Cape Peninsula
University of Technology

INVESTIGATION OF ARTIFICIAL NEURAL NETWORKS FOR MODELING, IDENTIFICATION AND CONTROL OF NONLINEAR PLANT

by

JULIUS N'GON'GA MUGA

Thesis submitted in fulfilment of the requirements for the degree

Master of Technology: Electrical Engineering

in the Faculty of Engineering

at the Cape Peninsula University of Technology

Supervisor: Professor Raynitchka Tzoneva
Co-supervisor: Mr. Carl Kriger

Bellville
December 2009

DECLARATION

I, Julius N'gon'ga Muga, declare that the contents of this dissertation/thesis represent my own unaided work, and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed **Date**

ABSTRACT

In real world systems such as the waste water treatment plants, the nonlinearities, uncertainty and complexity play a major role in their daily operations. Effective control of such systems variables requires robust control methods to accommodate the uncertainties and harsh environments. It has been shown that intelligent control systems have the ability to accommodate the system uncertain parameters. Techniques such as fuzzy logic, neural networks and genetic algorithms have had many successes in the field of control because they contain essential characteristics needed for the design of identifiers and controllers for complex systems where nonlinearities, complexity and uncertainties exist.

Approaches based on neural networks have proven to be powerful tools for solving nonlinear control and optimisation problems. This is because neural networks have the ability to learn and approximate nonlinear functions arbitrarily well. The approximation capabilities of such networks can be used for the design of both identifiers and controllers. Basically, an artificial neural network is a computing architecture that consists of massively parallel interconnections of simple computing elements that provide insights into the kind of highly parallel computation that is carried out by biological nervous system. A large number of networks have been proposed and investigated with various topological structures, functionality and training algorithms for the purposes of identification and control of practical systems.

For the purpose of this research thesis an approach for the investigation of the use of neural networks in identification, modelling and control of non-linear systems has been carried out. In particular, neural network identifiers and controllers have been designed for the control of the dissolved oxygen (DO) concentration of the activated sludge process in waste water treatment plants. These plants, being complex processes with several variables (states) and also affected by disturbances require some form of control in order to maintain the standards of effluent. DO concentration control in the aeration tank is the most widely used controlled variable. Nonlinearity is a feature that describes the dynamics of the dissolved oxygen process and therefore the DO estimation and control may not be sufficiently achieved with a conventional linear controller.

Neural networks structures are proposed, trained and utilized for purposes of identification, modelling and design of NN controllers for nonlinear DO control. Algorithms and programs are developed using Matlab environment and are deployed on a hardware PLC platform. The research is limited to the feedforward multilayer perceptron and the recurrent neural networks for the identification and control. Control models considered are the direct inverse

model control, internal model control and feedback linearizing control. Real-time implementation is limited to the lab-scale wastewater treatment plant.

ACKNOWLEDGEMENTS

I wish to thank:

- My wife Liz for the endurance, support, understanding and encouragement during all the time of study. Without her advice and moral support things would have not been the same.
- My supervisor, Professor Raynitchka Tzoneva for her supervision, guidance, advice and contribution throughout the thesis write up. To her, I say thank you very much.
- My employer the Mombasa Polytechnic University council for granting me the study leave and scholarship for this noble undertaking. I will always be grateful to the support I received during the course of study.
- Mr Carl Kriger in the Department of Electrical Engineering, Cape Peninsula University of Technology Bellville campus for the support, guidance and encouragement I received from him.
- My two sons, Frank and Yvess and step daughter Nedrose for the endurance during my long period absence from home. I love you so much.

DEDICATION

I wish to dedicate this thesis to my mother Flora, my wife Liz and my children Frank, Yvess and Nedrose for their endurance, patience and support during the long period I have been away.

TABLE OF CONTENTS

CHAPTER THREE: NEURAL NETWORKS THEORY AND OVERVIEW

CHAPTER FOUR: NN MODEL OF THE DO PROCESS - SYSTEM IDENTIFICATION.

LIST OF TABLES

GLOSSARY

Terms/Acronyms/Abbreviations	Definition/Explanation
ADALINE	ADAptive LInear Element
AI	Artificial Intelligent
ANN	Artificial Neural Network
ASM1	Activated Sludge Model number one
ART	Adaptive Resonance Theory
ASM	Activated Sludge Model
ASP	Activated Sludge Process
BOD	Biochemical Oxygen Demand
BNR	Biological Nutrient Removal
CMAC	Cerebellar Model Articulation Control network
COD	Chemical oxygen demand
COST EU	Project on optimal management of wastewater systems(action 624)
DAF	Dissolved Air Floatation
DIC	Direct Inverse Control
DO	Dissolved Oxygen
EPOCH	Training status information
GDP	Gross domestic product
GMDH	Group Method of Data Handling (GMDH) network
GP	Gaussian process model
IAWPRC	international Association on Water Pollution Research and Control
IAWQ	International Association on Water Quality
IMC	Inverse Model Control

I/O	Input – Output
LVQ	Learning Vector Quantization network
MATLAB	Matrix Laboratory
MIMO	Multi input multi output systems
MLP	Multi-layer Perceptron
MLSS	mixed liquor suspended solids
MSE	Mean Squared Error
N_2	*nitrogen gas*
NN	Neural Networks
PC	Personal Computer
pH	Amount of Acidity or Alkanity in the wastewater
PID	Proportional. Integral and Derivative
PLC	Programmable Logic Controller
RBF	Radial Basis Function
RMS	Root mean square
RNN	Recurrent Neural Networks
SBR	Sequencing Batch Reactor
SCADA	Supervisory Control and Data Acquisition
SIMULINK	Simulation and Link is an extension of Matlab by Math works Inc.
SISO	Single input Single output systems
SOFM	Self Organizing Feature Map
TKN	Total Kjeldahl Nitrogen
VFAs	Volatile fatty acids
VLSI	Very large scale integrated circuit
WWTP	Wastewater treatment plant

Mathematical notations

Symbols/ Abbreviations	Definition/Explanation
NO_2^-	Nitrite
NO_3^-	Nitrates
$S_O(t)$	DO concentration
WAS	Waste actived sludge
K_1 and K_2	Estimated parameters in the, K_La function
K_La	Oxygen mass transfer coefficient (h_1)
$S_{O\,sat}$	Saturated DO concentration
$u(t)$	Airflow rate to the aeration tank
K_L	An absorption coefficient in the K_La function
a	An area/volume ratio in the K_La function
$S_{O\,in}(t)$	The dissolved oxygen concentration in the input flow
$Q(t)$	Influent flow rate
V	The volume of waste water tank
$r_{so}(t)$	Oxygen Uptake Rate
S_I	Inert organic material (ASM1)
S_S	Readily biodegradable substrate (ASM1)
X_I	Particulate inert organic matter (ASM1)
X_S	Slowly biodegradable substrate (ASM1)
$X_{B,H}$	Active heterotrophic biomass (ASM1)
$X_{B,A}$	Active autotrophic biomass (ASM1)
X_P	Particulate products arising from biomass decay (ASM1)
S_O	Dissolved oxygen concentration (ASM1)

N_1^i, N_2^i and N_3^i	Input, hidden and output layer. Superscript i denotes the identification while 1,2 and 3 stand for the input layer, hidden layer and output layer.
S_{NO}	Nitrate and nitrite concentration (ASM1)
S_{NH}	The soluble ammonium nitrogen concentration (ASM1)
S_{ND}	Biodegradable organic nitrogen (ASM1)
X_{ND}	Particulate biodegradable organic nitrogen (ASM1)
$i1\_1(t-k), k = 0...2$, z(t - m), m = 0...2,	$i1\_1$ is the normalized airflow rate, z is the time trajectory.
$i2\_2(t-n), n = 0...2.$	The normalized output of the DO process
y^p	plant output
y^m	the neural output
d and d′	Plant disturbances.
S_{ALK}	Alkalinity (ASM1)
ko	The first delay value of $i1\_1$,
$k1$	The second delay value of $i1\_1$,
$k2$	Third delay value of $i1\_1$.
z	The symbol of time value
no	The first delay value of $i2\_2$
$n1$	is the second delay value of $i2\_2$
$n2$	The third delay value of $i2\_2$.
P	The range of network input vector
T	The range of network target output vector.
P1, T1	Represents the variables of the normalized inputs and targets for the feedforward network
f^{-1}	Inverse function
J	An optimization algorithm to approximate the model.
S_O^{sp}	

$S_O(k)$	Dissolved oxygen concentration set-point Dissolved oxygen concentration
$u(k-1)$	First delayed value of Airflow rate to the aeration tank
$u(k-2)$	Second delayed value of Airflow rate to the aeration tank Airflow rate to the aeration tank
$u(k-3)$	Third delayed value of Airflow rate to the aeration tank Airflow rate to the aeration tank
Y_A	Autotrophic yield (ASM1)
Y_H	Heterotrophic yield (ASM1)
η_g	Correction factor for anoxic growth of heterotrophs (ASM1)
η_h	Correction factor for anoxic hydrolysis (ASM1)
μA	Autotrophic max. specific growth rate (ASM1
μH	Heterotrophic max. specific growth rate (ASM1)
bA	Autotrophic decay rate (ASM1)
b_H	Heterotrophic decay rate (ASM1)
f_P	Fraction of biomass yielding particulate products (ASM1)
i_{XB}	Mass N/mass COD in biomass (ASM1)
i_{XP}	Mass N/mass COD in products from biomass (ASM1)
ka	Ammonification rate (ASM1)
kh	Max. specific hydrolysis rate (ASM1)
K_{NH}	Ammonium half saturation constant for autotrophs (ASM1
K_{NO}	Nitrate half saturation constant for heterotrophs (ASM1)
$K_{O,A}$	Oxygen half saturation constant for autotrophs (ASM1)
$K_{O,H}$	Oxygen half saturation constant for heterotrophs (ASM1)
K_S	Half saturation constant for heterotrophs (ASM1

	Dissolved oxygen concentration set-point
$S_O(k)$	Dissolved oxygen concentration
$u(k-1)$	First delayed value of Airflow rate to the aeration tank
$u(k-2)$	Second delayed value of Airflow rate to the aeration tank Airflow rate to the aeration tank
$u(k-3)$	Third delayed value of Airflow rate to the aeration tank Airflow rate to the aeration tank
Y_A	Autotrophic yield (ASM1)
Y_H	Heterotrophic yield (ASM1)
η_g	Correction factor for anoxic growth of heterotrophs (ASM1)
η_h	Correction factor for anoxic hydrolysis (ASM1)
μA	Autotrophic max. specific growth rate (ASM1
μH	Heterotrophic max. specific growth rate (ASM1)
bA	Autotrophic decay rate (ASM1)
b_H	Heterotrophic decay rate (ASM1)
f_P	Fraction of biomass yielding particulate products (ASM1)
i_{XB}	Mass N/mass COD in biomass (ASM1)
i_{XP}	Mass N/mass COD in products from biomass (ASM1)
ka	Ammonification rate (ASM1)
kh	Max. specific hydrolysis rate (ASM1)
K_{NH}	Ammonium half saturation constant for autotrophs (ASM1
K_{NO}	Nitrate half saturation constant for heterotrophs (ASM1)
$K_{O,A}$	Oxygen half saturation constant for autotrophs (ASM1)
$K_{O,H}$	Oxygen half saturation constant for heterotrophs (ASM1)
K_S	Half saturation constant for heterotrophs (ASM1

Q_{RAS}	Return sludge flow rate
Q_w	Wastage flow rate
Q_{int}	Internal recirculation flow rate

CHAPTER ONE
INTRODUCTION

1.1 Awareness of the problem

Research in non-linear control is motivated by the inherently non-linear characteristics of the dynamic systems we often try to control. In reality, virtually many practical systems are significantly non-linear in nature so that the important features of their performance may be completely overlooked if they are analysed and designed through linear control techniques. Nevertheless, linear controller design methods have proved to be adequate in many applications.

In recent years many authors became interested in how to asses the nonlinearity problems of general dynamic systems and many different approaches can be found in the following literatures (Desoer, and Wang, 1980(2); Astorga, 1992; Nikolau, and Mannousionthakis, 1989; Schweickhardt, and Allgower, 2005; Kihas, and Marquez, 2004; Engvist, and Ljung, 2002(a), 2004(b)) etc.

The system is non-linear if the principle of superposition is not fulfilled for at least one of its elements. All methods for the analysis of linear systems and the Laplace transformation are not applicable to such system. There is no equivalent mathematical theory, on which base a full theory of non-linear systems of control can be built. That is why engineers use local linear approximations of the nonlinear systems to develop control laws (Guay *et al*, 2005). But this approach can be used only for "weak" non linearities and small deviations from the steady state. In this way it is necessary for the system to be analysed and designed totally like a non-linear system. Such methods like: the phase plane method, the describing function technique and the Lyapunov stability criterion are often used for the analysis. However, nonlinear controllers have some drawbacks when compared to linear controllers due to the increased complexity introduced by the non linearity of the model. Additionally, the investigation of control-relevant properties of nonlinear systems can be extremely time-consuming because no general framework has yet been evolved, (Hahn, and Marquardt, 2003).

Effective control of real world applications requires robust methods to accommodate system uncertainties and harsh environments. Intelligent control systems have the ability to accommodate these uncertain parameters. Techniques such as fuzzy logic, neural networks and genetic algorithms have had many successes in the field of control. In recent years a great deal of attention has been paid to the application of neural networks to the modelling and identification of dynamic processes and time

series prediction. The novelty about neural networks lies in their ability to learn and *approximate non-linear functions arbitrary well. The control design is then based on* neural network model rather than the actual system. This therefore provides a possible way of modelling complex non- linear functions. A large number of networks have been proposed and investigated with various topological structures, functionality and training algorithms for the purposes of control. Previous applications are available in the works of Psalties, *et al* (1988); Zerman, *et al* (1989); Li, and Slotine, (1989); Narendra, and Parthsarathy, (1990); Funahasi, (1989); Cybenko, (1985); Hornic, *et al* (1989); Hecht-Nielsen, 1990).

In this research neural networks are utilized for purposes of identification, modelling and design of NN controllers for a nonlinear system. Algorithms and programs are developed using Matlab environment for deployment on a hardware platform to control DO concentration of the activated sludge process of a wastewater treatment plant in real-time.

1.2 Description of Nonlinear systems

The most fundamental property of a linear system from the control point of view is the validity of the principle of superposition. It is on account of this property that it can be guaranteed that a linear system designed to perform satisfactorily when excited by a standard test signal, will exhibit satisfactory behaviour under any circumstance. In *contrast to the linear case, the response of nonlinear systems to a particular test signal* is no guide to their behaviour to a standard input, since the superposition principle does not hold. In fact, a nonlinear system giving its best response for a certain step input may exhibit highly unsatisfactory behaviour when the input amplitude is changed. Further, the Laplace and z-transforms which are simple and powerful tools of linear theory are inapplicable and hence the analysis of nonlinear systems for different time-varying inputs is rendered quite difficult. (Nagrath, and Gopal, 1982)

In control systems, nonlinearities can be classified as *incidental* and *intentional.* Incidental nonlinearities are those which are inherently present in the system. The designer strives to design the system so as to limit the adverse effects of these nonlinearities. Common example of these nonlinearities are saturation, dead-zone, coulomb friction, stiction, backlash, etc. Research project targets these types of nonlinearities which are common in many practical systems. The intentional nonlinearities on the other hand, are those which are deliberately inserted in the system to modify system characteristics. The most common of this type of nonlinearity is a relay. The relay is a nonlinear amplifier which can provide large power

amplification inexpensively and is therefore deliberately introduced in control systems and find wide applications in the field of control. When the nonlinear component (relay) is specially introduced into a feedback loop it is to act as a controller or part of the controller. In this research the nonlinear plant under consideration is the dissolved oxygen concentration of the activated sludge process in the WWTP. It falls in the group of the incidental nonlinearities.

1.2.1 General approach for analysis of the nonlinear systems

Nonlinear control theory is based on methods used in physics, in particular mechanics, to describe the nonlinear phenomena. There is no single technique for analysis which is suitable for all nonlinear systems.

Consider a general time-invariant system described by the state equations:

$$\dot{x}(t) = f[x(t), u(t)]$$
$$y(t) = h[x(t)] \qquad (1.1)$$

where, $u(t)$, $x(t)$, and $y(t)$ represent, respectively the input, state and the output of the system at time t, and f, h, are the nonlinear functions. In the control problem the object is to determine the input $u(t)$ so that the system behaves in a desired fashion. As mentioned at the introduction, even if the nonlinear functions f and h are known, the solution to equation 1 is not quite so easy unless certain assumptions concerning the controllability of the system and hence f and h are made. A number of methods for nonlinear system analysis have been applied with success. The methods used most for stability assessment are:

1.2.2 Phase plane Analysis

This method is based on a graphical representation of a given system response which aids in understanding of the nature of the dynamic behaviour. The state space approach is used with state space vector equal to the output (or error) and its first derivative. The term phase is equivalent to state. The main points of this method are as listed (Torkel, and Ljung, 2000).

- It is possible to get a good idea of the properties of a nonlinear system close to equilibrium by studying the eigenvalues and eigenvectors of the linear approximation when the results can be plotted.
- Two real distinct eigenvalues of the same sign give a tangent node, stable for negative values, unstable for positive ones.
- Complex eigenvalues with nonzero real parts give a focus, stable for negative real parts, unstable for positive ones.
- Real eigenvalues of opposite signs give a saddle point.

1.2.3 Technique for describing functions

This is an extension of the frequency response approach used to predict whether limit cycles will occur, and if so, to estimate their amplitude and frequency and facilitate design to avoid or minimise them. When the nonlinear element is forced by a sinusoidal input function the steady state output is periodic with the same period as the forcing function, but not sinusoidal. The output can then be represented by a Fourier series consisting of a sine wave of the fundamental frequency together with a number of harmonic components at integer multiples of the fundamental frequency. As this signal passes round the loop the linear elements act as a low pass filter and attenuate the higher frequency components (whose amplitudes are usually smaller than that of the fundamental) to a greater extent than the fundamental. The signal returning to the linearity is thus little different to what it would be if the output of the non-linearity had simply been the fundamental component.

1.2.4 Lyapunov stability criterion.

Lyapunov second method of stability assessment is based on energy considerations, which does not require solution of the systems equations. It involves searching for a function of Lyapunov, which satisfies certain requirements. In the context of non-linear systems, several classes of stability can be defined. For a system described by state equations, and with a singularity in the state space where the system is at rest, then the system is said to be stable if for a given disturbance every trajectory remains within a certain defined region around the singularity, and asymptotically stable if it returns to rest at the singularity. There is local stability if the trajectories remain within the small region only after small perturbations, finite stability if they return from all points within the entire state space. A system in which trajectories end on a limit cycle within the region is defined as stable in this sense, although it is not asymptotically stable. A linear system, if stable, could be said to be globally asymptotically stable.

1.2.5 Control of Waste Water Treatment plants

Wastewater treatment plants, being complex processes with several nonlinear variables (states) and also affected by disturbances require some form of control in order to maintain the standards of effluent. The complexity of the treatment process arises due to the fact that significant perturbations in flow and load together with variations in the composition of the incoming wastewater are normally difficult to predict and control. From a control engineer's point of view, the activated sludge plant is the most important of the biochemical processes for wastewater treatment because of the big number of the state variables and sub-processes describing the reactions that take place inside the reactors leading to carbon and nitrogen removal. This gives

an insight on the type of control strategy to be designed for the process because control can then reduce the externally perceived complexity of the plant.

The dissolved oxygen concentration control is the most widely used manipulated variable in WWTPs since the DO level in the aerobic reactors plays a significant influence on the behavior and activity of the heterotrophic and autotrophic microorganisms living in the activated sludge. The DO concentration in the aeration tank must be sufficient to sustain at all times the desirable microorganisms in the aeration tank, clarifier, and return sludge line back to the aeration tank. Without good DO control, there is a danger that the plant may suffer one or both extreme DO conditions. For any activated sludge plant, a very low DO concentration inhibits treatment effectiveness and possibly promotes filament growth. Unnecessarily high DO concentrations can create excess turbulence and may result in the break up of the biological floc and waste energy (Carlson et al, 1994; Lindberg and Carlson, 1996; Cho, Sung, and Lee, 2002).

To overcome the time-consumed manual tuning procedures of PID controllers, many on-line identification methods have been proposed to obtain the process information. Neural networks and fuzzy logic have gained increasing attention to solve control problems characterized by ill-defined systems, especially the nonlinear time-varying biological wastewater treatment processes considered in this thesis. As far as empirical data models are concerned, neural network models are normally used because of the following advantages:

- they constitute universal approximators (Hornik *et al.*, 1989), hence are able to approximate any nonlinear behavior of technological dynamic processes (Hussain, 1999; Norgaard *et al.*, 2000; Cybenko, 1989),
- efficient identification algorithms and structure optimisation techniques have been developed (Haykin, 1999; Osowski, 1996),
- they have a relatively small number of parameters and simple structures ensure computational efficiency and accuracy,
- they can be easily incorporated into model predictive control algorithms and efficiently used on-line (Hussain, 1999; Lawrynczuk and Tatjewski, 2006; 2003; 2002; Norgaard *et al.*, 2000),
- they do not require a deep knowledge about the process or system to be treated (they act rather like black-box)

The neural network theory is applied in this thesis for modelling and control design of the DO concentration process. Thus different control structures based on the feedforward and the recurrent neural networks are developed in the accompanying literature based on the nonlinear mass balance model of the DO concentration (Olssen and Newell, 1999) whose state space equation is:

$$\frac{dS_O}{dt} = \frac{Q}{V}\left[(S_{O\,in}(t) - S_O(t)\right] + K_L a(u(t))\left[S_{O\,sat}(t) - S_O(t)\right] + r_{so}(t) \quad (1.2)$$

where, $S_{O\,in}(t)$ is the dissolved oxygen concentration in the input flow, $S_O(t)$ is the dissolved oxygen concentration in the aerobic tank, $S_{O\,sat}(t)$ is the saturation value of the dissolved oxygen concentration. $Q(t)$ is the flow rate of the waste water V is the volume of waste water. $K_L a(u)$ is the dissolved oxygen transfer function, $u(t)$ is the airflow rate in the aerobic tank from the air production system. $r_{so}(t)$ is the respiration rate. It is assumed that $S_O(t)$, $S_{in}(t)$ $u(t)$ and $Q(t)$ are measured and that $S_{sat}(t)$ and V are known constants. The oxygen transfer function $K_L a$ is nonlinear and this makes Equation1.2 to be nonlinear.

1.3 Descriptions of Neural networks

1.3.1 Historical perspective

Artificial neural networks are computational models of the human brain, i.e. they represent the brain's structure and operation with varying degree of sophistication. The history of neural networks can be traced back to McCulloch and Pitts who published a paper in 1943(McCulloch, and Pitts, 1943) that described a formal calculus of networks of simple computing elements and these basic ideas that were developed by them forms the basis of artificial neural networks. In their publication, they showed that networks of artificial neurons could, in principle, compute any arithmetic and logical function. Then in 1949, Donald Hebb (Hebb, 1949) developed what is known as the 'Hebbian learning rule'. He found that if two connected neurons were simultaneously active, then the connection between them is proportionately strengthened. In other words, the more frequently a particular neuron is activated, the greater the weight between them (i.e., learning by weight adjustment).

In 1958, Rosenblatt devised the *perceptron* model, which generated much interest because of its ability to solve some simple pattern classification problems. However the interest started to fade in 1969 when Minsky and Papert [1969] provided mathematical proofs of the limitations of the perceptron and pointed out its weaknesses in computation. In particular, they proved that it was not capable of solving the classic exclusive-or (XOR) problem. Such drawbacks led to the temporary decline of the field of neural networks.

The developments of more-powerful networks, better training algorithms, and improved hardware have all contributed to the revival of the field of neural-networks. Neural

network paradigms in recent years includes, Widrow and Hoff's learning rule(1960), the recurrent network (Hopfield's, 1982), Kohonen's network (1972), Rumelhart's competitive learning mode l (1986) with regards to back propagation techniques of training multilayer perceptron, Fukushima's model, Carpenter and Grossberg's Adaptive Resonance Theory model [Wasserman 1989; Freeman and Skapura, 1991]. Thus, the field of neural networks has generated interest from researchers in such diverse areas as engineering, computer science, psychology, neuroscience, physics, and mathematics etc.

1.3.2 Neural network structure

In the most general sense, a neural network is a dynamic system consisting of simple processing units, often called "neurons" or "nodes," and information passing links between these nodes often called "interconnects" or "synapses," which can perform information-processing by responding to a set of input nodes containing information requiring processing. Each neuron in a neural network usually combines the outputs from other neurons in the network or external inputs with values from previous computations (local memory) to produce inputs to other neurons in the network. How the inter-neuron connections are arranged and the nature of the connections determines the structure of a network. The processing elements each have a number of internal parameters called weights. Changing the weights of an element will alter the behavior of the element and, therefore will also alter the behavior of the whole network (Pham, and Liu, 1996).The goal here is to choose the weights of the network to achieve a desired input/output relationship. The process is known as training the network. How the weights (strengths) of the connections are adjusted or trained to achieve the desired overall behavior of the network is governed by its learning algorithm. Networks can be classified according to their structures and learning algorithms.

The most common neural network architecture that has been applied in the field of control is the multi-layer perceptron. The architecture of a multilayer perceptron is described in details in many textbooks and articles e.g. Chen, and Billings (1990), Narendra, and Parthasarathy (1990), Rumelhart, and McClelland, (1986), Hecht-Nielsen, (1990), Hagan, and Demuth, (1999) etc.
It is thus useful to investigate this topology, discuss its capabilities and show how it is used in control system configuration.

A multilayer perceptron is a feed forward artificial neural network model that maps sets of input data onto a set of appropriate output. It is a modification of the standard linear perceptron in that it uses more than one layer of neurons (nodes) with nonlinear

activation functions, and is more powerful than the perceptron in that it can distinguish data that is not linearly separable. The output of neurons in the output layer is computed similarly. Typically the transfer function is chosen by the designer, and the weights and biases are adjusted by some learning rule (algorithm) so that the relationship meets some specific goal. One of the most commonly used functions with multilayer networks that are trained using back propagation algorithm is the log-sigmoid transfer function. This is because this function is differentiable.

1.3.3 Approximation capabilities of multilayer networks

Two layer networks, with sigmoid transfer functions in the hidden layer and linear transfer function in the output layer, are universal approximators (Hornik, et al, 1989). They provide proof that multilayer perceptron network with sigmoid transfer functions in the hidden layer and a linear transfer functions in the output layer can approximate virtually any function of interest to any degree of accuracy, provided that sufficiently many hidden units are available.

1.3.4 Use of neural networks in the control of nonlinear systems

Neural networks have been applied very successfully in the field of control engineering and the literature in this area is quite voluminous. However, the theory of nonlinear systems neurocontrol is still a developing one compared to the vast amount of knowledge and tools that are available in the study and analysis of linear systems. This is because of the diversity of non-linear systems in practice.

A neural control system is one in which an artificial neural network is used in the process of determining the control signal that will bring the system to the required performance specifications. The classification of neural network control schemes is accomplished based on the methods by which the error signal for control is determined e.g., supervised, Adaptive, Reinforcement, Predictive, Optimal, Model reference etc. Research has shown that neural network controllers have important advantages over conventional controllers in the following aspects:

- Neural network controllers can effectively utilise a much larger amount of sensory information in planning and executing a control action than a conventional industrial controller can.
- A neural network controller has the collective processing capability that enables it to respond quickly to complex sensory inputs while the execution speed of sophisticated control algorithms in a conventional controller is severely limited.
- Good control can be achieved through learning The universal approximation capabilities of the multilayer perceptron make it a popular choice for modelling non-linear systems and for implementing general purpose non-linear controllers(Hagan, and Demuth, 1999).

There are typically two steps involved when using neural networks for control:

- System Identification

- Control Design

In the system identification stage, a neural network model of the plant that requires to be controlled is first developed. In the control design stage, the neural network plant model developed is then used to design (or train) the controller. As mentioned earlier, the multilayer neural networks are used for identification as well as for controller design. Several neural network architectures have been used for control. Each controller has its own strengths and weaknesses. However, no single controller is appropriate for every application.

1.4 Research Aim and Objectives

The aim of this research is development and investigation of the structures of the feedforward and recurrent neural networks in solution of the problems for identification and control of nonlinear systems.

The project objectives are:

- Development of mathematical models for the nonlinear system.
- Development of nonlinear identifier models for nonlinear systems by use of neural networks.
- Development of neural network controllers based on the inverse and internal models
- Development of neural network controllers based on the closed loop linearization approach
- Development of algorithms and programs for the identifiers and controllers in Matlab for deployment on a hardware platform.
- Real time implementation of the developed program on a real nonlinear system.

1.5 Statement of the problem

The main research problem is to investigate artificial neural networks (ANNs) for the purposes of identification, modelling and control of the nonlinear DO concentration of the activated sludge process in WWTP.

1.6 Design and Implementation based problems

- Nonlinear mathematical model development of the nonlinear system
- Investigation of the various techniques for implementation of neural networks in the context of control.
- Use neural networks for system identification by developing nonlinear identifier models for nonlinear discrete time systems
- Use function inversion to provide neural network control with reinforcement learning for systems with multiple nonlinearities.
- Develop algorithms and programs for nonlinear controllers in Matlab for deployment on a hardware platform.
- Develop software for embedding the neural network controller's algorithm in the software environment of a programmable logic controller (PLC)

1.7 The Hypothesis

The hypothesis is based on the possibility of applying neural network theory and control algorithms to develop controllers for a nonlinear system to ensure that the system meets the performance specifications.

1.8 Delimitation of the research

The research project will concentrate upon the design of nonlinear control systems and the use of neural network theory for purposes of non-linear NN identification and control of the DO concentration process of the activated sludge process in the WWTP. The results are to be implemented in real time. The research is limited to the feedforward multilayer perceptron and the recurrent neural networks for the identification and control. Control models considered are the direct inverse model control, internal model control and feedback linearizing control. Real-time implementation is limited to the lab-scale WWTP located in the department of electrical Engineering, Cape Peninsula University of Technology, Bellville Campus.

1.9 Motivation for the project development

Industrial applications present challenging problems to face when dynamic models are required for control of nonlinear systems. In contrast to a linear case, the response of nonlinear systems to a particular test signal is no guide for their behaviour to other inputs since the principle of superposition no longer holds. In model based control, input-output nonlinear models can either be developed from physics principles or obtained from a system of identification procedure. The first approach is most adequate but, in practice, it often has some problems e.g.

- It is difficult to set the correct values for physical parameters in order to get a relevant model for a specific application.
- The identification of physical parameters from data is not trivial; due to the structure of the nonlinear equations that govern their dynamic behaviour.
- The model based on theoretical fundamentals can be very complicated and its use for control purposes is not straight forward.

An alternative solution is to use identification algorithms in order to obtain a control model for the nonlinear system. There is an ever increasing volume of literature on neural networks for control.

1.10 Assumptions

The following assumptions are made based on different parts of the research work:

- Good strong understanding of nonlinear control theory including stability theory, discrete-time theory and the analytical techniques for nonlinear systems.
- The mathematical model of the system to be controlled can be derived according to the physical laws in order to determine the solution of the above problems.
- The nature of the nonlinear function and the region where controllability and stability of the system are of interest.

- Familiarity with different neural network topologies, learning techniques and control architectures as well as the rigorous mathematical analyses of neural network controllers is enough for development of the project.
- Neural networks are capable to meet the objectives because they are invaluable for applications where formal analysis is difficult or impossible, such as nonlinear system identification and control.
- Hardware platform for deployment of the controllers is available.

1.11 Research methods

The field of Neural networks for control and especially for waste water treatment process is quite new to the author and hence required integrated research approach has been adopted. Methods adopted include:

- **literature review method:** whereby information on nonlinear systems and on neural networks for control are gathered from related books, published journals, previously done work, internet search engines, questionnaires, etc in order to come up with the best approach to the research project. This entails good understanding of nonlinear control theory including stability theory, discrete-time theory and the analytical techniques for nonlinear systems. In addition, familiarity with different neural network topologies, learning techniques and control architectures is essential.

- Design method: Rigorous mathematical analyses of nonlinear systems, neural network controllers to obtain the mathematical models. Development of mathematical models and simulation in Matlab environment are done.

- Design method: Algorithms and programs in Matlab software are designed and developed for the prototype controllers.

- Experimental method: Simulation of the programs in Matlab environment is done on a hardware platform in real time in order to test the effectiveness of the neural network identification structure.

- Descriptive method: The designed system is described in terms of characteristics, functionality, and methodology for the practitioners in the field of control in line with the stated objectives.

1.12 Outline of the thesis

The thesis consists of seven chapters detailing the background information, model development, neural networks identification, NN controller design, system simulation algorithms implementation, results of the research project and conclusion.

Chapter 1 presents the background study of the project. The challenges of the control of nonlinear systems are highlighted and solution of the control by neural networks is suggested followed by a brief introduction to neural networks and the nonlinear plant to be controlled.

Chapter 2 of the research project gives an overview of the necessity of clean water and the need of proper wastewater treatment plants. It also contains a brief description of how a wastewater treatment plant works, the various wastewater treatment models, and the different control strategies applied to the process. It also gives an overview of the models for dissolved oxygen concentration.

Chapter 3 examines artificial neural networks. It discusses artificial neural networks theory including the fundamental principles, the various structures and configurations and the neural network application techniques for Identification and control of nonlinear processes.

Chapter 4 presents the theory of neural network modelling to waste water treatment plants and in particular, the emphasis is on the development of the models for dissolved oxygen (DO) process using different approaches and different neural network structures for identification. The chapter also gives an overview of the existing solutions using neural networks for the identification of the nonlinear process. Then nonlinear dynamics of the system is identified using the neural network based methods. Simulation results for the identified models are also presented.

Chapter 5 presents an overview of how to design neural network controllers. This is followed by the design and development of the neural network based controllers for the DO process. The design is based on the identified models of the DO process. Particular emphasis are put on Internal model control and feedback Linearising Control design strategies to maintain the desired dissolved oxygen (DO) level of an activated sludge system by manipulating the air flow rate in the aerobic reactor of an activated sludge process. Simulation results are also presented.

Chapter 6 describes the real process under consideration which is a lab-scale wastewater pilot plant located in the Electrical department at the Cape Peninsula University of Technology. The Real-Time implementation platform configuration structure and the PLC implementation for the designed neural network nonlinear linearizing controller are described.

Chapter 7 provides the detailed conclusions and the future direction for the research in the field of neural networks for control of the nonlinear DO concentration process. The list of papers describing the results of the thesis is shown at the end of the chapter. The developed software and models are given in the appendices.

CHAPTER TWO
WASTE WATER TREATMENT PROCESS

2.1 Introduction

Water is something very special to humanity and for every living creature on earth. This part of the research project gives an overview of the necessity of clean water and the need of proper wastewater treatment plants. It also contains a brief description of how a wastewater treatment plant works, the various wastewater treatment models, and the different control strategies applied to the process. It also gives an overview of the models for dissolved oxygen concentration.

2.2 Need of Waste Water treatment

There are a lot of good reasons why keeping water clean is an important priority to all. Having clean water is a necessity for life of almost all creatures on earth. For example, clean water is critical to plants and animals that live in water, human beings who live, work and play close to water, water birds that use water catchment areas for resting and feedings, species of fish which use water to habituate. In addition to being a necessity to life for all creatures on earth, water is used for numerous purposes among them, domestic, commercial, public supply, agricultural, industrial production, energy production, mining, etc. But for whatever the purpose is, processing and use normally results in the water being polluted. For health concerns and environmental conservation, water has to be treated before it enters to a recipient. Installing water treatment equipment not only improves the quality of the water but also raises the hygienic standard of the environment. Environmental protection is a priority in every developed country with every country apportioning a percentage of its Gross Domestic Product (GDP) for environment conservation.

Wastewater treatment is the process that removes the majority of the contaminants from wastewater and produces a liquid effluent suitable for disposal to the environment. During the contaminants (especially the solid materials) decays, oxygen which is needed by the plants and animals living in the water is used up. Because of the effects of dissolved oxygen concentrations on aquatic life, wastewater treatment engineers historically focused on the removal of pollutants that would deplete the dissolved oxygen concentration in the receiving waters. The history of wastewater treatment can be traced as far back to the beginning of the 20th century where the treatment process was mainly mechanical, that is, the removal of larger objects by use of a grit and sand filter because by then the environment was not as polluted as it is in the present day. The biological treatment process was introduced late in 1950's, whereby, microorganisms (e.g. bacteria) were used to remove organic matter present

in the incoming wastewater. Later in the seventies, the chemical treatment was employed to reduce the content of phosphorous. Nowadays, the biological processes are also used to reduce the content of nitrogen and phosphorous (Halvarsson, B., 2003).

Generally, while the primary goal of a wastewater treatment plant is to achieve an average reduction in nutrient levels, the secondary goal is to achieve good effluent quality in spite of the many disturbances and a variety of characteristics of the wastewater. In fact a key goal of the wastewater treatment process (WWTP) is to maintain quality standards of effluents at a minimum cost. Depending on the characteristics of the wastewater, the desired effluent quality, and other environmental or social factors, the treatment of wastewater can be achieved in different ways as is briefly outlined below.

2.3 The Waste Water Treatment Process

The treatment of wastewater generally uses a combination of primary, secondary and tertiary treatment processes. Figure 2.1 shows a principal layout of a typical Wastewater Treatment Plant (WWTP). Operations of the various stages of the treatment process are subsequently explained.

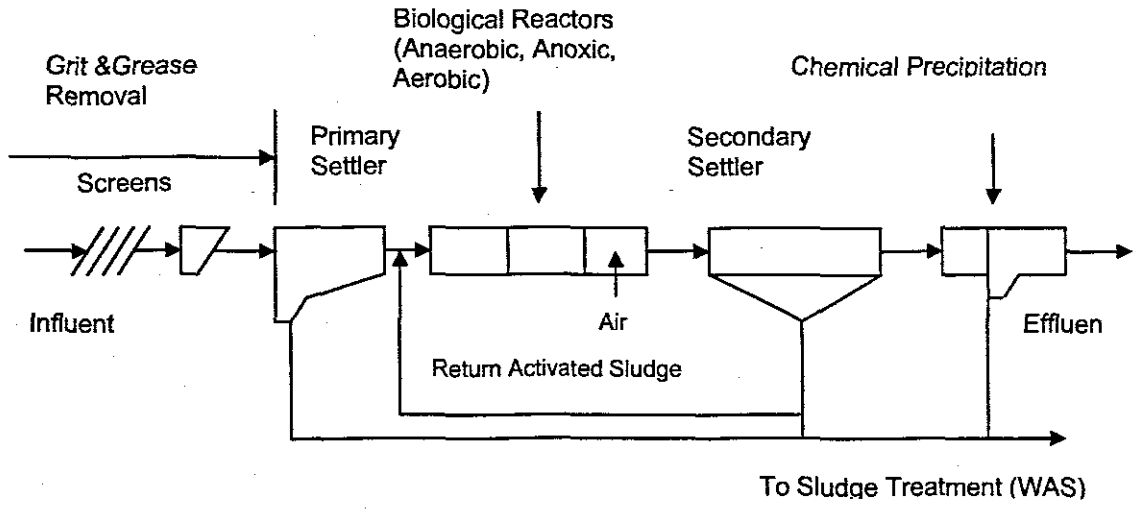


Figure 2.1 *Principal layout of a typical Wastewater Treatment Plant.*

Primary treatment simply screens and settles large particles by sedimentation and filtering. Secondary treatment biologically removes organic carbon and in newer plants soluble nitrogen and/ or phosphates. Tertiary treatment attempts to limit the microorganisms and other pathogens in treated water by membrane filtration and some form of disinfection using chlorine, etc. A brief description of the various treatment stages is subsequently provided.

2.3.1 Pre-treatment Process

Pre-treatment involves use of simple mechanical processes to remove rubbish at very low cost, so that the efficiency of the wastewater treatment plant is increased and cost is minimized. Pre-treatment process can be carried out by means of Screens, Grit chambers, Skimming tanks, and Grease Traps. The first step is to remove floating and suspended matter such as cloth, paper, kitchen refuse, pieces of wood, cork, hair, fiber, faecal solids etc. This is achieved through screening process. Then the Grit chamber is used to remove solids having specific gravity greater than water. The Grit chamber is an enlarged channel or long basin in which the cross-section is increased to reduce the velocity of the flowing sewage sufficiently to cause heavy inorganic matter such as grit, sand and gravel to settle, while the lighter organic matter remains in suspension.

Skimming tanks are also used to remove the floating solids from wastewater. A skimming tank is a chamber so arranged that the floating matter, oil, fat, grease etc., rise and remain on the surface of the sewage until removed, while the liquid flows out continuously under partitions or baffles. Grease traps are designed with submerged inlet and bottom outlet and have sufficient capacity to permit the sewage to cool and grease to separate.

2.3.2 Secondary treatment Process

Secondary treatment is designed to substantially degrade the biological content of the waste water. Secondary treatment typically utilizes biological treatment processes, in which microorganisms convert non settleable solids to settleable solids. Sedimentation typically follows, allowing the settleable solids to settle out. The secondary or biological treatment of water utilises a number of key processes. These include primary processes for biological reactions and processes for settling and clarification. Chemical precipitation to remove phosphorous is sometimes utilised before the settling and clarification. For this to be effective, the biota requires both oxygen and a substrate on which to live. There are a number of ways in which this is done. In all these methods, the bacteria and protozoa consume biodegradable soluble organic contaminants and bind much of the less soluble fractions into floc. These options include:

- Activated Sludge- The most common option uses microorganisms in the treatment process to break down organic material with aeration and agitation, then allows solids to settle out. Bacteria-containing "activated sludge" is continually re-circulated back to the aeration basin to increase the rate of organic decomposition
- Trickling Filters- These are beds of coarse media (often stones or plastic) 1-4m deep. Wastewater is sprayed into the air (aeration), and then allowed to trickle through the media. Microorganisms attached to and growing on the media, break

down organic material in the wastewater. Trickling filters drains at the bottom, and the wastewater is collected which then undergoes sedimentation.

- Lagoons- These are slow, cheap, and relatively inefficient, but can be used for various types of wastewater. They rely on the interaction of sunlight, algae, microorganisms, and oxygen (sometimes aerated).

2.3.3 Tertiary treatment Process

After primary and secondary treatment, wastewater is usually disinfected using chlorine (or other disinfecting compounds, or occasionally ozone or ultraviolet light).This process kills any harmful organisms remaining in the treated water before it is discharged. An increasing number of wastewater facilities also employ tertiary treatment; often using advanced treatment methods. Tertiary treatment may include processes to remove nutrients such as nitrogen and phosphorus, and carbon adsorption to remove chemicals. These processes can be physical, biological, or chemical. Settled solids (sludge) from primary treatment and secondary treatment settling tanks are given further treatment and undergo several options for disposal.

2.3.4 Activated Sludge System (Biological Reactors)

Most of the wastewater treatment plants use it. The activated sludge process aims to achieve, at minimum costs, a sufficiently low concentration of biodegradable matter in the effluent together with minimal sludge production. In order to create conditions favourable for the growth of bacteria in an activated sludge system, the treatment plant must have three distinct zones where conditions are carefully controlled to promote the required biological activity:

1)Anaerobic zone (a zone deficient in nitrate and oxygen),

2)Anoxic zone (a zone deficient in oxygen but nitrate is present) and

3)Aerobic zone (a zone with oxygen and nitrate present).

These zones are meant for removal of Carbon, Nitrogen and Phosphorous.

2.3.5 Anaerobic zone

Anaerobic treatment of wastewater involves the breakdown of complex organic material by bacteria in the absence of oxygen and air.The primary objective of most anaerobic processes is stabilization of biodegradable organic matter through its conversion to methane. The acid-producing bacteria metabolize the carbohydrates and proteins with the production of organic acids and ammonia. The methane bacteria take organic acids produced by acid-forming bacteria and the fatty acids in the water and convert them to methane. Thus, the key to any successful anaerobic system appears to lie in the proper balance between the acid-forming bacteria and the methane-forming bacteria (Richard R. Dague *et al, 1996*).

2.3.6 Anoxic zone

Anoxic refers to the presence of nitrates and the absence of dissolved oxygen (Buchan, 1984; Pitman, 1984; Streichan et al., 1990). Anoxic zone is defined as one where no free oxygen is available, and thus respiring organisms are forced to utilise the nitrate present as an oxygen source. Introducing an anoxic zone into the flow scheme provides for denitrification of nitrate. In this zone, the endogenous oxygen demand of mixed liquor suspended solids (MLSS) plus the carryover of BOD (biochemical oxygen demand) from the anaerobic zone causes denitrification of the nitrate produced in the aerobic zone. Because of this, the oxygen molecules are stripped from the nitrate, causing the production of nitrogen gas (N_2). Carbon dioxide and water are also produced in the process, which results from the degradation of BOD. In addition, a portion of the alkalinity consumed during the nitrification process is restored through the denitrification process. Readily biodegradable volatile fatty acids (VFAs) such as acetic acid, acetate, formic acid, etc., are used by the organisms in the mixed liquor. Methanol, ethanol, and acetate may be used as an additional carbon source when required in the process stream.

2.3.7 Aerobic zone

In this zone three important processes take place: organic carbon compounds are converted into water and carbon dioxide; ammonia is combined with oxygen to form nitrites and nitrates; and selected bacteria use food that they stored in the anaerobic zone to grow and take up phosphorus within their cell structure. The flow is then split. Some of the flow is recycled to the anoxic zone and some to the clarifiers where the bacteria settle in the bottom as sludge. The sludge from the clarifiers is recycled to the anaerobic tanks, while the liquid proceeds to the tertiary treatment stage. Sludge is also removed from the aerobic tanks to be processed in the sludge treatment part of the plant. Phosphorous and other unwanted compounds bound up in the bacteria cell structure are removed with the sludge.

2.3.8 Sludge treatment

Sludge from the aerobic zone is pumped to a Dissolved Air Floatation (DAF) thickener. This uses very fine air bubbles which attach themselves to the sludge particles and float with them to the surface where they are skimmed off. The thickened sludge is then pumped to the sludge treatment plant for dewatering before being thermally dried. The liquids from the DAF unit are returned to the beginning of the biological treatment process.

2.3.9 Basics of Waste Water Treatment nitrate removal steps

Biological conversion of ammonium to nitrogen gas is a two step process. Ammonium must first be oxidized to nitrate; nitrate is then reduced to nitrogen gas. These reactions require different environments and are often carried out in separate areas in the wastewater treatment system.

The first step in the process, the conversion of ammonia to nitrite and then to nitrate, is called nitrification. This process requires and consumes oxygen. The process is mediated by bacteria, which require an aerobic (presence of oxygen) environment for growth and metabolism of nitrogen. Thus, the nitrification process must proceed under aerobic conditions.

The second step of the process is the conversion of nitrate to nitrogen gas, and is called denitrification. This process is also mediated by bacteria. For the reduction of nitrate to nitrogen gas to occur, the dissolved oxygen level must be at or near zero; the denitrification process must proceed under anaerobic conditions.

Thus, any waste water treatment unit to be able to remove nitrogen during the nitrification/denitrification process must be designed to provide both aerobic and anaerobic zones.

2.4 Models of the Waste Water Treatment plants (WWTP)

Many models have been developed to provide a description of the complex biological reactions that take place within the activated sludge process (ASP). These models were developed as a tool to study the complex interactions that occur between the different microbiological communities in an activated sludge plant, among them are: the ASM No. 1 (ASM No. 2, ASM No. 2d (Henze et al., 1987, 1995, 1998) and ASM No.3 (Gujer et al., 1999)). The internal processes in the cell are explicitly taken into account in the Delft model (Murnleitner et al., 1996).The EAWAG BioP-Module (Rieger et al., 2001) enhances the ASM3 processes by describing the biological and chemical phosphorus removal

2.4.1 International Association on Water Quality Activated Sludge models

In order to encourage practicing engineers to use modelling more extensively during analysis of alternative wastewater treatment systems, in 1983 the International Association on Water Quality (IAWQ) [formerly the international Association on Water Pollution Research and Control (IAWPRC)] appointed a task group to review models for suspended growth cultures and produce one capable of depicting the performance of waste water treatment systems. They completed their task in 1986 and submitted a report to IAWQ which was published in 1987, (Henze et al, Activated sludge model No.

1. *IAWPRC Scientific and Technical Reports*, No. 1, 1987.), outlining the major features of activated sludge model (ASM) No. 1. The task group was influenced by the published work of many researchers, but that of Marais and colleagues at the University of Cape Town in South Africa had an impact on their thinking. Modelling is now used extensively in biological waste water treatment, in large part because of the success of ASM No.1.

2.4.2 The Activated Sludge model (ASM NO.1)

The Activated Sludge Model No. 1 (ASM1; Henze et al., 1987) can be considered as the reference model (benchmark model published by the International Water Association Quality - IWAQ), since this model triggered the general acceptance of WWTP modelling, first in the research community and later on also in industry. This model is regarded as one of the most popular mathematical descriptions of the biochemical processes in the reactor for nitrogen and chemical oxygen demand removal. The model is amenable to describe the process of carbon oxidation, nitrification and denitrification. The model is characterized by thirteen state variables and eight processes. Each reactor is defined by differential equations comprising the state variables and the eight processes describing the reactions that take place inside the reactor leading to carbon and nitrogen removal.

To cope with the complexities involved, Petersen's matrix format (Petersen et al., 2002) is adopted for model representation where the model state variables and the processes are characterized as matrix columns (given the index, *i*) and rows (given the index, *j*), respectively. Process rates (ρ_j) are formulated mathematically and listed down the right hand side of the matrix in line with the respective process. Stoichiometric coefficients (η_{ij}) for conversion from one state variable to the other are expressed using consistent units or COD equivalents along each process row. The overall reaction rate of each state variable r_i is obtained by moving down each column and multiplying the Stoichiometric coefficient, η_{ij} with the respective process rate ρ_j, then summing up:

$$r_i = \sum_{j=1}^{m} \eta_{ij} \rho_j, i = \overline{1,n} \qquad (2.1)$$

Where m is the number of processes, and n is the number of variables.

To obtain the complete set of equations describing the system, these rates need to be complemented by mass balance equations according to the principle of conservation of mass:

Rate of accumulation = Rate of input - Rate of output + Rate of production by reaction – Rate of reduction by death or consumption.

Assuming the reactor is fully mixed with constant volume, V, this results in a set of ordinary differential equations of the general form:

$$\frac{dC_i}{dt} = \frac{Q}{V}(C_{i,in} - C_i) + r_i \qquad (2.2)$$

Where C_i is the concentration of the state variable *i*, inside the reactor and $C_{i,in}$ is its concentration in the influent to the reactor or zone. Q represents the flow rate through the continuously stirred aerated reactor where inflow and outflow rates are equal. Following this basis, if the reactor is divided into separate anoxic and aerobic zones, then each zone should be treated as a separate unit with its own characteristics and volume, flows and process constants (Samuelsson, 1998).

As the reaction rates are mostly growth rates based on nonlinear Monod-type expressions, the set of ordinary differential equations characterizing the model is nonlinear as well and can not be solved analytically. Therefore, after appropriate initial conditions of the model state variables, numerical integration techniques based on Euler and the Runge-Kutta methods are usually adopted for model solution (Billing et al, 1998). The list of state variables, with their definition and appropriate notation, is given in Table 2.1, extracted from ASMI benchmark model (Henze et al, 1987).

Table 2.1 State Variables as defined by the ASM 1 model.

Definitions	Notations
Soluble inert organic matter	S_I
Readily biodegradable substrate	S_S
Particulate inert organic matter	X_I
Slowly biodegradable substrate	X_S
Active heterotrophic biomass	$X_{B,H}$
Active autotrophic biomass	$X_{B,A}$
Particulate products arising from biomass decay	X_P
Oxygen	S_O
Nitrate and nitrate nitrogen	S_{NO}
$NH_4^+ + NH_3$ nitrogen	S_{NH}
Soluble biodegradable organic nitrogen	S_{ND}
Particulate biodegradable organic nitrogen	X_{ND}
Alkalinity	S_{ALK}

The eight basic processes used to describe the biological behavior of the system are listed:

1) Aerobic growth of heterotrophs; the process converts readily biodegradable substrate, dissolved oxygen and ammonium into heterotrophic biomass.
2) Anoxic growth of heterotrophs; the process converts readily biodegradable substrate, nitrate and ammonium into heterotrophic biomass.
3) Decay of heterotrophs; heterotrophic biomass is decomposed into slowly degradable substrate and other particulates.
4) Aerobic growth of autotrophs; the process converts dissolved oxygen and ammonium into autotrophic biomass and nitrate.
5) Decay of autotrophic biomass; autotrophic biomass is decomposed into slowly degradable substrate and other particulates.
6) Hydrolysis of X_S to S_S; slowly biodegradable substrate is converted into readily biodegradable substrate.
7) Hydrolysis of X_{ND} to S_{ND}; particulate biodegradable organic nitrogen is transformed to biodegradable organic nitrogen.
8) Ammonification of S_{ND} to S_{NH}; Biodegradable organic nitrogen is transformed into ammonium.

The equations describing the overall process rates (production, dead, consumption) for every process variables are formed on the basis of Peterson's matrix, as follows:

The dynamic behaviour of the heterotrophic biomass concentration is affected by three different processes in the form of aerobic growth, anoxic growth and decay according to the differential Equation 2.3.

$$r_{XBH} = \left[\mu_H \left(\frac{S_S}{K_S + S_S} \right) \left\{ \left(\frac{S_O}{K_{O,H} + S_O} \right) + \eta_g \left(\frac{K_{O,H}}{K_{O,H} + S_O} \right) \left(\frac{S_{NO}}{K_{NO} + S_{NO}} \right) \right\} - b_H \right] X_{B,H} \qquad (2.3)$$

The autotrophic biomass concentration is simpler only because the autotrophs do not grow in an anoxic environment hence the situation is depicted by Equation 2.4,

$$r_{XBA} = \left[\mu A \left(\frac{S_{NH}}{K_{NH} + S_{NH}} \right) \left(\frac{S_O}{K_{O,A} + S_O} \right) - bA \right] X_{B,A} \qquad (2.4)$$

The readily biodegradable substrate concentration is reduced by the growth of heterotrophic bacteria and by hydrolysis of slowly biodegradable substrate and the differential equation describing the above conditions is Equation 2.5,

$$r_{SS} = \left[-\frac{\mu_H}{Y_H}\left(\frac{S_S}{K_S + S_S}\right)\left\{\left(\frac{S_O}{K_{O,H} + S_O}\right) + \eta_g\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right\} + \right.$$
$$\left. + k_h \frac{X_S / X_{B,H}}{K_X + \left(X_S / X_{B,H}\right)}\left\{\left(\frac{S_O}{K_{O,H} + S_O}\right) + \eta_h\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right\}\right] X_{B,H} \quad (2.5)$$

The concentration of slowly biodegradable substrate is increased as a means of recycling of dead bacteria as described by the death-regeneration hypothesis and as a result decreased by the hydrolysis process according to the Equation 2.6,

$$r_{XS} = (1 - f_P)(b_H X_{B,H} + b_A X_{B,A}) -$$
$$- k_h \frac{X_S / X_{B,H}}{K_X + \left(X_S / X_{B,H}\right)}\left\{\left(\frac{S_O}{K_{O,H} + S_O}\right) + \eta_h\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right\} X_{B,H} \quad (2.6)$$

The concentration of inert particulate products arising from biomass decay is given by Equation 2.7,

$$r_{XP} = f_P(b_H X_{B,H} + b_A X_{B,A}) \quad (2.7)$$

the concentration of particulate organic nitrogen is increased by biomass decay and decreased by the hydrolysis process and the resulting differential equation becomes Equation 8 as,

$$r_{XND} = (i_{XB} - f_P i_{XP})(b_H X_{B,H} + b_A X_{B,A}) -$$
$$- k_h \frac{X_{ND} / X_{B,H}}{K_X + \left(X_S / X_{B,H}\right)}\left\{\left(\frac{S_O}{K_{O,H} + S_O}\right) + \eta_h\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right\} X_{B,H} \quad (2.8)$$

The concentration of soluble organic nitrogen is affected by ammonification and hydrolysis, according to Equation 2.9.

$$r_{SND} = \left[-k_a S_{ND} + k_h \frac{X_{ND} / X_{B,H}}{K_X + \left(X_S / X_{B,H}\right)}\left\{\left(\frac{S_O}{K_{O,H} + S_O}\right) + \right.\right.$$
$$\left.\left. + \eta_h\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right\}\right] X_{B,H} \quad (2.9)$$

The ammonia concentration is affected by growth of all micro-organisms as ammonia is used as the nitrogen source for incorporation into the cell mass. The concentration is also decreased by the nitrification process and increased as a result of ammonification

of soluble organic nitrogen, resulting to a complex differential equation formulated as Equation 2.10,

$$r_{SNH} = \left[-i_{XB}\mu_H \left(\frac{S_S}{K_S + S_S} \right) \left\{ \left(\frac{S_O}{K_{O,H} + S_O} \right) + \eta_g \left(\frac{K_{O,H}}{K_{O,H} + S_O} \right) \left(\frac{S_{NO}}{K_{NO} + S_{NO}} \right) \right\} \right] X_{B,H} - \\ + k_a S_{ND} \\ -\mu_A \left(i_{XB} + \frac{1}{Y_A} \right) \left(\frac{S_{NH}}{K_{NH} + S_{NH}} \right) \left(\frac{S_O}{K_{O,A} + S_O} \right) X_{B,A} \quad (2.10)$$

The concentration of nitrate employs two processes, which are increased by nitrification and decreased by denitrification, hence the dynamic behaviour describing this, is formulated by Equation 2.11.

$$r_{SNO} = \mu_H \eta_g \left(\frac{1 - Y_H}{2.86 Y_H} \right) \left(\frac{S_S}{K_S + S_S} \right) \left(\frac{K_{O,H}}{K_{O,H} + S_O} \right) \left(\frac{S_{NO}}{K_{NO} + S_{NO}} \right) X_B + \\ + \frac{\mu_A}{Y_A} \left(\frac{S_{NH}}{K_{NH} + S_{NH}} \right) \left(\frac{S_O}{K_{O,A} + S_O} \right) X_{B,A} \quad (2.11)$$

The oxygen concentration in the wastewater is reduced by the aerobic growth of heterotrophic and autotrophic biomass, as described by the differential Equation 2.12.

$$r_{SO} = -\mu_H \left(\frac{1 - Y_H}{Y_H} \right) \left(\frac{S_S}{K_S + S_S} \right) \left(\frac{K_{O,H}}{K_{O,H} + S_O} \right) X_B - \\ -\mu_A \left(\frac{4.57 - Y_A}{Y_A} \right) \left(\frac{S_{NH}}{K_{NH} + S_{NH}} \right) \left(\frac{S_O}{K_{O,A} + S_O} \right) X_{B,A} \quad (2.12)$$

Tables 2.2 and 2.3 describe the meaning and the values of the parameters used in the above equations. Tables 2.2 and 2.3, list Stoichiometric and Kinetic simulation benchmark parameter values for the ASM1 for a temperature of 15°C (Cost 624) that are used in the research project.

Table2.2 Stoichiometric parameter values for ASM1 in the 'benchmark'

Parameter description	symbol	Value
Autotrophic yield	Y_A	0.24
Heterotrophic yield	Y_H	0.67
Fraction of biomass to particulate products	f_p	0.08
Fraction nitrogen in biomass	X_{ni}	0.08
Fraction nitrogen in particulate products	X_{ri}	0.06

Table 2.3 Kinetic parameter values for ASM1 in the 'simulation benchmark'

Parameter description	Parameter symbol	Value	Units
Maximum heterotrophic growth rate	μ_{mH}	4.0	day-1
Half-saturation (hetero. Growth)	K_S	10.0	$gCODm^{-3}$
Half-saturation (hetero. oxygen)	K_{OH}	0.2	$gO_2\ m^{-3}$
Half-saturation (nitrate)	K_{ON}	0.5	$gNO_3\text{-}N\ m^{-3}$
Heterotrophic decay rate	b_H	0.3	day-1
Anoxic growth rate correction factor	η_g	0.8	dimensionless
Anoxic hydrolysis rate correction factor	η_h	0.8	dimensionless
Maximum specific hydrolysis rate	K_h	3.0	$g\ X_s\ (g\ X_{BH}\ COD\ day)^{-1}$
Half-saturation (hydrolysis)	K_x	0.1	$g\ X_s (g\ X_{BH} COD)^{-1}$
Maximum autotrophic growth rate	μ_{mA}	0.5	day^{-1}
Half-saturation (auto. growth)	K_{NH}	1.0	$gNH_3\text{-}N\ m^{-3}$
Autotrophic decay rate	b_A	0.05	day^{-1}
Half-saturation (auto-oxygen)	K_{OA}	0.4	$gO_2\ m^{-3}$
Ammonification rate	k_a	0.05	m^3(g COD day)-1

Figure 2.2 is the Benchmark **ASM 1** model used and is the one created by the working group for the COST Benchmark plant. It provides comprehensive description of a standardized simulation and evaluation procedure including plant layout, simulation models and model parameters, a detailed description of disturbances to be applied during testing and evaluation criteria for testing the relative effectiveness of simulated strategies. It has two anoxic compartments and three aerobic compartments and a secondary settler. The benchmark plant thus combines nitrification with pre-denitrification in a configuration that is commonly used for achieving biological nitrogen removal in full-scale plants. The benchmark platform is a definition of a standard wastewater treatment plant with standard influent conditions.

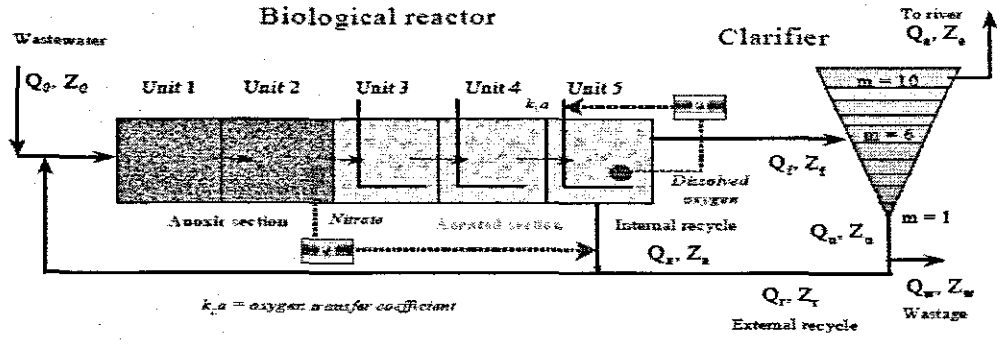


Fig 2.2: General overview of the Benchmark ASM1 plant (Henze et al., 1987).

The plant layout is fully defined and has the following characteristic features: Five biological tanks-in-series with a secondary settler. Total biological volume of the plant is 5999m^3. The first two tanks make up the anoxic zone with individual volume of 1000m^3, and the next three tanks create the aerobic zone with individual volume of 1333m^3. Dissolved oxygen saturated concentration in aerated tanks is 8 gO_2m^{-3}. Ideal settler without any reactions is assumed. The oxygen mass transfer coefficient rate is set to 240day^{-1}, while $K_L a$ at the last tank is controlled in order to maintain the dissolved oxygen concentration at 2mg/l. The flow rate of the internal recirculation is kept at 55338m^3/day. The secondary settler has a conical shape with the surface of 1500m^2 and depth of 4m. The flow rate of the sludge recirculation is 18446m^3/day and excess sludge is removed from the settler at 385m^3/day.

Since disturbances play an important role in the evaluation of controller performances, influent disturbances are defined for different weather conditions, that is, rain-weather, dry-weather and stormy weather conditions. In this research, dry-weather data are considered containing 2 weeks of influent data at 15 minutes sampling interval. The primary goal of control is to maintain the dissolved oxygen concentration at 2mg/l level in the last tank.

2.4.3 Mass-balances for the process variables concentration in a vector form

In order to obtain the complete set of differential equations for the different states, the reaction rates previously presented have to be complemented by the terms for the mass balance. Assuming a system such as that shown in Figure 2.2, with a bioreactor divided into 5 tanks, each of which is assumed to have a constant volume and to be ideally mixed, the process variables are organized in a vector form as $Z = [S_S \; X_{BH} \; X_S \; X_I \; S_{NH} \; S_I \; S_{ND} \; X_{ND}]^T$. The vector Z has different values for every tank. The general mass-balance model for the Benchmark process is as follows (reference):

For k = 1 (unit 1 in the input of the influent flow):

$$\frac{dZ_1(t)}{dt} = \frac{1}{V_1}(Q_a(t)Z_a(t) + Q_r(t)Z_r(t) + Q_o(t)Z_o(t) + r_1(t)V_1 - Q_1(t)Z_1(t))$$

$$Q_1(t) = Q_a(t) + Q_r(t) + Q_o(t) \quad (2.13)$$

Where: $Q_a(t)$ = internal recycle flow rate ($m^3 day^{-1}$)

$Q_r(t)$ = return sludge recycle flow rate ($m^3 day^{-1}$)

$Q_o(t)$ = waste sludge flow rate ($m^3 day^{-1}$),

And $Q_1(t)$ = Total flow rate ($m^3 day^{-1}$), $r_k(t)$ is the observed conversion rate for the k-th tank, $Q_k(t)$ is the flow rate and Z_k is the concentration in each tank.

For k = 2 to 5

$$\frac{dZ_k(t)}{dt} = \frac{1}{V_k}(Q_{k-1}(t)Z_{k-1}(t) + r_k(t)V_k - Q_k(t)Z_k(t)) \quad (2.14)$$

$$Q_k = Q_{K-1}$$

For special case of oxygen ($S_{O,k}$), $k = 3$ to 5

$$\frac{dS_{O,k}(t)}{dt} = \frac{Q_{k-1}(t)S_{O,k-1}(t) - Q_k(t)S_{O,k}(t)}{V_k} + r_{S_{O,k}}(t) + (K_L a(Q_{air}(t)))_k (S_{O,sat} - S_{O,k}(t)) \quad (2.15)$$

Where, $K_L a(\bullet)$ is the oxygen transfer function, Q_{air} is the air flow (that comes from the blowers), and $S_{O,sat}$ is the oxygen saturation concentration and the reaction rate is r_k.

2.4.4 University of Cape Town Process (Plant layout)

The University of Cape Town (UCT) Process is shown in Figure 3 In the UCT process, both the return activated sludge and aeration tank contents are recycled to the anoxic zone, and the contents of the anoxic zone are then recycled to the anaerobic zone. The recycle sequence decreases the chance of introducing residual nitrate to the anaerobic zone. The internal recycle can be controlled to maintain zero nitrates in effluent from the anoxic reactor, thereby ensuring that no nitrates will be returned to the anaerobic reactor.

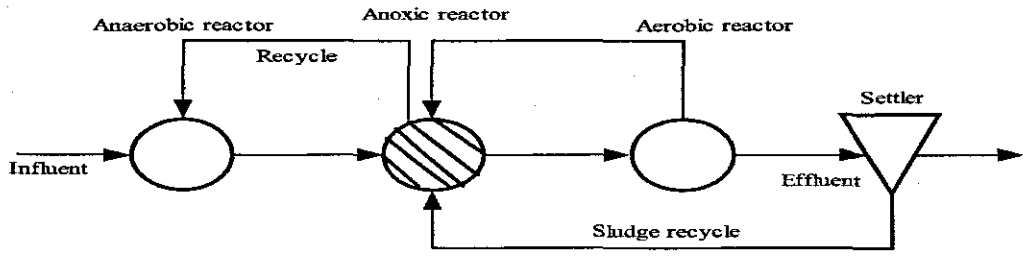


Figure 2.3: UCT model for biological nutrients removal.

The above plant layout is used with the UCT biological model describing the process rates. This model is characterized with more process variables and processes.

2.4.5 Measures of Water Quality

The standard reference of the water quality in the measurement of chemical, physical, and biological characteristics of water is the journal, *'Standard Methods for Examination of Water and Wastewater'* (Apha, 1998). It lists the following measures for measurement of the water quality:

- Dissolved oxygen, a significant determinant of water quality in systems and other water courses;
- Biochemical Oxygen demand, a parameter indicating the pollution potential a discharge would have on water courses;
- Solids, including suspended solids and dissolved solids, some of which could be detrimental to aquatic life or to people who drink the water;
- Nitrogen, a useful measure of water quality and a source of oxygen demand;
- Phosphorous, a critical nutrient, which when in excess is responsible (along with nitrogen) for accelerated eutrophication; and
- Bacteriological measurements, which are necessary to determine the potential for presence of infectious agents such as pathogenic bacteria and viruses and are usually indirect due to the problems of sampling for a literally infinite variety of microorganisms.

2.4.6 Dissolved Oxygen

The availability of oxygen in various treatment processes, and in the waters receiving the plant effluent, is perhaps the single most important measure of water quality. Wastewater in the primary clarifiers that are devoid of oxygen will have odours and might result in the floating sludge. Depressed oxygen levels in biological reactors can generally disrupt biological treatment. Similarly, low oxygen levels in the receiving waters can cause anaerobic conditions, with the commensurate odour and loss of desirable aquatic life.

2.4.7 Biochemical Oxygen Demand

The Biochemical oxygen demand (BOD) measures the rate at which oxygen is used by the microorganisms decomposing organic matter. This parameter reflects both rate at which organic matter is assimilated by microorganisms, and the quantity of organic matter available to the microorganisms. If the discharge to a water course has a high BOD, the microorganisms in the water will rapidly use up the available oxygen. If the reoxygenation from the atmosphere cannot keep pace with the use of oxygen, the dissolved oxygen levels drops, and creating unacceptable conditions in the stream. One of the primary objectives of wastewater treatment plants is the removal of this demand for oxygen, or BOD.

2.4.8 Solids

Solids can be a significant water pollutant and the separation of these solids from the water is one of the primary objectives of waste water treatment. If high concentrations of solids are discharged to the water course, they can cause unsightly floating scum, or they can sink and form potentially hazardous "mud" banks in streams. Most of these solids are also organic, so these particles will start to decompose and create a demand for oxygen. Finally, the floating solids are also a public health hazard because pathogenic organisms tend to "hide" on these solids and resist disinfection.

2.4.9 Nitrogen

Nitrogen is an important element in biological reactions. In the organic and ammonia form, nitrogen demands oxygen for oxidation to nitrate, NO_3^-, and is partially responsible for depressing oxygen levels in streams. One of the intermediate compounds formed during biological metabolism is ammonia-nitrogen. Together with organic nitrogen, ammonia is considered an indicator of recent pollution. These two forms of nitrogen are often combined in one measure, known as *Kjldahl nitrogen*, named after the scientist who first suggested the analytical procedure.

Aerobic decomposition eventually leads to nitrite (NO_2^-) and finally nitrates (NO_3^-)-nitrogen forms. A high nitrate and low ammonia-nitrogen therefore suggests that pollution has occurred, but quite some time ago. To remove the nitrogen from waste water in an ASP, two biological processes, nitrification, and denitrification are needed. In aerobic zone ammonia may be converted to nitrate (nitrification) and in anoxic zone nitrate may be converted into gaseous nitrogen (denitrification).

2.4.10 Phosphorous

Phosphorous is often the limiting nutrient in Lake Eutrophication. Because only a small quantity of phosphorous is available in a lake ecosystem, the metabolic activity is

limited. Were it not for the limited quantity of phosphorous, the ecosystem metabolic activity could accelerate and eventually fill up with biomass because all other chemicals are in plenty supply. In order for the system to remain at steady state, some key component, in this case phosphorous, must limit the rate of metabolic activity by acting as a break in the process.

2.4.11 Bacteriological Measurements

Bacteriological measurements are important in the protection of public health. Seeking the presence of pathogens presents several problems. First, there are many kinds of pathogens. Each pathogen has a specific detection procedure and must be screened individually. Second, the concentration of these organisms can be so small as to make their detection virtually impossible. Indicator organisms are used to define bacteriological water quality. The indicator most often used is a group of microbes called coliforms. The concentration of coliforms in water is measured by filtering a known quantity water through a sterile filter, thus capturing any coliforms, and then placing the filter in a petri dish containing agar that soaks into the filter and promotes the growth of coliforms while inhibiting other organisms.

2.5 Control of Waste Water Treatment plants

Generally, the aim of an automatic control system is to maintain, direct or regulate some dynamic variable or state of the process to a desired value in the presence of disturbances. Wastewater treatment plants, being complex processes with several variables (states) and also affected by disturbances require some form of control in order to maintain the standards of effluent. The complexity of the treatment process arises due to the fact that significant perturbations in flow and load together with variations in the composition of the incoming wastewater are normally difficult to predict and control. From a control engineer's point of view, the activated sludge plant is the most important of the biochemical processes for wastewater treatment because of the big number of the state variables and sub-processes describing the reactions that take place inside the reactors leading to carbon and nitrogen removal. This gives an insight on the type of control strategy to be designed for the process because control can then reduce the externally perceived complexity of the plant. A review on the historical evolution of the activated sludge process and its automatic control schemes can be found in, for example, (Jeppsson, 1996; Olsson *et al*, 1998).

Control techniques used in waste water treatment processes can be simple classical control strategies or more advanced controllers implemented in computers, using

advanced software algorithms. Traditionally, there was little interest in using the latter technique because of several constraints (Olsson, 1993) among them:

- Legislation-In many countries there were lack of tight regulatory standards which made it unnecessary to tighten the control on wastewater treatment plants.
- Education, training and understanding - In most countries public awareness was small due to delayed investments in better treatment facilities, also operators were not properly trained to deal with on-line instrumentation and control.
- Economy-Waste water treatment was considered a non-profit making venture and therefore there was no need to invest heavily on it.
- Measuring devices-There was lack of reliable-on-line sensors for ammonia, phosphorus, and for chemical oxygen demand monitors.

However, the development towards tighter effluent demands, increasing public awareness, plant complexity, better equipments, formulated effluent standards and improved tools have since contributed to an increasing interest in applying advanced control techniques for waste water treatment (Olsson, 1993; Olsson and Newell, 1999; Jeppsson, *et al.,* 2002). Several papers based on control of waste water have been proposed, for example, Sotomayor *el al,* (2001) proposed a simulation benchmark to evaluate the performance of advanced control techniques in biological wastewater treatment plants, Flanagan *et al.* (1977) proposed automatic dissolved oxygen control, Carlsson et al. (2002) proposed a benchmark study on control of an activated sludge process with nitrogen removal, Lindberg et al.,(1994) proposed an On-line estimation of the respiration rate and the oxygen transfer rate, Bastin and Dochain (1990), Lukasse (1999),Marian Barbu (2007) etc. proposed several model based automatic control strategies for wastewater treatment plants.

2.5.1 Control handles for nitrogen removal,

As explained in the preceding section, the objective of automatic control is to regulate some system so that the output signals of the system follows some desired trajectory by manipulating certain input signals by a control law. In the case of nitrogen removal in waste water treatment plants, typical outputs are ammonium and nitrate concentrations and the typical inputs that could be manipulated are: Air flow rates, Internal recycle flow rate, External sludge flow rate, the flow rate of an external carbon source to improve sludge sedimentation (Ingildsen, 2002; Halvarsson. B.2003). For example, the air flow rate is employed to affect the dissolved oxygen concentration in the aerated zones, thereby affecting the performance of the autotrophic nitrification bacteria. The most common procedure is to manipulate the air flow rate so as to maintain a specific dissolved oxygen level. The internal recirculation flow rate affects

the supply of nitrate for denitrification process and also the dissolved oxygen concentration in the anoxic tank. The external carbon dosage can be applied when the influent water does not have enough readily biodegradable substrate to feed the denitrification bacteria.

Never the less, the dissolved oxygen concentration control is the most widely used manipulated variable since the DO level in the aerobic reactors plays a significant influence on the behavior and activity of the heterotrophic and autotrophic microorganisms living in the activated sludge. The DO concentration in the aeration tank must be sufficient to sustain at all times the desirable microorganisms in the aeration tank, clarifier, and return sludge line back to the aeration tank. Without good DO control, there is a danger that the plant may suffer one or both extreme DO conditions. For any activated sludge plant, a very low DO concentration inhibits treatment effectiveness and possibly promotes filament growth. Unnecessarily high DO concentrations can create excess turbulence and may result in the break up of the biological floc and waste energy (Carlson et al, 1994; Lindberg and Carlson, 1996; Cho, Sung, and Lee, 2002). The DO concentration control is considered in this thesis on the basis of the application of the neural network theory.

2.5.2 Comparison of different Models for Dissolved oxygen concentration Control

Because of the importance of dissolved oxygen control many different control strategies have been proposed in the past. The DO level in the aerobic reactors has *significant influence on the performance of the microorganism in the activated sludge* process. Several papers based on simulation and control can be found for DO process. These simulation models can be identified as linear models, nonlinear models, or those based on artificial intelligence e.g. neural network models, Fuzzy networks and those modeled by genetic algorithms. Although the ASP is multiple input *multiple output system, the different time constants of the treatment processes allows* to decouple the many control actions into separate single input /single output controllers (Olsson and Jeppsson, 1994; Steffen et al., 1997) and use linear control techniques if it is possible.

Linear Control techniques commonly used are simple Conventional feedback control (on/off and PID controllers). PID controllers are familiar to process operators and very popular because of the simplicity, easiness in operation and robustness to modeling error. But it is well known that the DO concentration cannot be controlled effectively by using the PID controllers with fixed gain parameters. And the manual tuning of PID controller is tedious and laborious. Additionally, PID controllers show poor performance

for an integrating and with large time delay processes. Also, it can not incorporate ramp-type set point change or slow disturbance (Sung W. and Beum L., 1996). In case a linear model adequately describes the process behaviour, classical linear quadratic regulator theory can be used to design an optimal feedback controller (Marsili-Libelli, 1998; Vaccari *et al.*, 1998; Heinzle *et al.*, 1993; Spanjers *et al.*, 1998).

For nonlinear models, linearization around the desired operating points is normally used (Fan et al., 1973; Steyer et al., 1995; Weijers et al., Chang Yoo et al., 2003). Nonlinearity is a feature that describes the dynamics of the dissolved oxygen process. DO estimation and control may not be sufficiently achieved with a conventional linear controller. Several adaptive control strategies have been suggested for the DO control. The design approach starts from the nonlinear model to devise a nonlinear controller which ensures that the closed loop behaviour is linear (Ko *et al.*, 1982; Dochain and Bastin, 1990; Lindberg, 1997; Holmberg et al., 1989; Carlsson, 1993).

To overcome the time-consumed manual tuning procedures of PID controllers, many on-line identification methods have been proposed to obtain the process information. Neural networks and fuzzy logic have gained increasing attention to solve control problems characterized by ill-defined systems, especially the nonlinear time-varying biological wastewater treatment processes considered in this thesis. As far as empirical data models are concerned, neural network models are normally used because of the following advantages:

- they constitute universal approximators (Hornik *et al.*, 1989), hence are able to approximate any nonlinear behavior of technological dynamic processes (Hussain, 1999; Norgaard *et al.*, 2000; Cybenko, 1989),
- efficient identification algorithms and structure optimisation techniques have been developed (Haykin, 1999; Osowski, 1996),
- they have a relatively small number of parameters and simple structures ensure computational efficiency and accuracy,
- they can be easily incorporated into model predictive control algorithms and efficiently used on-line (Hussain, 1999; Lawrynczuk and Tatjewski, 2006; 2003; 2002; Norgaard *et al.*, 2000),
- they do not require a deep knowledge about the process or system to be treated (they act rather like black-box)

The neural network theory is applied in this thesis for modelling and control design of the DO concentration process.

A sample of papers about the control design and model development for WWTP and DO concentrations are tabulated in the Table 2.4 below.

Table 2.4: Sample of papers based on control for WWTP and dissolved oxygen concentration.

PAPER	PROBLEM	STATEMENT	OBJECTIVE(S)	MODEL USED	RESULTS
Chang Kyoo Yoo et al, 2003 *"Nonlinear model based DO control in a biological WWTP".*	A nonlinear control scheme to maintain the DO level of an activated sludge system is formulated	To take $K_L a$ and $r_{so}(t)$ of the DO nonlinear process model into account in the controller design.	Estimate the $K_L a$ rate and the time varying $r_{so}(t)$ by using the Kalman filter in the controller design.	Nonlinear Generic-model based control strategy imbedded directly into a PI controller.	Proposed controller shows enhanced control performance than PID.
Oscar Sotomayor, et al, 2001. *"Software sensor for on-line estimation of microbiological activity in ASPs".*	Design of a software sensor for on-line estimation of biological activities in ASPs where degradation and nitrification are considered.	.An estimation technique issued from the control and systems theory is applied in the development of soft sensors for on-line estimation of bio process variables.	$r_{so}(t)$, and $K_L a$ are estimated through a software sensor, which is based on modified version of the discrete extended Kalman filter.	Filtered random walk model for respiration rate estimation and an exponential model for the oxygen transfer function estimation.	The obtained results have shown that the used methodology was successful in estimating both $K_L a$ and $r_{so}(t)$.
B. Holenda, et al, 2007. *"DO control of ASP using model predictive control".*	Activated sludge WWTPs are difficult to be controlled because of their complex and nonlinear behaviour. MPC is investigated on two examples using systematic evaluation criteria.	Simulation of the alternating sludge process in order to investigate the efficiency of MPC in following rapidly changing DO set point.	To design and test the MPC controllers on the benchmark by controlling the DO level in the final compartment by manipulating of the $K_L a$ value.	Model predictive control strategy based on a linear state-space model on two simulated case-studies.	The results show that model predictive control can be effectively used for DO control in wastewater treatment plants.
Marian Barbu, et al, 2005. *"Control strategies of a multivariable wastewater treatment process. Comparative study."*	Design of multivariable controller for a MIMO wastewater process.	The simplified ASM1 model of the waste water treatment process is used to for the controller designs.	Develop two control strategies considering the channels of interaction as unobservable and then as observable disturbances.	PI type controller based on decoupling procedure and feed-forward control structure.	The numerical simulations illustrate a good behaviour of the control structure proposed for the control of the nonlinear WWTP.
Azwar M. A. et al, 2005. *"The study of neural network-based controller for controlling DO concentration in a sequencing batch reactor."*	This paper proposes an application of ANN to control the DO concentration in a sequential batch reactor (SBR).	NN- based controller for the DO concentration in a SBR is designed. The performance is evaluated through set point and disturbance rejection studies	Design and development of a NN-based controller performance for the activated sludge process in a SBR by manipulating the airflow rate.	Hybrid NN consisting of a basic NN controller in parallel with a PI controller; Direct inverse neural network, and internal model control scheme.	Although in the HNN controller some overshoot occurs, however, compared to the DINN and IMC schemes, the performance of the HNN controller in dealing with disturbances is very

					satisfactory.
B. Ra´duly, et al, 2006. *"ANNs for rapid wastewater performance evaluation: Methodology and case study."*	The influent model was used to represent sewer system effects on dynamics of WWTP influent data, in order to *obtain a realistic* framework for comparison of the result of mechanistic model simulations with the ANN predictions.	The approach combines an influent disturbance generator with a mechanistic WWTP models for generating a *limited sequence of* training data. ANN is then trained on the available data and subsequently used to simulate the remainder of data generated.	Provide different solutions for the reduction of simulation time by demonstrating of the capability of ANNs to *substantially reduce* simulation time compared to mechanistic models.	ASM3-based WWTP model and ANN simulations	The accuracy of the ANN is sufficient for applications in simulation-based WWTP design and simulation of *integrated urban* wastewater systems, especially when taking into account the uncertainties related to mechanistic WWTP modelling.
Kunwar P. Singh, et al, 2009. *"ANNs modelling of the river water-A case study."*	This paper describes the training, validation and application of ANN models for computing the DO and BOD levels in the Gomti river (India).	Two ANN models were used for computation of DO and BOD concentrations in the Gomti river water. The performance of the ANN models was assessed through the coefficient determination, square correlation coefficient, root mean square error and bias.	To construct an artificial neural network model for Gomti river water quality (DO and BOD) and demonstrate its application to complex water quality data and how it can improve the interpretation of the results.	Different ANN models were constructed and tested in order to determine the optimum number of nodes in the hidden layer and transfer functions.	This study shows that the optimal ANNs are capable to capture long-term trends observed for tedious river water quality variables (DO and BOD) both in time and space. The ANN can be seen as a powerful predictive alternative to traditional modelling techniques
O. A. Z. Sotomayor, et al, 2002, *"model -based predictive control of a pre-denitrification plant: a Linear state-space model approach."*	Paper focuses on the design of a model-based predictive control (MPC or MBPC) technique to regulate the concentration levels of nitrate in both anoxic and aerobic zones of a pre-denitrifying activated sludge plant, aiming to improve the nitrogen (N)-removal from wastewater.	The synthesis of the MPC controller is based on a linear Extended state-space model of the process, where an identification horizon is added to include a sequence of past inputs/outputs. Thus *eliminating the need for a* state observer	a MPC based on an extended linear state-space model of the process is developed, aiming to control the – NO3 concentrations in the anoxic and aerobic zones of an ASP and, therefore, to inferentially control the *effluent inorganic nitrogen* concentration.	MPC controller	The MPC controller performance is tested by simulation employing the ASWWTP-USP benchmark (Sotomayor *et al.*, 2001a).and the results show the efficiency of the proposed strategy
M. Fikar_, B. et al, *"Optimal Operation of Alternating Activated*	The study presents dynamic optimisation of a small size single basin	Based on the optimal state trajectories results, two simple feedback rules	An optimal sequence of aeration/non-aeration times so that for a	The model used is based on the ASM 1(Henze et al., 1987).	Simulation results with these rules show very satisfactory control

Sludge Processes".	wastewater treatment plant (WWTP).	are Proposed and are formulated.	diurnal pattern of disturbances, effluent constraints are respected, the plant remains in periodical steady state, and energy consumption is minimized.	Based on the problem specifications, the optimisation task has been defined and solved using the dynamic optimisation solver DYNO.	performance.
Mohamed T. S. and Laila M.F., 2005 *"Application of Activated Sludge Models in Traditionally Operated Treatment Plants--- A Software Environment Overview"*	The purpose of the paper is to evaluate the benefits that can be brought to an existing industrial AAS treatment plant upon application of optimal control theory	Optimal control of the aeration system is considered to improve efficiency and reliability of the activated sludge process, with application to an industrial alternating activated sludge (AAS) treatment plant.	Two problems are formulated: one deals with the minimization of nitrogen discharge and two addresses the minimization of the energy consumption. Special emphasis is placed on the long-term behavior of the plant in order to ensure optimality of aeration strategies.	Activated Sludge Model No. 1 (ASM1) is used to describe the transient behavior of the plant, with a set of parameters being estimated from input/output experimental data in order to optimize a realistic system.	This application to an industrial AAS treatment plant confirms the great potential of dynamic optimization methods for improving the performances of WWTPs.
M. A. STEFFENS and P. A. LANT., 1998. *"Multivariable control of nutrient-removing Activated sludge systems"*	The aim of this paper is to evaluate several multivariable model-based control algorithms for controlling nitrogen removal in activated sludge processes.	In this paper two questions are addressed. 1. What is the best control system structure for biological nutrient removal (BNR) wastewater treatment processes 2. What are the most appropriate algorithms?	Evaluates linear quadratic control (LQC), dynamic matrix control (DMC) and nonlinear predictive control (NPC) in this study for nitrogen removal in activated sludge processes.	Dynamic matrix controller (DMC) Nonlinear predictive controller (NPC) Linear quadratic controller (LQC), PI controller and Base case controller.	Five control algorithms were quantitatively evaluated using three disturbance scenarios. NPC performed the best of any control algorithm in terms of the process performance indicator and has associated costs which are comparable to the base case scenario. DMC and LQC control also allowed the process to meet objectives in the face of the disturbances investigated.
Baruch et al. 2003. *"Adaptive Recurrent Neural Network Control of Biological Wastewater*	Three adaptive neural network control structures to regulate a biological wastewater treatment process are introduced: indirect,	In this article a report on the application of three recurrent trainable neural network models for real-time identification and state feedback and	Keep concentration of recycled biomass proportional to the influent flow rate in the presence of periodically acting disturbances, parameter	An indirect, an inverse model, and a direct adaptive neural control models are developed.	A comparative study among the three neural control structures led to the conclusion that the indirect and the inverse model adaptive RTNN

Treatment."	inverse model, and direct adaptive neural control.	feedforward control of biological wastewater treatment is studied.	variations, and measurement noise. This is achieved by Recurrent Trainable Neural Network, structure, permitting the use of the obtained parameters, *during the learning phase*, directly for control system design.		schemes give qualitatively similar results. Nevertheless, the latter structure is more favorable with respect to rejecting *sensor noise and output* oscillations due to unknown perturbations.
P. Samuelsson and B. Carlsson. 2001. *"Feedforward control of the external carbon flow rate in an activated sludge process"*	A novel model based feedforward control strategy is developed using a steady state analysis of a simplified ASM1. This strategy is combined with a PI controller to compensate for disturbances and model simplifications.		To develop an automatic control strategy for adjusting the external carbon flow rate so that the nitrate concentration in the last anoxic compartment of an activated sludge process is kept at a low pre-specified level.	ASM1 model, PI and feedforward-feedback controller.	The controller was derived from a simplified version of the ASM1 model. In a simulation study, the controller performed very well and the nitrate level was kept close to its set-point despite major process disturbances

2.5.3 Oxygen transfer function

When air is blown into the waste water of an activated sludge process, oxygen is transferred to the water. The function which relates the oxygen transfer to the wastewater by aeration system is called the $K_L a$ function. The most common way to describe the rate of change of DO concentration, due to the air blown into the water, is by the following expression

$$\frac{dS_O(t)}{dt} = K_L a(u(t))(S_{O_{sat}} - S_O(t)) \qquad S_{O_{sat}} \rangle S_O(t) \qquad (2.16)$$

Where, $S_O(t)$ is the DO concentration, $S_{O_{sat}}$ is the saturated DO concentration and $u(t)$ is the airflow rate. Equation 2.16 shows that $K_L a$ depends only on the airflow rate $u(t)$. However, in real plants the oxygen transfer function depends on several factors, among them; type of the diffusers used, wastewater composition, temperature, design of the aeration tank, tank depth, etc, but the main time varying dependence is the airflow rate.

The oxygen transfer function $K_L a$ can be considered to consist of two parts, K_L and a. The K_L part can be seen as an absorption coefficient and the a part as an area/volume ratio. Both K_L and a are, however, usually unknown so they are lumped into one parameter denoted $K_L a$. Similarly, when analysing the $K_L a$ function, two parameters, α and β should be taken into consideration since they relate the $K_L a$ values for clean water and wastewater. The $K_L a$ value for wastewater is found by multiplying the: $K_L a$ for clean water ($K_L a_{CW}$) by the constant α, i.e. the constant α is defined as:

$$\alpha = \frac{K_L a}{K_L a_{CW}} \qquad (2.17)$$

In a similar way β compensates for the difference in saturated dissolved oxygen concentration between waste water $S_{O_{sat}}$ and the clean water $S_{O_{satCW}}$.

$$\beta = \frac{S_{O_{sat}}}{S_{O_{sat\,CW}}} \qquad (2.18)$$

2.5.4 Dynamics of Dissolved oxygen process

In this section the dissolved oxygen dynamics are outlined. Both continuous-time model and its corresponding discrete-time model are treated.

2.5.4.1 A Continuous time model

Control of dissolved oxygen concentration requires airflow rate control in the aerobic tanks. The mass balance equation relates a biomass activity to air supply and input organic load and is of the form:

$$\frac{dS_O}{dt} = \frac{Q}{V}\left[(S_{O_{in}}(t) - S_O(t)\right] + K_L a(u(t))\left[S_{O_{sat}}(t) - S_O(t)\right] + r_{so}(t) \qquad (2.19)$$

Where,

$S_{O_{in}}(t)$ is the dissolved oxygen concentration in the input flow

$S_O(t)$ is the dissolved oxygen concentration in the aerobic tank

$S_{O_{sat}}(t)$ is the saturation value of the dissolved oxygen concentration

$Q(t)$ is the flow rate of the waste water

V is the volume of waste water

$K_L a(u)$ is the dissolved oxygen transfer function

$u(t)$ is the airflow rate in the aerobic tank from the air production system.

