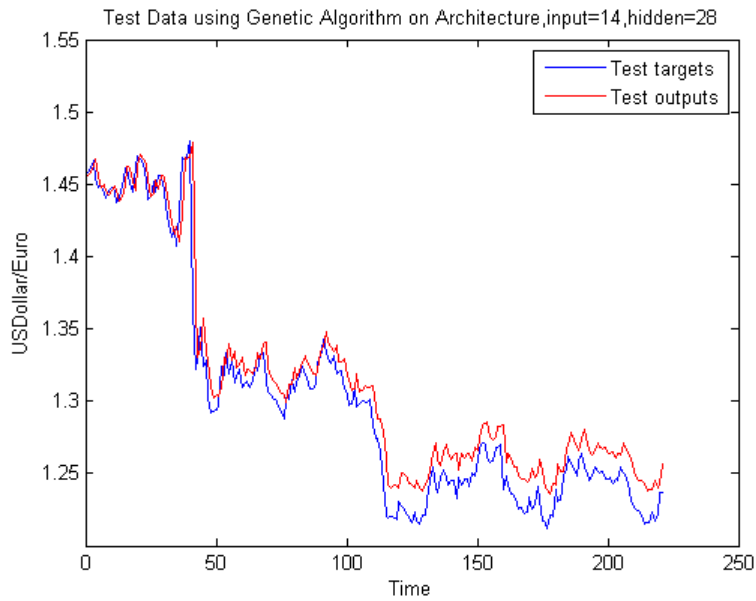


Figure 2.16: Test Data of Network with Genetic Algorithm on Architecture

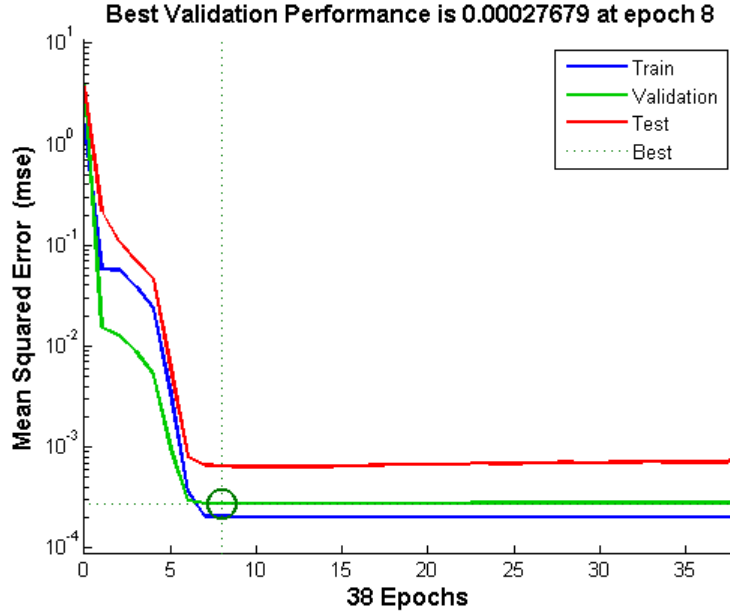


Figure 2.17: Performance of Network with Genetic Algorithm on Architecture

A part of table that shows the training performance and test performance for a part of population is shown as follows:

Population number	Mean MSE of population	Test Performance
1	1.186607e-03	6.587150e-04
2	9.039590e-04	6.483756e-04
3	8.422916e-03	6.336099e-04
4	6.608447e-03	6.700080e-04
5	1.432892e-02	6.472539e-04
6	1.370173e-03	6.330162e-04
7	7.333263e-04	6.374157e-04
8	1.192145e-03	6.337274e-04
9	7.786330e-04	6.158535e-04
10	1.302913e-03	6.281796e-04

Table 2.5: Mean MSE and Test Performance of Populations of number of input and hidden nodes

2.3 Neuro- Fuzzy Systems

Classically, the general theories in mathematics are constructed on sets which are clearly defined. However, human language is not as precise and definitive as mathematics is. For example, if we say the temperature is cold, it may require a broad set of temperatures to define what "cold" means to different people. In classical logic, the truth values are assigned to a quantity is true or false i.e. the temperature is either cold or hot. In fuzzy logic, however, the temperature is characterised by various boundaries of truth values which accommodate temperatures which may fall in the range of "very cold", "cold", "moderately cold", "moderately hot", "hot" and "very hot". Fuzzy logic as conceptualised by Lofti Zadeh in 1965 is an attempt to bridge this vagueness of human linguistics into a mathematical concept.

A fuzzy concept is an intuitive way of mathematically conceptualising the qualitative description used in everyday language. Additionally, it can easily accommodate imprecise data on which the logic has to be applied. We will begin by formally defining what fuzzy logic is.

2.3.1 Fuzzy Logic and Inference

Definition 9. Fuzzy Logic *Fuzzy logic is a superset of conventional(Boolean) logic that has been extended to handle the concept of partial truth- truth values between "completely true" and "completely false". The importance of fuzzy logic derives from the fact that most modes of human reasoning and especially common sense reasoning are approximate in nature.*

Fuzzy logic and the rules associated were formally defined by Zadeh in a groundbreaking paper in 1965. The basic premise on which defines the characteristics of a fuzzy logic are listed below (Zadeh,1965)[11]:

1. In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning.
2. In fuzzy logic everything is a matter of degree.
3. Any logical system can be fuzzified.
4. In fuzzy logic, knowledge is interpreted as a collection of elastic or, equivalently, fuzzy constraint on a collection of variables
5. Inference is viewed as a process of propagation of elastic constraints.

