Nonlinear Time Series Forecasting using Neural Networks and Fuzzy System Techniques

Margarita Papadopoulou
Supervised by Jason Levesley

Dissertation submitted for the MSc in Mathematics with Modern Applications
Department of Mathematics
University of York
UK
"Prediction is very difficult, especially if it's about the future."

Niels Bohr
Acknowledgements

I first of all wish to thank my supervisor Dr. Jason Levesley for his guidance and his invaluable support throughout the project and his lectures in C++ and Fractals and Chaos which were very helpful. I wish also to thank Dr. Zaq Coelho for his help during this MSc course.
Contents

Introduction.............................................................................................................................. 5
Chapter 1: Time Series Analysis .............................................................................................. 6
  1.1 Time Series: An Introduction ...................................................................................... 6
  1.2 Methods and Models .................................................................................................... 9
    1.2.1 Autoregressive Model ......................................................................................... 9
    1.2.2 Moving Average Model ..................................................................................... 10
    1.2.3 Autoregressive Moving Average Model ........................................................... 10
    1.2.4 Autoregressive Integrated Moving Average model ........................................ 10
  1.3 Extending the Linear Cases – Embedding Theorem .................................................. 11
  1.4 Objectives of the Project ............................................................................................ 12
Chapter 2: Neural Networks ................................................................................................. 13
  2.1 The Biological Approach ............................................................................................ 13
  2.2 McCulloch-Pitts Neural Network ............................................................................... 13
  2.3 The Architecture of Neural Networks ........................................................................ 14
  2.4 The activation Function .............................................................................................. 16
  2.5 Training Neural Networks ........................................................................................ 16
    2.5.1 Supervised learning ............................................................................................ 16
      2.5.1.1 Gradient Descent or Delta Rule ................................................................. 17
      2.5.1.2 An Implementation of the Gradient Descent Method .............................. 18
    2.5.2 Unsupervised Learning ....................................................................................... 19
    2.5.3 Reinforcement Learning ..................................................................................... 20
  2.6 Overfitting Problem .................................................................................................... 20
  2.7 Neural Networks and Forecasting .............................................................................. 20
  2.8 Experiment: The Developed Neural Network ............................................................ 22
  2.9 Prediction of Typical Chaotic Time Series using a Neural Network ......................... 26
  2.10 Prediction of Typical Chaotic Series Using an ARMA Model .................................. 32
  2.11 Extending the Experiment: Working with NARMA models .................................... 38
  2.12 Santa Fe Competition and Real World Data ............................................................. 45
Chapter 3: Fuzzy Neural Networks ....................................................................................... 46
  3.1 Fuzzy Logic: Key ideas .............................................................................................. 46
  3.2 Operations on Fuzzy Sets ........................................................................................... 47
  3.3 Fuzzy Inference Systems (FIS) ................................................................................ 48
  3.4 Fuzzy Neural Networks key ideas ............................................................................. 49
  3.5 Experiment: Prediction of Lorenz attractor with Fuzzy Neural Networks .............. 51
Conclusion and Possible Areas for Future Research ......................................................... 54
Bibliography ........................................................................................................................ 55
**Nonlinear Time Series Forecasting using Neural Networks and Fuzzy System Techniques**

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A</td>
<td>Gradient Descent Method</td>
<td>57</td>
</tr>
<tr>
<td>Appendix B</td>
<td>MENFIS Software</td>
<td>64</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Preprocessor</td>
<td>65</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Construction of a Fuzzy Neural Network</td>
<td>71</td>
</tr>
</tbody>
</table>
Introduction

There has been a lot of motivation over the last few decades in predicting the near or the distant future with accuracy. One could wonder the reason for this major interest and why the research concerning the area of forecasting is becoming more and more popular and is becoming increasingly demanding.

Common examples are all the different companies which need to make forecasts for the future for various reasons. More specifically, companies have to predict the future of their sales, stock prices, and based on that to develop their policy and most of the time modifying it during the year. Forecasts have been getting more complex through the years due to high competitiveness and the demand for accurate forecasts. A good forecast will help the company to exploit all the potential opportunities that the environment can offer or protect them from a disastrous choice.

Moreover, different phenomena and the time series which define them are in abundance in the real world. Predictions of global temperature or of the concentration of CO₂ in the atmosphere based on the time series data sets that have been collected through the years have to be accurate, as they are related to the burning issue of climate change.

How easily can all of these predictions be achieved and how accurate can they be?

When the systems to be predicted are governed by deterministic equations, prediction is a trivial issue as it is restricted to the solution of the equations (Weigund & Gershenfeld, 1994). In addition, the systems whose underlying dynamics are linear, can easily be treated with one of the various statistical tools or linear prediction models giving really satisfactory results.

What if these deterministic equations of the system are not known or what if the underlying dynamics of the system do not indicate linear behaviour? Can we make nonlinear generalisations extending the linear case? Real world series are many times more complex as some of them are characterised by rich chaotic behaviour. Kantz and Schreiber (2004) state that “the most direct link between chaos theory and the real world is the analysis of time series from real systems in terms of nonlinear dynamics”.

In this project our target is to give answers to the questions which are set above. We start with the key ideas of time series and linear modelling and we then move to nonlinear cases when even chaos is intermingled. We want to show how we can model and predict these systems focusing mainly with the use of a very power nonlinear tool, neural networks, while then moving to the rule-based fuzzy logic and to fuzzy neural networks. Beyond the construction of a neural network and of a fuzzy neural network, for the purpose of this project, we extend the linear prediction models in order to have at the end a tangible comparison.
Chapter 1: Time Series Analysis

1.1 Time Series: An Introduction

Time series \( X = \{X_1, X_2, \ldots, X_n\} \) is an ordered sequence of values of a variable at equally spaced time intervals. Time Series Analysis is used in many applications such as Process and Quality Control, Economic Forecasting, Sales Forecasting, Physical sciences etc. For example, the closing prices of the Dow Jones index, the number of earthquakes per year magnitude 5.0 or greater or the monthly sales of a company would comprise time series.

Two main targets of time series analysis are to extract information for the phenomenon represented by the sequence of observations and predict the future values of the time series variables. There are two main approaches that someone can treat time series with. The first one is the time domain approach and the second one is the frequency domain approach. One may possibly be interested in constructing or developing mathematical models or statistical methods in order to predict one of the infinite sets of time series data that is available. Which of the two approaches someone will finally choose to analyse the data with depends both on the dynamics that the time series dataset hides, and on the interest of the researcher, i.e. what the exact information that they are interested most in and wish to extract from the time series.

**Time domain approach**

The target of the time domain approach is to model the future values with the assumption that there is a correlation-dependence between them and the past values. Common tools in this direction are auto-correlation and cross-correlation analysis.

Auto-correlation is measured between a series and is basically a correlation of a signal with itself (Parr & Phillips, 1999). It is a very important measurement as it helps to identify the repeated patterns in a signal. Cross-correlation is measured between two series and represents the degree that these series are correlated and is used in many cases for pattern recognition as well. Given here is the cross correlation equation, as the auto-correlation is usually considered as the continuous cross correlation of the signal with itself. So for two time series \( y_t \) and \( x_t \),

\[
\rho_{xy}(s,t) = \frac{\gamma_{xy}(s,t)}{\sqrt{\gamma_x(s,s)\gamma_y(t,t)}}
\]

where \( \gamma_{xy} \) denotes the cross-covariance function which equals to:

\[
\gamma_{xy}(s,t) = E[(x_s - \mu_x)(y_t - \mu_y)]
\]

where \( \mu_x \) the mean of the initial series and \( \mu_y = E(y_t) \) is the mean of the new series.

**Frequency Domain Approach**

On the other hand, the frequency domain approach focuses its main interest on the periodic variations that characterise many data. This approach utilizes spectral analysis which is associated with the French mathematician Joseph Fourier and his transformation, Fourier transforms. A Fourier transformation maps a one-to-one relationship between a signal at certain times and how certain frequencies contribute to the signal, to how the phases of oscillations are related to the phases of other oscillations.

The Fourier transform of a function \( s(t) \) is defined as:

\[
\tilde{s}(f) = \frac{1}{2\pi} \int_{-\infty}^{\infty} s(t)e^{2\pi ift} dt
\]
Where in a discrete time series it is given by (Kantz & Schreiber, 2004):

\[ \tilde{S}_k = \frac{1}{\sqrt{N}} \sum_{n=1}^{N} s_n e^{j2\pi kn/N} \]  

(1-4)

Where \( s_n \) a measurement denotes the state at time \( n \). So, the direct Fourier transform basically describes the distribution of the signal along frequency.

The frequencies here are \( f_k = \frac{k}{N\Delta t} \) where \( k = \frac{N}{2}, \ldots, \frac{N}{2} \) and \( \Delta t \) the sampling interval. With the inverse Fourier transform the signal is represented as a summation of the frequencies.

In signal processing though, the main interest focuses on the power spectrum which is the square amplitude of the Fourier transform given by:

\[ S(f) = |\tilde{S}(f)|^2 \]  

(1-5)

The frequency domain approach can be proved really convenient as it is characterised for the simple interpretation of the spectrum that they generate and for the transformation of differential equations to algebraic equations which in many cases simplifies the problem (Berthold & Hand, 1999).

**Wiener-Khinchin theorem**

It is important here to mention the Wiener-Khinchin theorem which links the power spectrum to the autocorrelation function. Their theorem states that spectral density equals the Fourier transform of the auto-correlation function (Kantz & Schreiber, 2004).

The implementation of their theorem as given by Zbilut & Marwan (2008) is that for a deterministic, discrete signal \( x(i) \) the power spectrum is:

\[ S_x(f) = \frac{1}{N} \left| \sum_{i=0}^{N-1} x(i) e^{-j2\pi f i} \right|^2 \]  

(1-6)

which is the square amplitude of the Fourier transform as described above. Applying the Wiener-Khinchin theorem and considering an auto-correlation function \( C_x \) the power spectrum equals now the Fourier transform of \( C_x \) which is:

\[ S_x(f) = \sum_{\tau=-\infty}^{\infty} C_x(\tau) e^{-j2\pi f \tau} \]  

(1-7)

where the auto-correlation function \( C_x \) is defined as:

\[ C_x(\tau) = \frac{1}{N} \sum_{i=0}^{N-1} x(i)x^*(i+\tau) \]  

(1-8)

**Stationary Time Series**

Stationary time series are those time series where their statistical measurements like it’s mean and variance are constant over the time which also means that the behavior of the time series in time \( t \) is identical to its behavior at time \( t + h \). Obviously, the prediction of stationary time series is an easy task, however most time series are far from stationary.
**Decomposition of time series**

An important technique which sometimes makes easier and more efficient the forecast of the time series is the decomposition of the time series, which separates it into individual components such as trend and seasonality.

Trend is described as a long term movement and can be a linear or nonlinear component. When the time series data contains considerable error, the identification of the trend is achieved by the process of smoothing. The most popular technique for this is moving average smoothing which replaces each element of the series by the weighted average of \( n \) surrounding elements.

Seasonality is a systematic effect that describes effects that are considerably stable with respect to time or direction. Some examples are the increase of sales during the Christmas period or the increase of water consumption during the summer. It can be examined by correlograms which are plots of the sample autocorrelations versus the time lags.

There are several decomposition models. The simplest one is the *additive model* where the time series can be simple written as:

\[
y_i = t_i + s_i + e_i
\]

(1-9)

where \( t_i \) are the trends, \( s_i \) are the seasonal and \( e_i \) are the error components.

Another common model for which actually stems from the additive model is the multiplicative model where \( y_i \) is decomposed as

\[
y_i = t_i * s_i * e_i
\]

(1-10)

Thus, taking the logarithms of the model above we detect that it is again an additive model in log scale.

Modern techniques in time series analysis do not use decomposition techniques so regularly and instead use detrending. Detrending is a mathematical process of removing the trend of a time series as trend can produce false covariance among the variables.

**White noise**

Observing different examples of time series datasets, illustrated in Figure 1.1.1, it is easy to detect that different time series have different degrees of smoothness. This is because of the assumption made earlier that a state \( x_t \) at time \( t \) depends on the previous values \( x_{t-1}, x_{t-2}, \ldots \), which means that there is a correlation in time.

![Figure 1.1.1](image-url)

*Figure 1.1.1* – Three different time series datasets. From the left: plot of average air temperatures at Nottingham Castle in degrees Fahrenheit for 20 years; plot of monthly totals of international airline passengers, 1949-1960; plot of the hormone in blood at 10 minute intervals from a human female, 48 samples.

Because of the observation above, in an attempt to make time series appear more realistic, ways of producing time series using uncorrelated random variables were developed. The most popular example of time series generated by uncorrelated random variables is white noise.
White noise is characterized by constant power at all frequencies and it is the most popular noise model. It takes values between 0 and $2\pi$, and each value at any frequency is completely irrelevant to the phase of any other frequency.

A popular white noise is the Gaussian white noise with zero mean and variance $\sigma^2_w$, though it is restricted to a more limited range of frequencies.

1.2 Methods and Models

As mentioned earlier in this chapter, when working with time series one is always trying to understand the underlying dynamics that the time series is characterised from, with the main target being to build appropriate models for the prediction of the time series.

Popular methods for modelling linear time series are the Autoregressive (AR) and the Moving Average models.

1.2.1 Autoregressive Model

The general idea of an Autoregressive model is that a present value can be forecasted with respect to previous $p$ values where $p$ indicates the number of steps that someone has to go back to the past in order to predict the current value. The Autoregressive model of order $p$, $AR(p)$, is given by the form:

$$x_t = \mu + \varphi_1 x_{t-1} + \varphi_2 x_{t-2} + \ldots + \varphi_p x_{t-p} + \epsilon_t$$  \hspace{1cm} (1-11)

Where $\varphi_1, \varphi_2, \ldots, \varphi_p$ are constants, $\mu$ is the mean of the series and $\epsilon_t$ is white noise with zero mean and variance $\sigma^2$.

Equation (1-11) is easily explained if someone starts from the first order Autoregressive model AR(1) and iterate backwards k times. So AR(1) can be written as:

$$x_t = \varphi_1 x_{t-1} + \epsilon_t$$  \hspace{1cm} (1-12)

But,

$$x_{t-1} = \varphi_1 x_{t-2} + \epsilon_{t-1}$$  \hspace{1cm} (1-13)

So substituting (1-13) in (1-12), (1-12) becomes:

$$x_t = \varphi_1^2 x_{t-2} + \varphi_1 \epsilon_{t-1} + \epsilon_1$$  \hspace{1cm} (1-14)

Finally, continuing the same process

$$x_t = \varphi_1 x_{t-1} + \epsilon_t$$
$$= \varphi_1^2 x_{t-2} + \varphi_1 \epsilon_{t-1} + \epsilon_1$$
$$= \ldots$$
$$= \varphi_1^{p-1} x_{t-p+1} + \varphi_1^{p-2} \epsilon_{t-p+2} + \ldots + \varphi_1^2 \epsilon_{t-3} + \varphi_1 \epsilon_{t-2} + \epsilon_1$$  \hspace{1cm} (1-15)

In order to use an AR model for prediction it is essential to specify $p$ which indicates the number of terms that will be used for the model. So, the equation (1-11) represents a linear function of the current value $x_t$ with respect of the past values.