$r_{so}(t)$ is the respiration rate.

In general, it is assumed that $S_O(t), S_{in}(t)$ $u(t)$ and $Q(t)$ are measured and that $S_{sat}(t)$ and V are known constants. Equation (2.19) describes the rate of change in the dissolved oxygen concentration. The first term on the right hand side in (2.19) is a pure mass balance, influent-effluent oxygen times the dilution rate Q/V. The next term, $K_L a(u(t))(s_{sat} - s(t))$ caused by aeration is the addition of oxygen, and the last term $r_{so}(t)$ describes the oxygen consumption by the microorganisms.

2.5.4.2 A Discrete-time model

There exists several ways to obtain a discrete-time model for a continuous-time model like (2.19). *One way is to use zero-order-hold sampling. Zero-order-hold sampling* assumes that all signals except $S_O(t)$ are constant during the sampling interval. When this is true then the discrete-time-model describes the continuous model exactly at the sampling instants and that the approximation is reasonable for sufficiently fast sampling rates (Holmberg, 1990; Cook and Jowitt, 1995). The derivation is as follows: (Lindberg. 1997).

Rewrite equation (2.19) to:

$$\frac{dS_O(t)}{dt} = A(t)S_O(t) + B(t) \qquad (2.20)$$

Where

$$A(t) = -(K_L a(u(t)) + \frac{Q(t)}{V})$$

$$B(t) = +r_{so} + \frac{Q(t)}{V} S_{O\,in}(t) + K_L a(u(t)) S_{O\,sat}$$

Assuming that $A(t)$ and $B(t)$ are constant within each sampling period, model (2.20) is sampled by using theory in (Astrom and Wittenmark, 1990), it gives:

$$S_O(kh+h) = e^{A(kh)h} S_O(kh) + \frac{B(kh)}{A(kh)}(e^{A(kh)h} - 1) \tag{2.21}$$

Where, h is the sampling interval and k is the discrete time. If it is set

$$h^* = \frac{1}{A(kh)}(e^{A(kh)h} - 1) \tag{2.22}$$

Then the sampled model becomes:

$$\frac{S_O(kh+h) - S_O(kh)}{h^*} = A(kh) S_O(kh) + B(kh) \tag{2.23}$$

For simplicity, replacing kh with t and substituting $A(t)$ and $B(t)$ in equation (2.23) the discrete equation is obtained as:

$$S_O(t+1) \approx S_O(t) + h^*[\frac{Q(t)}{V}[(S_{O\,in}(t) - S_O(t)] + K_L a(u(t))[S_{O\,sat}(t) - S_O(t)] + r_{so}(t)] + w(t) \tag{2.24}$$

Where, a term w has been added to describe measurement noise and model misfit. Equation 2.24 represents the discrete mathematical model of the process of dissolving, up taking and balancing the concentration of oxygen into the wastewater. The block diagram of the model is illustrated in Figure 2.4.

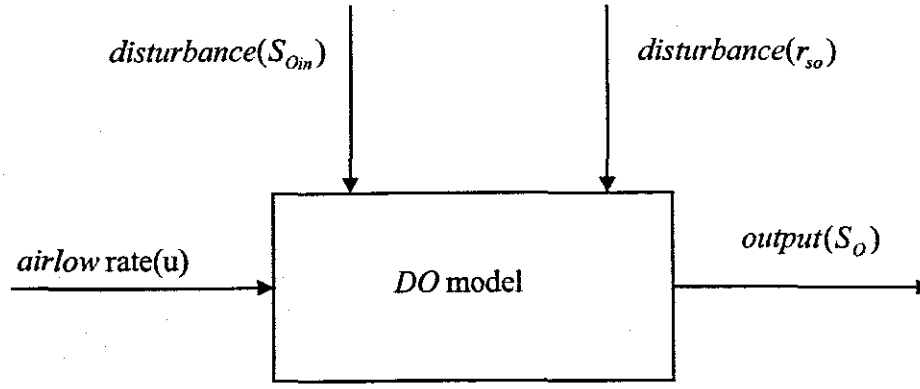


Figure 2.4: Block diagram of DO model.

The main disturbance for the process is the oxygen uptake rate r_{so}. It depends on the load COD and TKN in the input flow, on the toxic elements in the water flow rate. Other disturbance for the process is the input concentration for the given tank obtained from

model simulation. These disturbances will influence the process behaviour and also will require recalculation of the process controller parameters.

The respiration rate r_{so} and oxygen transfer rate $K_L a$ are two of the most important variables for monitoring biological activity and assessing process control performance. Knowledge of the two variables is therefore of interest in both process diagnosis and process control. What makes it not easy to identify the oxygen dynamics is that:

- The respiration rate $r_{so}(t)$ is time-varying.
- The oxygen transfer function $K_L a$ is nonlinear.
- The absolute levels of the respiration rate and $K_L a$ are hard to be uniquely determined.

$K_L a$ is a nonlinear function. That $K_L a$ function is not linear has been evaluated and discussed by different researchers amongst them: (Bennett, 1980) who presents various physical models to estimate $K_L a$. These models are: piecewise linear models, polynomial, exponential and spline models (Bennett, 1980; Holmberg, 1990; Holmberg *et al*., 1989) etc.

2.5.4.3 An exponential model of $K_L a$

If adequate dissolved oxygen control is assumed at steady state then the derivative term in equation (2.19) may be assumed zero. Thus the steady state oxygen mass balance is used here to relate $K_L a$ to the dissolved oxygen set-point $S_O(t)$, as follows:

$$K_L a = \frac{r_{so} + \frac{Q}{V}(S_O(t) - S_{O_{in}}(t))}{S_O(t) - S_{O_{sat}}(t)} \tag{2.25}$$

To maintain the dissolved oxygen at some value the $K_L a$ must be varied, which in turn implies manipulating the air flow rate. Hence the requirement for a function relating flow rate to $K_L a$.An exponential $K_L a$ model is a suitable choice (Lindberg and Carlsson, 1996) in natural sense since the oxygen transfer deteriorates for high airflow rate. The fitted exponential model of $K_L a$ is Equation (2.25)

$$K_L a(u(t)) = K_1(1 - e^{-K_2 u(t)}) \tag{2.26}$$

Where,

K_1 *and* K_2, are model parameters associated with aerobic tanks, and which require experimental estimation. $u(t)$ is the air flow rate. Rearranging for $u(t)$ gives:

From here the coefficients K_1 and K_2 can calculated if the values of the control $u(t)$ and the transfer function $K_L a$ are known.

$$u(t) = -\frac{1}{K_2} \bullet \ln\left[1 - \frac{K_L a}{K_1}\right]$$

$$K_2 = -\frac{1}{u(t)} \ln\left[1 - \frac{K_L a}{K_1}\right] \qquad (2.27)$$

$$K_1 = \frac{K_L a}{1 - e^{-K_2 u(t)}}$$

Where K_1 and K_2 are the estimated parameters in the, $K_L a$ function.

Thus, by controlling the air flow rate, dissolved oxygen concentration can be controlled. Hence the function $K_L a$ must be incorporated in the design strategy of the envisaged controller. Coefficients K_1 and K_2 can be found using curve fitting method. To calculate the air flow-rate, therefore, involves solving Equation (2.26) for K_{La} and then calculating the air flow-rate from Equation (2.27).

2.5.4.4 Estimation of the coefficients K_1 and K_2

The coefficients K_1 and K_2 can not be calculated together on the basis of the control and oxygen transfer function because they depend on each other. To overcome this difficulty a function from Matlab '*polyfit*' is used to fit this value of the transfer function given by exponential equation (2.26) to the value of K_{La}=240d^{-1} as given by the benchmark data. The estimated values of parameters K_1 and K_2 for the oxygen transfer function $K_L a$ of Equation (2.26) are found to be 240 and 1.0961e-004 respectively. The simulated response is illustrated in Figure 2.5.

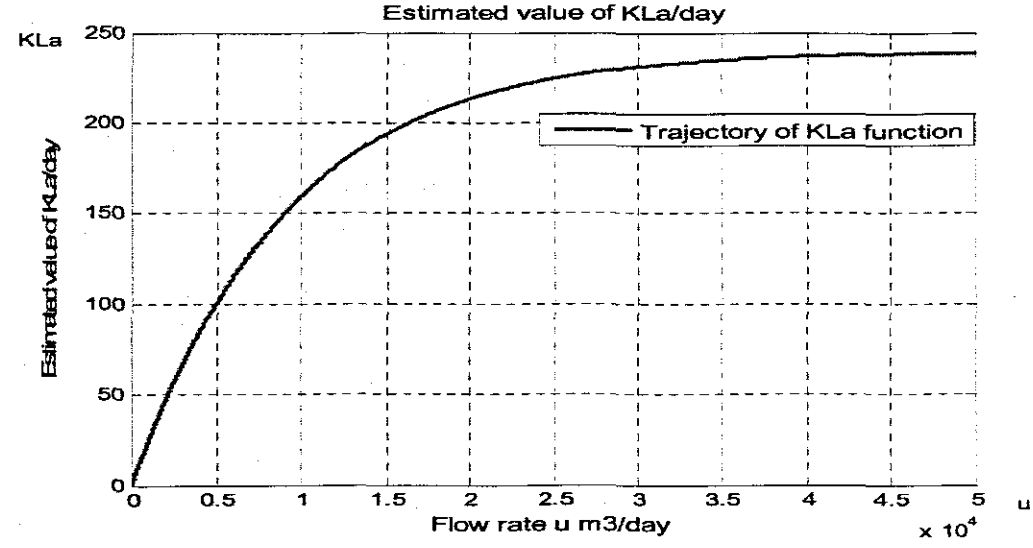


Figure 2.5: A typical curve of the $K_{La}(u)$ function

The values of the parameters for building of the model of the DO concentration process in Simulink are taken from the benchmark structure at steady state as illustrated in Tables 2.5 and 2.6.

Table2.5: Process parameters from the Benchmark structure

Parameters	$S_{o,sat}$	K_1	K_2	S_o	K_{OA}	V	$S_{o,in}$	t	Q_a	X_{BHS}	S_{NHS}
Values	8	240	1.0961e-004	0.491	0.4	1333	0.01	0:0.1	55338	2559.3	1.733

Table 2.6: Process parameters from the benchmark structure continuation

Parameters	Q_r	Q_o	μ_{uH}	μ_{uA}	y_H	y_A	K_{OH}	K_S	K_{NH}	X_{BAS}	$S_{S,SS}$
Values	18446	18446	4	0.5	0.67	0.24	0.20	10	1.0	149.78	0.889

2.5.4.5 Simulation of the DO concentration model

The differential equation of the plant representing the dissolved oxygen concentration dynamics as depicted in equation (2.24) is used to develop an open loop block diagram model in the software environment of Matlab/Simulink where the input to the model is the air flow rate (u) and the output is the DO concentration (S_O). The values of the parameters for the building of the model of DO concentration in the Simulink are taken from the Benchmark ASM1 plant (Henze et al., 1987) structure at steady state as illustrated in Tables 2.5 and 2.6. These parameter values are taken to the Matlab workspace. From there they are transferred to the Matlab/Simulink using blocks "From workspace". The resulting model is as illustrated in Figure 2.6(a) and 2.6(b).

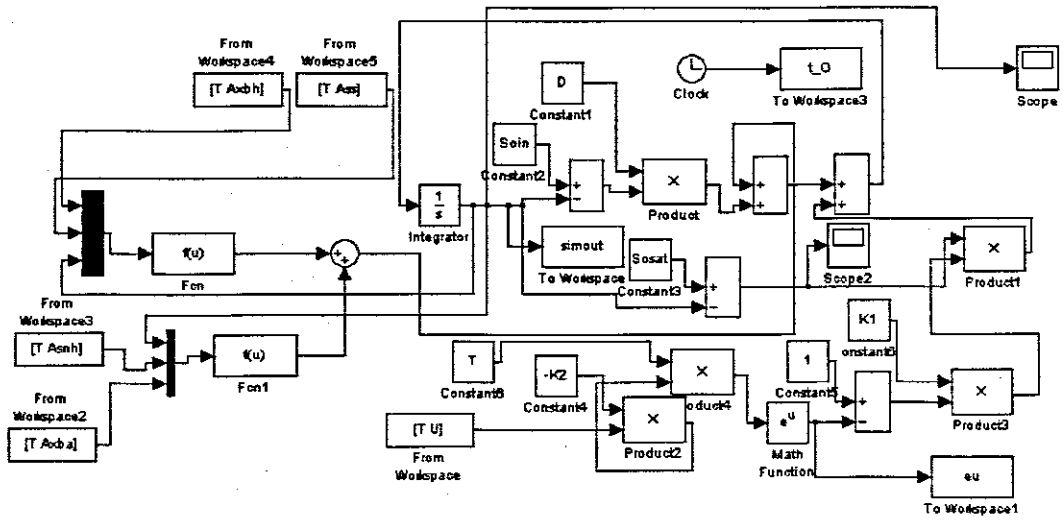


Figure 2.6(a): Open loop DO concentration model with full model of r_{so} in Simulink.

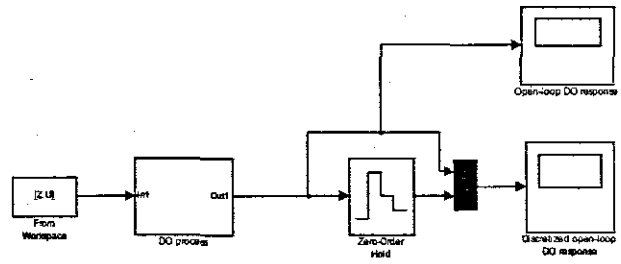


Figure 2.6(b): Open loop DO concentration sub-system model in Simulink.

Figure 2.8 shows the resulting time response obtained for the DO concentration model depicting the dynamic behaviour given by Equation 2.24 when simulated with an airflow rate of 50000m^3/day over a time interval equivalent to 0.1 of a day with fixed step of integration as shown in Figure 2.7 . A total of 960 samples are obtained with a sampling interval of 1min 30seconds. The low sampling time is used in order to capture the dynamic properties of the system. The steady state DO concentration as can be seen from the Figure 2.8 is 2.5mg/l. However the primary goal of control is to maintain the dissolved oxygen concentration at 2mg/l level in the last tank. Therefore, the data obtained from this model will be used in chapter 4 to design a neural network controller to achieve the specified goal. The 960 data points sampled will be used to train neural network in order to identify the DO concentration model.

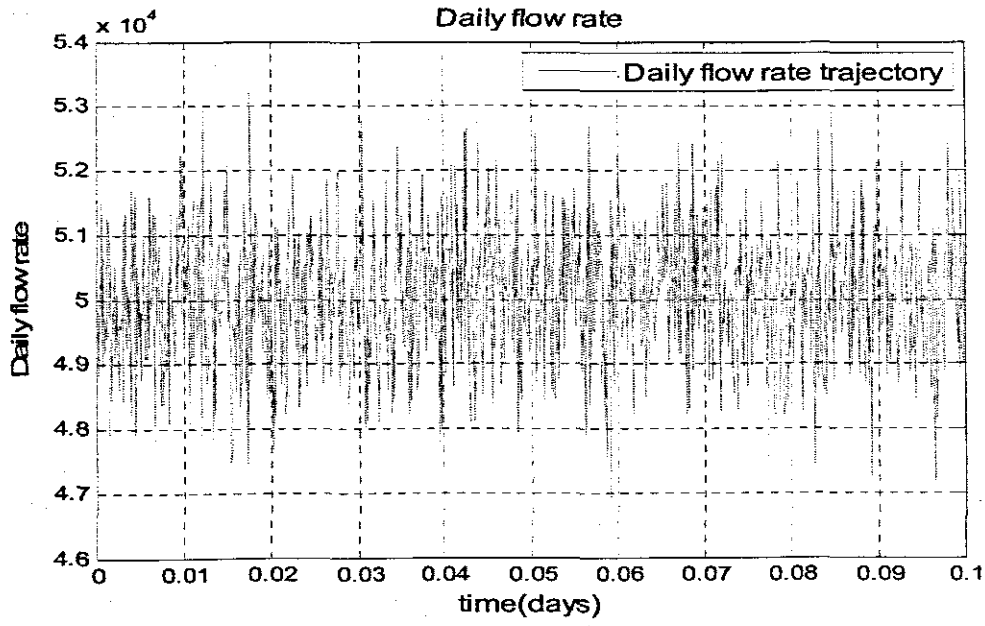


Figure 2.7: Graph showing the Airflow rate

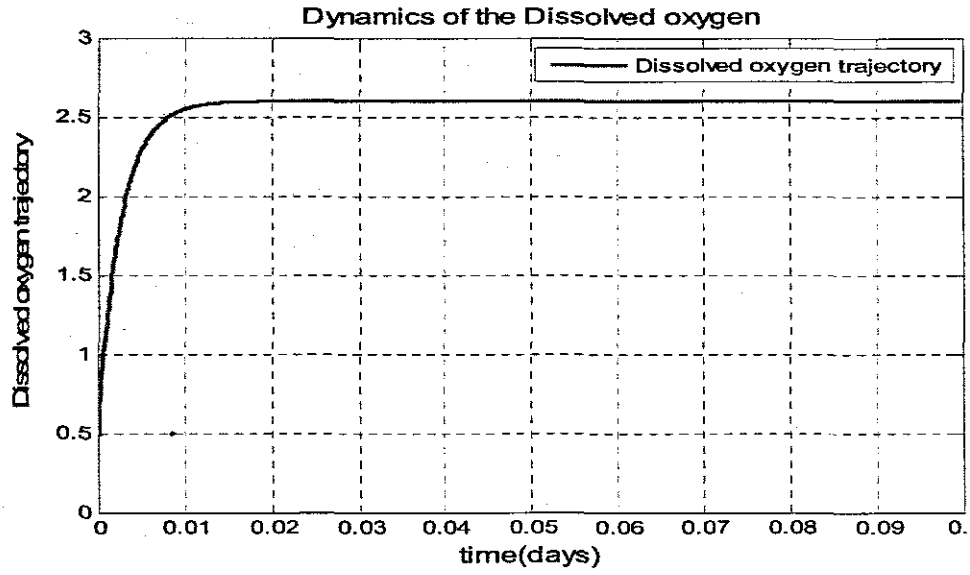


Figure 2.8: Time response of the DO concentration model.

2.6 Conclusion

This chapter provides a theoretical overview of Wastewater treatment process. It starts with an explanation of the operations of the various stages of the treatment process. Then emphasis is put on the biological process of wastewater treatment. The activated sludge Benchmark model based on ASM1 biological model is explored in depth since most of wastewater treatment processes use it for simulation and control verification. Sample literature review about the control design and model development for WWTP and for DO concentration is also provided.

The International Association on Water quality activated sludge model, ASM1(Henze et al, 1987), considered as the reference model with defined characteristic features, state variables and the basic processes used to describe the biological behaviour of the system is described. The equations describing the overall process rates (production, dead, consumption) for every process variables are given on the basis of Peterson's matrix.

In the literature review, it is found that the dissolved oxygen concentration control in the activated sludge plants is the most widely used manipulated variable since the level of the dissolved oxygen in the aerobic reactor plays a significant influence on the behaviour and activity of the heterotrophic and autotrophic microorganisms leaving in the activated sludge. It is also found that many different control strategies have been employed for dissolved oxygen concentration control. However, neural networks and fuzzy logics have gained attention to solve control problems especially for nonlinear

time-varying biological wastewater treatment plants because of the special features they posses. Thus the neural network theory is applied in this thesis for modelling and control design of the DO concentration.

Chapter 3 that follows examines artificial neural networks. It discusses artificial neural networks theory including the fundamental principles, the various structures and configurations and the neural network application techniques for Identification and *control of nonlinear processes*.

CHAPTER THREE

NEURAL NETWORKS THEORY AND OVERVIEW

This chapter discusses artificial neural networks theory and its application techniques for Identification and control of nonlinear processes. It covers the fundamental principles of neural networks, their biological inspiration from the central nervous system, various types of activation functions used and the various structures and configurations.

3.1 Introduction

The field of artificial neural networks have seen an explosion of interest over the years, and are being successfully applied across an extraordinary range of problem domains. Areas as diverse as finance, medicine, engineering, geology, defence, entertainment, manufacturing, medical, telecommunication, physics and indeed, anywhere that there are problems of prediction, artificial neural networks can be used. The success in the field of neural networks can be attributed to a few key factors such as: capability of modelling extremely complex functions and ease of use. From the control systems point of view, the ability to deal with non-linear systems is perhaps the most significant. The networks are used to provide the non-linear system models required by the techniques for synthesis of non-linear controllers.

3.2 Biological Inspiration

The original inspiration for the neural network technique was from examination of the central nervous system and the neurons (and their axons, dendrites and synapses) which constitute one of the most significant information processing elements. Figure 3.1 shows basic layout of a biological neuron (U.S national institution of health).

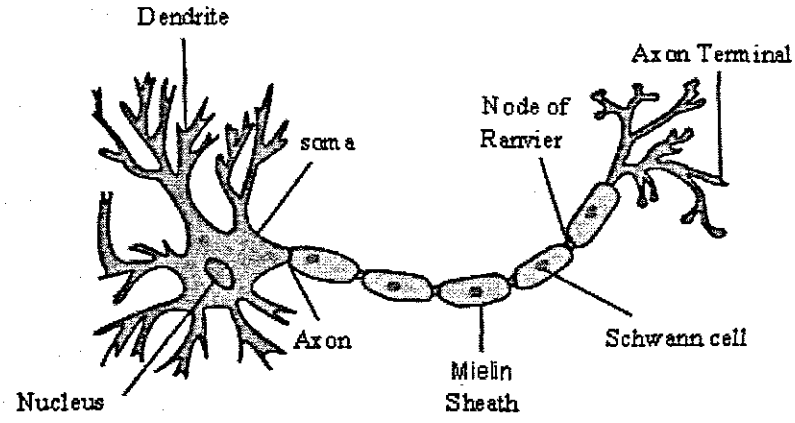


Figure 3.1: Structure of a typical neuron (U.S national institution of health).

An Axon or a nerve fiber is a long, slender projection of a nerve cell that conducts electrical impulses away from the neuron's cell body or soma. *Dendrites* are the branched projections of a neuron that act to control the electrochemical stimulation received from other neural cells to the cell body from which dendrites project. Electrical stimulation is transmitted onto dendrites by upstream neurons via *synapses* which are located at various points throughout the dendrite arbor. Dendrites play a critical role in integrating these synaptic inputs and in determining the extent to which action potentials are produced by the neuron.

The brain is principally composed of a very large number of neurons, massively interconnected with an average of several thousand interconnects per neuron. Each neuron is a specialized cell which can propagate an electrochemical signal. The neuron has a branching input structure (the dendrites), a cell body (soma), and a branching output structure (the axon). The axon of one cell connects to the dendrites of another via a synapse. When a neuron is activated, it fires an electrochemical signal along the axon. This signal crosses the synapses to the other neurons, which may in turn fire. A neuron fires only if the total signal received at the cell body from the dendrites exceeds a certain level termed the firing threshold. Thus, the strength of the signal received by a neuron will determine its chances of firing (Patterson, 1996) which critically depends on the efficacy of the synapses. Each synapse actually contains a gap, with neurotransmitter chemicals poised to transmit a signal across the gap. Thus, from a very large number of extremely simple processing units, each performing a weighted sum of its inputs, and then firing a binary signal if the total input exceeds a certain level, the brain manages to perform extremely complex tasks.

3.3 Artificial neural networks

In the most general sense, a neural network is a dynamic system consisting of simple processing units, called "neurons" or "nodes," and information passing links between these nodes called "interconnects" or "synapses," which can perform information-processing by responding to a set of input nodes containing information requiring processing. There are many definitions of neural networks available in the literature, a sample of these are quoted below:

- A neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes. This is according to the *DARPA Neural network Study* (1988, AFCEA International press, p.60).

- A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects (Haykin, 1994):

  1. Knowledge is acquired by the network through a learning process.
  2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.

- A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information. Furthermore each element operates asynchronously; thus there is no overall system clock (Nigrin, 1993).
- Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiential knowledge (Zurada, 1992),

Each neuron in a neural network usually combines the outputs from other neurons in the network or external inputs with values from previous computations to produce inputs to other neurons in the network. How the inter-neuron connections are arranged and the nature of the connections determines the structure of a network. The processing elements each have a number of internal parameters called weights. Changing the weights of an element will alter the behavior of the element and, therefore will also alter the behavior of the whole network (Pham, and Liu, 1996; Haykin, S., 1998). The goal here is to choose the weights of the network to achieve a desired input/output relationship. The process is known as training the network. How the weights (strengths) of the connections are adjusted or trained to achieve the desired overall behavior of the network is governed by its learning algorithm. An Artificial neural network is normally configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of artificial neural networks as well.

3.4 Historical background of Neural Networks

The field of neural networks simulation was established some decades ago, well before the advent of computers but has found solid application only in the past thirty years or so, and has also survived at least one major setback and several eras. However many important advances have been boosted by the use of computer emulations. The history of neural networks can be traced back to McCulloch and Pitts who published a paper (McCulloch, and Pitts, 1943) that described a formal calculus of networks of simple computing elements and these basic ideas that were developed by them form the basis of artificial neural networks. They developed models of neural

networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons which were considered to be binary devices with fixed thresholds. The results of their models were simple logic functions such as AND, and OR models. The technology available at that time did not allow them to do too much.

Then in 1949, Donald Hebb (Hebb, 1949) developed what is known as the 'Hebbian learning rule'. He found that if two connected neurons were simultaneously active, then the connection between them is proportionately strengthened. In other words, the more frequently a particular neuron is activated, the greater the weight between them (i.e., learning by weight adjustment). In 1958, Rosenblatt (1958) devised the *perceptron* model, which generated much interest because of its ability to solve some simple pattern classification problems. Perceptron had three layers, the input layer (a sensory area), the output layer (response area), with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit. The perceptron convergence theorem stated: that if a pattern can be learned by the perceptron, it means that it will be learned in finite number of training iterations.

However, the interest on neural networks started to fade in 1969 when Minsky and Papert [1969] provided mathematical proofs of the limitations of the single layer perceptron to multilayered systems and pointed out its weaknesses in computation. In particular, they proved that it was not capable of solving the classic exclusive-or (XOR) problem. It was only capable of classifying problems that were linearly separable at the output (Demuth *et al.*, 2004). Such drawbacks led to the temporary decline of the field of neural networks. Another system was the ADALINE (ADAptive LInear Element) which was developed in 1960 by Widrow and Hoff of Stanford University. The ADALINE was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron; it employed the Least-Mean-Squares (LMS) learning rule, Widrow and Hoff's learning rule (1960).

The developments of more-powerful networks, better training algorithms, and improved hardware have all contributed to the revival of the field of neural-networks. A significant breakthrough in neural networks was made by the introduction of the sigmoid activation function (Cowan, 1967). The function has the ability to approximate any nonlinear function. Instead of switching either on or off like the perceptron, this function turns the output value between plus and minus infinity, and squashes the output into the range 0 to 1. Neural network paradigms in recent years includes, the recurrent network (Hopfield's, 1982), Kohonen's network (1972), Rumelhart's competitive learning model (1986) with regards to back propagation techniques of training

multilayer perceptron, Fukushima's model, Carpenter and Grossberg's Adaptive Resonance Theory model [Wasserman 1989; Freeman and Skapura, 1991]. One of the reasons for the renewed excitement on neural networks was the paper by Rumelhart, Hinton, and McClelland (1986), which made the backpropagation algorithm famous. The paper discussed how back-propagation learning had emerged as the most popular learning set for the training of multi-layer perceptron. The backpropagation method could train multilayered neural networks to perform tasks that the simple perceptron had failed.

Thus, the field of neural networks has generated interest from researchers in such diverse areas as engineering, computer science, psychology, neuroscience, physics, and mathematics, etc. Today, significant progress is being made in the field of neural networks in many fronts and neural networks can be trained to solve problems that are difficult for conventional computers or human beings. Neurally based chips are emerging and applications to complex problems are being developed.

3.5 Neuron model and network Architectures

Figure 3.2 shows an artificial neuron. It includes a summer and an activation function *g*. It receives a number of inputs. Each input comes via a connection that has strength (*weight*). These weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value. The weighted sum of the inputs is formed, and the threshold subtracted, to compose the *activation* of the neuron. The activation signal is passed through an activation function, *g* (also known as a transfer function) to produce the output of the neuron.

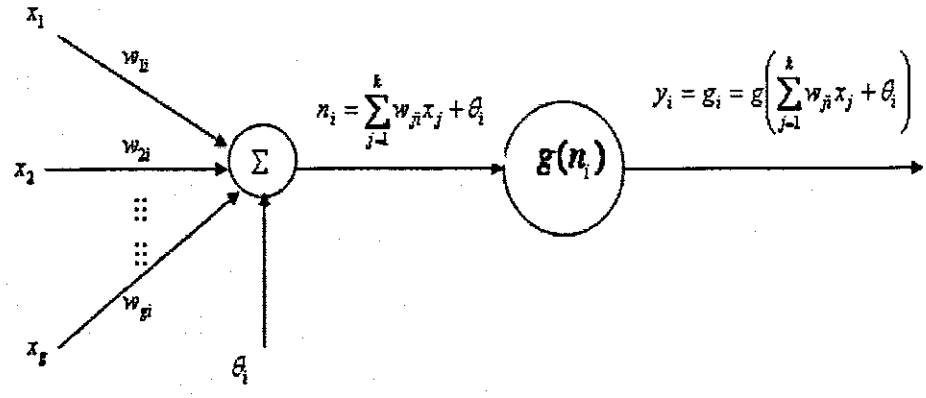


Figure 3.2: Single layer model of artificial neural network.

The inputs, $x_j, j = \overline{1,k}$ to the neuron are multiplied by weights w_{ji} and summed up together with the constant bias term θ_j. The resulting n_i is the input to the activation function *g*. The most commonly used activation functions are hyperbolic tangent (*tanh*), hard limit *(hardlim)*, pure linear (*purelin*), radial basis or a sigmoid functions. However there are other functions like the positive linear, saturating linear which are used but less popular. The output of node i becomes:

$$y_i = g_i = g\left(\sum_{j=1}^{k} w_{ji} x_j + \theta_i\right) \tag{3.1}$$

Two or more of the neurons can be combined in a layer, and a particular network could contain one or more such layers to form architecture or a network. In terms of their structures, neural networks can be divided into two types; namely, the Feed-forward networks and the Feedback networks (recurrent networks).

3.5.1 Feedforward networks

Feed-forward networks as shown in Figure 3.4 allows signals to travel one way only from input to output. There are no feedback loops, i.e. the output of any layer does not affect that same layer. The neurons are generally grouped into layers. Signals flow from input layer through to the output layer via unidirectional connections, the neurons being connected from one layer to the next, but not within the same layer.

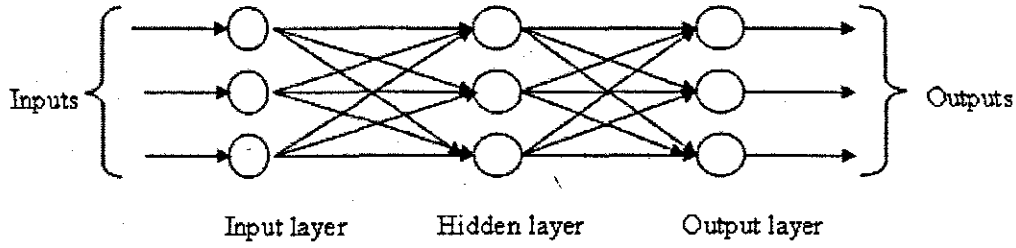


Figure 3.3: Feed-forward Neural networks.

Feed-forward neural networks tend to be straight forward networks that associate inputs to outputs. The feedforward neural networks were the first and arguably the simplest type of artificial neural networks to be devised. They are the most popular and most widely used models in many practical situations. Examples of feedforward networks include the multi-layer perceptron (Rumelhart and McClelland, 1986), the learning vector quantization network (Kohonen, 1989), the cerebellar model articulation control network (Albus, 1975a) and the group method of data handling network (Hecht-Nielsen, 1990). Feedforward networks can mostly naturally perform static mappings between input space and output space, the output at a given instant is a function only of the input at that instant.

3.5.2 Feedback networks

In the feedback neural networks (also known as recurrent neural networks), there are feedback connections. This means that they a closed loop topology. The outputs of some neurons are fed back to the same neurons or to neurons in the preceding layers. Thus signals can flow in both forward and feedback directions. Figure 3.4 shows an example of a partially connected recurrent neural network (Schalkoff, 1997).

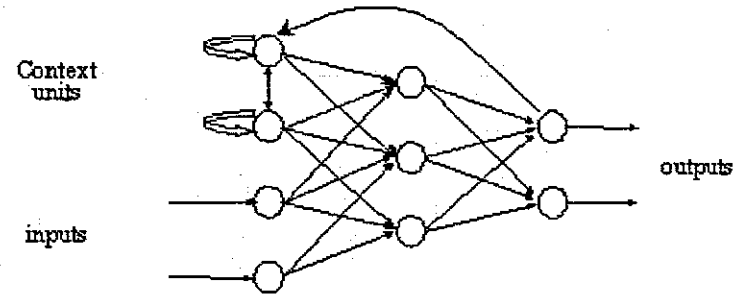


Figure 3.4: Example of partially connected recurrent neural network (Schalkoff, 1997).

Recurrent neural networks (RNN) were developed to deal with time varying or time-lagged patterns and are usable for problems where the dynamics of the considered process is complex and the measured data is noisy. A RNN is a network with feedback (closed loop) connections (Fausett, 1994). Examples of recurrent networks include BAM, Hopfield, Boltzmann machine (Hopfield, 1982) and the recurrent backpropagation networks (Hecht-Nielsen, 1990).The architectures range from fully interconnected to partially connected networks. In a fully connected RNN all hidden units are connected recurrently, whereas in a partially connected RNN, the recurrent conditions are omitted partially. Recurrent networks have a dynamic memory in that there outputs at a given instant reflect the current input as well as the previous inputs and outputs.

3.6 Neural networks activation functions

Two functions determine the way signals are processed by neurons. The activation function acting on the input vector determines the total signal a neuron receives, and the output function, operating on a scalar activation, determines the scalar output. The composition of the activation and the output functions is called the transfer function. For some transfer functions there is no natural division between activation and output functions. Typically, activation functions have a "quashing" effect such that the output of a neuron in a neural network is between certain values (usually 0 and 1, or -1 and 1). There is a wide range of activating functions in use with neural networks. Only a few of these are used by default; others are available for customization.

Also, activation functions are needed to introduce nonlinearity into the network. Without nonlinearity, hidden units would not make networks more powerful than just plain perceptron (which do not have any hidden units, just input and outputs units). For backpropagation learning, the activation function must be differentiable, and helps if the function is bounded. The sigmoid functions such as *logistic* and *tanh* and the *Gaussian functions* are the most common choice. Functions such as *tanh* or *arc tan* that produce only positive and negative values tend to yield faster training than functions that produce only positive values such as *logistic*, because of better numerical conditioning. A brief overview of the characteristics of the most common activation functions in use is illustrated (Pham, D.T, 1994).

1. *Threshold function:* For this type of activation function, described by the characteristic of Figure 3.5,can be written as:

$$g(n) = \begin{cases} 1 \; if \; n \geq 0 \\ 0 \; if \; n\langle 0 \end{cases} \tag{3.2}$$

In engineering, this form of a threshold function is commonly referred to as a *Heaviside function*. Correspondingly, the output of neuron *i* employing such a threshold function is expressed as:

$$y_i = \begin{cases} 1 \; if \; g_i \geq 0 \\ 0 \; if \; g_i\langle 0 \end{cases} \tag{3.3}$$

Where g_i is the induced local field of the neuron: that is,

$$y_i = g_i = g\left(\sum_{j=1}^{k} w_{ji} x_j + \theta_i\right) \tag{3.4}$$

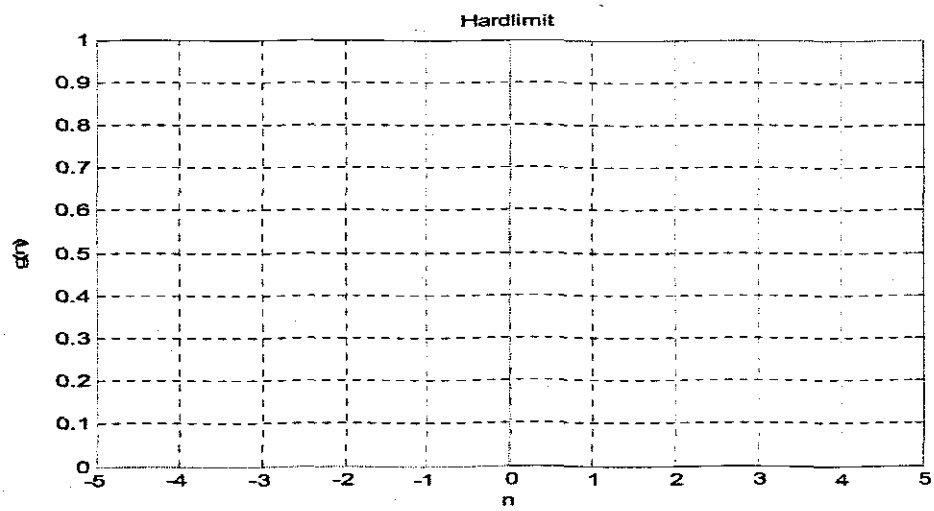


Figure 3.5: Threshold Activation function

The shown hard-limit transfer function limits the output of the neuron to either 0, if the input argument is less than **0**, or **1**, if **n** is greater than or equal to **0**. This function is normally used in perceptions to create neurons that make classification decisions.

2. *The linear transfer function.* The linear activation function is essentially no activation at all. It is probably the least commonly used of the activation functions. The linear activation function does not modify a pattern before outputting it. Neurons of this type are used as linear approximators. By simply returning the value passed to it, the function performs linear approximation. The linear activation function may be useful in situations when entire range of numbers to be outputted is needed. Figure 3.6 shows the characteristics of this function which can be generated by the equation:

$$a = purelin\ (n). \qquad (3.5)$$

Neurons of these types are used as linear approximators in "Adaptive Linear Filters".

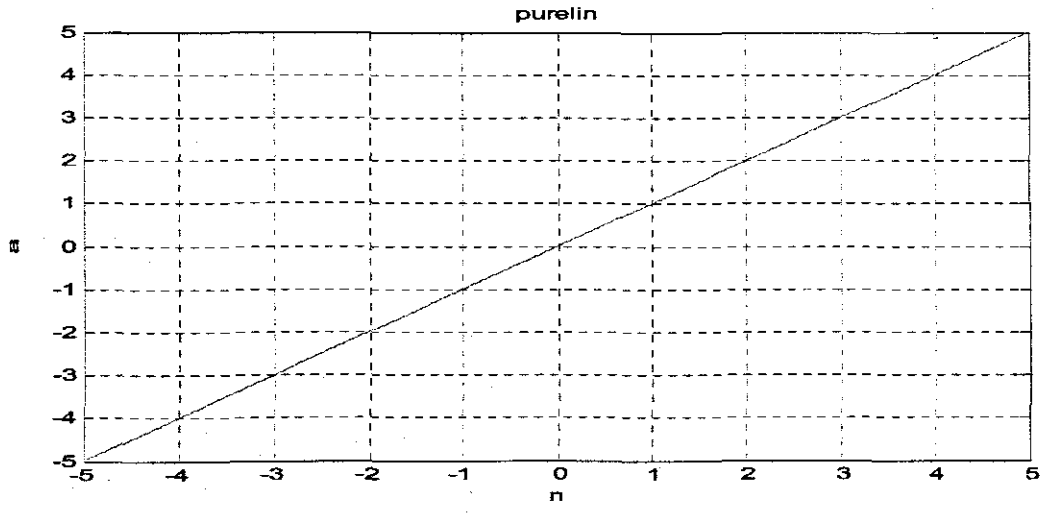


Figure 3.6: Pure linear Activation function.

3. *Sigmoid transfer function:* The sigmoid transfer function shown in Figure 3.7 takes the input, which can have any value between plus and minus infinity, and quashes the output into the range **0** to **1**.This transfer function is commonly used in backpropagation networks as shall be seen later, because this function is differentiable. The sigmoid activation function plays a critical role in neural network modelling. The purpose of the sigmoid transfer function to the neural network design is to normalize and shrink the weight estimates in the hidden layer or output layer with a distribution that is centered about zero in order to assure convergence to the correct values.

The sigmoid function is defined by $f(n) = \frac{1}{1+e^{-n}}$.The term sigmoid means curved in two directions, like the letter **"S"**. One important thing about sigmoid activation function is that it only returns positive values. If the neural network is needed to return negative numbers, then the sigmoid function will be unsuitable. The characteristics of this function can be generated by the equation:

a = logsig (n) (3.6)

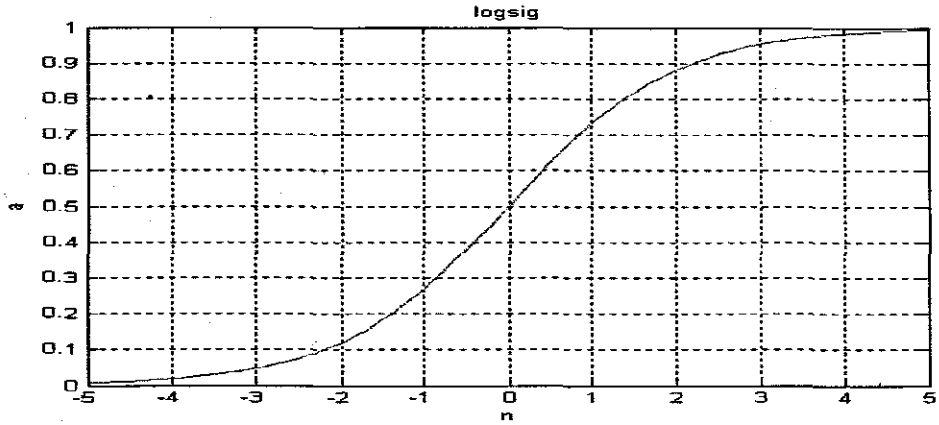


Figure 3.7: Logistic Sigmoid Activation function.

4 *Hyperbolic Tangent Function:* Another common choice of activation function in the neural network is the hyperbolic tangent function. One reason is that it is a very flexible, non-linear function. It is also continuous and differentiable. The hyperbolic tangent and sigmoid functions are equivalent. The difference is that the sigmoid activation function does not return values less than zero. However, it is possible to "move" the sigmoid function to a region of the graph so that it does provide negative numbers. This is done using hyperbolic tangent function. The hyperbolic tangent function has an ideal network modeling property, in that the function leads to faster convergence in optimization process with comparison to the various logistic functions given a sufficiently large sample sizes. The equation for the hyperbolic function is given by $f(n) = \frac{e^{2n}-1}{e^{2n}+1}$.The graph of the hyperbolic function is shown in Figure 3.8. One important thing about hyperbolic function is that it returns both positive and negative values.

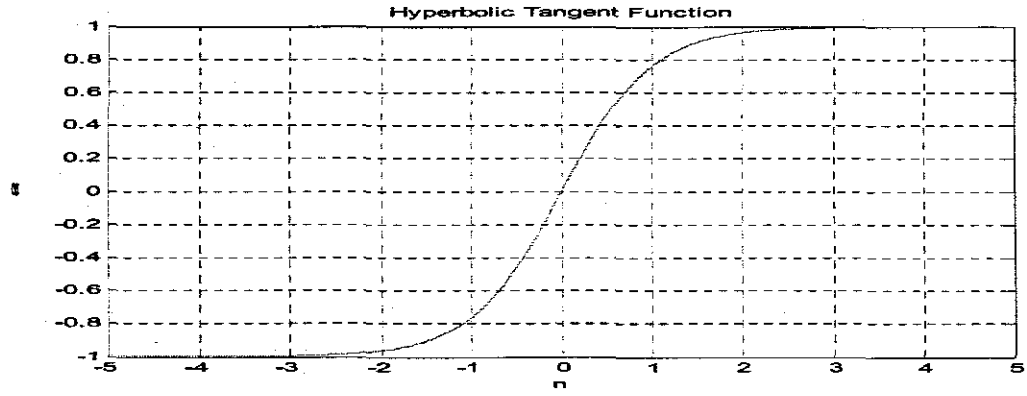


Figure 3.8: Hyperbolic Tangent Activation function

5. *The Radial Basis Function:* Radial functions are usually Gaussian. They are used in approximation theory and in pattern recognition (Haykin, S., 1994). The procedure starts with a vector of inputs. The distance between the inputs and the vector of weights is calculated, multiplied by the vector of bias and sent to the radial basis function. This can be expressed as a function:

a = radbas (n). (3.7)

The commonly used radial basis activation functions are the multi quadratic functions and the Gaussian function given by $radbas(n) = e^{-n^2}$ as is illustrated in Figure 3.9. Typically the radial basis functions are used in radial basis networks to construct local approximations to nonlinear input-output mapping, with the result that the networks are capable of fast learning and reduced sensitivity to the order of presentation of training data.

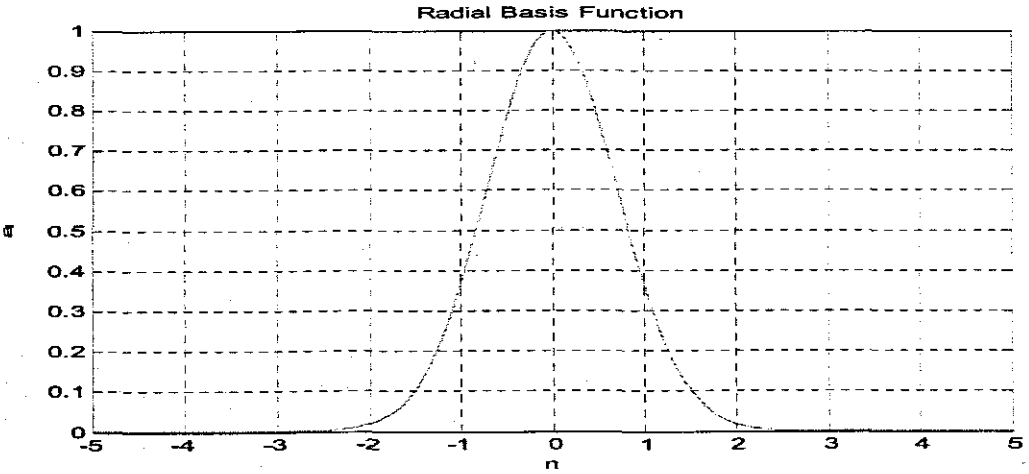


Figure 3.9: Gaussian Radial Basis Activation function

3.7 Neural networks' learning categorisation

Updating the weights by training the neural networks is known as the learning feature of the neural networks. Learning algorithms may be carried out in continuous-time (via

differential equations for the weights), or in the discrete-time (via difference equations for the weights). Learning algorithms in neural networks are many but can be broadly classified into three basic types: supervised, unsupervised and reinforced learning methods.

3.7.1 Supervised learning

In this method, every input pattern that is used to train the network is associated with an output pattern, which is the target or the desired pattern. Thus, a supervised learning algorithm adjusts the strength or weights of the inter-neuron connections according to the difference between the desired and the actual network outputs corresponding to a given input. The supervised learning requires a teacher or a supervisor to provide desired or target output signals. The difference (error) can then be used to change the network parameters, which results in an improvement in performance. Examples of supervised learning algorithms include the delta rule (Widrow and Hoff, 1960), the generalized delta rule or backpropagation algorithm (Rumelhart and McClelland, 1986), and the learning vector quantization algorithm (Kohonen, 1989).

3.7.2 Unsupervised learning

Unsupervised learning algorithms (also called self-organizing behavior) do not require the desired outputs to be known. The target output is not presented to the network. During training, only input patterns are presented to the network which automatically adapts the weights of its connections to cluster the input patterns into groups with similar features. It is as if there is no teacher to present the desired patterns as the system learns on itself. Examples of unsupervised learning algorithms include the Kohonen (Kohonen, 1989), the Adaptive Resonance Theory (Carpenter and Grossberg, 1988) competitive learning algorithms.

3.7.3 Reinforced learning

This method can be regarded as a special form of supervised learning. Instead of using a teacher to give target outputs, a reinforcement learning algorithm employs a critic only to evaluate the goodness of the neural network output corresponding to a given input. The weights associated with a particular neuron are not changed proportionally to the output error of that neuron, but instead are changed in proportion to some reinforcement signal. However, reinforced learning is not one of the popular forms of learning. An example of a reinforcement learning algorithm is the genetic algorithm (Holland, 1975; Goldberg, 1989).

3.8 Types of neural networks

There are quite a number of neural networks configurations available in the present world, among them; the multi-layer perceptron (MPL), the Learning Vector Quantization (LVQ) network, the Cerebellar Model Articulation Control (CMAC) network, the Group Method of Data Handling (GMDH) network, the Hopfield network, the Elman and Jordan networks, the Kohonen network, and the Adaptive Resonance Theory (ART) network. For an overview of different systems engineering application to these networks reference is made in (Pham, 1994). However, the most important classes of neural networks for real world problem solving which are covered in this literature review include:

- Multi-layer Perceptron.
- Radial Basis Function Networks.
- Kohonen Self organizing Feature Maps.

3.8.1 Multi-layer Perceptron (MLP)

A perceptron neural network with one or more layers of nodes between the input and output nodes is called multi-layer perceptron network. MPLs are perhaps the best known type of feedforward neural networks in use today, due to its originality to Rumelhart and McClelland (1986) and discussed at length on most neural network textbooks and journals. It is also the most popular form of neural network architecture possessing the following features:

- Has any number of inputs.
- Has one or more hidden layers with any number of units
- Uses linear combination functions in the input layers.
- Uses generally sigmoid activation function in the hidden layers.
- Has any number of outputs with any activation function.
- Has connections between the input layer and the first hidden layer, between the hidden layers and between the last hidden layer and the output layer, i.e. signals flow in the forward directions only.

Given enough data, enough hidden units, and enough training iterations, a multi-layer perceptron with only one hidden layer can approximate virtually any function to any degree of accuracy. For this reason, multi-layer perceptron are known as universal approximators and can be used to approximate virtually any function (Hornik, et al, 1989). The number of layers, and the number of units in each layer, determines the function's complexity. Important issues in the multi-layer perceptron design therefore include specification of the number of hidden layers and the number of units in these layers (Haykin, 1994; Bishop, 1995).

The topology representation of a multi-layer perceptron network with one input layer with three nodes, one hidden layer with four nodes and an output layer with two nodes is illustrated in Figure 3.10.

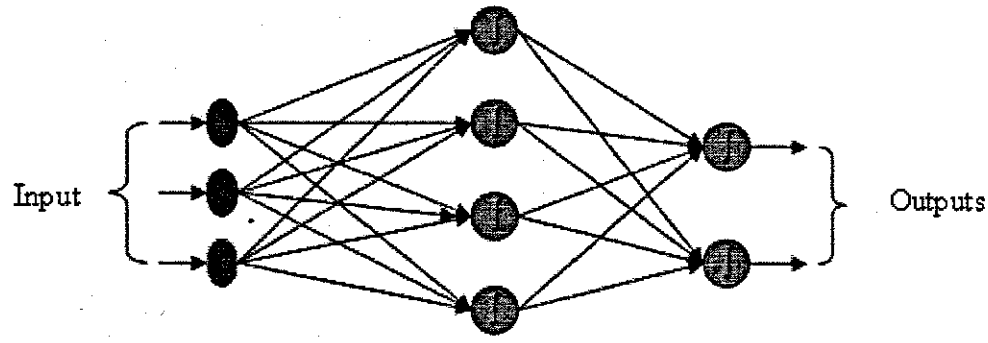


Figure 3.10: Multilayer Perceptron Neural networks.

3.8.2 Radial Basis Function Networks

Radial basis functions (RBF) networks are also feedforward, but they have only one hidden layer. In a nutshell, the RBF networks posses the following features:

- Has any number of inputs.
- Typically has one hidden layer with any number of units.
- Uses radial combination functions in the hidden layer, based on the squared Euclidean distance between the input vector and the weight vector.
- Typically uses exponential or Gaussian activation functions in the hidden layer, in which case the network is a Gaussian RBF network.
- Has any number of outputs with any activation function.
- Have connections between the input layer and the hidden layer, and between the hidden layer and the output layer.

Figure 3.11 illustrates an RBF network with inputs $\overline{x_1, x_n}$ and output y. The arrows in the figure symbolize parameters in the network. The RBF network consists of one hidden layer of basis functions, or neurons. At the input of each neuron, the distance between the neuron center and input vector is calculated. The output of the neuron is then formed by applying the basis function to this distance. The RBF network is formed by a weighted sum of the neuron outputs and the unity bias. The governing equation is:

$$y(x) = \sum_{i=1}^{N} w_i \rho(\|x - c_i\|) \tag{3.8}$$

Where N is the number of neurons in the hidden layer, c_i is the center of the vector neuron i, and w_i are the weights of the linear output neuron. In the basic form all inputs are connected to each hidden neuron. The norm is typically taken to be the Euclidean distance and the basis function ρ is taken to be Gaussian.

$$\rho(\|x - c_i\|) = \exp[-\beta\|x - c_i\|]^2 \tag{3.9}$$

The Gaussian basis functions are local in the sense that

$$\lim_{\|x\| \to \infty} \rho(\|x - c_i\|) = 0 \,. \tag{3.10}$$

This means that changing parameters of one neuron has only a small effect for input values that are far away from the center of that neuron. RBF networks are universal approximators. Provided that RBF network has enough hidden neurons it can approximate any continuous function with arbitrary precision. The weights w_i, c_i and β are determined in a manner that optimizes the fit between the output *y* and the input data.

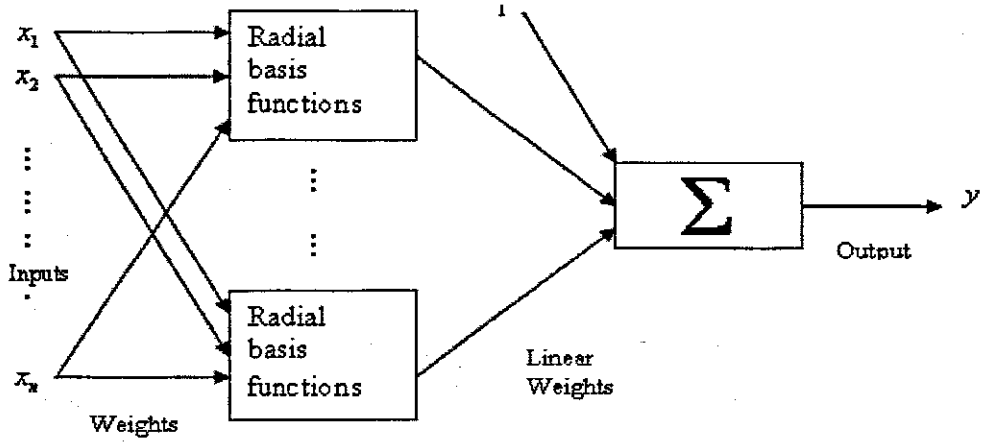


Figure 3.11: Radial Basis Function network

The basis Gaussian bell function is the most commonly used, although other basis functions may be chosen. RBF networks are normally used to solve a common set of problems like, Function approximation, classification, modeling of dynamic systems and time series prediction (Moody and Darken, 1989; Roger D. Jones *et al*, 1990).

3.8.3 Kohonen Self organizing Feature Maps

Self Organizing Feature Map (SOFM, or Kohonen) networks are quite different to the other networks. Whereas all the other networks are designed for supervised learning, the SOFM networks are designed primarily for unsupervised learning (Patterson,

Where N is the number of neurons in the hidden layer, c_i is the center of the vector neuron i, and w_i are the weights of the linear output neuron. In the basic form all inputs are connected to each hidden neuron. The norm is typically taken to be the Euclidean distance and the basis function ρ is taken to be Gaussian.

$$\rho(\|x - c_i\|) = \exp[-\beta\|x - c_i\|]^2 \tag{3.9}$$

The Gaussian basis functions are local in the sense that

$$\lim_{\|x\|\to\infty} \rho(\|x - c_i\|) = 0. \tag{3.10}$$

This means that changing parameters of one neuron has only a small effect for input values that are far away from the center of that neuron. RBF networks are universal approximators. Provided that RBF network has enough hidden neurons it can approximate any continuous function with arbitrary precision. The weights w_i, c_i and β are determined in a manner that optimizes the fit between the output *y* and the input data.

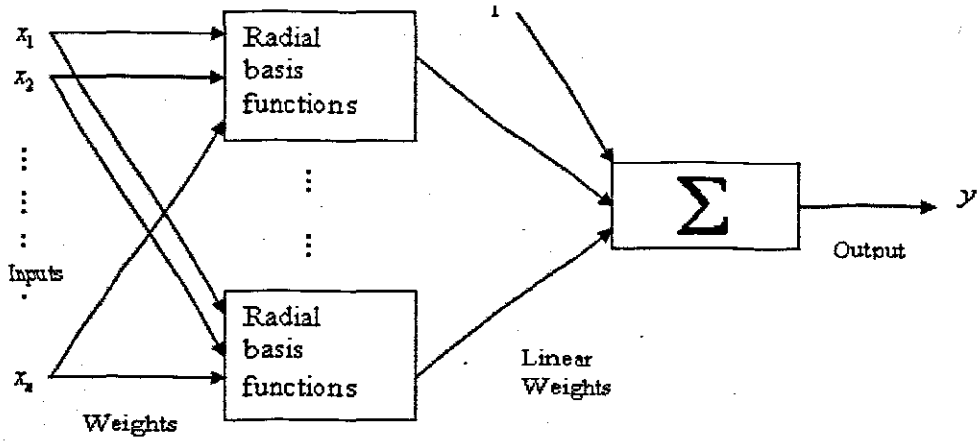


Figure 3.11: Radial Basis Function network

The basis Gaussian bell function is the most commonly used, although other basis functions may be chosen. RBF networks are normally used to solve a common set of problems like, Function approximation, classification, modeling of dynamic systems and time series prediction (Moody and Darken, 1989; Roger D. Jones *et al*, 1990).

3.8.3 Kohonen Self organizing Feature Maps

Self Organizing Feature Map (SOFM, or Kohonen) networks are quite different to the other networks. Whereas all the other networks are designed for supervised learning, the SOFM networks are designed primarily for unsupervised learning (Patterson,

1996). A Kohonen network or self-organizing feature map has two layers, an input buffer layer to receive the input pattern and an output layer. Neurons in the output layer are usually arranged into a regular two-dimensional array. Each neuron is connected to all input neurons. The weights of the connections form the components of the reference vector associated with the given output neuron. The Kohonen technique creates a network that stores information in such a way that any topological relationships within the training set are maintained. Figure 3.12 shows the structure of a Kohonen network.

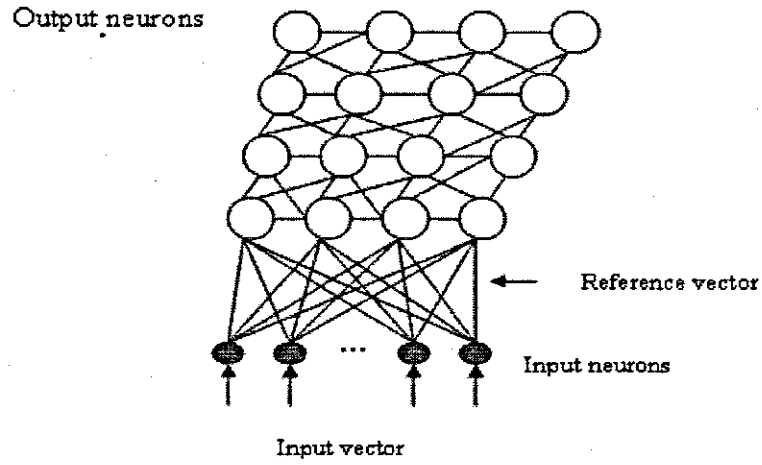


Figure 3.12: Kohonen network.

Training of a Kohonen network involves the following steps:

i. Initialise the reference vectors of all output neurons to small random values;
ii. Present a training input pattern;
iii. Determine the winning output neuron, i.e. the neuron whose reference vector is closest to the input pattern. The Euclidean distance between a reference vector and the input vector is usually adopted as the distance measure;
iv. Update the reference vector of the winning neuron and those of its neighbours. These reference vectors are brought closer to the input vector. The adjustment is greatest for reference vector of the winning neuron and decreased for reference vectors of neurons further away. The size of the neighbourhood of a neuron is reduced as the training proceeds until, towards the end of training, only the reference vector of the winning neuron is adjusted.

In a well-trained Kohonen network, output neurons that are close to one another have similar reference vectors. After training, a labelling procedure is adopted where input patterns of known classes are fed to the network and class labels are assigned to output neurons that are activated by those input patterns. That means that an output

neuron is activated if it wins the competition against other output neurons, if the reference vector is closest to the input pattern.

3.9 Training Algorithms

Training a neural network to learn patterns in the data involves iteratively presenting it with examples of correct known answers. The objective of training is to find the set of weights between neurons that determine the global minimum error function. This involves decision regarding the number of iterations, i.e., when to stop training and the selection of the learning rate (a constant of proportionality which determines the size of the weight adjustments made after iteration), and momentum values (how past weight changes affect current weight changes). The procedure for selecting the network parameters (weights and biases) for a given problem is called training the network. There are a number of algorithms for training; most of them can be viewed as a straightforward application of optimization theory and statistical estimation. Given a cost function, there are several ways to find an optimal solution. Optimization techniques can be categorized onto the following approaches:

- ***Calculus Gradient techniques*** which adjust a parameter θ based on the sensitivity ($\Delta J / \Delta \theta$) = variation of cost function over variation of parameter). Examples include the Delta Rule, Hill Climbing, Back-Propagation, etc.
- ***Artificial Intelligent Heuristic techniques*** which expand a branch in a tree search based on evaluation of cost function. Examples include Breadth first, Depth first, A* algorithm, etc.
- ***Evolution type techniques*** which use a population of parameters to evolve into better and better generations of populations. Examples are the Evolution algorithm, the Genetic algorithm, Genetic programming etc.

3.9.1 The Delta Rule

Suppose a cost function $J(\theta)$ which has a minimum as shown in the Figure 3.13 is considered. The Delta rule for searching for a minimum is to:

- Step backwards if the gradient is uphill
- Step forward if the gradient is downhill
- Stop if the gradient is close to being flat,

θ may be a vector, although drawn in the figure as a scalar.

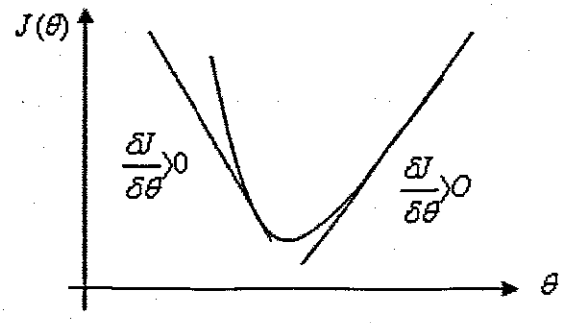


Figure 3.13: The Delta Rule.

Mathematically the rule can be described as: Change in parameter value

$$\Delta\theta = -\gamma \frac{\delta J}{\delta \theta} \tag{3.11}$$

Parameter update, $$\theta_{new} = \theta_{old} + \Delta\theta \tag{3.12}$$

Where, γ = a factor that determines the search step size. Back-propagation is a well-known gradient search technique for training feed-forward neural networks that is based on the Delta Rule. Many of the training algorithms are based on the gradient decent algorithm. The following section describes the backpropagation algorithm.

3.9.2 Back-propagation algorithm

Back-propagation can train multi-layer feed-forward networks with differentiable transfer functions to perform function approximation, pattern association, and pattern classification. It has been shown (Hornik, Stinchcombe, & White, 1989; Cybenko, 1989; Hartman, Keeler, & Kowalski, 1990) that only one layer of hidden units can successfully approximate any function with finitely many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear. The proof is given by the universal approximation theorem. Other types of networks can be trained as well, although the multilayer network is commonly used.

The back-propagation learning law is a very popular and consequently important learning rule applied to multi-layer feedforward neural networks. Because of its popularity, multi-layer perceptron are often referred to as the **back-propagation neural networks**. The back-propagation law is a supervised error-correction rule in which the output error, that is, the difference between the desired and the actual output is propagated back to the hidden layers. Now, if the error output at each layer can be determined, it is possible to apply any method which minimizes the performance index to each layer sequentially. A common cost function (performance index) is the sum of the squared errors (A.P.Paplinski, 2002),

$$J = \frac{1}{2}\sum_r \varepsilon_r^{\,2} \tag{3.13}$$

The summation is over all the output neurons of the network. The network is trained by minimizing J with respect to the weights, and this leads to the gradient method. The factor 1/2 is used to simplify the differentiation when minimising J. Consider a single layer of nonlinear neurons as in Figure 3.14 (A.P.Paplinski, 2002), where h is the input signal, W is the $L \times m$ weight matrix in the input layer, $\frac{dy}{du}$ is the derivative of the output signal with respect to the manipulate signal u, ϑ is the activation function (e.g. sigmoid) and δ is the delta error.

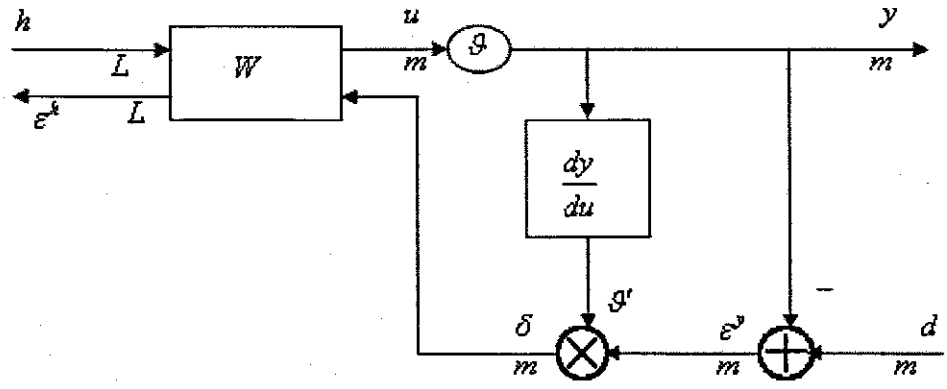


Figure 3.14: Back-Propagation Errors (A.P.Paplinski, 2002).

If the error measured at the neuron's output is

$$\varepsilon^y = d - y \tag{3.14}$$

where, d is the desired output,

Then an equivalent input error, ε^h that is, an input perturbation which would cause the output error, ε^y can be determined as:

$$\varepsilon^h = W^T \cdot \delta; \delta = [\delta_1 \ldots\ldots\ldots\ldots \delta_m]^T \tag{3.15}$$

Where the *k*-th delta signal is

$$\delta_k = \vartheta_k' \cdot \varepsilon_k^{\,y} = \frac{dy_k}{du_k} \cdot \varepsilon_k^{\,y} \tag{3.16}$$

There are several different types of back-propagation training algorithms. They have a variety of different computation and storage requirements, and no one algorithm is best suited to all applications.

The back-propagation algorithm has the following steps (Lippmann, 1987) as follows:

- *Initialise weights*. Set all weights to small random values.

- *Present inputs and desired outputs (training pairs).* Present an input vector u and specify the desired outputs d. If the network is used as a classifier then all the desired outputs are typically zero, (except one set to a 1). The inputs could be new on each trial, or samples from a training set could be presented cyclically until weights stabilize.
- *Calculate actual outputs.* Calculate the outputs by successive use of $y = f(W^T \cdot u)$ where f is a vector of the activation functions and W^T is a vector of the weights.
- *Adapt weights.* Start at the output neurons and work backwards to the first hidden layer while adjusting weights by: $w_{ji} = w_{ji} + \eta\delta_j y_i$ (3.17)

In equation (3.17) w_{ji} is the weight from the hidden neuron i (or an input node) to neuron j, y_i is the output of neuron i (or an input), η is the learning rate parameter, and δ_j is the gradient,

- *Go to step 2*

The standard back-propagation algorithm suffers from a number of problems. The main problems are:

- If it does converge (this usually happens when the learning rate is small), it will be very slow.
- Training performance is sensitive to the initial conditions
- Oscillations may occur during learning (this usually happens when the learning rate is high).
- If the error function is shallow, the gradient is very small leading to small weight changes. This is known as the "flat spot" problem.
- The convergence to a local minimum of the error function can occur under certain circumstances and there is no proof that the standard back-propagation algorithm finds a global minimum.

For these reasons several methods of speeding up back-propagation algorithm for quick convergence have been proposed (Donald L. Prados, 1992; Martin Riedmiller, 1994; Femando, S. *et al*, 1990; Yoshio and Hijiya, 1991; Dilip Sarkar, 1995). These high performance algorithms can converge from ten to hundred time's faster than back-propagation algorithms. These faster algorithms fall into two main categories: heuristic technique (variable learning rate back-propagation, resilient back-propagation) and numerical optimization techniques (Conjugate gradient, quasi-Newton, Levenberg-Marquardt (Bishop, 1995; Shepherd, 1997). Levenberg-Marquardt is the fastest algorithm but as the number of weights and biases in the network

increase, the advantage of this algorithm decreases. From an optimization point of view, learning in a neural network is equivalent to minimising a global error function, which is a multivariable function that depends on the weights in the network.

3.9.2.1 Conjugate gradient method (Bishop, 1995)

The idea here is that, once the algorithm has minimized along a particular direction, the second derivative along that direction should be kept to zero. Conjugate directions are selected to maintain this zero second derivative on the assumption that the surface is a nice smooth surface. If this condition holds, N epochs (where N is the number of iterations it takes to converge) are sufficient to reach a minimum. In reality, on complex error surface the conjugate deteriorates, but the algorithm still typically require far less epochs than back-propagation, and also converges to a better minimum.

3.9.2.2 Quasi-Newton method

Quasi-Newton training is based on the observation that the direction pointing directly towards the minimum on a quadratic surface is the so-called Newton direction. This is very expensive to calculate analytically, but quasi-Newton iteratively builds up a good approximation to it. Quasi- Newton method is usually a little faster than the conjugate gradient descent, but has substantially larger memory requirements and is occasionally numerically unstable

3.9.2.3 Levenberg-Marquardt method

Levenberg-Marquardt (Levenberg, 1994; Marquardt, 1963; Bishop, 1995) is typically the fastest of the training algorithms, although it has some limitations, specifically: it can only be used on single output networks, can only be used with the sum squared error function, and has memory requirements to the number of weights in the network; this makes it impractical for reasonably big networks.

3.9.2.4 Generalization (Interpolation and Extrapolation)

Generalization can be defined as the ability to have the outputs of the neural networks approximate target values given the inputs that are not in the training set. Generalization requires a prior knowledge of what the relevant inputs are in order for the network to generalize. The critical issue in developing a neural network is Generalization, i.e. how well will the network make predictions for cases that are not in the training set. There are three conditions that are typically necessary for good generalization:

- The inputs to the network must contain sufficient information pertaining to the target, to ensure the existence of a mathematical function relating correct outputs to

inputs with the desired degree of accuracy. Finding good inputs for a network and collecting enough training data often takes far more time and effort than training the network.

- The function to be learnt must be smooth. In other words, a small change in the inputs should, most of time, produce a small change in the outputs. Often a nonlinear transformation of the input space can increase the smoothness of the function and improve generalization.
- The training cases should be sufficiently large and be representative subsets of the set of all classes that is to be generalized. The importance of this condition is related to the fact that there are, two different types of generalization: interpolation and extrapolation. Interpolation applies to the cases that are more or less surrounded by nearby training case; everything else is extrapolation. Interpolation can often be done reliably, but extrapolation is notoriously unreliable.

Neural networks, like other flexible nonlinear estimation methods such as kernel regression, and smoothing splines, can suffer from either under-fitting or over-fitting. A network that is not sufficiently complex can fail to fully detect the signal in a complicated data set, leading to under-fitting. Similarly, a network that is too complex may fit the noise, not just the signal, leading to over-fitting. Over-fitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of neural networks. Over-fitting can also produce wild predictions in multilayer perceptron even with noise-free data (Smith, 1996; Geman *et al*, 1992).

While there is little to be done to improve the network performance outside the range of training data, the ability to interpolate between data points can be improved. Given a fixed number of training data, there are at least six approaches to avoiding underfitting and overfitting, and hence getting good generalization. These are: model selection, Jittering, Early stoppage, Weight decay, Bayesian learning, and combining networks.

- *Model selection*: The complexity of a network is related to both the number of weights and the size of the weights. Model selection is concerned with the number of weights, and hence the number of hidden units and layers. The more weights there are, relative to the number of training cases, the more overfitting amplifies noise in the targets (Moody 1992).
- *Jittering:* Jitter is artificial noise deliberately added to the inputs during training. Training with jitter is a form of smoothing related to kernel regression. It is also closely related to regularization methods such as weight decay and ridge regression. Training with jitter works mostly with smooth functions. In other words, if

there are two cases with similar inputs, the desired outputs will usually be similar. That means, any training case can be taken and new cases training can be generated by adding small amounts of jitter to the inputs. As long as the amount of jitter is sufficiently small, it can be assumed that the desired output will not change enough to be of any consequence. So the same target value can be used. The more training cases, the merrier, so this looks like a convenient way to improve training. But too much jitter will obviously produce garbage, while too little jitter will have little effect (Koistinen and Holmström 1992).

- *Early stoppage*: A portion of the training data is placed into a validation data set. The performance of the network on the validation set is monitored during training. During the early stages of training the validation error will come down. When overfitting begins, the validation error will begin to increase, and at this point the training is stopped. However, the validation error is not a good estimate of the generalization error. One method for getting an unbiased estimate of the generalization error is to run the network on a third set of data, the test set that is not used at all during the training process.
- *Weight decay*: Weight decay adds a penalty term to the error function. The usual penalty is the sum of squared weights times a decay constant. Weight decay is a subset of regularization methods. The penalty term in weight decay, by definition, penalizes large weights. The weight decay penalty term causes the weights to converge to smaller absolute values than they otherwise would. Other regularization methods may involve not only the weights but various derivatives of the output function (Bishop 1995).
- *Bayesian Learning*: The Bayesian School of statistics is based on a different view of what it means to learn from data, in which probability is used to represent uncertainty about the relationship being learned. The result of Bayesian training is a posterior **distribution** over network weights. If the inputs of the network are set to the values for some new case, the posterior distribution over network weights will give rise to a distribution over the outputs of the network, which is known as the **predictive distribution** for this new case. If a single-valued prediction is needed, one might use the mean of the predictive distribution, but the full predictive distribution also tells how uncertain this prediction is.
- *Regulization*: With this method the performance index is modified to include a term which penalizes network complexity. The most common penalty term is the sum of squares of the network weights. This performance index forces the weights to be small, which produces a smoother network response.

3.10 Neural networks and control

With specific reference to neural networks in control the following characteristics and properties of neural networks are important as compared with other conventional modelling methods (K. J. Hunt et al, 1992; Moscinski, J. and Ogonowski, Z, 1995):

- *Nonlinear systems.* Neural networks have greatest promise in the realm of nonlinear control problems. This stems from their theoretical ability to approximate arbitrary nonlinear mappings. Networks may also achieve more parsimonious modeling than alternative approximation schemes.
- *Parallel distributed processing.* Neural networks have a highly parallel structure which lends itself immediately to parallel implementation. Such an implementation can be expected to achieve a higher degree of fault tolerance than conventional schemes. The basic processing element in a neural network has a simple structure. This, in conjunction with parallel implementation, results in very fast overall processing.
- *Hardware implementation.* Not only can the networks be in parallel, a number of vendors have recently introduced dedicated very large scale integration (VLSI) hardware implementations. This brings additional speed and increases the scale of networks which can be implemented.
- *Learning and adaption.* Networks are trained using past data records from the system under study. A suitably trained network then has the ability to generalize when presented with inputs not appearing in the training data. Networks can also adapt on-line.
- *Data fusion.* Neural networks can operate simultaneously on both quantitative and qualitative data. In this respect networks stand somewhere in the middle ground between traditional engineering systems (quantitative data) and processing techniques from the artificial intelligence field (symbolic data).
- Multivariable systems. Neural networks naturally process many inputs and have many outputs; they are readily applicable to multivariable systems.

It is clear that a modeling paradigm which has all of the above features has a great promise. From the control point of view, the ability of neural networks to deal with nonlinear systems is perhaps the most significant. The great diversity of nonlinear systems is the primary reason why no systematic and generally applicable theory for nonlinear control design has yet evolved. However, a range of 'traditional' methods for the analysis and synthesis of nonlinear controllers for specific classes of nonlinear systems, exist. Such methods include: plane analysis methods, linearization techniques and techniques for describing functions.

3.11 Control system applications

There are typically two steps involved when using neural networks for control: system identification and control design. In the system identification stage, a neural network model of the plant that has to be controlled is developed. In the control design stage, the neural network plant model is used to design (or train) the controller (Martin T. Hagan et al, 1999). The nonlinear functional mapping properties of the neural networks are central to their use in control. In each of the control architectures the identification phase is similar.

A number of results have been published showing that a feedforward network of the multilayer perceptron type can approximate arbitrarily well a continuous function (Cybenko (1988, 1989); Carol and Dickson (1989); Funahashi (1989); Hornik et al. (1989). These papers prove that a continuous function can be arbitrarily well approximated by a feedforward network with only one hidden layer where each unit in the hidden layer has a continuous sigmoid nonlinearity. The use of nonlinearities other than sigmoid is also discussed. In summary, a large body of theoretical results relating to approximation using neural networks exists. These results provide theoretically important possibility theorems and deep insight into ultimate performance of networks and their application in the field of control. The section that follows provides an overview of papers based on neural networks and their successful applications.

3.12 Overview of existing solutions of ANN in Identification and Control applications

The following provides a sample of papers reviewed for modeling and control design applications of neural network theories.

Mohamed Azlan Hussain, (1998) provided an extensive review of the various applications utilizing neural networks for chemical process control, both in simulation and online implementation. The review was categorized under three major control schemes; predictive control, inverse-model-based control, and adaptive control methods, respectively. The review revealed the tremendous prospect of using neural networks in process control. It also showed that the multilayered neural network was the most popular network for such process control applications.

Cosme Rafael and Marcano-Gamero, (2004) proposed plant identification and control using a neural Controller based on a reference model. This work presented an application example of the neural network controller based on reference model, as had been treated by researchers such as Prof. Marios Polycarpou of the Southern California University, From the results reported by simulations, some conclusions were

(Chapman and Patry, 1989) for applying a real-time expert system shell to monitor and operate WWTP processes.

In **T.Y. Pai et al, (2007),** Grey model and artificial neural networks were employed to predict suspended solids, chemical oxygen demand and pH in the effluent from conventional activated process of an industrial wastewater treatment plant using simple online monitoring parameters. According to the results, the online monitoring parameters could be applied on the prediction of effluent quality.

D. Aguado et al; (2009) presented a case study on methodology for sequencing batch reactor (SBR) identification with artificial neural networks. The results successfully illustrated that the developed ANNs could be practical and cost-effective tools for monitoring SBR systems as both ANNs provided accurate and on-line predictions of Phosphorus concentration in the SBR.

K. A˘zman et al, (2007). In this paper the Gaussian process model is used for dynamic systems identification with emphasis on some of its properties: model predictions containing the measure of confidence, low number of parameters and facilitated structure determination. The GP model identification procedure with emphasis on the validation was illustrated with a simulated example and applied to two case studies. The wastewater treatment plant case study resulted in a purposeful model, while the algae growth model of the second case study showed that the lack of information content in training data prevented the development of a trustworthy Gaussian process model.

Sundarambal Palani et al, (2008) developed ANN models to predict salinity, water temperature, dissolved oxygen in Singapore's coastal water both temporally and spatially using continuous weekly measurements of water quality variables at different stations. In spite of largely unknown factors controlling seawater quality variation and the limited data set size, a relatively good correlation was observed between the measured and predicted values.

Willis M.J., et al, (1992) demonstrated that given the appropriate network topology, the neural network could be trained to characterize the behavior of a system. Additionally, the ability of ANN nonlinear controller to improve control when compared to the conventional "linear" techniques was demonstrated.

Emmanoulides, C. and Petrou, (1995) demonstrated the use of adaptive, on-line trained neural networks for identification and control of a complex, nonlinear bioprocess (an anaerobic digester). The developed controller exhibited desired tracking, regulation and robustness properties in such cases as set points or process variations and overcame the problems arising from the nonlinear nature of the process and the presence of measurement noise.

S. Boulabel et al, (2008) studied neural networks extensively for identification and prediction of wastewater process parameters. Three methods are implemented to drive the system's opefation. In the first case difference neural networks inputs are ignored but considered in the second case. An extended Kalman filter is used for the third case. High performance was derived from the approaches adopted which considered input difference effects and therefore fitted better to overcome the complex wastewater problems for parameter estimation.

A. Andrášik et al, (2003) developed a special control structure, incorporating the algorithm for on-line tuning of PID weight coefficients, where the convergence was guaranteed by proper selection of the learning rate. The performance of the proposed adaptive neural control system was assessed through the simulation of the controlled operation of a nonlinear continuous biochemical process. For prediction of process output, a hybrid model, consisting of a mechanistic mass balance complemented by Neural black-box pre-trained off-line for kinetic prediction, was utilized. The results of simulation experiments confirmed good regulatory and tracking performance of the augmented adaptive neural controller that was implemented.

3.13 Conclusion

This chapter provides a theoretical overview of artificial neural networks and their application to the field of control engineering. It starts with an introduction to the neural network, its biological inspiration and gives the historical background which outlines some perspectives as to the development of the field of neural networks and to the current direction as pertaining to the research.

Neural network model and different types of network architectures: The feedforward and the feedback (recurrent network) structures are discussed. Next, the different types of the networks activation functions such as the logarithmic sigmoid, hyperbolic tangent, pure linear, hard limit (thresh-hold), the radial basis functions and their signifance are investigated. Various learning methods such as supervised, unsupervised and reinforced learning are discussed. The most important class of

neural networks for real world problems covered in this literature review includes, Multi-layer Perceptron, Radial Basis Function Networks, and Kohonen Self organizing Feature Maps.

The algorithms for training, most of them can be viewed as a straightforward application of optimization theory and statistical estimation, are also discussed leading to the most frequently used, the backpropagation algorithm. It is very popular and consequently important learning rule applied to multi-layer feedforward neural networks. Its weaknesses and several methods of speeding up back-propagation algorithm for quick convergence are also proposed e.g. Conjugate gradient, Quasi-Newton, Levenberg-Marquardt methods. Methods to improve generalization model selection, Jittering, Early stoppage, Weight decay, Bayesian learning, and Regulization are also discussed. An overview of neural network identification and control is also provided.

Chapter 4 that follows examines neural network modeling methods for the DO process using different approaches and different structures that are used in this thesis.

CHAPTER FOUR

NN MODEL OF THE DO PROCESS (SYSTEM IDENTIFICATION)

This chapter applies the theory of neural network modelling to waste water treatment plants and in particular, the emphasis is on the development of the models for dissolved oxygen (DO) process using different approaches and different neural network structures for identification. The chapter gives an overview of the existing solutions using neural networks for the identification of the nonlinear process. Then nonlinear dynamics of the system is identified using the neural network based methods. The identified model will be used further as a part of predictive control algorithm and linearizable control for the DO plant that is used as a case study in this thesis.

4.1 Introduction

The DO concentration control in activated sludge system in WWTPs is important because it can give improved performance and provide economic incentive to minimize the oxygen consumption by supplying the necessary air to meet the time-varying biological oxygen demand in the system. The mathematical model of DO concentration process of the activated sludge process of WWTP was developed in chapter 2. For simulations of the DO concentration's characteristic, the ASM1 benchmark model parameters and state variable values listed in chapter 2 were used. The system was developed and simulated in the Matlab/Simulink environment and the time response as a function of airflow rate was plotted in Figure 2.7. The aim of this chapter is to use neural network theory for the identification of the DO process using the data obtained in chapter 2 and then to use the identified model in the following chapters to construct neural network controllers for the DO plant. Due to their impressive capability in dealing with severe nonlinearity and uncertainty in systems, the application of neural network method for the design of controllers is promising. Before the NN-based controller can be designed, an identification procedure of obtaining the neural network model to predict the plant output of the DO concentration has to be performed.

In this research thesis, two types of neural network model structures, the feedforward multilayer perceptron and the recurrent neural networks are studied for identification and control. The procedural steps are discussed in the next few sections.

4.2 NN in Identification and control

Experimental data are the only source of information used to build a neural network model. The first step is to generate some experimental input/output data from the

process to be modeled. In the case of the DO process this would be the input air flow rate and the output is the DO concentration.

An essential characteristic of the use of neural networks is the learning stage that precedes the application. During this stage, examples of the desired behaviour are applied to the network and with a learning algorithm; the parameters of the network are adjusted. Once the neural network is trained, then it can be applied for different tasks, such as process control (Miller et al., 1990; Hunt et al., 1992). For control systems applications, the inputs of the neural networks consist of the measurements of the process. A control action is then obtained as the neural network output.

4.3 The ANN model

Neural networks are composed of simple elements operating in parallel. These elements are inspired by the biological nervous systems as was explained in chapter 3. As in nature, the network function is determined largely by the connections between elements. A neural network can be trained to perform a particular function by adjusting the values of the connections (weights) between elements. For a detailed overview of ANN see chapter 3.

Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. Such a situation is shown in Figure 4.1 below. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are needed to train a network.

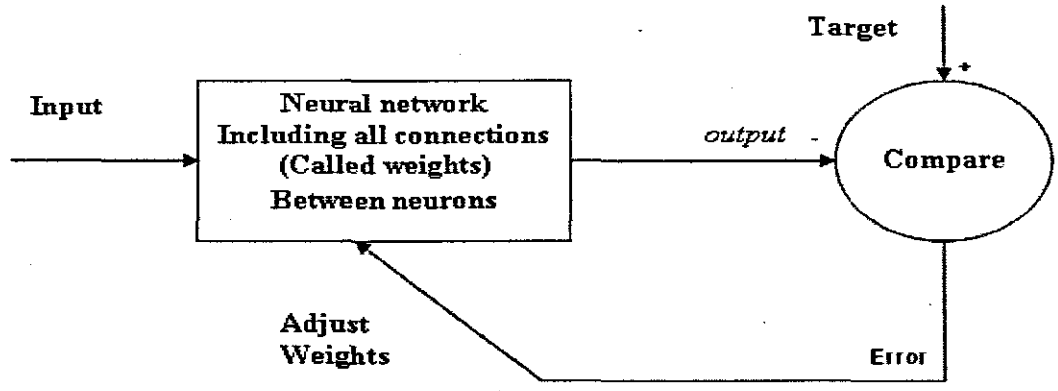


Figure 4.1: Training neural networks

Each neuron in a neural network usually combines the outputs from other neurons in the network or external inputs with values from previous computations (local memory) to produce inputs to other neurons in the network. How the inter-neuron connections are arranged and the nature of the connections determines the structure of a network.

4.4 Identifiability theory

Identification is used to model dynamic systems from experimental data. According to Norton (1986) in control engineering, model building from measurements of a dynamic system is known as *identification*. It is the process of constructing a mathematical model of a dynamic system from *observations* and *some prior knowledge* (Norton, 1986:1)

The first stage in the use of neural networks for control is to train the network to represent the forward dynamics of the plant to be controlled. The error between the plant output and the neural network output is used as the neural network training signal. This stage is known as the identification phase. An important question in system identification is that of the system identifiability (Ljung and Soderstrom, 1983; Ljung, 1987; and Soderstrom and Stoica, 1989). I.e. given a particular model structure, can the system under study be adequately represented within that structure? In the absence of such concrete theoretical results for neural networks it is assumed that all systems under study belong to the class of systems that the chosen network is able to represent within the structure. For nonlinear black box identification, it is necessary to be sure that the system inputs and outputs cover the operating range for which the controller will be applied.

4.4.1 Design of neural network Identifier models

The discrete-time mathematical model for the DO concentration process based on ASM1- model data and obtained through a zero-order-hold circuit with a sampling time of 1min 30sec is used to generate simulation data. The model equation is:

$$S_O(k+1) \approx S_O(k) + h^*[\frac{Q(k)}{V}[(S_{O_{in}}(k) - S_O(k)] + K_L a(u(k))[S_{O_{sat}}(k) - S_O(k)] + r_{so}(k)] \quad (4.1)$$

Where h^*, is the sampling period and t is is time. The Simulink model circuit discretized with the zero order hold together with tapped delays incorporated at the input and outputs to provide for previous values of the input airflow and the previous values of the output DO concentration process is illustrated in Figure 4.2 together with simulated response trajectories illustrated in Figure 4.3 and 4.4 respectively.

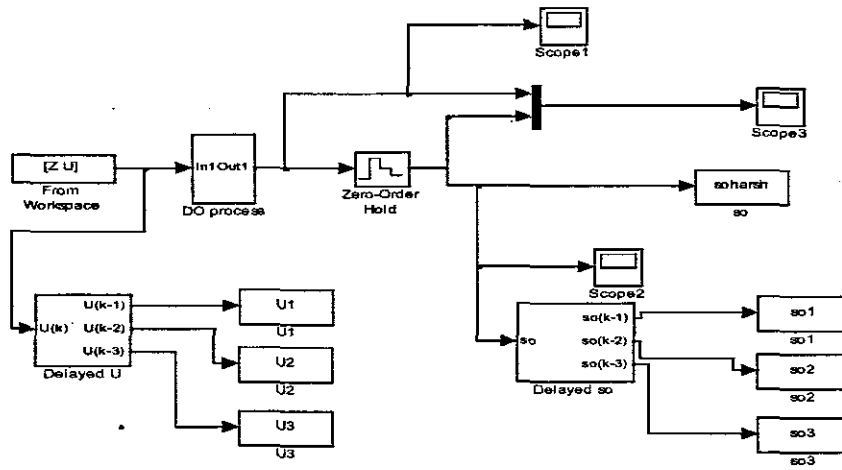


Figure 4.2: Simulink Discrete model of DO process

From which the simulated response trajectories of both the airflow rate $u(m^3/day)$ and the dissolved oxygen trajectories $S_O(mg/l)$ are illustrated in Figures 4.3 and 4.4.