Fuzzy logic, as seen in the construct above, is seen as a generalization of classical logic. It is a system which can be used to deal with problems which have imprecise data or in which the rules of inference are formulated in a very general way making use of diffuse categories. Fuzzy logic offers a whole continuum of truth values for logical propositions instead of just 0 or 1. For example, a proposition P in a fuzzy logic can be assigned a truth value of 0.3

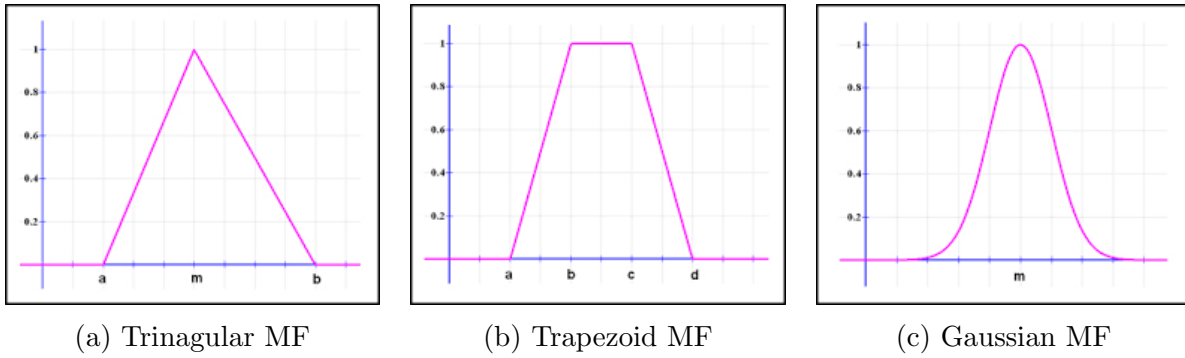


Figure 2.18: Types of Membership Functions

whereas its complement is assigned a truth value of 0.5. In fuzzy logic, truth values of a proposition and its negation need not necessarily add up to 1.

The most important mathematical construct that is involved in the fuzzy logic is the membership function. Suppose there are elements $\{x_1, x_2, x_3 \dots x_n\}$, which belong to set A . In a classical setup, the elements either belong to set A or they do not. Thus, if x_1 belongs to set A , it has a "membership" of 1. Otherwise it has a "membership" of 0. However, in a fuzzy logic setup, the element x_i can be present in the set A with a degree of membership which can be any real number between (0,1). For example, if the membership of temperatures is defined in the sets "Very Cold", "Cold" and "Moderately Cold", then a temperature of 10°C can belong to each of the sets with degree of memberships 0.4, 0.5 and 0.05 respectively.

We will now formally define the membership function for a fuzzy logic setup.

Definition 10. Membership Functions Let X be a classical universal set. A real function $\mu_A : X \rightarrow [0, 1]$ is called the membership function of A and defines the fuzzy set A of X . This is the set of all pairs $(x, \mu_A(x))$ with $x \in X$. (Rojas, 1996)[12]

A fuzzy set A described on the set X with elements $\{x_1, x_2, \dots, x_n\}$ can be defined as can be described in the following way

$$A = \mu_1/x_1 + \mu_2/x_2 + \dots + \mu_n/x_n$$

. There are different types of membership function, of which we will describe three.

- Triangular function: defined by a lower limit a , an upper limit b , and a value m , where

$a < m < b$.

$$\mu_A(x) = \begin{cases} 0 & \text{if } x \leq a \\ \frac{x-m}{m-a}, & \text{if } a < x \leq m \\ \frac{b-x}{b-m}, & \text{if } m < x < b \\ 0 & \text{if } x \geq b \end{cases}$$

- Trapezoidal function: defined by a lower limit a , an upper limit d , a lower support limit b , and an upper support limit c , where $a < b < c < d$.

$$\mu_A(x) = \begin{cases} 0 & \text{if } x < a \text{ or } x > d \\ \frac{x-a}{b-a}, & \text{if } a \leq x \leq b \\ 1 & \text{if } b < x < c \\ \frac{d-x}{d-c}, & \text{if } c < x \leq d \end{cases}$$

- Gaussian function: defined by mean m and a standard deviation $k > 0$.

$$\mu_A(x) = e^{-\frac{(x-m)^2}{2k}}$$

The set operations defined on a fuzzy set is:

1. Union: $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)) \forall x \in X$
2. Intersection: $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) \forall x \in X$
3. Complement: $\mu_{A^c}(x) = 1 - \mu_A(x) \forall x \in X$.

In a fuzzy set, classical rules of complement may not apply. Thus,

$$\begin{aligned} A \cup A^c &\neq X \\ A \cap A^c &\neq \Phi \end{aligned}$$

In classical set theory, there exists an isomorphism between the there operators of set theory (union, intersection and complement) and logic (OR, AND, NOT) respectively. A similar isomorphism can be drawn for fuzzy set operations and fuzzy propositional logic. The fuzzy (OR,AND, NOT) operators must be defined in such a way that the relations defined by the operators are analogous to their equivalents in classical set theory and logic.

One example of defining (OR,AND, NOT) for a fuzzy logic set is :

1. OR ($\tilde{\vee}$) =maximum function
2. AND ($\tilde{\wedge}$) =minimum function

3. NOT ($\bar{\sim}$) = 1-x.

Other examples of the operators can include multiplication x AND $y = (x * y)$ and x OR $y = 1 - (1 - x) * (1 - y)$. With the operators (OR,AND,NOT) now defined, we show with an example how **Fuzzy Inference** is done. Fuzzy inference rules have the same structure as classical ones[12].

- Let

$$R_1 : If(X\tilde{\wedge}Y) \text{ then } W.$$

$$R_2 : If(X\tilde{\vee}Y) \text{ then } Z.$$

- $\tilde{\wedge}$ = min function and $\tilde{\vee}$ = max function.
- Let the truth values of X and Y be 0.6 and 0.8 respectively. Then,

$$(X\tilde{\wedge}Y) = \min(0.6, 0.8) = 0.6$$

$$(X\tilde{\vee}Y) = \max(0.6, 0.8) = 0.8$$

This is interpreted by the fuzzy inference mechanism as meaning that the rules R_1 and R_2 can only be partially applied, that is percentage of application of rule R_1 is 60% and that of rule R_2 to 80%. The result of the inference is a combination of the propositions W and Z.

2.3.2 Fuzzy Inference System

From the example above, we can see that there are five functional blocks in any **Fuzzy Inference System**[13]:

1. A **fuzzification interface** which transforms the crisp inputs into degrees of match with linguistic values.
2. A **rule base** containing a number of fuzzy if-then rules.
3. A **database** which defines the membership functions of the fuzzy sets used in the fuzzy rules.
4. A **decision-making unit** which performs the inference operations of the rules.
5. A **defuzzification interface** which transform the fuzzy results of the inference into a crisp output.