In autoregressive models the prediction of the current value does not depend obviously from the future. In such cases these approaches will be called causal (Schumway & Stoffer, 2000).
1.2.2 Moving Average Model

The moving average model is one of the most popular and useful tools for identifying the trend of the time series. A Moving Average model of order $q$, $MA(q)$, calculates each term of the time series from the moving average of the last $i$ terms of the error sequence and is defined as:

$$X_t = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$  \hspace{1cm} (1-16)

where $\mu$ is the mean of the series, $\theta_1, \theta_2, \ldots, \theta_q$ are the parameters of the model and $\varepsilon_t$ is the white noise. So, the Moving Average model is again a linear function, which makes the prediction of the current value based on past errors, it is also worth mentioning that it is a stationary process too.

1.2.3 Autoregressive Moving Average Model

The combination of the two models above, gives the Autoregressive Moving Average model $ARMA(p, q)$. It is a general form of models that is generated from a mixture of AR and MA models. For example an AR model of order (3) can be written as $ARMA(3,0)$ or a MA model of order (2) can be written as $ARMA(0,2)$. The general form of the general model is $ARMA(p,q)$ which means an AR model of order $p$ with a moving average error of order $q$ and is defined as:

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$  \hspace{1cm} (1-17)

So obviously the current value in an $ARMA$ model is a combination of the past values and the past errors. This model basically introduced by Box & Jenkins (1976) as a tool for forecasting time series. They assumed that the data is stationary which means that the mean and variance do not change over time and they considered three iterative stages for their model:

i. Model identification
ii. Parameter Estimation
iii. Model Validation

1.2.4 Autoregressive Integrated Moving Average Model

As mentioned above in order to use an $ARMA$ model, it is essential to ensure stationarity. If the time series are not stationary one can change it by differencing them. Box & Jenkins (1976) also introduced a more complex technique that is used for time series forecasting and is a generalisation of the $ARMA$ model, the Autoregressive Integrated Moving Average model. $ARIMA$ models uses differenced time series. Kihoro et al. (2004) state that ‘arima model order involves matching the patterns in the sample autocorrelation functions (ACF) and sample partial autocorrelation functions (PACF) with the theoretical patterns of the known models to identify the orders’. The crucial difference is that the $ARMA$ model is quite successful when the time series present stationary behaviour but it fails in non-stationary cases while $ARIMA$ model is very effective and flexible among non-stationary and continuous data. If the time series $Y$, has a polynomial trend of order $d$, then the trend can be removed by differencing $Y$, $d$ times (Falk, Mahron, Michael, & Macke, 2006).
The autoregressive Integrated Moving Average Model of order \((p, d, q)\) where

- \(p\) is the order of the autoregressive part
- \(d\) is the order of differencing needed to stationarise the data
- \(q\) the order of the moving average part

is defined as:

\[
Y_t = a_1 Y_{t-1} + a_2 Y_{t-2} + \ldots + \beta_1 Z_{t-1} + \beta_2 Z_{t-2} + \ldots + \beta_p Z_{t-q} + Z_t
\]

(1-18)

where:

\[
Y_t = \nabla^d X_t
\]

(1-19)

### 1.3 Extending the Linear Cases – Embedding Theorem

All the models described above are really important for the prediction of a time series. When the main target is prediction one might think of several cases for choosing the most appropriate tools. For example a linear regression on a sample gives a straight line through the sample by the method of least squares. As long as the trend continues to be linear, the linear regression is going to give a good prediction. But what if this trend is not linear? What happens for example when the rising sales of a company change and there is a noticeable drop in sales? In such cases the linear model will give fault results.

One way to encounter cases like this, is not to rely on the entire sample, but in successive windows of the sample, e.g. windows of 20 days. So a moving average of 20 days can be constructed. This moving average will be the mean and the sample will vacillate around this. As long as the seasonality over the series is constant and the cyclical fluctuations stable, all the models above are appropriate. Unfortunately though, in many problems we have varying cyclical components and all the methods above are not useful at all.

When we have to deal with more complex cases of time series, we treat them differently as the relationship among the data is not linear. As we mentioned earlier, time series are observations made at spaced time. If we consider a non-linear dynamical system that is defined by some difference equations containing an initial condition, the system is then defined as \(s_t = f^t(s_0)\) where \(s_t\) is the position vector, \(s_0\) the initial condition and \(f^t : R^d \rightarrow R^d\). A time series can be considered as being generated from a dynamical system as a projection \(x_t\) of every point \(s_t\) of the trajectory of the system such that \(x_t = h(s_t)\) where the projection function \(h : R^d \rightarrow R\) is called an observation function.

But given a time series of a single state variable \(x_i(t)\) in a \(d\) dimensional dynamical system it is difficult to make observations like: which is the dimension of the state space that the trajectories generated by the system are lying; or, observations for the observation \(f\) function \(h\). One way to do this, is by creating a projection of \(R^d\) and then going back to the initial state space \(R^d\). The most efficient way though, is to construct a reflection of the initial attractor.

Takens theorem allows us under certain conditions to reconstruct the state space and have a new one of dimension \(m\) in which the reconstructed attractor preserves all the topological properties of the of the original attractor. So there is an embedding of the original state space to the new one. He states that the essential condition for the new dimension \(m\) is: \(m \geq 2D + 1\) where \(D\) is the fractal dimension of the attractor. The dimension \(m\) of the new state space has to be big enough to be able to unfold the time series otherwise the initial attractor presents intersections and is not topologically equivalent to the new one.
1.4 Objectives of the Project
This project will focus mainly in cases that the linear models fail to learn the dynamics of a time series and hence are inappropriate for their prediction.

Our target is to handle nonlinear time series, finding appropriate tools that offer a good prediction even in cases that time series have rich chaotic behavior. The main tools used are neural networks which are constructed for the purposes of this project. A lot of different software has been used, and many different models were studied in order to prove the superiority of the neural network in comparison to the linear models. Wanting to go a bit further with neural network analysis, we have also introduced the main principles of fuzzy logic and how these can be applied on the neural network giving a new form called fuzzy neural networks.
Chapter 2: Neural Networks

2.1 The Biological Approach

The traditional definition of the term neural network refers to the biological neurons that the human brain is composed of, which are densely interconnected. Many living creatures including human beings have the ability of adjusting themselves in environments that regularly change. They achieve that by a control system which has the ability to learn. This system for human beings is the brain, which is constituted of billions of interconnected neurons.

Each neuron consists of the cell body (soma), the axon which is usually unique for each neuron and the dendrites (Figure 2.1.1). Dendrites receive the input signals which are then summed and this input is processed through the cell body. The axon’s target is the transmission of the electro-chemical signal to the other neurons. At the very end of the axon, there are the terminal buttons which basically convert the electro-chemical signal to chemical information before that passes to the other neurons. Different neurons are connected to each other by a structure called a synapse which is very crucial as it is the part that the chemical signal passes from one neuron to another. Each synapse, depending on the chemical signal that it transfers, is able to strengthen the signal or weaken it.

In order to explain how accurate and complicated all of the brain system that is described above is, Haykin (1998, pp. 23-24) states that ‘the brain is a highly, complex, nonlinear and parallel computer’.

2.2 McCulloch-Pitts Neural Network

The first model of an artificial neural network that was completely inspired by the structure of the biological neuron, was given by an American neurophysiologist, Warren McCulloch, and an American logician, Walter Pitts. Their model was based on the “all-or-none” law which supports that the response of a neuron to a stimulus is independent to the stimulus strength but there exists a threshold of activation which if the stimulus strength exceed, the neuron is activated otherwise is not. Thus, they suggest that their neural network, consists of input and output channels. Some of these input channels act as stimulus to the excitation of the neuron while others restrain this process. The threshold then, is estimated by the sum of the ‘excitatory’ inputs (McCulloch & Pitts, 1943). Thus, inputs are considered as binary states with 1 represent the input channels that cause excitation to the neuron and 0 the ‘silent’ ones. This idea is based on mathematical logic operations and Boolean operators. So if for example the neural network has two inputs one active and one silent represented by [10], the output after the summation will be [1] and the neuron will be activated (Figure 2.2.1).
Thus, the output of their model is given by:

\[ y = \theta(\sum_{i=1}^{n} w_i(t)x_i - s) \]  

(2.1)

Where \( w_i \) are the synaptic weights which represent how strong the connection between two neurons are and \( x_i \) represents the neuron \( i \) at time \( t \), \( \theta(x)=1 \) if \( x \geq 0 \) and \( \theta(x)=0 \) if \( x < 0 \).

Even though this approach was really innovative when it was introduced by McCulloch & Pitts, it had two main disadvantages (Ritter, Martinetz, & Schulten, 1992). The first one is that they do not refer clearly, how the connections between the neurons are set or the possibility of training the neurons which in its simple form means to set the weight-strength for each connection of the neurons. In addition, biological neural networks present fault-tolerance which means that in a possible error or damage of a part of the nervous system, there is not a destruction of the human brain or a loss of a life and the system continues its function even with less efficiency. Thus, the inability of McCulloch-Pitts model to exhibit error-tolerance makes it less efficient.

Improving the neural network described above, a psychologist, Donald Hebb (1949) introduced a ‘learning’ rule for the synaptic weights of the neurons. More specifically, he considered two active neurons \( N_1 \) and \( N_2 \) where the synaptic weights between the nodes can be adjusted i.e. where \( N_1 \) participates in excitation of \( N_2 \), the connection strength will be increased. So, for two neurons \( i \) and \( j \) which are active simultaneously and \( i \) receives input from \( j \) then if \( w_{ij} \) is the synaptic weight between the two neurons the simplest form of the Hebbian rule is given from the following relationship:

\[ \Delta w_{ij} = \eta y_i x_j \]  

(2.2)

Where \( x_j \) represents the prosynaptic signal and \( y_i \) the postsynaptic signal and \( \eta \) a positive constant that determines the learning rate.

McCulloch-Pitts and Hebb’s approaches were really important as much research on the neural network’s function followed, which greatly increased today’s knowledge on them.

2.3 The Architecture of Neural Networks

Neural networks are divided into two kinds based on their structure, feedforward and recurrent neural networks.

Feedforward neural networks as their name imply is a loop free form which means that its input layer is mapped to the output layers but not vice versa (Haykin, 1998) i.e. they are unidirectional, they only move forward. They are the most simple forms of neural networks and they are divided two types, single layer feedforward neural networks and multilayer feedforward neural networks. Single layer feed forward neural networks just have input and output layers (Figure 2.3.1).
The difference of multilayer neural networks which are also usually called multilayer perceptrons (Figure 2.3.1) is that apart from input and output layers, they also have one or more hidden layers which are layers in the centre of the network and can learn to recode all the inputs. It is important at this point to mention that all the inputs units are connected to every single unit of the first hidden layer and every unit of the output is connected with all the units of the last hidden layer. In case that any of these connections are missing, the neural network is called partially connected.

Recurrent neural networks also constitute of an input, an output and possible hidden layers with the crucial difference from the feedforward neural networks, that they include cyclic components which means that this output is fed back to the input (Figure 2.3.2).

Although feedforward neural networks are widely used in signal processing problems, they are restrictive in completing some tasks as their main function is simple mapping transformations and signal classification. On the contrary, the fact that feedback neural networks such as recurrent networks offer more parallels, gave rise to their study especially in cases of parallel, complex problems that can be found in dynamic phenomena characterised by nonlineairties and chaotic behaviours (Subhash, 1992). Their more realistic structure and the added complexity that they offer can better explained by thinking the relation among feedback neural networks such as recurrent which have a global feedback and biological systems such as the human nervous system where feedback always occur.
2.4 The activation Function

The activation or squashing function Figure 2.4.1 is a function that basically restricts the output to permitted levels. Activation functions are really crucial as there are those that introduce nonlinearity to the systems and make the hidden neurons powerful (Warren, 2010). In practice there are many different activation functions. Common activation functions are the threshold function which is basically the form that McCulloch and Pitts introduced, Gaussian functions, sigmoid or more generally logistic functions which are widely used.

The form of the sigmoid activation function which is by far the most popular is given by Haykin (1998) and is given by the following:

\[
\varphi(v) = \frac{1}{1 + \exp(-av)}
\]  
(2-3)

where \( a \) is slope parameter of the function.

2.5 Training Neural Networks

Neural networks have a bidirectional relationship with their environment. In the neural networks function, the synaptic weights between the neurons are set randomly. Thus, there are cases that the performance of the neural network is not the best possible, and the prediction error is not small enough. In these cases it would be advisable to change the synaptic weights and provide a new input vector to it, this process is known as training.

Dudul (2005) describes training data as inputs that correspond to desired outputs. He suggests that the training data could be described by the relationship:

\[
Z^N = \{[u(t), y(t)] | t = 1, ..., N\}
\]  
(2-4)

where \( u(t) \) is the input, \( y(t) \) is the desired output and \( N \) the number of samples in training data.

There are many ways that the neural networks can be ‘trained’ in order to improve their performance. These ways are divided into three categories: The supervised, the unsupervised and the reinforcement training method. The main target of these methods is to minimise the prediction error.

2.5.1 Supervised learning

Supervised learning could be attained either by an external ‘teacher’ – the programmer of the neural network- or by the network itself (self supervised learning). In supervised learning, the programmer has desired outputs where the inputs are mapped to.

One of the most popular supervised learning techniques that is used for the training of multilayer feedforward neural networks is backpropagation. A training rule which forms a crucial step for the backpropagation learning method, is the gradient descent or delta rule.
2.5.1.1 Gradient Descent or Delta Rule

Gradient descent is an optimization method which is able to find the best possible set of weights for the training data (Mitchell, 1997) with the main target to minimise the error function. As a tool for supervised learning it enables the ‘teacher’ to search for the local minimum of the error surface.

The error function is defined as \( f(w) \) with \( n \) variables where \( \vec{w} = (w_1, w_2, \ldots, w_n) \) and its partial derivative exists. The target of the gradient descent method is to minimise the error function by setting an arbitrary weight vector in the beginning of the process and then change the weight vector at each iteration of the process until a global minimum is achieved (Mitchell, 1997).

So, the gradient of \( f(\vec{w}) \) is denoted as:

\[
\nabla f(\vec{w}) = \left[ \frac{\partial f(\vec{w})}{\partial w_1}, \frac{\partial f(\vec{w})}{\partial w_2}, \ldots, \frac{\partial f(\vec{w})}{\partial w_n} \right]
\]

(2-5)

And it points locally in the direction of the steepest increase in \( f(\vec{w}) \). Obviously, \(-\nabla f(\vec{w})\) points locally in the direction of the steepest decrease.