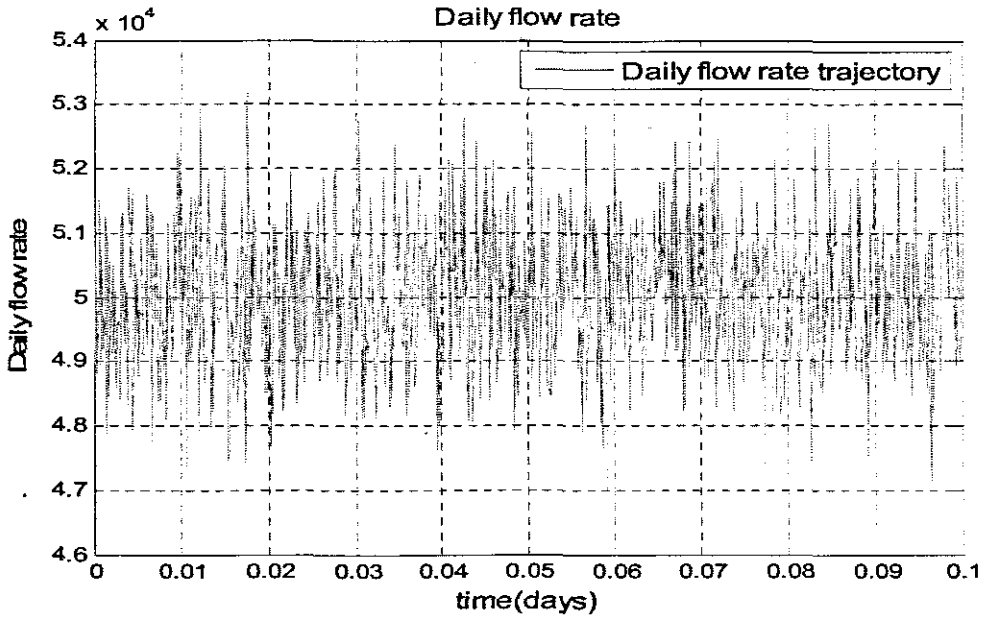


Figure 4.3: Airflow trajectory to the Simulink Discrete model of DO process

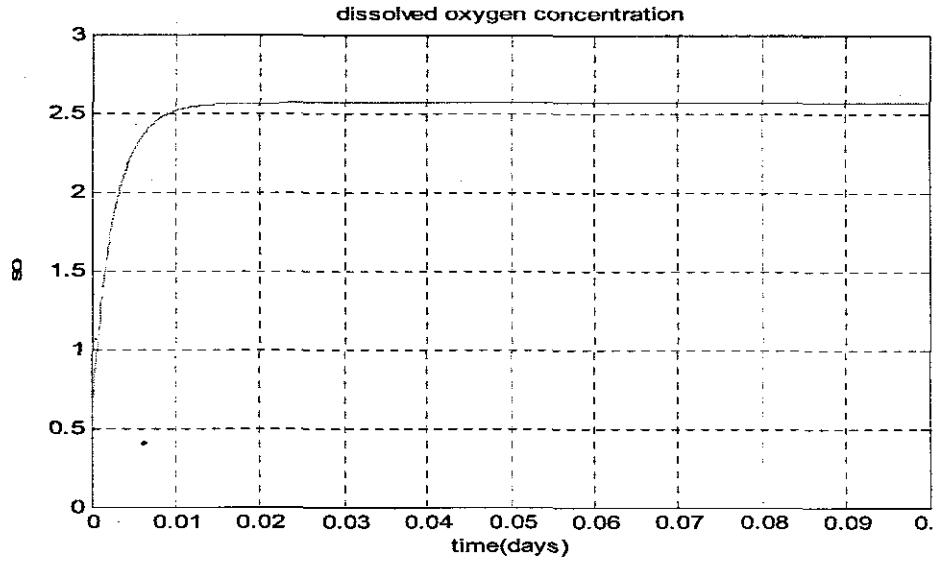


Figure 4.4: DO process trajectory from the Simulink Discrete model

The data resulting from the DO concentration simulation were sampled at 1min 30sec sampling interval, to make the ANN training less computationally demanding (hence 960 samples were used for training). These simulation data are used to design neural network models. Before the NN-based controllers can be applied, the procedure for obtaining the NN models, i.e., the forward and inverse model used in these strategies has to be developed.

The neural network for identification is designed as a three-layer network. It has an input layer, a hidden layer and an output layer. N_1^i, N_2^i and N_3^i, are their output values respectively. Superscript i denotes the identification while the subscripts $1,2$ and 3 stand for the input layer, hidden layer and output layer respectively. The numbers of the neuron in the hidden layer can be chosen depending on the practical training results at any given instant. The identifier models are trained to learn the forward dynamics of the plant (DO concentration). Twelve inputs and one output are selected to build the identifier model for the system. Initially one input and one output model is tested and successively more inputs are added. The twelve inputs are: $i1\_1(t-k), k = 0...2, \mathrm{z(t-m)}, \mathrm{m} = 0...2,$ and $i2\_2(t-n), n = 0...2.$ where $i1\_1$ is the normalized airflow rate, z is the time trajectory and $i2\_2$ is the normalized output of the DO process. These signals provide the present and previous input air flows and also incorporate the historical trend from the present and last time steps, and finally the present and three delayed values of the DO concentration are also used. A set of corresponding input and output training patterns is selected from the open-loop continuous signal response of the system.

4.4.2 Forward modeling

The procedure of training a neural network to represent the forward dynamics of a system is referred as forward modeling, and is illustrated in Figure 4.5 where y^p is the plant output, y^m is the neural output d and d' are various plant disturbances.

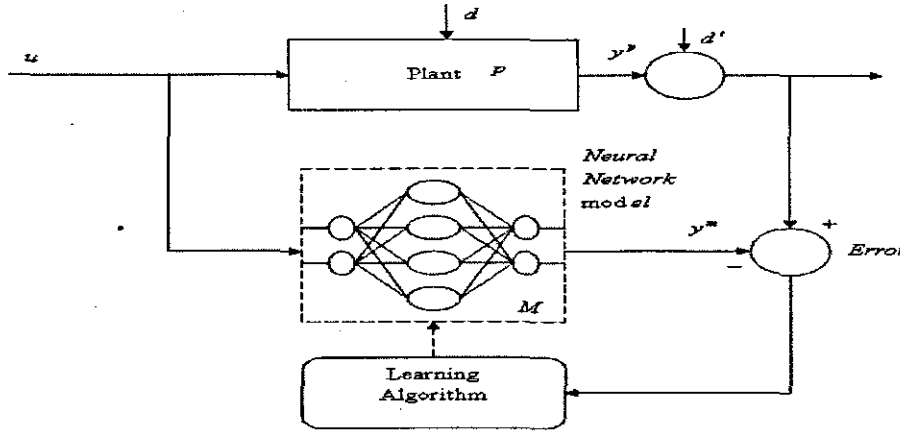


Figure 4.5: Plant Identification procedure.

The neural network model is placed in parallel with the system and the error between the system and the network outputs (the prediction error) is used as the network training signal. As pointed out (Jordan and Rumelhart, 1991) this learning structure is a classical supervised learning problem where the teacher (i.e. the system) provides target values (i.e. the outputs) directly in the output coordinate system of the learner (i.e. the network model). In the particular case of a multilayer perceptron type neural networks straightforward backpropagation of the prediction error through the network would provide a possible training algorithm.

In training a neural network to learn the forward dynamic model of the plant the backpropagation error signal between the plant output and the output layer is expressed as:

$$\delta_3 = y^p(k) - y^m(k) \tag{4.2}$$

Where, $y^p(k)$ is the target pattern and $y^m(k)$ is the actual output of the identifier, and the backpropagation error signal between the hidden and input layers is expressed as

$$\delta_2 = f'(net_2)\sum_3 \delta_3 . w_{32} \tag{4.3}$$

Where, $f'(net_2)$ is the derivative of the activation function $f(net_2)$, the one used is the sigmoid activation function as explained in Chapter 3 where,

$$f(net_2) = \frac{1}{1+\exp(-net_2)} \tag{4.4}$$

Subscripts 1,2 and 3 stand for the input layer, hidden layer and output layer respectively. The weights between the input and hidden layers are updated as,

$$\Delta w_{21i}(k+1) = \eta \cdot \delta_2 \cdot N_1^i + \alpha \cdot \Delta w_{21}(k) \tag{4.5}$$

And the weights between the hidden and output layers are updated as,

$$\Delta w_{32}(k+1) = \eta \cdot \delta_3 \cdot N_2^i + \alpha \cdot \Delta w_{32}(k) \tag{4.6}$$

Where, N_1^i and N_2^i are outputs of the input and hidden layers respectively, η is the learning rate, α is the momentum coefficients. These constants are all chosen empirically.

The structure of a neural network plant model is given in Figure 4.6, where the blocks labelled TDL are tapped delay lines that store the previous values of the input signal. The forward modelling in this case refers to training the plant output of DO concentration at the next instant of time $(k+1)$.

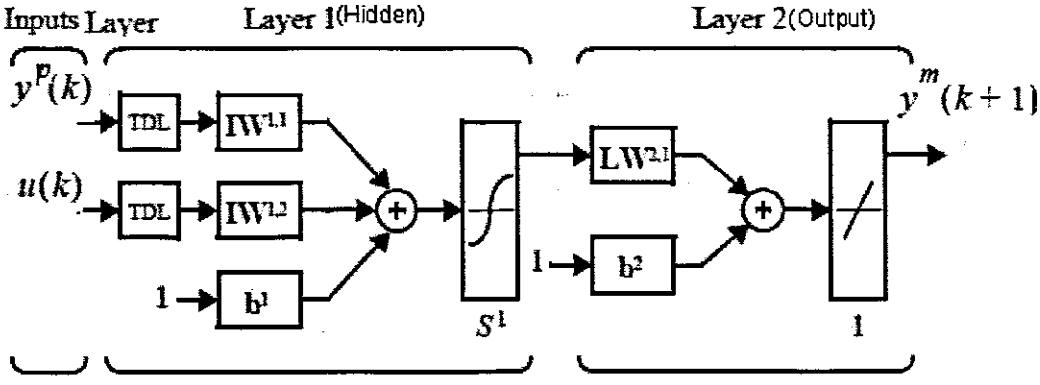


Figure 4.6: Neural Network Plant model (NN Toolbox-math works, 2002).

$IW^{i,j}$ is a weight matrix from input number j to layer number i. $LW^{i,j}$ is a weight matrix from layer number j to layer number i. It is important that the training data covers the entire range of plant operation, because it is known from previous discussion that nonlinear neural networks do not extrapolate accurately. The input to this network is an $B = (n^y + n^u + n^k)$ - dimensional vector of the previous plant outputs y and inputs u which have to be determined. It is this space that must be covered adequately by the training data, where n^y is the dimension of the vector of the plant output, n^u is the dimension of the vector of the plant input and n^k is the dimension of the vector of time. .

4.4.3 Multi-layer feed forward neural networks design

Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear

transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors. The training of the network is done in the Matlab software environment. Thus, all the neural network algorithms presented in this thesis are implemented using the neural network tool box for use with Matlab. Using neural network toolbox and Matlab environment, it is possible to develop multi-layer perceptron neural networks for identification. The first step in training a feedforward network is to create the network object. The Matlab function *"newff"* creates a *feedforward network. It requires four types of inputs and returns the network object.* The first input is a P by 2 matrixes of minimum and maximum values for each of the P elements of the Input vector. The second input is an array containing the sizes of each layer. The third input is a cell array containing the names of the transfer functions to be used in each layer. The final input contains the name of the training function to be used. *Before training a feedforward network, the weights and biases must be* initialized. The *"newff"* command automatically initializes the network.

Once the network weights and biases have been initialized, the network is ready for training. The network can be trained for function approximation (nonlinear regression), pattern association, or pattern classification. The training process requires a set of examples of proper network behavior - network inputs (P) and target outputs (T). During training the weights and biases of the network are iteratively adjusted to minimize the network performance function *"net.performFcn"*. The default performance function for feedforward networks is mean square error *"mse"* - the average squared error between the network outputs and the target outputs. It is very difficult to know which training algorithm will be the fastest for a given problem. It will depend on many factors, including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the network, the error goal, and whether the network is being used for pattern recognition or function approximation (regression).

Neural network training can be made more efficient if certain pre-processing steps are performed on the network inputs and targets. Before training, it is often useful to scale the inputs and targets so that they always fall within a specified range. This is because neural networks can not cope with a big range of data values. If scaled data is presented to the network; the weights can remain in small but predictable ranges. To produce the most efficient training, it is often helpful to preprocess the data before training. It is also helpful to analyze the network response after training. One method of prepossessing the data is normalization. The original network inputs and targets with the given matrices are normalized such that the normalized inputs and targets, which are returned, will all fall in the interval **[0, 1]**. The new vector components that are received are the fraction of the maximum value of the original inputs, and the fraction

of the maximum value of the original targets. The normalization method used in this work is:

$$Normalized\ value(\text{i1\_1}) = \frac{actual\ \text{value(i1.)}}{\max imum\ value\ of\ the\ \text{var}iable(\text{i1})} \tag{4.7}$$

The simulation data given in section (4.4.1) resulting from the Benchmark process with the biological model ASM1 and which describes the trajectory of $u(t)$ and the trajectory of $S_O(t)$, are pre-processed by normalization to reduce their sizes for neural network training. This means that the input air flow rate of $\text{u}(4.7 \bullet \text{E}+004 \text{ to } 5.3 \bullet \text{E}+004)\text{m}^3/day$ is normalized in the range i1_1 [0, 1]. Similarly the dissolved oxygen concentration of values 0 to 2.5624mg/l is normalized by Equation 4.7 to range of i2_2 [0, 1]. The resulting normalized graphs are illustrated in Figures 4.7(a) and (b) respectively.

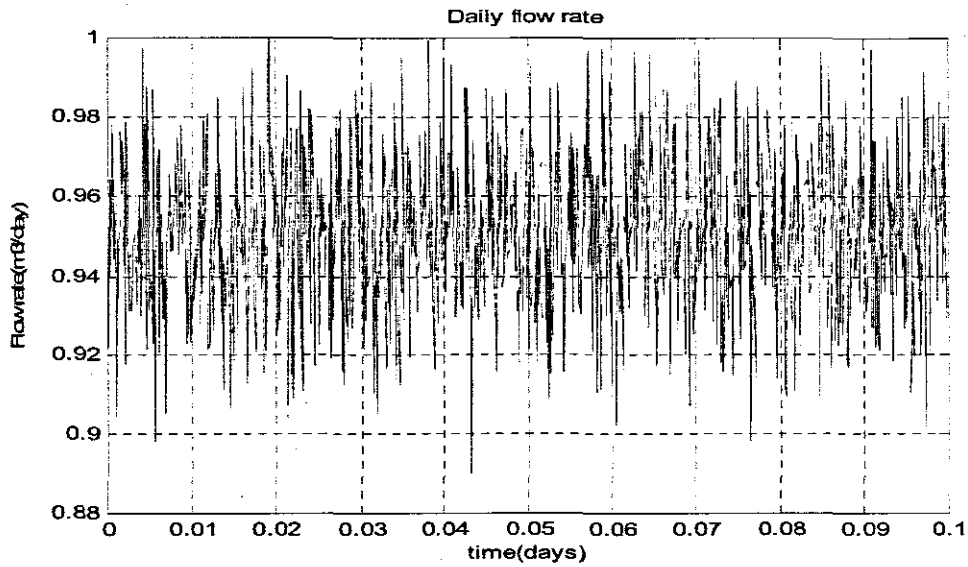


Figure 4.7(a): Graph showing the Normalized Airflow rate

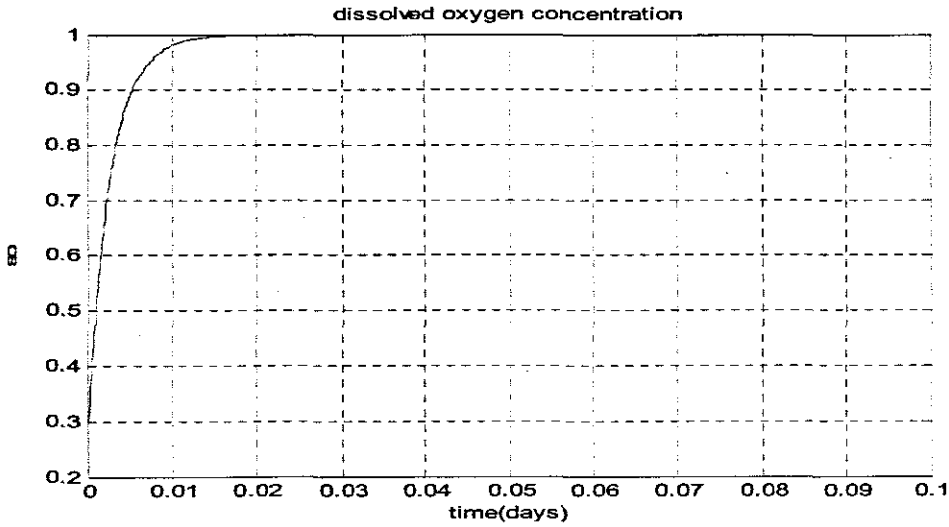


Figure4.7 (b): Graph showing the Normalized time response of DO concentration

The next step is to divide the data up into training, validation and test subsets. Half of the data (480 points) was used as training set, one fourth of the data (240 points) was used for the validation set and another one fourth (240 points) was used as the test set. These sets were picked as equally spaced points throughout the original data. Figures 4.8(a), (b) and (c) shows the division of training, validation and test data points used. After the network has been trained with the normalized data, the vectors so formed are used to transform any future inputs that are applied to the network since they effectively become a part of the network, just like the network weights and biases. The data subsets for training, validation and test for different cases of input matrices to the networks were designed using different number of plant inputs, outputs and time domain.

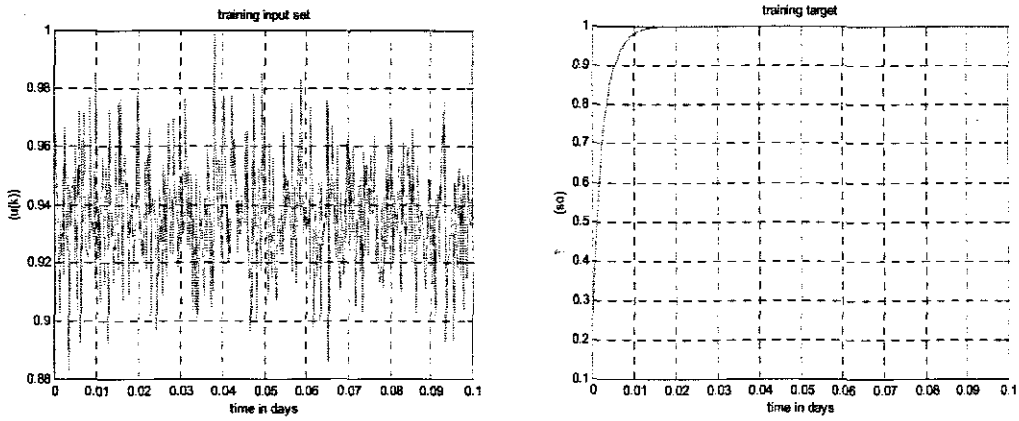


Figure 4.8(a): Graph showing the training data set for Normalized Airflow rate and DO concentration

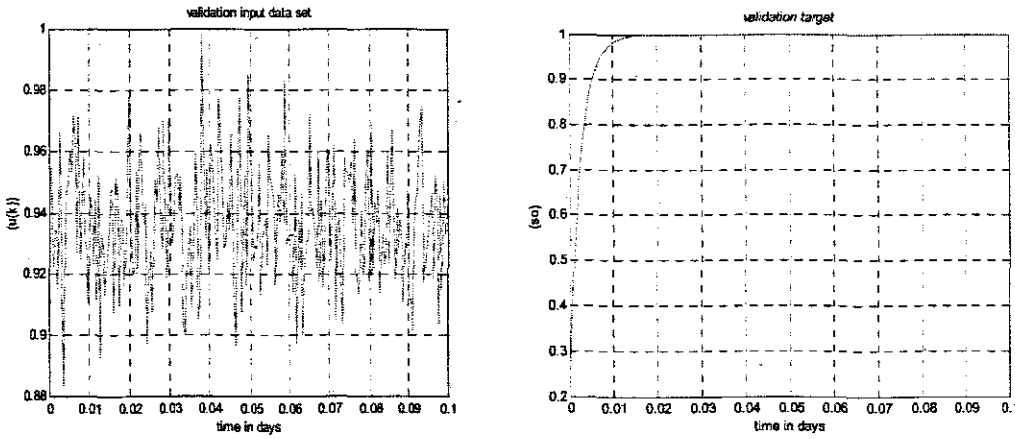


Figure 4.8(b): Graph showing the validation data set for Normalized Airflow rate and DO concentration

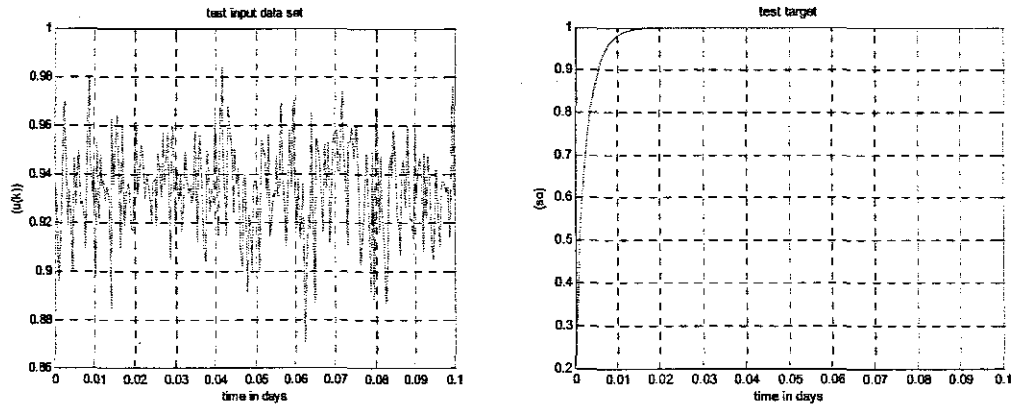


Figure 4.8(c): Graph of the test data set for Normalized Airflow rate and DO concentration

4.4.4 Construction of Multi-layer feed forward neural networks

The architecture of a multilayer network is not completely constrained by the problem to be solved. The number of inputs to the network is constrained by the problem, and the number of neurons in the output layer is constrained by the number of outputs required by the problem. However, the number of layers between network inputs and the output layer and the sizes of the layers are up to the designer. However, the two-layer sigmoid/linear network can represent any functional relationship between inputs and outputs if the sigmoid layer has enough neurons as explained in chapter 3. There are several different backpropagation training algorithms. They have a variety of different computation and storage requirements, and no one algorithm is best suited to all locations.

The different Multilayer perceptron (MLP) ANNs structures implemented in this work are as illustrated in table 4.1. The used ANN structures are as a result of systematic identification process which considered the influence of the number of neurons, the number of hidden layers, the type of training algorithms used, the number of lagged input variables used, etc. The first network consists of a single input vector of the normalized air flow rate and a target vector of the normalized Dissolved oxygen concentration. The second ANN structure implemented has two input vectors of air flow rate and time while the target is Dissolved oxygen concentration. The third ANN implemented consists of three input vectors (airflow rate, time and Dissolved oxygen concentration while the target output is the Dissolved oxygen concentration. The fourth one consists of airflow rate, three delays of airflow rate, time, three delays of time function, dissolved oxygen concentration, three delays of dissolved oxygen concentration as input vector while the target is the dissolved oxygen concentration, etc. One of the topology of the Feedforward neural MLP network as used in this thesis excluding biases is shown in Figure 4.9, where, ko is the first delay value of il_1, kl is

the second delay value of $i1\_1$, and $k2$ is the third delay value of $i1\_1$. z is the time value of no is the first delay value of $i2\_2$, $n1$ is the second delay value of $i2\_2$ and $n2$ is the third delay value of $i2\_2$.

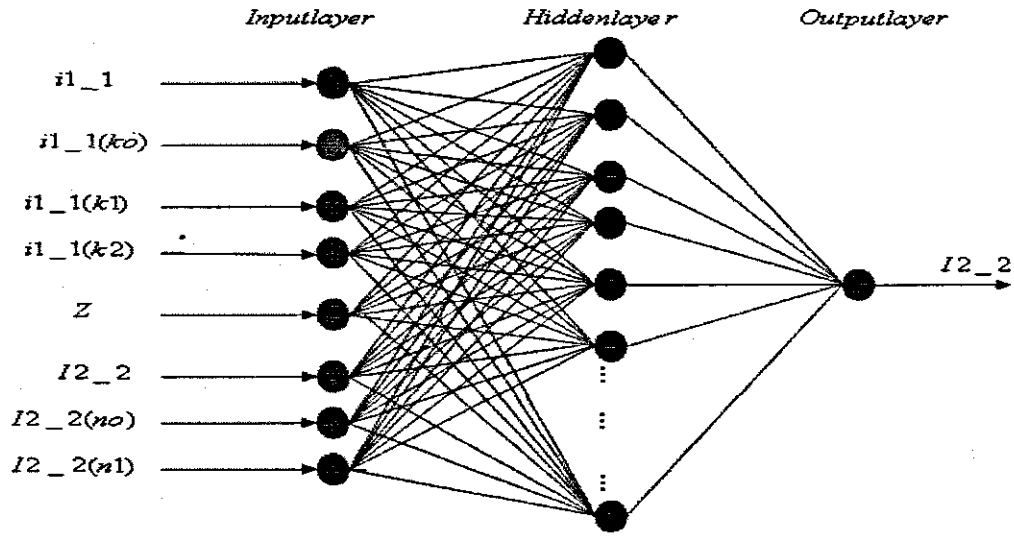


Figure 4.9: Topology of the MLP Feedforward neural network

A summary of the structures implemented are tabulated in the in table 4.1 for MLP feedforward networks as implanted and tested using the three different training algorithms.

Table4.1: Input combinations to the NN with DO as the target.

Number of input vectors	Model	Input description	Input symbol	output
1	Model 1	Normalized Airflow rate vector	I1_1	DO
2	Model 2	Normalized Airflow rate and Time vector	I1_1, Z	DO
3	Model 3	Normalized Airflow rate, Time and Normalized DO vector	I1_1, Z and i2_2	DO
12	Model 4	Normalized Airflow rate, 3 delayed normalized airflows, Normalized DO, 3 delayed Normalized DO, Time, and 3 delayed Time vectors	I1_1, I1_1(ko), I1_1(k1), i1_1(k2), Z, Z(mo), Z(m1), Z(m2), and i2_2, I2_2 (no), I2_2 (n1), I2_2 (n2),	DO

4.5 Training the feedforward networks

For the feedforward MLP, three different types of backpropagation training algorithms are used to compare their effectiveness in identifying the DO concentration process model. These are:

- The Scaled conjugate gradient algorithm *("trainscg")*. This is because it is the only conjugate gradient algorithm that requires no line search and a very good general purpose training algorithm.

- Levenberg-Marquardt algorithm *("trainlm")*. This is because it is the fastest training algorithm for networks of moderate size. Has memory reduction feature for use when the training set is large.
- Resilient backpropagation *("trainrp")*. Simple batch mode training algorithm with fast convergence and minimal storage requirements.

Networks can be trained incrementally or in batch. In incremental training the weights and biases of the network are updated each time an input is presented to the network. In batch training the weights and biases are updated after all the inputs are presented. The type of training used here is batch training. This is because it typically has access to more efficient training algorithms e.g. Levenberg-Marquardt. In addition batch training can be applied to both static and dynamic networks. For this case, the input vectors can either be placed in a matrix of concurrent vectors or in a cell array of sequential vectors which is converted to a matrix of concurrent vectors. The function 'train' always operates in batch mode. Batching concurrent inputs is computationally more efficient training and the matrix notation used in MATLAB makes batching simple.

```
net=newff(minmax(P),[10 1],{'tansig','purelin'},'trainscg');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.show=25;
net.trainParam.epochs=10000;
net.trainParam.goal=1e-8;
net1=train(net,P,T);
g1=input('Stike any key....');
a=sim(net1,P);
```

where **P** is the range of network input vector and **T** is the range of network target output vector. The above Matlab code creates a feedforward network. *Minmax (P)* represents the range from minimum to maximum of the input vector (P).

The function *'newff'* allows the user to specify the number of layers, the number of neurons in the hidden layers and the activation functions used. For the example of the code given above, the network contains 10 neurons in its hidden layer. The hidden layer contains *'tansig'* activation functions and the output layer contains a pure linear function. The type of training used is also set here, and for the example above it is the scaled conjugate gradient descent training algorithm.

The show parameter *(net.trainParam.show)* allows the number of epochs between feedbacks during training to be set. The result is shown after every 25thiterations (epoch). It gives the training status information.

The sections that follow will consider different models developed and trained with the three different algorithms selected, together with the results achieved and the various conclusions derived from the considered cases of Table 4.1.

4.6.1 Feedforward identification with one input and trainrp algorithm–model1

Table 4.2 shows the results of the feed forward multilayer perceptron neural networks trained with *"trainrp"* algorithm with different number of hidden neurons, training epochs, learning rates and the mean squared error MSE achieved for one single input variable: the normalized input airflow rate (i1_1) as a function of time. The target is the normalized DO concentration (i2_2), where MSE stands for the mean squared error between the desired target and the network output.

Table4.2: Feedforward identification with one input and *trainrp* algorithm model1

Type of ANN(Network structure)	Activation functions& training algorithm*(trainrp)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	10000	0.05	0.00721334
NEWFF	Tansig, Purelin	3	45	0.05	0.00721499
NEWFF	Tansig, Purelin	4	321	0.05	0.00721186
NEWFF	Tansig, Purelin	5	10000	0.05	0.00721094
NEWFF	Tansig, Purelin	10	3914	0.05	0.00711717
NEWFF	Tansig, Purelin	15	10000	0.05	0.00700723
NEWFF	Tansig, Purelin	20	10000	0.05	0.00701271

The trajectories of the calculated outputs of the NN in comparison with the target outputs and the trajectories of the errors between them are shown in the Figures 4.10 to 4.13.

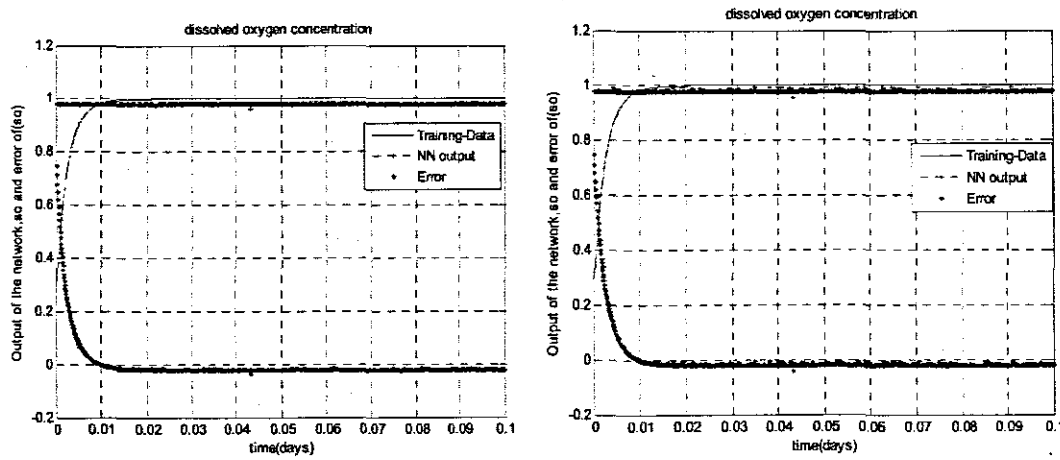


Figure4.10: Feedforward networks response with 1 input, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 1) respectively.

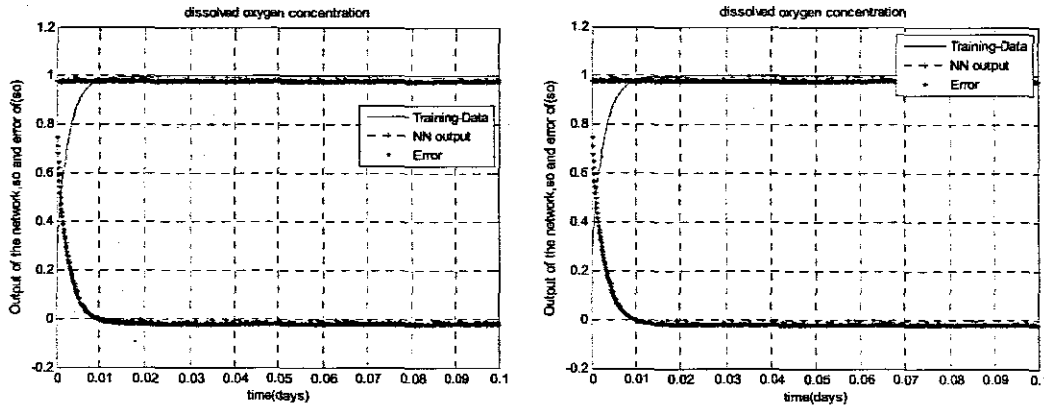


Figure4.11: Feedforward network response with 1 input, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 1) respectively.

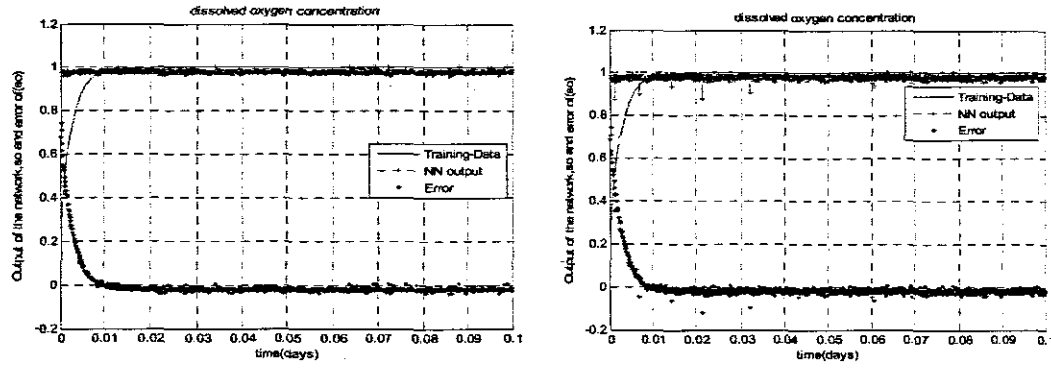


Figure 4.12: Feedforward networks response with 1 input, 1 hidden layer, 15 hidden neurons and 1 output (model 1).

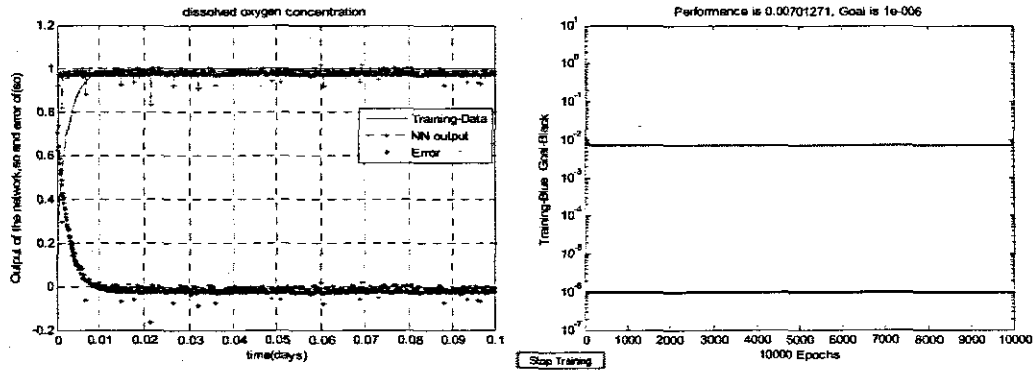


Figure 4.13: Feedforward networks response with 1 input, 1 hidden layer, 20 hidden neurons and 1 output (model 1) and the Performance MSE achieved.

Figure 4.13 shows the MSE achieved with the Feedforward network having 20 neurons in the hidden layer. The number of neurons was incrementally added from 2 to 20 as depicted in Table 4.2. Analysis of the graphs plotted for the training data, NN output and the error as a function of time illustrated in Figures 4.10 - 4.13, and in addition analysis of Table 4.2 for the performance index (MSE) have shown that:

- With only one input vector to the Feedforward network, the network does not map the required target when trained with the *trainrp* algorithm irrespective of the number of neurons in the hidden layer.
- The lowest MSE of 0.00700723 is obtainable after several numbers of training epochs. This value confirms that the function is not well approximated.
- The graphs show that as the number of neurons is increased, the response is no better. Therefore more than 3 neurons in the hidden layer are unnecessary for only one input vector range.

4.6.2 Feedforward identification with one input and *trainlm* algorithm-model1

Table 4.3 shows the results of the feed forward multilayer perceptron neural networks trained with *trainlm* algorithm with different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for one single input variable: the normalized input airflow rate (i1_1) as a function of time. The target is the normalized DO concentration (i2_2).

Table4.3: Feedforward identification with one input and *trainlm* algorithm-model1

Type of ANN(Network structure)	Activation functions& training algorithm(*trainlm*)	No of hidden nodes	Training Epochs	Learning rate	MSE
NEWFF	Tansig, Purelin	2	10000	0.05	0.00717659
NEWFF	Tansig, Purelin	3	10000	0.05	0.00698306
NEWFF	Tansig, Purelin	4	10000	0.05	0.00706282
NEWFF	Tansig, Purelin	5	10000	0.05	0.00690652
NEWFF	Tansig, Purelin	10	10000	0.05	0.00687287
NEWFF	Tansig, Purelin	15	10000	0.05	0.00675698
NEWFF	Tansig, Purelin	20	10000	0.05	0.00666949

The trajectories of the calculated outputs of the NN in comparison with the target outputs and the trajectories of the errors between them for the model trained with *trainlm* algorithm are shown in the Figures 4.14 to 4.17.

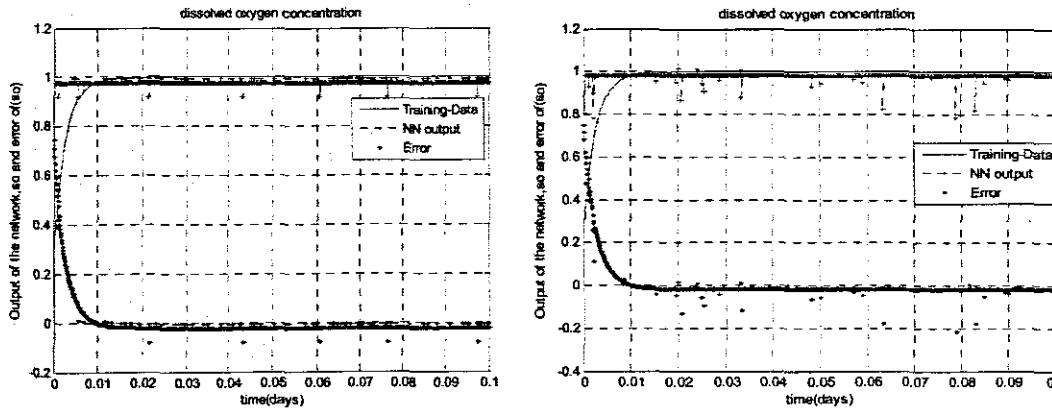


Figure4.14: Feedforward networks response with 1 input, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 1) respectively.

- With only one input vector to the Feedforward network, the network does not map the required target when trained with the *trainrp* algorithm irrespective of the *number of neurons in the hidden layer.*
- The lowest MSE of 0.00700723 is obtainable after several numbers of training epochs. This value confirms that the function is not well approximated.
- The graphs show that as the number of neurons is increased, the response is no better. Therefore more than 3 neurons in the hidden layer are unnecessary for only *one input vector range.*

4.6.2 Feedforward identification with one input and *trainlm* algorithm-model1

Table 4.3 shows the results of the feed forward multilayer perceptron neural networks trained with *trainlm* algorithm with different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for one single input variable: the normalized input airflow rate **(i1_1)** as a function of time. The target is the normalized DO concentration (i2_2).

Table4.3: Feedforward identification with one input and *trainlm* algorithm-model1

Type of ANN(Network structure)	Activation functions& training algorithm*(trainlm)*	No of hidden nodes	Training Epochs	Learning rate	MSE
NEWFF	Tansig, Pure_in	2	10000	0.05	0.00717659
NEWFF	Tansig, Pure_in	3	10000	0.05	0.00698306
NEWFF	Tansig, Pure_in	4	10000	0.05	0.00708282
NEWFF	Tansig, Pure_in	5	10000	0.05	0.00690652
NEWFF	Tansig, Pure_in	10	10000	0.05	0.00687287
NEWFF	Tansig, Pure_in	15	10000	0.05	0.00675698
NEWFF	Tansig, Pure_in	20	10000	0.05	0.00666949

The trajectories of the calculated outputs of the NN in comparison with the target outputs and the trajectories of the errors between them for the model trained with *trainlm* algorithm are shown in the Figures 4.14 to 4.17.

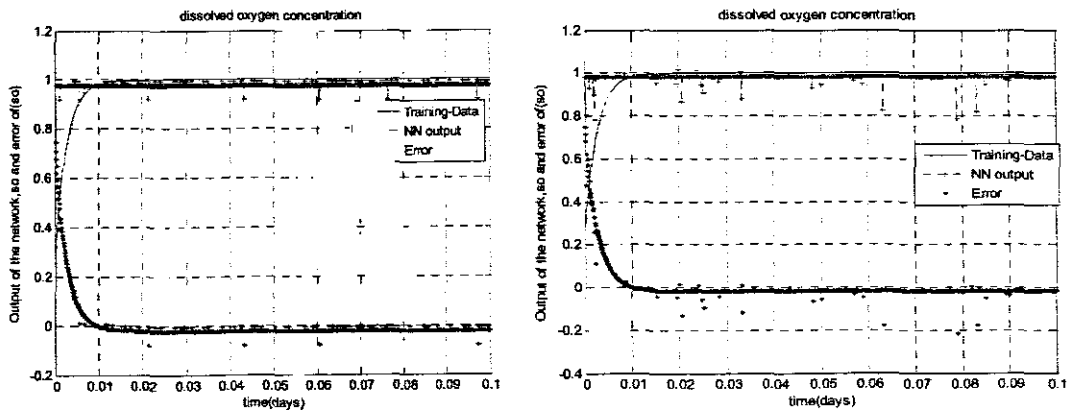


Figure4.14: Feedforward networks response with **1** input, **1** hidden layer, **2 and 3** hidden neurons and **1** output (model **1**) respectively.

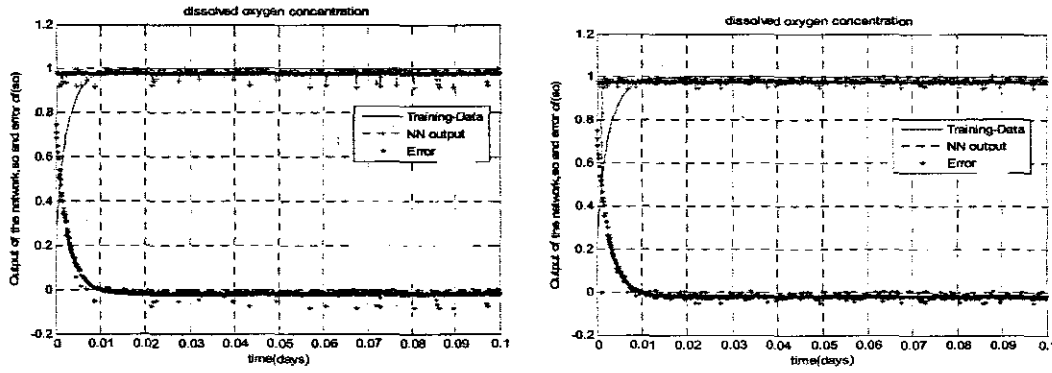


Figure 4.15: Feedforward networks response with 1 input, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 1) respectively.

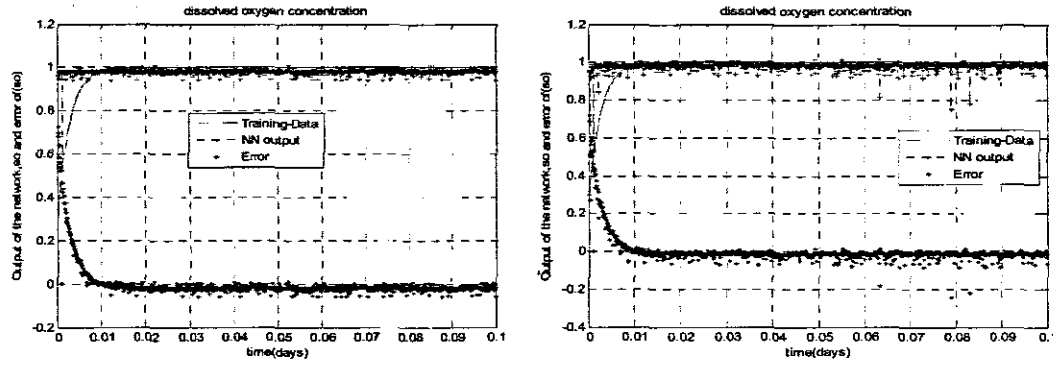


Figure 4.16: Feedforward networks response with 1 input, 1 hidden layer, 10 and15 hidden neurons and 1 output (model 1) respectively.

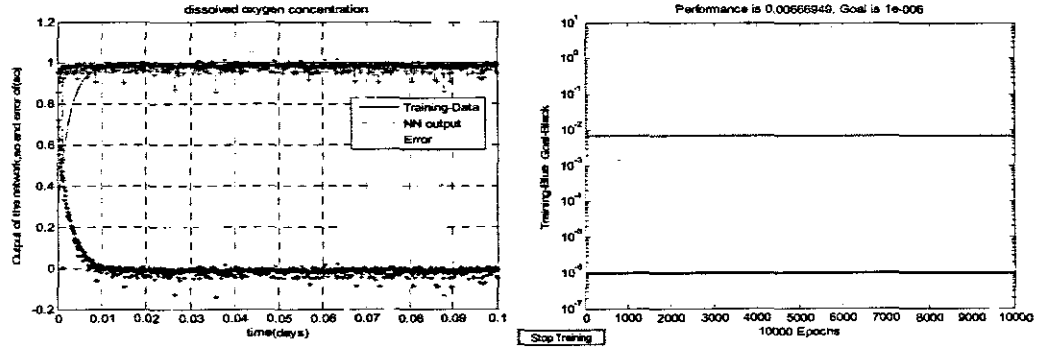


Figure 4.17: Feedforward networks response with 1 input, 1 hidden layer, 20 hidden neurons and 1 output (model 1) and Performance MSE achieved.

Figure 4.17 shows the MSE achieved with the Feedforward network having 20 neurons in the hidden layer. The number of neurons was incrementally added from 2 to 20 as depicted in Table 4.3. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.14 - 4.17, and in addition analysis of Table 4.3 for the performance index (MSE) have shown that:

- With only one input vector to the Feedforward network, the network does not map the required target when trained with the *trainlm* algorithm irrespective of the number of neurons in the hidden layer

- The lowest MSE of 0.00666949 is obtainable after several numbers of training epochs. This value confirms that the function is not well approximated.
- However the rate of convergence for the network when trained with *trainlm* is much faster than *trainrp* algorithm.
- Analysis between the graphs shows that as the number of neurons is increased, the response is no better. Therefore more than 3 neurons in the hidden layer are unnecessary for only one input vector range.

4.6.3 Feedforward identification with one input and *trainscg* algorithm- model1

Table 4.4 shows the results of the feed forward multilayer perceptron neural networks trained with *trainscg* algorithm with different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for one single input variable: the normalized input airflow rate (i1_1) as a function of time. The target is the normalized DO concentration (i2_2). These combinations constitute model 1.

Table4.4: Feedforward identification with one input and *trainscg* algorithm-model1

Type of ANN(Network structure)	Activation functions& training algorithm*(trainscg)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	31	0.05	0.00721295
NEWFF	Tansig, Purelin	3	286	0.05	0.00720517
NEWFF	Tansig, Purelin	4	234	0.05	0.00720332
NEWFF	Tansig, Purelin	5	667	0.05	0.00719574
NEWFF	Tansig, Purelin	10	732	0.05	0.00703853
NEWFF	Tansig, Purelin	15	5063	0.05	0.00695917
NEWFF	Tansig, Purelin	20	10000	0.05	0.00688632

The trajectories of the calculated outputs of the NN in comparison with the target outputs and the trajectories of the errors between them for the model trained with *trainscg* algorithm are shown in the Figures 4.18 to 4.33.

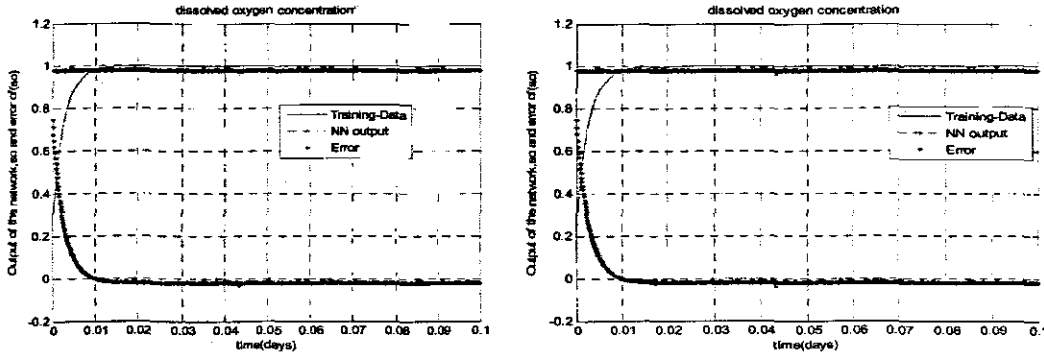


Figure 4.18: Feedforward networks response with 1 input, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 1).

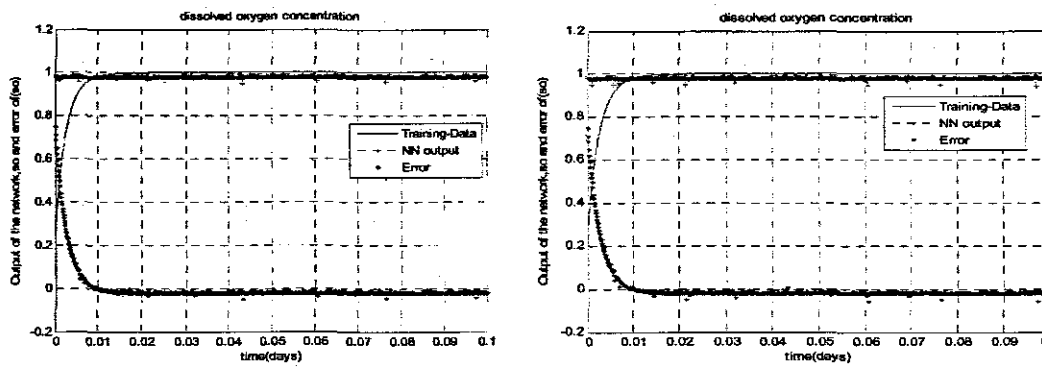


Figure 4.19: Feedforward networks response with 1 input, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 1).

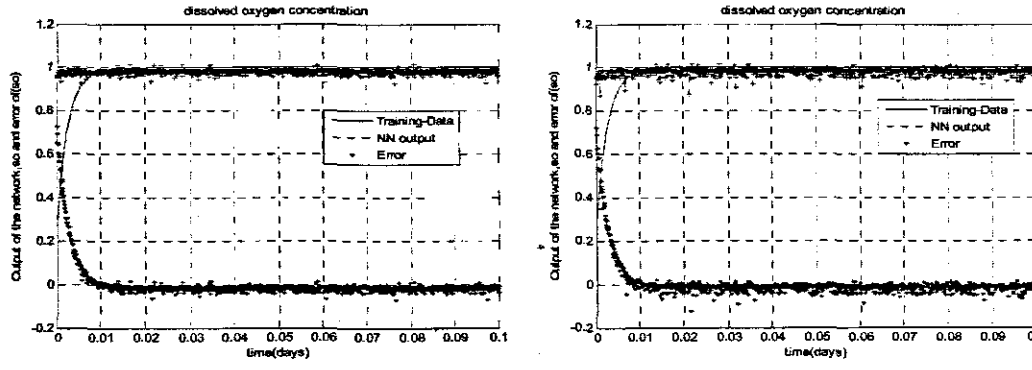


Figure 4.20: Feedforward networks response with 1 input, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 1).

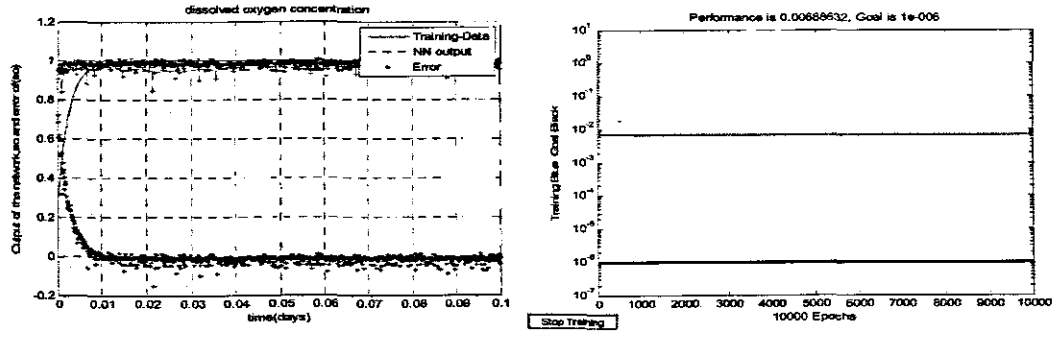


Figure 4.21: Feedforward networks response with 1 input, 1 hidden layer, 20 hidden neurons and 1 output (model 1) and Performance MSE achieved.

Figure 4.21 shows the MSE achieved with the Feedforward network having 20 neurons in the hidden layer. The number of neurons was incrementally added from 2 to 20 as depicted in Table 4.4. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.18 - 4.21, and analysis to Table 4.4 for the performance index (MSE) have shown that:

- With only one input vector to the Feedforward network, the network does not map the required target when trained with the *trainscg* algorithm irrespective of the number of neurons in the hidden layer

- The lowest MSE of 0.00688632 is obtainable after several numbers of training epochs when the network has 20 neurons in the hidden layer.
- *If there are very few neurons in the hidden layer then the convergence speed is* very fast. This can be seen from the data of Table 4.4.
- The value of MSE achieved confirms that the model does not approximate the target when trained with *trainscg* algorithm. However the rate of convergence for the network when trained with *trainscg* is much faster than with *trainrp* algorithm but *a bit slower than the network trained with trainlm algorithm.*
- Analysis between the graphs shows that as the number of neurons is increased, the response is no better. Therefore more than 3 neurons in the hidden layer are unnecessary for only one input vector range.

In conclusion, from the experimental work done on model1 it is found that irrespective of the number of neurons in the hidden layer the different types of algorithms used for training, the network does not map the required target if the input vector has only one input variable. The trainlm algorithm is the fastest and requires very few numbers of iterations for convergence than the other two. The next network tested is with two input vectors consisting of the variable of the normalized input flow rate trajectory and the variable of the time function. The procedure adopted was the same as that of one input vector. The results are explained in the proceeding sections.

4.6.4 Feedforward identification with 2 input vectors and *trainrp* algorithm-model2

Table 4.5 shows the feed forward multilayer perceptron neural networks trained with *trainrp* algorithm, different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for a vector of two input variables: the normalized input airflow rate **(i1_1)** and time function **(Z)**. These combinations constitute model 2. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.22 to 4.25.

Table4.5: Feedforward identification with two inputs and *trainrp* algorithm-model2

Type of ANN(Network structure)	*Activation functions&* training algorithm*(trainrp)*	*No of* hidden nodes	*Training* Epochs	*Learning* rates	*MSE*
NEWFF	Tansig, Purelin	2	10000	0.05	0.000268546
NEWFF	Tansig, Purelin	3	10000	0.05	0.000235694
NEWFF	Tansig, Purelin	4	10000	0.05	0.000226955
NEWFF	Tansig, Purelin	5	10000	0.05	2.94758e-005
NEWFF	Tansig, Purelin	10	10000	0.05	7.68119e-006
NEWFF	Tansig, Purelin	15	10000	0.05	4.70537e-005
NEWFF	Tansig, Purelin	20	10000	0.05	0.000357682

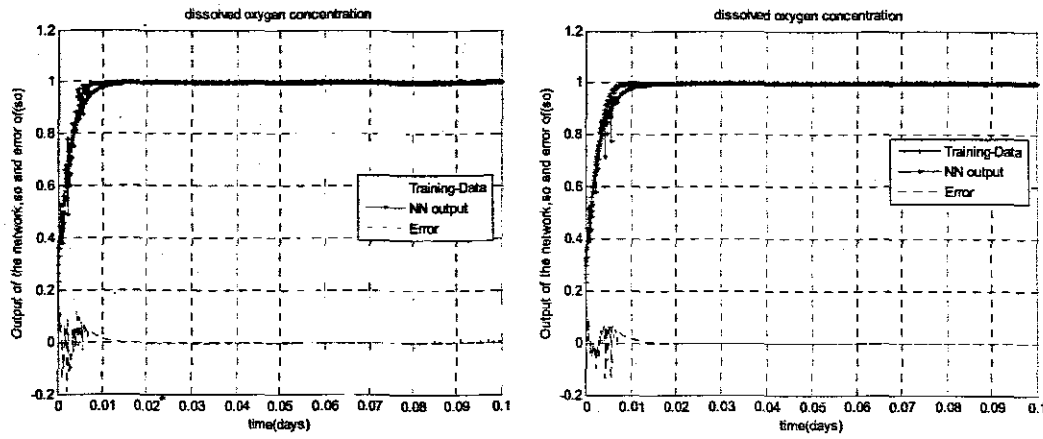


Figure 4.22: Feedforward networks response with 2 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 2).

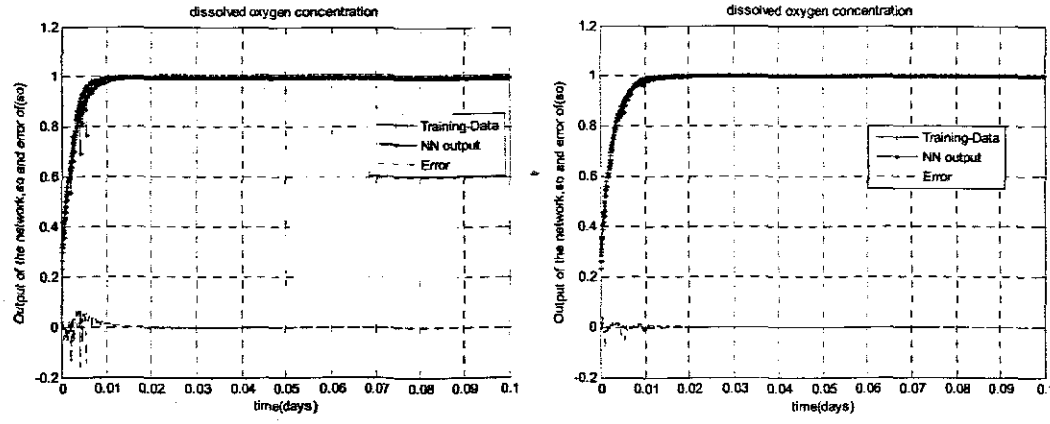


Figure 4.23: Feedforward networks response with 2 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 2).

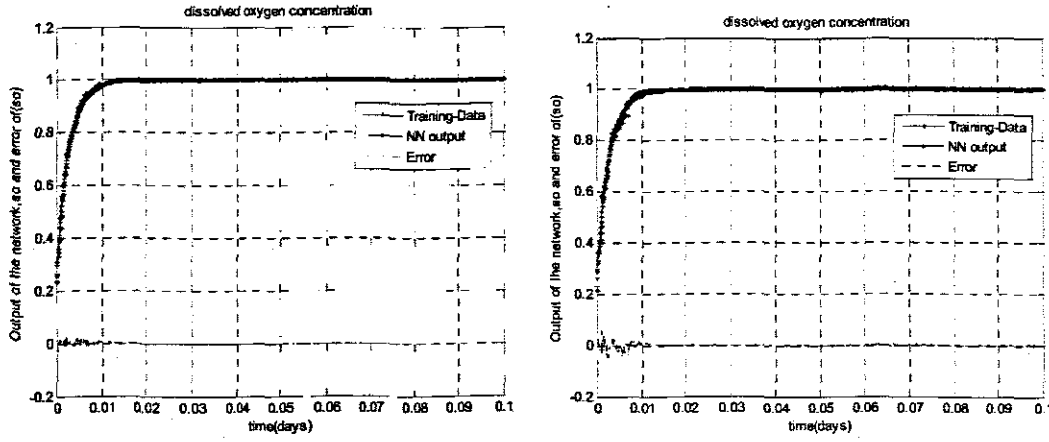


Figure 4.24: Feedforward networks response with 2 inputs, 1 hidden layer, 10 and15 hidden neurons and 1 output (model 2).

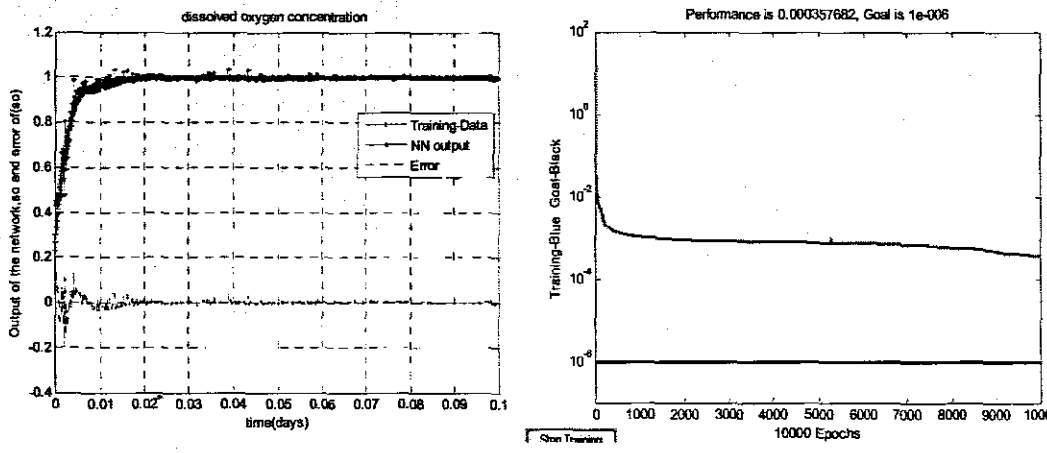


Figure 4.25: Feedforward networks response with 2 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 2) and Performance MSE achieved.

Figure 4.25 shows the MSE achieved with the Feedforward network having 20 neurons in the hidden layer. The number of neurons was incrementally added from 2 to 20 as depicted in Table 4.5. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.22 - 4.25, and analysis of Table 4.5 for the performance index (MSE) have shown that:

- With two input vector variables to the Feedforward network, the network can map the required target when trained with the *trainrp* algorithm if the hidden layer has at least 5 neurons.
- The lowest MSE of 3.10931e-005 is obtainable after several numbers of training epochs when the network has 5 neurons in the hidden layer. This can be seen from the data of Table 4.5. However, it requires at least 10000 training epochs in order to converge to this low value. The value of MSE achieved confirms that the model does approximate the target when trained with *trainrp* algorithm.

4.6.5 Feedforward identification with 2 input vectors and *trainlm* algorithm-model2

Table 4.6 shows the feed forward multilayer perceptron neural networks trained with Levenberg-Marquardt algorithm *(trainlm),* using different number of hidden neurons in the hidden layer, training epochs, learning rates and the mean squared error achieved for a vector of two input variables: the normalized input airflow rate (i1_1) and time function (Z). These combinations constitute model 2. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.26 to 4.48.

Table4.6: Feedforward identification with two input and *trainlm* algorithm-model2

Type of ANN(Network structure)	Activation functions& training algorithm*(trainlm)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	Matrix close to singularity.		
NEWFF	Tansig, Purelin	3	317	0.05	9.17779e-007
NEWFF	Tansig, Purelin	4	757	0.05	8.99907e-007
NEWFF	Tansig, Purelin	5	404	0.05	9.26623e-007
NEWFF	Tansig, Purelin	10	242	0.05	9.68056e-007
NEWFF	Tansig, Purelin	15	163	0.05	7.48259e-007
NEWFF	Tansig, Purelin	20	420	0.05	9.92369e-007

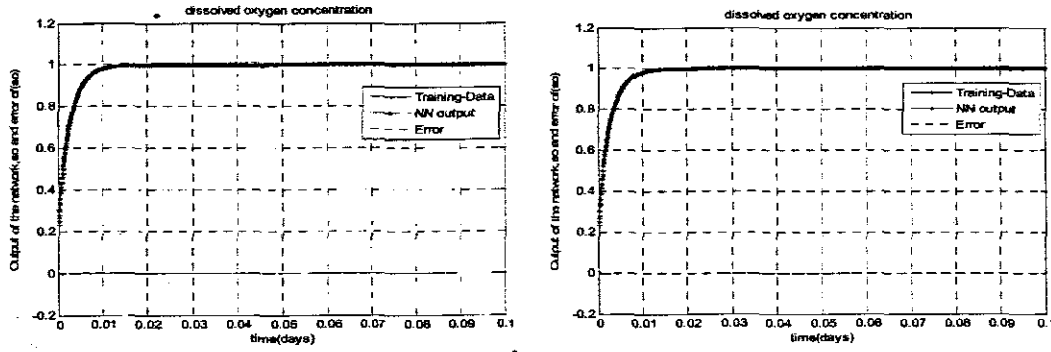


Figure 4.26: Feedforward networks response with 2 inputs, 1 hidden layer, 3 and 4 hidden neurons and 1 output (model 2).

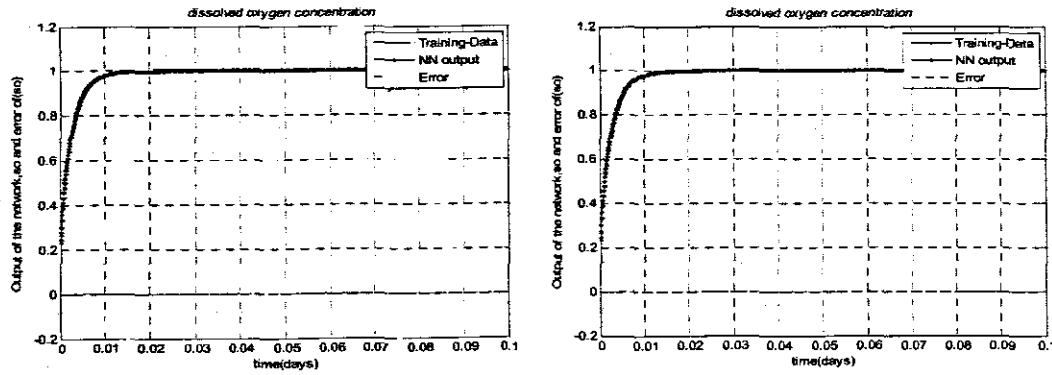


Figure 4.27: Feedforward networks response with 2 inputs, 1 hidden layer, 5 and10 hidden neurons and 1 output (model 2).

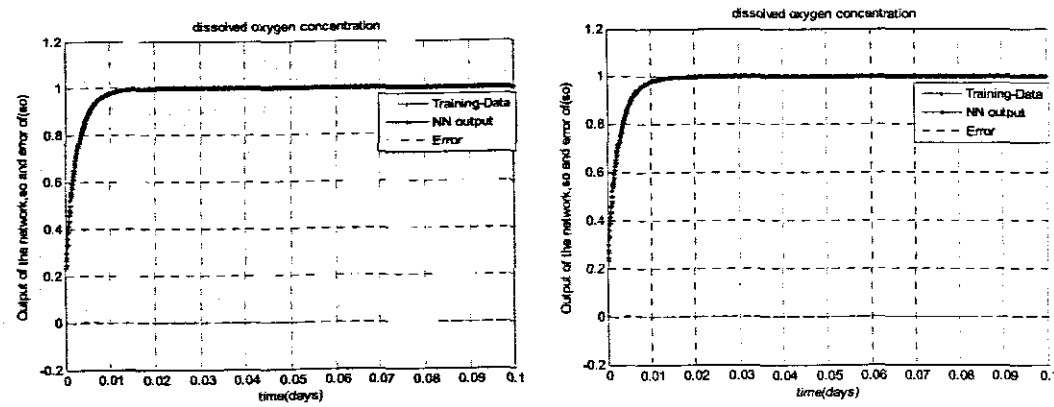


Figure 4.28: Feedforward networks response with 2 inputs 1 hidden layer, 15 and 20 hidden neurons and 1 output (model 2).

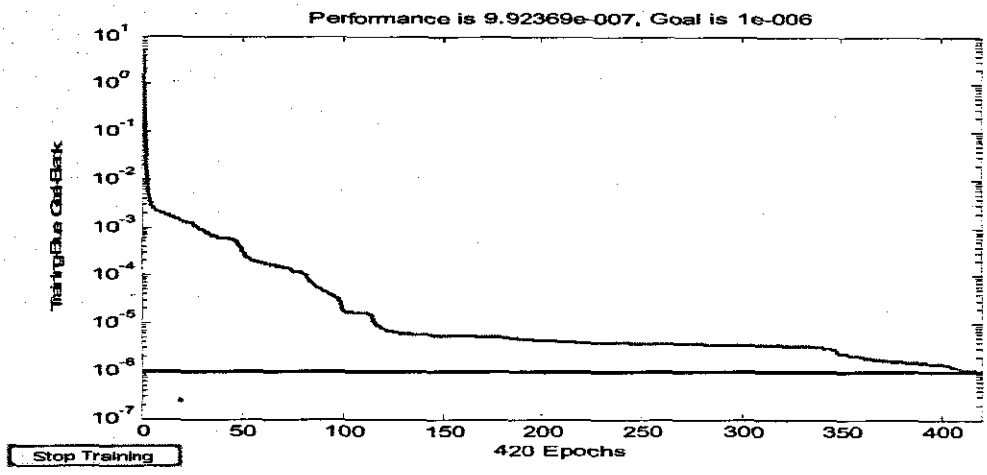


Figure 4.29: Performance MSE with 2 inputs. 1 hidden layer, 20 hidden neurons and 1 output (model 2).

Figure 4.29 shows the MSE achieved with the Feedforward network having 20 neurons in the hidden layer The numbers of neurons were incrementally changed from 2 to 20 with each set being tested as depicted in Table 4.6. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.25 - 4.29, and in addition the analysis of Table 4.6 for the performance index (MSE) have shown that:

- With two input vector variables to the Feedforward network, the network can map the required target when trained with the *trainlm* algorithm if the hidden layer has at least 3 neurons.
- The lowest MSE of 7.48259e-007 is obtainable after only 163 iterations of the training epochs when the network has 15 neurons in the hidden layer. This can be seen from the data of Table 4.6. The low values of MSE achieved from all the tests confirm that the model does approximate the target when trained with *trainlm* algorithm. However the convergence time is very small and therefore the results may not be relied upon if the training data is large.

4.6.6 Feedforward identification with 2 input vectors and *trainscg* algorithm-model2

Table 4.6 shows the feed forward multilayer perceptron neural networks trained with scaled conjugate gradient algorithm training function, the different number of hidden neurons in the hidden layer, training epochs, learning rates, the mean squared errors achieved for a vector of two input variables: the normalized input airflow rate (i1_1) and time function (Z). These combinations constitute model 2. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.30 to 4.33.

Table4.7: Feedforward identification with two inputs and *trainscg* algorithm-model2

Type of ANN(Network structure)	Activation functions& training algorithm(trainscg)	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	10000	0.05	0.000121779
NEWFF	Tansig, Purelin	3	10000	0.05	9.75194e-005
NEWFF	Tansig, Purelin	4	10000	0.05	0.000893383
NEWFF	Tansig, Purelin	5	10000	0.05	1.30457e-005
NEWFF	Tansig, Purelin	10	10000	0.05	0.000783204
NEWFF	Tansig, Purelin	15	10000	0.05	0.000698148
NEWFF	Tansig, Purelin	20	10000	0.05	0.000938997

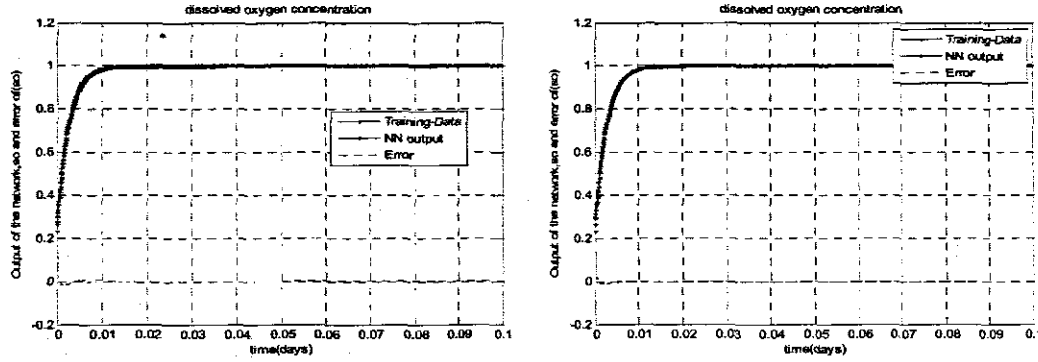


Figure 4.30: Feedforward networks response with 2 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 2).

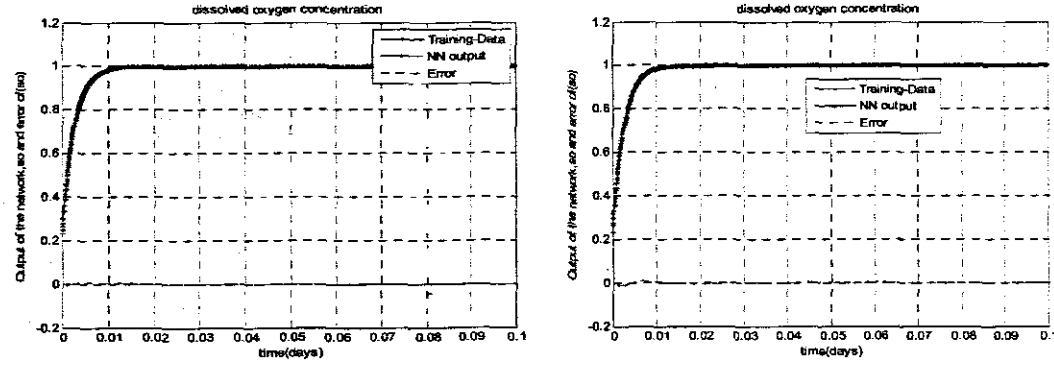


Figure 4.31: Feedforward networks response with 2 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 2).

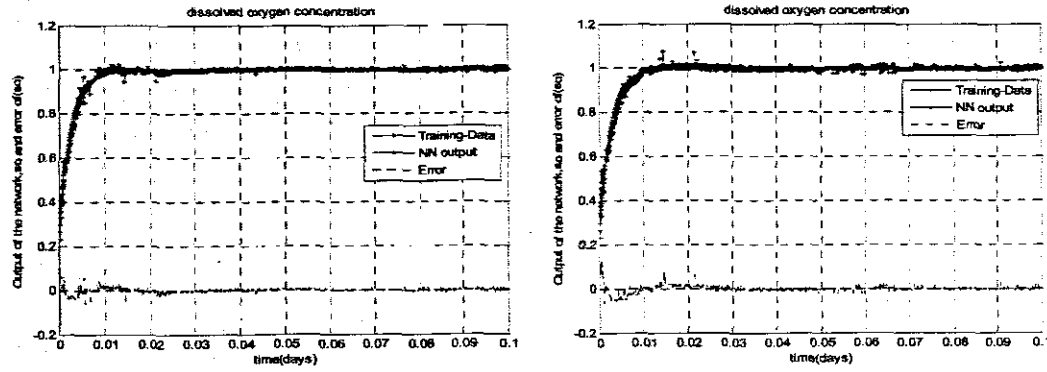


Figure 4.32: Feedforward networks response with 2 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 2).

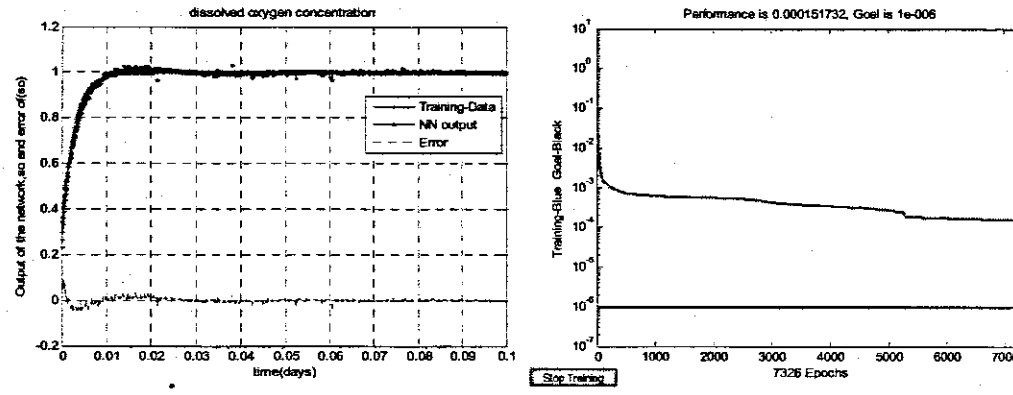


Figure 4.33: Feedforward networks response with 2 inputs. 1 hidden layer, 20 hidden neurons and 1 output (model 2) and Performance MSE achieved.

Figure 4.33 shows the MSE achieved with the Feedforward network trained by the scaled conjugate gradient (*trainscg*) algorithm and having 20 neurons in the hidden layer. The numbers of neurons were incrementally changed from 2 to 20 with each set of hidden neurons being tested as depicted in Table 4.7. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.30 - 4.33, and analysis in addition to Table 4.7 for the performance indices (MSE) have shown that:

- With two input vector variables to the Feedforward network, the network can map the required target when trained with the *trainscg* algorithm if the hidden layer has 3 neurons.
- The lowest MSE of 0.000151732 is obtainable after 10000 iterations of the training epochs when the network has 3 neurons in the hidden layer. This can be seen from the data of Table 4.7. However the convergence time for achievement of the low values of MSE from all the tests is quite long compared with *trainlm* and *trainrp* algorithms.
- Having more than 3 neurons in the hidden layer the model does not approximate the target when trained with *trainscg* algorithm.

In conclusion, from the experimental work on model2 it is found that with the number of neurons in the hidden layer increasing from 3 onwards, the network trained with *trainlm* algorithm does map the targeted trajectory however; the convergence time is very short, that shows the results may not be relied upon. The other two algorithms used for training, does map the required target approximately if the input vector has two variable ranges of values. The next network tested was three input vector consisting the variables of the normalized input flow rate trajectory, normalized DO process trajectory and the variable of time function. The procedure adopted was the same as that of one input vector. The results are explained in the proceeding sections.

4.6.7 Feedforward identification with 3 input variables and *trainrp* algorithm-model3

Table 4.8 shows the feed forward multilayer perceptron neural networks model trained with resilient backpropagation *(trainrp)* algorithm, different number of hidden layers, training epochs, learning rates and the mean squared error achieved for three input variables: the normalized input airflow rate **(i1_1)**, time function (Z) and normalized output **(i1_2)** as input variables vector to the model and the target is the DO concentration variable **(i2_2)**. These combinations constitute model 3. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.34 to 4.37 for varying number of hidden nodes.

Table4.8: Feedforward identification with three inputs and *trainrp* algorithm-model3

Type of ANN(Network structure)	Activation functions& training algorithm(trainrp)	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	10000	0.05	3.10031e-005
NEWFF	Tansig, Purelin	3	6724	0.05	2.54358e-005
NEWFF	Tansig, Purelin	4	10000	0.05	4.38057e-006
NEWFF	Tansig, Purelin	5	10000	0.05	6.17743e-006
NEWFF	Tansig, Purelin	10	10000	0.05	3.86504e-006
NEWFF	Tansig , Purelin	15	10000	0.05	1.56163e-005
NEWFF	Tansig , Purelin	20	10000	0.05	3.78744e-005

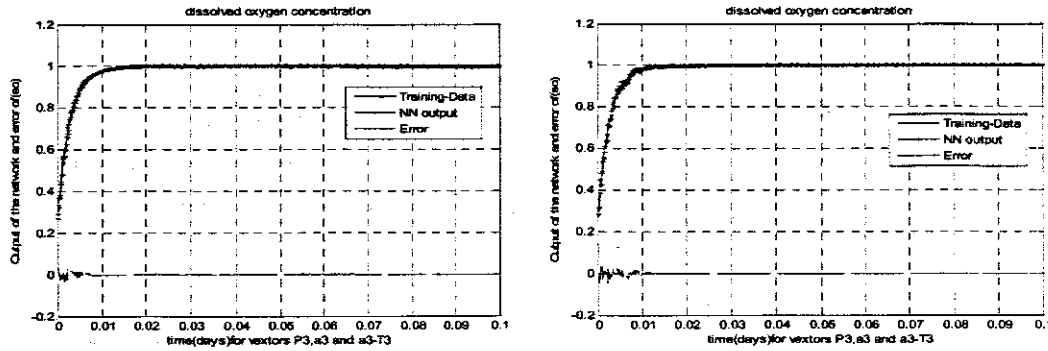


Figure 4.34: Feedforward networks response with 3 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 3).

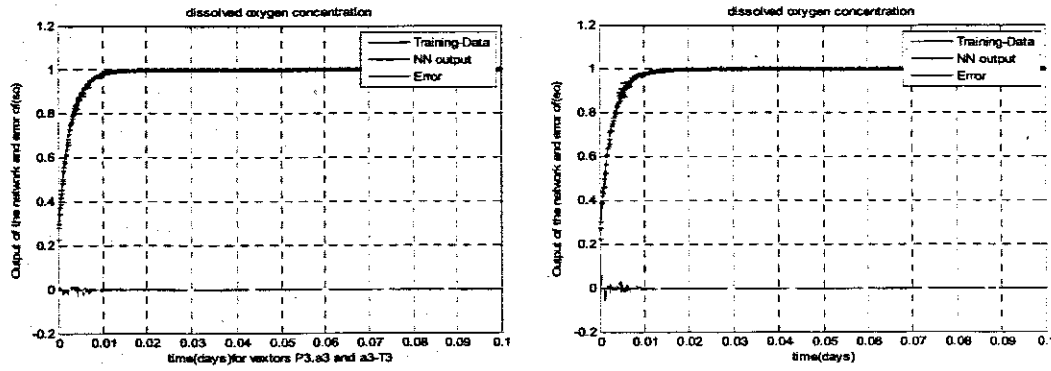


Figure 4.35: Feedforward networks response with 3 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 3).

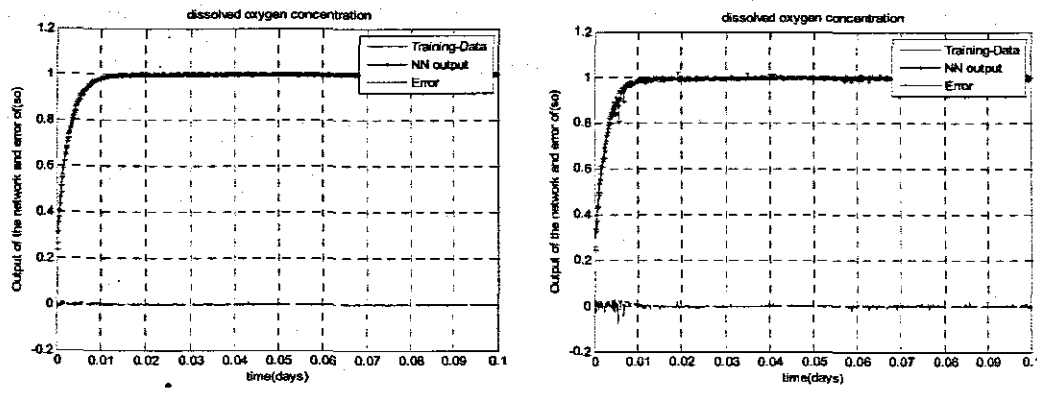


Figure 4.36: Feedforward networks response with 3 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 3).

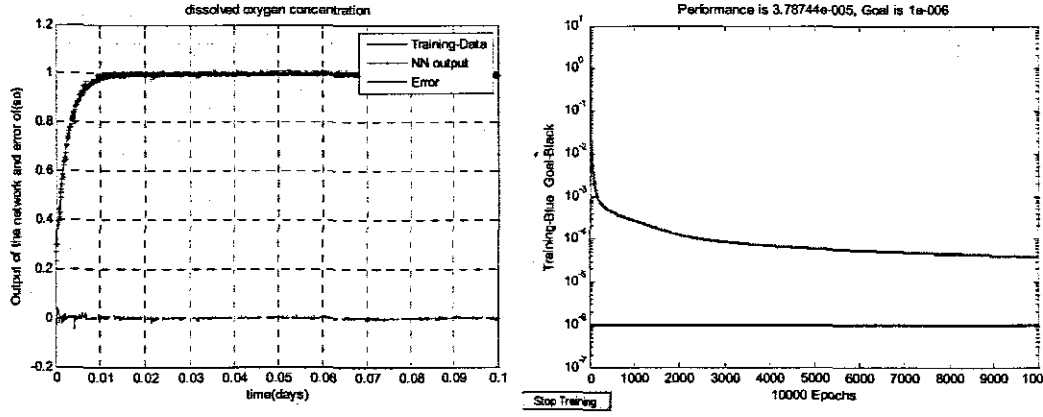


Figure 4.37: Feedforward networks response with 3 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 3) and Performance MSE achieved.

Figure 4.37 shows the MSE achieved with the Feedforward network trained by the resilient backpropagation *(trainrp)* algorithm and having 20 neurons in the hidden layer. However, the numbers of neurons were incrementally changed from 2 to 20 with each set of hidden neuron being tested as depicted in Table 4.8. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.34 - 4.37, and analysis in addition to Table 4.8 for the performance indices (MSE) have shown that:

- With three input variables to the Feedforward network, the network can map the required target when trained with the *trainrp* algorithm if the hidden layer has at least 4 or more neurons in the hidden layer.
- The lowest MSE is obtainable after 10000 iterations of the training epochs. This can be seen from the data of Table 4.8. However the convergence time for achievement of the low values of MSE from all the tests is quite long with the algorithm.

4.6.8 Feedforward identification with 3 input variables using *trainlm* algorithm-model3

Table 4.9 shows the feed forward multilayer perceptron neural networks model trained with Levenberg-Marquardt algorithm *(trainlm)* algorithm, different number of hidden nodes in the hidden layer, training epochs, learning rates and the mean squared error achieved for three input variables: the normalized input airflow rate (i1_1), time function (Z) and normalized output (i1_2) combined as input variables vector to the model and the target is the DO concentration variable (i2_2). These combinations constitute model 3. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.38 to 4.41 for varying number of hidden nodes.

Table4.9: Feedforward identification with three inputs and *trainlm* algorithm-model3

Type of ANN(Network structure)	Activation functions& training algorithm(*trainlm*)	No of hidden neurons	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	405	0.05	7.86427e-007
NEWFF	Tansig, Purelin	3	137	0.05	8.65295e-007
NEWFF	Tansig, Purelin	4	226	0.05	9.66612e-007
NEWFF	Tansig, Purelin	5	308	0.05	9.70407e-007
NEWFF	Tansig, Purelin	10	189	0.05	4.89254e-007
NEWFF	Tansig , Purelin	15	73	0.05	9.80598e-007
NEWFF	Tansig , Purelin	20	15	0.05	9.95758e-007

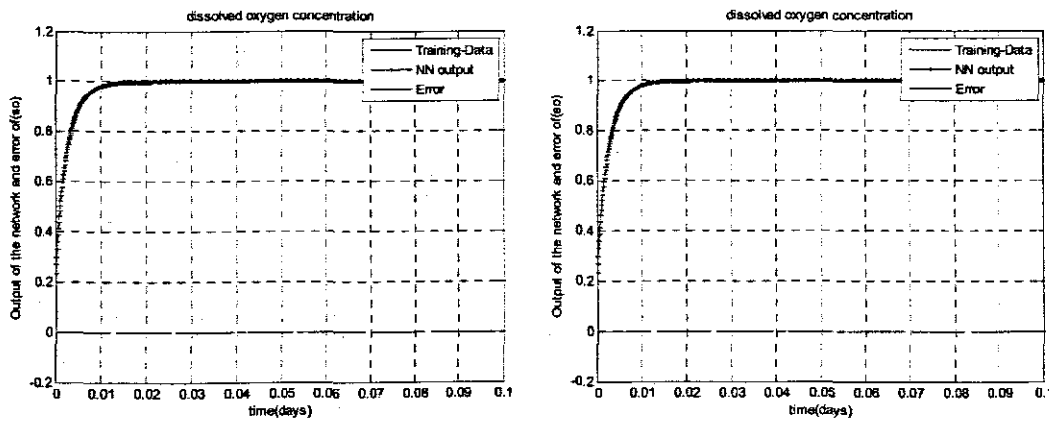


Figure 4.38: Feedforward networks response with 3 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 3).

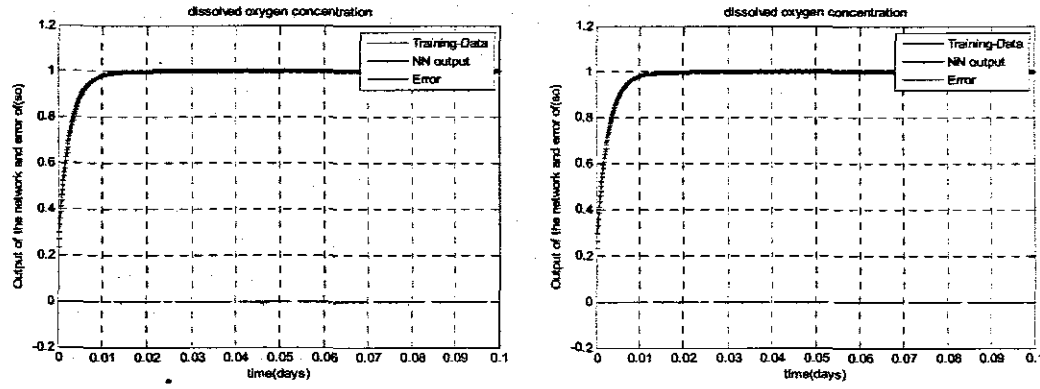


Figure 4.39: Feedforward networks response with 3 inputs, 1 hidden layer, 4 and5 hidden neurons and 1 output (model 3).

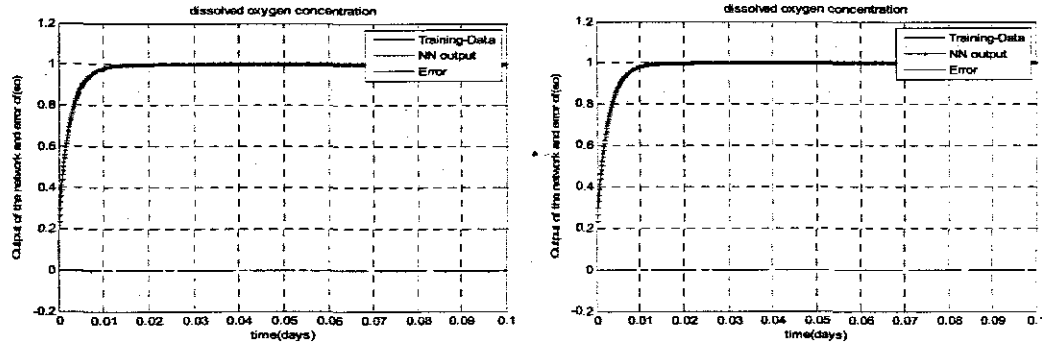


Figure 4.40: Feedforward networks response with 3 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 3).

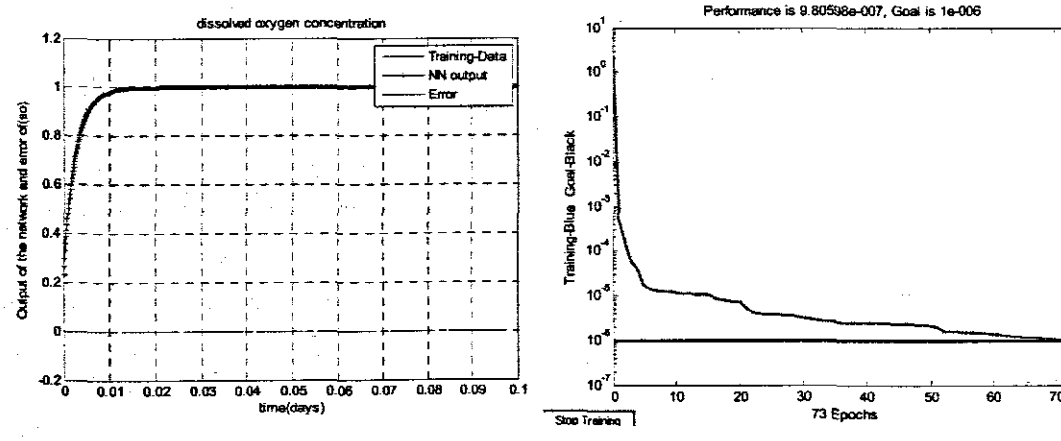


Figure 4.41: Feedforward networks response with 3 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 3) and Performance MSE.