The inference by the fuzzy inference systems is then done in the following sequence[13]:

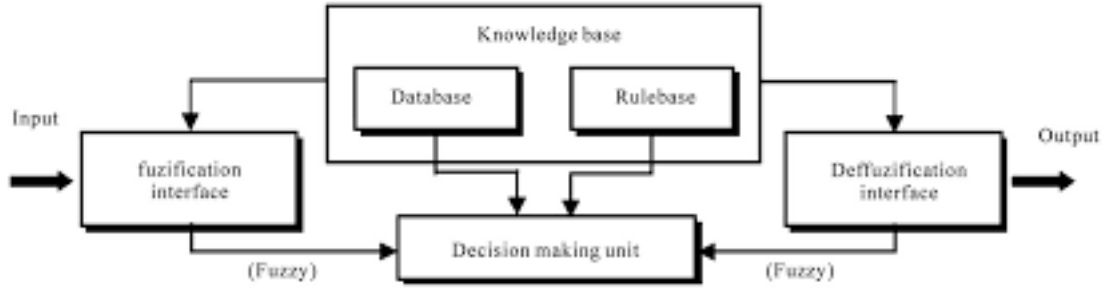


Figure 2.19: Fuzzy Inference System

1. **Fuzzification**-Compare the input variables with the membership functions on the premise part to obtain the membership values of each linguistic label.
2. Combine (through a specific fuzzy logic operator(OR, AND, NOT)) the membership values on the premise part to get the weight of each rule.
3. Generate the fuzzy output (consequent) of each rule depending on the weight obtained in the previous step.
4. **Defuzzification**-Aggregate the weighted consequents to produce a crisp output.

The three main types of Fuzzy inferences that exist are as follows[13]:

1. The output is the weighted average of each rules crisp output. The weight of each output is detemined by the output membership function.The output membership functions used in this scheme must be monotonic functions.
2. **Mamdani Inference System**The overall fuzzy output is derived by applying max operation to the fuzzy outputs (each of which is equal to the value obtained when fuzzy operator applied to the inputs and the output membership function of each rule)., at each point. This gives us the membership function that represents "max" output at each point.The final crisp output is then chosen using various criterion; some of them are centroid of area, bisector of area, mean of maxima, maximum criterion, etc .
3. **Sugeno-Takagi Inference System** The output of each rule is a linear combination of input variables plus a constant term, and the final output is the weighted average of each rules output.

2.3.3 ANFIS Architecture and Training

Since we would be using a Sugeno-type Inference System to train our ANFIS For simplicity, we assume the fuzzy inference system under consideration has two inputs x and y and

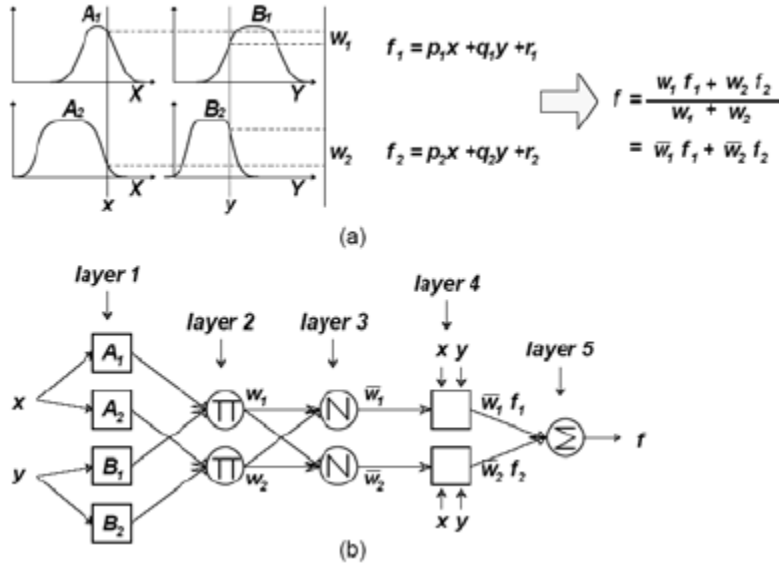


Figure 2.20: Architecture of ANFIS for Sugeno-type Inference

one output z . Suppose that the rule base contains two fuzzy if-then rules of Takagi and Sugenos type Since we would be using a Sugeno-type Inference System to train our ANFIS for exchange rate data, we would present a simple ANFIS model to illustrate how it is trained.

Suppose we have two inputs x and y and one output z . The rule-base accoring to Sugeno-Takagi inference system would be :

1. Rule 1: If x is A_1 and y is B_1 , then $f_1 = p_1x + q_1y + r_1$.
2. Rule 2: If x is A_2 and y is B_2 , then $f_2 = p_2x + q_2y + r_2$.

As shown in figure 2.8, we describe each of the layers in the ANFIS structure.

1. **Layer 1:** This is the layer where the inputs x and y are associated to their linguistic variables A_i s and B_i s respectively. The membership function is defined

$$O_i^1 = \mu_{A_i}(x)$$

This value detemines to the degree to which the given x satisfies the linguistic value A_i . The membership function can be any function as described in the previous section. Suppose we take the membership function to be the Gaussian function. Then,

$$\mu_{A_i}(x) = e^{-\frac{(x-a_i)^2}{2b_i}}$$

The parameters used in the membership function $\{a_i, b_i\}$ are known as *premise parameters*. As the values of these parameters change, the membership functions varies accordingly.

2. **Layer 2:** This is the node where the operator is applied. In the rule above, we have stated the operator to be "AND" operator, which signifies multiplication. Other "AND" operators could also be used instead of multiplication, like maximum. This gives us the weight obtained for each rule.

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y)$$

3. **Layer 3:** The i-th node in this layer normalises the weights obtained in the previous layer for the i-th rule.

$$\bar{w}_i = \frac{w_i}{\sum w_j}$$

4. **Layer 4:** The nodes at this layer calculate the output for each rule.

$$O_i^4 = \bar{w}_i f_i = \bar{w}_i (p_i x + q_i y + r_i)$$

$\{p_i, q_i, r_i\}$ are known as the *consequent parameters*.