Mitchell (1997), also suggests that an efficient way to grasp the idea behind the gradient descent method, is to think of the task of training an ‘unthreshold perceptron’, which is a linear unit with an output given by

\[
o(x) = \vec{w} \cdot x
\]

(2-6)

For the implementation of the gradient descent method, it is essential to choose an arbitrary error function. According to Mitchell (1997) the most popular form of the error function is:

\[
f(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2
\]

(2-7)

where \( D \) is the set of training data, \( t_d \) the target output of the training example \( d \) and \( o_d \) the output of the linear unit. Then, the training rule for the gradient descent is:

\[
\vec{w}_i \rightarrow \vec{w}_i + \Delta \vec{w}_i
\]

(2-8)

Where:

\[
\Delta \vec{w}_i = -\eta \frac{\partial f}{\partial \vec{w}_i}
\]

(2-9)

and \( \eta \) is a positive constant chosen between 0 and 1 known as the learning rate. The combination of the equations (2-5) and (2-7), leads to the calculation of \( \frac{\partial f}{\partial \vec{w}_i} \). So,

\[
\frac{\partial f}{\partial \vec{w}_i} = \frac{\partial}{\partial \vec{w}_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial \vec{w}_i} (t_d - o_d)^2
\]

\[
= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial \vec{w}_i} (t_d - o_d) = \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial \vec{w}_i} (t_d - \vec{w}_i \cdot x_d)
\]

\[
= \sum_{d \in D} (t_d - o_d) (-x_d) = \frac{\partial f}{\partial \vec{w}_i}
\]

(2-10)
Substituting equation (2-10) in (2-9), finally yields to:
\[ \Delta w_i = \eta \sum_{d \in D} (t_d - o_d)x_{id} \]

(2-11)

Even though the gradient descent method is quite simple, and definitely the target is trying to move to the direction that \( f \) decreases most quickly, there are always doubts as to how big the step should be to this direction. Apart from this, such algorithms are possible to find many local minimum, which is undesired as it will not be an absolute convergence. Thus, a good algorithm which is based on the gradient descent method should ensure that it will choose the best step length and direction in order to achieve the global minimum. Many algorithms have been written to this end either with less or more efficiency in achieving the absolute convergence (Bengio, Simard, & Frasconi, 1994; Kivinen & Warmuth, 1997; Pendharkar, 2007).

2.5.1.2 An Implementation of the Gradient Descent Method
A successful approach that is used again later in this project, is given by Ma & Wei (1999) in their paper “Research on Nonlinear Models of Time Series”. Ma & Wei basically tried to find an appropriate model in cases that time series present nonlinear characteristics or have dynamic behaviour where the linear tools proved unhelpful.

Working in this direction and after testing many different models that should fit to time series with nonlinearities, they concluded with a nonlinear ARMA (NARMA) model which is based on a polynomial and is given by the equation:

\[ \hat{x}_n = f(x_{n-1}, x_{n-2}, \ldots, x_{n-p}, e_{n-1}, e_{n-2}, e_{n-q}) + \sum_{k=1}^{q_1} c_k x_{n-k} + \sum_{k=1}^{q_2} d_k e_{n-k} \]

(2-12)

Finding a possible appropriate model for the prediction of the non-linear time series was their first step, though another crucial step in time series analysis is finding an efficient training method which will learn the data in order to fit correctly to the model suggested above. They then suggested a learning algorithm based on the gradient descent method, but being aware of the risk that the learning speed of this method can be slow and the possible local minimum problem, they paid particular attention to the choice of the direction and the steps length at each iteration.

Their main idea is based on the general gradient descent principles. So, the observation that if the real-valued function \( F(x) \) is defined and differentiable in a neighbourhood of a point \( a \), then \( F(x) \) decreases fastest if one goes from \( a \) in the direction of the negative gradient of \( F \) at \( a \), \(-\nabla F(a)\).

Their algorithm consists of eight steps which are the following:

1. Choose initial weighting parameters \( w_0 \) and step length \( \lambda \)
2. Calculate the gradient vector \( \nabla w_0 \) of the target function at \( w_0 \) and take \( S_0 = -\nabla w_0 \)
3. Start from \( w_0 \), move along direction \( S_0 \) by \( \lambda \) to \( w_1 \), calculate \( \nabla w_1 \) and take \( S_1 = -\nabla w_1 \)
4. Let \( M = 0 \) and choose \( K \). Calculate the target function \( J(w_1) \) at \( w_1 \) and suppose the searching target \( TAR = J(w_1) \)
5. Let \( M = M + K \), \( S = S_0 + M \times S \). Move along \( S \) by \( \lambda \) from \( w_1 \) to \( w_2 \)
6. If \( J(w_j) < J(w_i) \) then \( w_i = w_j, K = 2K \) and return to step 5, else if \( J(w_j) > J(w_i) \) and \( |K| < \xi_1 \), which is a threshold value, or \( |M| > \xi_2 \), which is another threshold value, then go to step 7, otherwise \( K = -\frac{K}{3} \) and return to step 5.

7. Suppose \( D = (TAR - J(w_i))/J(w_i) \). If \( D > \xi_3 \), which is the third threshold value, then \( S_i = -\nabla w_i \) and return to step 4, else if \( D < \xi_3 \), then go to step 8.

8. If \( J(w_j) < J(w_2) \) then \( w_0 = w_1 \) and return to step 2, else if \( \lambda > \xi_4 \) which is the fourth threshold value, then \( \lambda = \frac{\lambda}{2} \), else if \( \lambda < \xi_4 \) then stop calculating.

In this project, a variation of the gradient descent method was implemented, which is based on Ma Ni and Wei Gang’s algorithm above. Two different functions have been used to test the search of minima from various starting points (Figure 2.5.1). The results are identical as expected and the iterations number low proving the efficiency of Ma Ni’s and Wei Gang’s approach.

![Figure 2.5.1 – Function \( F(x, y) = (x^2 - y)^2 + (1 - x)^2 + (1 - y)^2 \), \( x_0 = 0.55 \), \( y_0 = 0.55 \), Minimum at \((1, 1) = 1.47911 \times 10^{-9} \approx 0\) ](image)

The backpropagation method is simply the gradient descent rule, applied on a feed forward multilayer neural network. That’s because all that someone has to cope with in this case, is being able to compute the partial derivatives of the multilayer perceptron with respect to the output values, something that can be easily achieved by the gradient method (Berthold & Hand, 1999).

### 2.5.2 Unsupervised Learning

In contrast to supervised learning, unsupervised learning has no target outputs so there is no prior knowledge about the output. The network has to discover for itself an efficient strategy to be followed, so in many cases the system during the unsupervised learning is trying to organise the data according to some similarities that they might have. By this process, the system succeeds in creating groups with similar characteristics known as clusters.
Popular techniques of supervised learning are Hebbian learning and competitive learning. In Hebbian learning, as mentioned earlier in this project, when neurons get to excitation and fire together, the connection between them become stronger while in competitive learning just one neuron is allowed to fire each time dependent on whether it has the largest weight.

2.5.3 Reinforcement Learning
In the reinforcement learning, the training of the neural network, is obtained by its interaction with the surrounding environment. Bertsekas and Tsitsiklis (1996) give in their book an apt example of an infant’s actions such as waving its hands, playing and looking around as a result of the stimulus that it accepts from the environment. In the same way with reinforcement learning, the learner, which in our case is the neural network, has no teacher, and the decision for the future actions are taken by the neural network itself.

The basic idea behind the theory of reinforcement learning is that the system has every time to decide among two actions, exploration or exploitation. So, the system could exploit actions that proved successful in the past and continue using them or trying to explore new ones and possibly improve its performance. Both actions have their risks however. With exploitation, the system takes the safest choice without risking a disastrous decision and its environment rewards it, but its performance remains stable as there are no tries for improving it.

In the case of exploration, the system takes the risk to search for a better action and the chance of failure leads it to ‘punishment’ from its environment.

The totally successful system, has the ability of evaluating the environment’s situation, and combines both choices, either exploration or exploitation, taking its time and taking the best possible choice for it, without having failures as a result from its environment, but just rewards.

2.6 Overfitting Problem
In cases of complex problems of time series where the model includes many degrees of freedom, or there is not noise in the training data, the problem of overfitting can occur. When using neural networks taking time series as inputs, this problem follows through with the output produced. More specifically, a definition given by Mitchell (1997) is: Let h, h’ ∈ H where H is a given hypothesis space and h and h’ are two alternate hypotheses.
If, h has smaller error than h’ over the training example but h’ has a smaller error than h over the entire distribution of instances. Then h is said to overfit the training data.

In cases where overfitting occurs, the neural network results in producing poor predictions for the time series. Some solutions to this problem have been developed. The most popular are the partition of the data into training and test set, the minimization of the hidden layers or to add some noise to the training data.

2.7 Neural Networks and Forecasting
The main target with the development of neural networks is to use them as a tool for prediction of nonlinear time series. Their main difference from the traditional techniques that are used for forecasting is their flexible structure and that they can learn either linear or nonlinear time series patterns and predict them with accuracy in cases that many traditional techniques fail. Their parallel nature protects the model from total failure as even if one element of the neural network fails to complete its task, the system will still continue its process.

Their operation for forecasting time series can be easily described as a black box with inputs being the time series data which undergoes some process in the black box known only by the programmer/researcher, giving as an output the time series prediction.
One really important application of artificial neural networks is when someone has to cope with cases where time series contains some chaotic behavior, which proves either unpredictable or at the very least really difficult to predict in the long term. One of many examples that characterised this behaviour is given by Lorenz (1963) in his paper “Deterministic Nonperiodic Flow” where he proposed a model for weather prediction where the lack of periodicity is more than obvious. He based his research on Navier-Stokes time dependent partial differential equations that are popular in fluid dynamics but have been used in different cases of modelling as well.

In Lorenz’s case he used them for weather prediction (Dudul, 2005), simplifying these equations greatly to conclude with a form of equations that finally got his name, the Lorenz equations, which are given by the following:

\[
\begin{align*}
\frac{dx(t)}{dt} &= \sigma [y(t) - x(t)] \\
\frac{dy(t)}{dt} &= x(t)[r - z(t)] - y(t) \\
\frac{dz(t)}{dt} &= x(t)y(t) - bz(t)
\end{align*}
\]  

(2-13)

where \( \sigma, \rho \) and \( b \) dimensionless parameters with typical values \( \sigma=10, b=8/3, r=28 \).

Lorenz’s approach was really important as he was a great meteorologist and computer modeller even though the computer technology was in an elementary stage in his days. In his effort to make good predictions for the atmosphere, he discovered chaos and chaotic attractors. He didn’t use a realistic weather model, simulating just simple models, as he was suspicious of how long it would take and if there was a chance for long term-predictability.

Many studies that followed used Lorenz’s equations as tools for prediction. Dudul (2005) in his study aimed to predict Lorenz’s equation using a neural network. In this project, following Dudul’s idea, a feedforward neural network is constructed for non-linear time series forecasting and many experiments are tried.
2.8 Experiment: The Developed Neural Network

A backpropagation supervised learning scheme is used by which a feedforward network is trained. The network starts out with interconnections of random weights. Our neural network may have a number of hidden layers each containing a number of neurons. The neurons in a given layer do not connect to each other and do not take input from subsequent layers or from layers before the previous one. So layer 1 sends output to layer 2, and layer 2 takes inputs form layer 1 and sends outputs to layer 3. There are no connections between layers 1 and 3.

However, when designing a neural network, there are always many tricky parts that someone has to pay much attention to, especially with how to present the data. How many inputs to consider? How many outputs? How many hidden layers? There are always many different ways to do it. Let us consider a time series of length $N=400$, which we want to predict. This time series is actually the sine function from 0 up to 15 radians. The step is $15/(400-1)$ radians.

Let’s assume, as we suspect, that each data point of the time series is related to its previous $M=10$ points. The choice of the number 10 is indicative though, as the basic idea is to get someone to understand the strong correlation among the data points due to the sampling rate. This allows us to construct a grid of 390 examples (training sets). Each training set is of the form $x_1, x_2, x_3 \ldots x_{10}$ inputs and one output.

![Figure 2.8.1](image)

Table 2.1 – Each row is a training example of 10 input fields and one output field.

<table>
<thead>
<tr>
<th>Example</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
<th>Input 5</th>
<th>Input 6</th>
<th>Input 7</th>
<th>Input 8</th>
<th>Input 9</th>
<th>Input 10</th>
<th>Output 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.05909</td>
<td>0.08865</td>
<td>0.67167</td>
<td>0.11254</td>
<td>0.14369</td>
<td>0.18686</td>
<td>0.22936</td>
<td>0.27083</td>
<td>0.31230</td>
<td>0.35377</td>
<td>0.20686</td>
</tr>
<tr>
<td>2</td>
<td>0.037585</td>
<td>0.075117</td>
<td>0.112543</td>
<td>0.14981</td>
<td>0.186866</td>
<td>0.223656</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
</tr>
<tr>
<td>3</td>
<td>0.079167</td>
<td>0.115454</td>
<td>0.14981</td>
<td>0.186866</td>
<td>0.223656</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
</tr>
<tr>
<td>4</td>
<td>0.112543</td>
<td>0.14981</td>
<td>0.186866</td>
<td>0.223656</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
</tr>
<tr>
<td>5</td>
<td>0.149819</td>
<td>0.186866</td>
<td>0.223656</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
</tr>
<tr>
<td>6</td>
<td>0.186866</td>
<td>0.223656</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
<td>0.534145</td>
</tr>
<tr>
<td>7</td>
<td>0.223656</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
<td>0.534145</td>
<td>0.565063</td>
</tr>
<tr>
<td>8</td>
<td>0.260131</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
<td>0.534145</td>
<td>0.565063</td>
<td>0.596871</td>
</tr>
<tr>
<td>9</td>
<td>0.296238</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
<td>0.534145</td>
<td>0.565063</td>
<td>0.596871</td>
<td>0.628217</td>
</tr>
<tr>
<td>10</td>
<td>0.329227</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
<td>0.534145</td>
<td>0.565063</td>
<td>0.596871</td>
<td>0.628217</td>
<td>0.659978</td>
</tr>
<tr>
<td>11</td>
<td>0.362143</td>
<td>0.401848</td>
<td>0.435981</td>
<td>0.468468</td>
<td>0.502221</td>
<td>0.534145</td>
<td>0.565063</td>
<td>0.596871</td>
<td>0.628217</td>
<td>0.659978</td>
<td>0.691571</td>
</tr>
</tbody>
</table>

Margarita Papadopoulou
To accommodate this data structure, we need a neural network of 10 inputs. Further, one or more hidden layers and one output, all interconnected. In order to acquire a useful predictor, we are obliged to restrict to as few hidden neurons as possible. This will assure that the produced network will not just memorize the data, but will also be able to generalize and predict unknown situations. As with all fitting methods, a great number of parameters leads to memorizing the data. A small number of parameters leads to learning disability. The best practice is to try to teach the network with the fewest possible neurons and if it cannot learn from the data, then to increase the number of neurons or add an extra hidden layer (Mitchell, 1997).

Usually, one has to shuffle the training sets and put aside 10 percent of them for validation reasons. Following this practice, we are left with 390-39=351 training sets and 39 test sets.

By feeding the network with all the 350 training sets (one epoch), the weights of the synapses are continuously adjusted so that the total mean squared error is minimized. A second and third, etc. epoch can lead to a desirable total mean squared error proving that the neural network has learned correctly the data.