Figure 4.41 shows the MSE achieved with the Feedforward network trained by the resilient backpropagation (trainlm) algorithm and having 15 neurons in the hidden layer. The numbers of neurons were incrementally changed from 2 to 20 with each set of hidden neurons being tested as depicted in Table 4.9. Analysis of the graphs plotted

for the training data, NN output and the error as a function of time as illustrated in Figures 4.38 - 4.41, and analysis in addition to Table 4.9 for the performance indices (MSE) have shown that:

- With three input variables to the Feedforward network, the network can map the required target when trained with the *trainlm* algorithm if the hidden layer has at least 3 or more neurons.
- This algorithm requires the least number of training epochs to converge to the lowest MSE. This can be seen from the set of data given in Table 4.9, and the performance index of Figure 4.41 that shows that with only 73 training epochs an MSE of 9.80598e-007 is obtainable.

4.6.9 Feedforward identification with 3 input variables and *trainscg* algorithm-model3

Table 4.10 shows the feed forward multilayer perceptron neural networks model trained with Scaled conjugate gradient algorithm *(trainscg)* algorithm, different number of hidden nodes in the hidden layer, training epochs, learning rates and the mean squared error achieved for three input variables: the normalized input airflow rate (i1_1), time function (Z) and normalized output (i1_2) as input variables vector to the model and the target is the DO concentration variable (i2_2). These combinations constitute model 3. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent figures 4.42 to 4.4 for varying number of hidden nodes.

Table4.10: Feedforward identification with three inputs and *trainscg* algorithm-model3

Type of ANN(Network structure)	Activation functions& training algorithm*(trainscg)*	No of hidden (layers)	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	5123	0.05	8.86238e-006
NEWFF	Tansig, Purelin	3	2217	0.05	9.99855e-006
NEWFF	Tansig, Purelin	4	3117	0.05	4.54658e-006
NEWFF	Tansig, Purelin	5	6485	0.05	4.09931e-006
NEWFF	Tansig, Purelin	10	6806	0.05	6.01692e-006
NEWFF	Tansig, Purelin	15	10000	0.05	8.39298e-006
NEWFF	Tansig, Purelin	20	1184	0.05	9.98936e-007

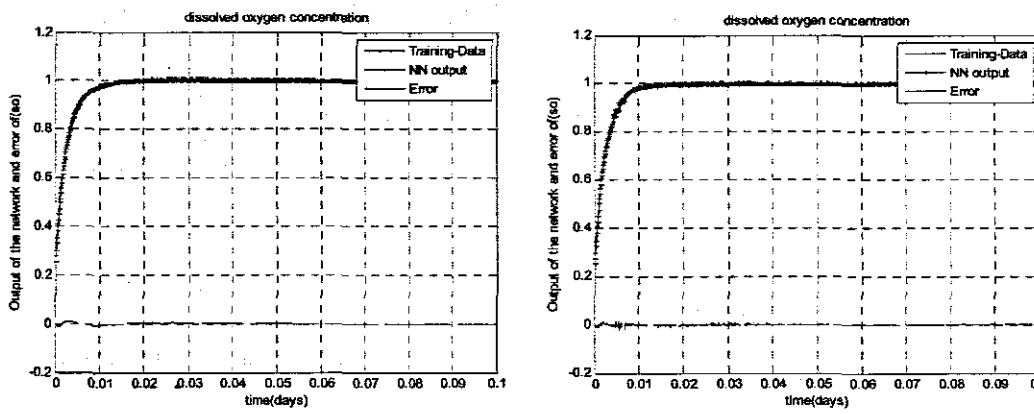


Figure 4.42: Feedforward networks response with 3 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 3).

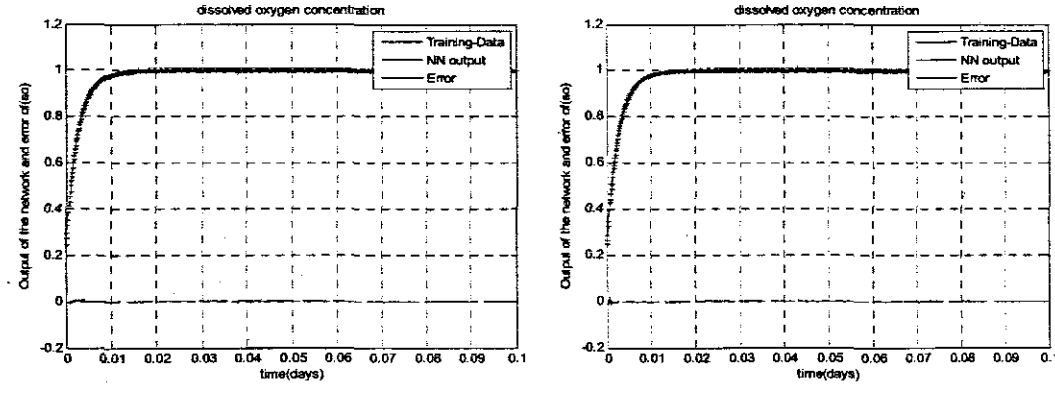


Figure 4.43: Feedforward networks response with 3 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 3).

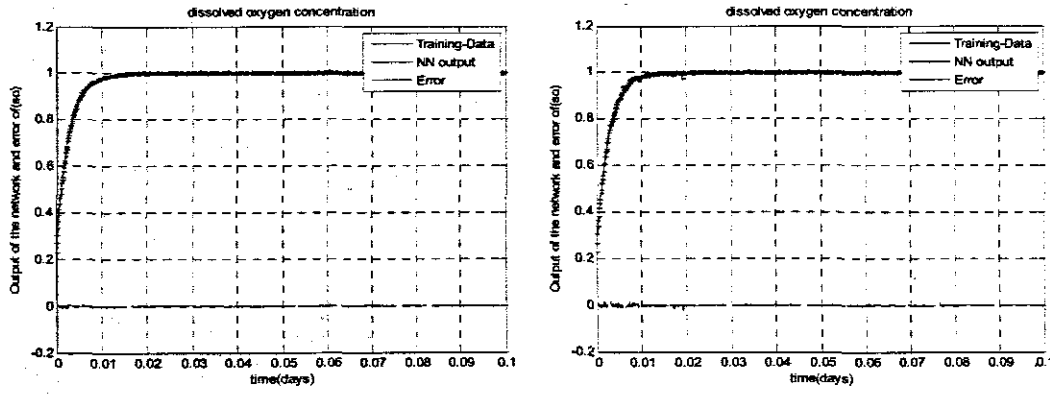


Figure 4.44: Feedforward networks response with 3 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 3).

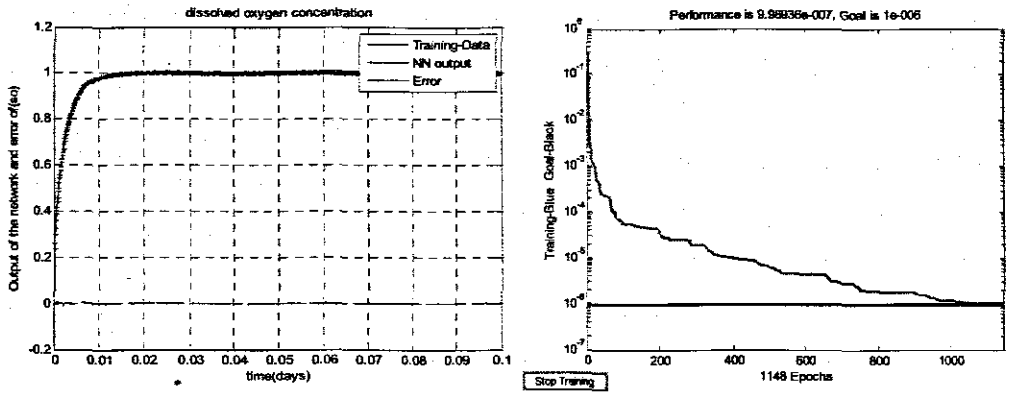


Figure 4.45: Feedforward networks response with 3 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 3) and Performance MSE.

Figure 4.45 shows the MSE achieved with the Feedforward network trained by the Scaled conjugate gradient algorithm *(trainscg)* algorithm and having 20 neurons in the hidden layer. The numbers of neurons were incrementally changed from 2 to 20 with each set of hidden neurons being tested as depicted in Table 410. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.42 - 4.45, and analysis in addition to Table 4.10 for the performance indices (MSE) have shown that:

- With three input variables to the Feedforward network, the network can map the required target when trained with the *trainscg* algorithm if the hidden layer has at least 3 neurons or more.
- However as the number of neurons in the hidden layer is increased, this algorithm requires more training epochs in order to converge to the lowest MSE. This can be seen from the set of data given in table 4.10, and the performance index of figure 4.80 that shows that an MSE of9.98936e-007 is obtainable with 1184 training epochs.

In conclusion, from the experimental work on model3, it is found that with the number of neurons in the hidden layer increasing from 3 onwards, and with the different types of algorithms used for training, the network maps the required target very well if the input vector has three variables of i1_1, i2_2 and Z. The *trainlm* algorithm performs better than the other algorithms since the convergence time is minimal and occurs only after very little number of iterations. It is also the fastest of the algorithms used. The next network tested is twelve inputs consisting of the variables of the normalized input flow rate trajectory together with its three delayed values trajectories, normalized DO process trajectory with its three delayed values trajectories and the variable of time

function with its three delayed values trajectories. The procedure adopted was the same as that of one input vector. The results are explained in the proceeding sections.

4.6.10 Feedforward identification with 12 input variables and *trainrp* algorithm-model4

Table 4.11 shows the feed forward multilayer perceptron neural networks model trained with resilient backpropagation algorithm *(trainrp)*, different number of hidden nodes in the hidden layer, training epochs, learning rates and the mean squared error achieved for twelve input variables: the normalized input airflow rate (i1_1), three delayed values of the normalized input airflow rates, time, three delayed time functions, normalized output (i2_2) and three delayed normalized values of the output forming the input vector P=[Z m0 m1 m2 i1_1 k0 k1 k2 i2_2 n0 n1 n2]', where the variables have been defined and an output vector T=[i2_2]'. These combinations constitute model 4.

Table4.11: Feedforward identification with twelve inputs and *trainrp* algorithm - model4

Type of ANN(Network structure)	Activation functions& training algorithm*(trainrp)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	10000	0.05	1.64191e-006
NEWFF	Tansig, Purelin	3	10000	0.05	1.4405e-006
NEWFF	Tansig, Purelin	4	7418	0.05	9.99985e-007
NEWFF	Tansig, Purelin	5	1000	0.05	6.87302e-006
NEWFF	Tansig, Purelin	10	10000	0.05	6.73703e-006
NEWFF	Tansig , Purelin	15	10000	0.05	1.37315e-005
NEWFF	Tansig , Purelin	20	10000	0.05	1.50929e-006

The trajectories of the calculated outputs of the NN in comparison with the target outputs and the trajectories of the errors between them for the model4 trained with *trainrp* algorithm for varying number of hidden nodes are shown in the Figures 4.46 to 4.49.

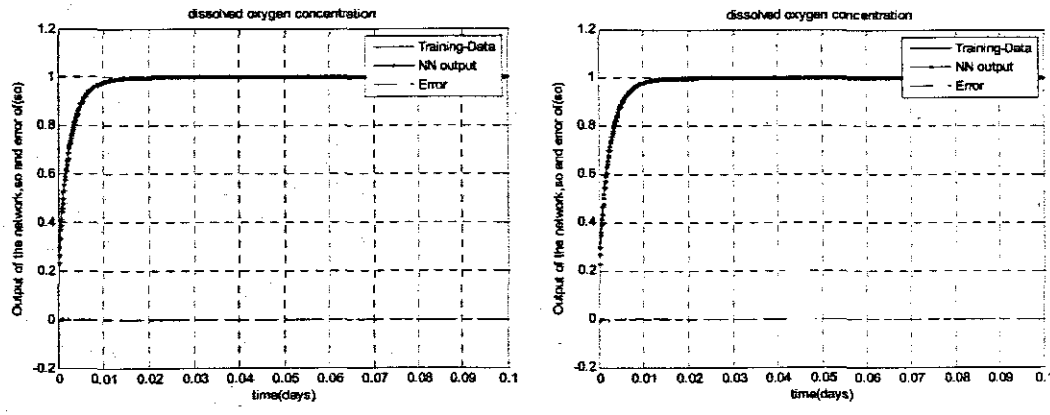


Figure 4.46: Feedforward networks response with 12 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 4).

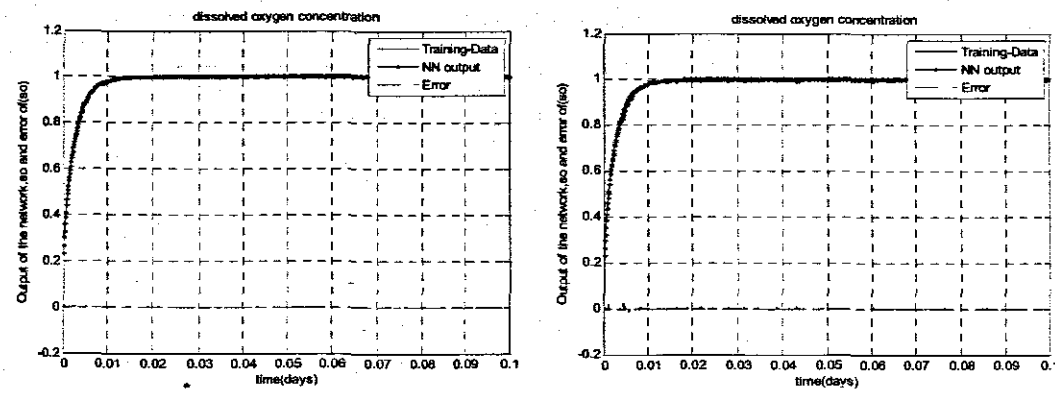


Figure 4.47: Feedforward networks response with 12 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 4).

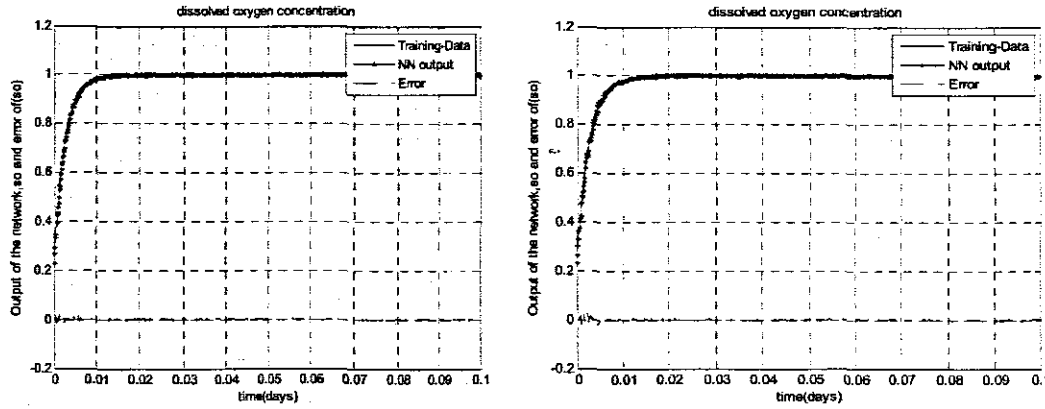


Figure 4.48: Feedforward networks response with 12 inputs, 1 hidden layer, 10 and15 hidden neurons and 1 output (model 4).

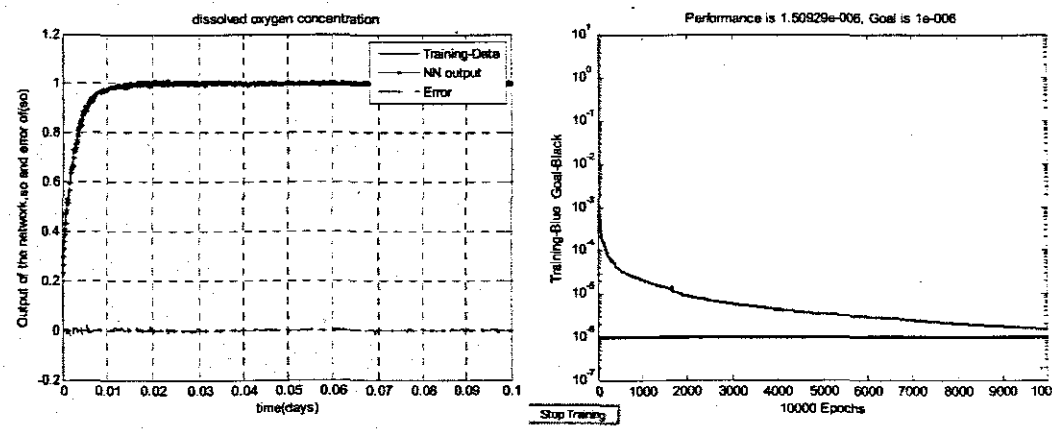


Figure 4.49: Feedforward networks response with 12 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 4) and Performance MSE achieved.

Figure 4.49 shows the MSE achieved with the Feedforward network trained by the resilient backpropagation *(trainrp)* algorithm and having 20 neurons in the hidden layer. The numbers of neurons were incrementally changed from 2 to 20 with each set of

hidden neurons being tested as depicted in Table 4.11. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.46 - 4.49, and in analysis addition to Table 4.11 for the performance indices (MSE) have shown that:

- With the twelve input vector variables to the Feedforward network, the network can map the required target when trained with the *trainrp* algorithm so long as the hidden layer has at least 4 neurons and above.
- However this algorithm requires more training epochs in order to converge to the lowest MSE. This can be seen from the set of data given in Table 4.11, and the performance index of Figure 4.49 that shows that an MSE of 1.50929e-006 is obtainable with 10000 training epochs.

4.6.11 Feedforward identification with 12 input variables and *trainlm* algorithm-model4

Table 4.12 shows the feed forward multilayer perceptron neural networks model trained by the Levenberg-Marquardt algorithm (trainlm), different number of hidden nodes in the hidden layer, training epochs, learning rates and the mean squared error achieved for twelve input variables: the normalized input airflow rate (i1_1), three delayed values of the normalized input airflow rates, time, three delayed time functions, normalized output (i2_2) and three delayed normalized values of the output forming the input vector P=[Z m0 m1 m2 i1_1 k0 k1 k2 i2_2 n0 n1 n2]', and an output vector T=[i2_2]'. These combinations constitute model 4. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.50 and 4.53 for varying number of hidden nodes.

Table4.12: Feedforward identification with twelve inputs and *trainlm* algorithm-model4

Type of ANN(Network structure)	Activation functions& training algorithm*(trainlm)*	No of hidden neurons	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	49	0.05	9.77948e-007
NEWFF	Tansig, Purelin	3	16	0.05	6.18441e-007
NEWFF	Tansig, Purelin	4	23	0.05	9.30693e-007
NEWFF	Tansig, Purelin	5	46	0.05	9.78218e-007
NEWFF	Tansig, Purelin	10	7	0.05	9.2187e-007
NEWFF	Tansig, Purelin	15	8	0.05	1.48482e-007
NEWFF	Tansig, Purelin	20	10	0.05	9.32963e-007

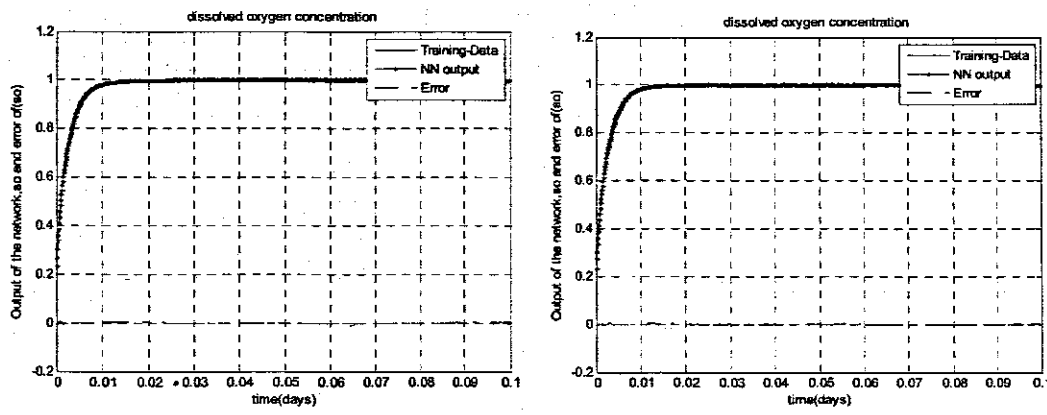


Figure 4.50: Feedforward networks response with 12 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 4).

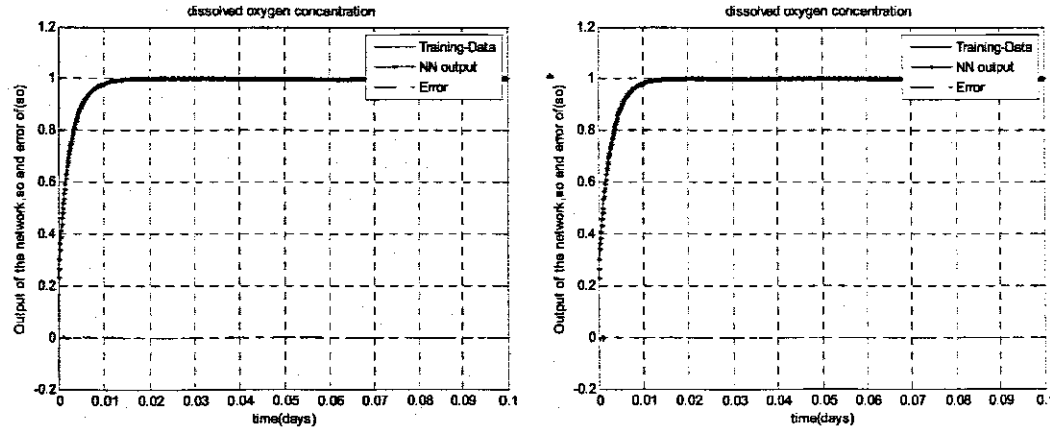


Figure 4.51: Feedforward networks response with 12 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 4).

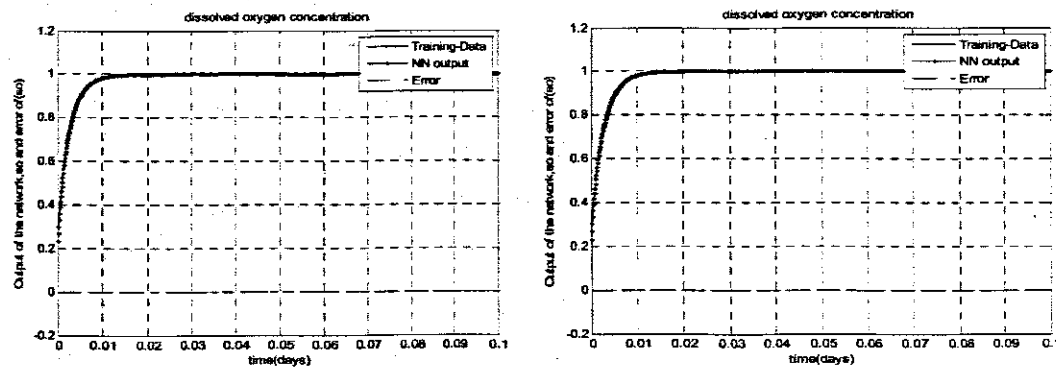


Figure 4.52: Feedforward networks response with 12 inputs, 1 hidden layer, 10 and15 hidden neurons and 1 output (model 4).

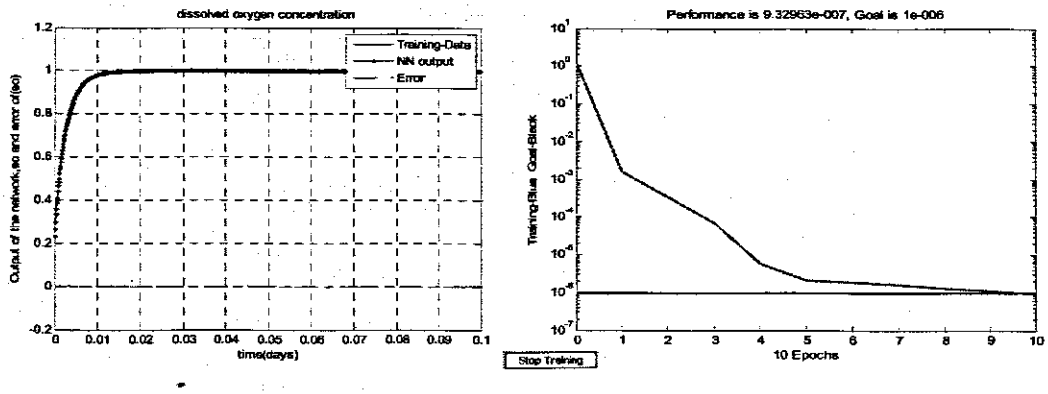


Figure 4.53: Feedforward networks response with 12 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 4) Performance MSE achieved.

Figure 4.53 shows the MSE achieved with the Feedforward network trained by the Levenberg-Marquardt algorithm *(trainlm),* and having 20 neurons in the hidden layer. The numbers of neurons were incrementally changed from 2 to 20 with each set of hidden neurons being tested as depicted in Table 4.12. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.89 - 4.54, and in addition to Table 4.12 for the performance indices (MSE) have shown that:

- With the twelve input vector variables to the Feedforward network, the network can map the required target when trained with the *trainlm* algorithm so long as the hidden layer has at least 3 or more neurons.
- This algorithm converges very fast to the lowest MSE. This can be seen from the set of data given in table 4.12, and the performance index of figure 4.93 that shows that an MSE of 9.32963e-007 is obtainable with 7 training epochs only. Therefore the results may not be accurate.

4.6.12 Feedforward identification with 12 inputs and *trainscg* algorithm-model4

Table 4.13 shows the feed forward multilayer perceptron neural networks model trained by the scaled conjugate gradient algorithm *(trainscg),* different number of hidden nodes in the hidden layer, training epochs, learning rates and the mean squared error achieved for the twelve input variables: the normalized input airflow rate (i1_1), three delayed values of the normalized input airflow rates, time, three delayed time functions, normalized output (i2_2) and three delayed normalized values of the output forming the input vector P=[Z m0 m1 m2 i1_1 k0 k1 k2 i2_2 n0 n1 n2]', and an output vector T=[i2_2]'. These combinations constitute model 4. The responses of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.54 to 4.57 for varying number of hidden nodes.

Table4.13: Feedforward identification with twelve inputs and *trainscg* algorithm – model4

Type of ANN(Network structure)	Activation functions& training algorithm*(trainscg)*	No of hidden (layers)	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	2879	0.05	1.58367e-005
NEWFF	Tansig, Purelin	3	5427	0.05	1.25745e-006
NEWFF	Tansig, Purelin	4	6165	0.05	2.62275e-006
NEWFF	Tansig, Purelin	5	3079	0.05	3.11522e-006
NEWFF	Tansig, Purelin	10	2410	0.05	9.91026e-007
NEWFF	Tansig, Purelin	15	2150	0.05	9.99971e-007
NEWFF	Tansig, Purelin	20	1607	0.05	9.99957e-007

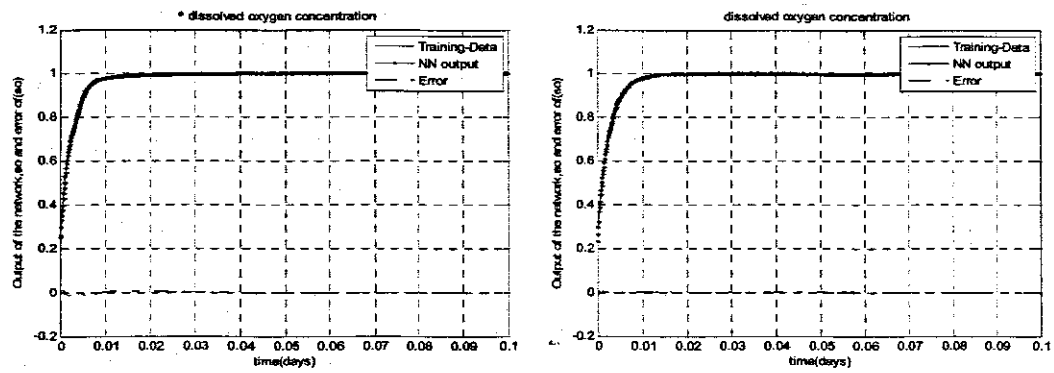


Figure 4.54: Feedforward networks response with 12 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 4).

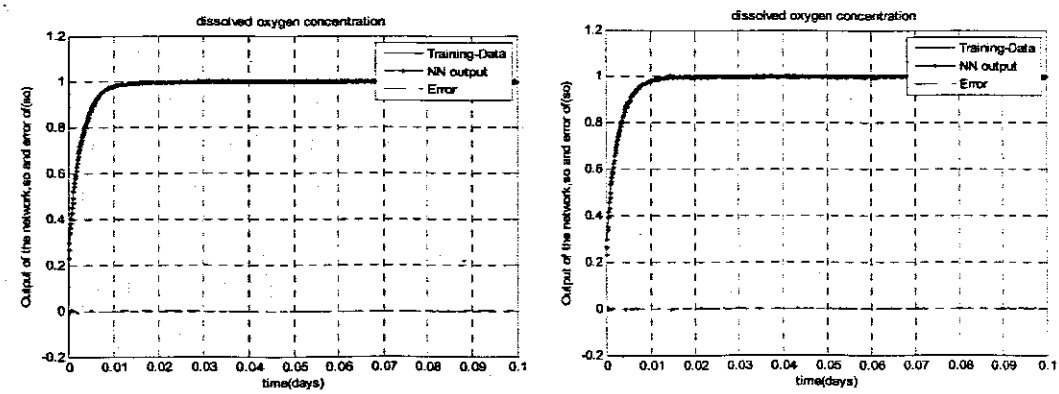


Figure 4.55: Feedforward networks response with 12 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 4).

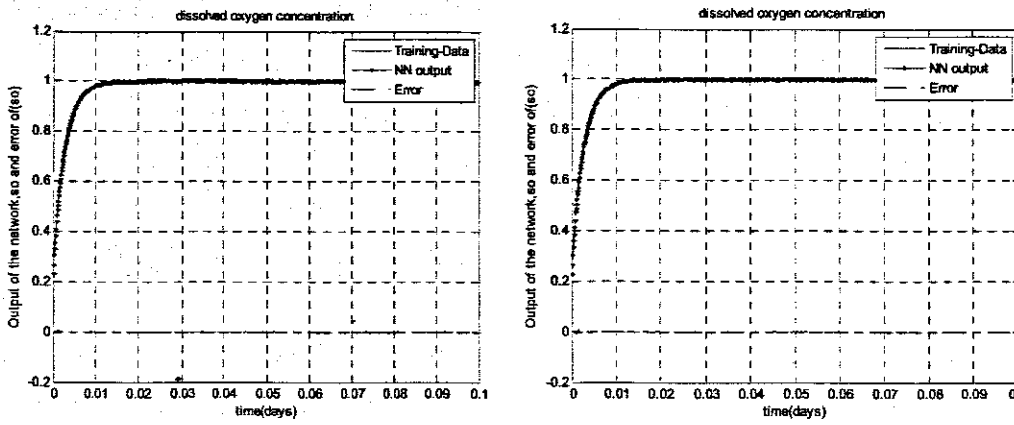


Figure 4.56: Feedforward networks response with 12 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 4).

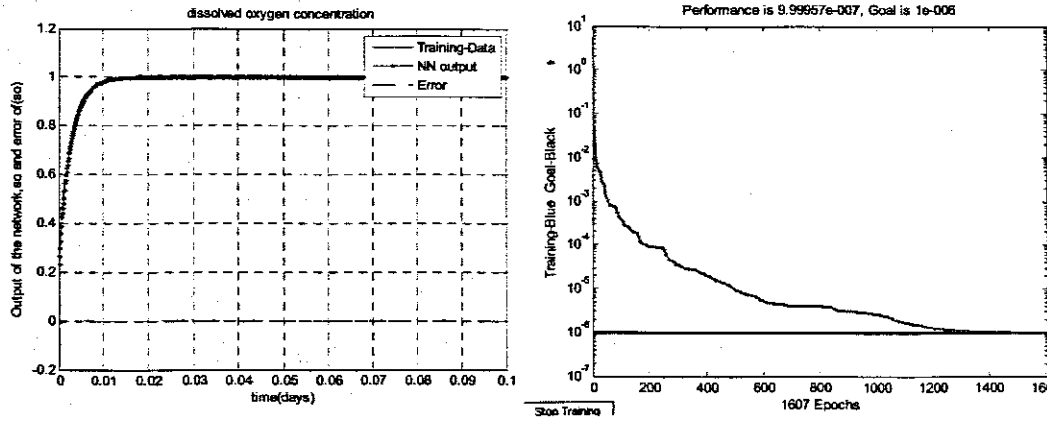


Figure 4.57: Feedforward networks response with 12 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 4) Performance MSE achieved.

Figure 4.57 shows the MSE achieved with the Feedforward network trained by the scaled conjugate gradient (trainscg), and having 20 neurons in the hidden layer. However, the numbers of neurons were incrementally changed from 2 to 20 with each set of hidden neurons being tested as depicted in table 4.13. Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.54 - 4.57, and in addition to table 4.13 for the performance indices (MSE) resulting from varying neurons in the hidden layer, have shown that:

- With the twelve input vector variables to the Feedforward network, the network can map the required target when trained with the *trainscg* algorithm so long as the hidden layer has at least 3 or more neurons.
- However this algorithm converges very fast to the lowest MSE as the number of neurons to the hidden layer is increased. This can be seen from the set of data given in Table 4.13, and the performance index of Figure 4.57 that shows that an MSE of 9.99957e-007 is obtainable with 1607 training epochs only.

4.6.13 Discussion of the results for feedforward networks

Four models have been developed and trained with three different types of training algorithms in order to determine which model describes the best mapping of the DO process dynamics and which training algorithm gives the best results. The models developed are summarized in Table 4.1. Models 3 and 4 gave better results of the mapping compared to models 1 and 2. However, a variation of model 4 where six input variables are used for the identification of the DO process as subsequently discussed is chosen as the best model. In terms of the training algorithms used, the Levenberg-Marquardt (*trainlm)* algorithm, performed better than the other two algorithms in terms of the speed of convergence, accuracy of approximation, ability to drive the mean squared error to the lowest level. Also, the error in the Levenberg-Marquardt (trainlm) algorithm decreases much more rapidly with time than the other two algorithms. Similarly the scaled conjugate gradient descent algorithm converges faster than the resilient backpropagation algorithm however it requires more iterations than *trainlm* algorithm to converge to minimum MSE. Thus from the three algorithms tested, the Levenberg-Marquardt *(trainlm)* algorithm is chosen for training the of model the DO process chosen because it requires few convergence epochs, it is also the fastest, it guarantees a low MSE, and gives better function approximation of the target trajectory than the other two.

The section that follows provides the analysis of the results for the chosen model trained with *trainlm* algorithm and verification of the suitability of the model and the algorithm.

4.6.14 Best selected model of the DO process and the verification results

The data for training, validation and testing were sampled at $1.041677E^{-4}$ days (1min.30s) intervals with 960 data sets collected. 480 data sets were used for training, 240 data sets for validation (for observing the performance of the models when faced with unknown situations or dynamic changes of different amplitudes and/or frequencies) and 240 data sets for testing. The division was such that the testing set starts with the second point of the considered time period of the DO process behavior and takes every fourth point. The validation set starts with the fourth point and take every fourth point of the time period, while the training set takes the remaining points. The data collected included the present and past values of the control input airflow rate $u(k)$ and the DO concentration dynamics $S_O(k)$ as required for identification purposes. The data gathered were scaled for training purposes. In this work the training data were normalized to the range [0 1], as i1_1 and i2_2 for $u(k)$ and $S_O(k)$ respectively and the network was created and trained using the normalized data and then

simulated. Eventually the network output was de-normalized, and a linear regression between the network outputs (de- normalized) and the targets to check the quality of the network training was performed. The following Matlab algorithm of Figure 4.58 depicts the procedure used to normalize data, split the data, train the network, simulate and perform linear regression analysis.

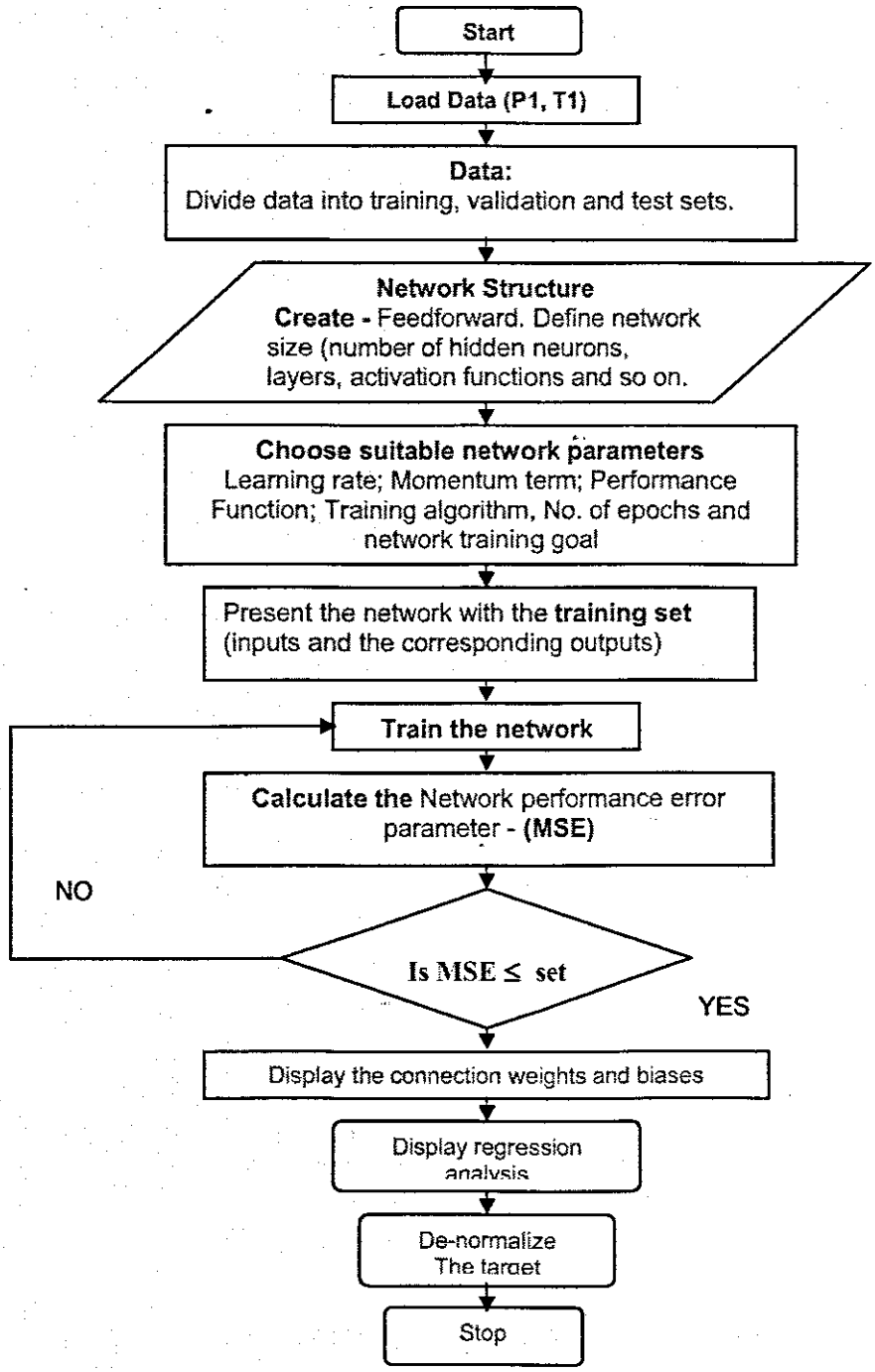


Figure 4.58: The algorithm used to train the model of the feedforward network

The above Matlab algorithm creates a feedforward network. *P1, T1* represents the variables of the normalized inputs and target vectors. k0, k1, n0 and n1 are the delays

in i1_1 and i2_2. The best structure of the identified neural model chosen was 6-8-1; i.e. 6 inputs of the normalized airflow rate with two delayed values k0 and k1 and normalized DO process with two delayed values n0 and n1, 8 nodes in the hidden layer and 1 node at the output.

Training was done by switching between the training, validation and test data set; a technique known as '*early stopping* methodology'. The training was performed in the normal way on the training set until a reasonably small error (root mean square - RMS) was achieved for this set. Once a reasonable and continuously decreasing RMS error was achieved for all the sets of data, the training was stopped automatically and the network was validated by applying different test data, not utilized before. Eventually the topology and configuration were finalized. The method of early stopping was to prevent over- training. The error on the validation set will normally decrease during the initial part of the training. However, when the network begins to over fit the data, the validation set error will start to rise. When this increase continues for a pre defined number of iterations the training is stopped and the weight values are kept. The test set is used to compare with the validation set to see if they exhibit a similar behavior. If validation errors and test errors do not show a similar behavior, this may indicate a poor division of data. In Figure 4.59, the mean squared error between the calculated values and the target training, validation and test sets are plotted against the epoch (number of iterations).

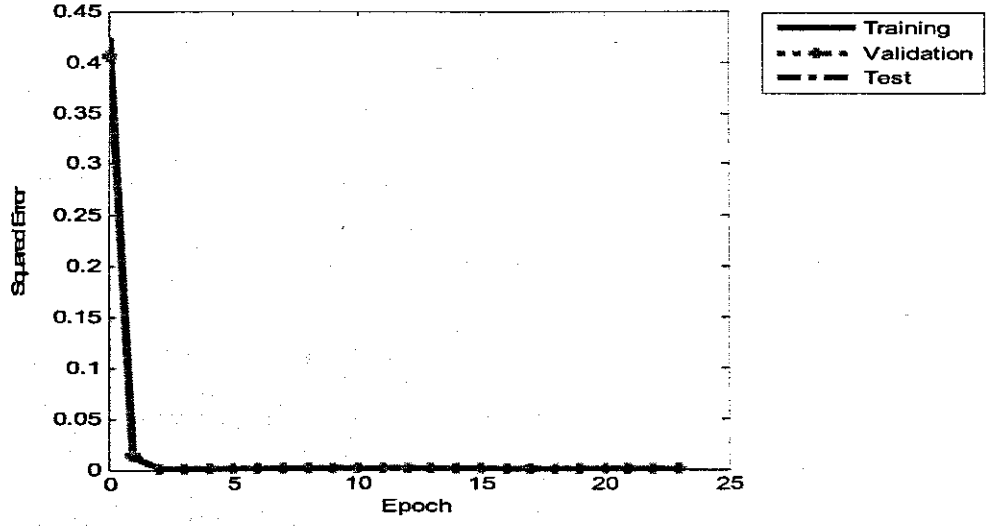


Figure 4.59: Plot of the training, validation and test errors for a generalized neural network.

The results of Figure 4.60 shows the plot of the mean squared error between the calculated values and the target training, validation and test sets plotted against the epoch (number of the training, validation and test errors) with the convergence performance of the MSE of $2.1631e^{-7}$ achieved for architecture with 8 hidden neurons.

Analysis of the plot tells that it requires only **15** iterations to map the target when trained with *trainlm* algorithm.

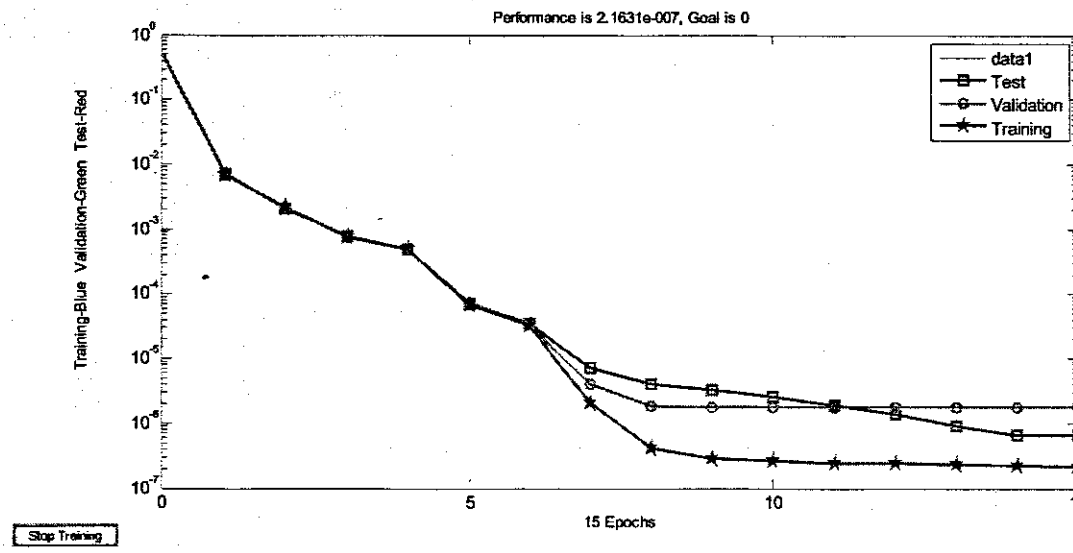


Figure 4.60: Performance MSE with **6** inputs, **1** hidden layer, 8 hidden neurons and **1** output.

The post analysis of the plots showing regression analyses between the network outputs and the corresponding targets (in original units) is subsequently done as illustrated in Figure 4.61.

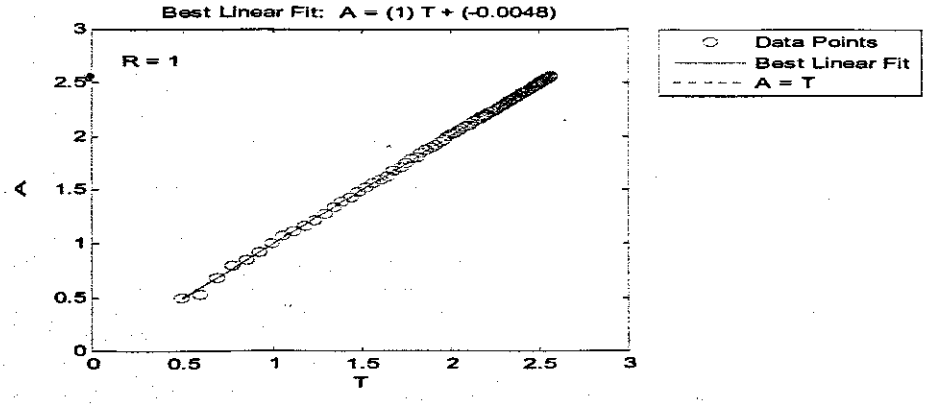


Figure 4.61: Post-Training analysis with Matlab command routine *(postreg)*

Figure 4.61 displays the plots showing regression analyses between the network outputs and the corresponding targets (in original units of data). The routine *postreg* is designed to perform this analysis. Here the network output and the corresponding targets are passed to *postreg*. It returns three parameters. The slope, and the y-intercept of the best linear regression relating targets to network outputs and the correlation coefficient (**R-value**) between the outputs and targets. In the analysis it can be seen that a perfect fit indicated by the solid line results (outputs are exactly equal to targets), because the slope is equal to one **1.0019**, and the y-intercept is **-0.0048**, and

the R-value is also equal to 0.9999 because the fit is good. This therefore confirms that the chosen model is a good identifier of the DO process.

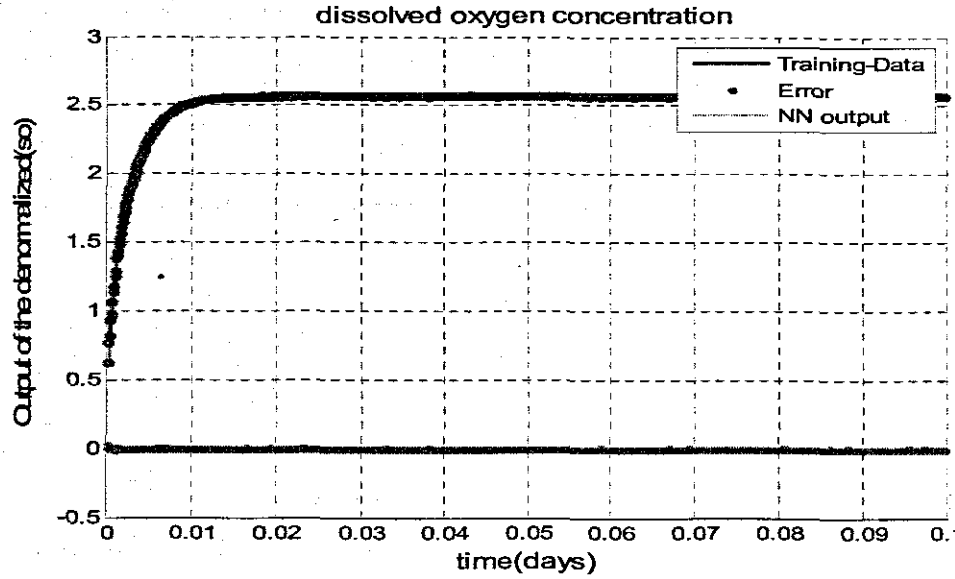


Figure 4.62: Training data, NN output and the error response of the de-normalized DO process.

Figure 4.62 shows the de-normalized neural network output. This trajectory is an approximate of the DO process model. This therefore conforms that the results of the verification is correct. The coefficients of the selected trained model are tabulated in the following Tables 4.14 and 4.15 respectively.

Table4.14: Hidden layer weights and biases

Hidden layer weights (w) for the inputs							Hidden layer bias (bw)
inputs / nodes	i1_1	k0	k1	i2_2	n0	n1	
1	0.2529	3.0721	1.8230	-1.4732	-0.0026	0.0390	-3.8655
2	14.9601	0.1787	2.0162	2.6966	0.4562	-1.9691	-1702286
3	-11.3550	0.6265	0.7572	-3.1186	1.3531	2.8114	9.2961
4	9.0209	-0.9406	1.9078	2.0463	-1.9158	-2.959	-10.9093
5	17.9717	-0.6396	-102004	-2.0904	2.4011	0.0346	-16.5687
6	11.8529	-0.5056	-0.6998	-1.0551	-2.2973	1.5074	-7.7795
7	24.1757	-0.8993	-2.7178	-0.7444	0.1513	0.9969	-19.5192
8	24.9854	-0.3815	-1.364	2.5122	-0.1113	-1.308	-21.2673

Table 4.15: Output layer weights and biases

Hidden weights(v)	-0.0466	0.0643	0.4622	-0.1818	0.2917	-0.3228	0.0945	0.1525
Hidden bias(bv)	0.6329							

The section that follows will consider the recurrent neural networks for the identification of the DO process. The models are developed and trained with three different algorithms.

4.7 Recurrent ANN identification with different activation functions

The next type of ANN tested are the Elman networks. These networks are discussed in Chapter 3. The built in feedback loops of the Elman networks enable them to have dynamic memory and hence they are more suitable for identification of dynamic systems.

4.7.1 Construction of the recurrent neural networks

The Elman network has *"tansig"* neurons in its hidden (recurrent) layer, and *purelin* neuron in its output layer. This combination is special because the two-layer networks with these transfer functions can approximate any function (with a finite number of discontinuities) with arbitrary accuracy. The only requirement is that the hidden layer must have enough neurons. However, more hidden neurons are needed as the function being fitted increases in complexity. An Elman network with two or more layers can be created with the Matlab function *"newelm"*. The hidden layers commonly have *"tansig"* transfer functions, so that it is the *default* for *newelm*. As shown in Figure 108, *purelin* is the commonly used output-layer transfer function. The feedback in the considered Elman neural network model is from the output of the hidden layer to its input using a delay of one step of time for all variables.

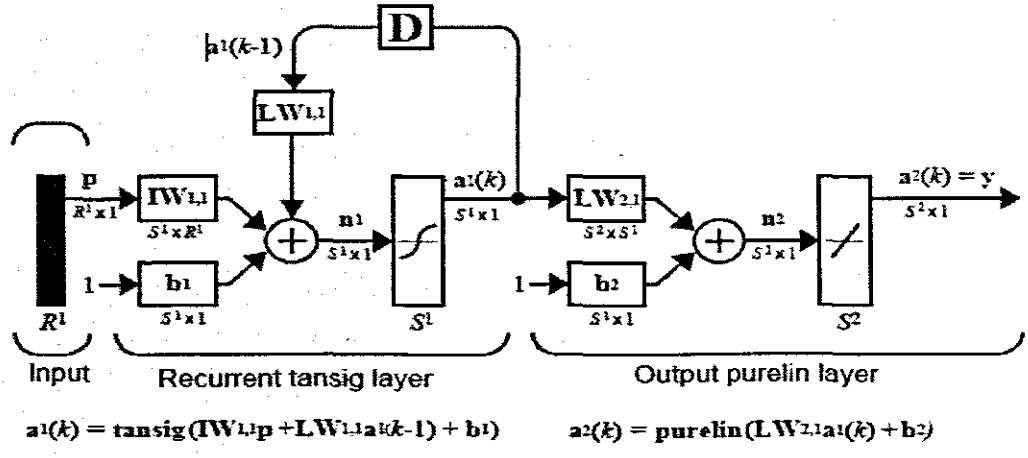


Figure 4.63: Architecture of Elman networks (Howard Demuth, Mark Beale, 2002)

The default backpropagation training function is *trainbfg*. One might use *trainlm*, but it tends to proceed so rapidly that it does not necessarily do well in the Elman networks. The backpropagation weight/bias learning function default is *traingdm*, and the default performance function is the mean square error (MSE). When the network is created, each layer's weights and biases are initialized with the Nguyen-Widrow layer initialization method implemented by the function *"initnw"*. This information is obtained from the Neural Network Toolbox for Use with MATLAB User's Guide (Howard Demuth Mark Beale, 2002).

4.7.2 Training an Elman Networks for identification of the DO concentration process

Elman networks can be trained with either of two Matlab functions, *train* or *adapt*. When using the function *train* to train an Elman network the following occurs. At each epoch:

1 The entire input sequence is presented to the network, and its outputs are calculated and compared with the target sequence to generate an error sequence.
2 For each time step, the error is back propagated to find *gradients* of errors for each weight and bias. This *gradient* is actually an approximation since the contributions of weights and biases to errors via the delayed recurrent connection are ignored.
3 This gradient is then used to update the weights with the backpropagation training function chosen. The function *traingdx* is recommended.

For an Elman network to have the best chance for learning a problem of function approximation it has to have more hidden neurons in its hidden layer than are actually required for a solution by another method. While a solution may be available with fewer neurons, the Elman network is less able to find the most appropriate weights for hidden neurons since the error gradient is approximated. Therefore, having a fair number of neurons to begin with makes it more likely that the hidden neurons will start out dividing up the input space in useful ways. The function *"train"* trains an Elman network to generate a sequence of target vectors when it is presented with a given sequence of input vectors. The input vectors and target vectors are passed to train as vectors P and T. Train takes these vectors and the initial weights and biases of the network, trains the network using backpropagation with momentum and an adaptive learning rate, and returns new weights and biases. Because Elman networks are an extension of the two-layer sigmoid/linear architecture, they inherit the ability to fit any input/output function with a finite number of discontinuities.

Elman networks with different sizes of hidden layer were tested. The learning rate, training algorithms, training epochs, and set up of activation functions were all used to determine the most accurate mapping of the recurrent network to the normalized DO concentration trajectory of Figure 4.7. The following commands create recurrent network.

```
net=newelm(minmax(P),[5 1],{'tansig','purelin'},'trainlm');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.epochs=1000;
net.trainParam.goal=1e-8;
```

For the code given above, the network contains 5 neurons in its hidden layer. The hidden layer contains *tansig* activation functions and the output layer contains a pure linear function. The type of training used is also set here, and for the example illustrated, it is the Levenberg-Marquardt *(trainlm)* algorithm training function.

The backpropagation training algorithms, in which the weights are moved in the direction of the negative gradient, are described in chapter 3. Three different types of more complex backpropagation algorithms that increase the speed of convergence are used to determine the effectiveness of each in the identification of the normalized DO concentration trajectory. These algorithms are:

- Resilient Backpropagation training function *(trainrp)* as the default. The function combines adaptive learning rate with momentum training parameter.
- Scaled Conjugate Gradient *(trainscg)*, since it tends to perform well over a wide variety of problems, and also because it is the only conjugate gradient algorithm that requires no line search and is a very good general purpose training algorithm.
- Levenberg-Marquardt *(trainlm)* algorithm, the fastest convergence in function approximation problems, for networks that contain up to a few hundred weights. It is used if very accurate training is required.

4.8 Elman NN models developed

The method followed to determine the correct combination of the input variables for successful identification involved quite a laborious task and time for empirical testing of each set of the input variables and different training algorithms. However, different combinations of input variables to the network and specific training algorithm, in order to achieve the targeted DO concentration trajectory, were developed. These model structures are summarized in Table 4.16.

Table4.16: Input combinations to the Elman ANNs with DO concentration as the target.

Number of input variables	Model	Input description	Input symbol	output
1	Model 1	Normalized Airflow rate vector	I1_1	DO
2	Model 2	Normalized Airflow rate and Time vector	I1_1, Z	DO
3	Model 3	Normalized Airflow rate, Time and Normalized DO concentration	I1_1, Z and i2_2	DO
4	Model 4	Normalized Airflow rate, 3 delayed normalized airflow variables, Normalized DO, 3 delayed Normalized DO, Time, and 3 delayed Time variables	I1_1, I1_1(ko), I1_1(k1), i1_1(k2), Z, Z(mo), Z(m1), Z(m2), and i2_2, I2_2 (no), I2_2 (n1), I2_2 (n2),	DO

4.8.1 Elman network model1 with one input variable and *traingdx* algorithm.

Table 4.17 shows the results of the Elman model trained with the Variable Learning Rate Backpropagation algorithm *(traingdx)* with different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for one single input variable: the normalized input airflow rate (i1_1) as a function of time. The function *traingdx* is normally recommended for Elman networks. The responses (trajectory) of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.64 to 4.67.

Table4.17: Results for the Elman ANNs with i1_1as input variable and DO as the target –model1

Type of ANN(Network structure)	Activation functions& training algorithm(*traingdx*)	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	2	10000	0.05	0.00721507
NEWELM	Tansig, Purelin	3	10000	0.05	0.00721366
NEWELM	Tansig, Purelin	4	10000	0.05	0.00721311
NEWELM	Tansig, Purelin	5	10000	0.05	0.00721366
NEWELM	Tansig, Purelin	10	10000	0.05	0.00721343
NEWELM	Tansig, Purelin	15	10000	0.05	0.00721290
NEWELM	Tansig, Purelin	20	10000	0.05	0.00721887

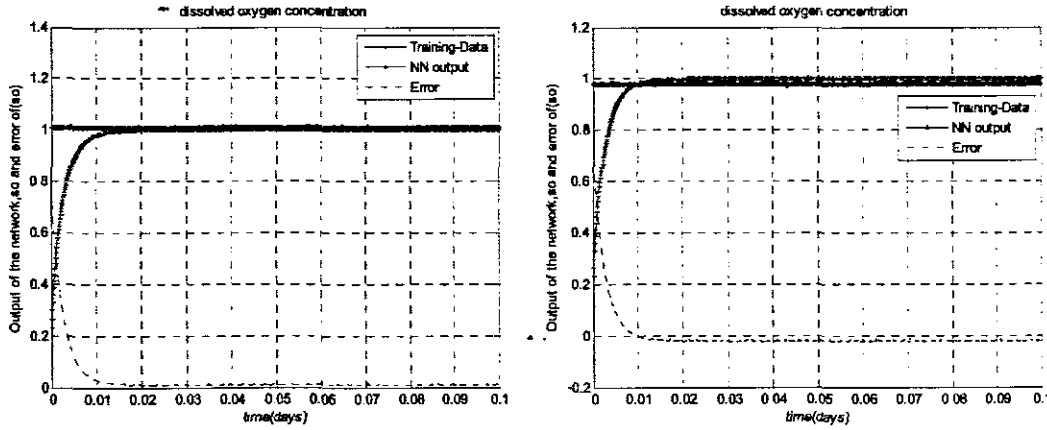


Figure 4.64: Elman networks response with 1 input, 1 hidden layer, 2 and 3 hidden layer neurons and 1 output (model 1).

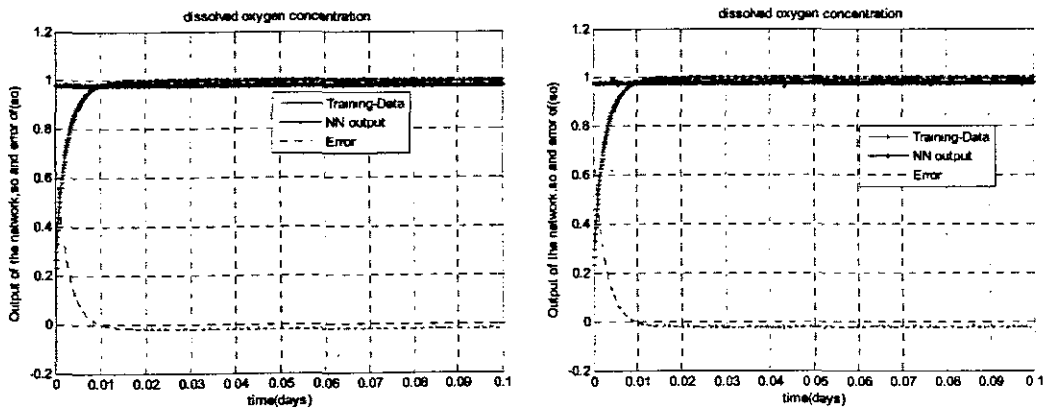


Figure 4.65: Elman networks response with 1 input, 1 hidden layer, 4 and 5 hidden layer neurons and 1 output (model 1).

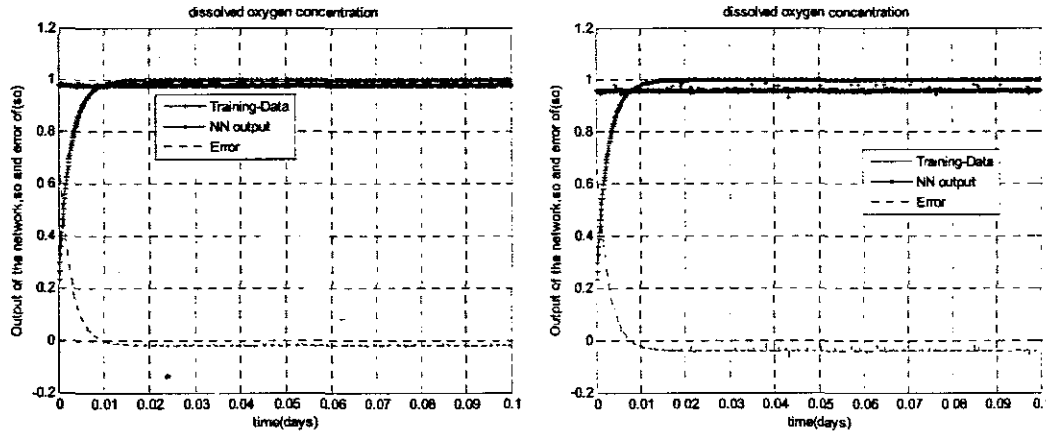


Figure 4.66: Elman networks response with 1 input, 1 hidden layer, 10 and 15 hidden layer neurons and 1 output (model 1).

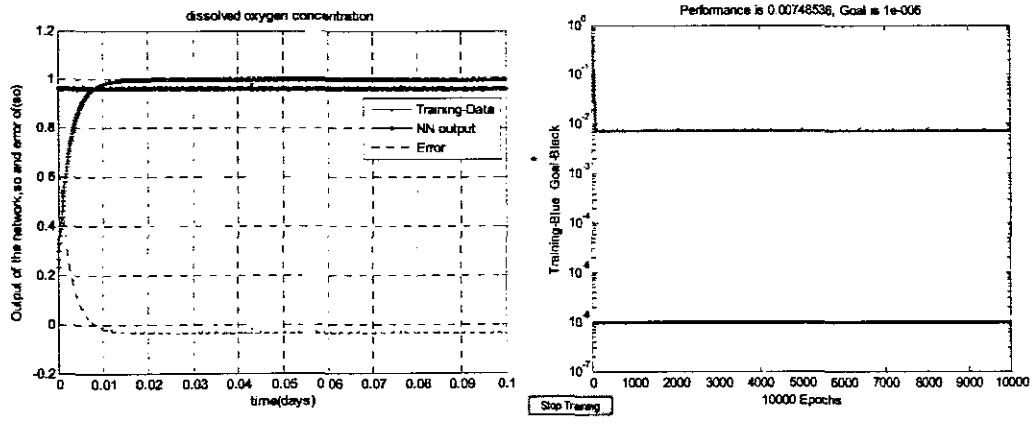


Figure 4.67: Elman networks response with 1 input, 1 hidden layer, 20 hidden layer *neurons and 1 output (model 1) MSE performance.*

Figure 4.67 shows the MSE achieved with the Elman network having 20 neurons in the hidden layer. The number of neurons was incrementally added from 2 to 20 for different models as depicted in Table 4.17. Analysis of the graphs plotted for the training data, NN output and the error as a function of time illustrated in Figures 4.63 - 4.67, and in addition to Table 4.17 have shown that:

- With only one input variable to the Elman network, the network does not map the required target when trained with the *traingdx* algorithm irrespective of the number of neurons in the hidden layer.
- The lowest MSE is obtainable after several numbers of training epochs.

4.8.2 Elman network model1 with one input vector and *trainscg* algorithm.

Table 4.18 shows the Scaled Conjugate Gradient *(trainscg)* training algorithm, used with different number of hidden neurons in the hidden layer, training epochs, learning rates and the mean squared error achieved for one single input variable: the normalized input airflow rate trajectory (i1_1) as a function of time. The responses

(trajectory) of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.68 to 4.71.

Table 4.18: Results for the Elman ANNs trained with *(trainscg)*,

Type of ANN(Network structure)	Activation functions& training algorithm*(trainscg)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	2	71	0.05	0.00721865
NEWELM	Tansig, Purelin	3	63	0.05	0.00720998
NEWELM	Tansig, Purelin	5	400	0.05	0.00720824
NEWELM	Tansig, Purelin	10	510	0.05	0.00720994
NEWELM	Tansig, Purelin	15	2550	0.05	0.00720993
NEWELM	Tansig, Purelin	20	2076	0.05	0.00721003

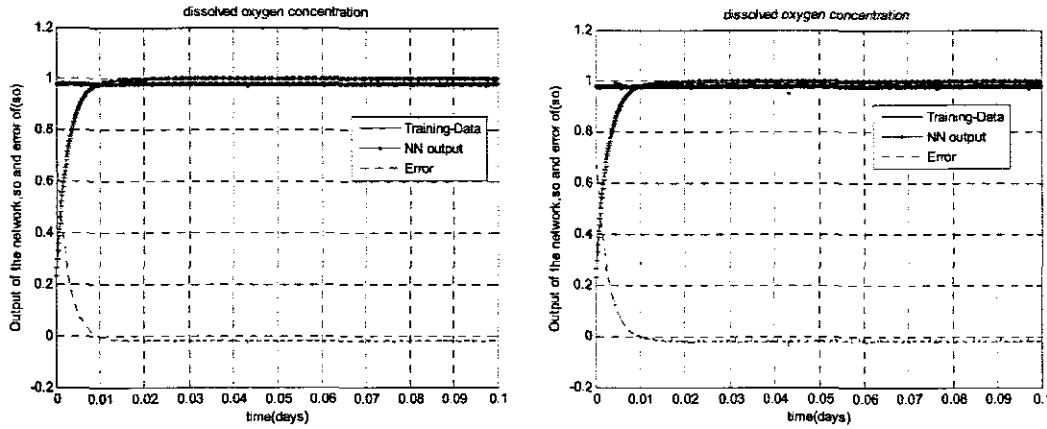


Figure 4.68: Elman networks response with 1 input, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 1).

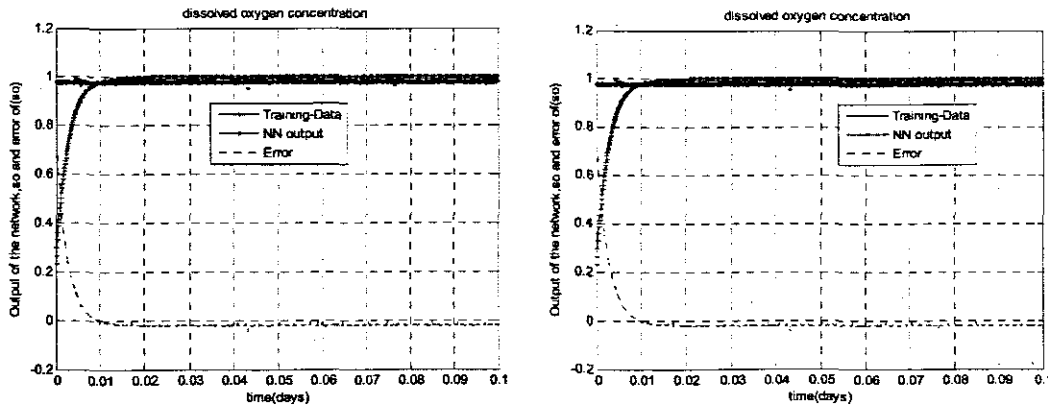


Figure 4.69: Elman networks response with 1 input, 1 hidden layer, 5 and 10 hidden neurons and 1 output (model 1).

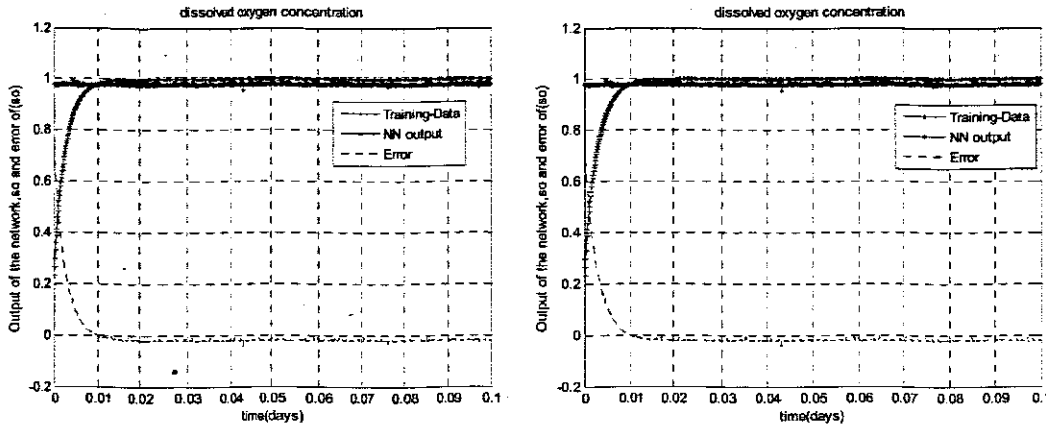


Figure 4.70: Elman networks response with 1 input, 1 hidden layer, 15 and 20 hidden neurons and 1 output (model 1).

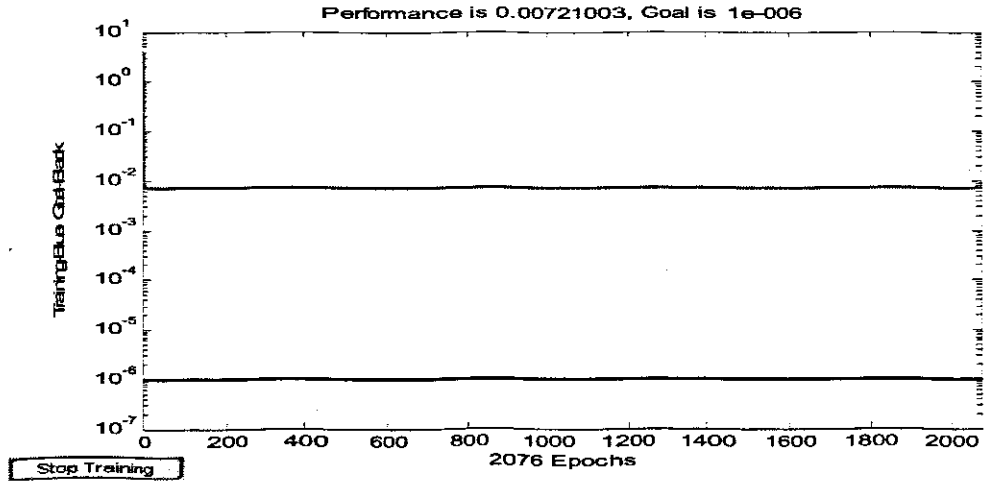


Figure 4.71: MSE performance with 1 input, 1 hidden layer, 20 hidden neurons and 1 output (model 1).

Figure 4.71 shows the MSE of 0.00721003 achieved with the Elman network having 20 neurons in the hidden layer. The number of neurons was incrementally added from 2 to 20 as depicted in Table 4.18. Analysis of the graphs plotted for the training data, NN output and the error as a function of time illustrated in Figures 4.68 - 4.71, and analysis in addition to Table 4.18 for the performance index (MSE) have shown that:

- With only one input variable of the airflow rate to the Elman network, the network does not map the required target when trained with the *trainscg* algorithm irrespective of the number of neurons in the hidden layer.
- The convergence to the lowest MSE is obtainable after only a few numbers of iterations as compared to *traingdx* algorithm.

4.8.3 Elman network model1 with one input variable and *trainlm* algorithm.

Table 4.19 shows the Levenberg-Marquardt algorithm function *(trainlm)*, used with different number of hidden layers, training epochs, learning rates and the mean squared error achieved for one single input variable: the normalized input airflow rate **(i1_1)** as a function of time while the output is the normalized DO concentration **(i2_2)**. The responses of training data, NN output and the mse between the input and the target are illustrated in the subsequent Figures 4.72 to 4.74.

Table 4.19 Recurrent ANN with Levenberg-Marquardt training algorithm *(trainlm)* – model1

Type of ANN(Network structure)	Activation functions& training algorithm*(trainlm)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	2	1118	0.05	0.00714228
NEWELM	Tansig, Purelin	3	10000	0.05	0.00691485
NEWELM	Tansig, Purelin	5	10000	0.05	0.00693535
NEWELM	Tansig, Purelin	10	10000	0.05	0.00714415
NEWELM	Tansig, Purelin	15	10000	0.05	0.00720000
NEWELM	Tansig, Purelin	20	401	0.05	0.00720913

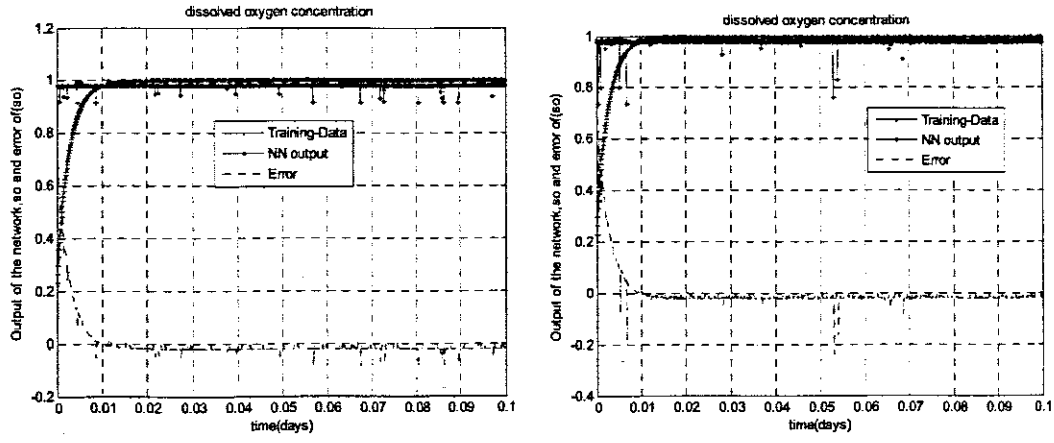


Figure 4.72: Elman networks response with 1 input, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 1).

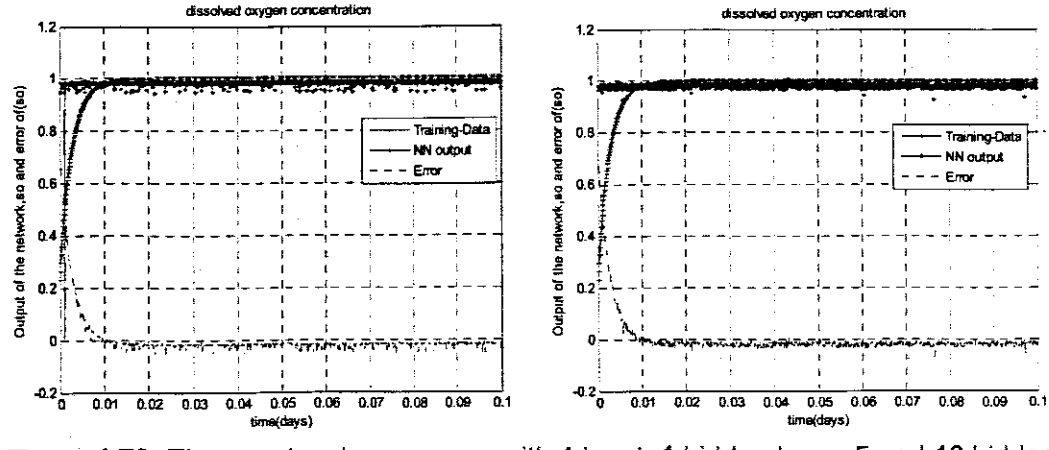


Figure 4.73: Elman networks response with 1 input, 1 hidden layer, 5 and 10 hidden neurons and 1 output (model 1).

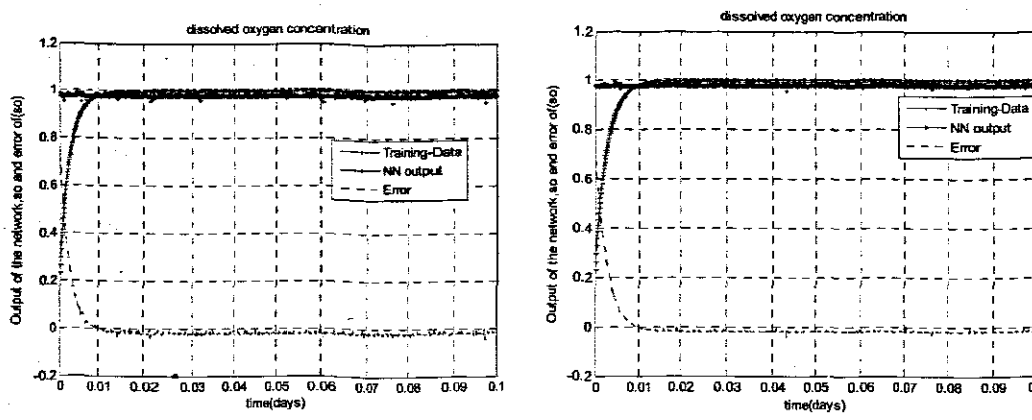


Figure 4.74: Elman networks response with 1 input, 1 hidden layer, 15 and 20 hidden neurons and 1 output (model 1).

Analysis of the graphs plotted for the training data, NN output and the error as a function of time illustrated in figures 4.72 - 4.74, and in addition to Table 4.19 for the performance index (MSE) have shown that:

- With only one input vector of the airflow rate to the Elman network, the network does not map the required target when trained with the *trainlm* algorithm irrespective of the number of neurons in the hidden layer.
- The convergence to the lowest MSE is obtainable after several numbers of iterations.

Thus, the general conclusion made from the three types of the gradient descent training algorithms used to train the Elman network for a single input variable is that the convergence to MSE is approximately the same. However, the network trained with *trainscg* converged much faster than the other two. Poor function approximation was achieved and hence the network did not identify the DO concentration model trajectory.

The next network tested was with two inputs composed of the variable of the normalized input flow rate trajectory and the variable of time function. The procedure adopted was the same as that of one input vector. These are explained in the proceeding sections.

4.8.4 Elman network with two inputs vectors and trained with traingdx algorithms

Table 4.20 shows the Variable Learning Rate Backpropagation training function *(traingdx)* used with different number of hidden neurons in the hidden layer, training epochs, learning rates and the mean squared error achieved for a vector of two input variables: the *normalized airflow rate trajectory (i1_1) and a time function Z.* The targeted output is the normalized DO concentration trajectory (i2_2). The responses

(trajectories) of training data, NN output and the MSE between the input and the target are illustrated in the subsequent Figures 4.75 to 4.78.

Table 4.20: Recurrent ANN for model2 with Adaptive backpropagation training algorithm *traingdx* - model2

Type of ANN(Network structure)	Activation functions& training algorithm*(traingdx)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	2	10000	0.05	0.00440213
NEWELM	Tansig, Purelin	3	10000	0.05	0.00454346
NEWELM	Tansig, Purelin	4	10000	0.05	0.00323276
NEWELM	Tansig, Purelin	5	10000	0.05	0.00332548
NEWELM	Tansig, Purelin	10	10000	0.05	0.00309161
NEWELM	Tansig, Purelin	15	10000	0.05	0.00346776
NEWELM	Tansig, Purelin	20	10000	0.05	0.00144195

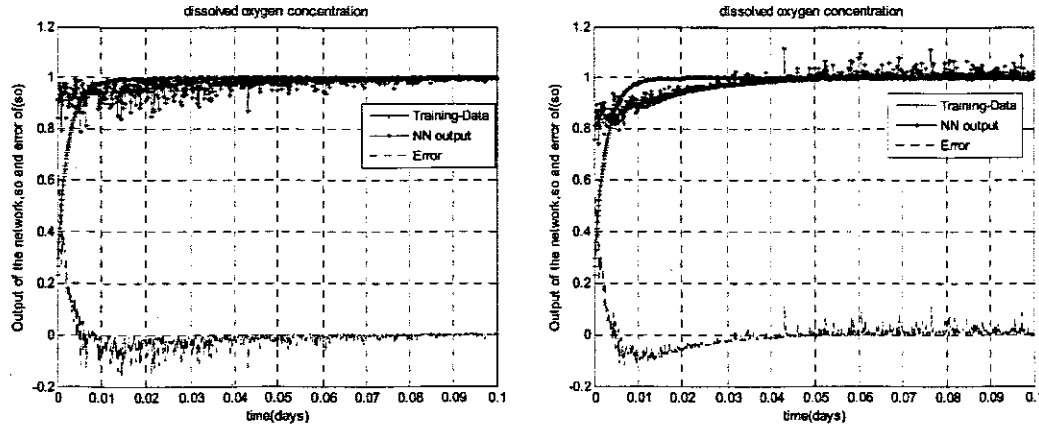


Figure 4.75: Elman networks response with 2 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 2).

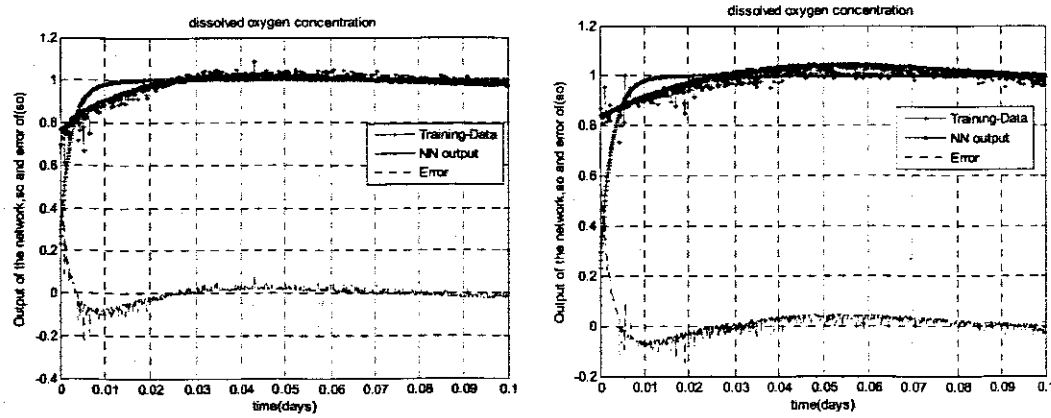


Figure 4.76: Elman networks response with 2 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 2).

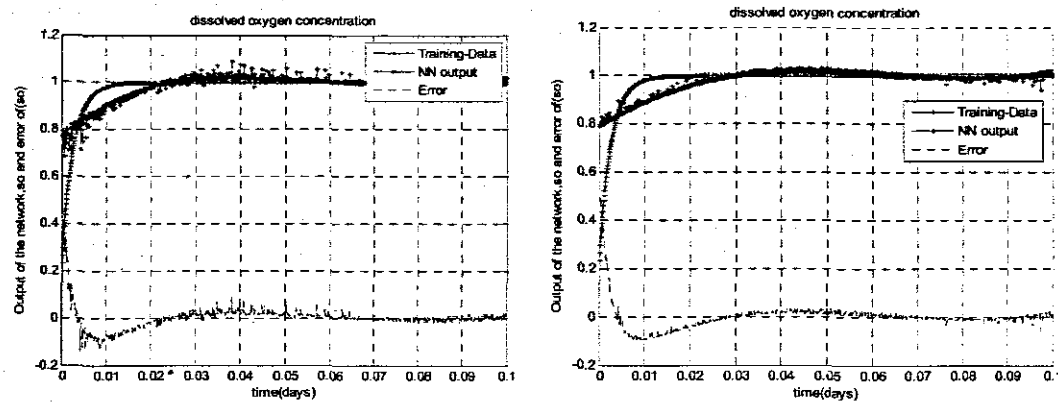


Figure 4.77: Elman networks response with 2 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 2).

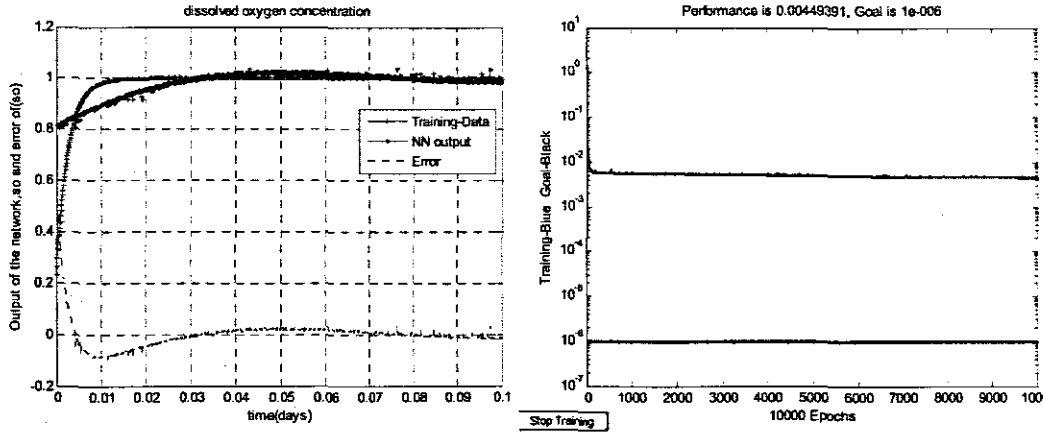


Figure 4.78: Elman networks response with 2 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 2) and Performance MSE achieved.

Analysis of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in figures 4.75 - 4.78, and analysis of Table 4.20 have shown that:

- There is a significant improvement on the network performance since the convergence MSE is improved. However the network still fails to map the required target when trained with *traingdx*.

4.8.5 Elman network model2 with two input variables trained with *trainscg* algorithms

Table 4.21 shows the Scaled Conjugate Gradient *(trainscg),* used with different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for a vector of two input variables: the normalized airflow rate (i1_1) and a time function Z. The targeted output is the normalized DO concentration (i2_2). The responses (trajectories) of training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.79 to 4.82.

Table 4.21: Results for the Elman ANNs trained with *(trainscg)* - model2

Type of ANN(Network structure)	Activation functions& training algorithm*(trainscg)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	2	3788	0.05	5.55371e-006
NEWELM	Tansig, Purelin	3	2953	0.05	2.43055e-006
NEWELM	Tansig, Purelin	4	6319	0.05	2.24714e-005
NEWELM	Tansig, Purelin	5	10000	0.05	0.000101717
NEWELM	Tansig, Purelin	10	10000	0.05	0.000704107
NEWELM	Tansig, Purelin	15	10000	0.05	0.000968941
NEWELM	Tansig, Purelin	20	10000	0.05	0.00163286

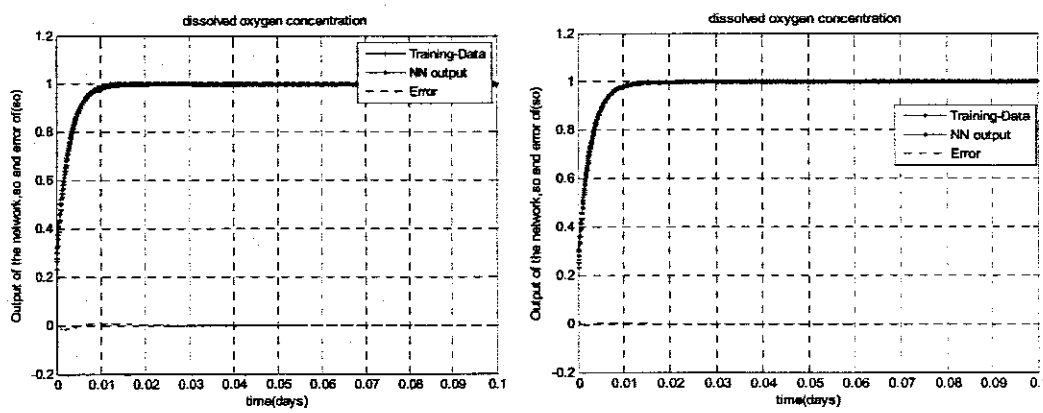


Figure 4.79: Elman networks response with 2 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 2).

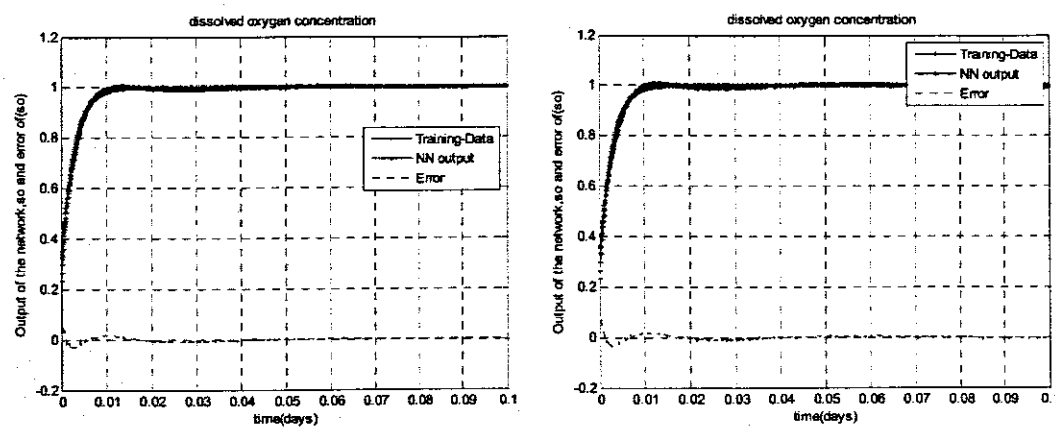


Figure 4.80: Elman networks response with 2 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 2).