5. **Layer 5:** The nodes at this layer calculate the overall output.

$$O_1^5 = \sum_i \bar{w}_i f_i$$

For type-1 fuzzy inference systems, the extension is quite straightforward. In an ANFIS for type-1 fuzzy inference system, the output of each rule is induced jointly by the output membership function and the firing strength. For type-2 fuzzy inference systems, we replace the centroid defuzzification operator with a discrete version which calculates the approximate centroid of area and then construct an ANFIS similar to that of type-3 ANFIS model. The training for an ANFIS is done using a hybrid training algorithm, which uses both gradient descent and Least Squares Estimate. The training is an online or sequential mode of training. We now describe the Least squares estimate or the LSE algorithm. Let the function that is approximated by the ANFIS be F . Then for input I and parameters S of the network, output O

$$O = F(I, S)$$

Algorithm 6: Hybrid Online Training for ANFIS

- 1 Initialize the input membership functions and the rules for the ANFIS structure
 - 2 Identify the premise parameters and the consequent parameters.
 - 3 In the forward pass, keep the premise parameters fixed. Pass the node outputs and use the values to adjust consequent parameters using Least Squares Estimate.
 - 4 In the backward pass, keep the consequent parameters fixed. Using the errors obtained from the target and the output and gradient descent algorithm, adjust the values of the premise parameters
 - 5 Repeat till convergence.
-

If there exists a function H such that the composite function $H \circ F$ is linear in some elements of S then these elements can be identified by Least Squares Method. Thus, if the parameter set S can be decomposed into two sets

$$S = S_1 \oplus S_2 \text{ (}\oplus\text{ direct sum)}$$

, such that $H \circ F$ is linear in the elements of S_2 . Thus,

$$H(O) = H \circ F(I, S)$$

which is linear in the elements of S_2 . If the values of S_1 are given, then $H(O) = B$ is calculated. $H \circ F = A$ is a linear function of given input I and unknown parameters $S_2 = X$. Thus, the function can now be written as

$$AX = B$$

, which is a standard linear least square problem.

$$X = (A^T A)^{-1} A^T B$$

. This computation is however expensive as it involves calculation of inverse of a matrix. Moreover, if $A^T A$ is a singular matrix, inverse does not exist. As a result, we use sequential formulas to compute the LSE of X . This sequential method of LSE is more efficient computationally. Specifically, let the i th row vector of matrix A be a_i and the i th element of B be

b_i . Then, then X can be calculated iteratively using the sequential formulas:

$$X_{i+1} = X_i + S_{i+1}a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i)$$

$$S_{i+1} = S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}$$

The initial conditions are $X_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$, where $M = |X|$. In online training, we modify the equation $S_{i+1} = \frac{1}{\lambda} [S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}]$. The factor $\lambda \in (0, 1)$ is to account for the time-varying characteristics of the incoming data. In an online training, we need to give more weightage to the newer incoming data pairs and also decay the effects of old data pairs. λ is a factor is added to the equations gives more weightage to the newer data pairs. The smaller λ is, faster the effects of old data decay.

2.3.4 Results

We used a Takagi-Sugeno Fuzzy Inference system for modelling the exchange rate data and the modelling was done using MATLAB. The exchange rate data was USDollar/Euro exchange Rate data from 2011 to present day. We assumed that the model was $y(t) = f(y(t-1), y(t-2), \dots, y(t-d))$. We took $d=12$ using the results of the neural network model. The input were put into fuzzy clusters who's numbers were differed, based on their values. Based on the number of clusters, rules were formed. These are listed as follows:

- If input-1 is in cluster-1, input-2 is in cluster-1,....., input-12 is in cluster-1, then output is in cluster-1.
- If input-1 is in cluster-2, input-2 is in cluster-2,....., input-12 is in cluster-2, then output is in cluster-2.
- If input-1 is in cluster-n, input-2 is in cluster-n,....., input-12 is in cluster-n, then output is in cluster-n.

The clusters were assigned using FCM subroutine in MATLAB. These clusters were then assigned gaussian membership functions. The rate of learning was 0.01 and the rate of decrease and increase of step size was 0.9 and 1.1 respectively. The performance was measured in terms of mean square error or MSE.

Clusters	Training Performance	Test Performance
4	1.9334e-04	7.9942e-04
5	1.9917e-04	0.0013
6	1.8353e-04	9.8560e-04
8	1.7377e-04	0.1484
10	1.5773e-04	6.4684e-04