### Table 2-2 – Shuffle training sets.

<table>
<thead>
<tr>
<th>Example 0</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
<th>Input 5</th>
<th>Input 6</th>
<th>Input 7</th>
<th>Input 8</th>
<th>Input 9</th>
<th>Input 10</th>
<th>Output 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td>0.12099101</td>
<td>0.394442</td>
<td>0.3666927</td>
<td>0.884459</td>
<td>0.8632971</td>
<td>0.843478</td>
<td>0.825393</td>
<td>0.807496</td>
<td>0.777665</td>
<td>0.754687</td>
<td>0.735243</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>500</td>
<td>0.3514435</td>
<td>0.959445</td>
<td>0.896445</td>
<td>0.906445</td>
<td>0.905455</td>
<td>0.907356</td>
<td>0.909356</td>
<td>0.910356</td>
<td>0.911356</td>
<td>0.912356</td>
<td>0.913356</td>
</tr>
<tr>
<td>502</td>
<td>0.3097812</td>
<td>0.803134</td>
<td>0.8077132</td>
<td>0.808868</td>
<td>0.80998</td>
<td>0.810998</td>
<td>0.811998</td>
<td>0.812998</td>
<td>0.813998</td>
<td>0.814998</td>
<td>0.815998</td>
</tr>
<tr>
<td>49</td>
<td>0.3497928</td>
<td>0.974949</td>
<td>0.976286</td>
<td>0.976496</td>
<td>0.976596</td>
<td>0.976696</td>
<td>0.976796</td>
<td>0.976896</td>
<td>0.976996</td>
<td>0.977096</td>
<td>0.977196</td>
</tr>
<tr>
<td>65</td>
<td>0.819446</td>
<td>0.799746</td>
<td>0.775419</td>
<td>0.753457</td>
<td>0.724919</td>
<td>0.696496</td>
<td>0.667164</td>
<td>0.642674</td>
<td>0.613425</td>
<td>0.58368</td>
<td>0.552369</td>
</tr>
<tr>
<td>230</td>
<td>0.874277</td>
<td>0.201024</td>
<td>0.202024</td>
<td>0.203024</td>
<td>0.204024</td>
<td>0.205024</td>
<td>0.206024</td>
<td>0.207024</td>
<td>0.208024</td>
<td>0.209024</td>
<td>0.210024</td>
</tr>
<tr>
<td>230</td>
<td>0.874277</td>
<td>0.201024</td>
<td>0.202024</td>
<td>0.203024</td>
<td>0.204024</td>
<td>0.205024</td>
<td>0.206024</td>
<td>0.207024</td>
<td>0.208024</td>
<td>0.209024</td>
<td>0.210024</td>
</tr>
<tr>
<td>230</td>
<td>0.874277</td>
<td>0.201024</td>
<td>0.202024</td>
<td>0.203024</td>
<td>0.204024</td>
<td>0.205024</td>
<td>0.206024</td>
<td>0.207024</td>
<td>0.208024</td>
<td>0.209024</td>
<td>0.210024</td>
</tr>
<tr>
<td>230</td>
<td>0.874277</td>
<td>0.201024</td>
<td>0.202024</td>
<td>0.203024</td>
<td>0.204024</td>
<td>0.205024</td>
<td>0.206024</td>
<td>0.207024</td>
<td>0.208024</td>
<td>0.209024</td>
<td>0.210024</td>
</tr>
<tr>
<td>230</td>
<td>0.874277</td>
<td>0.201024</td>
<td>0.202024</td>
<td>0.203024</td>
<td>0.204024</td>
<td>0.205024</td>
<td>0.206024</td>
<td>0.207024</td>
<td>0.208024</td>
<td>0.209024</td>
<td>0.210024</td>
</tr>
</tbody>
</table>

Margarita Papadopoulou
As far as the number of inputs is concerned, if very few input fields were chosen, there would be a danger that they would not include much information to be revealed. On the other hand, if a big number of input fields were chosen, there would be also some risk to lead to conflicting results, thus inhibiting good training. The number of input fields is decided by the quality of training and the quality of testing, namely, the maximum error during training, and most importantly the error on the forward testing of the network using the reserved data. Moreover, many hidden layers would lead to many neurons. By increasing the hidden layers, the neural network does absorb the learning information easier. It actually memorizes the data instead of optimally changes its self to become a useful predictor. In general, many neurons degrade the predictive power of the neural network.

**Testing the neural network**

After the neural network is constructed and trained, a very important stage is to validate it by testing its quality. Here we test the quality of our neural network by feeding it with the reserved 39 test examples, expecting a low total mean squared error. In our case, the learning phase had a total mean squared error of 0.0227481099353903, which is considered satisfactory, proving that the network structure is appropriate.
**Prediction**

To predict the $n+1^{th}$ value of the time series, we simply feed the network with the last 10 values to a forward run. We see that the prediction (shown as the red dot) fits perfectly in the sine function.

After having the prediction of the time series we conclude that using one hidden layer of three neurons, was the right choice as we obtained satisfactory results for the 39 reserved data sets and satisfactory prediction of the future.

*Figure 2.8.6 – The last dot in red is the predicted $n + 1$ value.*

*Figure 2.8.7 – Ten days prediction at the end of the series.*
2.9 Prediction of Typical Chaotic Time Series using a Neural Network

As mentioned earlier in this project, following Dudul’s idea in his paper “Prediction of Lorenz Attractor using two-layer perceptron neural network”, a two-layer neural network is designed. Now we want to predict the Lorenz system using the neural network that is developed.

Figure 2.9.1 below illustrates a Lorenz’s attractor formed of a time series of 16384 data points (data by Weeks). This data set was created by Runge-Kutta integration of the Lorenz equations with a step size of 0.01 (Press, W. H. et al., 2007). So as given earlier in this project, the Lorenz equations are given by:

\[
\frac{dx(t)}{dt} = \sigma[y(t) - x(t)]
\]

\[
\frac{dy(t)}{dt} = x(t)[r - z(t)] - y(t)
\]

\[
\frac{dz(t)}{dt} = x(t)y(t) - bz(t)
\]

We have used the standard values of \( \sigma=10.0, r=28.0, b=8/3 \). These values are fixed by Lorenz where \( \sigma \) denotes Prandtl number, \( r = \frac{r_c}{a} \) and \( b = \frac{4}{(1+a^2)} = \frac{8}{3} \) where \( a \) parameter of geometry which is fixed at \( a = \frac{1}{2} \). The values \( \sigma = 10 \) and \( r = 28 \) that Lorenz have chosen, correspond to two steady states of the differential equation, which correspond to the centres of the holes which are the chaotic attractors as shown in the schema above (Peitgen, Hartmut, & Dietmar, 2004).

The specific neural network that was used for the prediction of the Lorenz system has ten input fields, one hidden layer with ten neurons and one output. We usually try to normalize the time series before the prediction. Here a simple division with ten is adequate (Figure 2.9.2).
Nonlinear Time Series Forecasting using Neural Networks and Fuzzy System Techniques

A detail of a ten day prediction is given in Figure 2.9.3 where the green dots are fit after training and the brown dots are the prediction. It is obvious that the neural network succeeds in learning the dynamics of the times series with a high accuracy.

Table 2-3 – The last training examples grid.

<table>
<thead>
<tr>
<th>Example ID</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
<th>Input 5</th>
<th>Input 6</th>
<th>Input 7</th>
<th>Input 8</th>
<th>Input 9</th>
<th>Input 10</th>
<th>Output 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000668</td>
<td>0.459028</td>
<td>0.125023</td>
<td>0.354447</td>
<td>0.512029</td>
<td>0.051336</td>
<td>0.640498</td>
<td>0.724145</td>
<td>0.750868</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
</tr>
<tr>
<td>010670</td>
<td>0.133253</td>
<td>0.150451</td>
<td>0.013209</td>
<td>0.061205</td>
<td>0.089298</td>
<td>0.724405</td>
<td>0.750868</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
</tr>
<tr>
<td>020681</td>
<td>0.524491</td>
<td>0.610597</td>
<td>0.606366</td>
<td>0.696958</td>
<td>0.244095</td>
<td>0.750868</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
</tr>
<tr>
<td>030692</td>
<td>0.103971</td>
<td>0.651266</td>
<td>0.689095</td>
<td>0.724405</td>
<td>0.709100</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
</tr>
<tr>
<td>040702</td>
<td>0.651266</td>
<td>0.689095</td>
<td>0.724405</td>
<td>0.755968</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
<td>0.115347</td>
</tr>
<tr>
<td>050711</td>
<td>0.893995</td>
<td>0.724405</td>
<td>0.755968</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
</tr>
<tr>
<td>060721</td>
<td>0.744052</td>
<td>0.750868</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.885230</td>
</tr>
<tr>
<td>070732</td>
<td>0.709989</td>
<td>0.724405</td>
<td>0.755968</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.885230</td>
</tr>
<tr>
<td>080743</td>
<td>0.702479</td>
<td>0.606366</td>
<td>0.696958</td>
<td>0.244095</td>
<td>0.750868</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
</tr>
<tr>
<td>090754</td>
<td>0.893995</td>
<td>0.724405</td>
<td>0.755968</td>
<td>0.702479</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.181461</td>
<td>0.181461</td>
<td>0.885230</td>
<td>0.115347</td>
<td>0.885230</td>
</tr>
<tr>
<td>100765</td>
<td>0.801633</td>
<td>0.779530</td>
<td>0.747754</td>
<td>0.709989</td>
<td>0.666276</td>
<td>0.616304</td>
<td>0.567545</td>
<td>0.515310</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
</tr>
<tr>
<td>110776</td>
<td>0.792093</td>
<td>0.747754</td>
<td>0.759989</td>
<td>0.666276</td>
<td>0.616304</td>
<td>0.567545</td>
<td>0.515310</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
</tr>
<tr>
<td>120787</td>
<td>0.847754</td>
<td>0.792093</td>
<td>0.666276</td>
<td>0.616304</td>
<td>0.567545</td>
<td>0.515310</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
</tr>
<tr>
<td>130798</td>
<td>0.792093</td>
<td>0.666276</td>
<td>0.616304</td>
<td>0.567545</td>
<td>0.515310</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
</tr>
<tr>
<td>140809</td>
<td>0.602275</td>
<td>0.610204</td>
<td>0.567545</td>
<td>0.515310</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
<td>0.462852</td>
</tr>
</tbody>
</table>

Someone may also wonder why we have chosen to predict a Lorenz system using a neural network as a tool and not one of the simple linear predictors. Wanting to answer this question the last 685 data points of the Lorenz time series were imported to the statistical package MINITAB for a 10-day linear prediction. Observing the original data plotted in Figure 2.9.4 and then the prediction plotted in Figure 2.9.5 it is clear that the linear predictor miserably fails and linear models like AR, ARMA and state-space models have not been able to learn the dynamics of this time series, and thus these linear models are not at all perfect.
Table 2-4 – Final estimates of parameters. Forecasts from period 685.

<table>
<thead>
<tr>
<th>Period</th>
<th>Forecast</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>686</td>
<td>3.78775</td>
<td>2.87761</td>
<td>4.69789</td>
</tr>
<tr>
<td>687</td>
<td>3.78409</td>
<td>2.49794</td>
<td>5.07024</td>
</tr>
<tr>
<td>688</td>
<td>3.78044</td>
<td>2.20645</td>
<td>5.35443</td>
</tr>
<tr>
<td>689</td>
<td>3.77679</td>
<td>1.96070</td>
<td>5.59288</td>
</tr>
<tr>
<td>690</td>
<td>3.77315</td>
<td>1.74425</td>
<td>5.80204</td>
</tr>
<tr>
<td>691</td>
<td>3.76951</td>
<td>1.54867</td>
<td>5.99035</td>
</tr>
<tr>
<td>692</td>
<td>3.76588</td>
<td>1.36894</td>
<td>6.16282</td>
</tr>
<tr>
<td>693</td>
<td>3.76226</td>
<td>1.20178</td>
<td>6.32273</td>
</tr>
<tr>
<td>694</td>
<td>3.75864</td>
<td>1.04492</td>
<td>6.47235</td>
</tr>
<tr>
<td>695</td>
<td>3.75502</td>
<td>0.89671</td>
<td>6.61334</td>
</tr>
</tbody>
</table>

However, there is always much discussion around prediction as working with real world data there is a need for predictions to be accurate. For example a bad prediction for the Dow-Jones index can be really disastrous for the economic stability of a company and even influence the stability of the global economy or an inaccurate prediction of extreme weather phenomena can prove to be really dangerous for human life.

Another thing for discussion is how far the prediction model can go and how accurate these predictions are going to be. In the real world there is also some noise in the data even from the environment or from other sources which can reflect on the data and give biased predictions.

So, in this project taking into consideration all the constraints that were discussed above we want to see how effectively our neural network will work when we want to have long term predictions or when we add some noise to the data.
Testing various models with respect to prediction
Now, instead of making a ten day prediction for the time series data we decide to make a forty day prediction but also change various aspects of our neural network. Four different tests were designed in this direction. The last portion of the original time series is shown in Figure 2.9.6 to be compared with 40 steps of predictions which follow various neural network models.

The last 40 points of the original series were removed from training, to be used as testing the prediction of 40 steps ahead.

Tests
Test 1: For the first test, the neural network model used has 10 input fields, 1 hidden layer of 3 neurons and the prediction is very satisfactory.

Test 2: For our second test we keep the number of input fields the same as above (10), but we reduce the number of neurons in the hidden layer from three to one. The only noticeable difference is that the peak is slightly lower.

Test 3: For the third test we reduce the number of the input fields from 10 to 5. The results though are not satisfactory and it seems that the number of inputs are not adequate for 40 steps prediction.

Test 4: If we now retain the input fields to five but increase the number of neurons in the hidden layer from 1 to 3 we have definitely better results.

As it is expected, by increasing either the inputs or the internal neurons of the hidden layers, the neural network understands the data better but the prediction is not guaranteed as there is a great possibility of overfitting to occur.
Adding noise to the data

Adding noise to the data would not affect much the result. As the initial weights are random and also as the shuffle before training is a random procedure, this leads to result weights that always differ from training to training and the number of training cycles. The result weights would differ similarly after adding noise. So, not much of a difference is expected after adding noise.

Three cases of adding noise to the data have been studied retaining the same neural network model of ten inputs and one hidden layer with three neurons.

An example of adding 15% of the series range as noise is shown in Figure 2.9.11.

Figure 2.9.12 shows 40 prediction steps before adding noise for comparison with the prediction cases after noise.

Of all the training examples, 10% of them were put aside for testing purposes. The quality of learning was tested using these reserved examples and is shown in Figure 2.9.13.
The three cases

Case 1: 10 input fields, 1 hidden layer of three neurons, 5% of the range as noise

Case 2: 10 input fields, 1 hidden layer of three neurons, 10% of the range as noise

Case 3: 10 input fields, 1 hidden layer of three neurons, 15% of the range as noise

We then conclude, that the direction of the first prediction steps was not affected much after adding noise. On the contrary, the noise affected the prediction of further steps ahead.