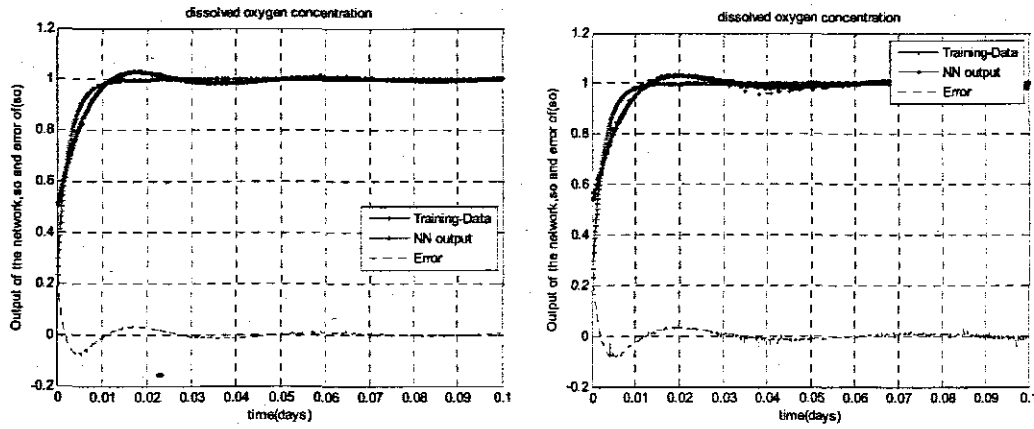


Figure 4.81: Elman networks response with 2 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 2).

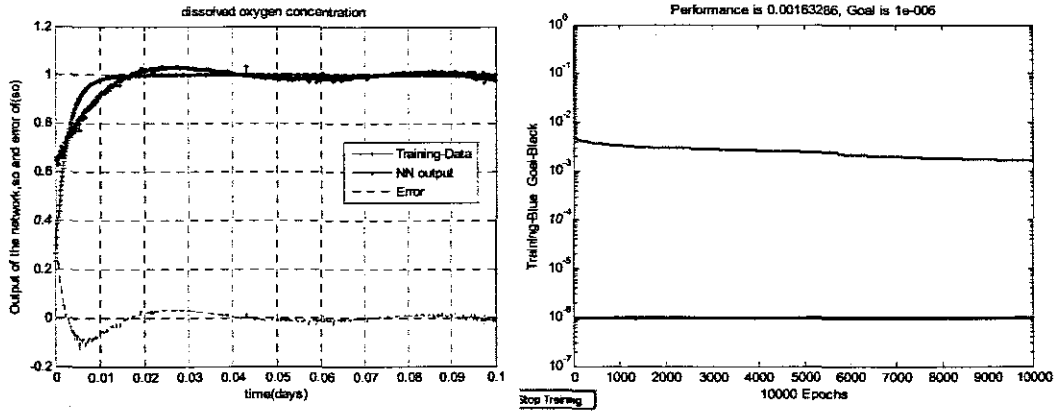


Figure 4.82: Elman networks response with 2 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 2) and Performance MSE achieved.

Analysis of this network from the perspective of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.79 - 4.82, and analysis in addition to Table 4.21 which details the performance index (MSE) have shown that:

- a network with 3, 4 and 5 neurons in the hidden layer and which is trained with *trainscg* algorithm is a good identifier of the DO concentration model
- a small convergence time of a low MSE is also achieved with this algorithm.
- the number of epochs for convergence is drastically reduced.

4.8.6 Elman network model2 with two input variables trained with *trainlm* algorithms

Table 4.22 shows the results of the Elman recurrent neural networks trained with the Levenberg-Marquardt algorithm *("trainlm")*. Different number of hidden neurons, training epochs, learning rates and the mean squared error achieved for a vector of two input variables: the normalized input airflow rate trajectory **(i1_1)** and a time function **Z**. The targeted output is the normalized DO concentration trajectory **(i2_2)**.

The responses (trajectories) of the training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.83 to 4.86.

Table 4.22: Results for the Elman ANN model2 trained with *(trainlm)*-model3

Type of ANN(Network structure)	Activation functions& training algorithm*(trainlm)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	2	1504	0.05	9.971324e-007
NEWELM	Tansig, Purelin	3	306	0.05	9.96699e-007
NEWELM	Tansig, Purelin	4	76	0.05	9.89386e-007
NEWELM	Tansig, Purelin	5	292	0.05	9.94264e-007
NEWELM	Tansig, Purelin	10	287	0.05	9.927e-007
NEWELM	Tansig, Purelin	15	1131	0.05	9.96884e-007
NEWELM	Tansig, Purelin	20	2343	0.05	9.96201e-007

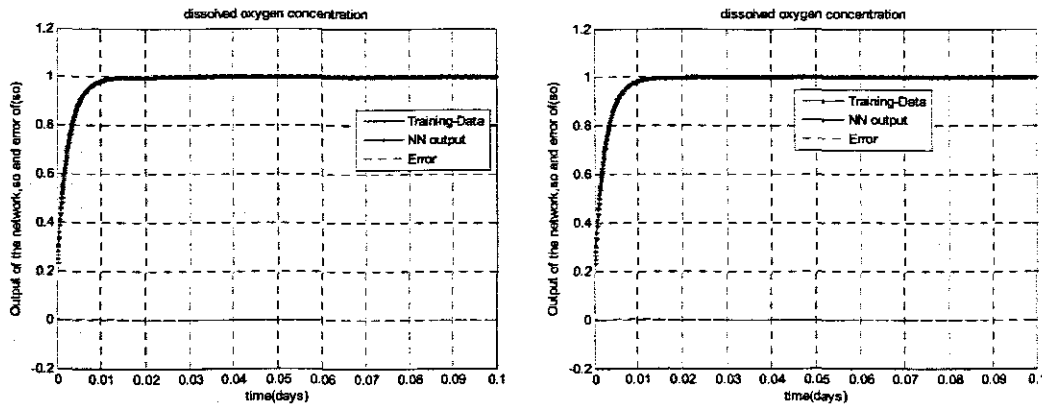


Figure 4.83: Elman networks response with 2 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 2).

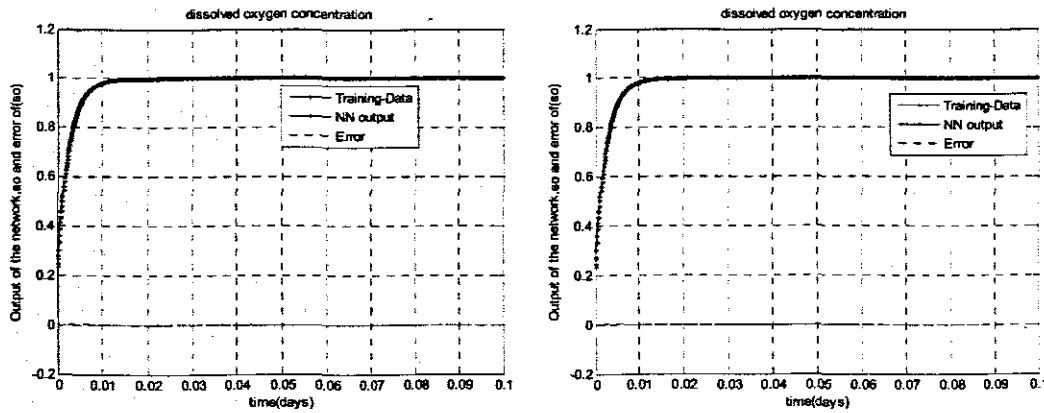


Figure 4.84: Elman networks response with 2 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 2).

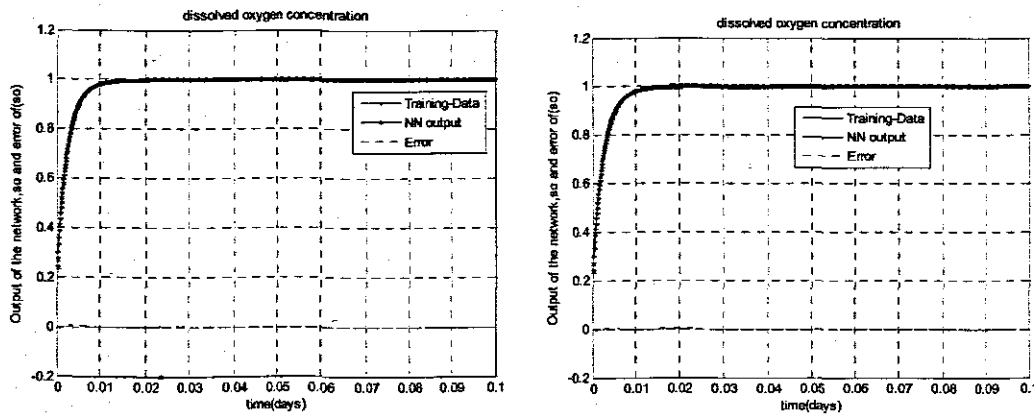


Figure 4.85: Elman networks response with 2 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 2).

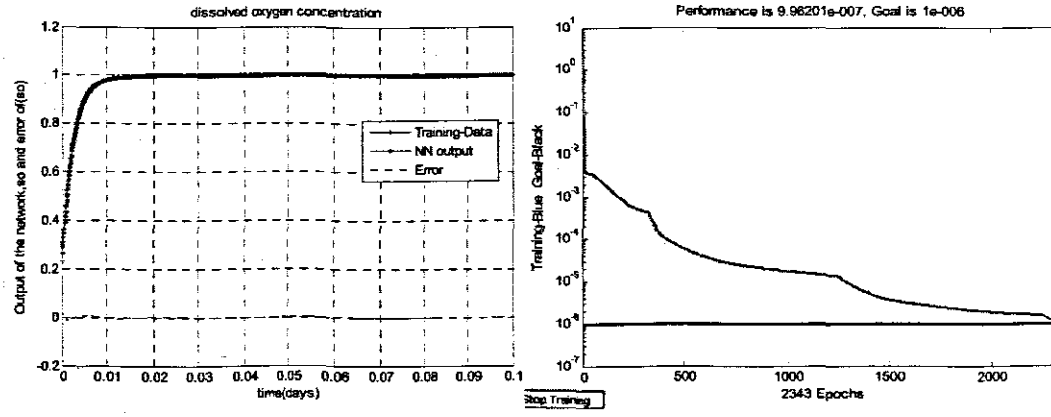


Figure 4.86: Elman networks response with 2 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 2) and Performance MSE achieved.

Analysis of this network from the perspective of the graphs plotted for the trajectories of the training data, NN output and the error as a function of time as illustrated in Figures 4.83 - 4.86, and in addition to Table 4.22 which details the performance indices (MSE) have shown that:

- a network with 3 neurons in the hidden layer and which is trained with *trainlm* algorithm is just enough to identify the DO concentration model trajectory.
- a small convergence time to a low MSE of 9.96699e-007 is also achieved after only 306 number of epochs for a network of 3 neurons in the hidden layer.

Thus, the general conclusion made from the three types of the gradient descent training algorithms used to train the Elman network model 2 for a two input variable is that the variable learning rate backpropagation algorithm *(traingdx)* produced moderately low value of the MSE but exhibited poor mapping of the target. The network trained with the scaled conjugate gradient algorithm *(trainscg)* produced much lower MSE after a fewer number of training epochs, converged much faster for a small

number oh hidden layer nodes, but exhibited poor mapping with increased number of hidden nodes. Levenberg-Marquardt algorithm *(trainlm)* was the fastest in convergence with only a few numbers of epochs, very low MSE, better mapping of the target than the other two.

The next network tested was with three input variables composed of the variable of the normalized input flow rate trajectory and the variable of time function and that of normalized DO process. The procedure adopted was the same as that of one input variable. The results are explained in the proceeding sections.

4.8.7 Elman network model3 with three input variables and *traingdx* training algorithm

Table 4.23 shows the results of the Elman recurrent neural networks trained with the Variable Learning Rate Backpropagation training function *(traingdx).* Different number of hidden layers, training epochs, learning rates and the mean squared error achieved for three input variables: the normalized input airflow rate trajectory (i1_1), a time function Z, and the normalized DO concentration trajectory (i2_2). The targeted output is the normalized DO concentration (i2_2). The responses (trajectories) of the training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.87 to 4.90.

Table 4.23: Results for the three input vector to Elman ANNs and trained with *traingdx* – model3

Type of ANN(Network structure)	Activation functions& training algorithm*(traingdx)*	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	3	10000	0.05	6.39714e-005
NEWELM	Tansig, Purelin	4	10000	0.05	1.4187e-005
NEWELM	Tansig, Purelin	5	10000	0.05	5.012e-005
NEWELM	Tansig, Purelin	10	10000	0.05	3.12111e-005
NEWELM	Tansig, Purelin	15	10000	0.05	1.10466e-005
NEWELM	Tansig, Purelin	20	10000	0.05	1.44177e-005

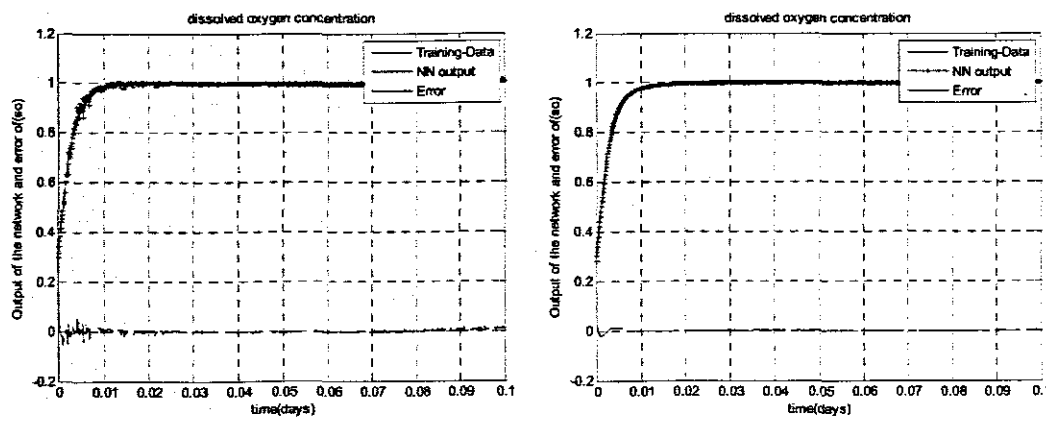


Figure 4.87: Elman networks response with 3 inputs, 1 hidden layer, 3 and 4 hidden neurons and 1 output (model 3).

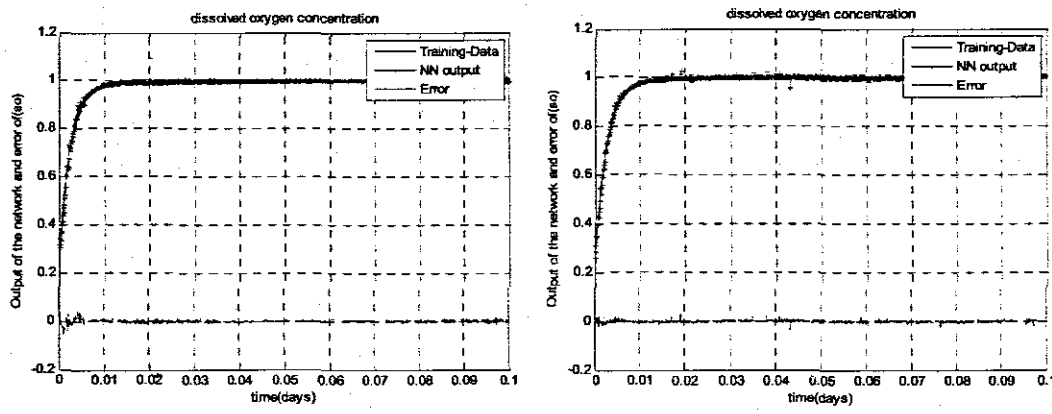


Figure 4.88: Elman networks response with 3 inputs, 1 hidden layer, 5 and 10 hidden neurons and 1 output (model 3).

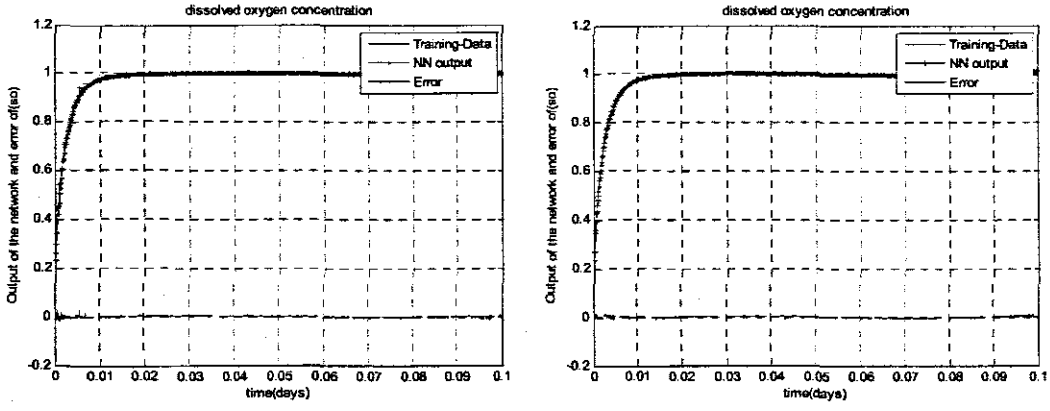


Figure 4.89: Elman networks response with 3 inputs, 1 hidden layer, 15 and 20 hidden neurons and 1 output (model 3).

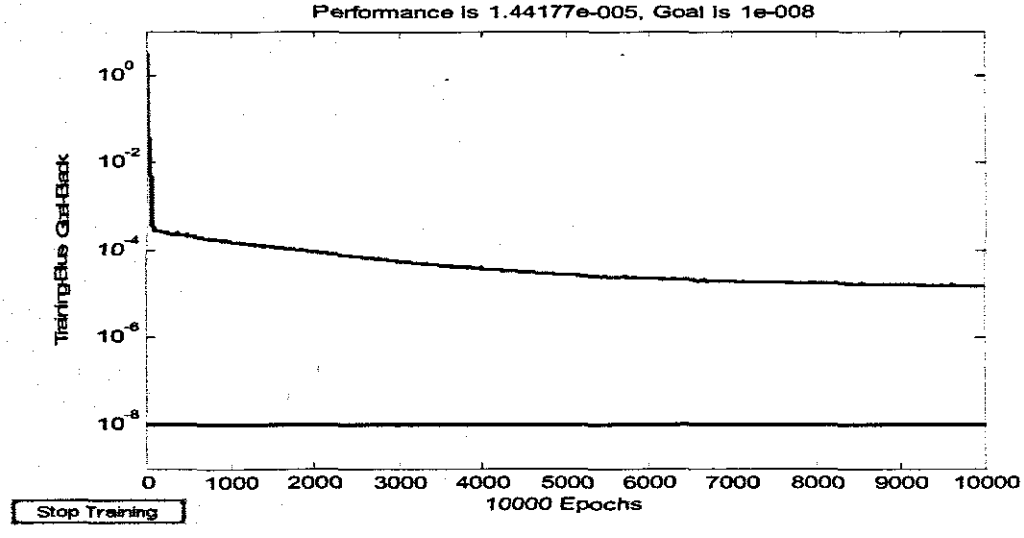


Figure 4.90: Performance MSE with 3 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 3).

Analysis of this network from the perspective of the graphs plotted for the training data, NN output and the error as functions of time as illustrated in Figures 4.87 - 4.90, and analysis in addition to Table 4.23 which details the performance index (MSE) have shown that:

- a network with 3 neurons in the hidden layer and which is trained with *trainlm* algorithm is just enough to identify the DO concentration model trajectory and that
- a small convergence time to a low mse of 1.44177e-005 is also achieved. However, the model requires a very large number of training epochs in order to achieve the minimum convergence.

4.8.8 Elman network model3 with three input variables and *trainscg* training algorithm

Table 4.24 shows the Scaled Conjugate Gradient training algorithm *(trainscg)*, used with different number of hidden nodes in the hidden layer, different number of training epochs, learning rates and the mean squared error achieved for three input variables: the normalized input airflow rate trajectory (i1_1), a time function Z, and the normalized DO concentration trajectory (i2_2). The targeted output is the normalized DO concentration trajectory (i2_2). The responses (trajectories) of the training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.91 to 4.94.

Table 4.24: Results for the three input variables to Elman ANNs and trained with *trainscg*

Type of ANN(Network structure)	Activation functions& training algorithm(*trainscg*)	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	3	2931	0.05	2.65284e-007
NEWELM	Tansig, Purelin	4	4151	0.05	3.14788e-006
NEWELM	Tansig, Purelin	5	3490	0.05	1.07039e-006
NEWELM	Tansig, Purelin	10	1498	0.05	2.39381e-007
NEWELM	Tansig, Purelin	15	3040	0.05	1.76323e-007
NEWELM	Tansig, Purelin	20	3208	0.05	1.06008e-007

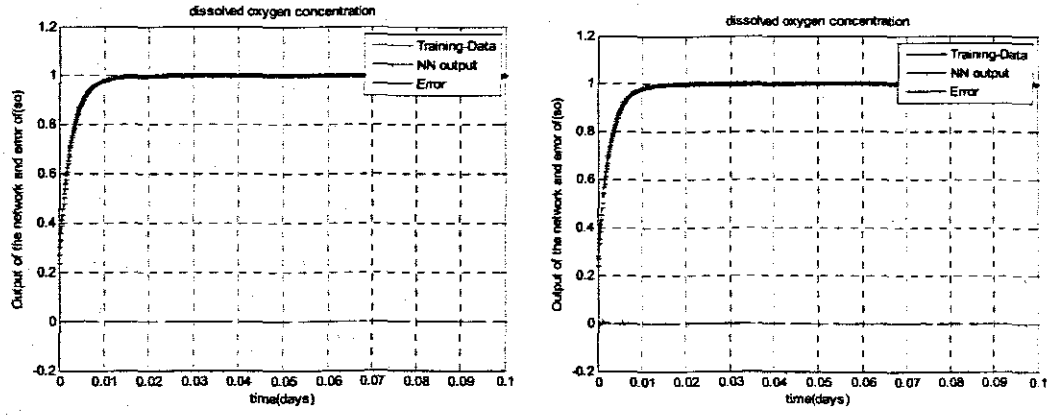


Figure 4.91: Elman networks response with 3 inputs, 1 hidden layer, 3 and 4 hidden neurons and 1 output (model 3).

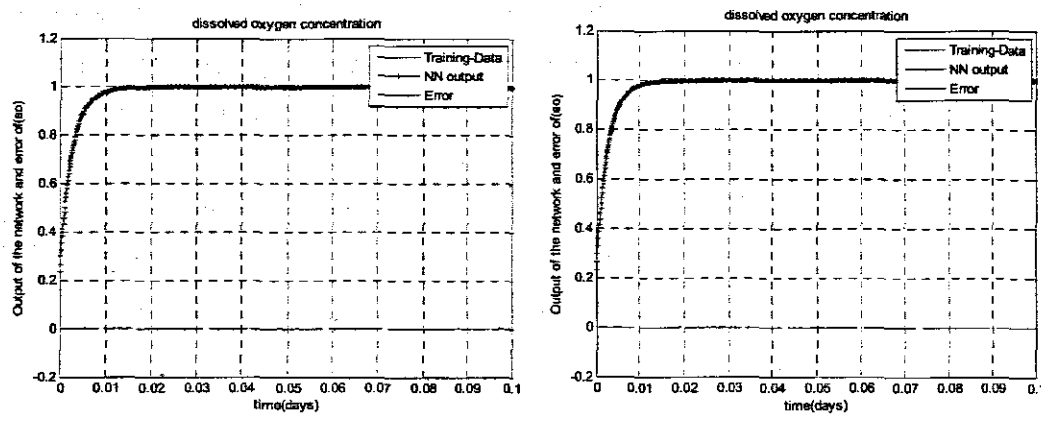


Figure 4.92: Elman networks response with 3 inputs, 1 hidden layer, 5 and 10 hidden neurons and 1 output (model 3).

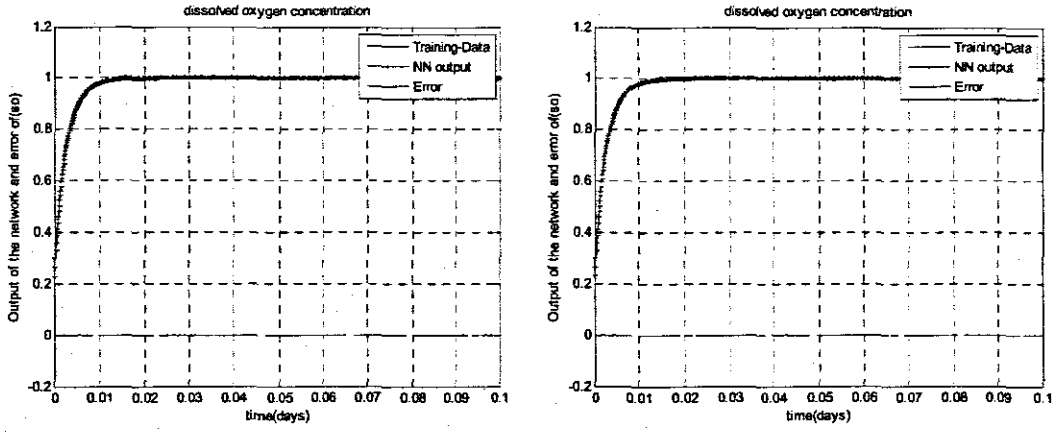


Figure 4.93: Elman networks response with 3 inputs, 1 hidden layer, 15 and 20 hidden neurons and 1 output (model 3).

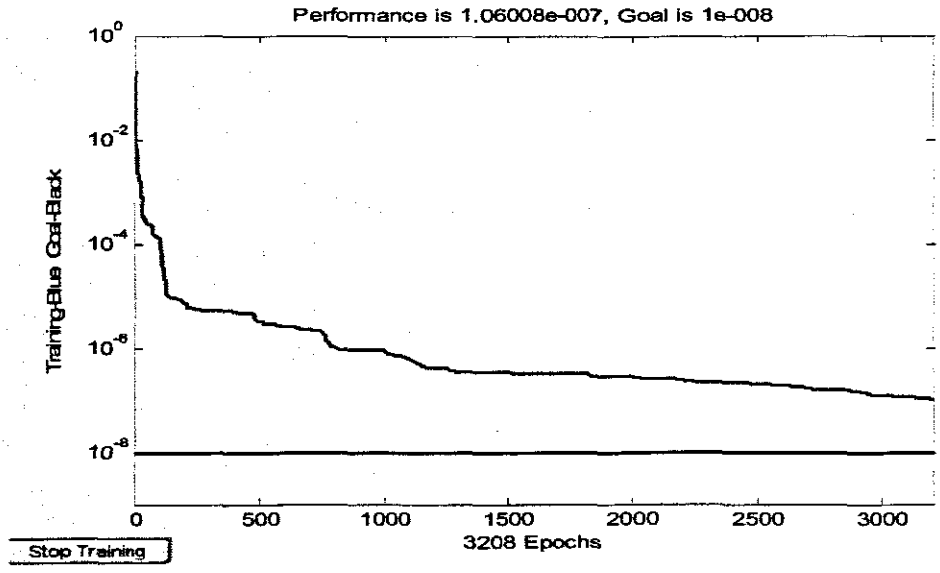


Figure 4.94: Performance mse with 3 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 3).

Analysis of this network from the perspective of the graphs plotted for the training data, NN output and the error as functions of time as illustrated in Figures 4.91 - 4.94, and analysis in addition to Table 4.23 which details the performance indices (MSE) have shown that:

- a network with 3 neurons in the hidden layer and which is trained using trainscg algorithm is enough to identify the DO concentration model trajectory and
- a small convergence time to a low MSE of 1.06008e-007 is also achieved after only 3208 training epochs. This is a considerable improvement to the network trained with *traingdx* algorithm which required a large number of training epochs in order to achieve the minimum convergence for the same model.

4.8.9 Elman network model3 with three input variables and *trainlm* training algorithm)

Table 4.25 shows the Levenberg-Marquardt training algorithm *("trainlm")*, used with different number of hidden layers, varying numbers of training epochs, learning rates and the mean squared error achieved for three input variables: the normalized input airflow rate trajectory (i1_1), a time function Z, and the normalized DO concentration trajectory (i2_2). The targeted output is the normalized DO concentration (i2_2). The *responses (trajectories) of the training data, NN output and the error between the input* and the target are illustrated in the subsequent Figures 4.95 to 4.98.

Table 4.25: Results for the three input vector to Elman ANNs and trained with *trainlm*

Type of ANN(Network structure)	Activation functions& training algorithm(*trainlm*)	No of hidden nodes	Training Epochs	Learning rates	MSE
NEWELM	Tansig, Purelin	3	227	0.05	9.95751e-009
NEWELM	Tansig, Purelin	4	260	0.05	9.92276e-009
NEWELM	Tansig, Purelin	5	227	0.05	9.97313e-009
NEWELM	Tansig, Purelin	10	9	0.05	8.03592e-009
NEWELM	Tansig, Purelin	15	7	0.05	2.13548e-009
NEWELM	Tansig, Purelin	20	17	0.05	9.45116e-009

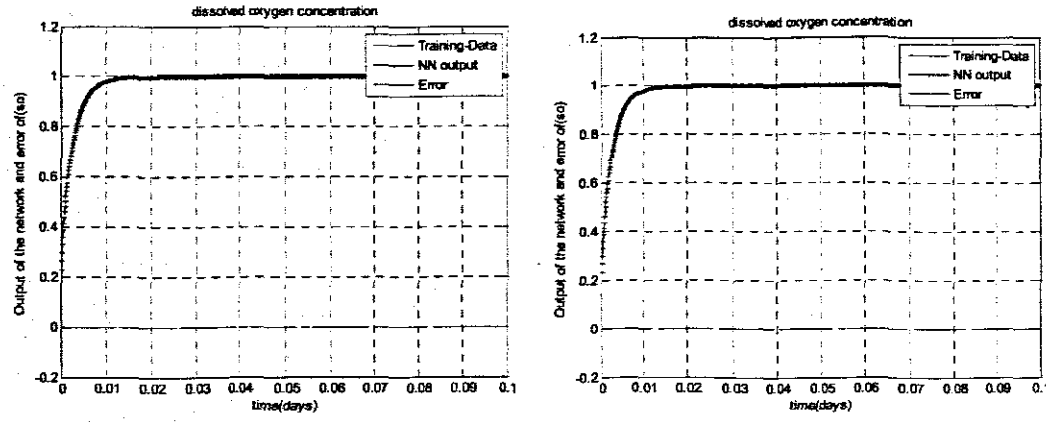


Figure 4.95: Elman networks response with 3 inputs, 1 hidden layer, 3 and 4 hidden neurons and 1 output (model 3).

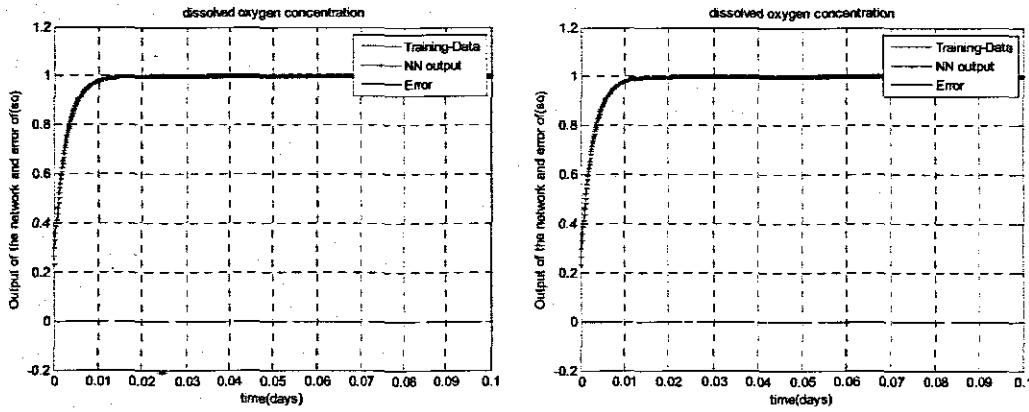


Figure 4.96: Elman networks response with 3 inputs, 1 hidden layer, 5 and 10 hidden neurons and 1 output (model 3).

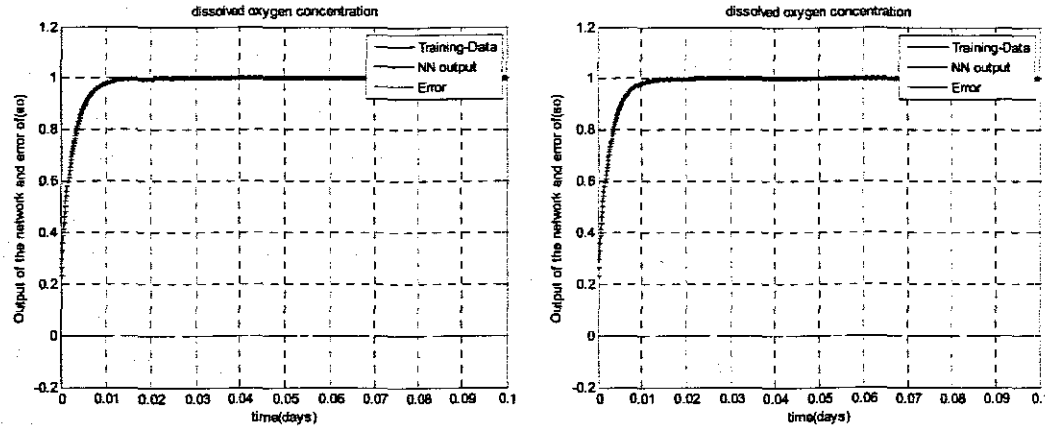


Figure 4.97: Elman networks response with 3 inputs, 1 hidden layer, 15 and 20 hidden neurons and 1 output (model 3).

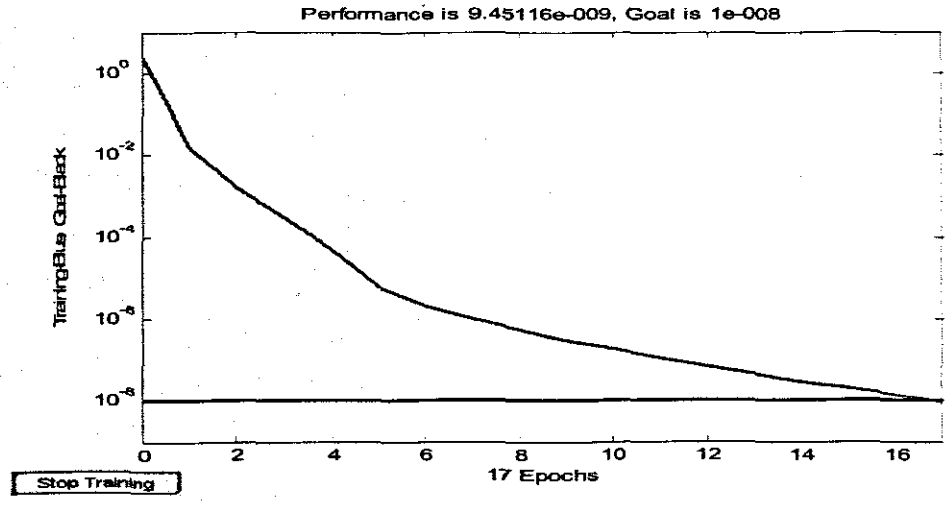


Figure 4.98: Performance mse with 3 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 3).

Analysis of this network from the perspective of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.95 - 4.98, and

analysis in addition to Table 4.25 which details the performance indices (MSE) have shown that:

- a network with 3 neurons in the hidden layer and which is trained with *trainlm* algorithm is just enough to identify the DO concentration model.
- *a small convergence time to a low MSE of 9.45116 e-009 is also achieved after* only 88 training epochs.
- This justifies that this training algorithm is much faster compared to the other two tested which required a considerable number of training epochs in order to achieve the minimum convergence for the same model.

Thus, the general conclusion made from the three types of the gradient descent training algorithms used to train the Elman network model 3 for three input variables is that the variable learning rate backpropagation algorithm *(traingdx)* produced moderately low value of the MSE and exhibited not so good mapping of the target. The network trained with the scaled conjugate gradient algorithm *(trainscg)* produced lower MSE after a fewer number of training epochs, converged much faster for a small number of hidden layer nodes, exhibited good mapping of the target. Levenberg-Marquardt algorithm *(trainlm)* was the fastest in convergence with only a few numbers of epochs, very low MSE, better mapping of the target than the other two.

The next network tested was twelve input variables composed of the variable of the normalized input flow rate trajectory with its two time delay variables and the variable of time function with its two time delays and that of normalized DO process with its two time delays as variables. The procedure adopted was the same as that of one input variable. The results are explained in the proceeding sections.

4.8.10 Elman network model4 with 12 Input variables and *traingdx* algorithm

Table 4.26 shows the results of the Elman recurrent neural networks when trained with the adaptive algorithm function *"traingdx".* The function *traingdx* combines adaptive learning rate with momentum training. It has the momentum coefficient as an additional training parameter. Different number of hidden neurons in the hidden layer, different number of training epochs, learning rates and the mean squared error achieved for twelve input variables consisting of the normalized input airflow rate trajectory (i1_1), three delayed values of the normalized input airflow rates, time, three delayed values of time function, normalized output trajectory (i2_2) and three delayed values of the normalized outputs forming the input vector P = [Z m0 m1 m2 i1_1 k0 k1 k2 i2_2 n0 n1 n2]' used, and an output vector T= [i2_2]' are also tabled. The targeted output is the normalized DO concentration (i2_2). The responses (trajectories) of the training

data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.99 to 4.102.

Table 4.26: Results for the twelve input vector to Elman ANNs and trained with *traingdx* algorithm-model4

Type of ANN(Network structure)	Activation functions& training algorithm*(traingdx)*	No of hidden (layers)	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	10000	0.05	0.000128831
NEWFF	Tansig, Purelin	3	10000	0.05	0.000228408
NEWFF	Tansig, Purelin	4	10000	0.05	2.57074e-005
NEWFF	Tansig, Purelin	5	10000	0.05	9.03985e-005
NEWFF	Tansig, Purelin	10	10000	0.05	1.98671e-005
NEWFF	Tansig , Purelin	15	10000	0.05	2.60324e-005
NEWFF	Tansig , Purelin	20	10000	0.05	1.54432e-005

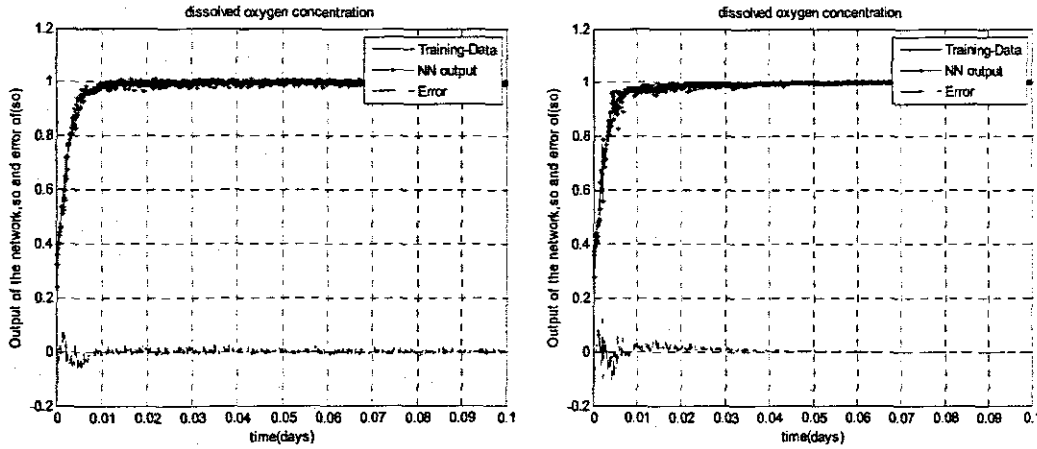


Figure 4.99: Elman networks response with **12** inputs, **1** hidden layer, **2** and **3** hidden neurons and **1** output (model **4**).

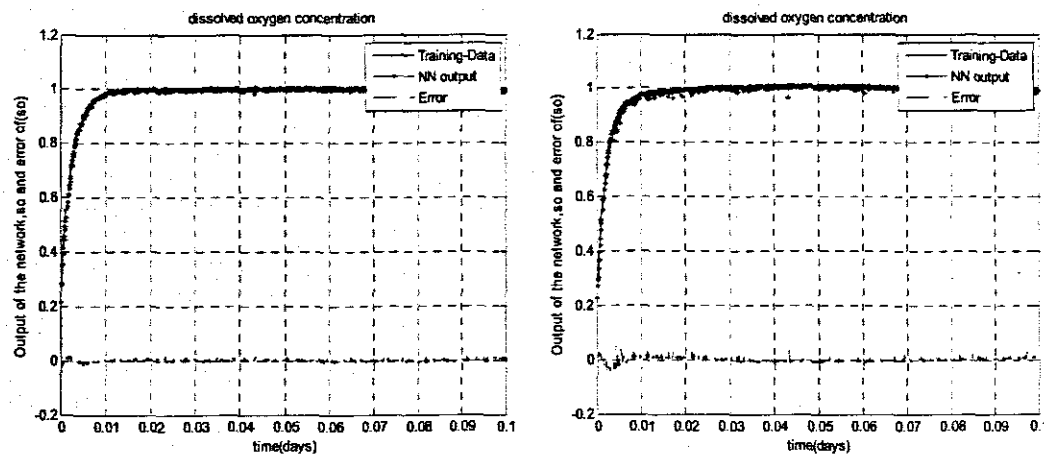


Figure 4.100: Elman networks response with **12** inputs, **1** hidden layer, **4** and **5** hidden *neurons* and **1** output (model **4**).

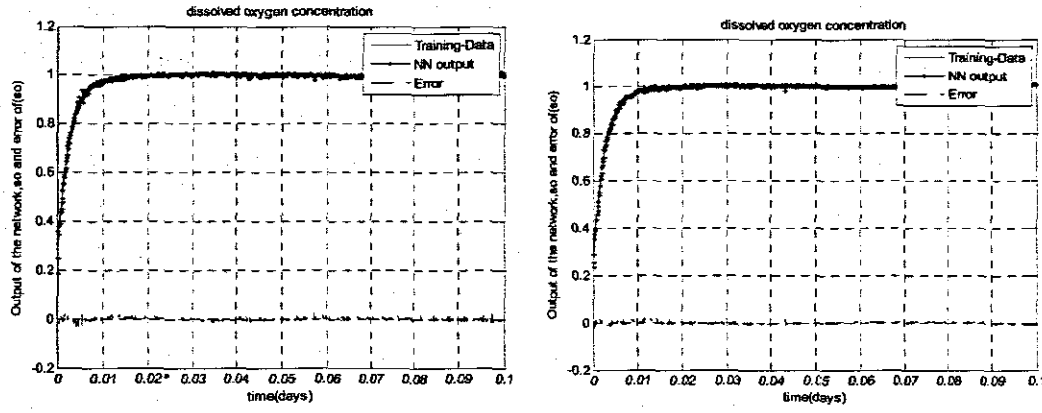


Figure 4.101: Elman networks response with 12 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 4).

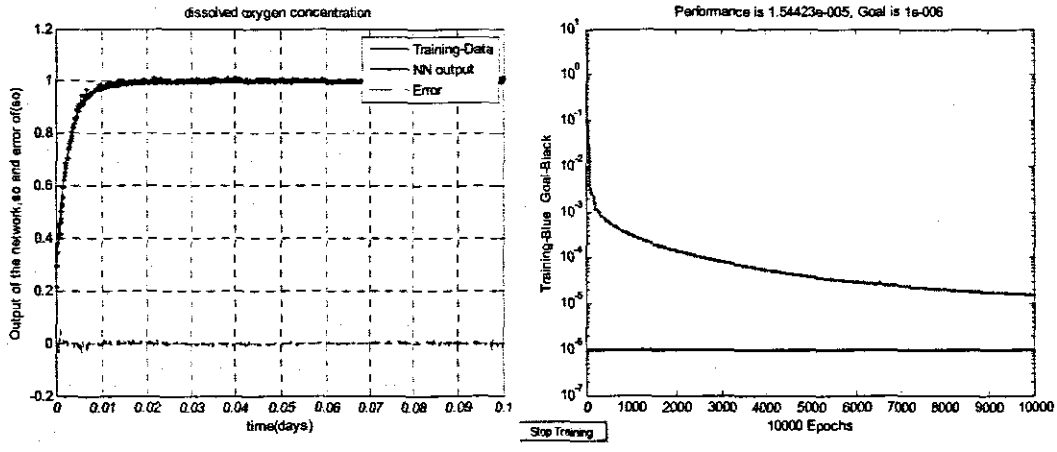


Figure 4.102: Elman networks response with 12 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 4) and Performance MSE achieved.

Analysis of this network from the perspective of the graphs plotted for the training data, NN output and the error as functions of time as illustrated in Figures 4.98 - 4.101, and analysis in addition to Table 4.26 which details the performance indices (MSE) have shown that:

- A network with 4 neurons in the hidden layer and which is trained with traingdx algorithm is just enough to identify the DO concentration model.
- A small convergence error as low mse of 1.57326e-005 also achieved.
- However, this algorithm, takes long time and a considerable large number of training epochs in order to achieve the minimum convergence the DO target trajectory.

4.8.11 Elman network model4 with 12 Input variables and *trainscg* algorithm

Table 4.27 shows the results of the Elman recurrent neural networks trained with the scaled conjugate gradient algorithm *"trainscg"* see (Moller, 1993) for a detailed

explanation of the algorithm. The function *trainscg* is a numerical optimization technique which requires that the network response to all training inputs to be computed through a line search in the shortest time possible. Different number of hidden neurons in the hidden layer, different number of training epochs, learning rates and the mean squared error achieved for the twelve input variables: the normalized input airflow rate trajectory (i1_1), three delayed normalized input values of the airflow rates, time, three delayed time functions, normalized output trajectory (i2_2) and three delayed normalized values of the outputs forming the input vector P=[Z m0 m1 m2 i1_1 k0 k1 k2 i2_2 n0 n1 n2]' are used. The targeted output is the normalized DO concentration (i2_2). The responses (trajectories) of the training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.103 to 4.106.

Table 4.27: Results for the twelve input vector to Elman ANNs and trained with *trainscg* algorithm

Type of ANN(Network structure)	Activation functions& training algorithm*(trainscg)*	No of hidden (layers)	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	3976	0.05	1.42529e-005
NEWFF	Tansig, Purelin	3	4224	0.05	1.24399e-006
NEWFF	Tansig, Purelin	4	7074	0.05	9.95962e-007
NEWFF	Tansig, Purelin	5	6083	0.05	9.9998e-007
NEWFF	Tansig, Purelin	10	2900	0.05	9.84093e-007
NEWFF	Tansig , Purelin	15	3100	0.05	9.99833e-007
NEWFF	Tansig , Purelin	20	1028	0.05	9.98951e-007

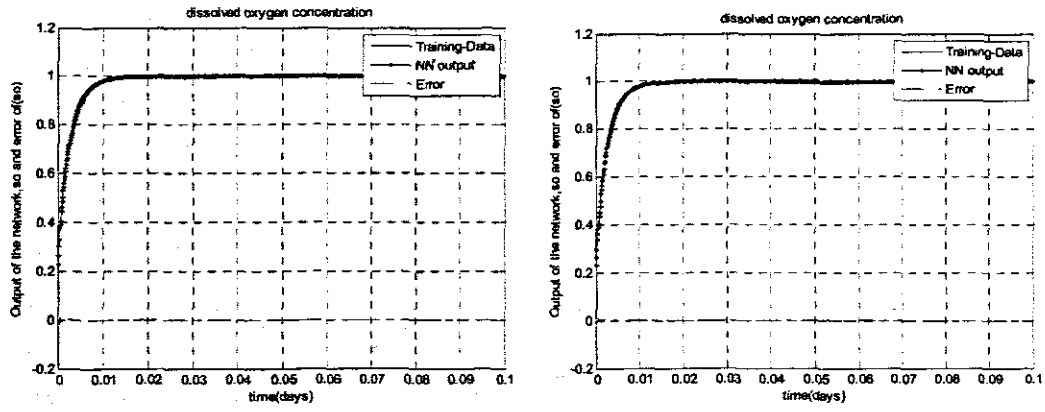


Figure 4.103: Elman networks response with 12 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 4).

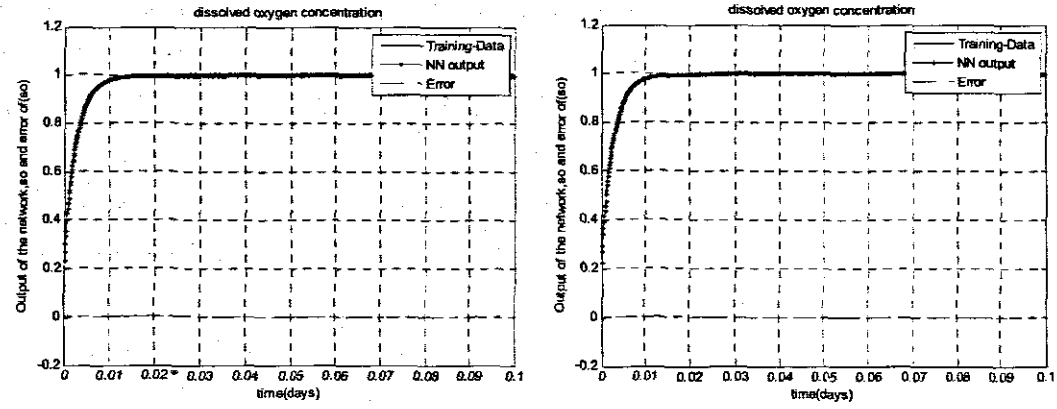


Figure 4.104: Elman networks response with 12 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 4).

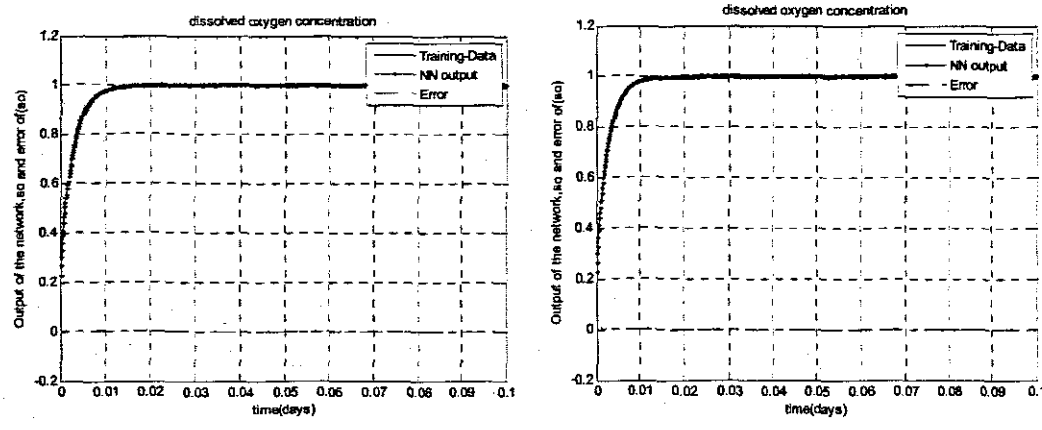


Figure 4.105: Elman networks response with 12 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 4).

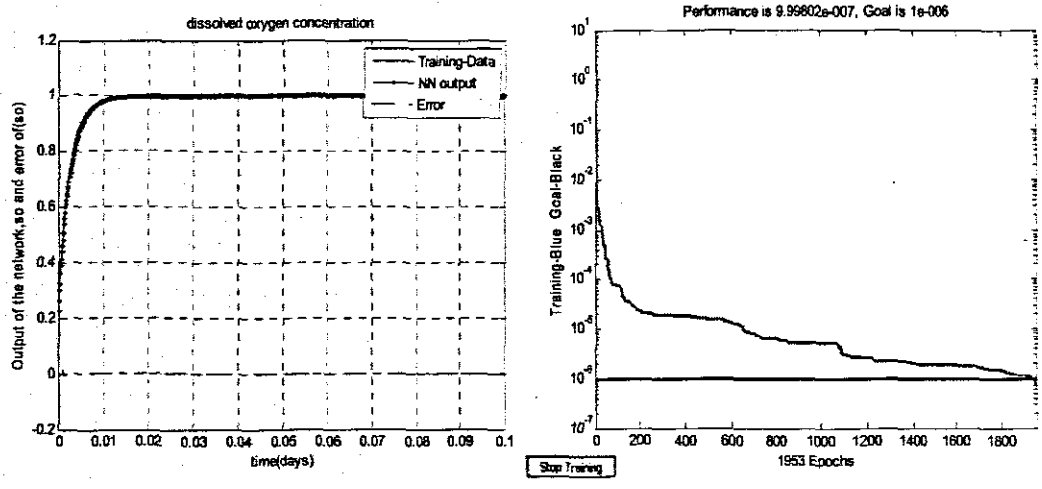


Figure 4.106: Elman networks response with 12 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 4) and Performance MSE achieved.

Analysis of this network from the perspective of the graphs plotted for the training data, NN output and the error as a function of time as illustrated in Figures 4.103 - 4.106, and analysis in addition to Table 4.27 which details the performance indices (MSE) have shown that:

- a network with 4 neurons in the hidden layer and which is trained with *trainscg* algorithm is just enough to identify the DO concentration model
- A small convergence error as low as an MSE of 9.99802e-007 is also achieved.
- The network model trained with this algorithm is much faster than the one trained with *traingdx*. Moreover it takes only a couple of hundreds of iterations for the mse to converge to minimum as compared to the model trained with *traingdx*.

4.8.12 Elman network model4 with 12 Input variable and *trainlm* algorithm.

Table 4.27 shows the results of the Elman recurrent neural networks trained with the Levenberg-Marquardt training algorithm *("trainlm")*. Different number of hidden neurons in the hidden layer, different number of training epochs, learning rates and the mean squared error achieved for an input vector of twelve input variables consisting of the normalized input airflow rate trajectory (i1_1), three delayed values of the normalized input airflow rates, time, three delayed values of time function, normalized output trajectory (i2_2) and three delayed values of the normalized outputs. The input vector formed P=[Z m0 m1 m2 i1_1 k0 k1 k2 i2_2 n0 n1 n2]', and an output vector T= [i2_2]' are also tabled. The targeted output is the normalized DO concentration (i2_2). The responses (trajectories) of the training data, NN output and the error between the input and the target are illustrated in the subsequent Figures 4.107 to 4.110.

Table 4.28: Results for the 12 input variables to Elman NN and trained with *trainlm* algorithm- model4

Type of ANN(Network structure)	Activation functions& training algorithm*(trainlm)*	No of hidden neurons	Training Epochs	Learning rates	MSE
NEWFF	Tansig, Purelin	2	24	0.05	9.89014e-007
NEWFF	Tansig, Purelin	3	51	0.05	9.92727e-007
NEWFF	Tansig, Purelin	4	28	0.05	9.6925e-007
NEWFF	Tansig, Purelin	5	6	0.05	9.94908e-007
NEWFF	Tansig, Purelin	10	6	0.05	8.03488e-007
NEWFF	Tansig , Purelin	15	6	0.05	4.34115e-007
NEWFF	Tansig , Purelin	20	6	0.05	1.4652e-007

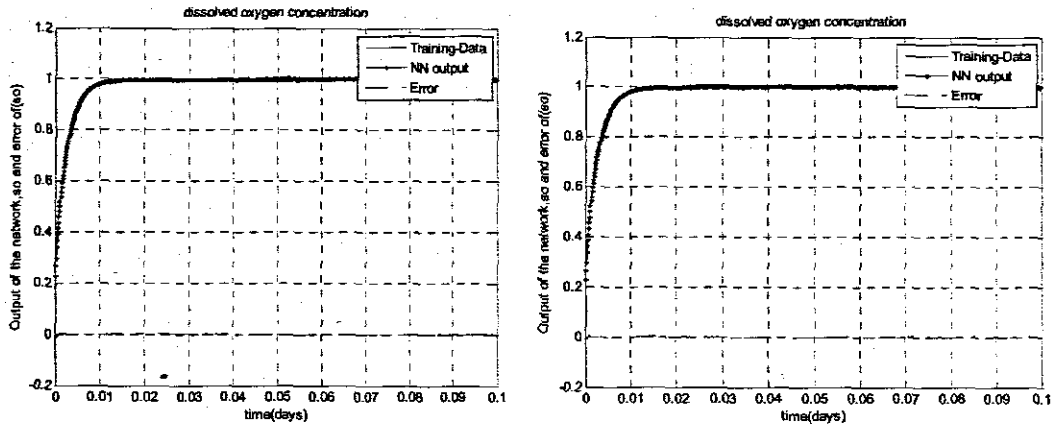


Figure 4.107: Elman networks response with 12 inputs, 1 hidden layer, 2 and 3 hidden neurons and 1 output (model 4).

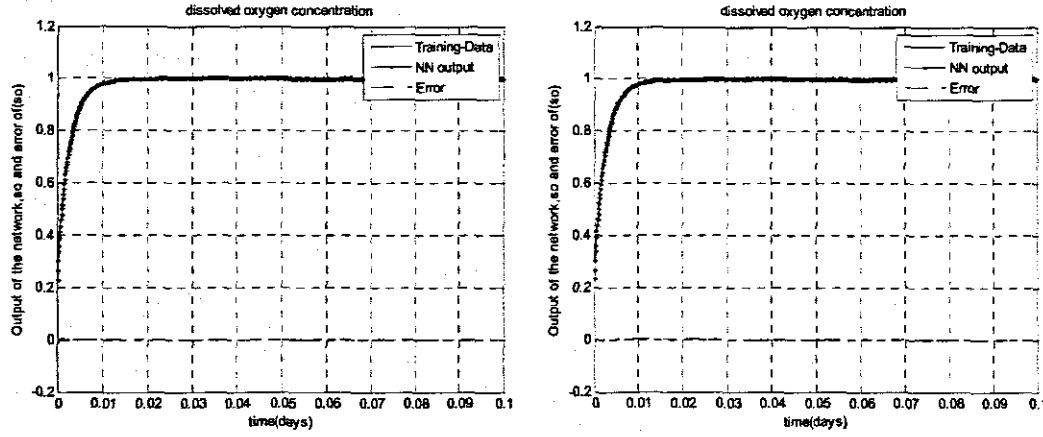


Figure 4.108: Elman networks response with 12 inputs, 1 hidden layer, 4 and 5 hidden neurons and 1 output (model 4).

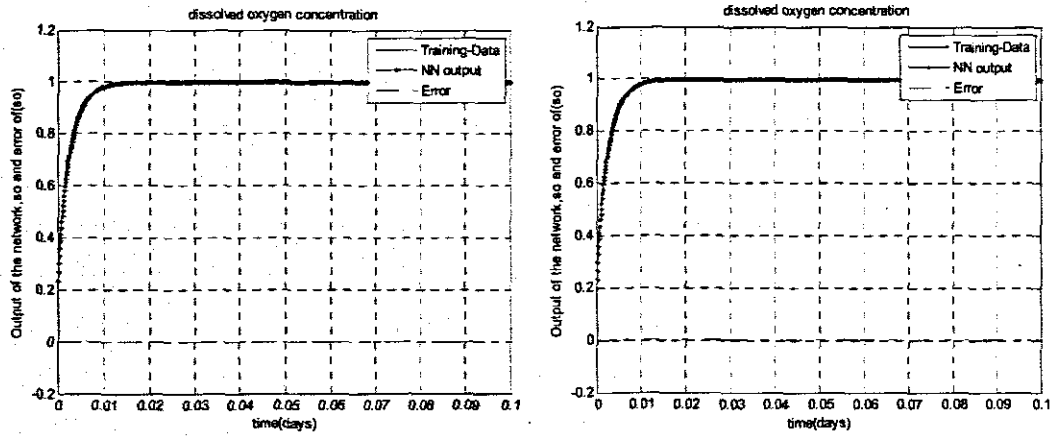


Figure 4.109: Elman networks response with 12 inputs, 1 hidden layer, 10 and 15 hidden neurons and 1 output (model 4).

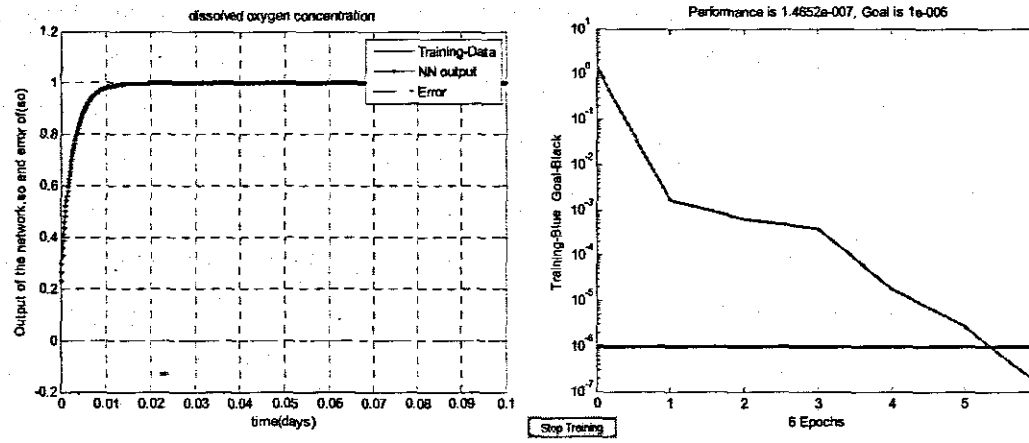


Figure 4.110: Elman networks response with 12 inputs, 1 hidden layer, 20 hidden neurons and 1 output (model 4) Performance MSE achieved.

Analysis of this network from the perspective of the graphs plotted for the training data, resultant NN output and the error as a function of time as illustrated in Figures 4.107 - 4.110, and analysis in addition to Table 4.28 which details the performance indices (MSE) have shown that:

- a network with at least 2 neurons in the hidden layer and which is trained with *trainlm* algorithm is just enough to identify the DO concentration model trajectory and that a convergence error as low as1.4652e-007 is also achieved with only 6 number of iterations if the network has 20 neurons in the hidden layer.
- The network model trained with this algorithm is the fastest of all the other algorithms tested.
- Moreover it takes only a couple of hundred iterations for the MSE to converge to minimum as compared to *trainscg* and *traingdx* algorithms.

Thus, the general conclusion made from the three types of the gradient descent training algorithms used to train the Elman network model4 for twelve input variables is that the variable learning rate backpropagation algorithm *(traingdx)* produced moderately low value of the MSE and exhibited not so good mapping of the target. The network trained with the scaled conjugate gradient algorithm *(trainscg)* produced lower MSE after a fewer number of training epochs, converged much faster for a small number of hidden layer nodes, exhibited good mapping of the target. Levenberg-Marquardt algorithm *(trainlm)* was the fastest in convergence with only a few numbers of epochs, very low MSE, better mapping of the target than the other two. Also with twelve inputs to the network two hidden nodes are just enough for the *trainlm* to map the target of the DO process.

4.8.13 Discussion of the results for Elman neural networks

Four models have been developed and trained with three different types of training algorithms in order to determine which model describes the best mapping for the DO process dynamics and which training algorithm gives the best results. The models developed are summarized in Table 4.16. Models 3 and 4 gave better results of the mapping compared to models 1 and 2. The other observation is that the number of input variables need not be twelve in order to approximate the DO process. Therefore a variation of model 4, i.e. six input variables i1_1, i2_2 and their two delays will be chosen for the DO process. This model is subsequently analysed and verified as the best model.

In terms of the training algorithms used, the Levenberg-Marquardt *(trainlm)* algorithm, performed better than the other algorithms in terms of the speed of convergence, accuracy of approximation, ability to drive the mean squared error to the lowest level. Also, the error in the Levenberg-Marquardt *(trainlm)* algorithm decreases much more rapidly with time than the other two algorithms. Similarly the scaled conjugate gradient descent algorithm converges faster than the resilient backpropagation algorithm however it requires more iterations than *trainlm* to converge to minimum MSE. Also as the number of input variables is increased, the number of hidden nodes also increases for best mappings. Thus, from the three algorithms tested, the Levenberg-Marquardt *(trainlm)* algorithm is chosen for training the model of the DO process.

To increase the efficiency of training, and to decide when to stop training, the data is divided into three subsets; one half is used for the training, one quarter for validation and the last quarter for testing. The section that follows provides the generalization analysis, post-training analysis and regression analysis, of the results for the chosen model trained with *trainlm* algorithm and verification of the suitability of the model and the algorithm for the DO process identification.

4.8.14 Best selected Elman model of the DO process and the verification results

The data for training, validation and testing were sampled at $1.041677*10^{-4}$ days (1min.30s) intervals with 960 data sets collected. 480 data sets were used for training, 240 data sets for validation (for observing the performance of the models when faced with unknown situations or dynamic changes of different amplitudes and/or frequencies) and 240 data sets for testing. The division was such that the testing set starts with the second point of the considered time interval of the process dynamic behavior and takes every fourth point. The validation set starts with the fourth point and take every fourth point, while the training set takes the remaining points. The data collected included the present and past values of the control input $u(k)$ and the DO

concentration $S_o(k)$ as required by for identification purposes. The data gathered were scaled for training purposes. In this work the training data were normalized to the range [0 1], as i1_1 and i2_2 for $u(k)$ and $S_o(k)$ respectively and the network was created and trained using the normalized data and simulated. Eventually the network output was de-normalized, and a linear regression between the network outputs (de-normalized) and the targets to check the quality of the network training was performed. Figure 4.111 shows Matlab commands algorithm that were used to normalize data, split the data, train the network, simulate and perform linear regression analysis.

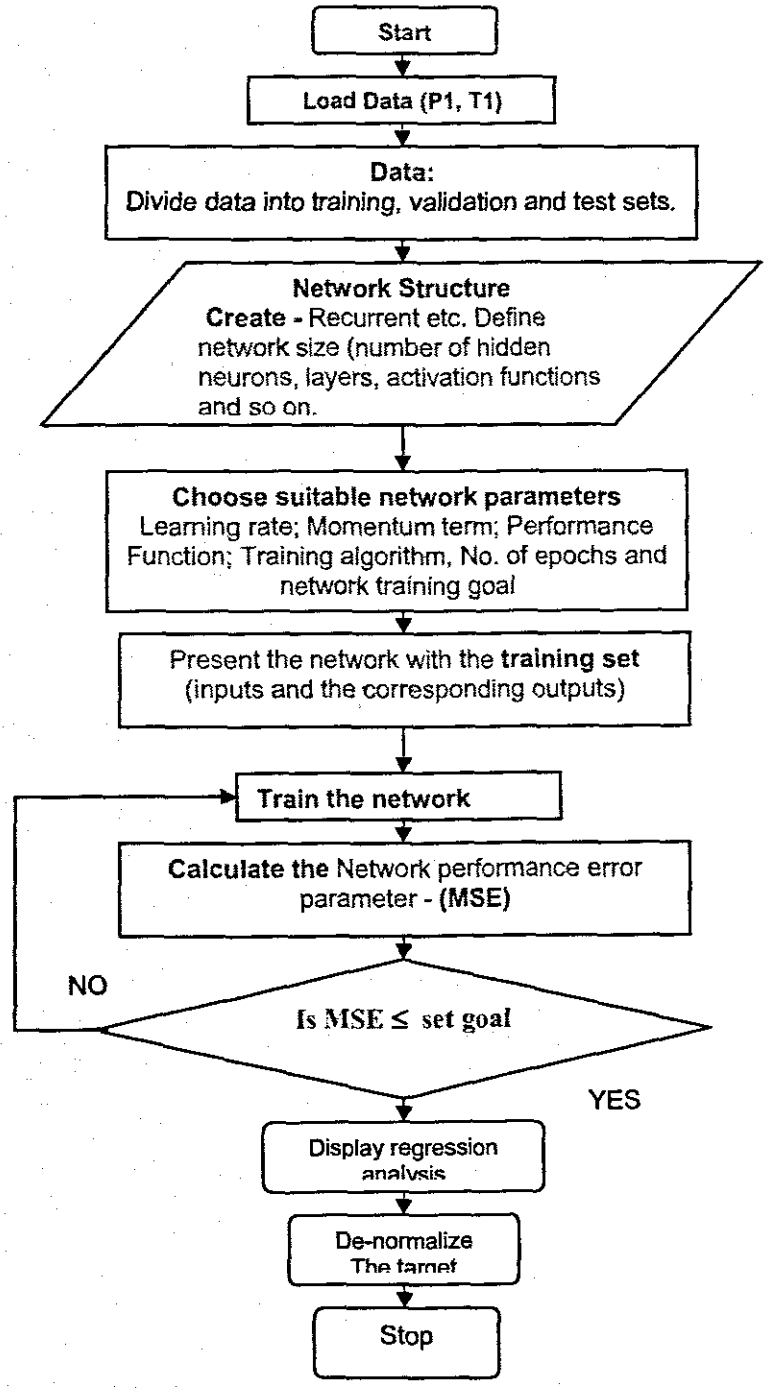


Figure 4.111: The algorithm used to train the model of the Elman network.

The above Matlab algorithm represents the code that creates a recurrent network. ***P1, T1*** represents the range of the normalized inputs and target vectors. **k0, k1, n0 and n1** are the delays in **i1_1** and **i2_2**. The best structure of the identified neural model chosen was **6-6-1**; i.e. 6 inputs of the normalized airflow rate with two delayed values **k0** and **k1** and normalized DO process with two delayed values **n0** and **n1**, 6 nodes in the hidden layer and 1 node at the output.

Training was done by switching between the training, validation and test data set; a technique known as *'early stopping'* methodology. The training was performed in the normal way on the training set until a reasonably small error (root mean square - RMS) was achieved for this set. Once a reasonable and continuously decreasing RMS error was achieved for all the sets of data, the training was stopped automatically and the network was validated by applying different test data, not utilized before. Eventually the topology and configuration were finalized. This method was used to prevent over-training. The error on the validation set will normally decrease during the initial part of the training. However, when the network begins to over fit the data, the validation set error will start to rise. When this increase continues for a pre defined number of iterations the training is stopped and the weight values are kept. The test set is used to compare with the validation set to see if they exhibit a similar behavior. If validation errors and test errors do not show a similar behavior, this may indicate a poor division of data. In Figure 4.112 the mean squared error between the calculated values and the target training, validation and test sets are plotted against the epoch (number of iterations).

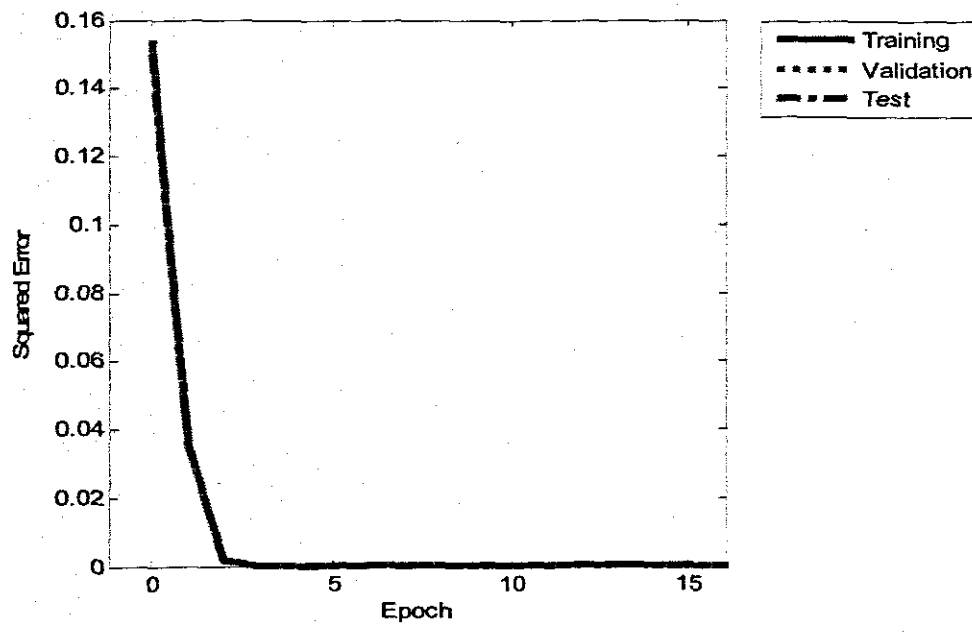


Figure 4.112: Plot of the training, validation and test errors for a generalized network.

The results of Figure 4.113 shows the plot of the mean squared error between the calculated values and the target training, validation and test sets plotted against the epoch (number of the training, validation and test errors) with the convergence performance of the MSE of $6.15821e^{-8}$ achieved for architecture with 6 hidden neurons. Analysis of the plot tells that it requires only 29 iterations to map the target when trained with *trainlm* algorithm.

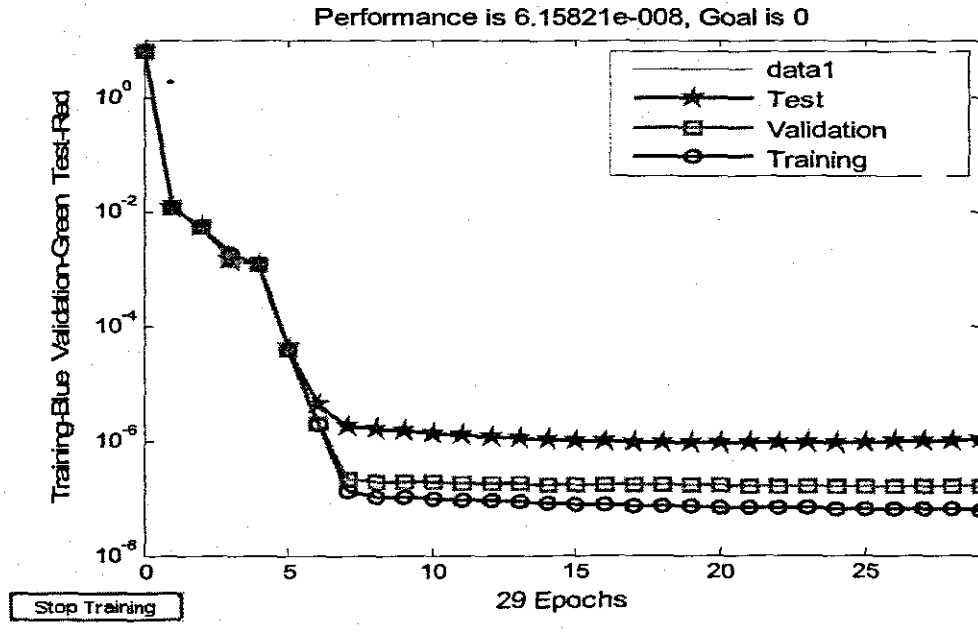


Figure 4.113: Performance MSE with 6 inputs, 1 hidden layer, 6 hidden neurons and 1 output.

The post-training analysis of the plots showing regression analyses between the network outputs and the corresponding targets (in original units) is subsequently done as illustrated in Figure 4.114.

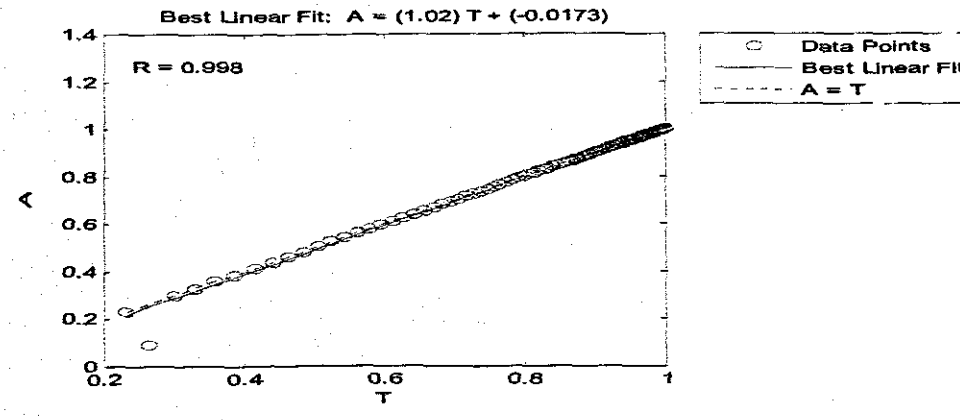


Figure 4.114: Post-Training analysis with Matlab command routine *(postreg)*

Figure 4.114 displays the plots showing regression analyses between the network outputs and the corresponding targets (in original units of data). The routine *postreg* is designed to perform this analysis. Here the network output and the corresponding targets are passed to *postreg*. It returns three parameters. The slope, and the y-intercept of the best linear regression relating targets to network outputs and the correlation coefficient (R-value) between the outputs and targets. In the analysis it can be seen that a perfect fit indicated by the solid line results (outputs are exactly equal to targets), because the slope is equal to one **1.02**, and the y-intercept is **-0.0173**, and the R-value is also equal to **0.998** because the fit is good. This therefore confirms that the selected neural model is a good identifier of the DO process.

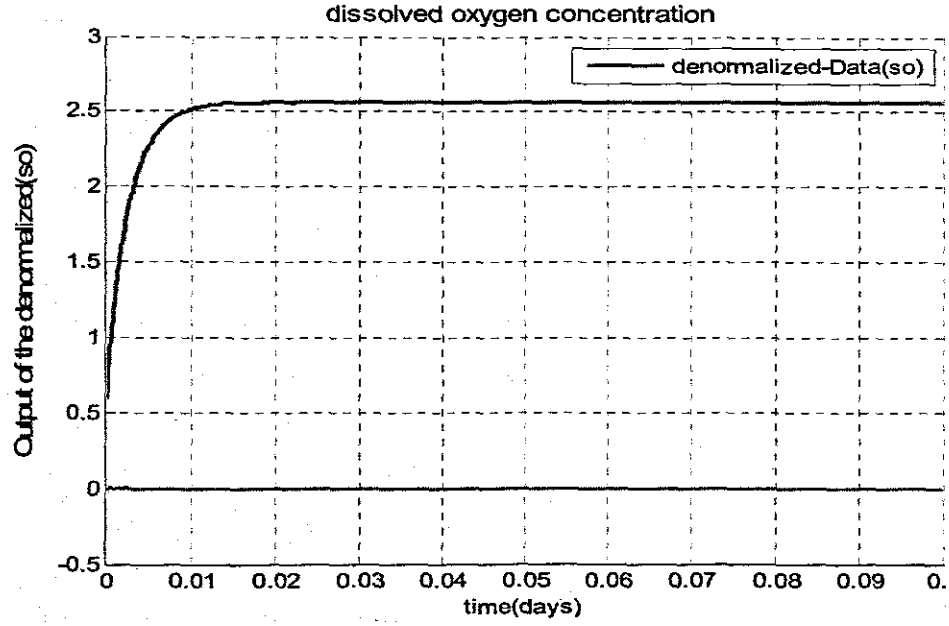


Figure 4.115: De-normalized NN output trajectory of the DO process.

Figure 4.115 shows the de-normalized neural network output. This trajectory is an approximate of the DO process model. This therefore conforms that the results of the verification is correct. The coefficients (weights and biases) of the selected model trained with the command *newelm* are tabulated in the following Tables 4.28 and 4.29 respectively.

Table 4.29: Hidden layer weights and biases

	Hidden layer weights (w1)						Hidden layer bias (bw1)
Inputs / nodes	i1_1	a k0	k1	i2_2	n0	n1	
1	-5.1483	b0.4401	0.1207	-0.7573	-1.3919	1.1188	6.7504
2	502529	-0.1234	-0.7384	1.5165	-0.8094	-0.5258	-6.151
3	-0.6311	-0.0336	0.0389	0.6746	0.0072	-0.3148	0.4866
4	12.8846	0614	1.1943	-0.9821	-0.2576	-1.7116	-10.7253
5	-608504	-0.1792	0.6988	1.4611	-1.8511	-0.3012	6.2623
6	5.2596	0.4855	0.2022	0.1024	-0.7314	1.2589	-3.9476

Table 4.30: Output layer weights and biases.

Output weights(v1)	-0.1514	0.2871	2.4115	-0.0087	-0.0925	1.0862
Output bias(bv1)	-0. 366					

4.9 Comparison of the Elman model and the Feedforward models developed.

From the analyses done and verification results of the two selected models both (the recurrent network and the feedforward networks trained with the Levenberg-Marquardt *(trainlm)* algorithm the conclusion derived is that both models chosen can be used for identification of the DO process since both networks yields satisfactory results for the DO process.

4.10 Conclusion.

System identification is concerned with developing the neural model of the DO concentration trajectory based on the input and output data of the actual system. The different methods of identifying the DO concentration trajectory using neural networks have been studied in this chapter. The two types of neural network model structures (the Feedforward and the Recurrent networks) used in this project for identification have been explored. Several simulation results have been provided for different types of the training algorithms (the Levenberg-Marquardt training algorithm, the scaled conjugate gradient descent algorithm, the Variable Learning Rate Backpropagation training algorithm, and the adaptive algorithm function).

The results have been used to show the algorithm effectiveness in identifying the dissolved oxygen trajectory. These algorithms have been chosen because of their salient features. Analysis of the results of all the models used and verified, together with the algorithms used, helped six input variables to be selected as the neural network input. A **6-8-1** feedforward neural network has been verified to be an ideal identifier of the DO process. Similarly a **6-6-1** recurrent network has been proven to identify the DO process. Therefore these two structures will be used to design the neural network controllers which are discussed in the chapter 5. The Levenberg-Marquardt training algorithm has also been adapted as the suitable training algorithm for these networks.

Chapter 5 that follows gives an overview of how to design neural network controllers. This is followed by the design and development of the neural network based controllers for the DO process. The design will be based on the identified models of the DO process. Particular emphasis will be put on Internal model control and feedback Linearising Control design strategies to maintain the desired dissolved oxygen (DO)

level of an activated sludge system by manipulating the air flow rate in the aerobic reactor of an activated sludge process.

CHAPTER FIVE

NN CONTROLLER DESIGN

The aim of controller design is to construct a controller that generates control signals that in turn generate the desired output of the plant subject to given plant's constraints. Control of nonlinear systems is still a challenging area in control systems theory and many efforts have been made to study this subject. This chapter provides some background concerning concepts in control theory, general issues involved in nonlinear control design, and the neural networks used for nonlinear control. Subsequently the objectives of this chapter is to design a MISO robust control law for regulation of DO concentration of an Activated sludge process of WWTP via the airflow rate(non-affine control input).This is to be achieved by various NN-based controller structures. In particular, a comparative study of the different approaches for NN controller design used in this project, as the NN direct Inverse model control, internal model control (IMC) and feedback linearization control strategies to stabilize the dissolved oxygen (DO) level of an activated sludge system according to a given set-point by manipulating the air flow rate is performed.

5.1 Concepts of control systems theory

As explained in section 2.4, generally the aim of an automatic control system is to maintain, direct or regulate some dynamic variable or state of the process to a desired value in the presence of disturbances. Control is only one area in which neural networks have been applied, yet control has its own unique set of problems to solve when applying any methodology. The main principle behind control is to change the performance of the system to conform to a set of specifications. This goal can be complicated by uncertainties in the system, including nonlinearities. As a starting point, the four building blocks of any closed loop control system are:

- the process to be controlled,
- the actuators that allow manipulation of the process,
- the control algorithm that calculates a proper action for disturbance rejection or set point tracking, and
- the measurement devices that provide information on the important output variables and disturbances.

These building blocks are illustrated in Figure 5.1 as a feedback control loop.

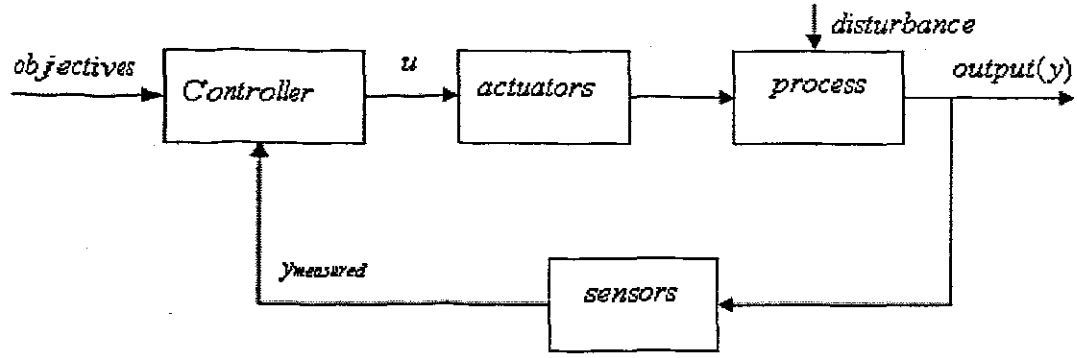


Figure 5.1: General feedback control loop (P. Vanrolleghem., 1994)

Normally the control structure design comprises of two steps, namely input /output selection and control configuration selection. This latter step is a very important one, because a wrong choice of control structure can put fundamental limits on the performance of the total system (P. Vanrolleghem., 1994). In this project a basic control strategy is proposed to test the benchmark model. Its aim is to control the dissolved oxygen level at a set point of 2 g/m^3 in the ASM1 model of the reactor by manipulation of the air flow rate. However it must be acknowledged that the activated sludge process of a WWTP is a multivariable process and that several variables can be manipulated in order to achieve nitrogen and phosphorus removal. However, the main control signal in this thesis is the manipulation of the air flow rate to the aeration tank in order to achieve an output which is the desired level of the dissolved oxygen concentration. The control should guarantee that aerobic bacteria are sufficiently aerated to achieve carbon consumption and nitrification.

5.2 Overview of controllers for wastewater treatment plants(WWTP)

In this section, a non-exhaustive overview is given on the various structural control models that have been used in wastewater treatment plants. In the control literature a number of well established and deeply analysed structures exist and can be quoted under the following headings:

5.2_1 Conventional classical feedback control

Many methods and approaches to control have been developed and are available in various literatures. These include: on-off control, e.g. the simple PLC-techniques, on-off control with a time delay (for time control), classical control, including PI and PID controllers, and cascade control. However, in the view of the characteristics of the WWTP process, which is nonlinear, time variant (as a result of changing parameters), multivariable and relatively slow, the performance in terms of effluent quality has not been perfect. For example, even though PID controllers are familiar to process

operators and very popular because of the simplicity, easiness in operation and robustness to modeling error, it is well known that the DO concentration cannot be controlled effectively by using the PID controllers with fixed gain parameters. And the manual tuning of PID controller is tedious and laborious. To overcome the time-consumed manual tuning procedures of PID controllers, many on-line identification methods have been proposed to obtain the process information.

Because of the importance of dissolved oxygen control many different control strategies have been proposed in the past. However, majority of these proposals have been on the classical control point of view, among them (Olssen and Newell, 1999; Flanagan et al, 1977; Chang Kyoo Yoo et al, 2003; Holmberg et al, 1989; Carlsson, 1992; Lindberg and Carlsson, 1996; Lindberg, 1997; Van Lupe et al, 1998, or Andrews, 1992; Olsson G., Jeppsson U., 1994; Steffens,. et al) to mention just a few, have contributed to the development of control strategies aimed at maintaining proper levels of dissolved oxygen concentration in the aerobic tanks.

5.2.2 Optimal control

Optimal control schemes have also been applied most widely in wastewater treatment processes for a long time (Andrews J.F., 1974; Marsili-Libelli S., 1989; Herremans C., et al., 1997; Spanjers H., 1998b; Olsson G., Newell R.B., 1999). The controller design methods have been developed which aim to devise a controller that extremises a certain criterion function on the basis of effluent substrate concentration measurements. When nonlinear models are the only reasonable means to describe the process dynamics, only a few results of an analytical solution of the optimal control law have been published, among them, (D'ans G., et al, 1971; Herremans C. e t al, 1997). A solution is to approach the optimisation problem by numerical means and a number of exercises have been performed. The main problem with the resulting control strategies is that no closed-loop solution is obtained and that the optimal control solution relies on the assumption of a perfect process model with fixed model structure and parameters.

5.2.3 Feedforward control

Another control scheme that has been implemented in the past for WWTP is the feedforward control scheme. In a feedforward control scheme the knowledge on upcoming disturbances is used to the benefit of a treatment plant's operation by preparing it to cope with for instance toxic loads, important hydraulic disturbances or increased organic loading. In such set-up, the feedforward part of the controller aims to anticipate for the effect of measured disturbances, while the feedback part corrects for

any deviations that result from the deficiencies in process model, control limits and inadequate measurement or prediction of the disturbance(s). Ratio control is a simple feedforward control algorithm that consists of setting a control variable proportional to the disturbance. A classical example is the ratio control of the sludge recycle flow rate to the influent flow rate that acts as the disturbance. The goal of the control action is to maintain the sludge concentration in the aeration tank and therefore the biodegradation capacity nearly constant (Brett R.W.J., et al, 1973; Andrews J.F., 1974).

5.2.4 Multi-Input/Multi-Output control

Multi-input Multi-output (MIMO) control strategies have also been applied to WWTPs with success. MIMO controller design techniques aim for complete elimination of the interaction between loops (Steffens M, et al, 1997). A good example of this is provided by (Weijers S.R, 2000; Nielsen M.K., and Önnerth T.B., .1995). They showed the advantage of employing a two-input control of a denitrification reactor. Both the carbon addition and the oxygen supply were manipulated for the control of effluent nitrate. Moreover, in this work the cost/benefit was clearly proven at a full-scale WWTP. However, (Steffens M, et al, 1984; Weijers S.R., 2000; Lech R.F., 1978a] And Lech R.F., 1978b) also exemplified for a wastewater treatment system that interactions between control loops may also lead to process instability.

5.2.5 Nonlinear control

As mentioned previously, there are cases when adaptive linear control schemes would not perform well when faced with a highly nonlinear process. This is because the adaptive mechanism may not be fast enough to track changes in process characteristics. Appropriately designed nonlinear controllers would therefore be expected to perform better. An emerging field is that of nonlinear controller design by the use of differential geometric concepts. The aim of the design is opposite to the use of Taylor series expansion to linearise the nonlinear model prior to application of linear model based controller designs. The nonlinear control technique can produce a linear closed loop system on the basis of a nonlinear process and a nonlinear controller. This approach is called feedback linearization. The method is based on either nonlinear change of coordinates or nonlinear state feedback. Several authors give a thorough treatment of input-output and input-state feedback linearization in a continuous time setting.

Illustrative examples of linearized models as portions of model based control systems to ensure that the closed-loop behaviour is linear can be found in (Weijers S.R., 1997; Ko K.Y, et al 1982; Bastin G., Dochain D., 1990; Lindberg C.-F., 1997). The extension

of linearising control towards MIMO models was presented by (Dochain D., 1991) and applied to wastewater treatment systems in (Vanrolleghem P.A., et all, 1991; Dochain D., et all, 1997).

5.2.6 Adaptive and robust control

Changes such as shifting the operating point in nonlinear control systems often lead to parameter variations. These variations can affect the system accuracy and stability severely. Through the use of adaptive control, control engineers aim to make the system to automatically redesign the controller when parameter changes occur. Self-tuning PID regulators have been used in the adaptive control of dissolved oxygen in activated sludge plants (Olsson G., et all, 1985; Marsili-Libelli S., 19 90; Carlsson B., et all, 1994).

Robust control means the design of a controller such that some level of performance of the controlled system is guaranteed irrespective of changes in the plant dynamics within a predefined class. Robust control theory has also been applied in the field of wastewater treatment. More literature work on robust control can be found in (Haarsma G.-J., Keesman K., 1995; Steyer J.P., et all, 1995; Weijers S.R., 2000).

5.2.7 Neural networks and Fuzzy control

Several researchers have provided rigorous mathematical analyses of neural networks in closed-loop control applications. There is a rich collection of nonlinear neural networks control techniques using neural models and the inverse models, with each technique dealing with different classes of nonlinear problems. However, the background of all these efforts have been provided by Narendra and co-workers in several seminal works in the early 1990s followed by Lewis and co-workers in the mid-1990s (Sarangapani, Jagannathan. 2006).

The objective of control design is to construct a feedback control law to make the closed loop system display the desired behaviour. Usually, stability and robustness are the performance requirements for closed-loop systems. For linear, time invariant systems it is straight forward to investigate stability of a system by examining the location of the poles in the s-plane. However, for nonlinear systems there are no direct techniques.

For a control application, the inputs of a neural network controller consist of measurements of the process. Control action is then obtained as the network output. Two approaches can be distinguished in choosing the control structure for such a process: the first one is process-driven and the second one is model-based. The first approach deals with the separate control of the most important variables. Within this category, the well-known problem of controlling the dissolved oxygen level is one of

the most important issues for a good operation of the wastewater treatment plants since a good level of dissolved oxygen allows for optimal growth of microorganisms used in the WWTP process.