Table 2.6: Training and Test MSE on ANFIS Model

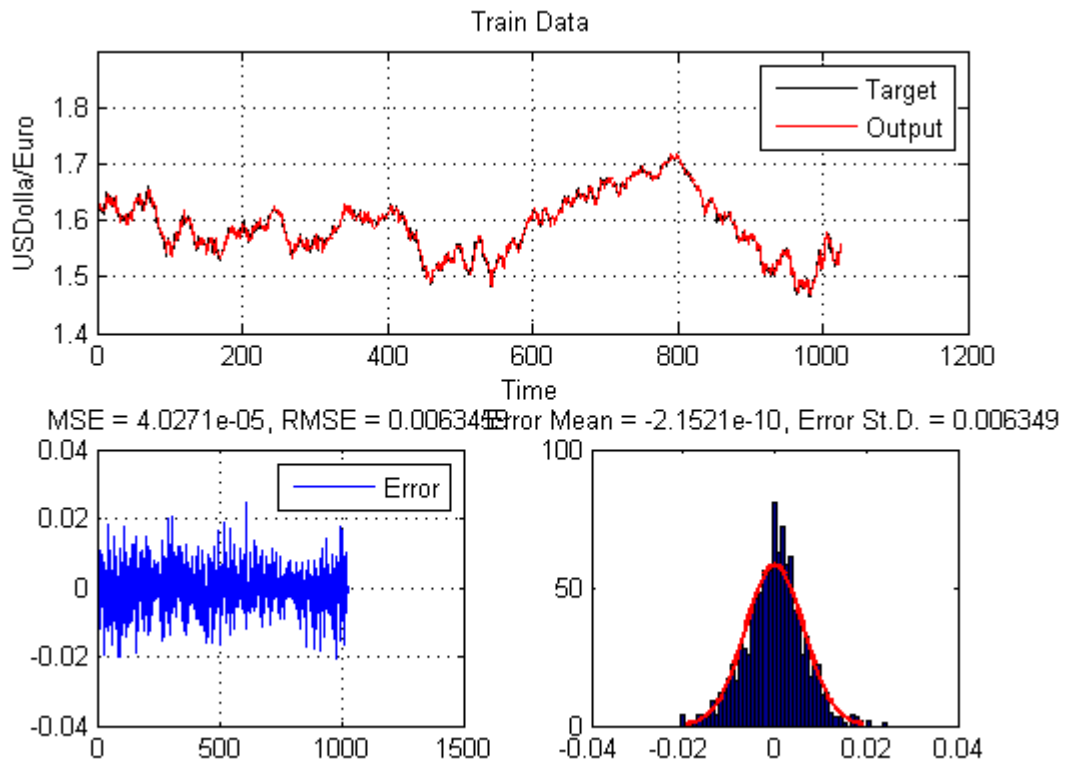


Figure 2.21: Training Data

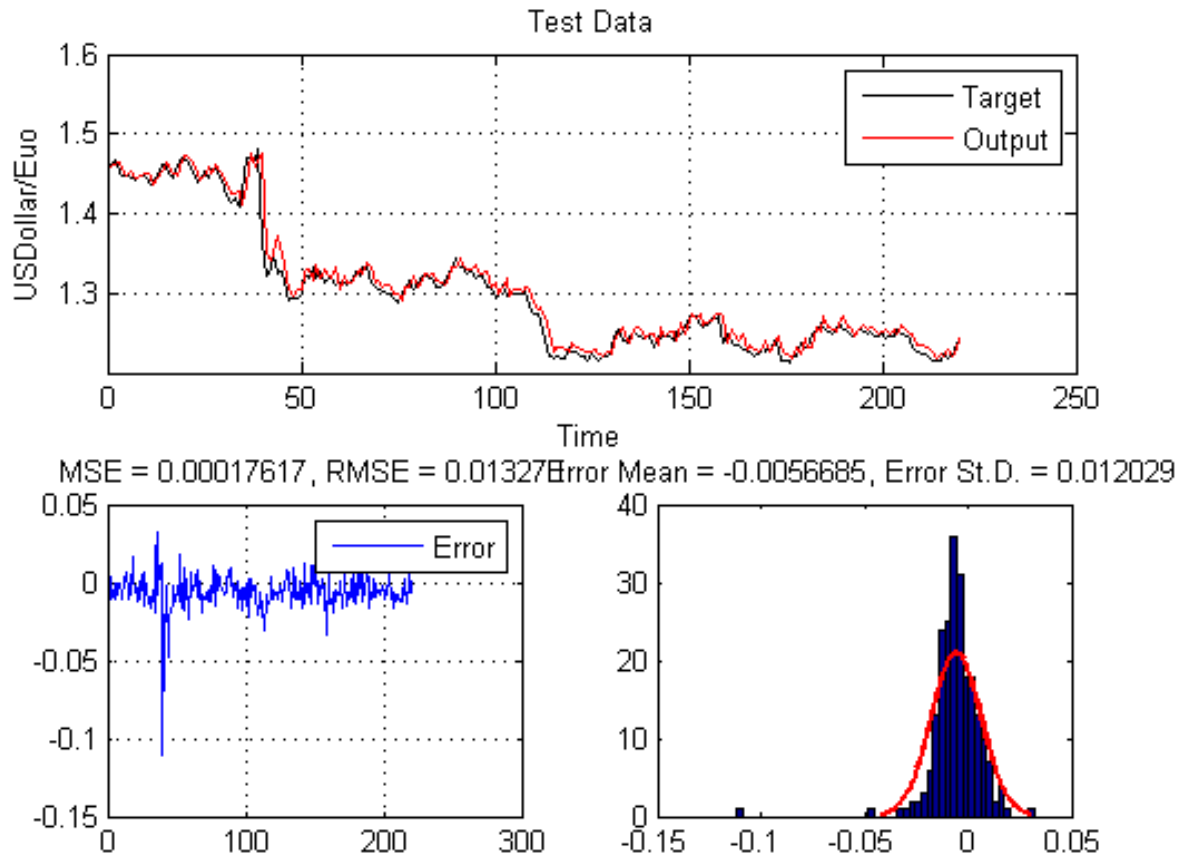


Figure 2.22: Test Data

2.4 Comparison with Random Walk Model

The four models that were studied in detail were Neural Network model, Neural Networks with Genetic Algorithms, Neuro-Fuzzy Models. The test performance was obtained for four of them and these are now compared to the Random Walk Model. As studied in literature, forecasts with time series model fail to beat the random walk model or AR(1).

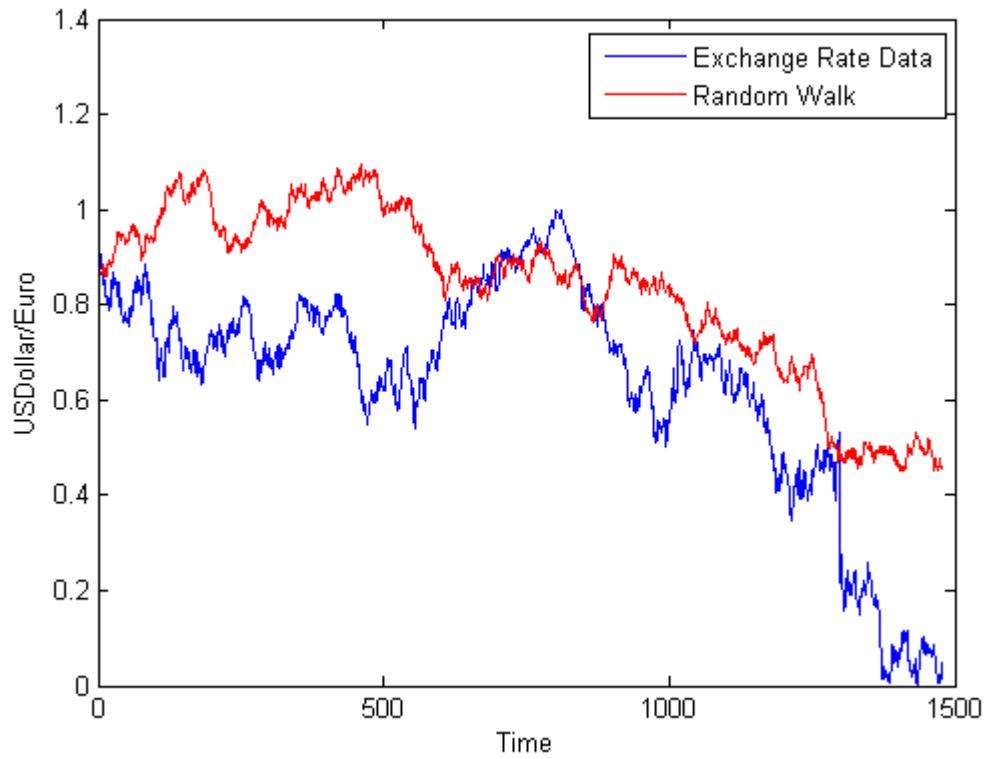


Figure 2.23: Forecasting using Random Walk

Model	Test Performance
Neural Network	6.6587e-04
GA-Weights	5.9878e-04
GA-Architecture	6.1585e-04
ANFIS	6.3072e-04
Random Walk	0.0615

Table 2.7: Test Performance of Different Models

Thus, forecasting using machine learning models yield much better forecasts compared to the random walk model.