We expected this result though, as we know that long term prediction for chaotic series is really difficult, if not impossible in many cases. Gomez-Gil (2007) in her article "Long-Term Prediction, Chaos and Artificial Neural Networks. Where is the Meeting Point?" in order to approach this problem, suggested the construction of a dynamical system using a neural network, where the new system will include chaotic invariants similar to those of the chaotic time series. She suggests that the ability of constructing such a system, will probably help us to define the bounds of such predictions.
2.10 Prediction of Typical Chaotic Series Using an ARMA Model

We now use an ARMA\((p, q)\) model as an alternative to the neural network that we have used previously for the prediction of typical chaotic series. We tried various cases with different \(p\) and \(q\) in order to investigate any significant differences with the prediction generated by the neural network. The ARMA model in time series is typically written as:

\[
x(t) - p_1 x(t-1) - p_2 x(t-2) - \ldots - p_p x(t-p) = w(t) - q_1 w(t-1) - \ldots - q_q w(t-q)
\]  

(2-14)

where the \(p\)'s and \(q\)'s are fixed parameters and the \(w(t)\)'s represent independent normal random variables called white noise. This series is typically written ARMA\((p, q)\).

There is a significant number of implementations of the ARMA programs in the literature. We have chosen to use an Excel Add-in written by Kurt Annen in order to test various \(p\) and \(q\) with respect to 40 steps prediction, as with the neural network 40 step prediction. Kurt Annen’s program proposes \(p=5\) and \(q=5\).

Again, the 40 last points have been removed from the original series to be compared with the ARMA 40-step forecast (see Figure 2.10.1).

![Figure 2.10.1 – Time series of actual (blue) and fitted (red) data. The diagram shows that the prediction perfectly fits the actual data.](image)

Results

We have studied various cases of \(p\) and \(q\) which are all listed below. Due to the high degree of determinism in the data series, the model gives rather accurate forecasts for various \(p,q\) except the case where \(p=1\).
Case 1 - p=10, q=0

Table 2-5 -

<table>
<thead>
<tr>
<th>timeseries: y</th>
<th>Method: Nonlinear Least Squares (Levenberg-Marquardt)</th>
<th>date: 07-22-10</th>
<th>time: 12:32</th>
<th>Included observations: 1022</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p = 10 - q = 0 - no constant - manual selection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-Statistic</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(1)</td>
<td>4.163313268</td>
<td>0.314418</td>
<td>13.24131808</td>
</tr>
<tr>
<td>AR(2)</td>
<td>-5.685297539</td>
<td>2.129067</td>
<td>-2.67032366</td>
</tr>
<tr>
<td>AR(3)</td>
<td>1.29205354</td>
<td>6.245828</td>
<td>0.20686658</td>
</tr>
<tr>
<td>AR(4)</td>
<td>2.781652197</td>
<td>10.49715</td>
<td>0.264991103</td>
</tr>
<tr>
<td>AR(5)</td>
<td>-0.438855256</td>
<td>12.17272</td>
<td>-0.036052343</td>
</tr>
<tr>
<td>AR(6)</td>
<td>-1.978792113</td>
<td>12.48415</td>
<td>-0.158504353</td>
</tr>
<tr>
<td>AR(7)</td>
<td>0.035980752</td>
<td>10.96669</td>
<td>0.003280911</td>
</tr>
<tr>
<td>AR(8)</td>
<td>1.595196102</td>
<td>6.479735</td>
<td>0.246182304</td>
</tr>
<tr>
<td>AR(9)</td>
<td>-0.910897077</td>
<td>2.185562</td>
<td>-0.416779364</td>
</tr>
<tr>
<td>AR(10)</td>
<td>0.14563352</td>
<td>0.319835</td>
<td>0.455339414</td>
</tr>
</tbody>
</table>

R-squared     | 1.0000000    | Mean dependent var | -1.475778   |
Adjusted R-squared | 1.0000000 | S.D. dependent var | 7.646238   |
S.E. of regression | 0.000268    | Akaike info criterion | -13.598707 |
Sum squared resid  | 0.000073    | Schwarz criterion | -13.550473 |
Log likelihood   | 6958.939057 | Durbin-Watson stat | 0.166226   |

Inverted AR-roots | 0.92-0.30i | 0.92+0.30i | -0.55-0.59i | -0.55+0.59i |
|                 | 0.97-0.14i | 0.97+0.14i | 0.99-0.04i  | 0.99+0.04i  |
|                 | -0.8      | 0.31      |            |             |

Figure 2.10.2 – Residual Plot

Figure 2.10.3 – Forecast
Case 2 - p=5, q=3

Table 2-6 -

<table>
<thead>
<tr>
<th>timeseries: y</th>
<th>Method: Nonlinear Least Squares (Levenberg-Marquardt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>date: 07-22-10</td>
<td>time: 12:20</td>
</tr>
<tr>
<td>Included observations: 1027</td>
<td>p = 5 - q = 3 - no constant - manual selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-Statistic</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(1)</td>
<td>4.88248782</td>
<td>0.007882</td>
<td>619.4131457</td>
<td>0</td>
</tr>
<tr>
<td>AR(2)</td>
<td>-9.5993818</td>
<td>0.030936</td>
<td>-310.2940805</td>
<td>0</td>
</tr>
<tr>
<td>AR(3)</td>
<td>9.50028345</td>
<td>0.045868</td>
<td>207.1237385</td>
<td>0</td>
</tr>
<tr>
<td>AR(4)</td>
<td>-4.73287288</td>
<td>0.030448</td>
<td>-155.433149</td>
<td>0</td>
</tr>
<tr>
<td>AR(5)</td>
<td>0.94947886</td>
<td>0.007635</td>
<td>124.3590529</td>
<td>0</td>
</tr>
<tr>
<td>MA(1)</td>
<td>1.91039688</td>
<td>0.016354</td>
<td>116.816069</td>
<td>0</td>
</tr>
<tr>
<td>MA(2)</td>
<td>1.8378565</td>
<td>0.021922</td>
<td>83.83501382</td>
<td>-2.22E-16</td>
</tr>
<tr>
<td>MA(3)</td>
<td>0.81768812</td>
<td>0.016059</td>
<td>50.91816347</td>
<td>-2.22E-16</td>
</tr>
</tbody>
</table>

| R-squared | 1.000000 | Mean dependent var | -1.464810 |
| Adjusted R-squared | 1.000000 | S.D. dependent var | 7.629199 |
| S.E. of regression | 0.000661 | Akaike info criterion | -16.548142 |
| Sum squared resid | 0.000004 | Schwarz criterion | -16.509705 |
| Log likelihood | 8505.470925 | Durbin-Watson stat | 0.886692 |
| Inverted AR-roots | 0.99 | 0.96-0.24i | 0.96+0.24i | 0.99-0.09i |
| Inverted MA-roots | -0.52-0.82i | -0.52+0.82i | 0.88 |

Figure 2.10.4 – Residual Plot

Figure 2.10.5 – Forecast
Case 3 - p=3, q=3

Table 2-7 -

timeseries: y
Method: Nonlinear Least Squares (Levenberg-Marquardt)
date: 07-22-10 time: 12:23
Included observations: 1029
p = 3 - q = 3 - no constant - manual selection

<table>
<thead>
<tr>
<th></th>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-Statistic</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(1)</td>
<td>2.9217765</td>
<td>0.004861</td>
<td>601,0129982</td>
<td>0</td>
</tr>
<tr>
<td>AR(2)</td>
<td>-2.8540112</td>
<td>0.00963</td>
<td>-296,353082</td>
<td>0</td>
</tr>
<tr>
<td>AR(3)</td>
<td>0.9319203</td>
<td>0.004797</td>
<td>194,2813736</td>
<td>0</td>
</tr>
<tr>
<td>MA(1)</td>
<td>2.2927243</td>
<td>0.021292</td>
<td>107,6816492</td>
<td>0</td>
</tr>
<tr>
<td>MA(2)</td>
<td>2.1779909</td>
<td>0.03599</td>
<td>60,51671008</td>
<td>1.11E-16</td>
</tr>
<tr>
<td>MA(3)</td>
<td>0.844663</td>
<td>0.021373</td>
<td>39,51943614</td>
<td>0</td>
</tr>
</tbody>
</table>

R-squared 1,000000 Mean dependent var -1,460053
Adjusted R-squared 1,000000 S.D. dependent var 7,622537
S.E. of regression 0,001223 Akaike info criterion -10,559413
Sum squared resid 0,001529 Schwarz criterion -10,530630
Log likelihood ######## Durbin-Watson stat 0,485708

Inverted AR-roots 0.98-0.09i 0.98+0.09i 0.96
Inverted MA-roots -0.68-0.67i -0.68+0.67i -0.93

Figure 2.10.6 – Residual Plot

Figure 2.10.7 – Forecast
Case 4 - p=1, q=1

Table 2-8 -

<table>
<thead>
<tr>
<th>Timeseries: y</th>
<th>Method: Nonlinear Least Squares (Levenberg-Marquardt)</th>
<th>Date: 07-22-10 time: 12:25</th>
<th>Included observations: 1031</th>
<th>p = 1 - q = 1 - no constant - manual selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(1)</td>
<td>0.998436</td>
<td>0.001798</td>
<td>555.3232196</td>
<td>0</td>
</tr>
<tr>
<td>MA(1)</td>
<td>0.997442</td>
<td>0.001277</td>
<td>781.1634856</td>
<td>0</td>
</tr>
<tr>
<td>R-squared</td>
<td>0.999135</td>
<td>Mean dependent var</td>
<td>-1.454913</td>
<td></td>
</tr>
<tr>
<td>Adjusted R-squared</td>
<td>0.999134</td>
<td>S.D. dependent var</td>
<td>7.616026</td>
<td></td>
</tr>
<tr>
<td>S.E. of regression</td>
<td>0.224080</td>
<td>Akaike info criterion</td>
<td>-0.151534</td>
<td></td>
</tr>
<tr>
<td>Sum squared resid</td>
<td>51.668200</td>
<td>Schwarz criterion</td>
<td>-0.141955</td>
<td></td>
</tr>
<tr>
<td>Log likelihood</td>
<td>80.115932</td>
<td>Durbin-Watson stat</td>
<td>0.014921</td>
<td></td>
</tr>
<tr>
<td>Inverted AR-roots</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverted MA-roots</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.10.8 – Residual Plot

Figure 2.10.9 – Forecast
Case 5 - p=5, q=5

Table 2-9 -

timeseries: y
Method: Nonlinear Least Squares (Levenberg-Marquardt)
date: 07-22-10 time: 12:30
Included observations: 1027
p = 5 - q = 5 - no constant - manual selection

<table>
<thead>
<tr>
<th></th>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-Statistic</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(1)</td>
<td>3.179428773</td>
<td>0.270236</td>
<td>11.75639993</td>
<td>3.331E-16</td>
</tr>
<tr>
<td>AR(2)</td>
<td>-2.98717111</td>
<td>1.041595</td>
<td>-2.867880826</td>
<td>0.0042177</td>
</tr>
<tr>
<td>AR(3)</td>
<td>-0.158894947</td>
<td>1.510178</td>
<td>-0.105216013</td>
<td>0.9162252</td>
</tr>
<tr>
<td>AR(4)</td>
<td>1.557755568</td>
<td>0.97589</td>
<td>1.59624033</td>
<td>0.1107458</td>
</tr>
<tr>
<td>AR(5)</td>
<td>-0.591133345</td>
<td>0.237075</td>
<td>-2.493440687</td>
<td>0.0128092</td>
</tr>
<tr>
<td>MA(1)</td>
<td>2.472681931</td>
<td>0.274466</td>
<td>9.00906983</td>
<td>1.11E-15</td>
</tr>
<tr>
<td>MA(2)</td>
<td>3.497541198</td>
<td>0.488295</td>
<td>7.16276095</td>
<td>1.514E-12</td>
</tr>
<tr>
<td>MA(3)</td>
<td>3.368473526</td>
<td>0.615703</td>
<td>5.470936899</td>
<td>5.636E-08</td>
</tr>
<tr>
<td>MA(4)</td>
<td>2.182676583</td>
<td>0.482148</td>
<td>4.526982651</td>
<td>6.69E-06</td>
</tr>
<tr>
<td>MA(5)</td>
<td>0.707873629</td>
<td>0.25416</td>
<td>2.785148126</td>
<td>0.0054497</td>
</tr>
</tbody>
</table>

R-squared 1.000000  Mean dependent var -1.464810
Adjusted R-squared 1.000000 S.D. dependent var 7.629199
S.E. of regression 0.000201 Akaike info criterion -14.165427
Sum squared resid 0.000041 Schwarz criterion -14.117380
Log likelihood 7283.946768 Durbin-Watson stat 0.108532

Inverted AR-roots

0.89
-0.67

Inverted MA-roots

-0.10-0.99i
-0.10+0.99i
-0.73-0.58i
-0.73+0.58i

-0.82

Figure 2.10.10 – Residual Plot

Figure 2.10.11 – Forecast
2.11 Extending the Experiment: Working with NARMA models

Recalling paragraph 2.5.1.2 of this project, an explicit non-linear model was being constructed by Ma & Wei (1999). In order for the neural network to be applicable to many non-linear models such as those suggested by Ma & Wei, where the sine functions, squares, etc. of the inputs where fed to the network in addition to the actual input values, we decided to modify the network feeding procedure by using an intermediate file. This intermediate file will contain any kind of pre-processed data, e.g. squaring the input fields or adding white noise input fields, etc. to accommodate for a nonlinear-ARMA. By doing this, one would need to have a general neural network flexible enough to process more models.

A series of tests of nonlinear ARMA (NARMA) models were conducted for the prediction of the Lorenz time series with and without noise. More specifically, the non-linear polynomial ARMA described in the Ma & Wei (1999) paper was checked for various parameters to see which one performs better. Again, we kept aside the last 40 points for testing purposes. Figure 2.11.2 shows the last portion of the time series after removing the 40 points.

According to Ma & Wei, the non-linear polynomial model is formally described as:

\[
\hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_j e_{n-j} + \sum_{k=1}^{r_1} c_k x_{n-k}^2 + \sum_{l=1}^{r_2} d_l e_{n-l}^2
\]  

(2-15)

where \( \hat{x}_n \) is the prediction of the time series at time \( n \), \( x_{n-i} \) are past values of the time series and \( e_{n-j} \) are white noise values. Then, the goal of prediction is to find the optimal parameters \( a_i, b_j, c_k, d_l \) of the model in a way that a total loss function (usually the total mean differences between of the actual and estimated values) is minimized. This can be done by an appropriately designed neural network (NN), the weights of which represent the ARMA model parameters to be estimated. A general NN structure of the above model consists of \( p + q + r_1 + r_2 \) inputs, 1 output, \( L \) hidden layers and \( N \) neurons. Figure 2.11.1 illustrates such a NN structure with 1 hidden layer.

We tested various parameters of the above NARMA model. Here we present some example cases which are worth mentioning.
Case 1 - p=5, q=0

Model (Ma & Wei): \[ \hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_j e_{n-j} + \sum_{k=1}^{r_1} c_k x_{n-k}^{k+2} + \sum_{l=1}^{r_2} d_l e_{n-l}^{k+2} \]

Table 2-10 – Case 1 parameters.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>5</td>
</tr>
<tr>
<td>q</td>
<td>0</td>
</tr>
<tr>
<td>r1</td>
<td>5</td>
</tr>
<tr>
<td>r2</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2-11 – Neural network parameters and results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Reserved data %</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training examples</td>
<td>16333</td>
<td>Alpha (step length)</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of inputs</td>
<td>15</td>
<td>Eta (degrading factor)</td>
<td>1</td>
</tr>
<tr>
<td>Number of layers</td>
<td>1</td>
<td>Total train MSE</td>
<td>0.00033604</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>3</td>
<td>Total test MSE</td>
<td>0.00035151</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>1</td>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Number of predictions</td>
<td>40</td>
<td>Noise %</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.11.3 – Neural network prediction of case 1.
Case 2 - \( p=10, q=3 \)

Model (Ma & Wei):

\[
\hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_j e_{n-j} + \sum_{k=1}^{r_1} c_{k} x_{n-k}^{k+2} + \sum_{k=1}^{r_2} d_{k} e_{n-k}^{k+2}
\]

Table 2-12 – Case 2 parameters.