Application of the NN in process control is usually done in one of the two ways:
1) Direct control and
2) Indirect control.
Direct NN control means that a NN is an active element of the controller; usually this type requires on-line training. Examples of direct NN controllers are: NN model reference control, feedforward with NN inverse model control and NN internal model control. The indirect methods represent a more conventional approach, where the design is based on NN model of the system to be controlled. In this case the controller is not itself a NN. The idea is to use a NN to model the system to be controlled, and then this model is employed in a more conventional controller design. Neural networks for indirect control have been widely applied to system identification and indirect control of nonlinear systems, while less work had been presented about the use of NN as direct controllers (Huang and Lewis, 2003).

In case a lot of qualitative knowledge is available, the fuzzy sets can provide an excellent means of representing the control scheme in mathematical terms. Fuzzy control systems have been designed for the different unit processes of wastewater treatment plants, *e.g.* controlling the influent pumping rate in a sewer system, anaerobic digestion regulation dissolved oxygen control (Fukano T., 1993; Muller A., *et al* 1997; Steyer J.P., et al 1997).

5.3 Nonlinear control Problem

Generally, the task of control systems can be divided into two categories: stabilization (or regulation) and tracking (or servomechanisms). In stabilization problems, a control system is designed such that the state of the closed loop system will be stabilized around an equilibrium point. In the tracking control problems, the design objective is to construct a controller, so that the systems output tracks a given time-varying trajectory. The following formal definitions suffice for the types of control schemes:

Asymptotic stabilization problem: *Given a nonlinear dynamic system described by* $x(k+1) = f(x(k),u(k))$ *find a control law* u *such that, starting from anywhere in the region in* Ω, *the state* x *tend to 0 as* $t \to \infty$.

Asymptotic tracking problem: *Given a nonlinear dynamic system described by* $x(k+1) = f(x(k),u(k))$

$y(k) = h(x(k))$

and the desired output trajectory y_d, *find a control law for the input* u *such that, starting from any initial state in a region in* Ω, *the tracking error* $y(k) - y_d(k)$ *goes to zero, while the whole state* x *remains bounded.*

The focus on this thesis is on stabilization problems, i.e. the design objective is to construct nonlinear neural networks controller so that the system output is stabilized to a given set point y^{sp} (nonlinear DO concentration system). When the closed loop system is such that proper initial states imply zero steady state error for all the time, i.e. $y(k) \equiv y^{sp} \quad \forall k \geq 0$, the control system is said to be capable of perfect stabilization.

There is no general method for the design of a nonlinear controller. However, there is rich collection of alternative and complementary techniques, each best applicable to particular class of nonlinear control problems. In this study; neural networks based control structures are selected for control problem design. These are presented in the following sections and eventually applied for the DO concentration process under consideration. As will be shown, the existence of such a controller is guaranteed if the system under consideration is controllable around the origin. The neural networks will be used to determine the control input u.

5.3.1 Design of nonlinear neural networks controllers for DO concentration

Models of dynamic systems and their inverses have immediate utility for control design. In the nonlinear control literature, a number of well established and deeply analysed structures have been proposed and used to control nonlinear plants subject to uncertainties and disturbances. These have been reviewed in the preceding sections. The universal approximation capabilities of the multilayer perceptron have made it a popular choice for modeling nonlinear systems and for implementing general-purpose nonlinear controllers. This section presents the design and development of neural network-based controllers for the DO concentration in the activated sludge process. In particular, the model of the DO process dynamics derived in chapter 2 equation (2.24) is used for the design. A comparative study of the various neural network-based controllers such as the direct inverse control, internal model control and Feedback linearization neural networks control strategies is provided first and then these techniques are used to design NN controllers in order to maintain the DO concentration level in an activated sludge system by manipulating the air flow rate.

5.3.2 The Inverse Model of a Dynamic System

A straight forward approach to model-based design of a controller for a nonlinear process is the adaptive inverse control. The inverse model of a system plays an important part in the theory of neural control design. This can be applied to a class of systems that are open-loop stable (that are stabilizable by feedback) and whose inverse is stable as well, i.e., the system that does not exhibit non minimum phase behavior. The strategy consists of the neural network inverse model that acts as the controller placed in series with the process under control. In this work, the neural network inverse model is trained in a similar way as for the process identification as described in chapter 4 and is utilized to predict the manipulated airflow rate u to the aeration tank to bring the process output (DO concentration) to desired conditions of the set-point of 2mg/l. The approach is explained for SISO mathematical models.

Consider the discrete time dynamic system described by Equation (5.1):

$$y(k+1) = f[y(k), y(k-1),.....y(k-n+1), u(k), u(k-1),.....u(k-m+1)] \quad (5.1)$$

In the control problem, the objective is to determine the input $u(k)$ so that the system behaves in a desired fashion. In many practical situations, the plant input is limited in amplitude, i.e. there exists $u_{\min}$ and $u_{\max}$ such that, for any k : $u_{\min} \le u(k) \le u_{\max}$. The vector:

$$x(k) = [y(k), y(k-1).....y(k-n+1), u(k-1).....u(k-m+1)]^T \quad (5.2)$$

where n is the number of delays in the input states, and m is the number of delays in the control inputs, denotes the actual state and thus it does not include the current control input $u(k)$ such that the system's output at the next sampling instant is equal to the desired (reference) input $r(k+1)$. Thus the task is to learn how to control the plant described in Equation (5.1) in order to follow a specified reference $r(k+1)$ minimizing some norm of the error $e(k) = r(k) - y(k)$. The control input can be calculated on the basis of the process model of Equation (5.1) according to:

$$u(k) = f^{-1}(x(k), r(k+1)) \quad (5.3)$$

This can be re-written as:

$$u(k) = f^{-1}[r(k+1), y(k),.....y(k-n+1), u(k-1),.....u(k-m+1)] \quad (5.4)$$

Generally, it is difficult to find the inverse function f^{-1} in an analytical form. It can however, always be found by numerical optimization techniques using the objective function of Equation (5.5):

$$J(u(k)) = [r(k+1) - f(x(k), u(k))]^2 \quad (5.5)$$

where the minimization of J with respect to $u(k)$ gives the control signal corresponding to the inverse function of Equation (5.4), if it exists, or the least-square

5.3.2 The Inverse Model of a Dynamic System

A straight forward approach to model-based design of a controller for a nonlinear process is the adaptive inverse control. The inverse model of a system plays an important part in the theory of neural control design. This can be applied to a class of systems that are open-loop stable (that are stabilizable by feedback) and whose inverse is stable as well, i.e., the system that does not exhibit non minimum phase behavior. The strategy consists of the neural network inverse model that acts as the controller placed in series with the process under control. In this work, the neural network inverse model is trained in a similar way as for the process identification as described in chapter 4 and is utilized to predict the manipulated airflow rate u to the aeration tank to bring the process output (DO concentration) to desired conditions of the set-point of 2mg/l. The approach is explained for SISO mathematical models.

Consider the discrete time dynamic system described by Equation (5.1):

$$y(k+1) = f[y(k), y(k-1), \ldots\ldots y(k-n+1), u(k), u(k-1), \ldots\ldots u(k-m+1)] \quad (5.1)$$

In the control problem, the objective is to determine the input $u(k)$ so that the system behaves in a desired fashion. In many practical situations, the plant input is limited in amplitude, i.e. there exists $u_{\min}$ and $u_{\max}$ such that, for any k : $u_{\min} \le u(k) \le u_{\max}$. The vector:

$$x(k) = [y(k), y(k-1) \ldots\ldots y(k-n+1), u(k-1) \ldots\ldots u(k-m+1)]^T \quad (5.2)$$

where n is the number of delays in the input states, and m is the number of delays in the control inputs, denotes the actual state and thus it does not include the current control input $u(k)$ such that the system's output at the next sampling instant is equal to the desired (reference) input $r(k+1)$. Thus the task is to learn how to control the plant described in Equation (5.1) in order to follow a specified reference $r(k+1)$ minimizing some norm of the error $e(k) = r(k) - y(k)$. The control input can be calculated on the basis of the process model of Equation (5.1) according to:

$$u(k) = f^{-1}(x(k), r(k+1)) \quad (5.3)$$

This can be re-written as:

$$u(k) = f^{-1}[r(k+1), y(k), \ldots\ldots y(k-n+1), u(k-1), \ldots\ldots u(k-m+1)] \quad (5.4)$$

Generally, it is difficult to find the inverse function f^{-1} in an analytical form. It can however, always be found by numerical optimization techniques using the objective function of Equation (5.5):

$$J(u(k)) = [r(k+1) - f(x(k), u(k))]^2 \quad (5.5)$$

where the minimization of J with respect to $u(k)$ gives the control signal corresponding to the inverse function of Equation (5.4), if it exists, or the least-square

approximation of it otherwise. A wide variety of optimization techniques are available and can be applied such as the Newton-Gaussian method or the Levenberg-Marquardt optimization. Thus, the simplest way to arrive at a system inverse neural model is it to train the neural network using the optimization algorithms to approximate the system inverse model.

5.3.3 Development of Inverse Model of the DO concentration process

The various steps of neural network based inverse model controller for the DO process are presented here. Inverse models provide the neural network structure which represents the inverse of the DO concentration dynamics in the region of the training/identification. There are several ways to carry out this identification process. The technique in the thesis here is known as the generalized inverse training method (Norgaard, M., 2000; Psaltis et al., 1988; Hunt & Sbarbaro, 1991, and Hunt et al., 1992). Here, the network is fed with the required future output (DO process) together with past outputs to predict the current input or the control action (airflow rate) $u(k)$.The trained network then represents the inverse of the model of the system (DO concentration dynamics). Once again, the assignment of the input nodes is as in the case of the forward model (identified DO process models) but with the prediction of $S_o(k+1)$ replaced by the control input $u(k)$ as is illustrated in Figure 5.2. The control input $u(k)$ data is given. Here l is the number of delays.

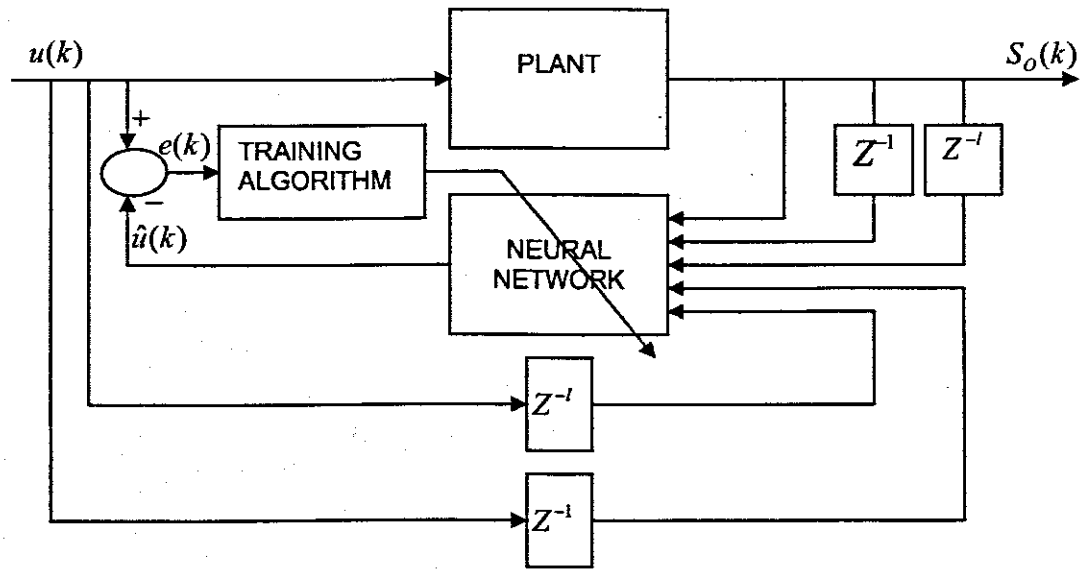


Figure 5.2: Generalized method for the training the Inverse Neural Networks for control

The inverse model is then trained based on the mean-squared error between the desired signal $u(k)$ and the signal produced by the model $\hat{u}(k)$.

5.3.3.1 Inverse model of the DO process

The above equations are applied to the model of the DO concentration under study, i.e.:

$$S_O(k+1) = S_O(k) + h^*\left\{\frac{Q}{V}\left[S_{O\,in}(k) - S_O(k)\right] + K_L a(u(k))\left[S_{O\,sat}(k) - S_O(k)\right] + r_{SO}(k)\right\} \quad (5.6)$$

$$S_O(k+1) = S_O(k) + h^*\left\{\frac{Q}{V}\left[S_{O\,in}(k) - S_O(k)\right] + K_1(1 - e^{-K_2 u(k)})\left[S_{O\,sat}(k) - S_O(k)\right] + r_{SO}(k)\right\}$$

where the output of the process, $y(k) = S_O(k)$ and all other terms have been defined. The considered time series is based on the measurements of system inputs and outputs for some period of time, $K = \overline{0, K-1}$.

The inverse model equation is derived as follows:

$$S_O(k+1) - S_O(k) - h^*\frac{Q}{V}(S_{Oin} - S_O(k)) - h^* r_{SO} - h^* K_1(S_{Osat} - S_O(k)) = \\ = -h^* K_1 e^{-K_2 u(k)}(S_{Osat} - S_O(k) \quad (5.7)$$

From here,

$$-e^{-K_2 u(k)} = \frac{S_O(k+1) - S_O(k) - h^*\frac{Q}{V}(S_{Oin} - S_O(k)) - h^* r_{SO} - h^* K_1(S_{Osat} - S_O(k)}{h^* K_1(S_{Osat} - S_O(k)} \quad (5.8)$$

Logarithm is taken to both sides of the equation, thus:

$$K_2 u(k) = \ln\left[\frac{S_O(k+1) - S_O(k) - h^*\frac{Q}{V}(S_{Oin} - S_O(k)) - h^* r_{SO} - h^* K_1(S_{Osat} - S_O(k))}{h^* K_1(S_{Osat} - S_O(k))}\right]$$

(5.9)

From here, for $S_O(k+1) = S_O{}^{sp}$

$$u(k) = \frac{1}{K_2}\ln\left[\frac{S_O{}^{sp} - S_O(k) - h^*\frac{Q}{V}(S_{Oin} - S_O(k)) - h^* r_{SO} - h^* K_1(S_{Osat} - S_O(k))}{h^* K_1(S_{Osat} - S_O(k))}\right] \quad (5.10)$$

The equation for $u(k)$ represents the inverse nonlinear model of the DO concentration. This equation can be implemented using NN theory. Inputs of the NN are formed on the basis of the measured present and past values of the DO concentration plus values of the control input, set point value, inflow concentration for the DO and oxygen

uptake rate. The output of the NN inverse model is the control signal $u(k)$. The structure of the inverse model is as shown in Figure 5.3.

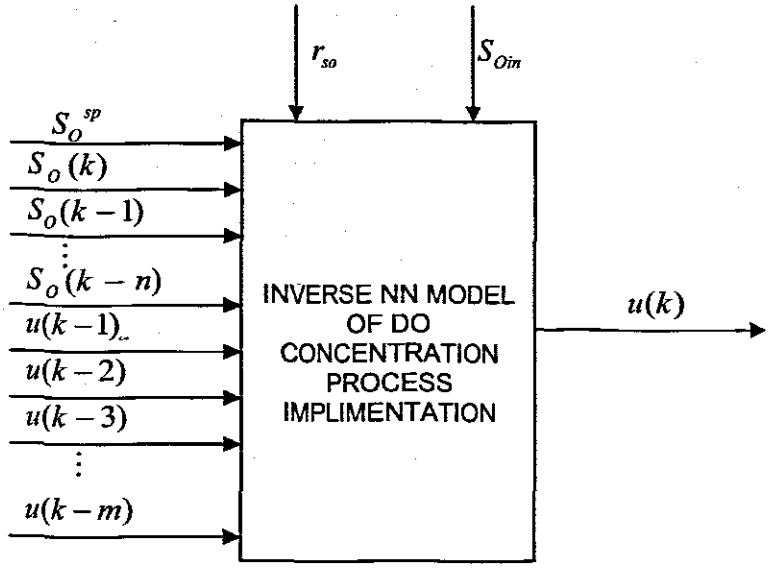


Figure 5.3: I/O NN Inverse model of the DO process for common case

5.3.3.2 Feedforward NN Inverse Model development

The data for training, validation and testing were sampled at 1.041677e-^{4} days (1min.30s) intervals with 960 data sets collected. These data sets are open loop data. 480 data sets were used for training, 240 data sets for validation (for observing the performance of the models when faced with unknown situations or dynamic changes of different amplitudes and/or frequencies) and 240 data sets for testing. The division was such that the testing set starts with the second point and takes every fourth point. The validation set starts with the fourth point and takes every fourth point, while the training set takes the remaining points of the system trajectory and control input. The data collected included the present and past values of the control input $u(k)$ and the DO concentration $S_O(k)$ as required by for identification of the inverse model. The data gathered were scaled by normalization to the range of [0 1], as i2_2 for $S_O(k)$ as the input to the network and i1_1 for $u(k)$ as the output of the network respectively. Similarly the reference was also scaled as well rso11 as the normalized oxygen uptake rate and Soin11 as the dissolved oxygen and the network was created and trained using the normalized data and simulated. Eventually the network output was un-normalized, and a linear regression between the network outputs (un- normalized) and the targets to check the quality of the network training was performed. The Matlab commands used to normalize data, split the data, perform linear regression and train the inverse feedforward network can be found in the appendix C.7:script-INVERFF.m

In this script file, **P1if** and **T1if** represent the range of the normalized inputs and target variables. **k0, k1, k2, n0** and **n1** are the delays in **i1_1** and **i1_2**. **rso11** represents the normalised oxygen uptake rate and Soin11, the dissolved oxygen concentration in the input airflow. The best structure of the identified neural model was **8-8-1**; i.e. 8 inputs consisting of the normalized reference signal **j1**, normalized forward model output, with delayed values **k0, and k1,** and the two delayed values **k0** and **k1**, of normalized airflow rate in addition to **rso11**and **Soin11** with 8 nodes in the hidden layer and 1 node at the output. The training algorithm used was the scaled conjugate gradient *trainscg.* This NN structure is used for the NN feedforward inverse model training.

Training was done in the same way as for the forward model identification by switching between the training, validation and test data set; a technique known as *'early stopping'* methodology. The training was performed in the normal way on the training set until a reasonably small error (root mean square - **RMS**) was achieved for this set. Once a reasonable and continuously decreasing **RMS** error was achieved for all the sets of data, the training was stopped automatically and the network was validated by applying different test data, not utilized before. Eventually the topology and configuration were finalized. This method was used to prevent over- training. The error on the validation set will normally decrease during the initial part of the training. However, when the network begins to over fit the data, the validation set error will start to rise. When this increase continues for a pre defined number of iterations the training is stopped and the weight values are kept. The test set is used to compare with the validation set to see if they exhibit a similar behavior. If validation errors and test errors do not show a similar behavior, this may indicate a poor division of data. In Figure 5.3, the mean squared error between the calculated values and the target training, validation and test sets are plotted against the epoch (number of iterations).

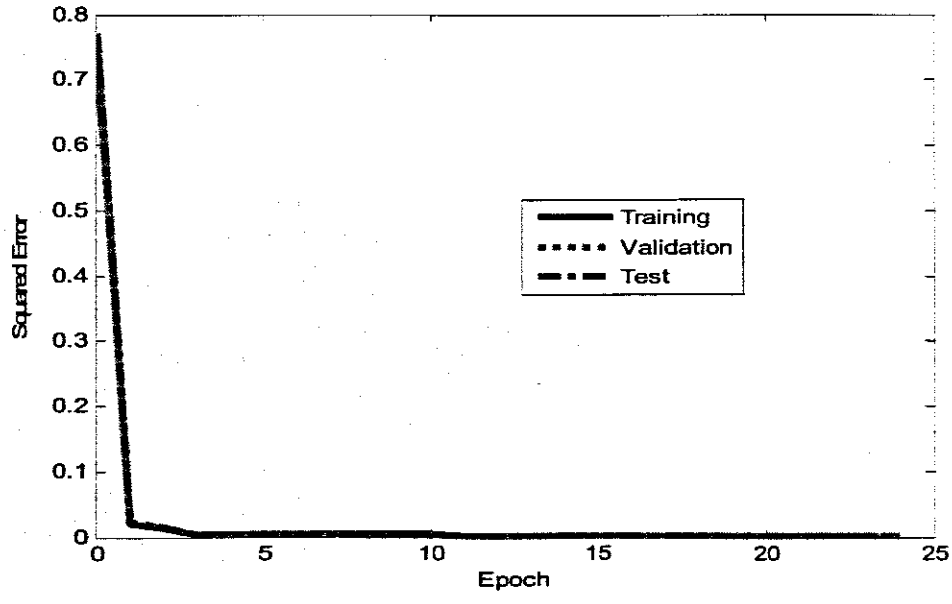


Figure 5.4: Plot of the training, validation and test errors for a generalized network.

This plot shows that with about 25 training epochs the training, validation and test errors reduces to minimum.

The results of Figure 5.4 show the plot of the mean squared error between the calculated values and the target training, validation and test sets plotted against the epoch (number of the training, validation and test errors) with the convergence performance of the MSE of $3.64254e^{-6}$ achieved for architecture with 3 hidden neurons. Analysis of the plot tells that it requires only 100 iterations to map the target when trained with *trainscg* algorithm.

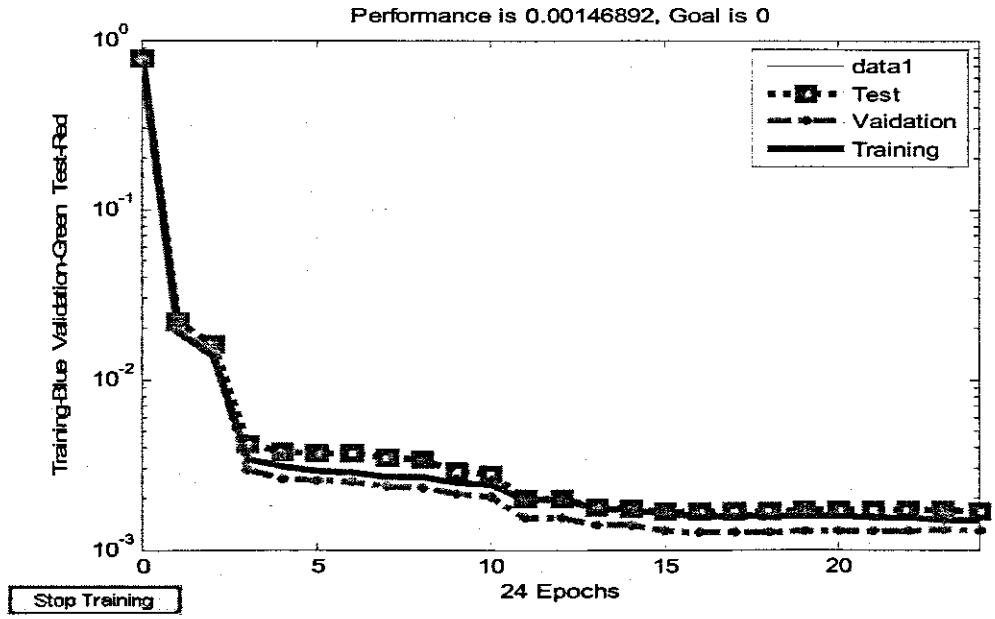


Figure 5.5: Performance MSE with 7 inputs, 1 hidden layer, 8 hidden neurons and 1 output.

The post-training analysis of the plots showing regression analyses between the network outputs and the corresponding targets (in original units) is subsequently done as illustrated in Figure 5.5.

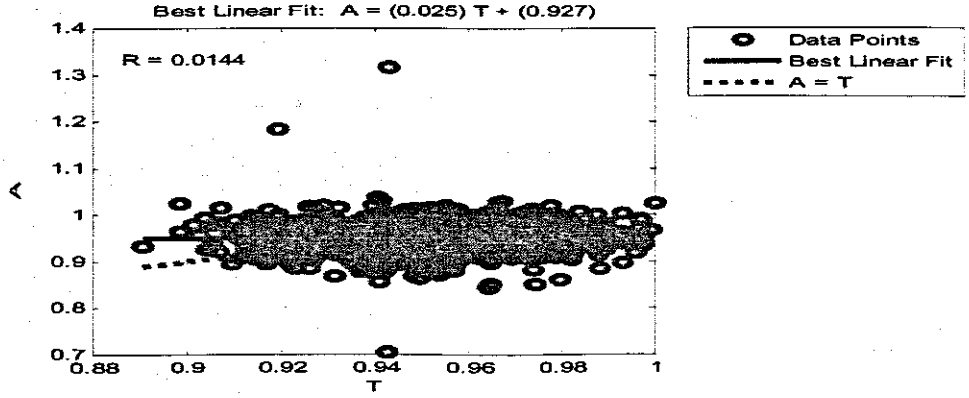


Figure 5.6: Post-Training analysis with Matlab command routine *(postreg)*

Figure 5.6 displays the plots showing regression analyses between the network outputs and the corresponding targets (in original units of data). The routine *postreg* is designed to perform this analysis. Here the network output and the corresponding targets are passed to *postreg*. It returns three parameters. The slope, and the y-intercept of the best linear regression relating targets to network outputs and the correlation coefficient (R-value) between the outputs and targets. In the analysis it can be seen that the fit is not perfect as indicated by the solid line results (outputs are not exactly equal to targets), because the slope is equal to **0.025**, and the y-intercept is **0.927**, and the R-value is also equal to **0.0144** because the fit is not good. This means that that the selected neural model does not identify of the inverse dynamics of the forward model exactly.

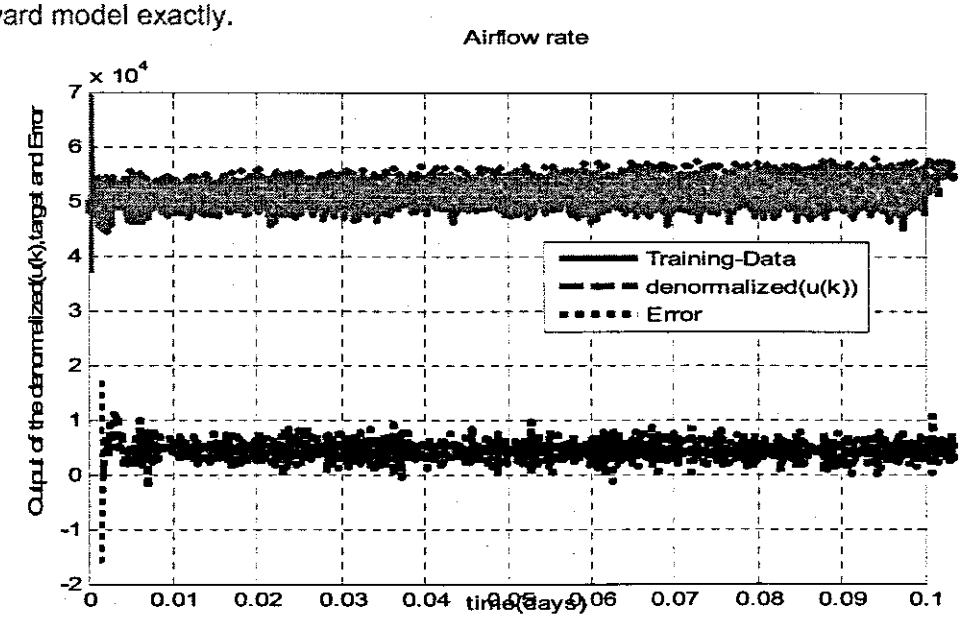


Figure 5.7: (a) De-normalized NN output trajectory of the Inverse feedforward neural model.

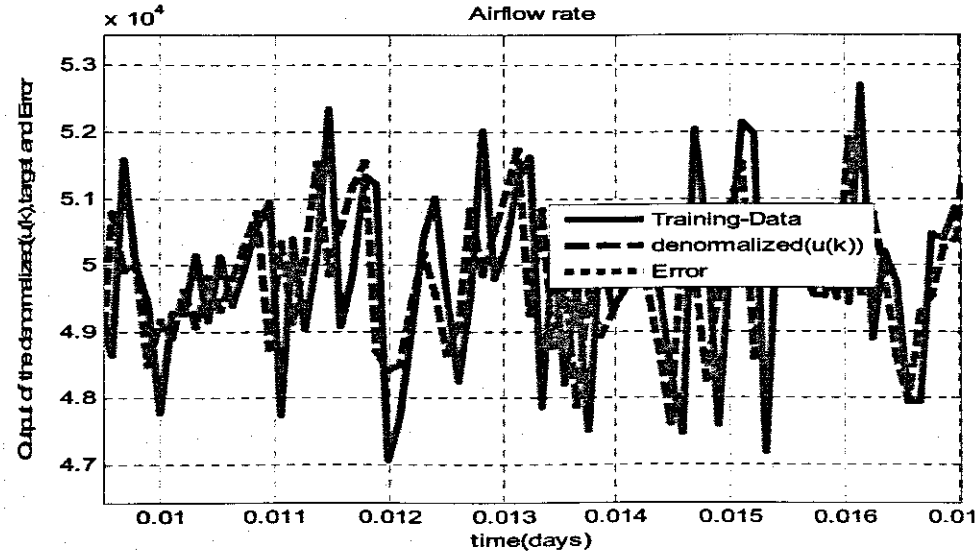


Figure 5.7: (b) De-normalized NN output trajectory of the Inverse neural model zoomed in the range (0.008-0.016) days.

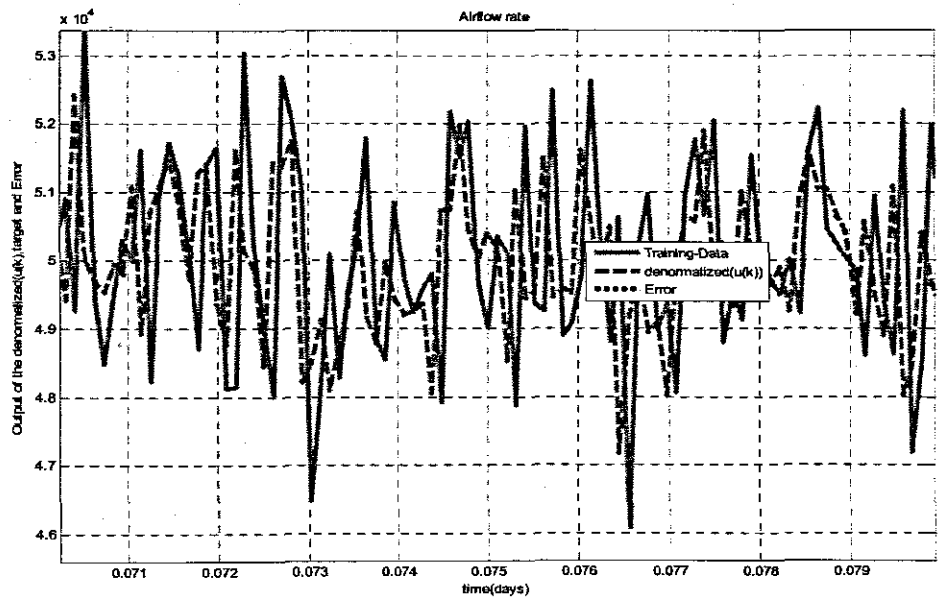


Figure 5.7 (c): De-normalized NN output trajectory of the Inverse neural model zoomed in the range (0.07-0.082) days.

Figure 5.7 shows the comparison between the de-normalized neural network output together with training data and the training data for the airflow rate. This is the closest trajectory as an approximate of the input. This therefore conforms that the results of the verification is moderately accurate. The coefficients (weights and biases) of the selected inverse model trained with the command *newff* are tabulated in the following Tables 5.1 and 5.2 respectively.

Table 5.1: Hidden layer weights and biases for the inverse model

Hidden layer weights (wif)								Hidden layer biases
inputs / nodes	ji	i2_2	no	n1	k0	k1	rsol1	bwif
1	0.5786	1.3864	2.0712	2.4239	-0.0933	-1.4307	-0.6528	5.3583
2	-3.7951	0.9564	1.4114	0.1770	1.7127	0.3367	0.2635	2.2936
3	-1.1977	-1.8641	-1.9687	1.7709	1.2627	-1.5979	-0.6042	0.4815
4	-2.1401	1.1146	-1.5476	-1.4362	1.7517	1.7461	-0.4261	-2.9119
5	-1.2051	-2.0097	1.7189	2.1615	0.4600	-0.5772	1.6436	4.0500
6	-1.3238	1.8247	-0.0580	1.6089	-1.8046	0.2274	-2.2986	2.3236
7	1.8547	0.3825	0.7962	1.6842	-1.8136	-0.5189	-2.2595	2.1345
8	-1.2390	1.7639	0.3414	1.7074	-1.9637	1.7937	-1.1506	-2.8342

Table 5.2: Output layer weights and biases for the inverse model

Output layer weights(vif)	-0.6028	0.7178	0.0075	0.5594	0.4341	0.2392	0.0591	0.0160
Output layer bias (bvif)	0.7962							

The same procedure was used to obtain the inverse model of the recurrent model and the verified results are given in the proceeding section.

5.3.3.3 Elman NN Inverse Model development

After testing different numbers of hidden layers and hidden neurons for the Elman network, a structure with 10 hidden tansig neurons and 1 linear output neuron was created to model the inverse of the DO process. The input variables to the network were selected as the normalized reference signal **j1**, the normalized DO process **i2_2** with its two time delays **n0**, **n1** and the first delay of the normalized target signal **k0**. Thus, the structure for the inverse neural model was **5-10-1**; i.e. 5 inputs, 10 hidden neurons and 1 output.

Training was done in the same way as for the feedforward inverse model by switching between the training, validation and test data sets. In Figure 5.8, the mean squared error between the calculated values and the target training, validation and test sets are plotted against the epoch (number of iterations).

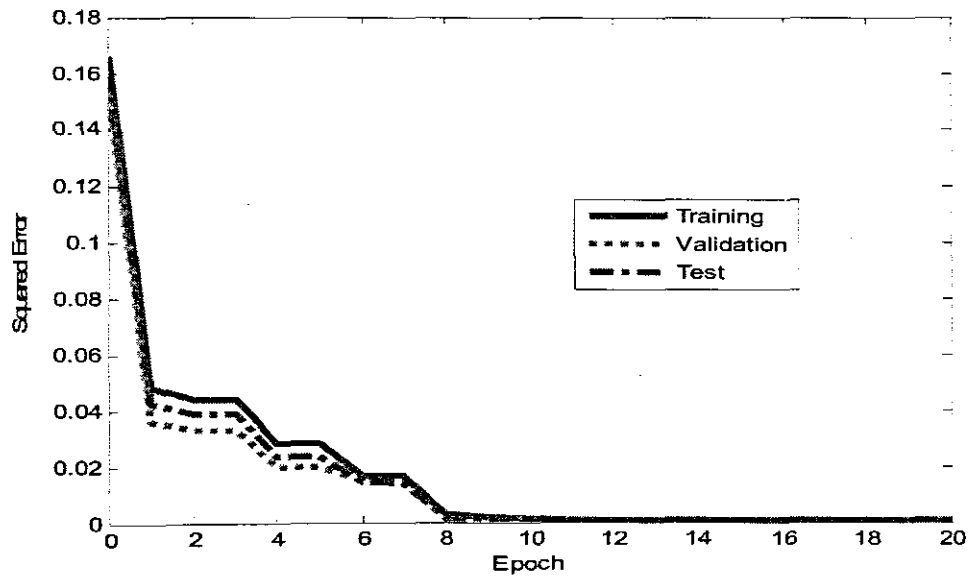


Figure 5.8: Plot of the training, validation and test errors for a generalized network.

The results of Figure 5.9 show the plot of the mean squared error between the calculated values and the target training, validation and test sets plotted against the epoch (number of the training, validation and test errors) with the convergence performance of the **MSE** of 0.000687654 achieved for architecture with 10 hidden neurons. Analysis of the plot tells that it requires only **13** iterations to approximate the target when trained with *trainscg* algorithm.

that the selected neural model does not approximate the inverse of the forward model exactly.

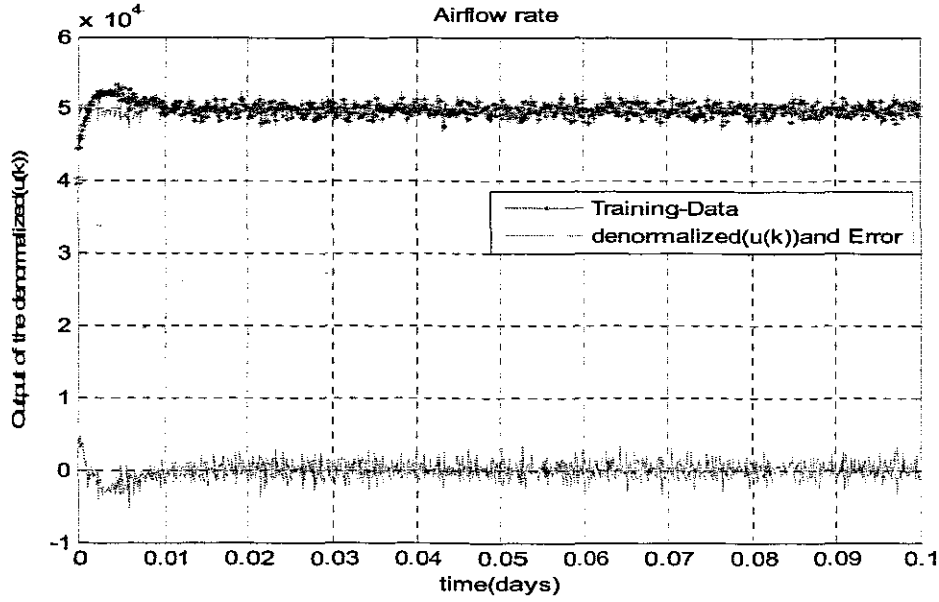


Figure 5.11: (a) De-normalized NN output trajectory of the Inverse Elman neural model.

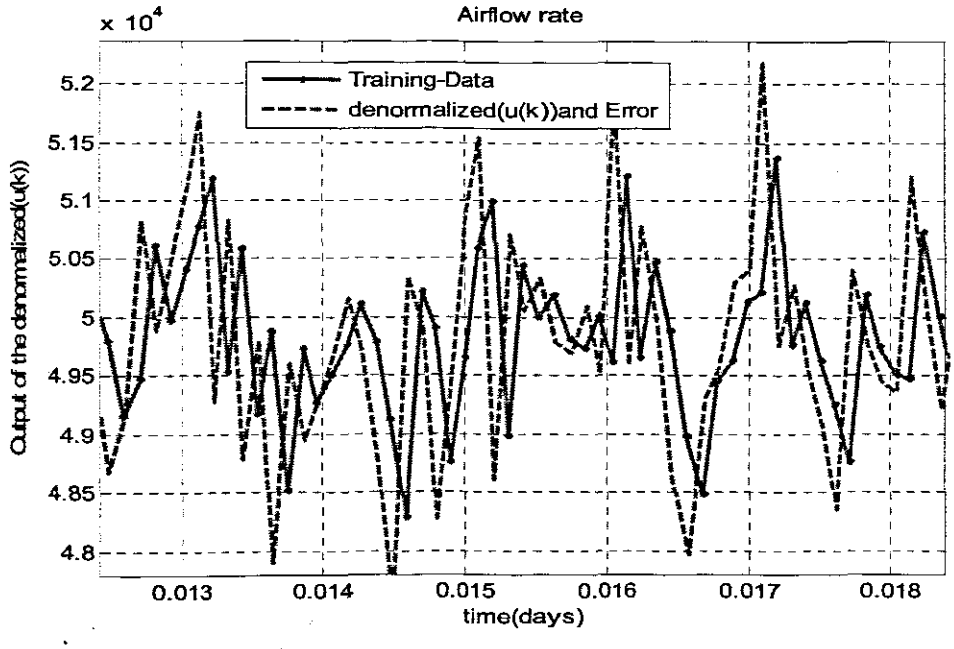


Figure 5.11: (b) De-normalized NN output trajectory of the Inverse neural model zoomed in the range (0.013-0.018) days.

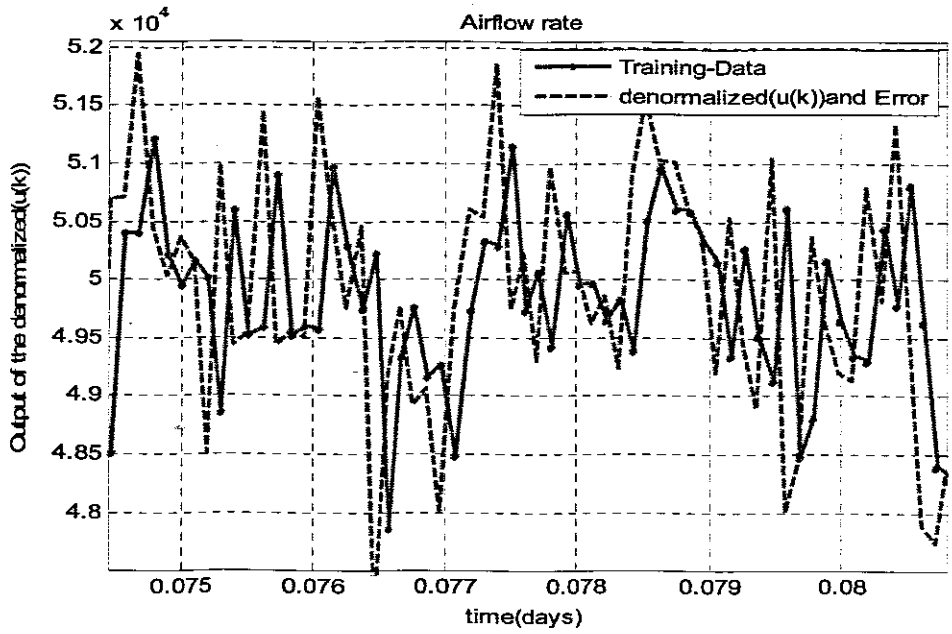


Figure 5.11 (c): De-normalized NN output trajectory of the Inverse neural model zoomed in the range (0.075-0.082) days.

Figure 5.11 shows comparison between the de-normalized neural network output together with training data and the training data for the airflow rate. This trajectory is an approximate of the control input airflow rate. This therefore conforms that the results of the verification is correct. The coefficients (weights and biases) of the selected trained inverse model trained with the command *newelm* are tabulated in the following Tables 5.3 and 5.4 respectively.

Table5.3: Hidden layer weights and biases for the inverse Elman model

Hidden layer weights									Hidden layer biases
Inputs / nodes	j1	i2_2	n0	n1	k0	k2	rso11	So11	bwie
1	0.0259	1.0573	-0.3594	0.2422	-0.0381	-1.5174	0.0052	0.0259	-1.3835
2	-0.0132	-1.2093	0.4153	0.6929	-1.1910	-0.5146	-0.0026	-0.0132	2.2933
3	0.0265	2.3953	0.9021	0.0437	0.7771	0.2636	0.0053	0.0265	-3.3242
4	-0.1008	0.2560	-0.4656	0.9709	-0.5710	-0.9379	-0.0202	-0.1008	-0.4769
5	0.0145	0.6288	-0.3021	-0.0295	-1.3047	0.2537	0.0029	0.0145	0.0998
6	-0.0271	-1.3552	0.6139	-1.2328	0.2269	-1.8230	-0.0054	-0.0271	1.6967
7	-0.0491	0.1414	0.8628	0.5006	0.7760	0.0628	-0.0098	-0.0491	-0.8513
8	0.00051	1.5158	-0.3462	1.3053	0.4735	0.5456	0.0001	0.0005	-1.0101
9	0.0406	-0.5124	-0.0855	1.0770	1.5674	1.3518	0.0081	0.0406	-2.6412
10	-0.0102	0.6046	0.2844	0.6115	1.3906	0.7530	-0.0020	-0.0102	-0.2687

Table 5.4: Output layer weights and biases for the inverse Elman model

Output bias (bvie)	0.2947	0.5457	-0.2183	-0.3960	0.3579	0.6312	0.9249	-0.1047	0.5588	-0.1392
Output weights (vie)	0.6853									

5.3.4 Discussion of the results for the two inverse models developed

The two inverse models developed; namely the Elman inverse network structure and the feedforward inverse model structure yields approximate results for the inverse problem. However, the Elman network required a large number of the hidden neurons in order to produce better results than the feedforward network. Although the training parameters set ensured a low convergence of MSE, the responses did not fit the targets exactly. This is evident on the trajectories plotted in both the Elman and the feedforward networks. Similarly, the regression analyses performed on both networks by passing the network outputs to the corresponding targets show that the fits were far from linear and thus are not very good. This can be seen from the R-values (correlation coefficients between the outputs and the targets) which are very low in both cases. However, because the target magnitude is quite big and fluctuating, the error margins are of accepted order.

Once an inverse model has been trained there are different ways in which it can be used for control. Thus the two inverse models developed are further used to implement the inverse model controller and the internal model controller in the following sections.

5.3.5 The Implementation of the Inverse Model Controller

Many control structures have been proposed, most of which are adapted from linear control. In this section direct inverse control and the internal model control structures are tested. Direct inverse control is the simplest solution for control that consists of connecting in series the inverse model of the DO process and the DO plant as can be seen in Figure 5.12. If the inverse model is accurate, then the output of the system $S_O(k)$ will follow the reference $S_O^{sp}(k)$ with the delayed samples of control and output feedback to the inverse model input.

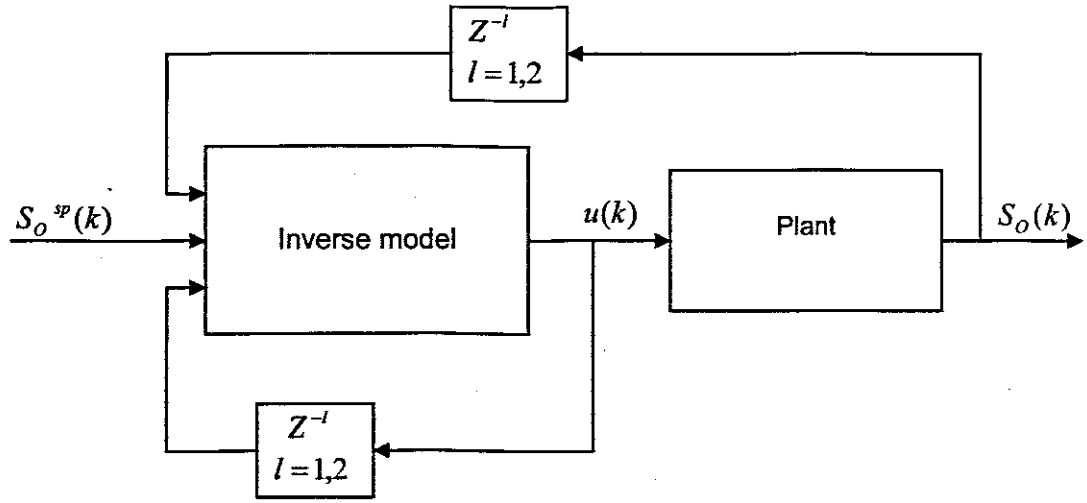


Figure 5.12: Structure for Direct inverse control

The two neural models for the recurrent and feedforward neural networks were created using Neural network toolbox residing in the Matlab environment and the structures of direct inverse control (feedforward and recurrent) were implemented in the Simulink environment as depicted in Figure 5.13 and 5.14.

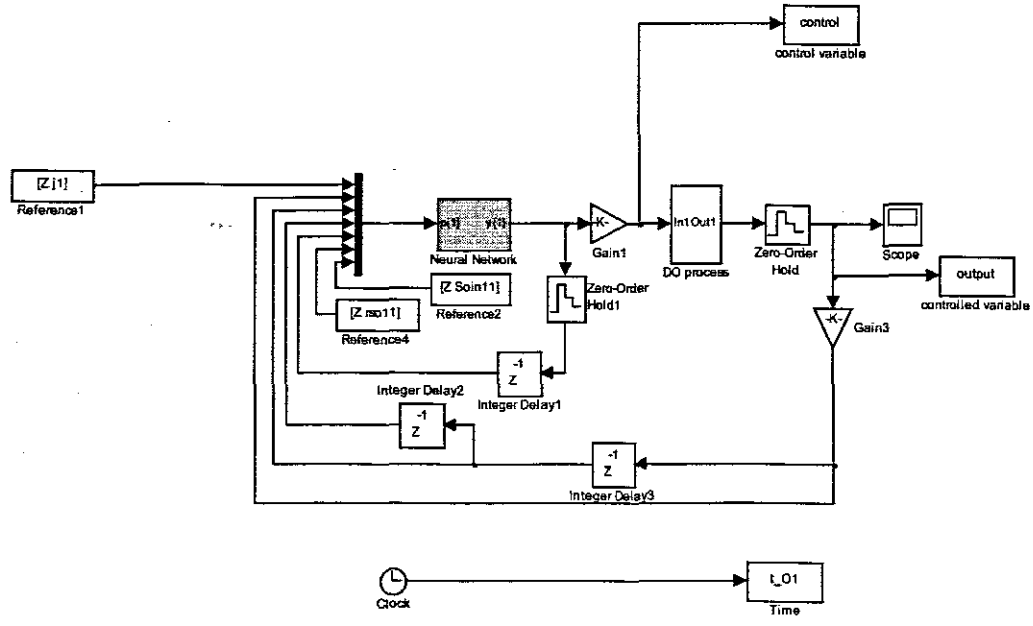


Figure 5.13: The simulation scheme of direct inverse control (Elman) structure.

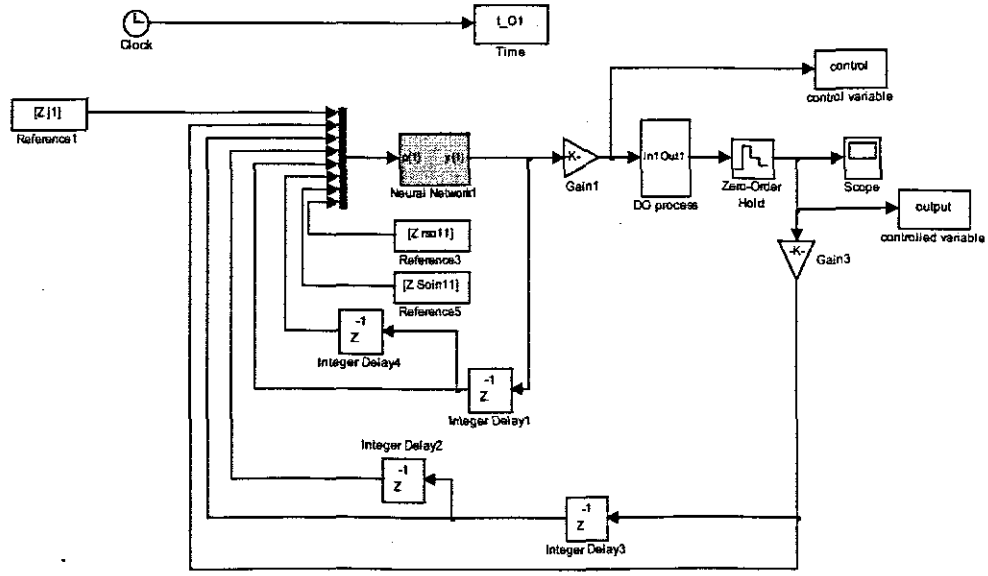
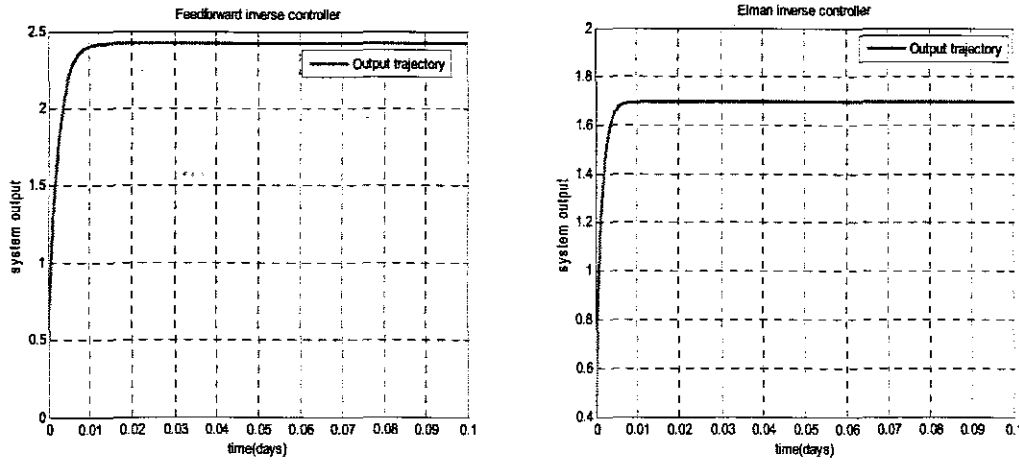


Figure 5.14: The simulation scheme of direct inverse control (Feedforward) structure

For each controller structure, off-line simulation was performed of the time responses for the control signal and system output for a constant reference value of the set-point of 2mg/l. Figure 5.15 shows the off-line simulated time responses for both structures. As can be seen the rise time for both schemes is the same. However the steady state values are different. From the output trajectory of the feedforward inverse control scheme as depicted in Figure 5.15 (a), the steady state error is -0.4, whereas for the

Elman inverse control scheme as depicted in Figure 5.15 (b), it is +0.3. This may be due to the fact that the exact inverse NN model of the plant was not achieved. The Elman network settles below the set point while the feedforward settled above the set point. The results show that this design and configuration of the NN control is not effective if it is based only on the inverse model approach.



(a)Feedforward output trajectory (b) Elman output trajectory

Figure 5.15: Time trajectories comparison of the two networks with a constant set-point

5.3.6 The Implementation of the Internal Model Controller

Once the inverse model has been trained, then a convenient way to implement an inverse neural network control technique is via a nonlinear internal model structure which is basically an extension of the linear internal model control (IMC) method. The block structure is illustrated in Figure 5.16. In this scheme, both the forward neural network model (i.e. one which predicts future outputs given previous inputs and outputs) and the inverse model (i.e. the one which predicts the required inputs given previous inputs and outputs and a desired future output) in the feedback loop are used. The relationship between the forward and inverse neural network models are as depicted in Figure 5.15. The IMC has been thoroughly examined and shown to yield transparently to robustness and stability of the closed loop system (Morari and Zafiriou, 1989, Economou *et al*, 1986). The subsystem F is usually a linear filter which can be designed to introduce desirable robustness and tracking response to the closed loop system.

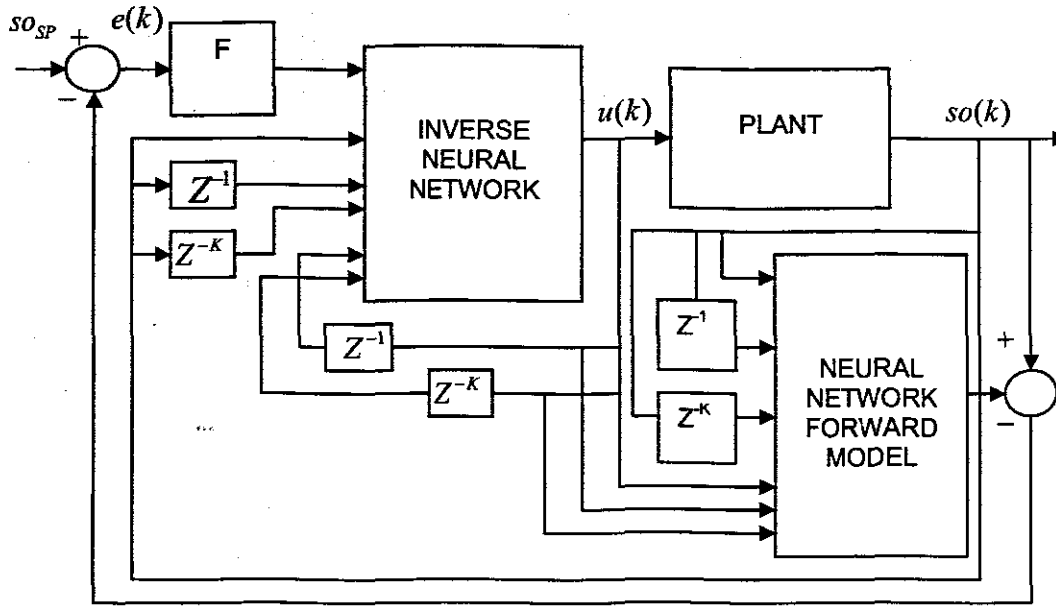


Figure 5.16: Neural network structures in internal model control (IMC) configuration

This structure is implemented in the Matlab/Simulink environment by the block diagrams of both the Feedforward and the Elman structures as illustrated in Figures 5.17 and 5.20.

5.3.6.1 The Implementation of the feedforward Internal Model Controller in Simulink

This structure is implemented in the Matlab/Simulink environment by the block diagram of the Feedforward structures as illustrated in Figure 5.17.

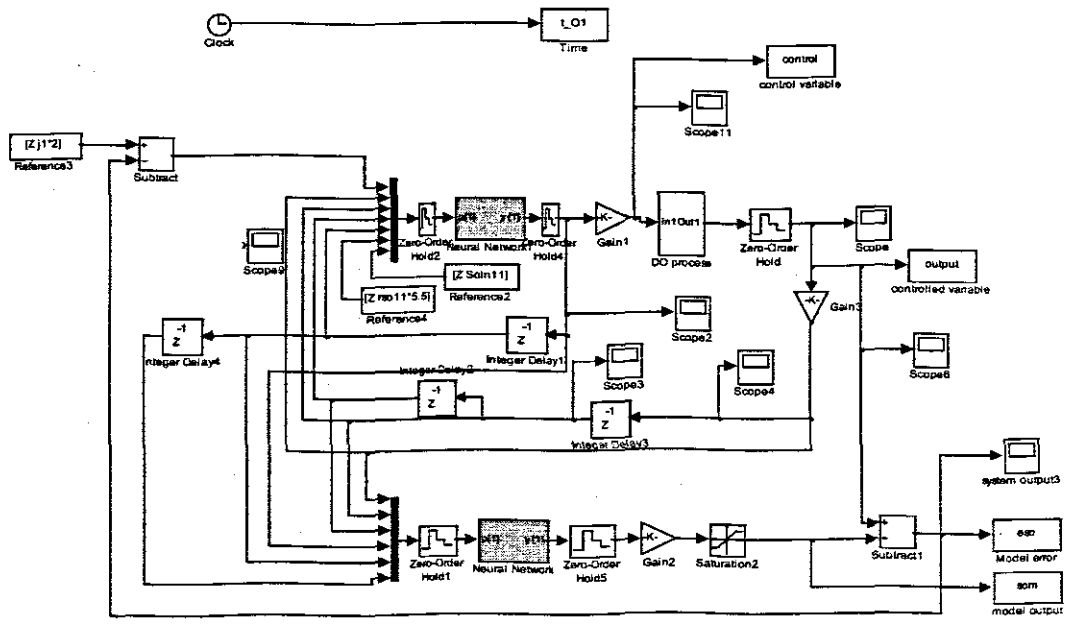
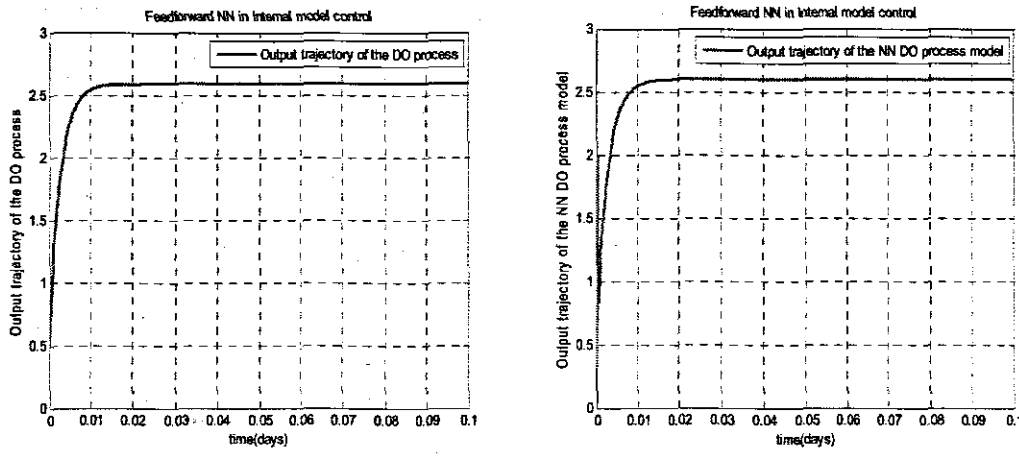


Figure 5.17: The simulation scheme of internal model control with Feedforward NN



(a)IMC DO process output trajectory (b) IMC NN DO model output trajectory

Figure 5.18: Comparison of time trajectories for the DO process and the feedforward NN DO process model in IMC structure with a constant set-point of 2mg/l.

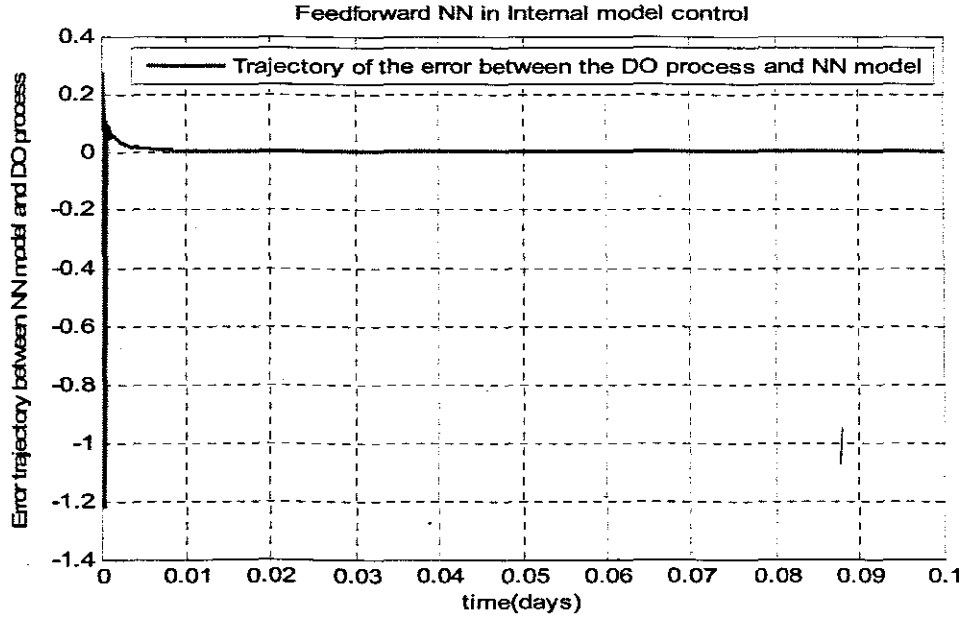


Figure 5.19: Error trajectory between the DO process and the feedforward NN DO process model in IMC structure with a constant set-point of 2mg/l.

Off-line simulation was performed of the time responses of the system output and NN DO process model for a constant reference value of the set-point of 2mg/l. Figure 5.18 shows the off-line simulated time responses for the forward and inverse structures connected in the internal model scheme. As can be seen the rise time for both schemes is the same. The trajectory of the error signal between the inverse model and the forward model is also illustrated in Figure 5.19. The steady state error tends towards zero.

5.3.6.2 The Implementation of the Elman Internal Model Controller in Simulink

This structure is implemented in the Matlab/Simulink environment by the block diagram of the Elman structures as illustrated in Figure 5.20.

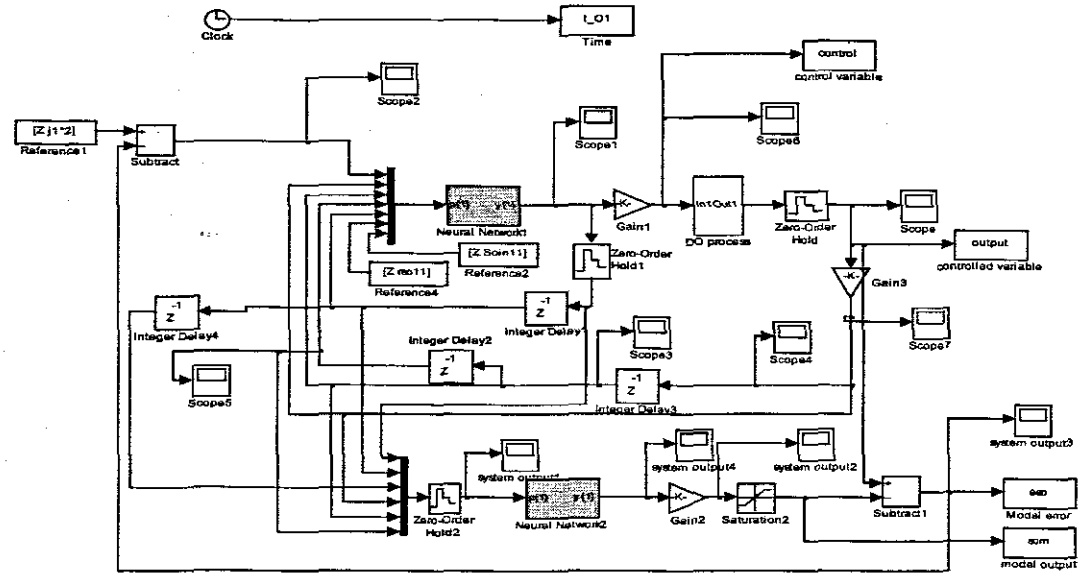
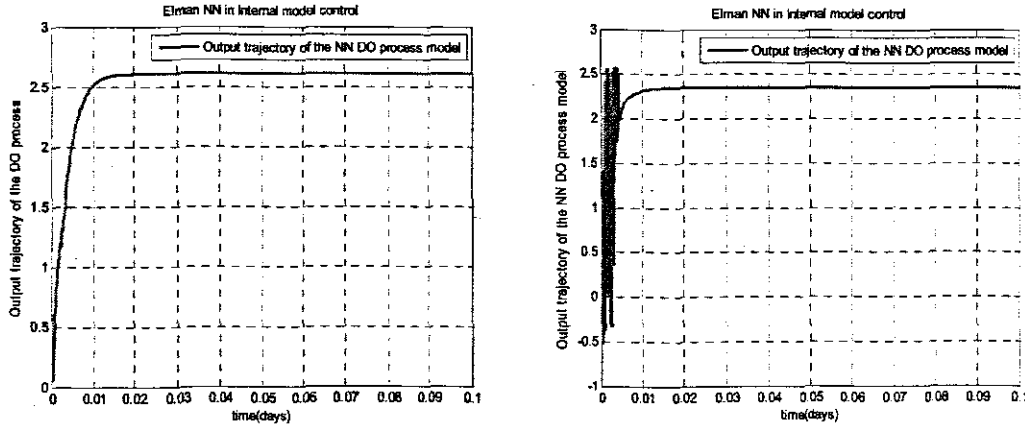


Figure 5.20: The simulation scheme of internal model control with Elman NN



(a)IMC DO process output trajectory (b) IMC NN DO model output trajectory

Figure 5.21: Comparison of time trajectories for the DO process and the Elman NN DO process model in IMC structure with a constant set-point of 2mg/l.

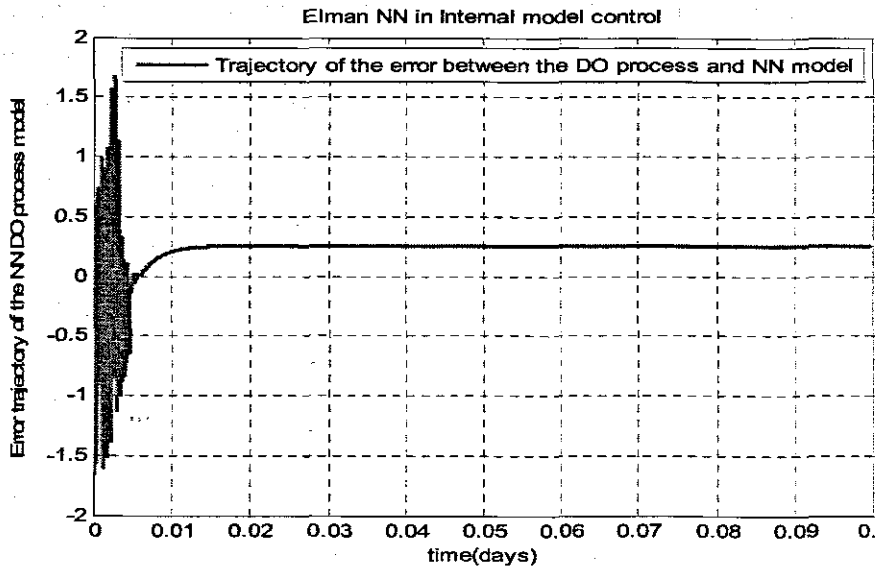


Figure 5.22: Error trajectory between the DO process and the Elman NN DO process model in IMC structure with a constant set-point of 2mg/l.

Off-line simulation was performed of the time responses of the system output and NN DO process model for a constant reference value of the set-point of 2mg/l. Figure 5.21 shows the off-line simulated time responses for the forward and inverse structures connected in the internal model scheme. As can be seen the rise time for both schemes is the same. However the NN DO process model exhibits initial oscillations which eventually die away. The trajectory of the error signal between the DO process model and the forward model is also illustrated in Figure 5.19. The steady state error 0.25

Thus the structure implemented with the feedforward networks performs better than the Elman though the required system output is not achieved for both networks simulated.

5.3.6.3 Discussion of the IMC simulation results

The results for both the internal model control structures (IMC) indicate better performance in comparison with the case of the inverse model control, but still the desired behavior of the system output can not be achieved. The filter is not implemented here. If a PI controller may have been included in the design to act as a filter, may be the results could have improved. Thus for future improvement a PI controller should be considered in the place of $e(k)$ which is usually a linear filter and can be designed to introduce desirable robustness and tracking response to the closed loop system.

5.4 Controller design for the model of the DO concentration by feedback Linearization

As mentioned previously, there are cases when adaptive linear control schemes would not perform well when faced with a highly nonlinear process. This is because the adaptive mechanism may not be fast enough to track changes in process characteristics. Appropriately designed nonlinear controller would therefore be expected to perform better.

The use of neural network model based controllers has already been mentioned. Another emerging field is that of nonlinear controller design by the use of differential geometric concepts (Isidori, 1989; Kraviaris and Kantor, 1990; Slotine and Li, 1991). Also the combination of feedback linearization and continuous-time recurrent neural networks has been studied (Nikolaou and Hanagandi, 1993).

Basically, two approaches of feedback linearization can be distinguished:

- Input-Output feedback linearization in which case, the input-output relationship of the closed loop system must be linear, and
- Input-State feedback linearization in which the relationship between the input and the states of the closed loop system must be linear.

In this work, only the first case is considered because the developed neural networks models are input-output models. A combination of neural networks based process models with input-output feedback linearization techniques is used to provide the control design. Input-Output feedback linearization is a method used to find a static state feedback control law Ψ; such that the closed-loop system has a linear input-output behavior. A schematic of this strategy is shown in Fig.5.23. In this figure the relationship between u and y is nonlinear and between y and v is linear; furthermore the relationship between u and v, u and x is also nonlinear where v is a reference input for the nonlinear controller.

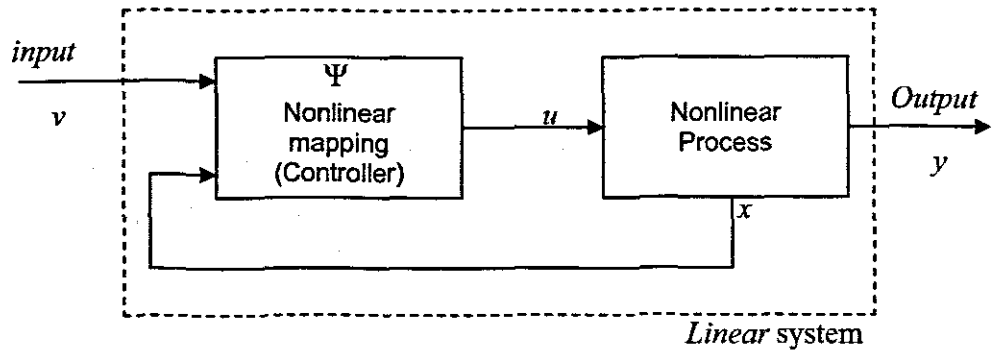


Figure.5.23. Input-Output feedback linearization

In the case of continuous-time systems, there exists a number of ways for the design of feedback linearization for affine systems. Affine systems are those in which the input u appears linearly in the state-space equation. Such a system can be represented as:

$$\dot{x}(t) = f(x,t) + g(x,t)u(t) \tag{5.6}$$

$$y(t) = h(x,t)$$

Where, $x \in \Re^n$ is the state vector, $u \in U \subset \Re^m$ is the input vector, and $y \in Y \subset \Re^p$ is output of the system. $f(x)$ and $h(x)$ are smooth vector fields and $g(x,t)$ is an $n \times m$ matrix whose columns are smooth vector fields having continuous partial derivatives of any required order.

5.4.1 Theory on Input-Output Feedback Linearization

To start with, consider the following analytic non affine MIMO discrete-time nonlinear system:

$$\dot{x}(t) = f(x(t)) + g(x(t))u(t) \qquad x(0) = x_0$$

$$y(k) = h(x,t) \tag{5.7}$$

Where, $x \in X$ is the state vector, $u \in U$ is the input vector, and $y \in Y$ is output of the system. X is the allowed values of x, U is the allowed values of u, and Y is the allowed values of y, f, g and h are smooth vector fields and g is an $n \times m$ matrix whose columns are smooth vector fields having continuous partial derivatives of any required order. The objective of input-output linearization is to obtain a nonlinear control in the form:

$$u = p(x) + q(x)v \tag{5.8}$$

Where p and q are variable parameters in such away that the resulting closed loop control system is linear and results in a linear transfer function between y and v given in Laplace domain by:

$$\frac{y(s)}{v(s)} = \frac{1}{\beta_r s^r + \ldots\ldots + \beta_1 s + \beta_0} \tag{5.9}$$

Where, r is the relative degree of the nonlinear system. Assume that the system given by Equation (5.7) has an equilibrium point x_o, u_o, i.e. $f(x_o, u_o) = x_o$. The input-output linearization is based on the idea of a relative degree of the process under control. The relative degree is the smallest order of derivatives of the output y_i for which the output depends explicitly on $u_j; j = \overline{1,m}$. The relative degree at operating point x_o is defined by the integer r which satisfies:

$$L_g L_f^{i-1} h(x) = 0 \ \forall \, i < r \text{ and } x \text{ near } 0$$
$$L_g L_f^{i-1} h(x) \neq 0 \ \forall \ x \text{ near } 0 \tag{5.10}$$

Where, L_g and L_f are the Lie derivatives defined as:

$$L_f h(x) = \sum_{i=1}^{n} f_i(x) \frac{\partial h}{\partial x_i}(x) \tag{5.11}$$

$$L_g L_f h(x) = \sum_{i=1}^{n} g_i(x) \frac{\partial L_f h}{\partial x_i}(x) \tag{5.12}$$

While the higher order Lie derivatives can be written as:

$$L_f^a h(x) = L_f (L_f^{a-1} h)$$

(5.13)

The time derivatives of the system output can be expressed as algebraic function of these Lie derivatives as follows:

$$\frac{dy}{dt} = L_f h(x)$$
$$\vdots$$
$$\frac{d^{r-1} y}{dt^{r-1}} = L_f^{r-1} h(x) \tag{5.14}$$
$$\frac{d^r y}{dt^r} = L_f^r h(x) + L_g L_f^{r-1} h(x) u$$

The proceeding equations show that the relative degree represents how many times the output must be differentiated with respect to time to explicitly recover the input u. From Equation5.9 and 5.14 the feedback control law can be expressed by the following transformations:

1. Express v from Equation 5.9 as

$$v(s) = (\beta_r s^r + ... + \beta_r s + \beta_0) y(s) \tag{5.15}$$

and transform it under zero initial conditions to time domain, then

$$v(t) = \beta_r \frac{d^r y}{dt^r} + ... + \beta_1 \frac{dy}{dt} + \beta_0 y(t) \tag{5.16}$$

2. Express the higher order derivative as $\frac{d^r y}{dt^r}$ as follows

$$\beta_r \frac{d^r y}{dt^r} = v(t) - \beta_{r-1} \frac{d^{r-1} y}{dt^{r-1}} + - .... - \beta_1 \frac{dy}{dt} - \beta_0 y(t) \tag{5.17}$$

3. Expressing the derivatives of $y(t)$ by their Lie derivatives according to equations 5.14, then

$$\beta_r [L_f^r h(x) + L_g L_f^{r-1} h(x) u(t)] = v(t) - \beta_{r-1} L_f^{r-1} h(x) ..... - \beta_1 L_f h(x) - \beta_0 y(t) \tag{5.18}$$

4. Expression of $u(t)$ from the above equation gives

$$u(t)=\frac{v(t)-\beta_r L_f{}^r h(x)-\beta_{r-1}L_f{}^{r-1}h(x).....-\beta_1 L_f h(x)-\beta_0 y(t)}{\beta_r L_g L_f^{r-1}h(x)} \tag{5.20}$$

where $y(t)=h(x)$. From the last equation 5.20 the expression for $u(t)$ can be written in short as:

$$u(t)=\frac{v(t)-\sum_{k=0}^{r}\beta_k L_f^k h(x)}{\beta_r L_g L_f^{r-1}h(x)} \tag{5.21}$$

...

Similar derivations can be written for the discrete time systems.

The above approach is applied for the design of NN nonlinear linearizing controller for the DO concentration. The considerations are in discrete time domain.

5.4.2 NN Nonlinear Controller design for the model of DO by I/O Linearization of the closed loop system

The process for the design of the nonlinear linearising controller is based on the following:

- Selection of a linear reference model as a desired model for the closed loop linearized system of the form described in the following way:

$$x(k+1)=ax(k)+bv(k) \qquad x(0)=x_0 \tag{5.22}$$

- Comparison of the desired and the DO model and derivation of the expression for the DO controller.

5.4.2.1 Representation of the DO model in a standard affine form

The model of the DO concentration process is nonlinear and the discrete-time dynamics are expressed as:

$$S_O(k+1)=S_O(k)+h^*\left\{\frac{Q}{V}\left[S_{O_{in}}(k)-S_O(k)\right]+K_L a(u(k))\left[S_{O\,sat}(k)-S_O(k)\right]+r_{SO}(k)\right\} \tag{5.23}$$

For $K_L a(u(k))=K_1(1-e^{-K_2 u(k)})$ the equation 5.23 becomes

$$\begin{aligned}&=S_O(k)+h^*\frac{Q}{V}S_{Oin}-h^*\frac{Q}{V}S_O(k)+h^*K_1(S_{Osat}-S_O(k)-\\&-h^*K_1e^{-K_2u(k)}(S_{Osat}-S_O(k))+h^*r_{so}\end{aligned} \tag{5.24}$$

If it is introduced that $u1(k)=e^{-K_2u(k)}$, Equation 5.24 can be written as

$$\begin{aligned}&S_O(k+1)=S_O(k)-h^*\frac{Q}{V}S_O+h^*K_1(S_{Osat}-S_O(k)-\\&-h^*K_1(S_{Osat}-S_O(k))u1(k)+h^*\frac{Q}{V}S_{Oin}+h^*r_{so}\end{aligned} \tag{5.25}$$

From here the standard affine presentation of Equation 5.23 is

$$S_O(k+1) = f(S_O(k)) + g(S_O(k))u1 + g_1(S_O(k))S_{O,in}(k) + g_2(S_O(k))r_{SO}(k) \quad (5.26)$$

Where, h^* is the sampling time and all the other terms have been defined in Chapter 2.

$$f(S_O(k)) = S_O(k)[1 - h^* \frac{Q}{V}] + h^* K_1 (S_{Osat} - S_O(k) \quad (5.27)$$

$$g(S_O(k)) = h^* [S_{O\,sat}(k) - S_O(k)]^* K_1 \quad (5.28)$$

$$g_1(S_O(k)) = h^* \frac{Q}{V} = \text{Constant} \quad (5.29)$$

$$g_2(S_O(k)) = h^* = \text{Constant} \quad (5.30)$$

$$u1(k) = e^{-K_2 u(k)} \quad (5.31)$$

The expression for the control function $u(k)$ is obtained as:

$$u(k) = -\frac{1}{K_2} \ln(u1(k)) \quad (5.32)$$

A block diagram of the linearized closed loop system is shown in Figure 5.24. Thus using the nonlinear model $u(k)$ can be easily obtained where additionally some type of a linear controller designed to control the linearized by the nonlinear controller closed loop system in order to follow the set point S_O^{sp} can be used.

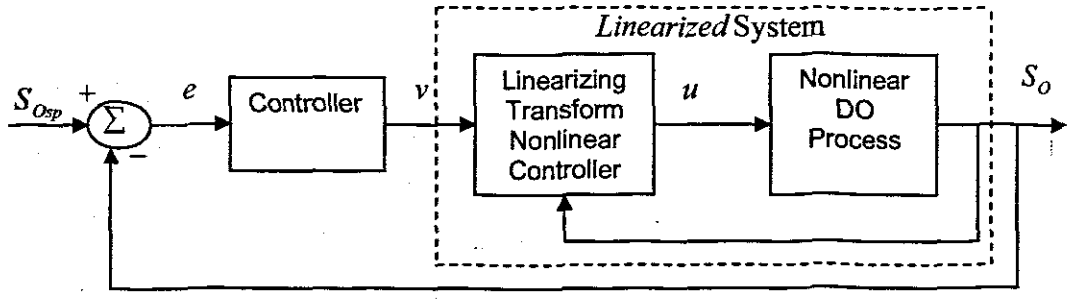


Figure 5.24: Basic idea with the nonlinear linearizing controller

5.4.2.2 Calculation of the linearizing controller

Feedback linearization is a common method for controlling certain classes of nonlinear processes. In this section the DO process control system applying the Feedback linearization technique that is based on a neural network model of the process is derived. The input to the DO process is the manipulated airflow rate $u(k)$ and the output is the DO concentration $S_O(k)$.

Knowing the relative degree, the theory of input-output feedback linearization which has been presented in the previous section is used to design the controller. A first order desired linear model is derived describing the DO process. The equation of the desired process is:

$$S_{O\,desired}(k+1) = aS_{Odesired}(k) + bv(k),\ S_O(0) = S_0 \tag{5.33}$$

Where a and b are the parameters of the desired linear model. The parameter a is selected in such a way that the desired model is stable and has desired behavior. By feedback linearization technique the closed loop system containing the neural network controller and the nonlinear DO process are required to behave like the linear state-space model given by equation (5.33). Thus, equating (5.26) with (5.33) the following feedback linearising control law is deduced.

$$\begin{aligned} &S_O(k+1) = S_{O\,desired}(k+1) \\ &= aS_O(k) + bv(k) = f(S_O(k)) + g(S_O(k))u1 + g_1 S_O(k)S_{Oin}(k) + g_2(S_O(k))r_{so}(k) \end{aligned} \tag{5.34}$$

From which the following feedback linearizing control law $u1(k)$ is deduced as:

$$u1(k) = \frac{aS_O(k) + bv(k) - f(S_O(k)) - g_1(S_O(k))(S_{Oin}(k)) - g_2 S_O(k)(r_{so}(k))}{g(S_O(k))} \tag{5.35}$$

This is the model of the linearizing controller.

Selecting the virtual control $v(k)$, as an appropriate linear combination of past outputs plus the reference enables an arbitrary assignment of the closed-loop poles of the linearized by the nonlinear controller system. Thus, Feedback linearization is a nonlinear counterpart to pole placement with all zeros canceled (see Astrom & Wittenmark, 1995).

5.4.2.3 NN Nonlinear controller design for the model of DO by I/O Linearization

In this section the NN nonlinear controller is designed based on the Feedback linearizing controller given by equation (5.35) derived in the preceding section. It can be seen that this equation describes an input/output process with inputs $v(k), S_O(k), S_{Oin}(k),$ and $r_{so}(k)$ and the output is $u1(k)$. Equation 5.35 is used further to calculate the real nonlinear control $u(k)$. In this way the NN linearizing controller has two parts; the NN and a nonlinear function connected in series as illustrated in Figure 5.25.

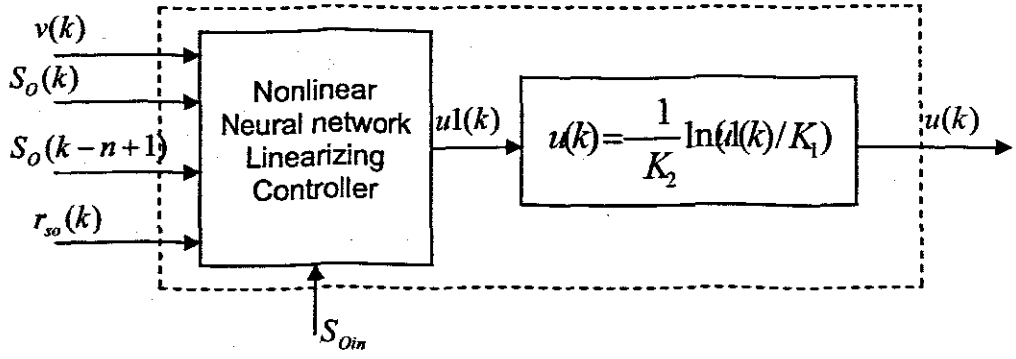


Figure 5.25: NN nonlinear controller.

A NN controller model can be estimated from the historical input-output data by letting neural networks approximate the equation 5.35. The data for the reference input function $v(k)$, and $S_O(k), S_{Oin}(k), r_{so}(k)$ used for approximating the functions are obtained from the closed loop response of the desired reference model trajectory controlled by a linear PI controller illustrated by the block diagram of Figure 5.26.

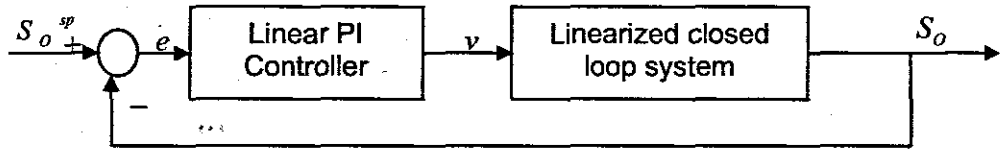


Figure 5.26: A closed loop system designed as hybrid one.

The NN linearizing controller is designed as a discrete time controller. The DO process is continuous process and in order to be close to its nature the closed loop system is designed as a hybrid one, where the linearized closed loop is considered as continuous one and the linear PI controller is designed as continuous one. Transformation of the signals between continuous and discrete time components of the closed loop system is implemented by the zero-order holds.

5.4.3 Linear controller design

Pole placement technique is used to choose suitable parameters of a PI regulator for the desired linear model. In the DO process control application considered in this thesis, the objective is to maintain the DO output at a desired set-point despite the presence of disturbances. Consequently a linear controller, a PI controller is applied (McLellan et al., 1990) as:

$$v = K_P\left((S_O{}^{sp} - S_O(t)) + \frac{1}{T_I}\int_{\tau=0}^{t}(S_O{}^{sp} - S_O(t))d\tau\right) \tag{5.36}$$

Where the gain K_P, and the time constant T_I, are additional controller tuning parameters. The transfer function for a continuous-time PI regulator is:

$$V(s) = K_P\left(1+\frac{1}{T_I s}\right)E(s) \tag{5.37}$$

Where $V(s)$ and $E(s)$ are the Laplace transforms of the controller output and the error signal. The closed loop block diagram of the desired reference model is shown in Figure 5.27.

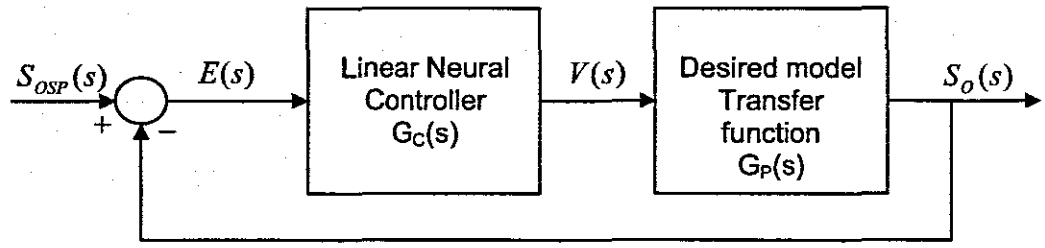


Figure 5.27: Block diagram of the desired closed loop system with PI regulator

As explained earlier, the desired model is of first order whose dynamics are:

$$S_{O\,desired}(k+1) = aS_O(k) + bv(k)$$

Assuming continuous- time dynamics, the Laplace transfer function

$$G_P(s) = \frac{S_O}{V}(s) = \frac{b}{s-a} \tag{5.38}$$

By combining Equations 5.38 and 5.39 the closed loop transfer function from S_{OSP}, *to* S_O is found to be:

$$\frac{S_O(s)}{S_{Osp}(s)} = \frac{K_P(T_I s+1)b}{T_I s^2 + T_I(K_P - a)s + K_P} \tag{5.39}$$

The characteristic equation is then found to be:

$$T_I s^2 + sT_I(K_P b - a) + K_P b = 0 \tag{5.40}$$

This equation is of second order form. To calculate suitable parameter values for gain K_P, and the time constant T_I, the closed loop poles are selected according to the requirements towards the performance specifications. In this case the following specifications are required for the desired linear system:

- The closed loop system must be stable.
- The dissolved oxygen dynamics to have minimum overshoot and quick rise-time,
- Zero steady state error.

Thus in order to achieve the specified performance characteristics, the closed loop poles are selected as complex conjugates with values of: $s_1, s_2 = -2 \pm j\sqrt{2}$ .Real poles are also a possible selection but are not considered here.

Comparing the poles of equation 5.40 with the desired pole location yields:

$$(s+2-j\sqrt{2})(s+2-j\sqrt{2}) = T_I s^2 + sT_I(K_P - a) + K_P \tag{5.41}$$

Comparing the coefficients of s yields two equations for the tuning parameters K_P, and T_I, as:

$$\begin{aligned} K_P &= 6T_I \\ 6T_I - a &= 4 \end{aligned} \tag{5.42}$$

From these two equations the coefficients of the PI controller are obtained as:

$$K_P = 4 + aT_I$$
$$T_I = \frac{(4+a)b}{6} \tag{5.43}$$

Then the equation of the PI controller can be written as;

$$v(t) = (4+a)[e(t) + \frac{1}{b(4+a)/6}\int e(t)dt] \tag{5.44}$$

Thus from equation 5.44 it is seen that only one parameter a of the desired DO state space equation is chosen for the desired pole location. Using the results obtained from the preceding analyses, the closed loop desired model is simulated in Matlab environment using the Simulink block diagram illustrated in Figure 5.27(a) where the zero order hold has been incorporated in order to discretize the trajectories with sampling time of 1.041667e-4 (1min.30s). 960 samples for each variable of interest are obtained. These samples are used for the neural network controller training.

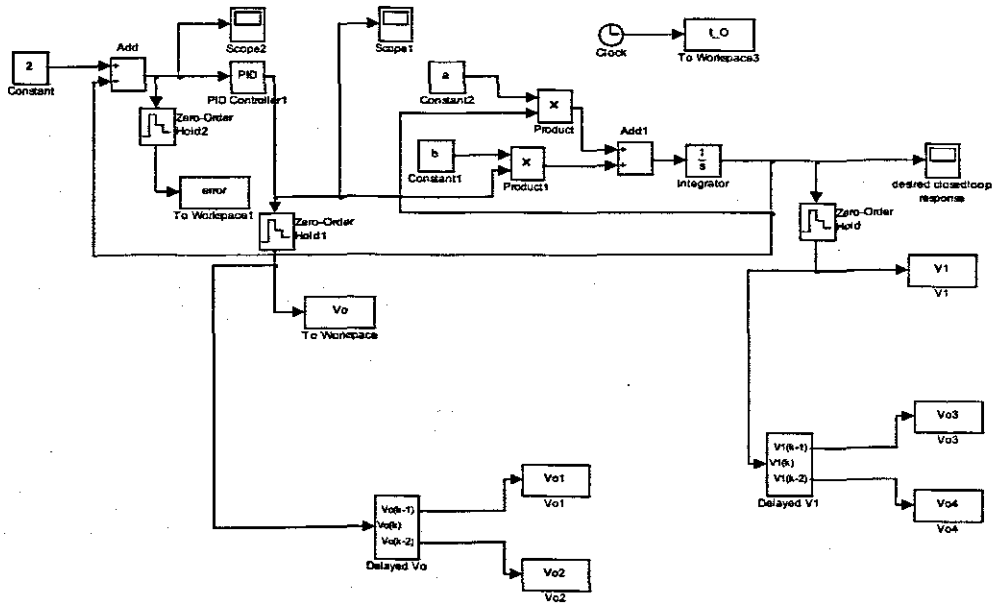


Figure 5.28(a): Simulink block diagram of the desired closed loop system with PI regulator.