Chapter 3

Combination of Forecasts

In most statistical modeling applications, there are several models that are a priori plausible. However, it is a common practice in statistics to select the best model out of those which give the best performance value. This performance can be measured by several statistical quantities like the Mean Square Error, Mean Absolute Error, Root Mean Square Error, etc. However Granger and Bates in a 1969 paper proposed a combination of the model forecasts and showed that it produced much better results. Hoeting et al 1999 also derived the Bayesian Model averaging for a linear model and generalised linear model.

This chapter will look at how to combine all the proposed models for a proposition and see if the forecasting results with the model is better. This is generally done by assigning weights to all plausible models and to work with the resulting weighted estimator. This leads to the class of model averaging estimators. The method that we are going to discuss in the next section is of assigning weights by a model selection criterion, Akaike's information criterion (AIC).

3.1 Information Theoretic Averaging

Let $M = (M_1, M_2, \dots, M_n)$ be the models in consideration that are going to be used for the estimation of a quantity μ . Model averaging by the definition involves finding non-negative weights (w_1, w_2, \dots, w_n) . Then the quantity μ that is calculated will be

$$\mu = \sum w_k \mu_k$$

, where μ_k is the quantity of interest estimated by each of the model. Let I_k denote an information criterion of the form

$$I_k = -2\log L_k + 2s_k$$

, where s_k refers to the number of parameters of the model and L_k defines the likelihood of the model in the AIC criterion. We use and the weight for the model M_k is then defined as

$$w_{AIC,k} = \frac{\exp(-AIC_k/2)}{\sum_{l=1}^n \exp(AIC_l/2)}$$

When the Bayesian information criterion (BIC) is used, we use with

$$s_k = \log(n)q_k$$

where n is the sample size of the data taken and q_k is the number of parameters of the model. The weights are then calculated similarly to the AIC criterion.

$$w_{BIC,k} = \frac{\exp(-BIC_k/2)}{\sum_{l=1}^n \exp(BIC_l/2)}$$

We now try to provide the motivation for the derivation of AIC.

We assume that there is a function f which denotes the true distribution of a quantity. Now, using statistical or machine learning techniques, we find out a model that approximates the given data. We call this model g , which depends on parameters θ . As, defined in the last chapter, the K-L information $I(f, g)$ is the information lost when model g is used to approximate f and is defined as

$$I(f, g) = \int_{-\infty}^{+\infty} f(x) \log\left(\frac{f(x)}{g(x|\theta)}\right) dx$$

As we can clearly infer, the best model loses the least amount of information and deviates the least from the true model. Mathematically, this is equivalent to minimising the function $I(f, g)$. However, calculating the integral requires the knowledge of f and the parameters θ .

Thus, we try to calculate the estimate of K-L information rather than its true value.

$$\begin{aligned}
 I(f, g) &= \int_{-\infty}^{+\infty} f(x) \log\left(\frac{f(x)}{g(x|\theta)}\right) dx \\
 &= \int_{-\infty}^{+\infty} f(x) \log(f(x)) dx - \int_{-\infty}^{+\infty} f(x) \log(g(x|\theta)) dx \\
 &= E_f(\log(f(x))) - E_f(\log(g(x|\theta)))
 \end{aligned}$$

In this equation, $E_f(\log(f(x)))$ becomes a constant if we assume that f is the true distribution. This leaves only $E_f(\log(g(x|\theta)))$ to be calculated.

Akaike now showed a critical relationship between the information-theoretic criterion of K-L divergence and the likelihood of the model. In the above integral, only $E_f(\log(g(x|\theta)))$ needs to be calculated. Assuming, we do not know the true values of θ , this leaves us to calculate

$$E_y E_x[\log(g(x|\hat{\theta}(y)))]$$

, where y is the data that we have. The $\hat{\theta}$ is the maximum likelihood estimator of the parameter θ of the model g . Although only y denotes data, we assume that both x and y are independent random samples from the same distribution. Akaike found an asymptotic result which showed that the maximized log-likelihood value was a biased estimate of $E_y E_x[\log(g(x|\hat{\theta}(y)))]$. This bias was estimated approximately equal to K , the number of estimable parameters in the model g . Mathematically, this result is:

$$\log(L(\hat{\theta}|data))K = E_f(\log(f(x))) - I(f, g(x|\hat{\theta}))$$

, This finding makes it possible to combine estimation (i.e., maximum likelihood or least squares) and model selection under a unified optimization framework.

$$\text{relative } E(K - L) = \log(L(\hat{\theta}|data))K$$

. Akaike multiplied this above equation by a factor of -2. This is statistically known as the Akaike's information criterion:

$$AIC = -2\log(L(\hat{\theta}|data)) + 2K$$

In the special case of least squares estimation for a regression problem, AIC is expressed as

$$AIC = n \log(\sigma^2) + 2K$$

Here, σ^2 is defined as mean square error of the residuals obtained from the fitted model.

$$\sigma^2 = \frac{\sum \epsilon^2}{n}$$

Here, ϵ refers to the residuals obtained from the fitted model and n is the number of data points. AIC for models then becomes very easy to compute when we know the values of K or number of parameters of the model. The BIC value of the model in consideration is calculated similarly with modified value of s_k as defined above.

For a set of models, these values are then calculated. However, since the number of parameters are arbitrary, they can also give large AIC values. To counter this, we calculate a quantity Δ_k for each model M_k in consideration. This quantity is calculated as:

$$\Delta_k = AIC_k - AIC_{min}$$

where AIC_{min} is the minimum AIC of a model in the set of models being considered. The best model then has an $\Delta_k=0$. Also, the larger the value of Δ_k , the worse the model is. Thus, we can remove those models from consideration by a principle known as "Occam's razor", where models with a threshold AIC values were removed from the set of models being considered.