<table>
<thead>
<tr>
<th>( p )</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
<td>3</td>
</tr>
<tr>
<td>( r_1 )</td>
<td>3</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2-13 – Neural network parameters and results.

<table>
<thead>
<tr>
<th>Training examples</th>
<th>16333</th>
<th>Reserved data %</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of inputs</td>
<td>19</td>
<td>Alpha (step length)</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of layers</td>
<td>1</td>
<td>Eta (degrading factor)</td>
<td>1</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>3</td>
<td>Total train MSE</td>
<td>0.00039862</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>1</td>
<td>Total test MSE</td>
<td>0.00039909</td>
</tr>
<tr>
<td>Number of predictions</td>
<td>40</td>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Noise %</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.11.4 – Neural network prediction of case 2.
Case 3 – Adding noise

Model \((\text{Ma & Wei})\): \[ \hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_j e_{n-j} + \sum_{k=1}^{l} c_k x_{n-k} + \sum_{k=1}^{r} d_k e_{n-k} \]

Table 2-14 – Case 3 parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p)</td>
<td>5</td>
</tr>
<tr>
<td>(q)</td>
<td>0</td>
</tr>
<tr>
<td>(r_1)</td>
<td>5</td>
</tr>
<tr>
<td>(r_2)</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2-15 – Neural network parameters and results.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training examples</td>
<td>16333</td>
</tr>
<tr>
<td>Reserved data %</td>
<td>0</td>
</tr>
<tr>
<td>Number of inputs</td>
<td>19</td>
</tr>
<tr>
<td>Alpha (step length)</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of layers</td>
<td>1</td>
</tr>
<tr>
<td>Eta (degrading factor)</td>
<td>1</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>3</td>
</tr>
<tr>
<td>Total train MSE</td>
<td>0.02248681</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>1</td>
</tr>
<tr>
<td>Total test MSE</td>
<td></td>
</tr>
<tr>
<td>Number of predictions</td>
<td>40</td>
</tr>
<tr>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Noise %</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 2.11.5 – Neural network prediction of case 3.
Case 4 – Increased noise

Model (Ma & Wei): \( \hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_j e_{n-j} + \sum_{k=1}^{r_1} c_{k} x_{n-k}^{k+2} + \sum_{k=1}^{r_2} d_{k} e_{n-k}^{k+2} \)

Table 2-16 – Case 4 parameters.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>10</td>
</tr>
<tr>
<td>q</td>
<td>2</td>
</tr>
<tr>
<td>r1</td>
<td>3</td>
</tr>
<tr>
<td>r2</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2-17 – Neural network parameters and results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Reserved data %</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training examples</td>
<td>16333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of inputs</td>
<td>20</td>
<td>Alpha (step length)</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of layers</td>
<td>1</td>
<td>Eta (degrading factor)</td>
<td>1</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>10</td>
<td>Total train MSE</td>
<td>0.049399228</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>1</td>
<td>Total test MSE</td>
<td></td>
</tr>
<tr>
<td>Number of predictions</td>
<td>20</td>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Noise %</td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.11.6 - Neural network prediction of case 4.
Case 5

Model (Ma & Wei): \( \hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_i x_{n-j} + \sum_{k=1}^{r_1} c_{k} x_{n-k} + \sum_{k=1}^{r_2} d_{k} e_{n-k} \)

Table 2-18 – Case 5 parameters.

| \( p \) | 10 |
| \( q \) | 0 |
| \( r_1 \) | 3 |
| \( r_2 \) | 0 |

Table 2-19 – Neural network parameters and results.

| Training examples | 16333 | Reserved data % | 0 |
| Number of inputs | 13 | Alpha (step length) | 0.7 |
| Number of layers | 1 | Eta (degrading factor) | 1 |
| Number of neurons | 3 | Total train MSE | 0.0001462 |
| Number of outputs | 1 | Total test MSE |
| Number of predictions | 45 | Epochs | 1 |
| Noise % | 0 |

Figure 2.11.7 – Neural network prediction of case 5.
Case 6

Model (Ma & Wei): \( \hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i} + \sum_{j=1}^{q} b_j e_{n-j} + \sum_{k=1}^{n} c_k x_{n-k}^{k+2} + \sum_{k=1}^{n} d_k e_{n-k}^{k+2} \)

Table 2-20 – Case 6 parameters.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>10</td>
</tr>
<tr>
<td>q</td>
<td>0</td>
</tr>
<tr>
<td>r1</td>
<td>3</td>
</tr>
<tr>
<td>r2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2-21 – Neural network parameters and results.

<table>
<thead>
<tr>
<th>Training examples</th>
<th>16333</th>
<th>Reserved data %</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of inputs</td>
<td>13</td>
<td>Alpha (step length)</td>
<td>0.7</td>
</tr>
<tr>
<td>Number of layers</td>
<td>1</td>
<td>Eta (degrading factor)</td>
<td>1</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>5</td>
<td>Total train MSE</td>
<td>9.65727971E-5</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>1</td>
<td>Total test MSE</td>
<td></td>
</tr>
<tr>
<td>Number of predictions</td>
<td>40</td>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Noise %</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.11.8 – Neural network prediction of case 6.

Although ARMA morels performed well in various configurations, the nonlinear ARMA were shown to be more stable especially for the case of the Lorenz time series where they performed better even when noise was present. Thus, the statement that was made in the beginning of this chapter for the flexibility and the efficiency of neural networks in comparison to linear models was verified.

Someone may wonder if in all these experiments it is possible to actually get hold of the coefficients so one can give the explicit model. The answer is that the coefficients result from each trial. Generally we should not make speculations of anything as every single case is different. During the testing period of our models we are trying to detect equilibrium in one region of the coefficients. This is the best indication that we are working in the right direction. Once we achieve this equilibrium it is of less importance if we have slightly different results for the coefficients than that which we are looking for. Coefficients should not be imposed in prediction but help it.
2.12 Santa Fe Competition and Real World Data

As mentioned quite a few times in this project, prediction of real world data can be even more complex because of the noise that is included from many different factors or the chaotic behaviour that they probably have.

The Santa Fe institute organised in 1991, a competition known as ‘Santa Fe Time Series Prediction and Analysis Competition’ with the main target to create a large database from real world data and give the chance to scientists from different disciplines to exchange data, detect common problems that occur in real world datasets and suggest different techniques to encounter them (Weigund & Gershenfeld, 1994). The final selection was six datasets varying from astrophysical data to data from Bach’s last unfinished fuga.

In the previous paragraphs we have tested our neural network by predicting typical chaotic series generated by Lorenz’s equations and the prediction was more than satisfactory. It is important though to examine the accuracy in prediction of the neural network when working with real data. For this purpose we have used a dataset from a clean physics laboratory experiment which was in the final six datasets of the Santa Fe competition. The data were recorded from a Far-Infrared-Laser in a chaotic state by Abraham & Hübner (1989). Feeding the data to our neural network we have surprisingly good results as shown in Figure 2.12.1 where the green colour represents the prediction.

![Figure 2.12.1](image-url)
Chapter 3: Fuzzy Neural Networks

3.1 Fuzzy Logic: Key ideas

Fuzzy sets were introduced by Zadeh (1965). The basic idea is that all propositions have a degree of truth that lies in the closed interval \([0,1]\). Thus, the so-called “Laws of Thought” of Aristotle do not hold. One of these, the “Law of Contradiction” states that every proposition must either be \textit{True (T)} or \textit{False (F)}. Another law, the “law of excluded middle”, states that if we have a universe set \(X\) and a subset \(A\) of \(X\), then \(A \cup A^c = X\). The above laws are important to achieve the foundation of any branch of mathematics. Indeed, we cannot say that a theorem is true with membership value 0.6. The theorem must either be \textit{True (T)} or \textit{False (F)}. However in our aim to model some physical or other phenomena the above laws are not valid.

Definition of fuzzy set

Let \(X\) be a universe set. Every map \(A : X \rightarrow [0,1]\) is called fuzzy subset of \(X\). If \(x \in X\), then the value \(A(x)\) is called \textit{membership value} of \(x\) and denotes the “degree” in which \(x\) “belongs” to the fuzzy set. Many times in the literature the following notation is used to denote a fuzzy subset of \(X\). As a fuzzy subset of \(X\) is considered the couple \((X, \mu_A)\), where \(\mu_A : X \rightarrow [0,1]\).

That is using this notation we consider as \(A = (X, \mu_A)\) and the function \(\mu_A\) is named the \textit{membership function}.

Obviously each crisp set is also a fuzzy one, since it can be “identified” with its characteristic function \(X_A : X \rightarrow \{0,1\}\), such that, \(X_A(x) = 1\) if \(x \in A\), or \(X_A(x) = 0\), otherwise.

Using crisp sets in many applications we face serious problems to describe many sets and many situations which occur in life.

Example 1

Let us describe the set of tall men.

a. Using crisp sets this can be described as follows:

\[
\begin{array}{c}
| x | y \\
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>190</td>
<td>1</td>
</tr>
</tbody>
</table>
\end{array}
\]

\[\text{Figure 3.1.1}\]

We understand that taking into consideration Figure 3.1.1, we should conclude that a man with height 1.89m is not tall, while a man with height 1.90m is a tall one.

So, we observe that using crisp sets we cannot describe effectively sets as the above. We would like to be able to characterize the man with height 1.89m as a tall one (not with membership value 1)

b. Using fuzzy sets this set can be represented as follows:

\[
\begin{array}{c}
| x | y \\
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>190</td>
<td>1</td>
</tr>
</tbody>
</table>
\end{array}
\]

\[\text{Figure 3.1.2}\]
3.2 Operations on Fuzzy Sets

Our aim is to show the importance of fuzzy sets, which relies on the fact that the laws of contradiction and of excluded middle are not valid. Before this, it is important though to define first the union, the intersection and the complement of fuzzy sets.

Union of two fuzzy sets
Let X be a universe set and let \( F(X) \) denote the set of all fuzzy subsets of X, i.e.
\[ F(X) = \{ \text{A} | \text{A} : X \to [0, 1] \}. \]
If \( \text{A} \in F(X) \), \( \text{B} \in F(X) \), then the fuzzy set \( \text{A} \cup \text{B} \) is defined as follows:

\[ (\text{A} \cup \text{B})(x) = \max\{\text{A}(x), \text{B}(x)\}, \text{ for each } x \in X. \]

So, the operation of union has the form:
\[ \cup : F(X) \times F(X) \to F(X), \text{ such that } (\text{A}, \text{B}) \to \text{A} \cup \text{B}. \]

Intersection of two fuzzy sets
Let X be a universe set and let \( F(X) \) denote the set of all fuzzy subsets of X, i.e
\[ F(X) = \{ \text{A} | \text{A} : X \to [0, 1] \}. \]
If \( \text{A} \in F(X) \), \( \text{B} \in F(X) \), then the fuzzy set \( \text{A} \cap \text{B} \) is defined as follows:

\[ (\text{A} \cap \text{B})(x) = \min\{\text{A}(x), \text{B}(x)\}, \text{ for each } x \in X. \]

So, the operation of intersection has the form:
\[ \cap : F(X) \times F(X) \to F(X), \text{ such that } (\text{A}, \text{B}) \to \text{A} \cap \text{B}. \]

Complement of a fuzzy set
Let X be a universe set and let \( F(X) \) denote the set of all fuzzy subsets of X, i.e
\[ F(X) = \{ \text{A} | \text{A} : X \to [0, 1] \}. \]
If \( \text{A} \in F(X) \) then the fuzzy set \( \text{A}^c \) is defined as follows:

\[ \text{A}^c \in F(X), \text{A}^c(x) = 1 - \text{A}(x), \text{ for each } x \in X. \]

The fuzzy set \( \text{A}^c \) has the following graph:

So, \( \text{A} \cup \text{A}^c \neq \emptyset \) and hence the law of contradiction is not valid.
So, \( A \cap A^C \neq X \), and hence the law of excluded middle is not valid.

### 3.3 Fuzzy Inference Systems (FIS)

A fuzzy inference system (FIS) is a system that uses fuzzy set theory to map inputs to outputs (Knapp, 1996). To construct an FIS we need the following:

- **Inputs:** We choose the input variables. For each variable \( X \) we determine its domain \([a_X, b_X]\). Then we determine the linguistic variables for each \( X \). The linguistic variables are fuzzy sets (triangular, trapezoid etc.) and they translate the terms “small”, “medium”, “tall” etc.

- **Outputs:** Similarly, we choose the output variables. For each variable \( Y \) we determine its domain \([a_Y, b_Y]\). Then we determine the linguistic variables for each \( Y \).

Finally, we define the rules:

**Rules:** These are rules between the linguistic variables of inputs and outputs. For example, let us suppose that we have two inputs, namely the temperature (T) and the humidity (H) and as an output the speed of the machine of an air-conditioner. The rules will have the form: If the temperature is high and the humidity is medium then the speed is high.

In MATLAB this FIS has the form as shown on Figure 3.3.1.

The terms “and”, “if”, “then” are founded mathematically. We must emphasize the fact that these terms are not uniquely defined. Indeed, we can choose the “and”, “implication” etc. between many mathematical expressions. Between all of them we choose those ones that are the most appropriate to our application. In MATLAB the first choice for “and” is the min. Also for the “implication” the first choice is the min too. More specifically, if we have the implication \( p \Rightarrow q \) and the truth values for \( p \) and \( q \) are \( a \) and \( b \) respectively, then the truth value of \( p \Rightarrow q \) is \( \min(a, b) \).

We have now defined the input, outputs and linguistic variables. Those variables are related to each other through fuzzy rules, once we define these rules the FIS is found. For example in MATLAB, if we construct such an FIS and we have now 26°C temperature and 67% humidity, then all the rules, which contain the values 26, 67, work and as a result we have a fuzzy number concerning the speed. For example if we have the following rules:

- a) If the temperature is medium and the humidity is high then the speed is medium.
- b) If the temperature is high and the humidity is high then the speed is high and if we also have the following linguistic variables (see Figure 3.3.2 & Figure 3.3.3) for the temperature and for humidity.
Then the above two rules will work, since $26^\circ C$ is medium and high with true values, which can be calculated via the below schema (Figure 3.3.2). Also the humidity is only high with a true value (Figure 3.3.3), which can be also calculated.

![Temperature Graph](image1)

*Figure 3.3.2 - Temperature*

![Humidity Graph](image2)

*Figure 3.3.3 – Humidity*

After an appropriate method, which is called defuzzification, we take as a result, a value $\sigma$, which shows us the speed of the machine of the aircondition. There are many methods of defuzzification and the most popular is the centroid one by which the mass center of the result provides the crisp value (Klir & Yuan, 1995).