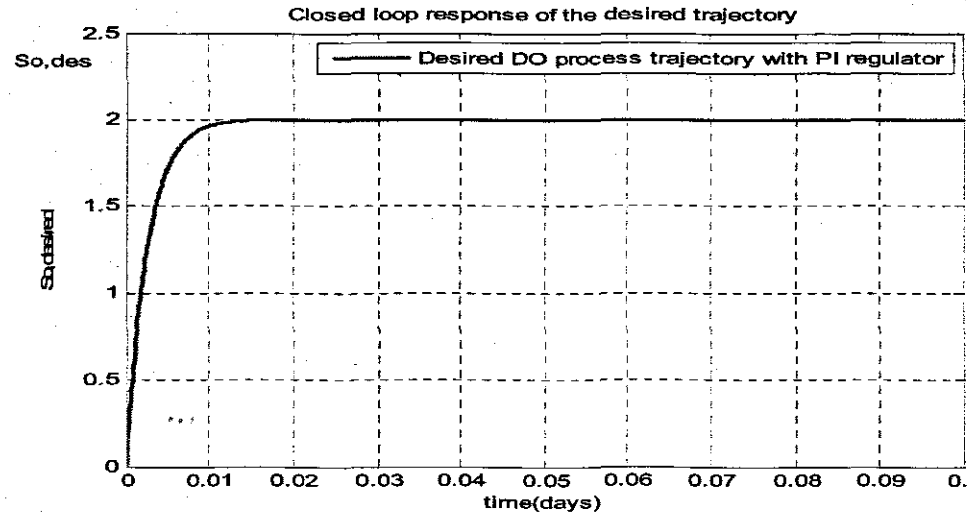


Figure 5.28(b): Trajectory of the desired closed loop system with PI regulator.

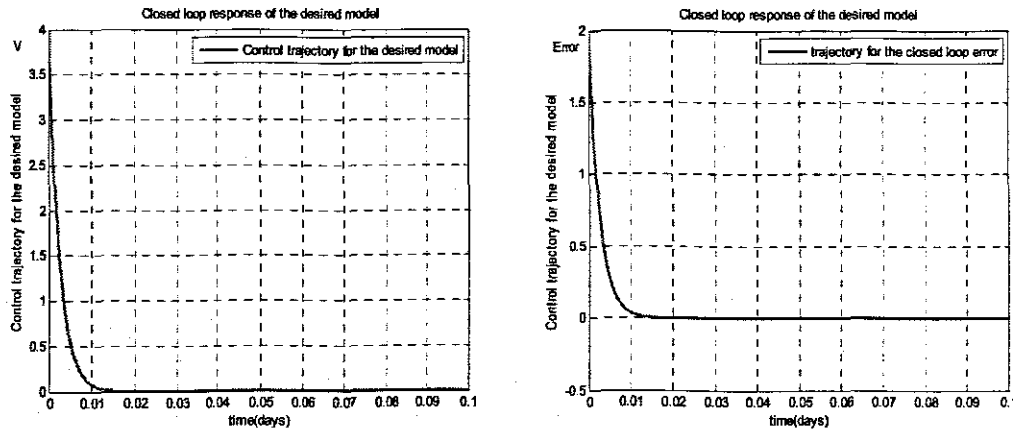


Figure 5.28(c): Controller and Error Trajectory of the desired closed loop system with PI regulator.

Figures 5.28 (b) and (c) illustrate the trajectories of the closed loop, error and the controller responses of the desired linear model controlled with a PI regulator. Data collected from these trajectories are used to design the neural controller depicted in equation 5.35.

5.4.3.1 Calculation of the function $u1$

From equation 5.35 the trajectory of $u1(k)$ is derived by simulation of the various functions to obtain the training data for the neural network. The trajectory of $u1(k)$ is as illustrated in Figure 5.29. This is achieved from the commands in the appendix c11: script file-parametersforu1.m derived from Equation 5.35: where all the parameters have been defined.

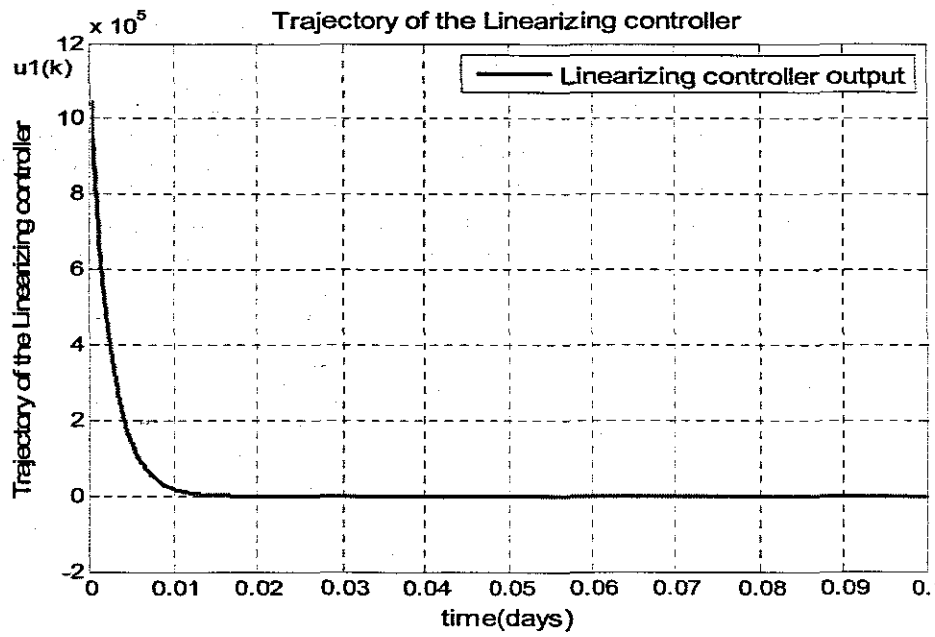


Figure 5.29: Trajectory of the model of the Linearizing controller.

Here also the data for the desired $S_O(k)$ and $v(k)$ for training, validation and testing were sampled at 1.041677e-^{4} days (1min.30s) intervals with 960 data sets collected from the simulation runs. 480 data sets were used for training, 240 data sets for validation (for observing the performance of the models when faced with unknown situations or dynamic changes of different amplitudes and/or frequencies) and 240 data sets for testing. The division was such that the testing set starts with the second point and takes every fourth point. The validation set starts with the fourth point and take every fourth point, while the training set takes the remaining points. The data collected included the present and two past values of the desired output $S_O(k)$ and the linear controller output variable $v(k)$ and the linearizing controller model output $u1(k)$ as required by for identification of the model for the linearizing controller in addition to the constants $r_{so}(k)$ and $S_{Oin}(k)$ for the inputs to the neural network. The data gathered for $u1(k)$ were scaled by normalization to the range of [0 1], as $u1\_1$ and that for $r_{so}(k)$ also in the range [0 1] as $r_{so}11$ and the network was created and trained using the normalized data and simulated. Eventually the network output was un-normalized, and a linear regression between the network outputs (un- normalized) and the targets to check the quality of the network training was performed. The closed loop block diagram for the linearizing control of the DO concentration is shown in Figure 5.30.

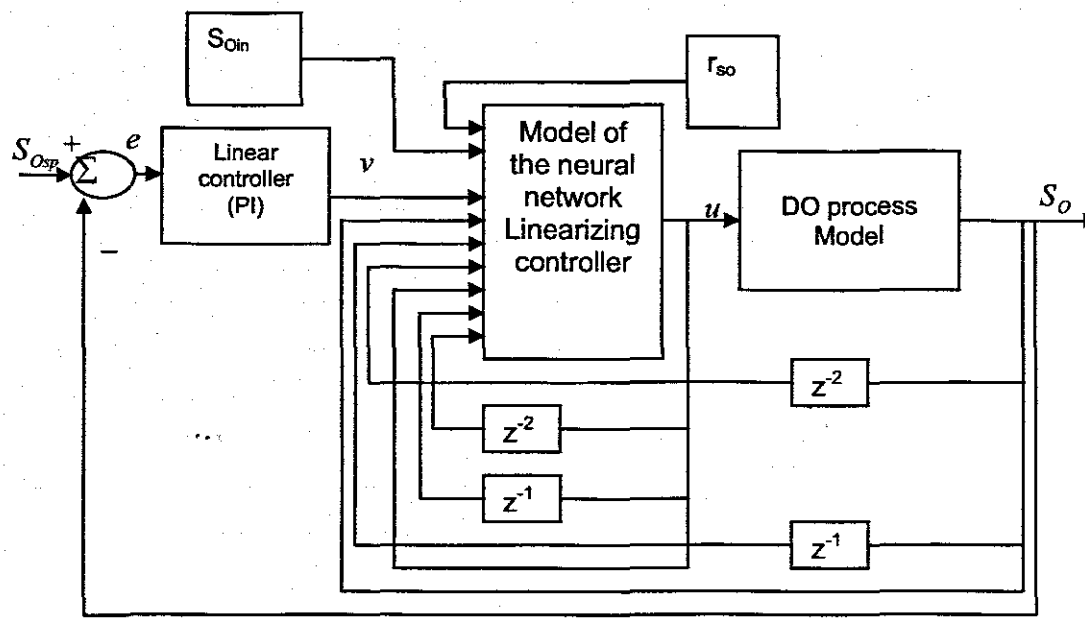


Figure 5.30: The closed loop block diagram for the linearizing control of DO concentration.

The following algorithm was used to train the model of the linearizing controller.

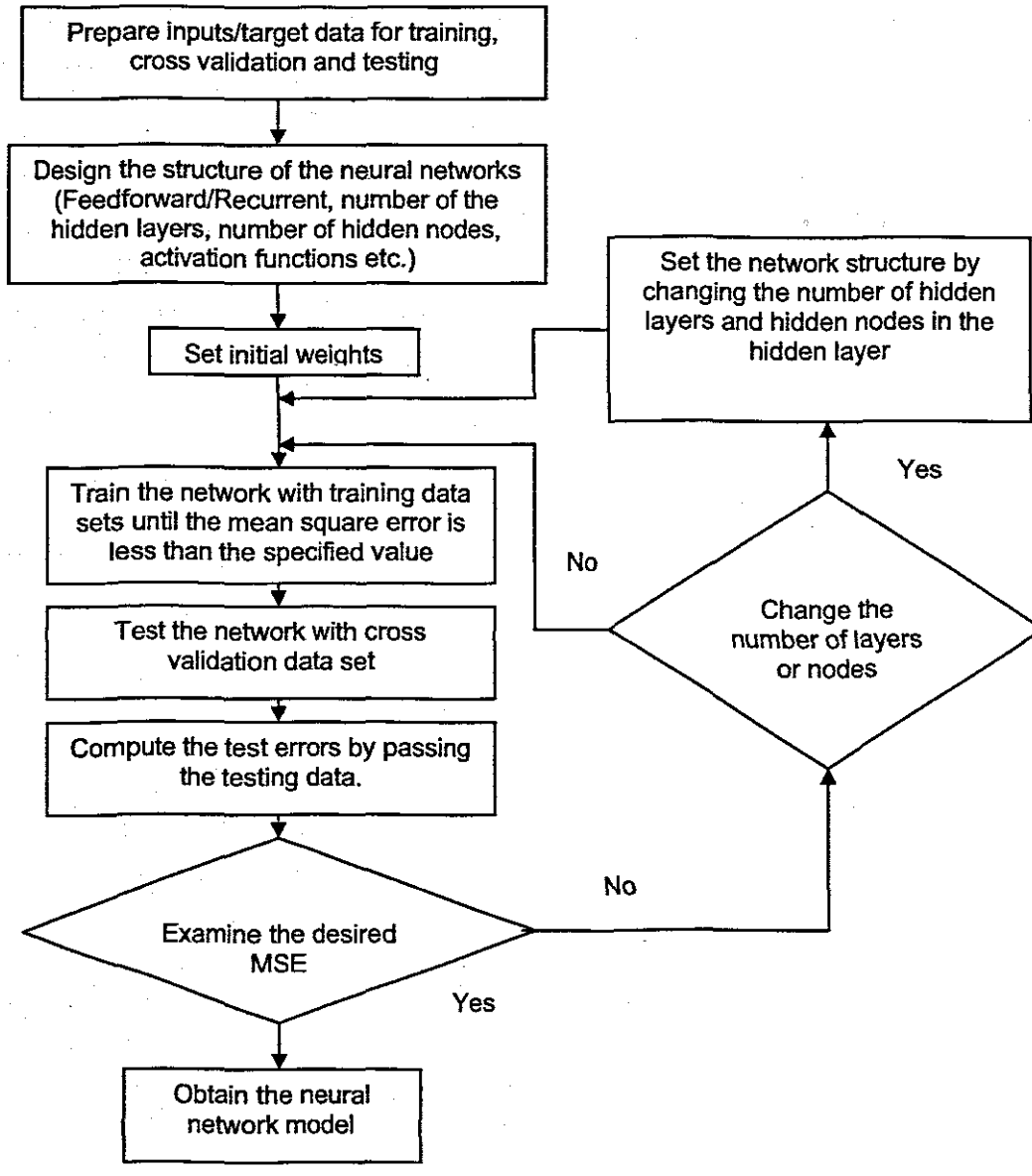


Figure 5.31: The algorithm used to train the model of the linearizing controller.

5.5 Elman NN Nonlinear linearizing controller model realization

The structure of the Elman network was constituted as follows: Pc and Tc represent the range of the normalized inputs and target variables Vo_11 V1_11, K0i, rso11, Soin11, and u1_11. rso11 is the normalized oxygen uptake rate and Soin11 is the dissolved oxygen concentration in the input airflow ,and , K0i is the delay in the desired dissolved oxygen concentration in $S_o(k)$. The best structure of the identified neural model was 5-7-1; i.e. 5 inputs of the normalized variables, 7 nodes in the hidden layer and 1 node at the output. The training algorithm used was the scaled conjugate

gradient *trainscg*. The Matlab commands used to normalize data, split the data, perform linear regression and train the Elman neural network as per the algorithm of Figure 5.31 are contained in the script file in appendix c 10: nonlinearconelm.m

Training was done in the same way as for the forward model identification by switching between the training, validation and test data set; a technique known as *'early stopping'* methodology. The training was performed in the normal way on the training set until a reasonably small error (root mean square - RMS) was achieved for this set. Once a reasonable and continuously decreasing RMS error was achieved for all the sets of data, the training was stopped automatically and the network was validated by applying different test data, not utilized before. Eventually the topology and configuration were finalized. This method was used to prevent over- training. The error on the validation set will normally decrease during the initial part of the training. However, when the network begins to over fit the data, the validation set error will start to rise. When this increase continues for a pre defined number of iterations the training is stopped and the weight values are kept. The test set is used to compare with the validation set to see if they exhibit a similar behavior. If validation errors and test errors do not show a similar behavior, this may indicate a poor division of data. In Figure 5.32, the mean squared error between the calculated values and the target training, validation and test sets are plotted against the epoch (number of iterations).

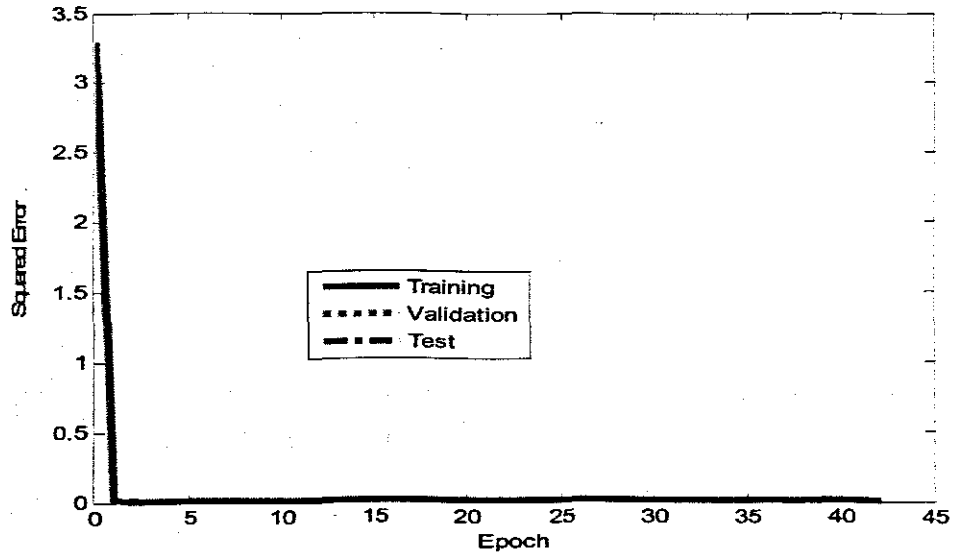


Figure 5.32: Plot of the training, validation and test errors for a generalized network.

The results of Figure 5.32 shows the plot the mean squared error between the calculated values and the target training, validation and test sets plotted against the epoch (number of the training, validation and test errors) with the convergence performance of the MSE of $8.19516e^{-9}$ achieved for architecture with 7 hidden neurons.

Analysis of the plot tells that it requires only 100 iterations to map the target when trained with *trainscg* algorithm.

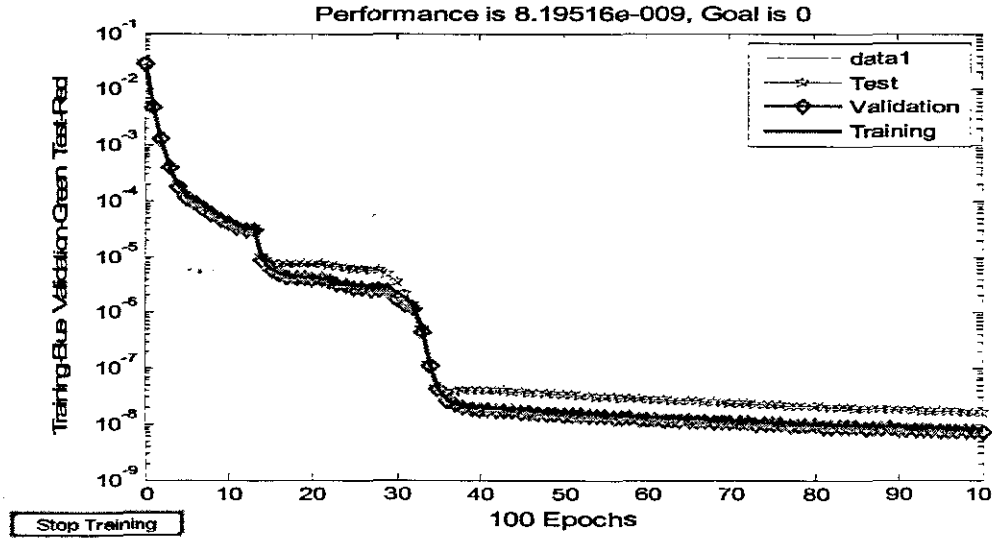


Figure 5.33: Performance MSE with 5 inputs, 1 hidden layer, 7hidden neurons and 1 output.

The post-training analysis of the plots showing regression analyses between the network outputs and the corresponding targets (in original units) is subsequently done as illustrated in Figure 5.34.

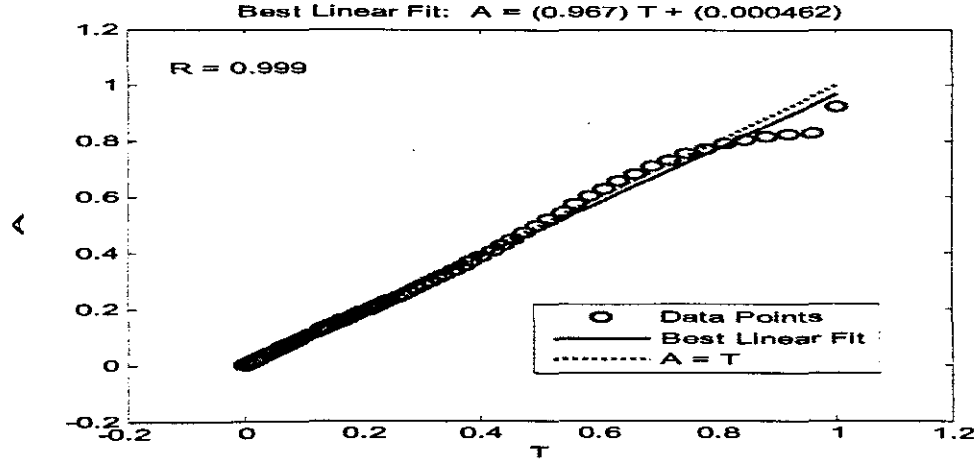


Figure 5.34: Post-Training analysis with Matlab command routine *(postreg)*

Figure 5.34 displays the plots showing regression analyses between the network outputs and the corresponding targets (in original units of data). The routine *postreg* is designed to perform this analysis. Here the network output and the corresponding targets are passed to *postreg*. It returns three parameters. The slope, and the y-intercept of the best linear regression relating targets to network outputs and the correlation coefficient (R-value) between the outputs and targets. In the analysis it can

be seen that a perfect fit indicated by the solid line results (outputs are exactly equal to targets), because the slope is equal to **0.981**, and the y-intercept is **0.0179**, and the R-value is also equal to **0.999** because the fit is good. This therefore confirms that the selected neural model is a good identifier of the NN model of the linearizing controller.

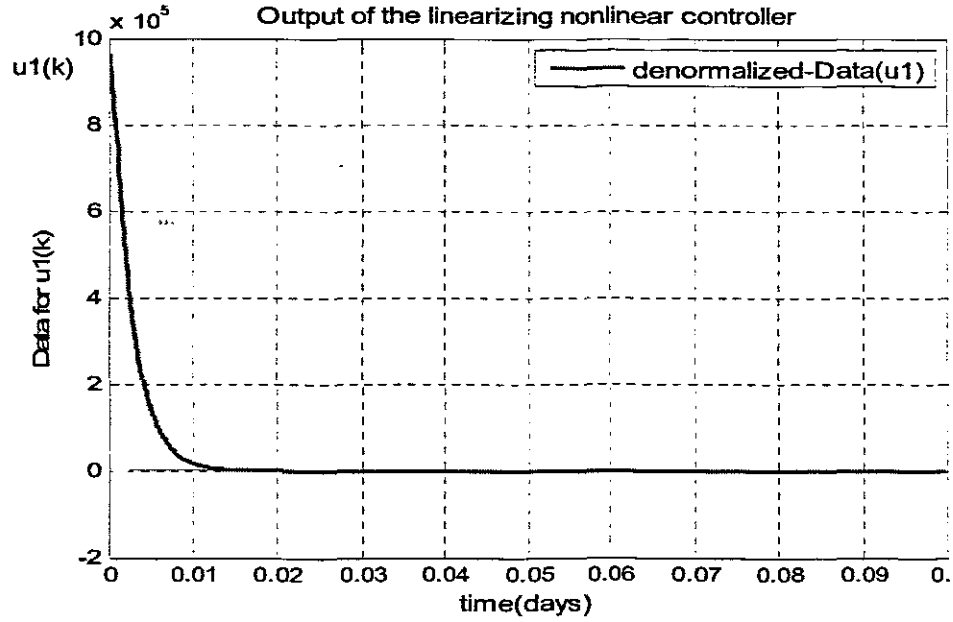


Figure 5.35:De-normalized NN output trajectory of the model of the nonlinear controller.

Figure 5.35 shows the de-normalized neural network output. This trajectory is an approximate of the output model of the linearizing controller derived from equation 5.35 for the trajectory of $u1(k)$. This conforms that the results of the verification is correct. The coefficients (weights and biases) of the selected model with the command *newelm* are tabulated in the following Tables 5.5 and 5.6 respectively.

Table5.5: Hidden layer weights and biases

Hidden layer weights (wc)						Hidden layer biases
Inputs / nodes	Vo_11 v(k)	V1_11 So(k)	K0i So(k-1)	rso11	Soin11	(bwc)
1	-0.0485	1.5340	0.1962	0.0222	0.0044	0.9107
2	-1.4293	-0.1752	-0.5193	0.3604	0.0721	2.9122
3	-0.7601	-1.3914	1.3399	-0.1071	-0.0214	0.8416
4	-0.1516	-1.0417	0.3693	0.1479	0.0296	0.6626
5	0.7284	-1.5844	1.2408	-0.2421	-0.0484	-0.0621
6	0.1273	-0.6790	0.7106	0.0087	0.0017	1.0904
7	-1.6022	-0.0120	0.6872	0.0415	0.0083	-1.1544

Table 5.6: Output layer weights and biases

Output weights (vc)	0.1953	0.7444	-0.5152	1.0066	0.2776	0.4574	0.5120
Output bias(bvc)	-0.5120						

5.5.1 Feedforward NN Nonlinear linearizing controller model realization

The structure of the Feedforward network was constituted with the same inputs and the same output as the Elman network as follows: **Pc and Tc** represent the range of the normalized inputs and target variables **Vo_11 V1_11, K0i, rso11, Soin11, and u1_11. rso11** is the normalized oxygen uptake rate and **Soin11** is the dissolved oxygen concentration in the input airflow ,and , K0i is the delay in the desired dissolved oxygen concentration in $S_O(k)$. The best structure of the identified neural model was 5-5-1; i.e. 5 inputs of the normalized variables, 7 nodes in the hidden layer and 1 node at the output. The training algorithm used was the scaled conjugate gradient *trainscg*. The Matlab commands contained in the script file - nonlinearconff.m in appendix c.9: were used to normalize data, split the data, perform linear regression and train the feedforward neural network as per the algorithm of Figure 5.31.

Training was done in the same way as for the Elman network by switching between the training, validation and test data set; a technique known as *'early stopping'* methodology. The training was performed in the normal way on the training set until a reasonably small error (root mean square - RMS) was achieved for this set. Once a reasonable and continuously decreasing RMS error was achieved for all the sets of data, the training was stopped automatically and the network was validated by applying different test data, not utilized before. Eventually the topology and configuration were finalized. This method was used to prevent over- training. The error on the validation set will normally decrease during the initial part of the training. However, when the network begins to over fit the data, the validation set error will start to rise. When this increase continues for a pre defined number of iterations the training is stopped and the weight values are kept. The test set is used to compare with the validation set to see if they exhibit a similar behavior. If validation errors and test errors do not show a similar behavior, this may indicate a poor division of data. In Figure 5.36, the mean squared error between the calculated values and the target training, validation and test sets are plotted against the epoch (number of iterations).

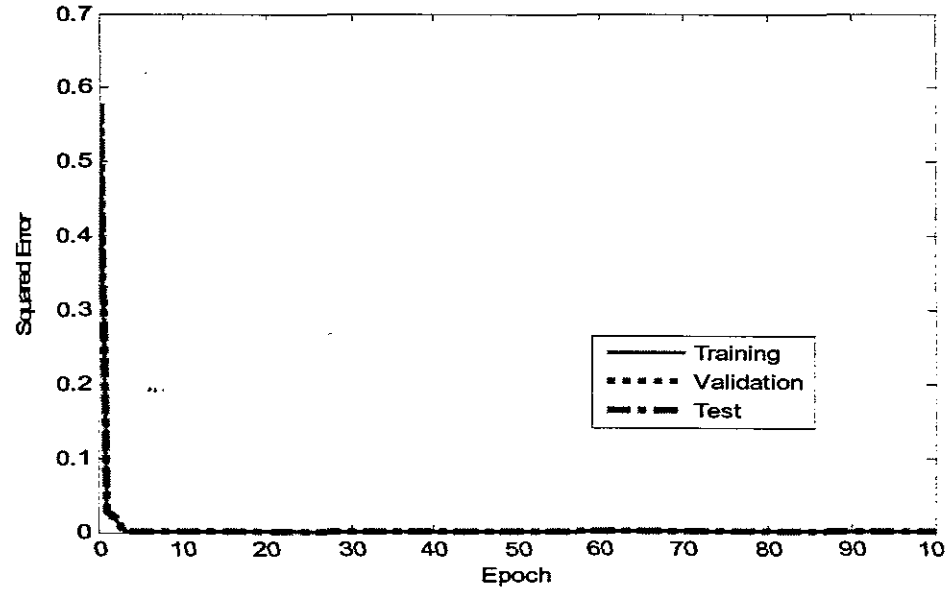


Figure 5.36: Plot of the training, validation and test errors for a generalized network.

The results of Figure 5.37 shows the plot the mean squared error between the calculated values and the target training, validation and test sets plotted against the epoch (number of the training, validation and test errors) with the convergence performance of the MSE of $5.60993e^{-6}$ achieved for architecture with 5 hidden neurons. Analysis of the plot tells that it requires only 49 iterations to map the target when trained with *trainscg* algorithm.

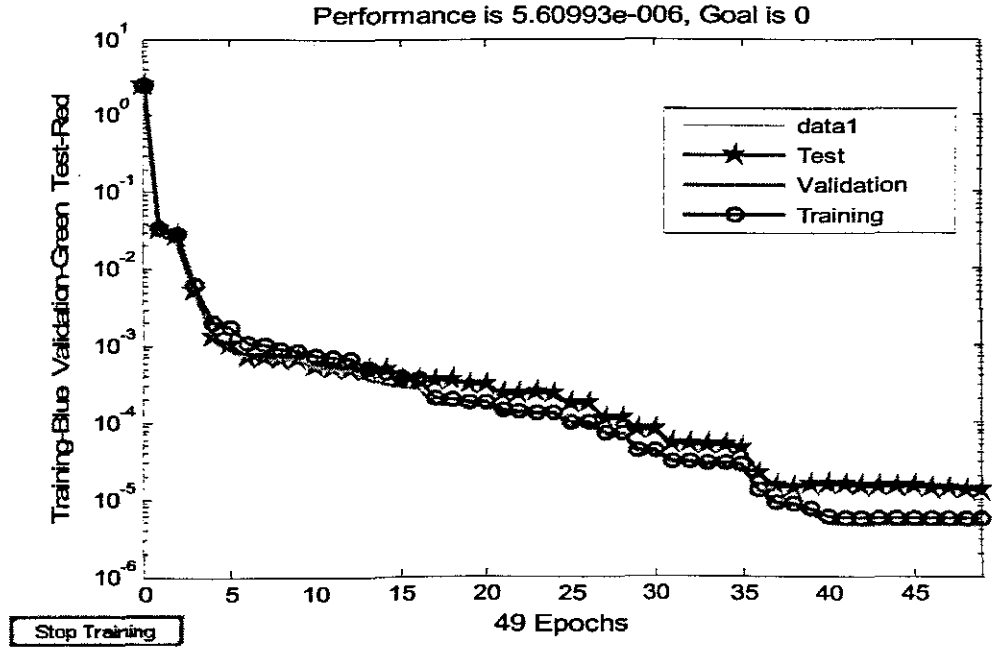


Figure 5.37: Performance MSE with 5 inputs, 1 hidden layer, 5 hidden neurons and 1 output.

The post-training analysis of the plots showing regression analyses between the network outputs and the corresponding targets (in original units) is subsequently done as illustrated in Figure 5.38.

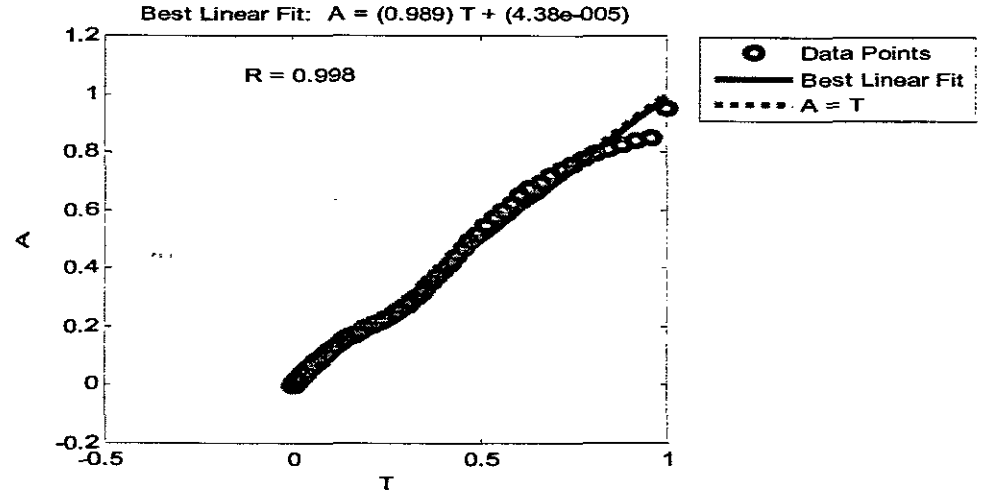


Figure 5.38: Post-Training analysis with Matlab command routine *(postreg)*

Figure 5.38 displays the plots showing regression analyses between the network outputs and the corresponding targets (in original units of data). The routine *postreg* is designed to perform this analysis. Here the network output and the corresponding targets are passed to *postreg*. It returns three parameters. The slope, and the y-intercept of the best linear regression relating targets to network outputs and the correlation coefficient (R-value) between the outputs and targets. In the analysis it can be seen that a perfect fit indicated by the solid line results (outputs are exactly equal to targets), because the slope is equal to **0.989**, and the y-intercept is **4.989e-005**, and the R-value is also equal to **0.998** because the fit is good. This therefore confirms that the selected feedforward neural model is a good identifier of the linearizing controller.

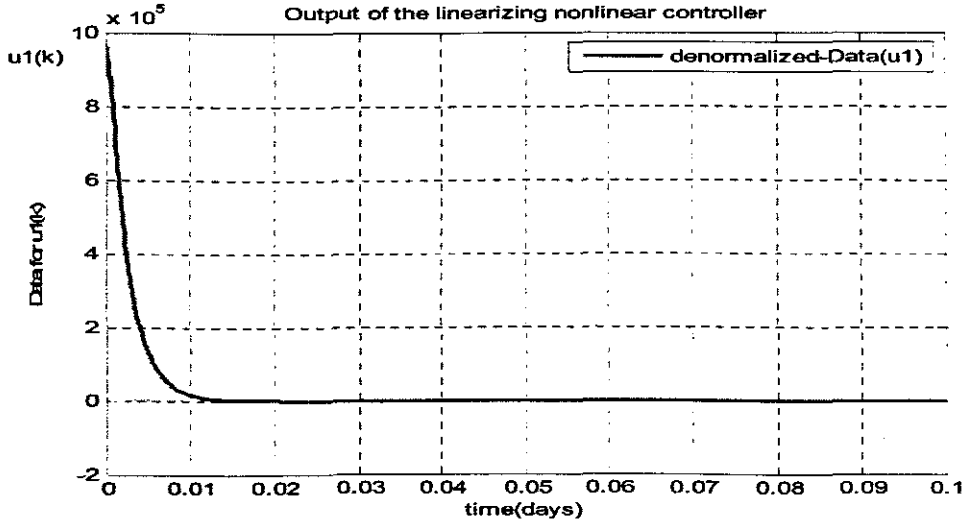


Figure 5.39: De-normalized NN output trajectory of the model of the nonlinear controller.

Figure 5.39 shows the de-normalized neural network output. This trajectory is an approximate of the output model of the linearizing controller considered earlier in . This conforms that the results of the verification is correct. The coefficients (weights and biases) of the selected model inputs with the command *newff* are tabulated in the following Tables 5.7 and 5.8 respectively.

Table5.7: Hidden layer weights and biases

	Hidden layer weights (wc)					Hidden layer biases (bwc)
Inputs / nodes	Vo_11 (norm v)	V1_11 (norm So(k))	K0i (norm So(k-1))	rso11	Soin11	
1	-4.22557	-0.09609	-0.938996	0.442493	0.0884965	5.826984
2	-2.76142	2.464213	3.158872	-0.075999	-0.015199	-0.328605
3	-2.34929	-0.286141	-4.082701	0.2172842	0.043456	3.669439
4	-1.75079	-1.076750	-4.479848	-0.224419	-0.044883	2.073979
5	2.96989	-0.412559	-3.888217	0.010486	0.002097	3.002949

Table 5.8: Output layer weights and biases

Output weights (vc)	0.19217	-0.44494	-0.13706	-0.20706	0.43572
Output bias (bvc)	0.07738				

5.5.2 Discussion of the results from the training of the two developed models of the NN linearizing controller

Both the feedforward and the Elman networks have been trained off-line to implement the model of the NN nonlinear linearizing controller for the dissolved oxygen process dynamics. A range of hidden layer neurons were tested. The models were trained with the *trainscg* algorithm. The results from the feedforward model indicate that with only 5 neurons in the hidden layer, an approximate trajectory of the target is achieved after only 47 iterations with a low MSE between the models of the linearizing controller output signal and the neural model signal. The same applies to the Elman network which indicates that after testing a range of neurons in the hidden layer, an approximate model of the target with a low MSE is achieved with 7 neurons in the hidden layer. To approximate the target it required only 42 training epochs. The conclusion derived therefore is that any of the two neural models developed (feedforward or the Elman network) can be implemented in the closed loop control of the nonlinear plant under consideration in this project. When the training was finished the two models developed with their weights are saved and were used in the construction of a complete closed loop system in the Simulink environment as illustrated in the proceeding section.

5.5.3 Simulation of the closed loop feedforward neural-control in the Simulink/Matlab environment

The Simulink block diagram representation of the closed loop realization of the feedforward neural-control scheme is as depicted in Figure 5.40.

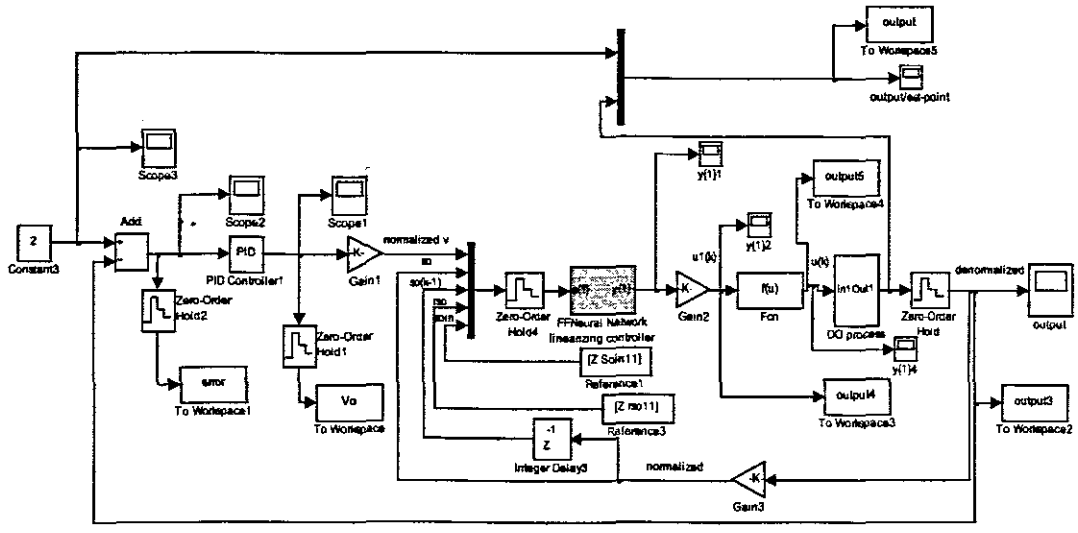


Figure 5.40: The closed loop block diagram for nonlinear linearizing and PI control of the process of DO concentration

Off-line simulation was performed of the time responses of the of the system output for a constant reference value of the set-point. Figures 5.41, 5.42 show the off-line simulated time responses of the closed loop structure with a set-point of 2 and whose linearizing neural controller is implemented by the feedforward network. As can be seen from the trajectories, the rise time for the scheme has improved when compared with the other schemes so far developed (inverse control, internal model control schemes). In addition, the controlled response tracks the set point trajectory with minimum steady state error. The overall performance analyses are tabulated in Table 5.5. This scheme will be tested for real time control of the DO process to determine the effectiveness of the control approach.

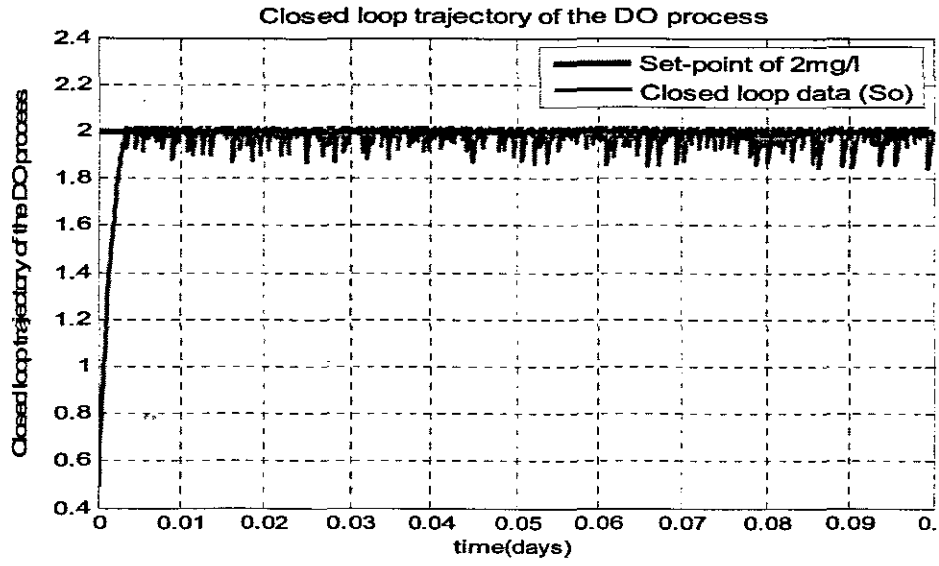


Figure 5.41: DO Time response trajectory of the closed system for a set-point of 2.

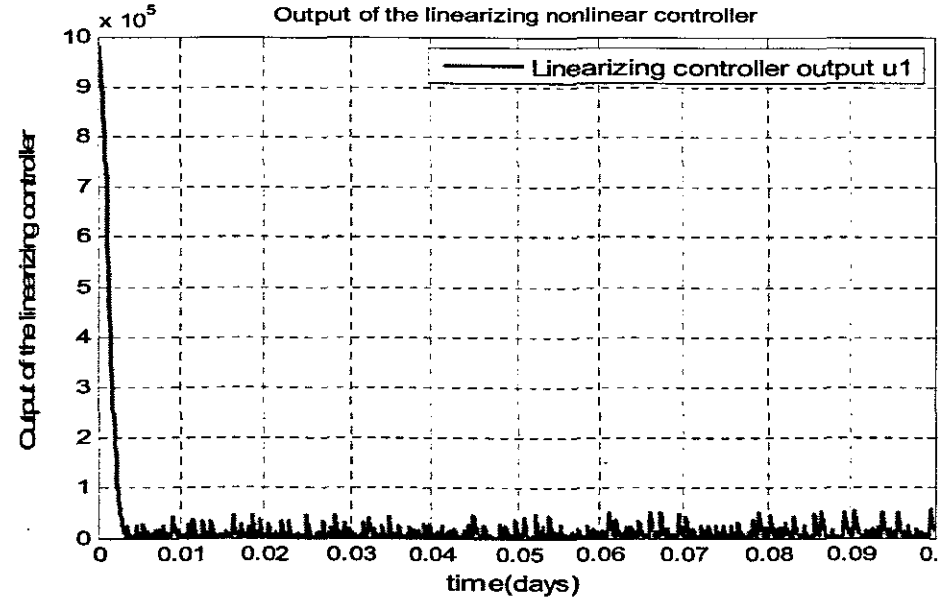


Figure 5.42: Time response trajectory of the model of the NN nonlinear controller (u1).

5.5.4 Investigations of the influence of the process set-point and disturbance values over the closed loop system behavior.

Investigations were carried out on the effectiveness of this scheme by varying some process parameters like set-point change, varying the disturbances like the oxygen uptake rate constant $r_{so}(k)$ and $Soin$. These parameter values are varied to be below and above their steady state values. They are used for simulations in order to observe their effects on the overall performance on the control system. The effects caused by various values of the set point are illustrated in Figures 5.43 and 5.44. The normalized values for $r_{so}(k)$ and $Soin$ are used for the simulation as 1 and 0.1 respectively.

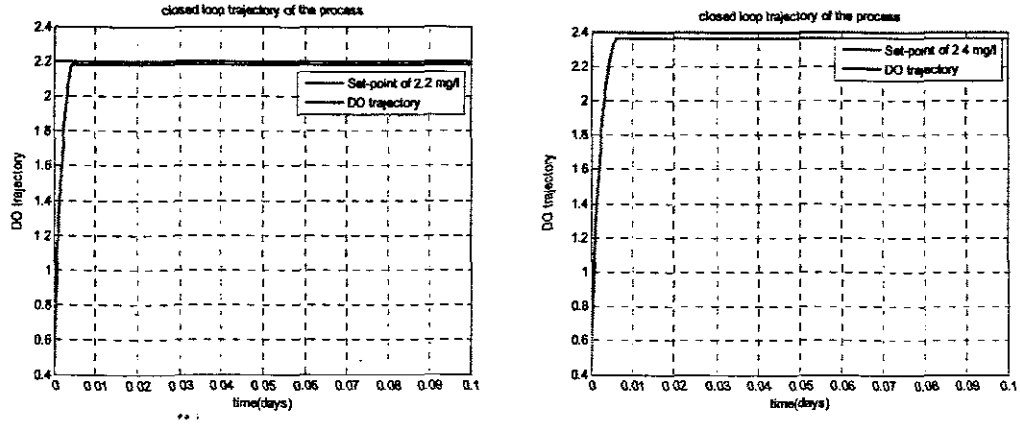


Figure 5.43: (a) and (b) DO Time response trajectories for set point of 2.2 and 2.4.

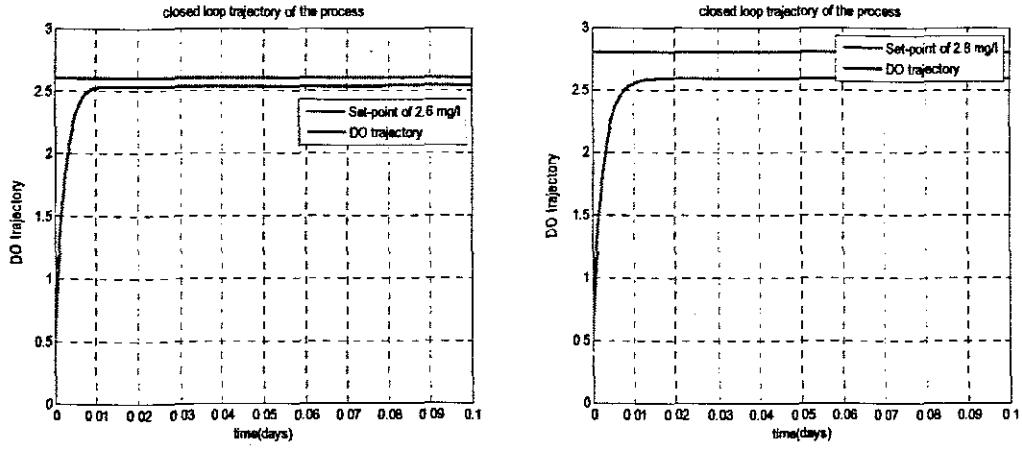


Figure 5.44: (c) and (d) DO Time trajectories for set point between 2.6 and 2.8.

The effects changes for the various values of $r_{so}(k)$ for a constant set point of 2 are depicted in Figures 5.45 and 5.46.

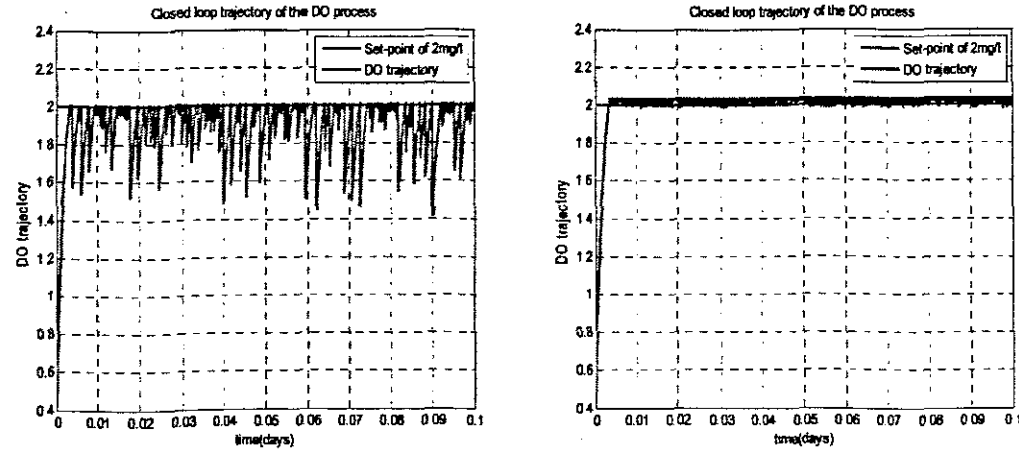


Figure 5.45: (a) and (b) DO Time trajectories for $r_{so}(k)$ of 0.9 and 1.2 for set point of 2.

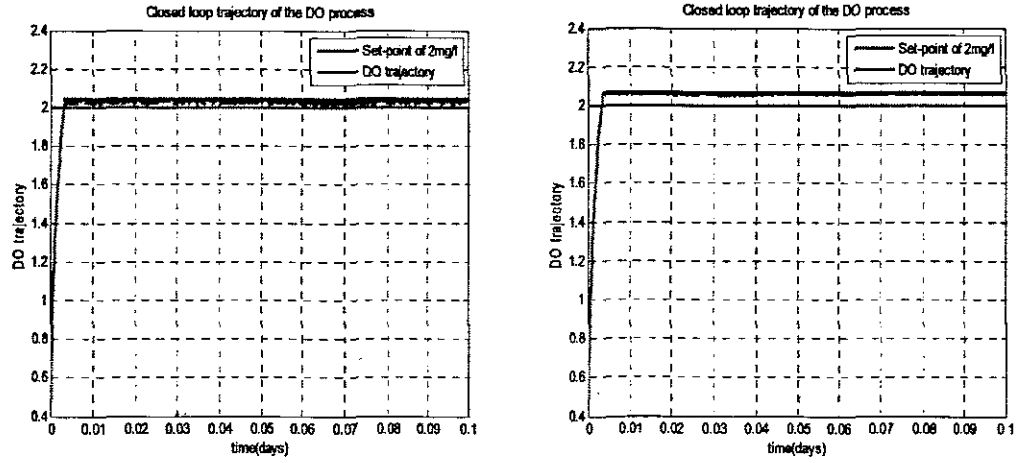


Figure 5.46: DO Time trajectories for $r_{so}(k)$ of 1.3 and 1.5 for set point of 2

Thus the simulation results show that any change in $r_{so}(k)$ has significant effect on the DO trajectory and steady state error. The effects changes for the various values of *Soin* for a constant set point of 2 are depicted in Figures 5.47, 5.48 and 5.49.

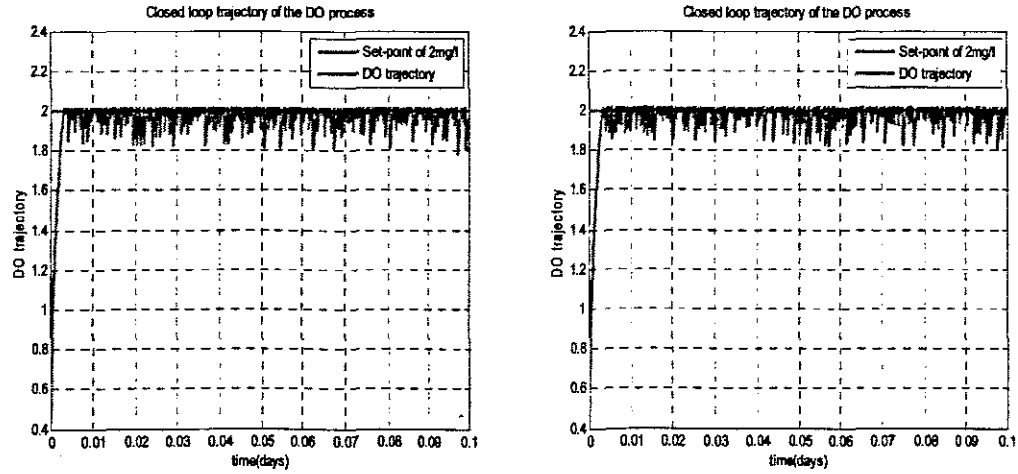


Figure 5.47: DO Time trajectories for *Soin* of 0.25 and 0.5 and set point of 2.

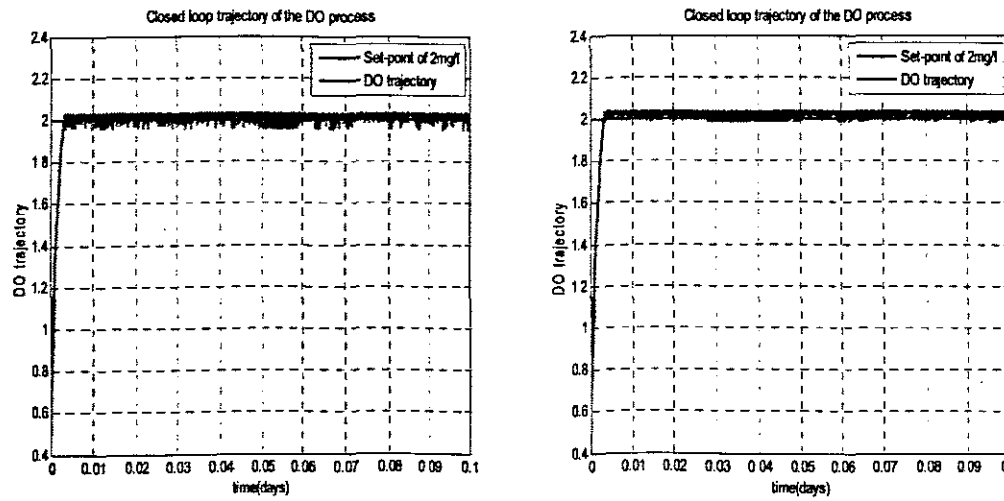


Figure 5.48: DO Time trajectories for *Soin* of 4 and 6 and set point of 2.

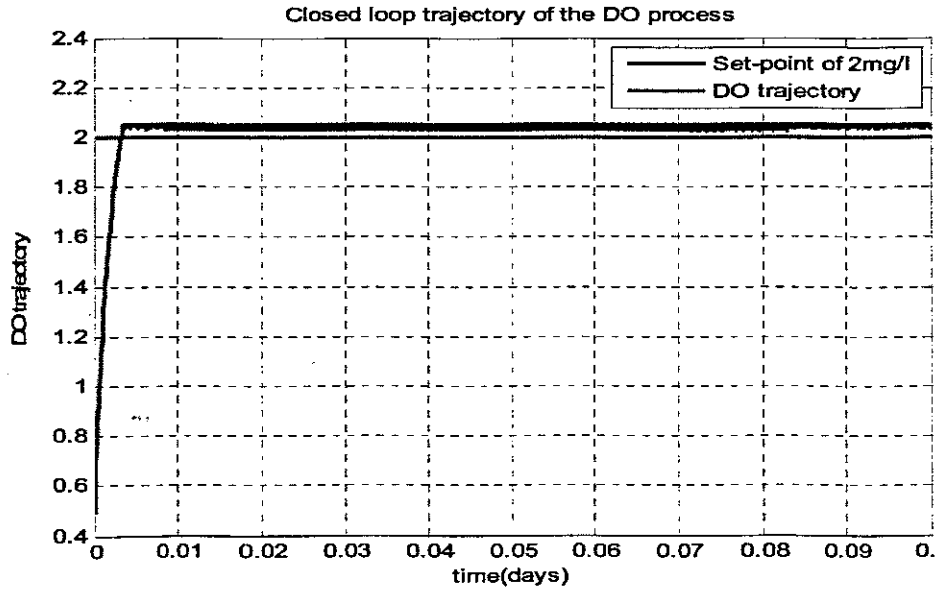


Figure 5.49: DO Time trajectories for $Soin$ of 10 and set point of 2.

Analyses of the simulation results similarly show that any change in $Soin$ has an effect on the DO trajectory and steady state error. However the steady state errors due to variations in $Soin$ are not as significant as the steady state errors due to variations in $r_{so}(k)$. The variations of the NN nonlinear controller outputs $u(k)$ for various set-point changes are as depicted in Figures 5.50 to 5.52. Analyses of the figures show that for a set-point of 2, the nonlinear controller output can be manipulated within a big range without affecting the controlled output. However, as the set-point is varied, the input must be maintained at specific values.

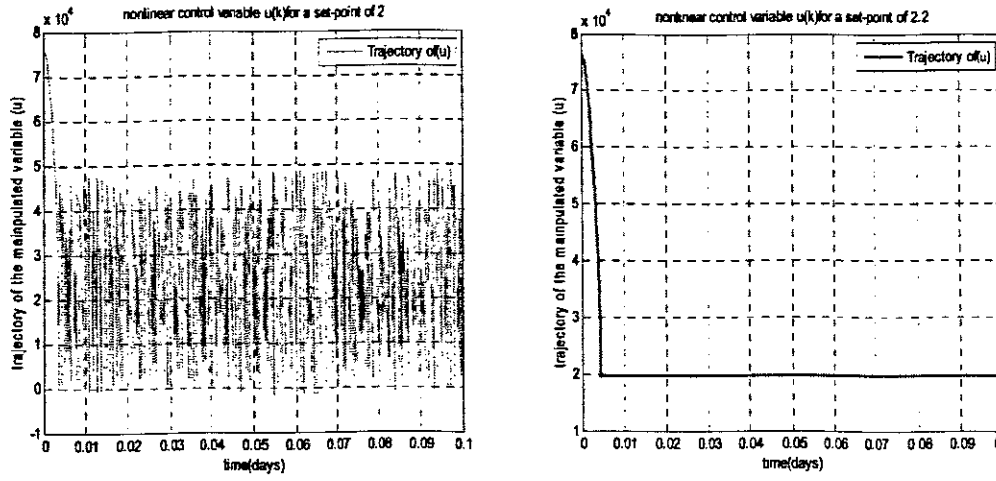


Figure 5.50: Nonlinear controller output trajectories for set points of 2 and 2.2.

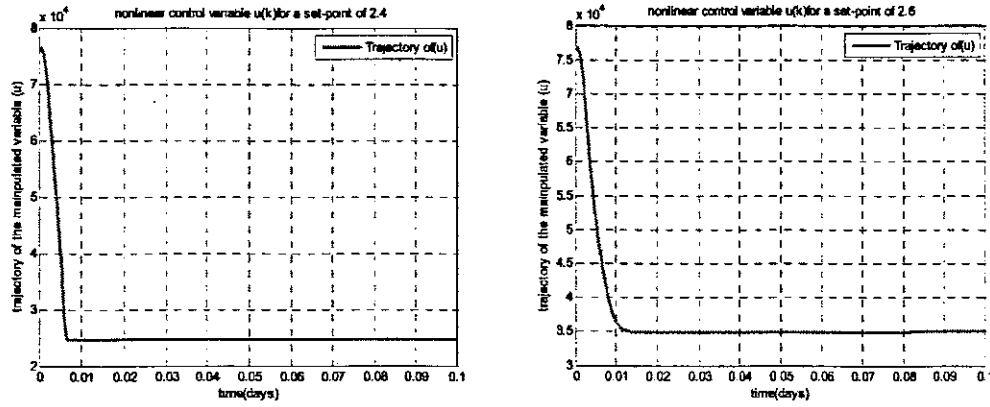


Figure 5.51: Nonlinear controller output trajectories for set points of 2.4 and 2.6

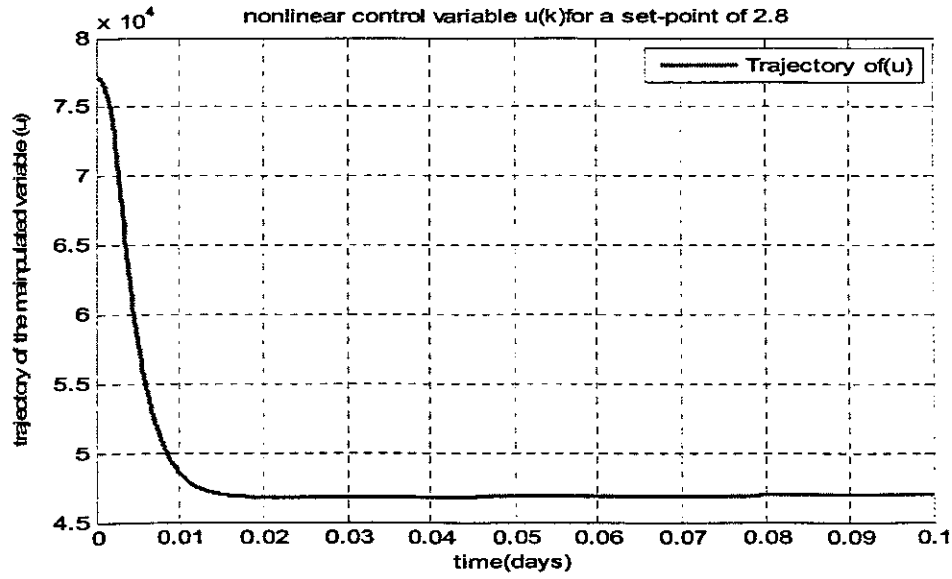


Figure 5.52: Nonlinear controller output trajectory for set point of 2.8.

Table 5.9 details the various performance indicators (rise-time, settling time, steady state error and peak overshoot) for the DO trajectories. Analyses of the selected values obtained for the behaviour of DO trajectory to changes in set-point and disturbances show that the control approach meets performance specifications.

Table 5.9: Performance analyses for the selected values

DO set-point (S_O^{sp})	Oxygen uptake rate r_{so}	S_{Oin}	Rise-time (t_r) (Days)	Settling time(t_s) (Days)	Steady state error (e_{ss})	Peak overshoot (M_p)
2	r_{so}	S_{Oin}	<0.002	<0.004	0	0%
2.2	r_{so}	S_{Oin}	<0.002	<0.004	0	0%
2.4	r_{so}	S_{Oin}	<0.002	<0.004	+0.05	+2.1%
2.6	r_{so}	S_{Oin}	<0.002	<0.004	-0.1	-3.1%
2.8	r_{so}	S_{Oin}	<0.002	<0.004	-0.2	-7.14%
2	r_{so}*0.9	S_{Oin}	<0.002	<0.004	±0.4	±20%

2	r_{so}*1.2	S_{Oin}	<0.002	<0.004	0	0%
2	r_{so}*1.3	S_{Oin}	<0.002	<0.004	~ -0.025	~1.25%
2	r_{so}*1.5	S_{Oin}	<0.002	<0.004	-0.07	3.5%
2	r_{so}	S_{Oin}*0.25	<0.002	<0.004	±0.2	±10%
2	r_{so}	S_{Oin}*0.5	<0.002	<0.004	±0.1	±5%
2	r_{so}	S_{Oin}*4	<0.002	<0.004	±0.04	±2%
2	r_{so}	S_{Oin}*6	<0.002	<0.004	~ -0.025	~1.25%
2	r_{so}	S_{Oin}*10	<0.002	<0.004	-0.07	3.5%

5.5.5 Simulation of the closed loop system with Elman nonlinear linearizing neural-controller in the Simulink/Matlab environment.

The Simulink block diagram representation of the closed loop realization with the Elman linearizing neural-controller scheme is as depicted in Figure 5.53.

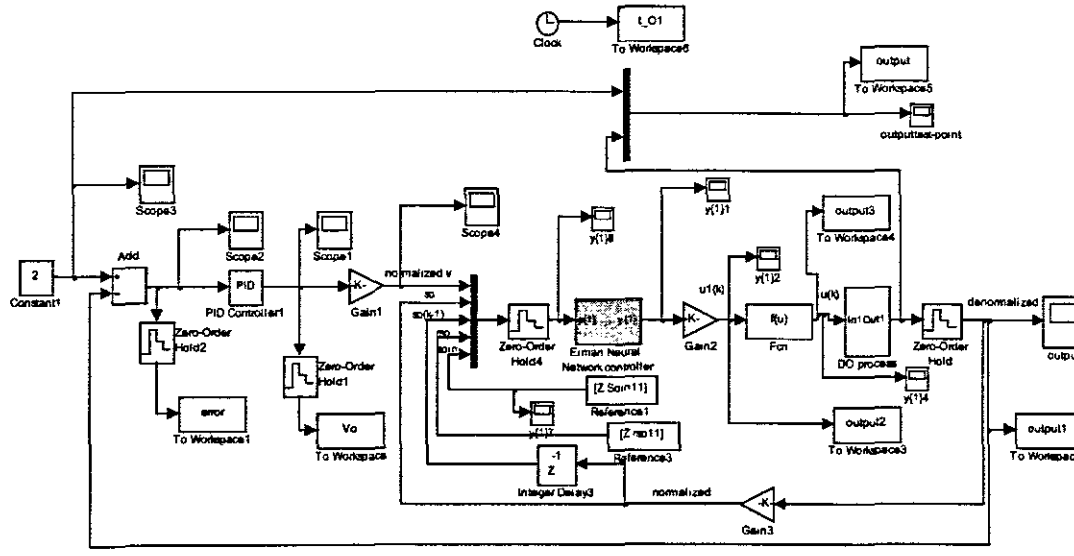


Figure 5.53: The closed loop block diagram with the Elman NN nonlinear linearizing controller and PI control of the process of DO concentration.

Similar to the Feedforward network discussed above, off-line simulation was also performed of the time responses of the system output for a constant reference value of the set-point. To obtain the given trajectories, the proportional gain of the linear PI regulator was increased by a factor of 10. Figure 5.54 shows the simulated time response of the closed loop structure with a set-point of 2. As can be seen from the trajectory, the output tracks the set-point with very low steady state error when compared with the other schemes so far developed (inverse control, internal model control schemes).

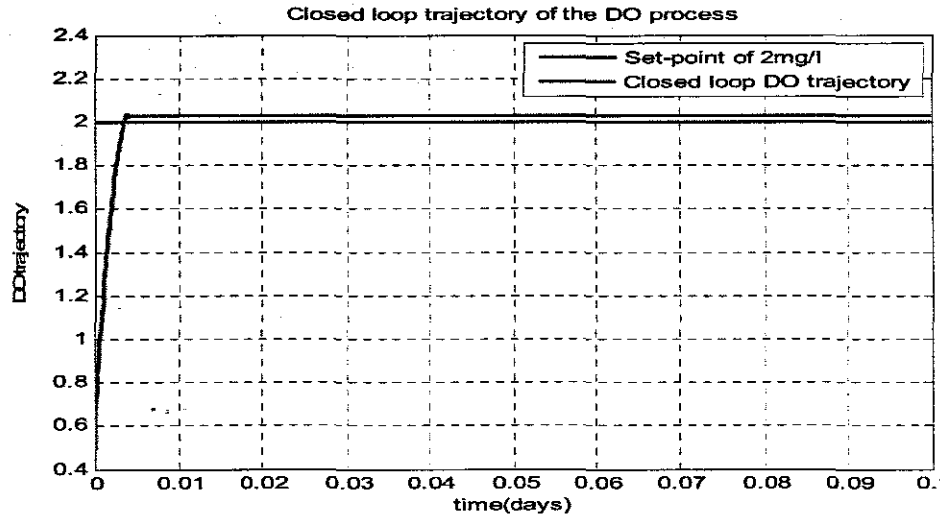
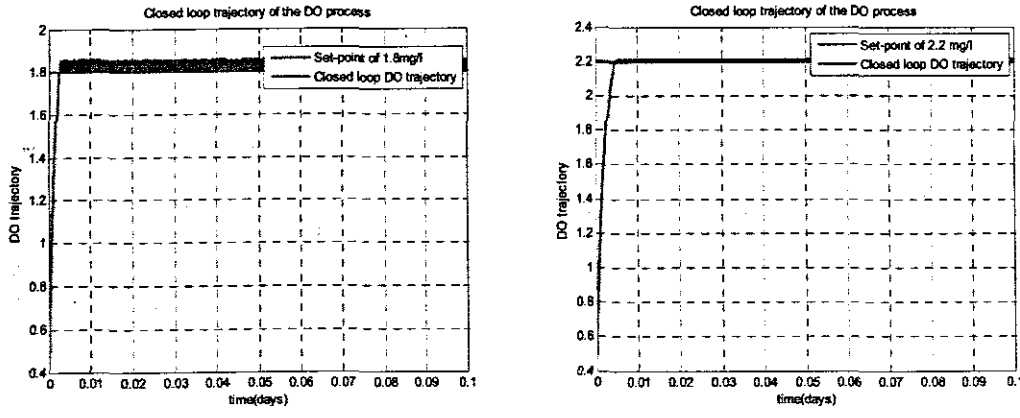


Figure 5.54: DO Time response trajectory of the closed system for a set-point of 2.

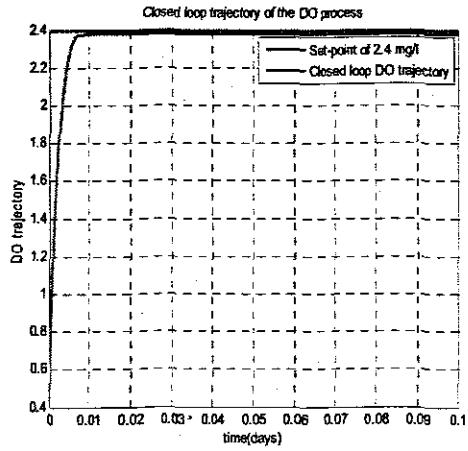
5.5.5.1 Investigations of the influence of the set point change and process disturbances over the results of simulations on the closed loop system with the Elman NN nonlinear linearizing controller

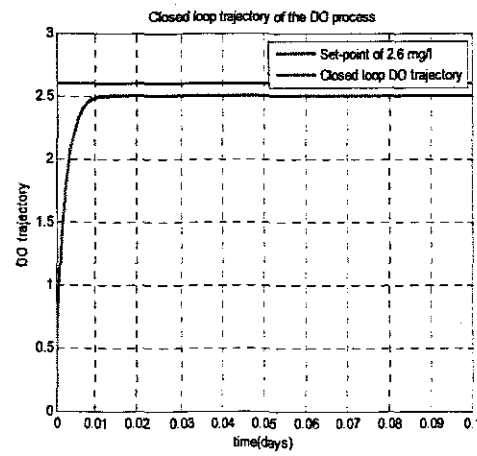
Investigations were carried out on the effectiveness of this control system by varying the set-point first and then the disturbance parameters like the oxygen uptake rate constant $r_{so}(k)$ and $Soin$ in order to observe their effects on the overall performance on the control system. These parameter values were varied to be below and above their steady state values. The effects for various values of set point change are illustrated in Figures 5.55(a), (b), (c), (d) and (e). The normalized values for $r_{so}(k)$ and $Soin$ are used for the simulation as 1 and 0.1 respectively.



(a) DO set-point 1.8 mg/l (b) DO set-point 2.2 mg/l

Figure 5.55: (a) and (b) DO Time response trajectories for set point of 1.8 and 2.2.





(c) DO set-point 2.4 mg/l (d) DO set-point 2.6 mg/l

Figure 5.55: (c) and (d) DO Time response trajectories for set point of 2.4 and 2.6.

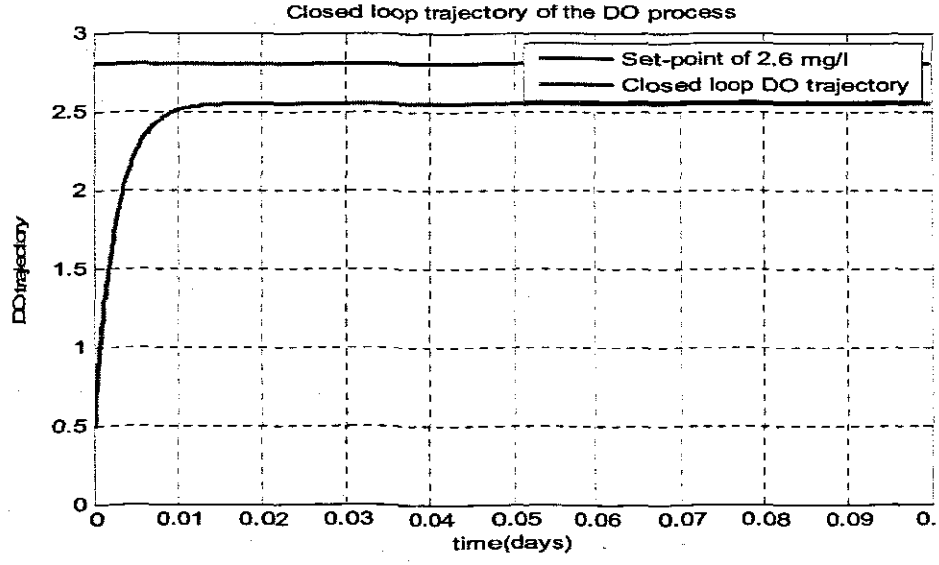
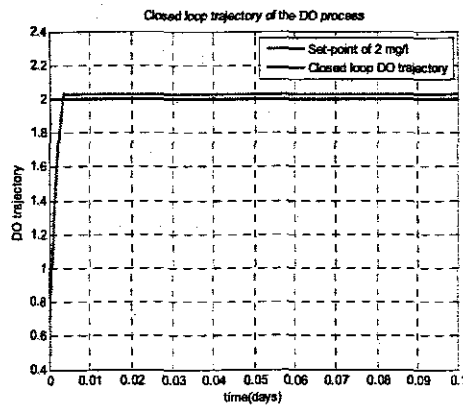
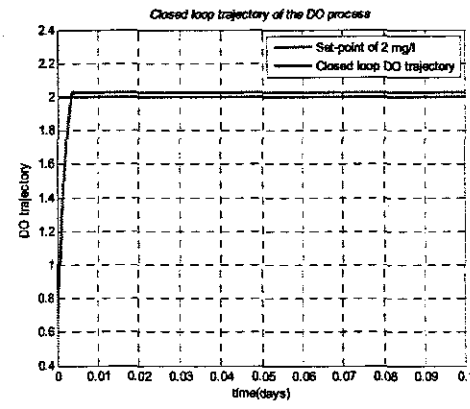


Figure 5.55: (e) DO Time response trajectories for set point of 2.8.

The results from the above figures demonstrate that this control system is capable tracking changes in set-point; hence it can be concluded that the closed loop system with the Elman NN nonlinear linearizing controller is effective in controlling the DO process trajectory. The effects of varying the disturbances constants $r_{so}(k)$ and $Soin$ are illustrated in Figures 5.56 and 5.57 that follows for a constant set point of 2mg/l.

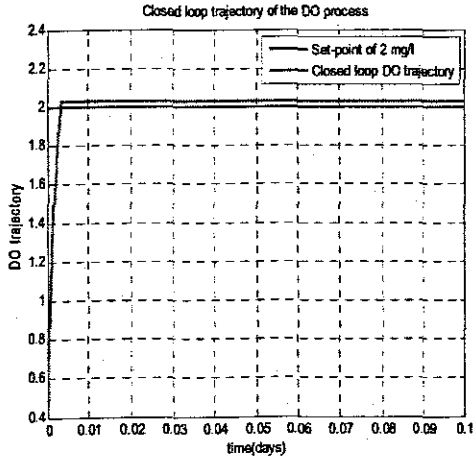


(a): $r_{so}(k)$ Of 0.9

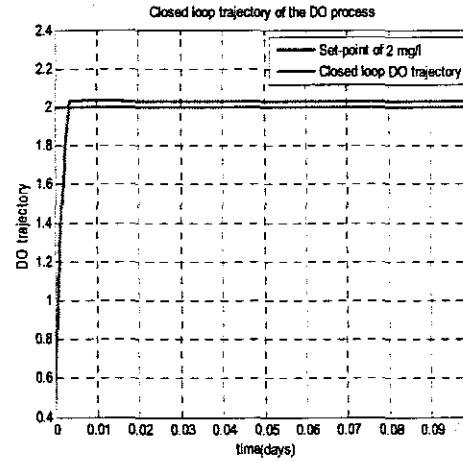


(b): $r_{so}(k)$ of 1.2

Figure 5.56: (a) and (b) DO Time trajectories for $r_{so}(k)$ of 0.9 and 1.2 for set point of 2

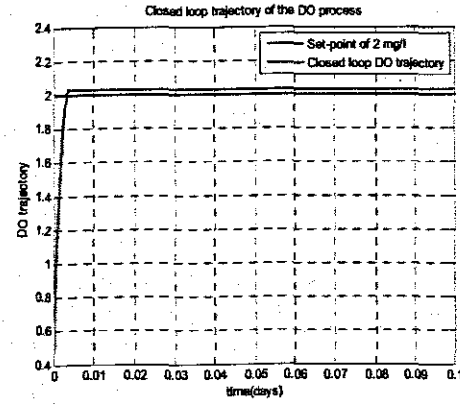


(c): $r_{so}(k)$ Of 1.3

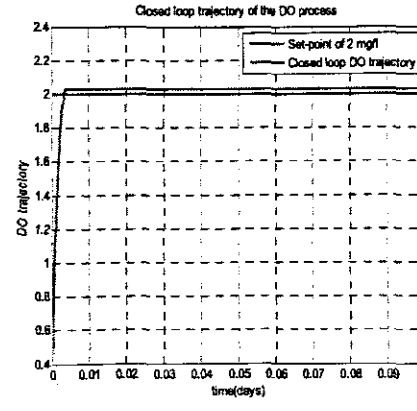


(d): $r_{so}(k)$ of 1.5

Figure 5.56: (c) and (d) DO Time trajectories for $r_{so}(k)$ of 1.3 and 1.5 for set point of 2

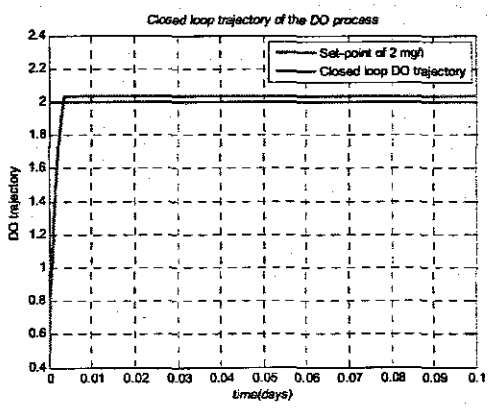


(c): $Soin$ Of 0.25

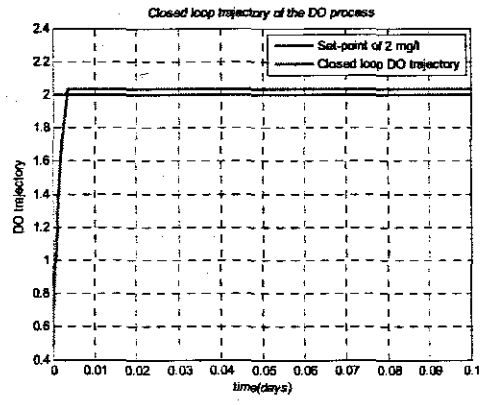


(d): $Soin$ of 0.5

Figure 5.57: (c) and (d) DO Time trajectories for $Soin$ of 0.25 and 0.5 for set point of 2



(c): $Soin$ Of 4



(d): $Soin$ of 6

Figure 5.57: (c) and (d) DO Time trajectories for $Soin$ of 4 and 6 for set point of 2

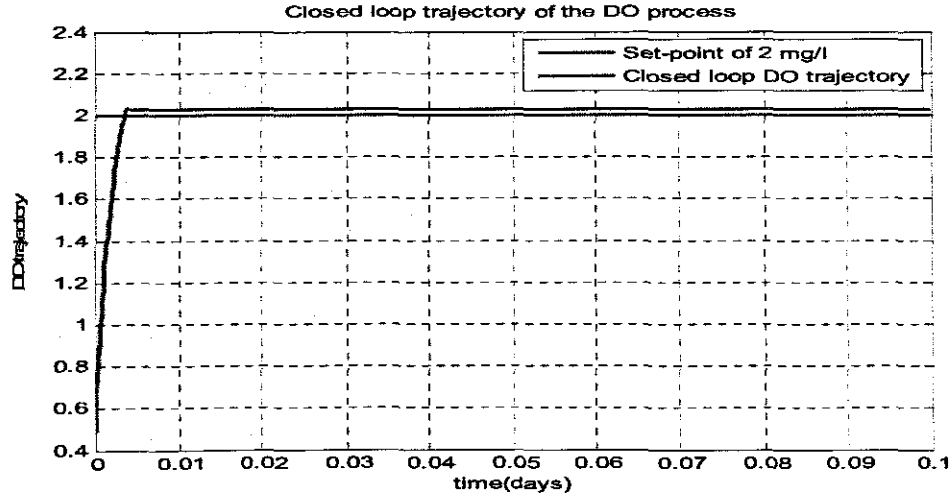


Figure 5.57: (c) and (d) DO Time trajectories for $Soin$ of 10 for set point of 2

Analyses of the trajectories of the DO process for the variation in the values of $r_{so}(k)$ and $Soin$ indicate that there is effect on the steady state errors as well as the peak overshoot. The results of investigation on the effects of changing various parameters are tabulated in Table 5.7 which details the various performance indicators (rise-time, settling time, steady state error peak overshoot) for the control system.

Table 5.10: Performance analyses for the selected values

DO set-point (S_O^{sp})	Oxygen uptake rate r_{so}	S_{Oin}	Rise-time (t_r) (days)	Settling time(t_s) (days)	Steady state error (e_{ss})	Peak overshoot (M_p)
1.8	r_{so}	S_{Oin}	<0.002	0.001	±0.025	±2.5%
2	r_{so}	S_{Oin}	<0.002	0.001	-0.05	-5%
2.2	r_{so}	S_{Oin}	<0.002	0.001	0	0%
2.4	r_{so}	S_{Oin}	<0.002	0.001	+0.02	+1%
2.6	r_{so}	S_{Oin}	<0.002	0.001	+0.1	+5%
2.8	r_{so}	S_{Oin}	<0.002	0.001	+0.3	-15%

2	r_{so}*0.9	S_{Oin}	<0.002	0.001	-0.2	-10%
2	r_{so}*1.2	S_{Oin}	<0.002	0.001	-0.22	-11%
2	r_{so}*1.3	S_{Oin}	<0.002	0.001	-0.23	-11.5%
2	r_{so}*1.5	S_{Oin}	<0.002	0.001	-0.25	-12.5%
2	r_{so}	S_{Oin}*0.25	<0.002	0.001	-0.2	-10%
2	r_{so}	S_{Oin}*0.5	<0.002	0.001	-0.21	-10.5%
2	r_{so}	S_{Oin}*4	<0.002	0.001	-0.22	-11%
2	r_{so}	S_{Oin}*6	<0.002	0.001	-0.23	-11.5%
2	r_{so}	S_{Oin}*10	<0.002	0.001	-0.25	-12.5%

Figure 5.58(a) shows the Non linear linearizing controller output trajectory $u1(k)$ for a set point of 2 and Figure 5.58 (b) the trajectory of the control output $u(k)$. Analysis of these two figures reveal that the control output signal required for a set-point of 2 is about 40000m^3/day.

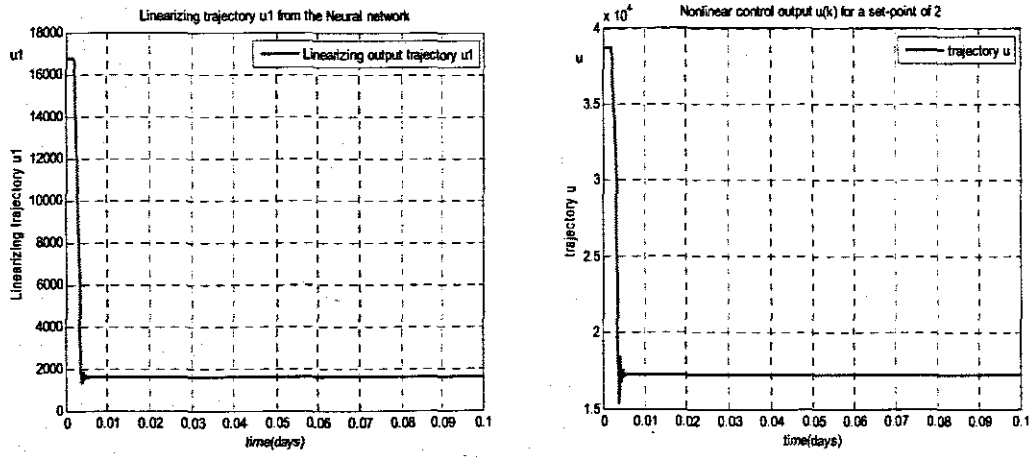


Figure 5.58: (a) Non linear linearizing controller trajectory for set point of 2.and (b) the trajectory of the control signal $u(k)$.

5.5.5.2 Discussion of the results from the two controllers developed

The results of simulation of the two control schemes considered in this section (the feedforward and the Elma nonlinear linearizing controller architectures) have shown the effectiveness of the schemes for the control of the dissolved oxygen trajectory. These have been measured on the specific performance indicators for the DO process on effects of disturbances, set-point changes etc. The controlled response tracks the set point trajectory very well. The overall performance analyses are tabulated in Tables 5.6 and 5.7. The controller structures will be implemented for real time control of the DO process to determine the effectiveness of the control approach in chapter six.

5.6 Conclusion

This chapter has been concerned with the design and development of the neural network based controllers for the DO process. Several control schemes have been studied based on the Feedforward and the Elman networks. Several off-line simulation results have been provided for the different types of the control schemes namely; the Inverse model, the Internal model control scheme and the nonlinear linearizing control structures utilizing the feedforward and the Elman networks for each of the structures developed.

The results have been used to show the effectiveness of the schemes for the control of the dissolved oxygen trajectory. These have been measured on the specific performance indicators for the DO process. Results for the Inverse model control as well as for the internal model control have not been satisfactory for the control of the DO process. Results from the nonlinear linearizing control scheme designed with the feedforward NN and the Elman network gave satisfactory simulation results for the different performance indices for the DO process. Thus these control structures will be implemented on the real-time lab-scale WWTP to see their effectiveness for the control of the DO process.

Chapter 6 that follows gives an overview of the lab-scale WWTP to be used for the implementation of the designed feedforward and Elman NN nonlinear linearizing controller structures for the DO process concentration via the airflow rate in order to maintain the dissolved oxygen concentration at desired level. The chapter also describes the hardware configuration of the various parts that form the process control loop together with the software used and the procedure adopted in the implementation of the real time control of the process.

CHAPTER SIX

REAL-TIME IMPLEMENTATION OF PROCESS CONTROL

The aim of this chapter is to program the structure and parameters of the PI and the neural networks nonlinear linearizing controller using PLC software in PLC environment, in order to determine the effectiveness of the controllers when applied on real-time situations. The process under consideration is the lab-scale wastewater pilot plant located in the Electrical Engineering department at the Cape Peninsula University of Technology. The real-time control law is for regulation of DO concentration via the airflow rate in order to maintain the dissolved oxygen concentration at a desired level. The chapter also describes the hardware configuration of the various parts that form the process control loop together with the software used and the procedure adopted in the implementation of the control system.

6.1 The DO Plant and the control system structure

The pilot plant of wastewater treatment is located in the Department of Electrical Engineering of Cape Peninsula University of Technology, Bellville campus. It creates possibilities to physically model different configurations of a real wastewater plant and to provide different types of experiments. Figure 6.1 shows the structure of the pilot plant.

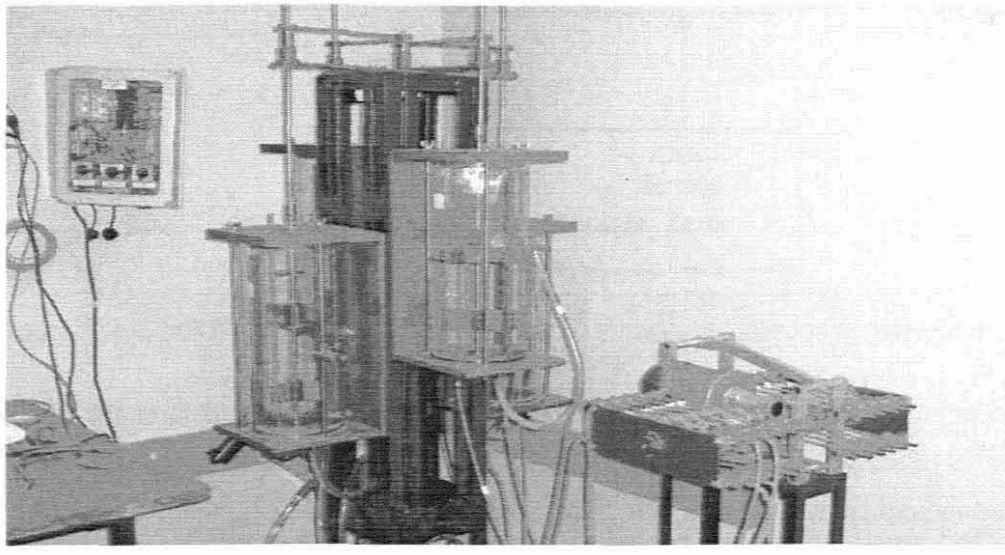


Figure 6.1: The pilot plant of the wastewater treatment located in the department

The plant consists of one anaerobic, one anoxic and one aerobic tank connected through two internal and sludge recycles. There are sensors placed inside these tanks which can measure variables such as dissolved oxygen concentration, conductivity, pH, turbidity and temperature. Each of these sensors has a transmitter which displays the currently measured variable and the signal can also be sent to the PLC. For the purpose of this study measurement and control of dissolved oxygen concentration is

considered. The dissolved oxygen is measured by a sensor and a transmitter/meter. The output of the meter from the measurement being sensed by the sensor is converted to a 4-20mA signals which are then sent to the input of the PLC. The PLC is connected to the PC with Adroit SCADA software via serial communication (RS232). The entire hardware and software structure layout utilized for data acquisition, calculations, and for process control is presented in Figure 6.2.