3.2 Averaging Machine Learning Models

Using these AIC values, we calculated the weights to be assigned to each machine learning model for forecasting. Further, we also saw that averaging the model forecasts yielded a better forecast than the individual forecast of each model. The AIC and BIC of each model was calculated using the training performance of the model. Lesser the training error, more likely the model would be closer to the true model. The quantity of interest that was estimated was the test performance or the forecasting power of the combination of models.

Model	Test Performance	AIC	BIC	Δ_{AIC}	Δ_{BIC}
Neural Network	6.6587e-04	-7.981×10^3	-8.6567×10^3	252.589	0
GA-Weights	5.9878e-04	-8.0016×10^3	-3.9669×10^3	232.2266	4.689844×10^3
GA-Architecture	6.1585e-04	-7.8063×10^3	-2.4425×10^3	427.5893	6.2142×10^3
ANFIS	6.3072e-04	-8.2339×10^3	-3.8075×10^3	0	4.849×10^3

Table 3.1: Model Averaging Parameters

Model Averaging	Test Performance
AIC	6.9161e-04
BIC	6.6587e-04

Table 3.2: Test Performance on Model Averaging using Test Performance Values

Model Averaging	Training Performance	Test Performance
AIC	4.0271e-05	1.7617e-04
BIC	5.4935e-05	1.7136e-04

Table 3.3: Test Performance on Model Averaging using actual data

Thus, model averaging showed a better result compared to all the individual models.