### 3.4 Fuzzy Neural Networks key ideas

In this chapter, we will be discussing the incorporation of fuzziness in neural networks. The motivation to include fuzziness in neural networks is to extend their ability to deal with “vague” in addition to “crisp” information. In the context of time series prediction, the underlying mechanism governing the time series can be expressed as a set of fuzzy input-output relations, then the task of predicting the future can be carried out by a fuzzy neural network predictor.
There are several types of fuzzy neural network (FNN) approaches (Liu & Li, 2004).

1. Fuzzy neural networks with fuzzy operators:
   A popular fuzzy neural network with fuzzy operators is called Fuzzy Associative Memory and is a feedforward Fuzzy Neural Network whose information travels from input layer to output layer (Liu & Li, 2004). Its main characteristic is that it is able to store or recall fuzzy patterns and many learning methods such as the Fuzzy Gradient Descent Method have been developed, in order to improve this ability.

2. Fuzzified neural networks:
   A fuzzified neural network is again a feedforward neural network with the main characteristic that its set of inputs outputs and its synaptic weights constitute a fuzzy set (Liu & Li, 2004).

3. Fuzzy systems and fuzzy inference networks:
   Fuzzified neural networks as their fuzzy nature implies, are able to cope with large amount of linguistic information and inference rules and handle them really efficiently. Someone can then can create a fuzzy system with an input-output form which can also have the ability of self-learning and self-improving (Liu & Li, 2004).

**Main advantages and disadvantages of both fuzzy and neural approaches**

Neural networks and fuzzy logic are commonly used and often prove really useful for solving different nonlinear problems. They both have advantages but also limitations while acting in different tasks.

Fuzzy logic is able to cope with real complex problems as it can approach and model quite successfully the uncertainty of a system (Jain & Martin, 1998). This is because it is governed by the basic principle, which is approximate reasoning, rather than being precise. Because one system in fuzzy logic is described by linguistic variables and human expressions e.g. IF-THEN rules, someone can use his knowledge to determine the behaviour of the system. Jain & Martin (1998) state “It is extremely difficult, if not impossible, to develop a mathematical model of a complex system to reflect nonlinearity, uncertainty, and variation over time”. Considering this, it is easy to understand the reason that the development of fuzzy logic was really innovative.

On the other hand, as fuzzy logic has as its main tool a number of linguistic rules to overcome the complexity of a system, this means that the higher the complexity of the system is, the bigger the number of rules to be determined should be. The fact also that fuzzy logic uses heuristic rules do not guarantee satisfactory results.

Furthermore, if you exclude fuzzy adaptive systems which have some flexibility in their structure, in conventional fuzzy logic, the rules should be pre-determined and they remain stable in fuzzy logic controller during the whole process, which cannot be trained.

Neural networks as described analytically in the previous chapter are popular for their generalisation capabilities (Jain & Martin, 1998). These capabilities enable them to cope with demanding nonlinear problems successfully.

However, the human ability to understand always the black box process, which has a dominant role in neural networks, is certainly smaller than the rule oriented description of the fuzzy systems.

An appropriate combination of these two technologies can effectively solve the problems of fuzzy logic and neural nets and, thus, can more effectively address the real world complex problems.
Neuro-fuzzy approaches

The co-operation of the two can possibly diminish the problems that each approach faces when working individually. Constructing a fuzzy neural net allows us to exploit the useful properties present from each of the techniques. So, the neural network can be trained and learn the dynamics of a system where this knowledge can be used afterwards by fuzzy logic in order to generate rules and membership functions. This method is very efficient and much more cost-effective. Jain & Martin (1998) support that with the appropriate combination of fuzzy logic and neural nets, someone is able to map with absolute accuracy the neural network knowledge to fuzzy logic.

3.5 Experiment: Prediction of Lorenz attractor with Fuzzy Neural Networks

The prediction of the Lorenz Attractor derived by the crisp Neural Network will now be compared with the prediction given by a Fuzzy Neural Network. The Artificial Neural Network Fuzzy Inference System (ANFIS) will be used for this purpose. The initial time series is shown in Figure 3.5.1.

![Figure 3.5.1 – Lorenz attractor.](http://lejpt.academicdirect.org/A15/001_018.htm)

The basic structure of this type of fuzzy inference system is a model that maps input characteristics to input membership functions, input membership functions to rules, rules to a set of output characteristics, output characteristics to output membership functions, and the output membership function to a single-valued output or a decision associated with the output. In this section we discuss the use of the function anfis and the ANFIS Editor GUI in the MATLAB toolbox. These tools apply fuzzy inference techniques to data modelling. The model is shown in Figure 3.5.2.

![Figure 3.5.2 – ANFIS model structure.](http://lejpt.academicdirect.org/A15/001_018.htm)

What Is ANFIS

The acronym ANFIS derives its name from adaptive neuro-fuzzy inference system. Using a given input/output data set, the MATLAB toolbox function anfis constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone or in combination with a least squares type of method. This adjustment allows your fuzzy systems to learn from the data they are modelling. In Figure 3.5.3 we see the fuzzy model. The Sugeno fuzzy inference used has chosen to have 5 input fields and one output.

![Figure 3.5.3 – The fuzzy model.](http://lejpt.academicdirect.org/A15/001_018.htm)
Figure 3.5.4 – Figure showing the imported training data consisting of 1200 data points.

Figure 3.5.5 – Figure displaying the membership functions.

Figure 3.5.6 – Figure displaying the training error. A quick and efficient convergence was noticed during learning.

Figure 3.5.7 – The quality of the fit is seen as the red trajectory perfectly fits the blue.
Discussion
We used both fuzzy and non-fuzzy models to predict a time series. Both procedures produced correct results. Only minor differences can be noticed. Still, considering these minor differences and the stability found during various learning sessions we can conclude that the fuzzy approach is less stable than the crisp predictor. This could be expected by the nature of the prediction being based on the previous predicted steps which make the prediction in case of noise, chaotic.

![ANFS Prediction](image1)

**Figure 3.5.8 –** Fuzzy model prediction: the predicted steps are shown in blue.

![Non-fuzzy model prediction](image2)

**Figure 3.5.9 –** Non-fuzzy model prediction: the predicted steps are shown in brown.
Conclusion and Possible Areas for Future Research

This project started by introducing the general tools and models that we treat time series with, with the main target to make forecasts for them when we have to cope with linear cases. Then the main objective of the project was to show how we can cope with cases that the underlying dynamics are nonlinear, we introduce a popular nonlinear tool, the neural network.

The feedforward neural network used based on the MENFIS software, was trained with a supervised backpropagation technique and tested in many different aspects which were analysed earlier. When the neural network was used for the prediction of chaotic time series or when we fed it with the Santa Fe’s competition real world data the results proved highly accurate. Better results though, were obtained after the modification of the neural network in order to be applicable to many non-linear models such as those suggested by Ma & Wei.

The NARMA based neural network succeeded in producing a stable prediction and converged faster as well. This is due to the polynomials which helped the neural network to learn better the non-linear nature of the time series.

Finally, we introduce the basic principles of fuzzy logic and we use a fuzzy inference technique for the prediction of chaotic time series. Even though, the results were not that different from the previous description, they proved less stable.

Possible future research may deal with many different aspects of the neural network. Someone can study: the effect of having multiple output fields in order to predict further on, instead of running one output prediction; or the effect of using an activation function other than the logistic function that we used in this study. Studying learning disabilities, ways of normalising data or learning evaluation graphs to depict the quality of learning achieved might also be a challenging approach.
Bibliography


Sfyris, D. (2010). MENFIS. PhD Candidate of the Section of Mathematics, Department of Engineering, Democritus University of Thrace.


Appendix A  Gradient Descent Method

In section 2.4.1.2 we implemented a variation of the gradient descent method as presented in the paper “Research on Nonlinear Models of Time Series” by Ma Ni and Wei Gang, Journal of Electronics, Vol.16, No.3, July 1999. This algorithm was implemented in C++. The code is the following:

```c++
#include <cstdlib>
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <stdio.h>
#include <math.h>
using namespace std;

int icode; //example number
double w0x; //initial weight x
double w0y; //initial weight y

int errorcode; //error code
double lambda0; //initial step length
double K0;

//thresholds
double ks1;
double ks2;
double ks3;
double ks4;

double outx; //minimum at position x
double outy; //minimum at position y
double outz; //value of minimum

double Jw(int icode, double x, double y)
{
    float pi = 3.141592;

    if (icode==0) {
        //F(x,y) = sin((x+Pi)^2)+cos((y+Pi)^2)
        return sin(pow(x+pi,2))+cos(pow(y+pi,2));
    }

    if (icode==1) {
        //F(x,y) = (x^2-y)^2+(1-x)^2+(1-y)^2
        return pow(x*x-y,2)+pow(1-x,2)+pow(1-y,2);
    }
}

double partialx(int icode, double x, double y)
{
    float pi = 3.141592;

    if (icode==0) {
        if (icode==0) {
            //Dx = cos((x+pi)^2)*(2*(x+pi))
            return cos(pow(x+pi,2))*(2*(x+pi));
        }
    }

    if (icode==1) {
        if (icode==1) {
            //Dx = cos((x+pi)^2)*(2*(x+pi))
            return cos(pow(x+pi,2))*(2*(x+pi));
        }
    }
}
```
Nonlinear Time Series Forecasting using Neural Networks and Fuzzy System Techniques

```c
//Dx = (2*(x^2-y))*(2*x)+(2*(1-x))*(-1)
return (2*(x*x-y))*(2*x)+(2*(1-x))*(-1);
}

double partialy(int icode, double x, double y)
{
    float pi = 3.141592;

    if (icode==0) {
        //Dy = (-sin((y+pi)^2))*(2*(y+pi))
        return (-sin(pow(y+pi,2)))*(2*(y+pi));
    }

    if (icode==1) {
        //Dy = (2*(x^2-y))*(-1)+(2*(1-y))*(-1)
        return (2*(x*x-y))*(-1)+(2*(1-y))*(-1);
    }
}

int descent(int icode, double x0, double y0, double lambda0, double K0)
{
    double lambda;
    double grad0x;
    double grad0y;
    double grad1x;
    double grad1y;
    double s0x;
    double s0y;
    double sx;
    double sy;
    double w0x;
    double w0y;
    double w1x;
    double w1y;
    double w2x;
    double w2y;
    double M;
    double K;
    double D;
    double TAR;

    int iter=0;  //iterations counter
    int maxiter = 1000;  //not many iterations allowed
    int errorcode;  //0 if success, otherwise 1

    step1:  //set initial weights and step length
    w0x = x0;
    w0y = y0;
```

Margarita Papadopoulou
\[
\lambda = \lambda_0;
\]

**step2:** //compute gradient at initial point
\[
\begin{align*}
\text{grad0x} &= \text{partialx}(\text{icode}, w_{0x}, w_{0y}) ; \\
\text{grad0y} &= \text{partialy}(\text{icode}, w_{0x}, w_{0y}) ;
\end{align*}
\]

//initial direction is set to be descending
\[
\begin{align*}
s_{0x} &= -\text{grad0x} ; \\
s_{0y} &= -\text{grad0y} ;
\end{align*}
\]

**step3:** //target point 1
\[
\begin{align*}
w_{1x} &= w_{0x} + \lambda s_{0x} ; \\
w_{1y} &= w_{0y} + \lambda s_{0y} ;
\end{align*}
\]

//gradient at target point 1
\[
\begin{align*}
\text{grad1x} &= \text{partialx}(\text{icode}, w_{1x}, w_{1y}) ; \\
\text{grad1y} &= \text{partialy}(\text{icode}, w_{1x}, w_{1y}) ;
\end{align*}
\]

//commented as above
\[
\begin{align*}
sx &= -\text{grad1x} ; \\
sy &= -\text{grad1y} ;
\end{align*}
\]

\[
\text{iter} = \text{iter} + 1; //count iterations just in case of no convergence
\]

if (iter>maxiter) 
{
    \text{errorcode} = 1;
    \text{goto exit};
}

**step4:**
\[
\begin{align*}
M &= 0; \\
K &= K_0; \\
\text{TAR} &= Jw(\text{icode}, w_{1x}, w_{1y});
\end{align*}
\]

**step5:**
\[
\begin{align*}
M &= M + K; \\
sx &= s_{0x} + M*sx; \\
sy &= s_{0y} + M*sy; \\
w_{2x} &= w_{0x} + \lambda sx; \\
w_{2y} &= w_{0y} + \lambda sy;
\end{align*}
\]

\[
\text{iter} = \text{iter} + 1;
\]

if (iter>maxiter) 
{
    \text{errorcode} = 1;
    \text{goto exit};
}

**step6:**
\[
\text{if (Jw(icode, w_{2x}, w_{2y}) < Jw(icode, w_{1x}, w_{1y}))}
\]
\[
\begin{align*}
w_{1x} &= w_{2x}; \\
w_{1y} &= w_{2y}; \\
K &= 2*K; \\
\text{goto step5};
\end{align*}
\]
Nonlinear Time Series Forecasting using Neural Networks and Fuzzy System Techniques

//check thresholds
if (((Jw(icode,w2x,w2y) > Jw(icode,w1x,w1y)) && (fabs(K) < ks1)) || (fabs(M) > ks2))
{
    goto step7;
}

K = -K/3;
goto step5;

step7:
D = (TAR-Jw(icode,w1x,w1y))/Jw(icode,w0x,w0y);
if (D > ks3)
{
    grad1x = partialx(icode,w1x,w1y);
    grad1y = partialy(icode,w1x,w1y);
    sx = -grad1x;
    sy = -grad1y;
    goto step4;
}
if (D < ks3)
{
    goto step8;
}

step8:
if (Jw(icode,w1x,w1y)<Jw(icode,w2x,w2y))
{
    w0x = w1x;
    w0y = w1y;
    goto step2;
}
if (lambda > ks4)
{
    lambda = lambda / 2;
}
if (lambda < ks4)
{
    errorcode = 0;
    outx = w1x;
    outy = w1y;
    outz = Jw(icode,w1x,w1y);
    goto exit;
}

errorcode = 1;
exit:
return errorcode;

}

int main(int argc, char *argv[])
{
    //initial step length
    lambda0 = 0.01;
    //initialize K
    K0 = 1;
    //thresholds
    ks1 = 4;
    ks2 = 0.1;
    ks3 = 0.15;
    ks4 = 1;

    //All cases produce identical minima, no matter the initial conditions
    cout << "EXAMPLE FUNCTION sin((x+Pi)^2)+cos((y+Pi)^2)" << endl;
    cout << "============================================" << endl;
    //example 1, case 1
    //=================
    icode = 0;
    //initial weights
    w0x = 0.25;
    w0y = 0.25;
    errorcode = descent(icode,w0x,w0y,lambda0,K0);
    if (errorcode==0)
    {
        cout << "CASE 1, INITIAL WEIGHTS " << w0x << " , " << w0y << endl;
        cout << "Minimum found at X = " << outx << ", Y = " << outy << endl;
        cout << "Minimum value = " << outz << endl;
    }
    if (errorcode==1)
    {
        cout << "FAILED" << endl;
    }

    //example 1, case 2
    //=================
    icode = 0;
    //initial weights
    w0x = 0.4;
    w0y = 0.4;
    errorcode = descent(icode,w0x,w0y,lambda0,K0);
    if (errorcode==0)
    {
        cout << "CASE 2, INITIAL WEIGHTS " << w0x << " , " << w0y << endl;
        cout << "Minimum found at X = " << outx << ", Y = " << outy << endl;
        cout << "Minimum value = " << outz << endl;
    }
}
if (errorcode==1)
{
    cout << "FAILED" << endl;
}

//===example 1, case 3
//===initial weights
icode = 0;
w0x = 0.55;
w0y = 0.35;
errorcode = descent(icode,w0x,w0y,lambda0,K0);

if (errorcode==0)
{
    cout << "CASE 3, INITIAL WEIGHTS " << w0x << " , " << w0y << endl;
    cout << "Minimum found at X = " << outx << ", Y = " << outy << endl;
    cout << "Minimum value = " << outz << endl;
}
if (errorcode==1)
{
    cout << "FAILED" << endl;
}

cout << "" << endl;
cout << "EXAMPLE FUNCTION F(x,y) = (x^2-y)^2+(1-x)^2+(1-y)^2" << endl;
cout << "============================================" << endl;

//===example 1, case 1
//===initial weights
icode = 1;
w0x = 2.25;
w0y = 2.25;
errorcode = descent(icode,w0x,w0y,lambda0,K0);

if (errorcode==0)
{
    cout << "CASE 1, INITIAL WEIGHTS " << w0x << " , " << w0y << endl;
    cout << "Minimum found at X = " << outx << ", Y = " << outy << endl;
    cout << "Minimum value = " << outz << endl;
}
if (errorcode==1)
{
    cout << "FAILED" << endl;
}

//===example 1, case 2
//===initial weights
icode = 1;
w0x = 2.4;
w0y = 2.4;
errorcode = descent(icode,w0x,w0y,lambda0,K0);

if (errorcode==0)
Nonlinear Time Series Forecasting using Neural Networks and Fuzzy System Techniques

{ cout << "CASE 2, INITIAL WEIGHTS " << w0x << " , " << w0y << endl;  
cout << "Minimum found at X = " << outx << " , Y = " << outy << endl;  
cout << "Minimum value = " << outz << endl; 
}

if (errorcode==1) 
{
    cout << "FAILED" << endl;
}

//example 1, case 3 
//=================
icode = 1;  
//initial weights 
 w0x = 2.55; 
 w0y = 2.35; 
errorcode = descent(icode,w0x,w0y,lambda0,K0);

if (errorcode==0)
{
    cout << "CASE 3, INITIAL WEIGHTS " << w0x << " , " << w0y << endl;  
cout << "Minimum found at X = " << outx << " , Y = " << outy << endl;  
cout << "Minimum value = " << outz << endl; 
}

if (errorcode==1)
{
    cout << "FAILED" << endl;
}

system("PAUSE");  
return EXIT_SUCCESS; 
}
Appendix B  MENFIS Software

In paragraph 2.7 based on Dudul’s idea (2005) a feedforward neural network that was developed using the program MENFIS (Maximum Entropy Neural Fuzzy Inference System) developed by Dimitris Sfyris (2010) was trained by a backpropagation technique. MENFIS is a back propagation neural network designed to process a variety of linear and non-linear problems. Any number of input fields, output fields, neurons per hidden layer, number of hidden layers can be modelled, as well as a variety of activation functions. The default activation function is the continuous log-sigmoid function. As standard procedure, the training examples must be prepared and appropriately normalized to have a standard range of values for better results. MENFIS supports both ARMA and NARMA models.
Appendix C  Preprocessor

In paragraph 2.10 the feedforward neural network was extended to be applicable to many non-linear models such as those suggested by Ma & Wei (1999). Here we provide the code of the pre-processor which normalizes the time series and prepares the training data according to the desired number of input and output fields as well as the desired NARMA parameters. Then the set of training examples is shuffled before it is fed to the neural network. The neural network that was developed, reads the shuffled training examples, and after learning it outputs the predictions.

```cpp
#include <cstdlib>
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <stdexcept>
#include <vector>
#include <math.h>

using namespace std;

int MAXINPUTSIZE = 20;          // Maximum number of input fields
int MAXOUTPUTSIZE = 20;         // Maximum output layer size
int MAXEXAMPLES = 20000;        // Maximum number of training examples

char* input_filename = "timeseries.dat";          // Default input file name
char* traindata_filename = "traindata.dat";       // Default training data set
int numofinputs = 10;                             // Default number of inputs
int numofoutputs = 1;                             // Default number of outputs
int numofexamples = 0;                            // Number of training examples
int maxexampleid = 0;                             // Maximum example id
int normalizeflag = 0;                             // Normalize if value = 1
int shuffleflag = 1;                              // Shuffle if value = 1
int p = 10;                                       // Parameter of AR
int q = 0;                                        // Parameter of MA
int r1 = 0;                                       // Parameter of polynomial AR
int r2 = 0;                                       // Parameter of polynomial MA

vector<double> dseries(30000);                    // Input data series
int dataseriescount = 0;                          // Length of data series

vector<double> tseries(30000);                    // Time series data (possibly normalized)
int timeseriescount = 0;                          // Length of time series

vector<vector<double> >                    nv examples(MAXEXAMPLES,
vector<double>(MAXINPUTSIZE+MAXOUTPUTSIZE));

typedef vector<std::string> TGridExamples;       // Define a class vector of examples
TGridExamples grexamples;                       // Dynamic array of examples
TGridExamples shexamples;                       // Dynamic array of shuffled examples

class BadConversion : public std::runtime_error {
public:
```
BadConversion(std::string const& s)  
 : std::runtime_error(s)  
 {  
 };

inline std::string stringify(double x)  
{  
    std::ostringstream o;  
    if (!(o << x))  
        throw BadConversion("stringify(double)");  
    return o.str();  
}

int check_parameters(int argc, char *argv[])  
{  
    std::string param1,param2;     //temporary string parameters  
    int errcode = 0;  
    for (int i=1; i<argc; i+=2)  
    {  
        param1 = argv[i];  
        //Check for acceptable switch  
        if (((param1=="-f")  
             and (param1=="-p")  
             and (param1=="-q")  
             and (param1=="-r1")  
             and (param1=="-r2")  
             and (param1=="-i")  
             and (param1=="-j")  
             and (param1=="-t")  
             and (param1=="-r")  
             and (param1=="-a"))  
        {  
            errcode = 1;  
            break;  
        }  
        //Check for last missing switch value  
        if (i+1>argc-1){  
            errcode = 2;  
            break;  
        }  
    }  
    param2 = argv[i+1];  
    //Check for all but last missing switch value  
    if (((param1=="-f")  
         and (param1=="-p")  
         and (param1=="-q")  
         and (param1=="-r1")  
         and (param1=="-r2")  
         and (param1=="-i")  
         and (param1=="-j")  
         and (param1=="-t")  
         and (param1=="-r")  
         and (param1=="-a"))  
    {  
        errcode = 2;  
        break;  
    }  
}
if (errcode==1)
{
    cout << "Invalid switch." << endl;
}
if (errcode==2)
{
    cout << "Missing switch value." << endl;
}
return errcode;
}

int random(int maxvalue)
{
    int rannum;
    bool found = 1;
    long sec;
    time( &sec);
    srand((unsigned)sec);

    while (found == 1) { // use loop to look for desired number
        rannum = rand();
        if ( rannum >= 0 && rannum < 2000) { // turn off the loop when number is found
            found = 0;
        }
    }
    return rannum;
}

void makeexamplesNARMA(int p, int q, int r1, int r2)
{
    int k = 0;
    int iexample = 0;

    iexample=0;
    maxexampleid=0;
    for (int i=p+numofoutputs; i<=timeseriescount; i++)
    {
        iexample+=1;
        k=0;
        nnexamples[iexample][k] = i; //Example ID in position 0
        if (i > maxexampleid) { maxexampleid = i; }

        for (int j=1; j<=p; j++)
        {
            k+=1;
            nnexamples[iexample][k] = tseries[i-p-numofoutputs+k];
        }
        for (int j=1; j<=q; j++)
        {
            k+=1;
            nnexamples[iexample][k] = random(2000)/1000-1;
        }
        for (int j=1; j<=r1; j++)
        {
            k+=1;
        }
    }
nnexamples[iexample][k] = pow(tseries[i-p-numofoutputs+1],j+2);
}
for (int j=1; j<=r2; j++)
{
    k+=1;
    nnexamples[iexample][k] = pow(random(2000)/1000-1,j+2);
}
for (int j=1; j<=numofoutputs; j++)
{
    k+=1;
    nnexamples[iexample][k] = tseries[i-p-numofoutputs+k-q];
}
}
numofexamples = iexample;
}

void shuffleexamples( void )
{
    std::string s;    // Temporary string
    for (int i=1; i<=numofexamples; i++)
    {
        s = "";
        s = s + stringify(nnexamples[i][0]); // begin with example id
        for (int j=1; j<=numofinputs; j++)
        {
            s = s+" "+stringify(nnexamples[i][j]);
        }
        for (int j=1; j<=numofoutputs; j++)
        {
            s = s+" "+stringify(nnexamples[i][numofinputs+j]);
        }
        shexamples.push_back(s);
    }
    random_shuffle( shexamples.begin(), shexamples.end() );
}

void gridexamples( void )
{
    std::string s;    // Temporary string
    for (int i=1; i<=numofexamples; i++)
    {
        s = "";
        s = s + stringify(nnexamples[i][0]); // begin with example id
        for (int j=1; j<=numofinputs; j++)
        {
            s = s+" "+stringify(nnexamples[i][j]);
        }
        for (int j=1; j<=numofoutputs; j++)
        {
            s = s+" "+stringify(nnexamples[i][numofinputs+j]);
        }
        grexamples.push_back(s);
    }
}
void saveexamples( void )
{
    // Save training data to file
    ofstream outfile;
    outfile.open (traindata_filename, ios::out);
    for (int i=0; i<numofexamples; i++)
    {
        if (shuffleflag==1)
        {
            if (i<numofexamples-1) { outfile << shexamples[i] << endl; }
            else { outfile << shexamples[i]; }
        }
        else
        {
            if (i<numofexamples-1) { outfile << grexamples[i] << endl; }
            else { outfile << grexamples[i]; }
        }
    }
    outfile.close();
}
void onetoone( void )
{
    timeseriescount = dataseriescount;
    for (int i=1; i<=timeseriescount; i++)
    {
        tseries[i] = dseries[i];
    }
}
void normalize( void )
{
    // Normalize LORENZ
    timeseriescount=dataseriescount;
    for (int i=2; i<=dataseriescount; i++)
    {
        tseries[i-1] = dseries[i]/20;
    }
}
int main(int argc, char *argv[])
{
    int errcode = check_parameters(argc, argv);
    if (errcode!=0) { return 0; }
    std::string s;       // Temporary string
    for (int i=1; i<argc; i++)
    {
        s = argv[i];
        if (s=="-f") { input_filename = argv[i+1]; }
        if (s=="-p") { numofinputs = atoi(argv[i+1]); }
        if (s=="-q") { numofoutputs = atoi(argv[i+1]); }
        if (s=="-r1") { numofinputs = atoi(argv[i+1]); }
        if (s=="-r2") { numofoutputs = atoi(argv[i+1]); }
        if (s=="-i") { numofinputs = atoi(argv[i+1]); }
    }
if (s=="-j") { numofoutputs = atoi(argv[i+1]); }
if (s=="-t") { traindata_filename = argv[i+1]; }
if (s=="-r") { normalizeflag = atoi(argv[i+1]); }
if (s=="-a") { shuffleflag = atoi(argv[i+1]); }
}

cout << "ARGUMENTS IN USE " << endl;
cout << "Input filename -f " << input_filename << endl;
cout << "Parameter of AR -p " << p << endl;
cout << "Parameter of MA -q " << q << endl;
cout << "Parameter of polynomial AR -r1 " << r1 << endl;
cout << "Parameter of polynomial MA -r2 " << r2 << endl;
cout << "Number of output fields -j " << numofoutputs << endl;
cout << "Training data filename -t " << traindata_filename << endl;
cout << "Normalize flag -r " << normalizeflag << endl;
cout << "Shuffle flag -a " << shuffleflag << endl;
}

/* =========================
Read data from input file
======================== */

ifstream indata(input_filename);
if(!indata) { cerr << "Input file error" << endl; system("PAUSE"); exit(1); }

double val;
dataseriescount = 0;
while (!indata.eof()) { // Read until EOF
    indata >> val;
    dataseriescount += 1;
    dseries[dataseriescount] = val;
} indata.close();

onetoone();

numofinputs = p+q+r1+r2;
if (normalizeflag==1) { normalize(); }
makeexamplesNARMA(p,q,r1,r2);
if (shuffleflag==1) { shuffleexamples(); } else { gridexamples(); }
saveexamples();

system("PAUSE");
return EXIT_SUCCESS;
Appendix D  Construction of a Fuzzy Neural Network

In Chapter 3.5 we introduce a fuzzy neural network. We implemented it in Matlab using ANFIS which is analytically described in the main project. The code is the following:

```matlab
load default.sam
a = default;
numofexamples = length(a);
time=[1:1:numofexamples]';
x_t = a(:, 6);

% Prepare training data
trn_data(:, 1) = a(:, 2);
trn_data(:, 2) = a(:, 3);
trn_data(:, 3) = a(:, 4);
trn_data(:, 4) = a(:, 5);
trn_data(:, 5) = a(:, 6);
trn_data(:, 6) = a(:, 7);

numMFs = 2;  % Number of MFs
mfType = 'gaussmf';   % 'gbellmf';
epoch_n = 2;     
in_fis = genfis1(trn_data, numMFs, mfType);
out_fis = anfis(trn_data, in_fis, epoch_n);

inp = trn_data(:, 1:5);
anfis_output = evalfis(input, out_fis);

% Predictions
v = trn_data(numofexamples, 1:6);
for j=2:6,
    v(j-1) = v(j);
end

npred = 150;
for i=1:npred,
    vinp = v(1:5);
pred(i) = evalfis(vinp, out_fis);
    v(6) = pred(i);
    fprintf(1, 'Prediction %d: %6.4f
', i, pred(i));
    for j=2:6,
        v(j-1) = v(j);
    end
end

figure;
plot(time, x_t);
xlabel('Time (sec)'); ylabel('x(t)');
title('Lorenz Chaotic Time Series');

figure;
for input_index=1:20,
    subplot(3,3,input_index)
    [x,y]=plotmf(in_fis,'input',input_index);
    plot(x,y)
    axis([-inf inf 0 1.2]);
    xlabel(['Input ' int2str(input_index)]);
```

In Chapter 3.5 we introduce a fuzzy neural network. We implemented it in Matlab using ANFIS which is analytically described in the main project. The code is the following:
title('Membership Functions');

figure;
plot(time, x_t);
xlabel('Time (sec)'); ylabel('x(t)');
hold on;

index = 1:length(a);
plot(time(index), [x_t(index) anfis_output], '--s', ...
    'MarkerEdgeColor','k',...
    'MarkerFaceColor','g',...
    'MarkerSize',3);

index = numofexamples+1:numofexamples+npred;
index2 = 1:npred;
time=[1:1:numofexamples+npred]';
plot(time(index), pred(index2));
title('ANFIS Prediction');