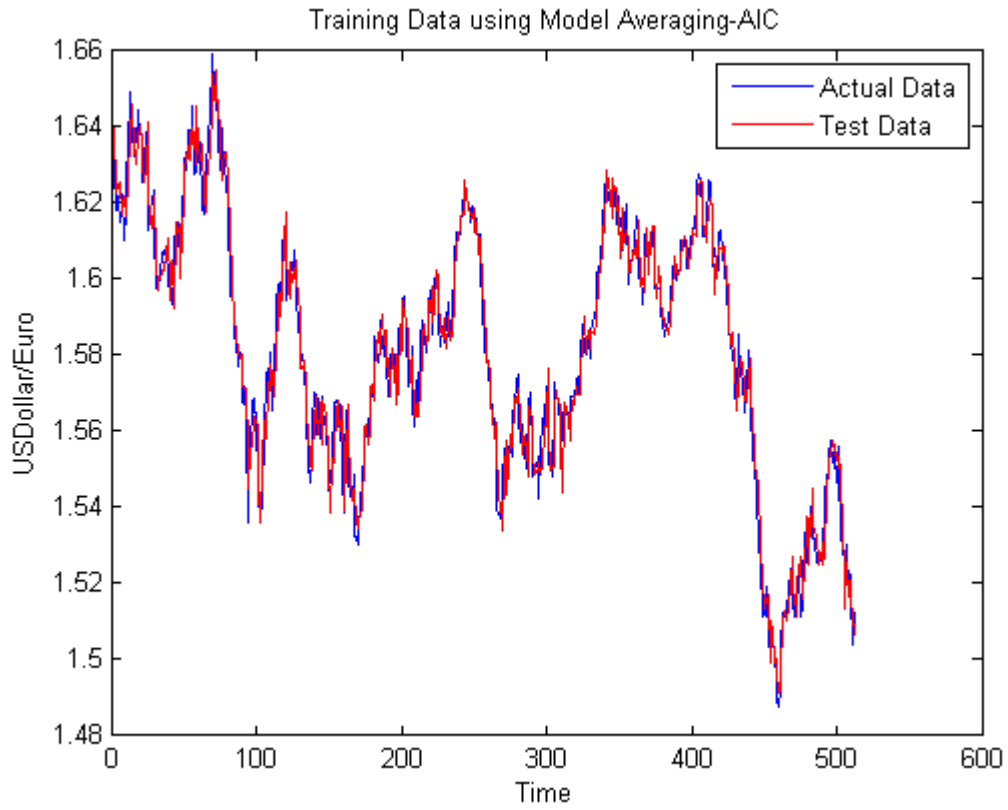


Figure 3.1: Training Data using Model Averaging- AIC

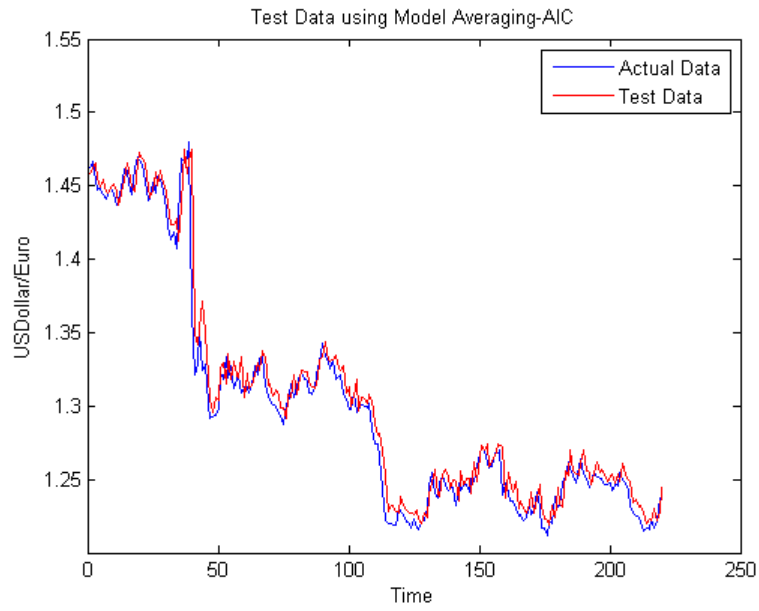


Figure 3.2: Test Data using Model Averaging- AIC

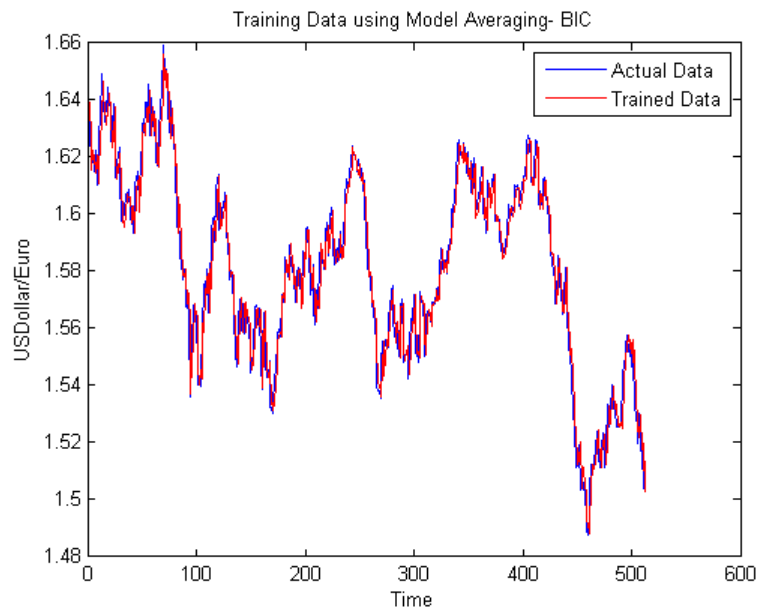


Figure 3.3: Training Data using Model Averaging- BIC

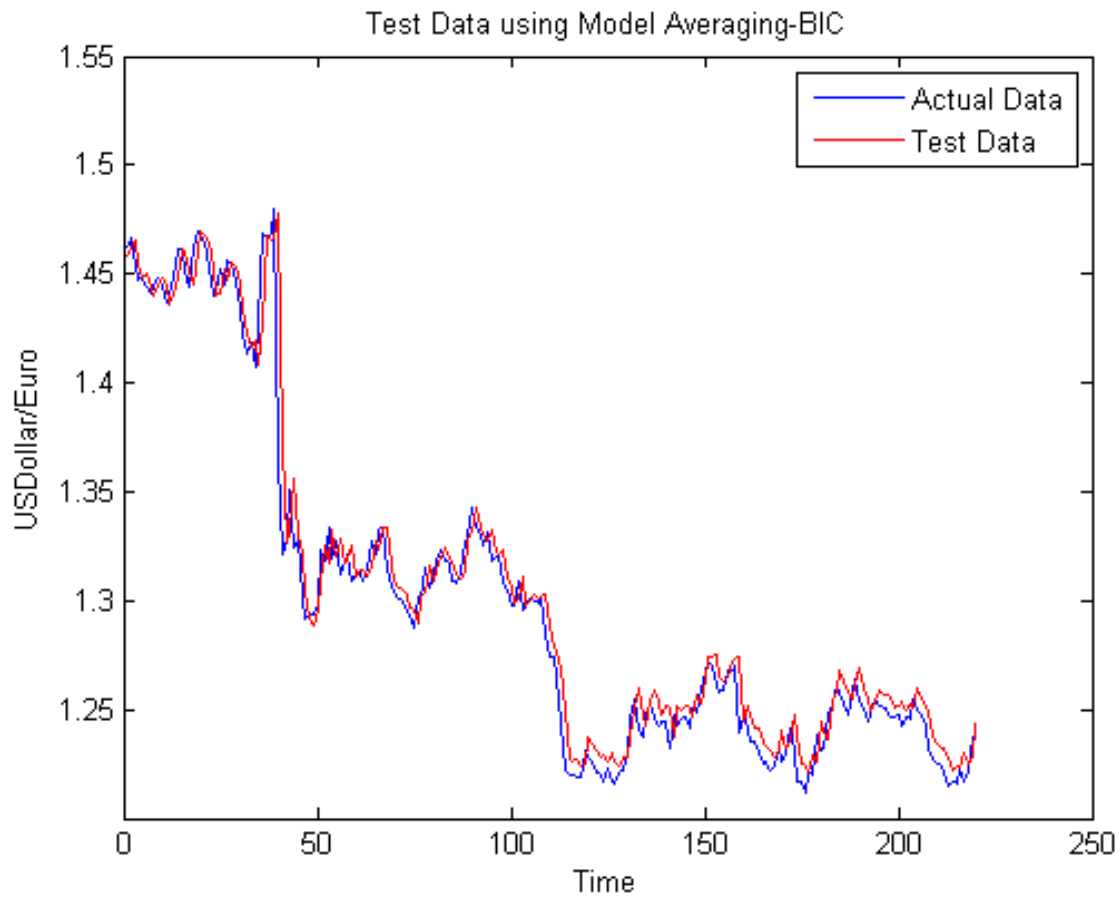


Figure 3.4: Test Data using Model Averaging- BIC

Bibliography

- [1] G. P. Zhang, “Time series forecasting using a hybrid arima and neural network model,” *Neurocomputing*, vol. 50, pp. 159–175, 2003.
- [2] A. K. Nag and A. Mitra, “Forecasting daily foreign exchange rates using genetically optimized neural networks,” *Journal of Forecasting*, vol. 21, no. 7, pp. 501–511, 2002.
- [3] J. Chao, F. Shen, and J. Zhao, “Forecasting exchange rate with deep belief networks,” in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pp. 1259–1266, IEEE, 2011.
- [4] M. Khashei, S. R. Hejazi, and M. Bijari, “A new hybrid artificial neural networks and fuzzy regression model for time series forecasting,” *Fuzzy sets and systems*, vol. 159, no. 7, pp. 769–786, 2008.
- [5] G. S. Atsalakis and K. P. Valavanis, “Forecasting stock market short-term trends using a neuro-fuzzy based methodology,” *Expert Systems with Applications*, vol. 36, no. 7, pp. 10696–10707, 2009.
- [6] S. Haykin and N. Network, “A comprehensive foundation,” *Neural Networks*, vol. 2, no. 2004, p. 41, 2004.
- [7] S. Kumar, *Neural networks: a classroom approach*. Tata McGraw-Hill Education, 2004.
- [8] M. T. Hagan and M. B. Menhaj, “Training feedforward networks with the marquardt algorithm,” *IEEE transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.
- [9] F. D. Foresee and M. T. Hagan, “Gauss-newton approximation to bayesian learning,” in *Neural Networks, 1997., International Conference on*, vol. 3, pp. 1930–1935, IEEE, 1997.
- [10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989.
- [11] L. A. Zadeh, “Fuzzy sets,” *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.
- [12] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

- [13] J.-S. Jang, "Anfis: adaptive-network-based fuzzy inference system," *IEEE transactions on systems, man, and cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.