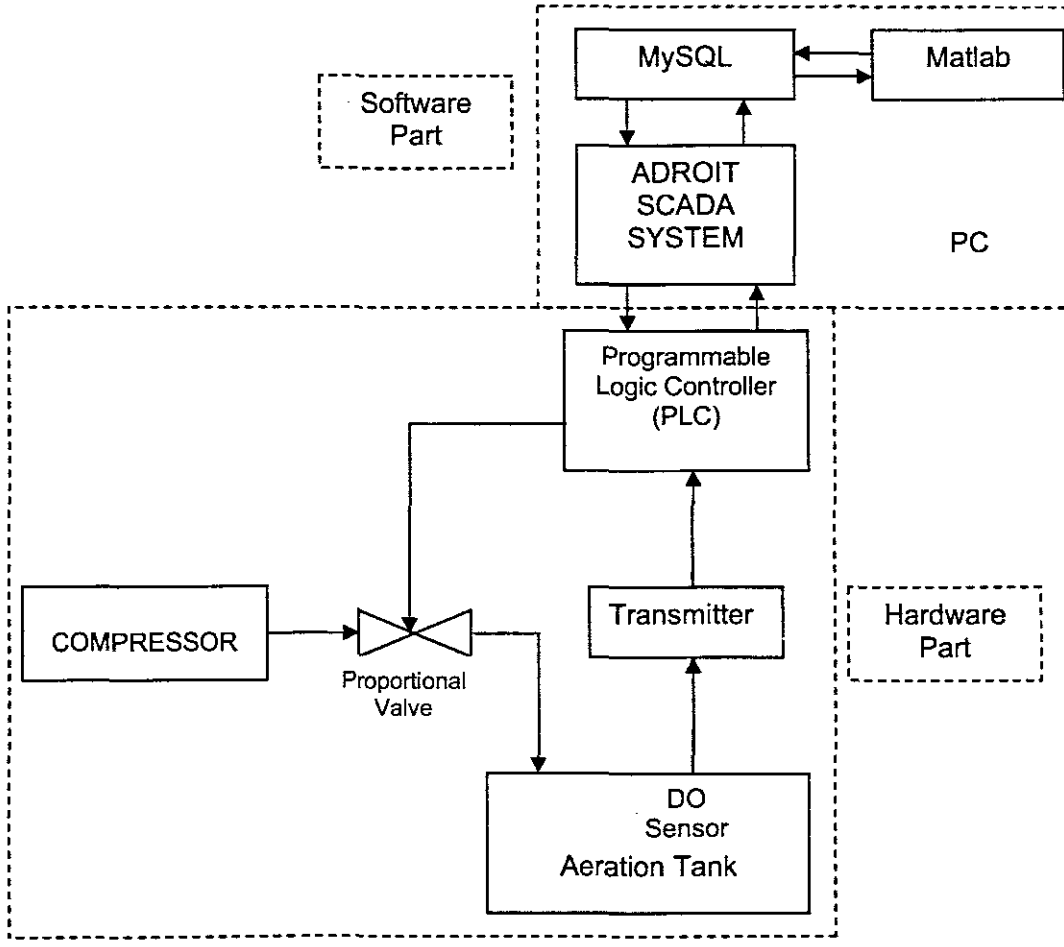


Figure 6.2: Block diagram of the Hardware and Software interaction for real-time control.

Figure 6.2 also illustrates at what level the interaction of programs and devices take place. Thus, for effective real-time control, a data acquisition system for the DO concentration using DO sensor, transmitter, PLC, PC and MySQL database has to be established. Communication between the software platforms used, namely; Adroit, MATLAB and MySQL has to be established and an Adroit GUI that implements the communication has to be developed. Brief discussions of the various block functions are given in the subsequent sections.

6.2 The Matlab block

MATLAB (*Matrix Laboratory*) *is a general, high-performance language for technical* computing. It integrates computation, visualization and programming in a common

environment. MATLAB is an interactive system and includes a large library of predefined mathematical functions. It also provides the user with the possibility to extend this library with new functions. MATLAB is available for most hardware platforms and is considered to be one of the most fundamental software tools at many technical universities as well as industries all over the world. Additional use is made of Matlab's Neural Network Toolbox. This toolbox has the ability to create a neural network topology suited for identification and control of the ASP. This includes derivation of transfer functions, learning rules, type of output weight adjustments etc.

The Adroit Agents are configured in order to achieve communication with the PLC. MATLAB communicates via the MySQL database to Adroit. The MATLAB program requires the measured signal from the PLC in order to perform the calculations of the PI and the NN nonlinear controller parameters. Therefore, for these signals to get to MATLAB they must be sent to MySQL database. MATLAB receives the mathematical model coefficients of the DO process, DO set point, and DO measurements. These are used for the design of the nonlinear linearizing controller and the PI controller for simulation and control of the entire closed loop system. The obtained NN controller weights, biases and PI controller parameters are sent to MySQL database. From there they are sent to Adroit and PLC registers. The communication between Modicon PLC and PC with Adroit as driving software is set up by communication agent Adroit for real-time operations.

6.3 The Adroit SCADA system

Adroit is an acronym for Advanced Distributed, Real-time, Object-oriented, Intelligent, Toolkit. Adroit is a technologically advanced 32-bit SCADA (Supervisory Control and Data Acquisition) System designed for process control, manufacturing systems and open automation applications. It is developed specifically for Windows NT. Adroit has a flexible object-oriented client-server architecture that supports from a stand-alone implementation to an installation that spans multiple-sites. Adroit does not have a separate development package. All the tools to fully configure the system, draw process mimics, etc. are packaged in the basic system - as are all the standard drivers. Adroit is an example of what is known as a client-server architecture or model. In this model the client portion, which is typically the part of the application that interacts with the user, communicates with the server portion, which is usually a data repository. Thus, in Adroit system the server part of the client-server architecture is called the Agent Server. This contains a collection of intelligent objects known as agents. Adroit agents are intelligent objects because, in addition to containing state information, as do conventional database records they also include functionality. The

Adroit user interface or client portion consists of several different user windows. The mimic windows within the user interface are themselves made up of objects, in this case graphical objects such as picture elements. In addition to state information, such as the properties of the size, location, colour, etc., Adroit graphical objects may be configured to display a variety of animation behaviours. By connecting these behaviours to data values within agents, different animated display effects can be configured, to depict real world events. The following are a summary of the language and terminology of Adroit.

- *User Interface*: This provides the windows through which the user can supervise and control the system. With the drawing tools provided, mimics representing the real world can be created. To each element the user can then assign specific properties and behaviours, and associate them with real time data residing in the Agent Server.

- *Agent Server:* This stores real time and historical data. The real time data resides within intelligent objects, known as agents, which exchange messages with other agents or picture objects either from the local host machine or from remote computer nodes on the network. Historical data is stored on disk where it is available for rapid retrieval.

- *Agents:* As well as being able to manipulate their own data, they have the ability to read and write to other objects and influence them. Each kind of object, or agent type, has well defined rules for interfacing with other agents ensuring that an extremely robust and versatile environment has been created to handle any real world situation.

Various types of agents provide the means by which different information storage and manipulation mechanisms are delivered. Some types of agents are designed to be purely data containers and other types have their functionality extended to provide intelligent conditioning of their own data or another agent's data. The user may create as many instances of each agent type as desired.

6.4 Development of an Operator interface in Adroit.

An operator interface is developed in Adroit for real time control and monitoring of the real plant. Lab measured process variables and DO values from the real time process are gathered via PLC and reflected on the Adroit interface window for analysis and determination of the DO process trajectories. Adroit communicates to MATLAB via MySQL database. Any change in the status of the process such as valve open or closed, are then reflected on Adroit.

Thus, the Adroit window created is used to achieve the following:

- Acquire sensor data from the PLC,

- Store this data in a Database,
- Acquire coefficients of the Neural networks linearizing controller and set point values from a Database,
- Update these values to PLC in real-time and
- Allow for the change in parameters and observe process responses and trends due to these changes.

The block diagram representation of the communication between the various blocks for the implementation of the control strategy is as depicted in Figure 6.3.

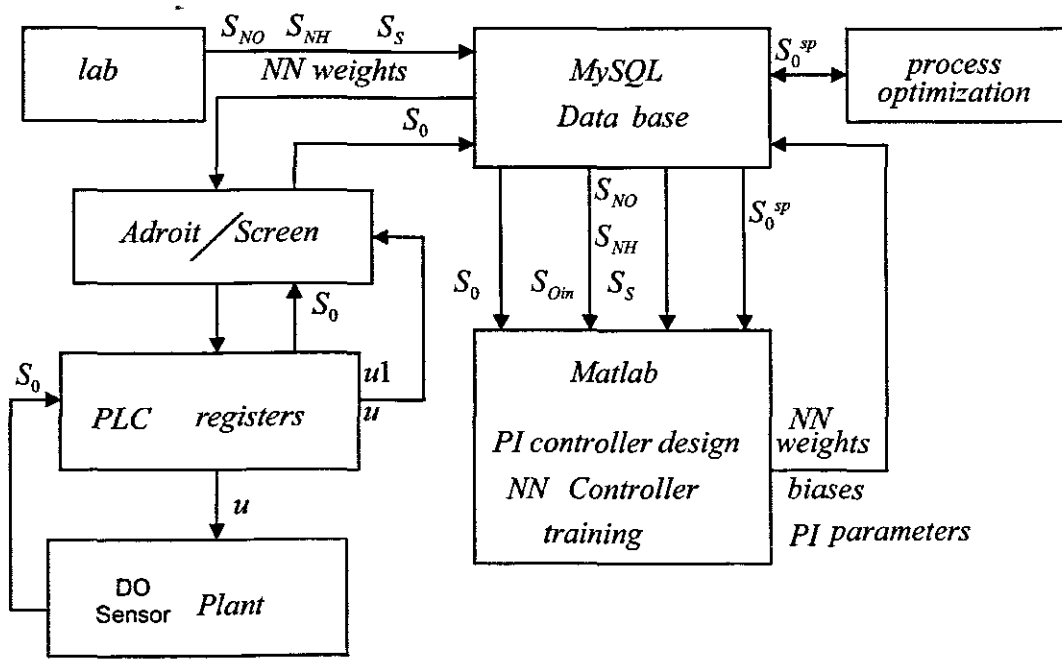


Figure 6.3: Block diagram representing communication for real-time control design and implementation.

6.5 Embedding the PI and NN controller to PLC.

The PLC's used in industry have very limited possibilities for implementation for different controllers. Normally the PLC has P, PI, PID controller algorithms embedded in the software environment. This situation is very restricting for the modern control applications. At the same time the PLC's are used widely in industry. The question is to use the existing PLC's but to find a way of programming them to implement the algorithms of the new nonlinear controller. The challenge is how to select some of the linear and nonlinear functions of the PLC and to apply them for the nonlinear NN controller implementation.

The development of the structure for embedding in PLC is going through two main steps:

- Building the controller structure in Simulink
- Building the controller structure in PLC.

The first step was necessary to be done for comparison and validation purposes. The experience of the first step is used during the second step.

6.5.1 Simulink implementation of the structure and parameters of the NN linearizing controller.

The NN controller can be implemented in Simulink using vector and matrix operations. PLC has not these types of variables and functions. This means that the NN controller structure has to be built using the simplest possible PLC software blocks as coefficient blocks and multiplication and summation functions. In this case the number of used elements is growing. This means that PLC memory volume can be limited for some applications and some balance between the complexity of the algorithm and the available memory has to be made. The NN nonlinear linearizing controller is configured in the Simulink environment first in order to validate the control structure. A block diagram of the structure with PI and NN nonlinear linearizing controller is shown in Figure 6.4.

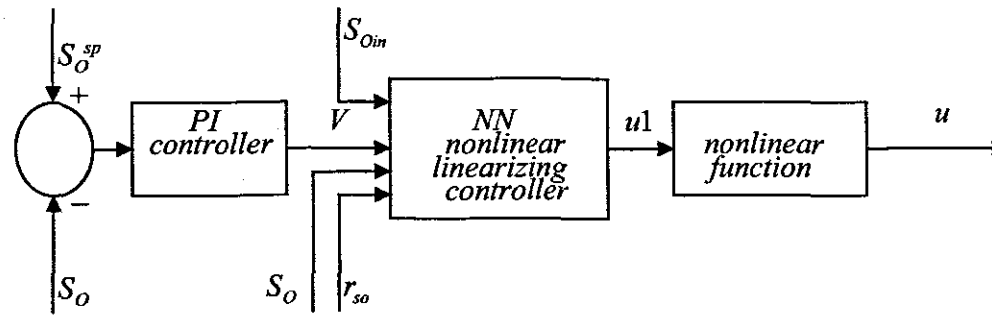


Figure 6.4: Structure with PI and NN nonlinear linearizing controller

The Simulink diagram of the realization of the NN nonlinear linearizing controller is as depicted in Figure 6.5. In this structure the NN controller is realized by the Simulink toolbox functions which are missing in the PLC.

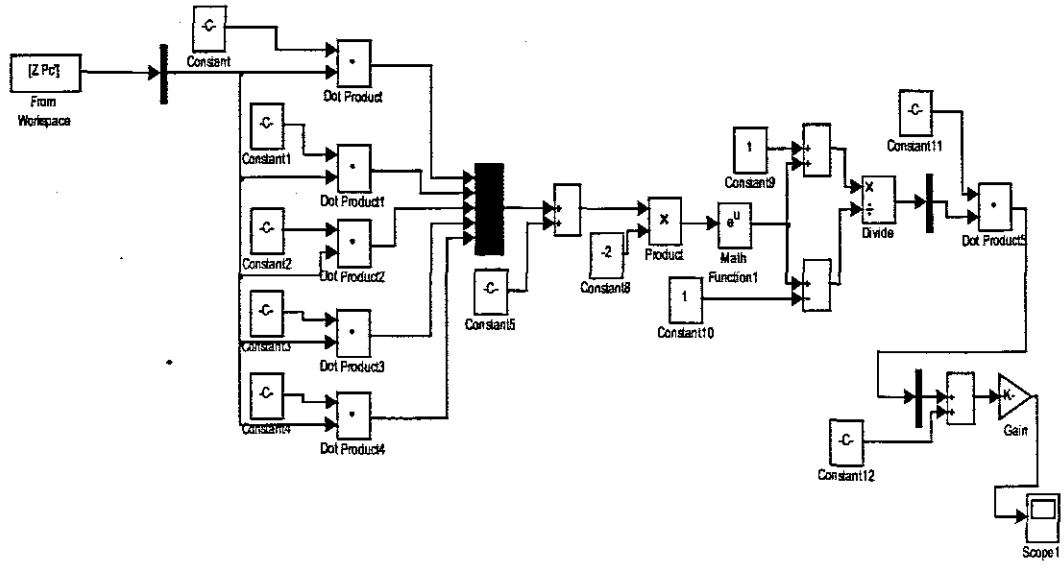


Figure 6.5: Simulink Block diagram representation of the NN linearizing controller.

In this Simulink block diagram the NN linearizing controller has two parts; the NN and a nonlinear function connected in series as was presented in Figure 5.25, and reproduced in Figure 6.4. It is constructed using simple functions from the Simulink library in order to realize the NN controller functions using the feedforward and the recurrent neural networks. This step is necessary because with such simple functions it is easy to wire up the PLC to act as NN. The simulated trajectory from the set up is as illustrated in Figure 6.6 for the function $u1(k)$.

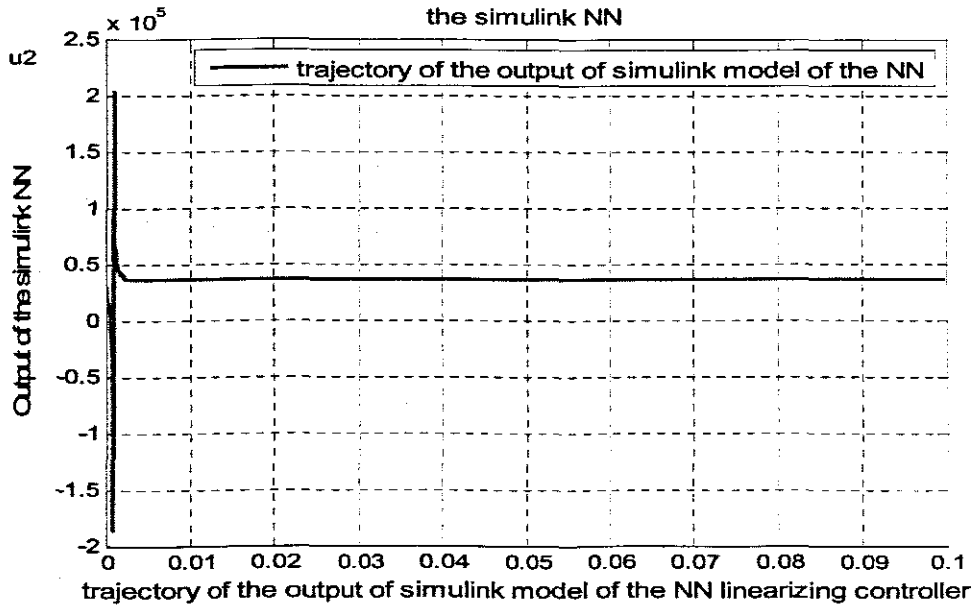


Figure 6.6: Simulink trajectory representation of the NN linearizing controller output

The trajectory is for the weights of the feedforward NN linearizing controller implemented earlier in chapter 5. The same Simulink block can be implemented using the Elman network. Analysis of the trajectory represented in Figure 6.4 reveals that the controller output trajectory at steady state is about 4000m3/day. This value is not the same as the one recorded using the neural networks. The difference may be as the result that some of the neural network activation functions such as the *tan sigmoid* and *the purelin* were approximated functions for implementation and not the actual functions. From practical point of view the obtained result is applicable to the real process.

6.5.2 The Programmable logic controller (PLC) NN controller implementation

This section shows the implementation of control algorithms to programmable logic controllers (PLC). The first phase is the development of the control algorithm in a simulation environment and its verification on simulated models. Next, a real control system is connected to the computer via a PLC to test the algorithm on a real physical model. The PLC serves only for I/O as all computations are still being done in the simulation environment - MATLAB/Simulink. In the third phase the control algorithm is completely ported into the PLC. Monitoring of the process and parameters configuration may still be done using the connected PC with MATLAB/Simulink. The PLC used is the Modicon M340 programmable controllers.

The Modicon M340 programmable controllers are programmed with Unity Pro, the latest IEC 61131-3 development software designed to affordably and effectively handle the requirements of very simple machine tasks to the most complex machine solutions. Figure 6.7 shows layout of the structure.

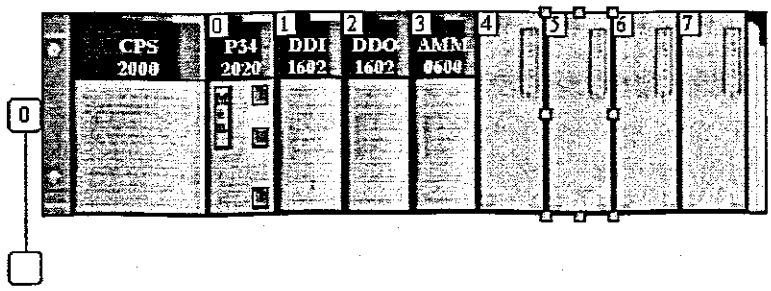


Figure 6.7: Structure of Software Modicon M340

Modicon M340 design is developed by Schneider Electric and possesses the following features:

- Compact, rugged and exceptionally powerful and is ideal for complex machine control and medium sized infrastructure applications.
- Open connectivity to other devices with serial, Ethernet, CANopen and USB all built into the processor

- On-board web server enables powerful and easily accessible remote control and monitoring and recipe and data management.
- Powerful integrated motion capabilities making "Plug & Move" solutions a reality. Modicon M340 counter modules are independent of the CPU enabling exceptional performance with simple configuration

Unity Pro software offers five IEC61131-31 programming languages as standards. Each section of the code can be programmed in the language of choice, adapted to each processing operation. All edit, debugging, and operation tools are assessable whatever language is used. The languages are as listed:

- LD: Ladder diagram
- IL: Instruction list
- ST: Structured text
- SFC: Sequential function chart
- FBD: Function block diagram

The Simulink block representing the structure and the parameters of the NN nonlinear linearizing controller, together with the PI controller parameters are to be implemented by programming the PLC with Unity Pro software. The structure of the neural network controller (activation functions, weights, biases, number of layers, and nodes) are to be represented by the Simulink structure of Figure 6.5, where simple elements given in the Simulink diagram are to be wired up in PLC. A control loop functions library is integrated in Modicon M340 as standard in Unity Pro software. IEC 61131-3 Function Block Diagram language enables graphic and highly flexible programming. The control loop algorithm can be optimized and control of operation kept without disturbances. This programming language is used to wire up the PLC.

The PLC program using functional block diagram is constructed in such a way that it calculates the relevant values needed from parameters calculated by Matlab and entered by the user or Adroit. The DO process value and set point value are measured and obtained from MySQL database. The controllers' outputs will attempt to achieve the set-point value. Mathematical operators, ADD (Addition), SUB (Subtraction), MUL (Multiplication) and DIV (Division), are used to calculate the controller outputs. The parts of the NN nonlinear linearizing controller implemented in the Simulink model Figure 6.5 are used for setting up of the various function block elements of the PI and NN controllers in addition to the mathematical operators mentioned above to implement the controller algorithm. After the function blocks are designed, they are downloaded to the PLC. The schematics of the structure are depicted in Figures 6.8.

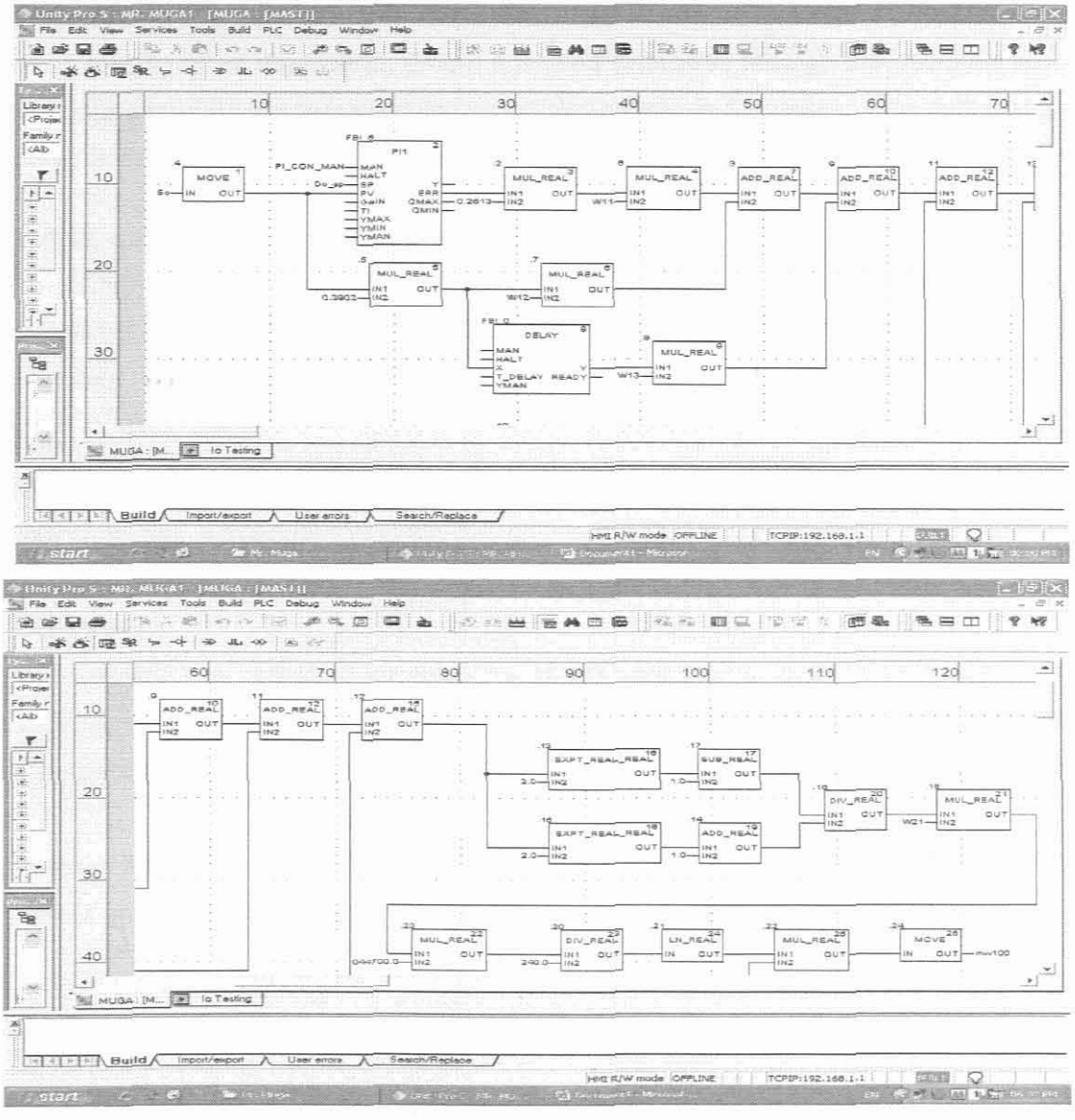


Figure 6.8: Function block diagram loaded on PLC

6.6 MySQL Database Manager Professional

A database is used as the middleman to establish communication between Adroit and Matlab. This connection is necessary for Matlab to be able to extrapolate data acquired by Adroit about the system to be controlled and the variable states. Furthermore this connection allows Matlab to be used for calculation as well as control purposes. In addition to the connection property of the Database its primary purpose is the gathering and logging of historical data of both the process and various variables of the process under control. MySQL is used to create and manage a database. It stores or retrieves all data received from or sent to Matlab, Adroit and the operator. In order for the other programs to make use and communicate with the database, it has to be added to the ODBC Data Source Administrator of Windows XP. Appendix D explains how to create and set up a new database in order to work with Windows and

communicate with other applications. MySQL Query Browser is a separate application, which forms part of the MySQL group. It is used to browse through the database and generate a query script and execute this script. This allows the user to view data that is stored in the database. The browser can be executed from the Adroit main menu.

6.7 MySQL communication with Matlab

The DO sensor signal is sent to the PLC for execution of the program after which the measured signal is communicated to Adroit. From the Adroit the data is sent to MySQL database. MySQL database stores and retrieves the data needed for training the NN (S_{Oin}, S_O etc.), weights and biases, PI controller parameters. The training is in the Matlab environment. After training, the data is sent to the database which then communicates with Adroit to transfer the data to the PLC registers where the data sits.

6.8 Conclusion

This chapter has provided an overview of the real time implementation approach undertaken for the project. The lab-scale WWTP at CPUT together with the software algorithms have been presented in addition to the PLC programs.

Chapter 7 that follows provides the conclusion for the entire work undertaken by this project and provides a direction on the future improvement of the results achieved.

CHAPTER SEVEN

CONCLUSION: DELIVERABLES, DISCUSSION OF THE RESULTS AND FUTURE DIRECTION OF RESEARCH.

This chapter gives an overview of the project, application results and findings of the investigation. In addition, the chapter summarizes the comparative analyses of various NN developed models and controllers and gives a summary of the deliverables and the possibilities for improvement of the obtained results in future research.

7.1 Introduction

Research in non-linear control is motivated by the inherently non-linear characteristics of the dynamic systems we often try to control. In reality, virtually many practical systems are significantly non-linear in nature so that the important features of their performance may be completely overlooked if they are analysed and designed through linear control techniques. There is no equivalent mathematical theory on which base a full theory of nonlinear systems for control can be built. Effective control of nonlinear systems requires robust methods to accommodate system uncertainties and harsh environments. Waste water treatment plant is one such nonlinear system with complex multivariable problems which require effective control because of the increased requirements of the quality of the effluent.

Intelligent control techniques such fuzzy logic, neural networks and genetic algorithms have had many successes in the field of control because of their ability to accommodate nonlinear parameters and harsh environment. The activated sludge processes are highly complex because they are characterised with a lot of variables, complex dependencies and nonlinear interconnections between variables. These difficulties require robust methods of control design and implementation. For this thesis, the problem of modeling of activated sludge process (ASP) and identification and control of dissolved oxygen (DO) concentration in WWTP using the neural networks theory has been considered. Table 7.1 presents a complete schedule of network algorithms of the models developed.

TABLE 7.1: THE SIMULINK MODELS AND MATLAB PROGRAMS DEVELOPED
This section presents a table of the algorithms and Simulink models developed

SIMULINK MODEL	NAME	FUNCTION
DO_MODEL2_2.mdl	DO process parameters obtained from ASM1	Simulation of the Simulink DO model trajectory
DESIRED.mdl	Implementation of desired reference model in closed loop	Simulation of the Simulink desired reference DO model trajectory
DESIREDNONLINEAR.mdl	Implementation of nonlinear linearizing control by Feedforward neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
DESIREDNONLINEARelm.mdl	Implementation of nonlinear	Simulation of the Simulink

	linearizing control by Elman neural network in closed loop	closed loop control trajectory of the DO process
DESIREDNONLINEAR6745elm.mdl	Implementation of feedforward nonlinear linearizing control by neural network in Simulink	Simulation of the Simulink closed loop control trajectory of the DO process
INVERFF1.mdl	Implementation of direct inverse control by Feedforward neural network	Simulation of the Simulink closed loop control trajectory of the DO process
INVERELM_model2.mdl	Implementation of direct inverse control by Elman neural network	Simulation of the Simulink closed loop control trajectory of the DO process
INVERFFimcmodel2.mdl	Implementation of internal model control by Feedforward neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
INVERELM_imcmodel2.mdl	Implementation of internal model control by Elman neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
MATLAB PROGRAMS (script file)	**NAME**	**FUNCTION**
Parameters_DO.m	Steady state parameters or the ASM1 model	script file for simulation of the DO process using ASM1steady state parameters
Model1.m	Script file for NN training with one input and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Model2.m	Script file for NN training with two input and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Model3.m	Script file for NN training with three inputs and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Model4.m	Script file for NN training with twelve inputs and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Nonlinearcon.m	Script file for simulation of nonlinear controller function	simulation of nonlinear controller function
Nonlinearconelm.m	Script file for Elman NN nonlinear linearizing controller	simulation of Elman NN nonlinear controller function
Nonlinearconff.m	Script file for feed forward NN nonlinear linearizing controller	simulation of feedforward NN nonlinear controller function
BESTFF.m	Script file for feed forward NN training with six inputs and one output	Forward identification of feed forward NN with six inputs and one outout
BESTELM.m	Script file for Elman NN training with six inputs and one outout	Forward identification of feed Elman NN with six inputs and one outout
INVERELM.m	Script file for Elman NN training	Inverse identification of feed Elman NN
INVERFF.m	Script file for inverse feedforward NN training	Inverse identification of feed Elman NN
Parametersforu1.m	Script file for nonlinear function u1	Simulation of the nonlinear function u1

7.2 Aims and Objectives

Based on the statement of the problem, the project aim and objectives are development and investigation of the structures of multilayer perceptron feedforward and recurrent neural networks in the solution of the problems for identification and control of nonlinear systems. The aim and objectives of the project are achieved through the following deliverables:

- Development of mathematical models for the nonlinear system.
- Development of nonlinear identifier models for nonlinear systems by use of neural networks.
- Development of neural network controllers based on the inverse and internal models
- Development of neural network controllers based on the closed loop linearization approach
- Development of algorithms and programs for the identifiers and controllers in Matlab for deployment on a hardware platform.
- Real time implementation of the developed program on a real nonlinear system.
- Deployment of the NN controller structure and parameters into PLC environment for purposes of real-time control.

7.3 Deliverables of the thesis

According to the aim and objectives of the project, the following work has been done:

7.3.1 Mathematical model development and simulation

- The nonlinear mathematical model of the DO concentration process was developed
- The nonlinear mathematical model of the DO concentration process was simulated in Matlab/Simulink software environment in order to obtain the Airflow rate/DO process trajectories in order to obtain the training data for the neural network identifiers and controller development. The developed model is documented in Chapter 2.

7.3.2 Development of nonlinear NN identifier models for nonlinear DO process.

System identification is concerned with developing the neural model of the DO concentration trajectory based on the input and output data of the actual system. Neural networks identifier models are developed for both the feedforward and the recurrent types. The inputs these two types are varied to include combination of the DO process, delayed values of DO process, the air flow rate and delayed values of the airflow rate in order to come up with the inputs that give the required targets. Initially single-input single-output networks were developed, the input being the airflow rate and the output being the DO process trajectory.

7.3.2.1 Development and investigation of Feedforward NN identifier models

The first type to be developed was the feedforward networks. Feedforward networks with a range of hidden layer neurons were tested. The MSE between the process and the target was monitored. Thus, increasing the number of hidden layer neurons and the number of inputs had a direct influence on the accuracy of the model until the best model was chosen. The developed models are documented in chapter 4.

7.3.2.2 Development and investigation of Elman NN identifier models

Recurrent 'Elman' networks are the second type of neural networks to be developed. Elman networks have built in feedback loops which enable them to model dynamic systems such as the DO process more accurately. Similarly, Elman networks with different size of the hidden layer neurons were tested in order to come up with the best model that represents the DO process. The developed models are also documented in chapter 4. The results from the developed models indicate that the networks (feedforward and Elman) with 6 inputs trained with Levenberg marquardt algorithm yielded an accurate model of the DO process.

From the analyses done and verification results of the two selected models both (the recurrent network and the feedforward networks trained with the Levenberg-Marquardt *(trainlm)* algorithm, the conclusion derived is that both models can be used for identification of the DO process since both networks yield satisfactory results for the DO process with minimum square error (MSE). The results of these models are tabulated in Tables 4.14 and 4.15 (for feedforward network) and Tables 4.29 and 4.30 (for Elman networks).

7.3.3 Development of NN controllers

Three different control strategies, including the inverse control, the internal model control, and the feedback linearizing nonlinear control mechanisms have been employed and evaluated based on the models developed to determine which control technique would be the most efficient to implement. These controllers have been developed for both the feedforward and the Elman networks. Summarized from the simulation results presented in chapter 4, 5 and 6, the following subsections give the conclusions derived.

7.3.3.1 Development of NN controller based on inverse model

NN based controllers for DO concentration based on model inversion technique for both the feedforward and the Elman network have been studied. The controllers' performances to control the DO process were evaluated through set-point stabilization studies. The two neural models for the recurrent and feedforward neural networks were created using neural network toolbox residing in the Matlab environment. The structures of closed loop direct inverse control (feedforward and recurrent) were implemented in the Simulink environment as depicted in Figure 5.13 and 5.14. The simulation results show that this design and configuration of the NN control is not effective if it is based only on the inverse model approach.

7.3.3.2 Development of NN controller based on internal model control (IMC)

These structures are also implemented in the Matlab/Simulink environment by the block diagrams illustrated in Figure 5.17 and 5.20. The results for both the internal model control structures (IMC) indicate better performance in comparison with the case of the inverse model control, but still the desired behavior of the system output can not be achieved. The filter shown on Figure 5.16 is not implemented here. If a PI controller may have been included in the design and control implementation to act as a filter in the forward path, may be the results could have improved. Thus for future improvement a PI controller should be considered to an input $e(k)$ and an output $v(k)$ going to be input for the NN controller. This PI controller can be designed to introduce desirable robustness and tracking response to the closed loop system.

7.3.3.3 Development of NN controllers based on closed loop linearization approach

NN based controllers for DO concentration based on closed loop linearization approach for both the feedforward and the Elman network have been studied. The results of simulation of the two control schemes (the feedforward and the Elman nonlinear linearizing controller architectures) have shown the effectiveness of the schemes for the control of the dissolved oxygen trajectory. These have been measured on the specific performance indicators for the DO process on effects of disturbances, set-point changes etc. The controlled response tracks the set point trajectory very well.

In conclusion, several control schemes have been studied based on the Feedforward and the Elman networks. Several simulation results have been provided for the different types of the control schemes namely; the Inverse model, the Internal model control scheme and the nonlinear linearizing control structures utilizing the feedforward and the Elman networks for each of the structures developed. The results have been used to show the effectiveness of the schemes for the control of the dissolved oxygen trajectory. Results for the Inverse model control as well as for the internal model control have not been satisfactory for the control of the DO process. Results from the nonlinear linearizing control scheme designed with the feedforward NN and the Elman network gave satisfactory simulation results.

7.4 Development of algorithms and programs

The methods for design described above are examined through simulation and Real-Time implementation using software programs and algorithms developed in Matlab environment as well as Simulink block diagrams. Separate software programs are developed for every of the deliverables from 7.3.1 till 7.3.4. The developed software for

simulations of the models is tabulated in the Table 7.1. For activations of the various neural network topologies, *tan sigmoid* and *pure linear* transfer functions were used in the hidden layer and output layer of the NN respectively. The training error was computed by means of the negative of the gradients and conjugate directions using the scaled conjugate gradient algorithm and the Levenberg marquardt algorithm. The mean squared error (MSE) was used to evaluate the performance of the models.

7.5 Embedding the NN nonlinear controller structure in PLC for real-time control

Matlab/Simulink/Neural Networks is the software utilized to implement the models and the algorithms, and to carry out identification and control simulation. Eventually the neural networks models obtained are transported through Simulink to PLC for real-time implementation. Communication between the software platforms Adroit, MATLAB and MySQL is established in order to implement the real time control on the lab-scale wastewater treatment plant. In any control problem a control loop will normally be implemented to facilitate the control procedure. The control loop that implements the real-time DO control is as illustrated in Figure 6.3. This control configuration is realized by interfacing various software packages to accomplish the control process.

7.6 Importance of deliverables

The main focus of this project has been about investigation of neural networks for purposes of identification and control of nonlinear system. The scientific and application results obtained have been as a result of the research work undertaken on the theory of neural networks as well as the study of the DO concentration process in waste water treatment plant being a nonlinear system. The deliverables are significant in that:

- They provide a knowledge base in nonlinear control by application of neural networks theory.
- This will enable utilisation of neural networks for control and provide an alternative method for nonlinear control design.
- For educational application, the control of nonlinear DO concentration by use of neural networks is of interest to undergraduate and postgraduate students towards the field of control. They can use these findings as a benchmark for design and testing of basic neural identifiers and controllers.
- The control algorithms developed in this project can also be used for demonstration purposes or as examples for students.
- The operators in the field of wastewater treatment industry can use the simulation results to show that the neural network controllers are designed successfully.

- The proposed PLC implementation of the NN linearizing controller can improve the control of the WWTP.

7.7 Application of the thesis results

The developed models, algorithms and software serve for

- Understanding: Gives insight in the wastewater treatment processes and how suitable mathematical models are derived.
- Education and training: Operators are not always adequately trained to operate advanced sensor and control equipment and most environmental engineers would need more basic understanding of process dynamics and control in order to appreciate the potential of instrumentation, control and automation.
- Neural network theory practitioners: The field of artificial intelligence and in particular neural networks is still a developing one. The results could be considered as a tool for future postgraduate students' projects at university level
- Communication: As a basis of communication between the various software platforms Adroit, Matlab/Neural networks and MySQL in order to solve the problem of DO control and other process control problems.
- Real time control. Function approximation by neural networks is a common feature in many areas of process modeling and identification both in on-line applications such as real time control and in off-line applications such as the modeling of the dynamics of DO processes. These results can be used as a basis of further development of the theory of neural networks for control of the DO process.

7.8 Future research work

Results for the Inverse model control as well as for the internal model control by neural networks have not been quite satisfactory for the control of the DO process. This may be due to the fact that a filter element was not included in the design. Thus for future work an improvement on the design of these controllers is necessary. Secondly, the practical implementation introduced various obstacles. For execution of the project, some notable problems were encountered such as the interconnection of MATLAB neural network weights to the PLC via the database. This necessitated the construction of a Simulink block diagram to imitate the neutral network and then used the block diagram to program the PLC. This connection is of extreme importance as it forms the link to the communication between the software packages. It is also through this link that that control is established as Adroit and MATLAB cannot communicate directly. Thus for future research work, improvements are necessary on this area.

7.9 List of publications related to the thesis

The work on this project resulted in a number of publications as listed below:

- Muga, JN., Tzoneva, R. 2009. 'Development of neural network process models and linearizing controllers for DO concentration process real-time control in wastewater treatment plants'. IEEE Transactions on Neural Networks (submitted to journal)
- Muga, JN., Tzoneva, R. 2009. 'Investigations on the structure of multilayer feedforward neural networks for identification of DO concentration process in wastewater treatment plants'. Submitted to SAIEE research journal.
- Muga, JN., Tzoneva, R. 2009. 'Investigation of the structure of multilayer recurrent neural networks for identification of DO concentration process in wastewater treatment plants'. Submitted to journal of Artificial Intelligence.
- Muga, JN., Tzoneva, R. 2009. 'Design of neural networks feedforward linearizing controller for DO concentration process in wastewater treatment plants'. Submitted to International journal of neural systems.
- Muga, JN., Tzoneva, R. 2009. 'Design of the recurrent neural networks linearizing controller for DO concentration process in wastewater treatment plants'. Control and automation journal.
- Muga, JN., Tzoneva, R. 2009. 'Design of the Inverse model and the internal model neural network controllers for DO concentration process in wastewater treatment plants'. Submitted to SAIEE research journal.

REFERENCES

Albus, J. S. 1975. A new Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). Transactions of the ASME, (97), 220-227.

Aguado, D., Ribes, J., Montoya, T., Ferrer, J., and Seco, A., 2008 A methodology for sequencing batch reactor identification with artificial neural networks: A case study. *Computers and Chemical Engineering* 33 (2009) 465–472

Andrášik, A., Mészáros, A., Azevedo, S. F., 2003. "On-line tuning of a neural PID controller based on plant's hybrid modeling". *Computers and Chemical Engineering* 28 (2004) 1499–1509

Andrews J.F., 1974. Review paper: Dynamic models and control strategies for wastewater treatment processes. Water Research. 8, 261-289.

Anwar, M., Hussain, A., Ramachandran, K.B., 2006. The study of neural network based Controller for controlling dissolved oxygen concentration in a sequencing batch reactor. Bioprocess Bio-system. Eng. 28, 251–265.

APHA (American Public Health Association), 1998. "*Standard Methods for the Examination of Water and Wastewater*", 20th ed. Washington, DC.

Astrom, K.J.., and Wittenmark, B., 1997. *Computer-Controller Systems–Theory and Design* (3rd Edition), Prentice Hall, Upper Saddle River, New York.

Az"man K, Kocijan J (2007) Application of Gaussian processes for black-box modeling of bio-systems. ISA Trans 46:443–457. DOI: 10.1016/j.isatra.2007.04.001.

Barbu, M., Caraman, S., and Ceanga, E, 2005. Control strategies of a multivariable wastewater treatment process. Comparative study. *University of Galati, Romania.* MCCS, Ayia Napa, Technical Program.

Baruch, I. S Georgieva, P., Cortes, B. J., and Sebastiao, F. 2003. Adaptive Recurrent Neural Network Control of Biological Wastewater Treatment." *International Journal Of Intelligent Systems,* VOL. 20. 173–193 (2005) © 2005 Wiley Periodicals, Inc. Published online in Wiley Inter Science (www.interscience.wiley.com).

Bastin G. and Dochain D., 1990. On-line Estimation and Adaptive Control of Bioreactors. Elsevier, Amsterdam. pp. 379.

Bennett, G.F., 1980. Oxygen-transfer rates, mechanisms and applications in biological wastewater-treatment, *Critical Review Environment Control* 9 (4), pp. 301–392.

Billing, A.E. and Dold, P.L. 1988. Modelling Techniques foe Biological Reaction Systems. Part 1: Mathematical Description and Model Representation". Water. Sci. Tech., Vol. 14, pp. 185-192.

Bishop C., 1997. Neural networks for pattern recognition. Oxford University Press, New York.
Brett R.W.J., Kermode R.I. and Burrus B.G., 1973. Feed forward control of an activated sludge process. Water. Res., 7, 525-535.

Carlsson, B., and Rehnström, 2002. A Control of an activated sludge process with nitrogen removal – a benchmark study. Water Science and technology vol. 45 No.4-5pp135-142

Carpenter, Gail A., and Grossberg, Stephen, 1987: A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine, *Computer Vision, Graphics and Image Processing, 37*, 54115.

Chang Kyoo Yoo, Jong-Min Lee and In-Beum Lee. 2003. Nonlinear model based DO control in a biological Wastewater treatment process. *Korean journal of Chemical Engineering* vol.21, pp14-19.

Chen, S., Billings, S.A. and Grant, P.M., 1992, "Recursive hybrid algorithm for non-linear system identification using radial basis function networks", *Int. J. of Control*, 5, 1051-1070.

Cook S.C., and Jowitt, P.W., 1985. Investigation of dissolved oxygen dynamics in the activated sludge process, IFAC Identification and System Parameter Estimation, York, UK (1985).

Cosme Rafael Marcano-Gamero., Universidad de los Andes, 1986. Plant identification and control using a neural Controller based on reference model. *Proceedings of the 17th IASTED international ..., 2006 - actapress.com*

Cybenko, G., 1989: Approximation by Superposition of a Sigmoidal Function, *Mathematics of Control, Signals and systems*, 2 pp. 492-499.

Dague, Richard R. Angenent, and Largus T. 1996. Temperature-phased anaerobic waste treatment process. Iowa State University Research Foundation, Inc. (Ames, IA)

Dilip Sarkar, 1995; Methods to speed up error back-propagation learning algorithm, ACM Computing Surveys (CSUR), v.27 n.4, p.519-544.

Economou, C. Morari, M. Palsson, O. 1986. IMC 5. Extension to Nonlinear Systems. *Ind. Eng. Chem. Process Des. Dev. 25*, 403–411.

Emmanoulides, C., Petrou, L., 1997. Identification and control of anaerobic digesters using adaptive, on-line trained neural networks. Computer. Chem. Eng. 21 (1), 113–143.
Flanagan, M.J., 1977. Control Strategies for the Activated Sludge Process. *Instrumentation Technology*, July, 35-43.

Fikar_ M., Chachuat, B. and Lati, M. A. 2005. Optimal Operation of Alternating Activated Sludge Processes. Journal Title Control engineering practice ISSN 0967-0661. Vol. 13, no7, pp. 853-861.

Funahashi, K., 1989. On the approximate realization of continuous mappings by neural networks. Neural Networks, 2:183-192.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma". Neural Computation, 4, 1-58.

Grieu, S., Thiery, F., Traore, A., Tien, P.N., Barreau, M., and Polit, M., 2006. *KSOM and MLP neural networks for on-line estimating the efficiency of an activated sludge process. Chemical engineering journal ISSN 1385-8947.* 2006, vol. 116, no1, pp. 1-11 [11 page(s) (article)] (19 ref.)

Gujer, W., Henze, M., Mino, T. and van Loosdrecht, M.C.M. 1999. *A modified Activated Sludge Model No. 3 (ASM3) with two-step nitrification-denitrification.* IAWQ Scientific and Technical Report No.3 IAWQ, London

Hagan, M.T., H.B. Demuth, and M. Beale, 1996. *Neural Network Design*, Boston, Massachusetts, PWS.

Halvarsson, B. 2003. Applications of Coupling Analysis on Bioreactor Models. MSc thesis, Uppsala University, Sweden.

Hartman, E., Keeler, K., and Kowalski, J.M.1989. Layered neural networks with Gaussian hidden units as universal approximators. (Submitted for publication).

Haykin, S., 1994. Neural Networks. A Comprehensive Foundation. Macmillan, New York, NY.

Hebb, D. O., 1949. *The Organization of Behavior.* Wiley

Hecht-Nielsen, Robert, "Kolmogorov, I987. Mapping Neural Network Existence Theorem. Proceeding of IEEE International Conference on Neural *Network,* IEEE Press, New York, III (11-13), 1987.

Henze, M., Grady Jr. C.P.L., Gujer W., Marais G.R., and Matsuo, T. 1987. *Activated Sludge Model No1.* IAWQ Scientific and Technical Report No.1 IAWQ, London, UK.

Henze, M., Gujer W., Takahasi Matsuo, T. Wentzel, M.C., and Marais G.R., and 1995. *Activated Sludge Model No2.* IAWQ Scientific and Technical Report No.3 IAWQ, London, UK.

Herremans C., Ryckaert V. and Van Impe J., 1997. Calculation of carbon addition during biological nitrogen removal by using optimal control theory. In: Proceedings 11th Forum Applied Biotechnology. Med. Fac. Landbouww. Univ. Gent, 62/4b, 1691-1694.

Holenda, B., Domokos E., Rédey, Á. and Fazakas, J, 2007. Dissolved oxygen control of the activated sludge process using model predictive control. *Computers and Chemical Engineering*, Vol. 32, pp 1270-1278

Holmberg, U., 1990. On the identifiability of dissolved oxygen concentration dynamics, in: IAWPRC's 25th Anniversary Conference and Exhibition, Kyoto, Japan.

Hopfield, J.J., 1982. Neural network and physical system with emergent collective computational capabilities. In *the proceeding of the National Academy of Sciences (USA)* 79, 2554-2558.

Hornik, K., Stinchcombe, M., and White, H., 1988. *Multilayer Feedforward Networks Are Universal Approximators.* Manuscript, Department of Economics, University of California at San Diego, June 1988.

Howard Demuth, and Mark Beale, "*Neural Network Toolbox: For Use with MATLAB*", Version 4, *the Math Works, Inc., 2002.*

Hunt KJ, Sbarbaro D, Zbikowski R, Gawthrop PJ. 1992. Neural network for control systems—A survey. Automatica; 28:1083–1112.

Hussain, Mohamed Azlan., 1999: Review of the applications of neural networks in chemical process control - simulation and online implementation. *Artificial Intelligence Engineering.* 13, No. 1, 55-68 (1999).

Ingildsen, P., 2002. Realizing Full-Scale Control in Wastewater Treatment Systems Using In Situ Nutrient Sensors, PhD Thesis, Lund University,

Isidori, A., 1989).*Nonlinear Control Systems*, Second Ed. Springer, New York.

Jagannathan S. Neural Network Control of Nonlinear Discrete Time Systemscopyright © 1997-2008 Culinary and Hospitality Industry Publications Services.

Jeppsson, U. 1996. "Modelling Aspects of Wastewater Treatment Process", PhD Thesis, Dept. of Industrial Electrical Engineering and Automation, Lund University, Sweden.

Jordan MI, Rumelhart DE (1992) Forward models: supervised learning with a distal teacher. Cognition Science 16:307–354

Kohonen, T., 1989. *Self-Organization and Associate Memory*, Third edition, New York, Springer-Verlag.

Koistinen, P. and Holmström, L., 1992; Kernel regression and backpropagation training with noise," NIPS4, 1033-1039.

Kriger, C., Tzoneva, R., 2007: A neural network model for control of wastewater treatment processes Nonlinear Control Systems, Volume # 7 Part# 1 Proceedings at CSIR International Convention Centre, South Africa *(IFAC-papers on-line*).

Kunwar P. Singh, Ankita Basant, Amrita Malik, Gunja Jain., 2009. ANNs modelling of the river water - A case study. Environmental Chemistry Division, Indian Institute of Toxicology Research, Post Box 80, MG Marg, Lucknow 226 001, India. Journal homepage: www.elsevier.com/locate/ecolmodel.

Lech R.F., Lim H.C., Grady, C. P. L. Jr. and Koppel L.B., 1978a. Automatic control of the activated sludge process. I. Development of a simplified dynamic model. Water. Res., 12, 81-90.

Lech R.F., Grady C. P .L. Jr., Lim H.C. and Koppel L.B., 1978b. Automatic control of the activated sludge process. II. Efficacy of control strategies. Water. Res., 12, 91-99.

Levenberg, K. A Method for the Solution of Certain Non-linear Problems in Least Squares. *Quart. Appl. Math.* **1944**, *2*, 164-168.
Lindberg, C.F. and Carlsson, B., 1996. Adaptive control of external carbon flow rate in an activated sludge process. *Water Science and Technology* **34** 3/4, pp. 173–180

Lindberg, C. F., 1997. Control and Estimation Strategies Applied to the Activated Sludge Process, PhD thesis, Uppsala University, Sweden.

Lippmann, R. P., 1987. An introduction to computing with neural nets. IEEE ASSP Magazine. pp: 4-22.

Ljung, L., and Gunnarsson, S., 1990. Adaptation and tracking in system identification – a survey", Automatica, 26, pp. 7–21, 1990.

Marquardt, D.W. An Algorithm for the Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal. Application. Math.* **1963**, *11*, 431–441.

McCulloch, W. S. and Pitts, W. H., 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133.

McLellan, P.J., Harris, T. J., and Bacon, D. W., Error Trajectory Descriptions of Nonlinear Controller Designs. Chem. Eng. Sci., 45(10), 3017 (1990)

Minsky, M. and Papert, S., 1969. *Perceptrons*. MIT Press, Cambridge, MA.

Mohamed T. S. and Laila M.F. 2005. Application of Activated Sludge Models in. Traditionally Operated Treatment Plants - A Software Environment Overview. The *Water Quality Research Journal of Canada*, vol. 39, no3, pp. 294-302. (www.cciw.ca/wqrjc).

Moller, M. F. 1993. Neural Networks, Vol.6, pp. 525-533, 1993.

Moody, J., and Darken, C.J., 1989, "Fast learning in neural networks of locally- tuned processing units", *Neural Computation*, **1**, 281-294.

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", NIPS 4, 847-854.

Morari and Zafiriou, 1989. M. Morari and E. Zafiriou. *Robust process control*, Prentice-Hall, Englewood Cliffs, New York (1989).

Moscinski, J., Ogonowski, Z., 1995: Advanced control with Matlab/Simulink. Hertfordshire: Prentice Hall Europe, 1995. p.34

Murnleitner, E., Kuba, T., van Loosdrecht, M.C.M. and Heijnen, J.J. 1996. An integrated metabolic model for the aerobic and denitrifying biological phosphorus removal process, Biotechnology. Bioengineering.

Narendra, K. S., 1990. Parthasarathy. Identification and control of dynamical systems using neural networks. IEEE Transaction on Neural Networks, 1(1), 4–27.

Nguyen, D., Widrow, B., 1990. Improving the learning speed of two-layer neural networks by choosing initial values of the adaptive weights. In: Proceedings of an IEEE International Joint Conference on Neural Networks, San Diego, CA, and pp 21–26.

Nielsen M.K. and Önnerth T.B., 1995. Improvement of a recirculating plant by introducing STAR control. Water. Sci. Tech., 31(2), 171-180.

Nikolaou, M., and Hanagandi, V. 1993. Control of nonlinear dynamical systems modelled by recurrent neural networks. *Artificial Intelligence in Chemical Engineering journal* ISSN 0001-1541 1993, vol. 39, no11, pp. 1890-1894 (18 ref.)

Nigrin, A., 1993. Neural networks for pattern recognition MIT Press, Cambridge, 413 pp., ISBN 0-262-14054-3

Norgaard, M., 2000. Neural Network Based Control System Design Toolkit, ver.2 Tech. Report. 00-E-892, Department of Automation, Technical University of Denmark.

Olsson, G., 1993. Reduced order models for on-line parameter identification of the activated sludge process. *Water Science Technology* **28** 11/12, pp. 173–183.

Olsson, G., Spanjers, H., Vanrolleghem, Peter A. and Dold, P. L.1998. Respirometry in control of the activated sludge process: principles Scientific and technical report No.7, University of Ottawa, Canada.

Olsson, G. and Newell, B.1999. Wastewater Treatment Systems: Modelling, Diagnosis and Control", IAW Publishing, UK

Pai, T.Y. Tsai, Y.P., Lo, H.M., Tsai, C.H., and Lin, C.Y., 2006. Grey and neural network prediction of suspended solids and chemical oxygen demand in hospital wastewater treatment plant effluent. *Computers and Chemical Engineering* 31 (2007) 1272–1281

Patterson, D., 1996. Artificial Neural Networks. Singapore: Prentice Hall.

Petersen, B., Gernaey, G., and Vanrolleghem, P.A. 2001. Practical identifiability of model parameters by combined respirometric-titrimetric measurements. *Water Sci. Technol.* 43: 347–355.

Pham, D. T., and Liu, X., 1996. Training of Elman networks and dynamic system modelling, *Int. J. Systems Science* **27**(2), 221-226 (1996).

Pons, M. N., Spanjers H. and Jeppsson U., 1999: Towards a benchmark for evaluating controls strategies in wastewater treatment plants by simulation. *Proceedings of 9th European Symposium on Computer Aided Process Engineering*, Budapest. Hungary, May 31-June 2, 1999.

Ra´duly, B., Gernaey, K.V., Capodaglio, A.G., Michelson, P.S., and Henze, M., 2006 artificial neural networks for, rapid wastewater performance evaluation: Methodology and case study. Environmental Modelling & Software 22 (2007) 1208-1216. www.elsevier.com/locate/envsoft

Riedmiller, Martin and Braun H., 1994: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of the IEE International conference on Neural Networks (ICNN)*, pp 586-591, San Francisco.

Rieger, L., Koch, G., Kühni, M., Gujer, W. and Siegrist, H. 2001. *The EAWAG Bio-P Module for Activated Sludge Model No. 3*, Water. Resources, 35(16), 3887-3903

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, **65**, 386-408.

Samuelsson, P. 1998. A Java based activated sludge process simulator, Master's thesis, Systems and Control Group, School of Engineering, Uppsala University,

Schalkoff, R.J., 1997. Artificial Neural Networks. McGraw-Hill, New York.

Slotine, J.-J.E. and Li, W., 1991. *Applied Nonlinear Control*. Prentice-Hall, Englewood Cliffs, NJ.

Smith, M. (1993), *Neural Networks for Statistical Modelling*, NY: Van Nostrand Reinhold

Soderstrom, T., and Stoica, P. 1989. *System Identification*. Prentice Hall, 1989.

Sotomayor, O.A.Z., Park, S.W., Garcia, C., 2001. A Simulation Benchmark to evaluate the performance of Advanced Control techniques In Biological Wastewater Treatment Plants. *In the proceeding of Braz. J. Chem. Eng*, vol.18 no.1 in São Paulo.

Spanjers H., Vanrolleghem P.A., Olsson G. and Dold P.L., 1998b. Respirometry in Control of the Activated Sludge Process: Principles. IAWQ Scientific and Technical Report No. 7. London, UK.

Steffens, M. A., and Lant P. A., 1998. "Multivariable control of nutrient-removing Activated sludge systems" Water Research. Vol. 33, No. 12, pp. 2864-2878, 1999 Elsevier Science Ltd.

Steyer J.-P., Esteban M. and Polit M., 1997. Fuzzy control of an anaerobic digestion process for the treatment of an industrial wastewater. In: Proceedings 6th International Conference on Fuzzy Systems - FUZZ-IEEE'97. Barcelona, Spain. July 1-51997. Vol. III, 1245-1250.

Stinchcombe, M., and White, H., 1989. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In Proceedings of the International Joint Conference on Neural Networks, pages I-607-I-611, Washington D.C. IEEE TAB Neural Network Committee.

Sundarambal P., Shie-Yui, L., Pavel, .T, and Jegathambal, P. 2008: Development of a network model for dissolved oxygen in seawater. Indian Journal of Marine Sciences Vol.38 (2), June 2009, pp.151-159.

Yoshio, H., Yamashita, K., and Hijiya, S., 1991; *Back-propagation algorithm which varies the number of hidden units*, pp. 61-66. Neural Networks, volume 4 (1991), number 1

Vanrolleghem P.A., Van Daele M. 1994. Optimal Experimental Design for Structure Characterization of Biodegradation Models: On-line Implementation in a respirographic Biosensor. *Water Science Technology, IAWQ*, Vol.34, No.1-2, pp213-220
Weijers S.R. (2000) Modelling, identification and control of activated sludge processes for nitrogen removal. PhD. Thesis, Technical University of Eindhoven, the Netherlands. pp. 235

Wen, C.H. and Vassiliadis, C.A., 1998. Applying hybrid artificial intelligence techniques in Wastewater treatment. *Engineering Applications of Artificial Intelligence*, Vol. 11, pp. 685-705.

Widrow, B., and Hoff, M. T., 1960. Adaptive Switching Circuits, IRE WESCON convention Record, Part 4, 96-104.

Willis, M.J. Montague, G.A. Massimo, C. Di. Tham M.T., and A.J. Morris., 1992. Artificial neural networks in process estimation and control. *Automatica* **28** (1992), pp. 1181–1187.

Zurada, J.M., 1992. Introduction to Artificial Neural Systems. PWS Publishing Company. ISBN 0-534-95460-X

APPENDICES

APPENDIX A1

MATRIX REPRESENTATION OF THE BIOLOGICAL MODEL ASM1 (HENZE ET AL., 1987)

The ASM1 – Matrix Format

Component → i	1	2	3	4	5	6	7	8	9	10	11	12	13	Process Rate, ρ_j [ML^{-3}T^{-1}]
j Process ↓	S_I	S_S	X_I	X_S	$X_{B,H}$	$X_{B,A}$	X_P	S_O	S_{NO}	S_{NH}	S_{ND}	X_{ND}	S_{ALK}	
1 Aerobic growth of heterotrophs		$\frac{1}{Y_H}$			1			$\frac{1-Y_H}{Y_H}$		$-i_{XB}$			$\frac{i_{XB}}{14}$	$\hat{\mu}_H\left(\frac{S_S}{K_S+S_S}\right)\left(\frac{S_O}{K_{O,H}+S_O}\right)X_{B,H}$
2 Anoxic growth of heterotrophs		$\frac{1}{Y_H}$			1				$\frac{1-Y_H}{2.86Y_H}$	$-i_{XB}$			$\frac{1-Y_H}{14\cdot 2.86Y_H}-\frac{i_{XB}}{14}$	$\hat{\mu}_H\left(\frac{S_S}{K_S+S_S}\right)\left(\frac{K_{O,H}}{K_{O,H}+S_O}\right)\left(\frac{S_{NO}}{K_{NO}+S_{NO}}\right)\eta_g X_{B,H}$
3 Aerobic growth of autotrophs						1		$-\frac{4.57}{Y_A}+1$	$\frac{1}{Y_A}$	$-i_{XB}-\frac{1}{Y_A}$			$-\frac{i_{XB}}{14}-\frac{1}{7Y_A}$	$\hat{\mu}_A\left(\frac{S_{NH}}{K_{NH}+S_{NH}}\right)\left(\frac{S_O}{K_{O,A}+S_O}\right)X_{B,A}$
4 'Decay' of heterotrophs				$1-f_P$	-1		f_P					$i_{XB}-f_P i_{XP}$		$b_H X_{B,H}$
5 'Decay' of autotrophs				$1-f_P$		-1	f_P					$i_{XB}-f_P i_{XP}$		$b_A X_{B,A}$
6 Ammonification of soluble organic nitrogen										1	-1		$\frac{1}{14}$	$k_a S_{ND} X_{B,H}$
7 'Hydrolysis' of entrapped organics		1		-1										$k_h\frac{X_S/X_{B,H}}{K_X+(X_S/X_{B,H})}\left[\left(\frac{S_O}{K_{O,H}+S_O}\right)+\eta_h\left(\frac{K_{O,H}}{K_{O,H}+S_O}\right)\left(\frac{S_{NO}}{K_{NO}+S_{NO}}\right)\right]X_{B,H}$
8 'Hydrolysis' of entrapped organic nitrogen											1	-1		$\rho_7(X_{ND}/X_S)$
Observed Conversion Rates [ML^{-3}T^{-1}]	$r_i=\sum_j \nu_{ij}\rho_j$									$r_i=\sum_j \nu_{ij}\rho_j$				
Stoichiometric Parameters: Heterotrophic yield: Y_H Autotrophic yield: Y_A Fraction of biomass yielding particulate products: f_P Mass N/Mass COD in biomass: i_{XB} Mass N/Mass COD in products from biomass: i_{XP}	Soluble inert organic matter [M(COD)L^{-3}]	Readily biodegradable substrate [M(COD)L^{-3}]	Particulate inert organic matter [M(COD)L^{-3}]	Slowly biodegradable substrate [M(COD)L^{-3}]	Active heterotrophic biomass [M(COD)L^{-3}]	Active autotrophic biomass [M(COD)L^{-3}]	Particulate products arising from biomass decay [M(COD)L^{-3}]	Oxygen (negative COD) [M(-COD)L^{-3}]	Nitrate and nitrite nitrogen [M(N)L^{-3}]	NH_4+NH_3 nitrogen [M(N)L^{-3}]	Soluble biodegradable organic nitrogen [M(N)L^{-3}]	Particulate biodegradable organic nitrogen [M(N)L^{-3}]	Alkalinity – Molar units	Kinetic Parameters: Heterotrophic growth and decay: $\hat{\mu}_H$, K_S, $K_{O,H}$, K_{NO}, b_H Autotrophic growth and decay: $\hat{\mu}_A$, K_{NH}, $K_{O,A}$, b_A Correction factor for anoxic growth of heterotrophs: η_g Ammonification: k_a Hydrolysis: k_h, K_X Correction factor for anoxic hydrolysis: η_h

APPENDIX A2: Script file: parmeters_DO.m

MATLAB PROGRAM TO SIMULATE THE DO PROCESS USING ASM1 STEADY STATE PARAMETERS

```
clear all
%Parameters for the design of the DO_MODEL2_2 for the input airflow
%trajectory and the DO process trajectory.
%This m-file is used to generate data for training all the neural
networks
%developed. The data is stored in a file called data_tr.
%All the parameters are taken from the steady state ASM1 data.

%Julius N'gon'ga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Sosat=8%saturated DO concentration
K1=240%exponential constant of KLa
K2= 1.0961e-004%exponential constant of KLa
Soo=0.491
V=1333%volume of the aeration tank
Soin=0.2;
t=0:1.041667e-4:0.1;
u=50000+1000.*randn(1,960);
Qq=55338
Qr=18446
Qo=18446
Q=Qq+Qr+Qo
D=Q/V
a=-0.02;
b=200;
muA=0.5
muH=4
Koh=0.20
Yh=0.67
Ya=0.24
Ks=10
Knh=1.0
Koa =0.4
Xbhs=2559.344
Xbas=149.789
Snhs=1.733
Sss=0.889
Snh=Snhs*ones(1,960);
Ss=Sss*ones(1,960);
Xba=Xbas*ones(1,960);
Xbh=Xbhs*ones(1,960);
Z=t';
U=u';
Ass=Ss';
Asnh=Snh';
Axba=Xba';
Axbh=Xbh';
KLa=K1*(1-exp(-K2*u));%oxygen mass transfer coefficient
rso=-muH*((1-Yh)/Yh)*(Sss/(Ks+Sss))*(Soo/(Koh+Soo))
*Xbhs+(-muA*((4.57-Ya)/Ya))*(Snhs/(Knh+Snhs))*(Soo/(Koa+Soo))*Xbas;
rsol=rso*ones(1,960);%oxygen uptake rate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
Kp=4+a;%maximum proportional gain
P=0.5*Kp;%proportional gain of the PI controller
Ti=(4+a)/6;%integral time constant of the PI controller
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r=2*ones(1,960)
Ref=r';%set-point
Soin1=0.2*ones(1,960);%the DO concentration in the inputflow
Soin11=Soin1';
i1=U(:,1);%airflow rate
i_1max=max(i1);%maximum airflow rate
i1_1=i1./i_1max;%normalized airflow rate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DO_Model2_2;%Model developed for the DO process
%To simulate the response characteristic of this model,
%load data-tr file to the command window
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
i2;%dissolved oxygen trajectory
i_2max=max(i2);%maximum value of dissolved oxygen trajectory
i2_2=i2./i_2max;%normalized dissolved oxygen trajectory
%end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

APPENDIX B:

TABLE GIVING THE SIMULINK MODELS DEVELOPED

This section presents a table of the algorithms and Simulink models developed

MODEL	NAME	FUNCTION
DO_MODEL2_2.mdl	DO process parameters obtained from ASM1	Simulation of the Simulink DO model trajectory
DESIRED.mdl	Implementation of desired reference model in closed loop	Simulation of the Simulink desired reference DO model trajectory
DESIREDNONLINEAR.mdl	Implementation of nonlinear linearizing control by Feedforward neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
DESIREDNONLINEARelm.mdl	Implementation of nonlinear linearizing control by Elman neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
DESIREDNONLINEAR6745elm.mdl	Implementation of feedforward nonlinear linearizing control by neural network in Simulink	Simulation of the Simulink closed loop control trajectory of the DO process
INVERFF1.mdl	Implementation of direct inverse control by Feedforward neural network	Simulation of the Simulink closed loop control trajectory of the DO process
INVERELM_model2.mdl	Implementation of direct inverse control by Elman neural network	Simulation of the Simulink closed loop control trajectory of the DO process
INVERFFimcmodel2.mdl	Implementation of internal model control by Feedforward neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
INVERELM_imcmodel2.mdl	Implementation of internal model control by Elman neural network in closed loop	Simulation of the Simulink closed loop control trajectory of the DO process
Program (script file)	**NAME**	**FUNCTION**
Parameters_DO.m	Steady state parameters or the ASM1 model	script file for simulation of the DO process using ASM1steady state parameters
Model1.m	Script file for NN training with one input and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Model2.m	Script file for NN training with two input and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Model3.m	Script file for NN training with three inputs and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Model4.m	Script file for NN training with twelve inputs and one output	Identifying the DO process using neural networks (feedforward and recurrent)
Nonlinearcon.m	Script file for simulation of nonlinear controller function	simulation of nonlinear controller function
Nonlinearconelm.m	Script file for Elman NN nonlinear linearizing controller	simulation of Elman NN nonlinear controller function
Nonlinearconff.m	Script file for feed forward NN nonlinear linearizing controller	simulation of feedforward NN nonlinear controller function
BESTFF.m	Script file for feed forward NN training with six inputs and one output	Forward identification of feed forward NN with six inputs and one output
BESTELM.m	Script file for Elman NN training with six inputs and one output	Forward identification of feed Elman NN with six inputs and one output
INVERELM.m	Script file for Elman NN training	Inverse identification of feed Elman NN
INVERFF.m	Script file for inverse feedforward NN training	Inverse identification of feed Elman NN
Parametersforu1.m	Script file for nonlinear function u1	Simulation of the nonlinear function u1

APPENDIX C:

SOFTWARE ROUTINES

This section presents the implemented MATLAB software algorithms that are used in the development of the neural networks for identification and for control of the dissolved oxygen concentration trajectories. The topologies considered are both the feedforward and the Elman networks. The description of these script files are contained in chapters 4, 5, and 6 as well as the Table in appendix B.

C.1: MATLAB script-feedforward/Elman model –model1.m

```
%Design a feedforward/recurrent NN with 1 inputs, 1
%hidden layer with (2,3,4,5,10,15,20) neurons and 1 target output
neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the (scaled conjugate gradient, levenberg-
marquardt
%and resilient backpropagation) algorithm
close all;
clear all;
load data_tr;
i1=U(:,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Normalize the inputs in the range (0,1)
i_1max=max(i1)
i1_1=i1./i_1max
%Plot the normalized inputs
figure(1)
plot(Z,i1_1,'k-');
grid
xlabel('time(days)')
ylabel('Flow rate(m3/day)')
title('Daily flow rate')
i2=so(:,1)
%Normalize the outputs in the range (0,1)
i_2max=max(i2)
i2_2=i2./i_2max
%Plot the normalized outputs
figure(2);
plot(Z,i2_2,'k-')
grid on
xlabel('time(days)')
ylabel('so')
title('dissolved oxygen concentration')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pr=[i1_1]';%inputs
Tr=[i2_2]';%target
net=newff(minmax(Pr),[2 1],{'tansig','purelin'},'trainscg');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.epochs=10000;
net.trainParam.show=50;
net.trainParam.goal=1e-6;
net1=train(net,Pr,Tr);
g1=input('Stike any key....');
```

APPENDIX C:

SOFTWARE ROUTINES

This section presents the implemented MATLAB software algorithms that are used in the development of the neural networks for identification and for control of the dissolved oxygen concentration trajectories. The topologies considered are both the feedforward and the Elman networks. The description of these script files are contained in chapters 4, 5, and 6 as well as the Table in appendix B.

C.1: MATLAB script-feedforward/Elman model –model1.m

```
%Design a feedforward/recurrent NN with 1 inputs, 1
%hidden layer with (2,3,4,5,10,15,20) neurons and 1 target output
neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the (scaled conjugate gradient, levenberg-
marquardt
%and resilient backpropagation) algorithm
close all;
clear all;
load data_tr;
i1=U(:,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Normalize the inputs in the range (0,1)
i_1max=max(i1)
i1_1=i1./i_1max
%Plot the normalized inputs
figure(1)
plot(Z,i1_1,'k-');
grid
xlabel('time(days)')
ylabel('Flow rate(m3/day)')
title('Daily flow rate')
i2=so(:,1)
%Normalize the outputs in the range (0,1)
i_2max=max(i2)
i2_2=i2./i_2max
%Plot the normalized outputs
figure(2);
plot(Z,i2_2,'k-')
grid on
xlabel('time(days)')
ylabel('so')
title('dissolved oxygen concentration')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pr=[i1_1]';%inputs
Tr=[i2_2]';%target
net=newff(minmax(Pr),[2 1],{'tansig','purelin'},'trainscg');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.epochs=10000;
net.trainParam.show=50;
net.trainParam.goal=1e-6;
net1=train(net,Pr,Tr);
g1=input('Strike any key....');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%simulate the network
a=sim(net1,Pr);
figure(3)
hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot the error,target and network output for comparison
plot(Z,Tr,'k-','markersize',2)
plot(Z,a,'k+:','markersize',3);
plot(Z,a-Tr,'k*','markersize',3)
grid
xlabel('time(days)');
ylabel('Output of the network,so and error of(so)');
title('dissolved oxygen concentration ');
legend('Training-Data','NN output','Error')
hold off
%end
```

C.2: MATLAB script-feedforward/Elman model –model2.m

```
%Design a feedforward/recurrent NN with 2 inputs, 1
%hidden layer with (2, 3, 4, 5, 10, 15, 20) neurons and 1 target
%output neuron
%Use tan-sigmoid activation function for the hidden layer and pure...
%transfer function for the output layer
%Train the network using the (scaled conjugate gradient, levenberg
marquardt
%and resilient backpropagation) algorithm
close all;
clear all;
load data_tr;
i_1max=max(i1)
i1_1=i1./i_1max
figure(1)
plot(Z,i1_1);
grid
xlabel('time(days)')
ylabel('Flow rate(um3/day)')
title('Daily flow rate')
i2;
i_2max=max(i2)
i2_2=i2./i_2max
figure(2);
plot(Z,i2_2)
grid on
xlabel('time(days)')
ylabel('so')
title('dissolved oxygen concentration')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%creates input and target variables to the network
P=[i2_2 Z]';%inputs
T=[i1_1]';%targets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Training the network with scaled conjugate algorithm,levenberg-
marquardt
%and resilient backpropagation algorithm
net=newelm(minmax(P),[3 1],{'tansig','purelin'},'trainlm');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.show=25;
net.trainParam.epochs=10000;
net.trainParam.goal=1e-6;
net2=train(net,P,T);
```

```
g1=input('Stike any key....');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%simulate the network
a=sim(net2,P);
figure(3)
hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot the target, error and the NN output for comparison
plot(Z,T,'b','markersize',3)
plot(Z,a,'r','markersize',3);
plot(Z,a-T,'g','markersize',3)
grid
xlabel('time(days)');
ylabel('Output of the network,so and error of(so)');
title('dissolved oxygen concentration ');
legend('Training-Data','NN output','Error')
hold off
%end
```

C.3: MATLAB script-feedforward/Elman model –model3.m

```
%Design a feedforward/recurrent NN with 3 inputs, 1
%hidden layer with (2,3,4,5,10,15,20) neurons and 1 target output
neuron
%Use tan-sigmoid activation function for the hidden layer and purel
%transfer function for the output layer
%Train the network using the (scaled conjugate gradient, levenberg-
marquardt
%and resilient backpropagation) algorithm
close all;
clear all;
load data_tr;
i_1max=max(i1)
i1_1=i1./i_1max
figure(1)
plot(Z,i1_1);
grid
xlabel('time(days)')
ylabel('Flow rate(um3/day)')
title('Daily flow rate')
i2;
i_2max=max(i2)
i2_2=i2./i_2max
figure(2);
plot(Z,i2_2)
grid on
xlabel('time(days)')
ylabel('so')
title('dissolved oxygen concentration')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create input and target variables to the network
P=[i2_2 i1_1 Z]';%inputs
T=[i1_1]';%targets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Training the network with scaled conjugate algorithm, Levenberg-
marquardt
%and resilient backpropagation algorithm
net=newelm(minmax(P),[3 1],{'tansig','purelin'},'trainlm');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.show=25;
net.trainParam.epochs=10000;
net.trainParam.goal=1e-6;
```

```
net=train(net,P,T);
g1=input('Stike any key....');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%simulate the network
a=sim(net,P);
figure(3)
hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot the target, error and the NN output for comparison
plot(Z,T,'b','markersize',3)
plot(Z,a,'r','markersize',3);
plot(Z,a-T,'g','markersize',3)
grid
xlabel('time(days)');
ylabel('Output of the network,so and error of(so)');
title('dissolved oxygen concentration ');
legend('Training-Data','NN output','Error')
hold off
%end
```

C.4: MATLAB script-feedforward/Elman model –model4.m

```
%Design a feedforward/recurrent NN with 12 inputs, 1
%hidden layer with (2,3,4,5,10,15,20) neurons and 1 target output
neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the (scaled conjugate gradient,levenberg-
marquardt
%and resilient backpropagation) algorithm
close all;
clear all;
load data_tr;
i_1max=max(i1)
i1_1=i1./i_1max
figure(1)
plot(Z,i1_1);
grid
xlabel('time(days)')
ylabel('Flow rate(um3/day)')
title('Daily flow rate')
i2;
i_2max=max(i2)
i2_2=i2./i_2max
figure(2);
plot(Z,i2_2)
grid on
xlabel('time(days)')
ylabel('so')
title('dissolved oxygen concentration')
length(i1_1);
k=i1_1(1:960);
k0=[0;k(1:959)];
k1=[0;0;k(1:958)];
k2=[0;0;0;k(1:957)];
length(Z);
m=Z(1:960);
m0=[0;m(1:959)];
m1=[0;0;m(1:958)];
m2=[0;0;0;m(1:957)];
i2_2;
n=i2_2(1:960);
n0=[0;n(1:959)];
```

```
n1=[0;0;n(1:958)];
n2=[0;0;0;n(1:957)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create input and target variables to the  network
P=[i2_2 n0 n1 n2 i1_1 k0 k1 k2 Z m0 m1 m2]';%inputs
T=[i1_1]';%targets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Training the network with scaled conjugate algorithm,levenberg-
marquardt
%and resilient backpropagation algorithm
net=newff(minmax(P),[2 1],{'tansig','purelin'},'trainlm');
net.trainParam.mem_reduc=2;
net.trainParam.lr=0.05;
net.trainParam.show=25;
net.trainParam.epochs=10000;
net.trainParam.goal=1e-6;
net=train(net,P,T);
g1=input('Stike any key....');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%simulate the network
a=sim(net,P);
figure(3)
hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot the target, error and the NN output for comparison
plot(Z,T,'b','markersize',3)
plot(Z,a,'r','markersize',3);
plot(Z,a-T,'g','markersize',3)
grid
xlabel('time(days)');
ylabel('Output of the network,so and error of(so)');
title('dissolved oxygen concentration ');
legend('Training-Data','NN output','Error')
hold off
%end
```

C.5: MATLAB script-feedforward model - BESTFF.m

```
%Design a feedforward NN with 6 inputs, 1 Hidden layer with 8
%neurons and 1 target output neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the Levenberg marquardt algorithm
%Use early stoppage training to prevent over fitting by dividing the
data
%up into training, validation and test sets.

%Julius N'gon'ga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
load data_tr;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create delayed values of the input variables and output variables
%Delayed values of i1_1
k=i1_1(1:960);
k0=[0;k(1:959)];
k1=[0;0;k(1:958)];
k2=[0;0;0;k(1:957)];
%Delayed values of i2_2
n=i2_2(1:960);
```

```
n0=[0;n(1:959)];
n1=[0;0;n(1:958)];
n2=[0;0;0;n(1:957)];
P1=[i1_1 k0 k1 i2_2 n0 n1]';%input variables to the NN
T1=[i2_2]';%NN target
save data_tr
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Divide the data up into training, validation and test sets.
%The testing set will start with the second point and take
%every fourth point.  The validation set will start with the
%fourth point and take every fourth point.  The training set
%will take the remaining points.
[R,T] = size(P1);
iitst = 2:4:T;
iival = 4:4:T;
iitr = 1:2:T;
validation.P = P1(:,iival);
validation.T = T1(:,iival)
testing.P = P1(:,iitst)
testing.T = T1(:,iitst)
ptr = P1(:,iitr)
ttr = T1(:,iitr)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause % Strike any key to continue...
net1 = newff(minmax(ptr),[8 1],{'tansig' 'purelin'},'trainlm')
%Create a feedforward network with 8 hidden neurons, 1 output
% neurons, TANSIG hidden neurons and linear output neurons
[net1,tr]=train(net1,ptr,ttr,[],[],validation,testing)
net1.trainParam.show=5;% Show intermediate results every five
iterations.
%Training begins...please wait...
%Train the network.  We use early stopping, so we are passing the
%validation data.  We also want the errors computed on a test
%set, so we are passing the testing data.
err=T1-sim(net1,P1)
%bar(err)
g1=input('Stike any key....');
%Simulate the trained network.
an=sim(net1,P1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Network weights and biases
net1;
w=net1.IW{1,1}
bw=net1.b{1}
v=net1.LW{2,1}
bv=net1.b{2}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the training, validation and test errors
figure(3)
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-
.b')
legend('Training','Validation','Test',-1);
xlabel('Epoch')
ylabel('Squared Error')
pause % Strike any key to display the regression analys s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(4)
%display plots showing regression analyses between the
%network outputs and the corresponding targets (in original units
[m,b1,r1] = postreg(an,T1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the target,NN simulated target and the error
figure(5)
hold on
```

```
plot(Z,T1,'b','markersize',6)
plot(Z,an,'r','markersize',6);
plot(Z,err,'g','markersize',6)
grid;
xlabel('time(days)');
ylabel('Output of the network,so and error of(so)');
title('dissolved oxygen concentration ');
legend('Training-Data','NN output','Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Denormalize the target in original units
y1 = an'.*i_2max;
figure(6)
hold on
plot(Z,y1,'markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the denormalized(so)');
title('dissolved oxygen concentration ');
legend('denormalized(so)')
hold off;
%end
```

C.6: MATLAB script - Elman model - BESTELM.m

```
%Design an Elman NN with 6 inputs, 1 Hidden layer with 6
%neurons and 1 target output neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the Levenberg marquardt algorithm
%Use early stoppage training to prevent over fitting by Dividing the
data
%up into training, validation and test sets

%Julius N'gon'ga Muga
%Cape Peninsula University of Technology
%2009

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
load data_tr;
%Create delayed values of the input variables and output variables
%delayed values of input flow rate i1_1
k=i1_1(1:960);
k0=[0;k(1:959)];
k1=[0;0;k(1:958)];
k2=[0;0;0;k(1:957)];
%delayed values of dissolved oxygen i2_2
n=i2_2(1:960);
n0=[0;n(1:959)];
n1=[0;0;n(1:958)];
n2=[0;0;0;n(1:957)];
P1=[i2_2 n0 n1 i1_1 k0 k1]';%input values to the NN
T1=[i2_2]';%NN target
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Divide the data up into training, validation and test sets.
%The testing set will start with the second point and take
%every fourth point.  The validation set will start with the
%fourth point and take every fourth point.  The training set
%will take the remaining points.
[R,T] = size(P1);
iitst = 2:4:T;
```

```
iival = 4:4:T;
iitr = 1:2:T;
validation.P = P1(:,iival);
validation.T = T1(:,iival)
testing.P = P1(:,iitst)
testing.T = T1(:,iitst)
ptr = P1(:,iitr)
ttr = T1(:,iitr)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause % Strike any key to continue.
net2 = newelm(minmax(ptr),[6 1],{'tansig' 'purelin'},'trainlm')
%Create an elman network with 6 hidden neurons, 1 output
% neurons, TANSIG hidden neurons and linear output neurons.
[net2,tr]=train(net2,ptr,ttr,[],[],validation,testing)
%net1.trainParam.mem_reduc=2;
%net1.trainParam.lr=0.05;
net2.trainParam.show=5;% Show intermediate results every five
iterations.
%Training begins...please wait...
%Train the network.  We use early stopping, so we are passing the
%validation data.  We also want the errors computed on a test
% set, so we are passing the testing data.
err2=T1-sim(net2,P1)
%bar(err)
g1=input('Stike any key....');
%Simulate the trained network.
a2=sim(net2,P1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Network weights and biases
net2;
w1=net2.IW{1,1};
bw1=net2.b{1};
v1=net2.LW{2,1};
bv1=net2.b{2};
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the training, validation and test errors.
figure(3)
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,'.g',tr.epoch,tr.tperf,'-
.b')
legend('Training','Validation','Test',-1);
xlabel('Epoch')
ylabel('Squared Error')
pause % Strike any key to display the regression analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(4)
%display plots showing regression analyses between the
%network outputs and the corresponding targets (in original units).
[m,b2,r2] = postreg(a2,T1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the target,NN simulated target and the error
figure(5)
hold on
plot(Z,T1,'b','markersize',3)
plot(Z,a2,'r','markersize',3);
plot(Z,err2,'g','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the network,so and error of so)');
title('dissolved oxygen concentration  );
legend('Training-Data','NN output','Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Denormalize the target in original units
y2 = a2'.*i_2max;
```

```
figure(6)
hold on
plot(Z,y2,'r','markersize',3);
plot(Z,err2,'g','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the denormalized(so)');
title('dissolved oxygen concentration ');
legend('denormalized-Data(so)')
hold off;
%end
```

C.7: MATLAB script-feedforward inverse model - INVERFF.m

```
%Design an inverse feedforward NN with8 inputs, 1 Hidden layer with 8
%neurons and 1 target output neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the scaled conjugate gradient algorithm
%Use early stoppage training to prevent over fitting by Dividing the
data
%up into training, validation and test sets.

%Julius N'gon'ga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
load best_modelff%forward model parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
diary inverseff
%Normalize the inputs in the range (0,1)
Ref;
j_max=max(Ref);
j1=Ref./j_max;
j2=j1(1:960);
%i2_2is the normalized(so)
%n0,n1 are delayed values of normalized (so)
%k0,k1,k2 are delayed values of normalized flow rate(Q)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rso1_1max=max(rso1);%Normalize dissoved oxygen uptake rate
rso11=rso1./rso1_1max;%divide by maximum value
rso11=rso11';% the normalized dissoved oxygen uptake rate into column
vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Soin1=0.2*ones(1,960);
Soin11=Soin1';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pif=[j1 i2_2 n0 n1 k0 k1 Soin11 rso11]';
Tif=[i1_1]';
%Divide the data up into training, validation and test sets.
%The testing set will start with the second point and take
%every fourth point.  The validation set will start with the
%fourth point and take every fourth point.  The training set
%will take the remaining points.
[R,W] = size(Pif);
iitst = 2:4:W;
iival = 4:4:W;
iitr = 1:2:W;
validation.P = Pif(:,iival);
validation.T = Tif(:,iival);
testing.P = Pif(:,iitst)
```

```
testing.T = Tif(:,iitst)
ptr = Pif(:,iitr)
ttr = Tif(:,iitr)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause % Strike any key to continue...
netif = newff(minmax(ptr),[8 1],{'tansig','purelin'},'trainscg')
%Create a feedforward network with 8 hidden neurons, 1 output
% neurons, tansig hidden neurons and linear output neurons.
[netif,tr]=train(netif,ptr,ttr,[],[],validation,testing)
netif.trainParam.show=5;% Show intermediate results every five
iterations.
%Training begins...please wait...
%Train the network, by early stopping, so validation data are passed
%The errors are computed on a test.
%Thus the testing dataset are passed.
errif=Tif-sim(netif,Pif)
%bar(err)
gl=input('Stike any key....');
% Simulate the trained network.
aif=sim(netif,Pif);
%Network weights and biases
netif;
wif=netif.IW{1,1}
bwif=netif.b{1}
vif=netif.LW{2,1}
bvif=netif.b{2}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the training, validation and test errors.
figure(3)
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-
.b')
legend('Training','Validation','Test',3);
xlabel('Epoch')
ylabel('Squared Error')
pause % Strike any key to display the regression analysis.
figure(4)
%display plots showing regression analyses between the
%network outputs and the corresponding targets (in original units)
[mif,bif,rif] = postreg(aif,Tif)
%Plot the target,NN simulated target and the error
figure(5)
hold on
plot(Z,Tif,'-*','markersize',3)
plot(Z,aif,'--r','markersize',3);
plot(Z,Tif-aif,'-+','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the network,u(k),target and error ');
title('Airflow rate ');
legend('Training-Data','NN output','Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Denormalize the target in original units
ylif = aif'.*i_1max;
figure(6)
hold on
plot(Z,ylif,'markersize',3)
plot(Z,il,'--r*','markersize',3)
plot(Z,il-ylif,':k','markersize',3);
grid;
xlabel('time(days)');
ylabel('Output of the denormalized(u(k),target and Error');
title('Airflow rate');
legend('Training-Data','denormalized(u(k))','Error )
```

```
hold off;
%end
```

C.8: MATLAB script - Elman inverse model - INVERELM.m

```
%Design an inverse Elman NN with 8 inputs, 1 Hidden layer with 10
%neurons and 1 target output neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the scaled conjugate gradient algorithm
%Use early stoppage training to prevent over fitting by Dividing the
data
%up into training, validation and test sets.

%Julius Ngonga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
close all;
clear all;
load best_modelelm;
diary inverseelm
%Normalize the inputs and outputs in the range (0,1)
j_max=max(Ref);
j1=Ref./j_max;
j2=j1(1:960);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create delayed values of the input variables and output variables
rsol_1max=max(rsol);%Normalize dissoved oxygen uptake rate
rsol1=rsol./rsol_1max;%divide by maximum value
rsol1=rsol1';% the normalized dissoved oxygen uptake rate into column
vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Soin1=0.2*ones(1,960);
Soin11=Soin1';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create input and target variables to the elman network
Pie=[j1 i2_2 n0 n1 k0 k1 Soin11 rsol1]';%input variables to the NN
Tie=[i1_1]';%NN target
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Divide the data up into training, validation and test sets
%The testing set will start with the second point and take
%every fourth point.  The validation set will start with the
%fourth point and take every fourth point.  The training set
%will take the remaining points.
[R,W] = size(Pie);
iitst = 2:4:W;
iival = 4:4:W;
iitr = 1:2:W;
validation.P = Pie(:,iival);
validation.T = Tie(:,iival)
testing.P = Pie(:,iitst)
testing.T = Tie(:,iitst)
ptr = Pie(:,iitr)
ttr = Tie(:,iitr)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pause % Strike any key to continue..
%Training the network with scaled conjugate algorithm
netie = newelm(minmax(ptr),[10 1],{'tansig','purelin'},'trainscg')
%Create a nelman network with 10 hidden neurons, 1 output
% neurons, TANSIG hidden neurons and linear output neurons.
```

```
[netie,tr]=train(netie,ptr,ttr,[],[],validation,testing)
netie.trainParam.show=5;% Show intermediate results every five
iterations.
%Training begins...please wait...
%Train the network.  We use early stopping, so we are passing the
%validation data.  We also want the errors computed on a test
%set, so we are passing the testing data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
errie=Tie-sim(netie,Pie);
saveas(gcf,'trainscg_errie','fig');
save tr tr;
%bar(err)
g1=input('Stike any key....');
%Simulate the trained network.
aie=sim(netie,Pie);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Network weights and biases
netie;
wie=netie.IW{1,1}
bwie=netie.b{1}
vie=netie.LW{2,1}
bvie=netie.b{2}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the training, validation and test errors.
figure(3)
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-
.b')
legend('Training','Validation','Test',3);
xlabel('Epoch')
ylabel('Squared Error')
pause % Strike any key to display the regression analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(4)
%display plots showing regression analyses between the
%network outputs and the corresponding targets (in original units).
[mie,bie,rie] = postreg(aie,Tie)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the target,NN simulated target and the error
figure(5)
hold on
plot(Z,Tie,'r+--','markersize',3)
plot(Z,aie,'-','markersize',3);
plot(Z,errie,'--','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the network,u(k+1)input(so(k+1 and error of(u(k))');
title('Airflow rate u(k)');
legend('Training-Data','NN output','Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Denormalize the target in original units
ylie = aie'.*i_1max;
figure(6)
hold on
plot(Z,ylie,'r-*','markersize',3)
plot(Z,i1,'--','markersize',3)
plot(Z,i1-ylie,'-','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the denormalized(u(k))');
title('Airflow rate ');
legend('Training-Data','denormalized(u(k),and Error')
hold off;
%end
```

C.9: MATLAB script-nonlinear linearizing controller feedforward model - nonlinearconff.m

```
%Design of nonlinear linearizing controller feedforward NN with 5
%inputs, 1 hidden layer with 5 neurons and 1 target output neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the scaled conjugate gradient algorithm
%Use early stoppage training to prevent over fitting by dividing the
%data up into training, validation and test sets.

%Julius N'gon'ga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
load data_tr;
load contdata_tr;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
diary nonlinearf
%Normalize the inputs and outputs in the range (0,1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rso1_1max=max(rso1);%Normalize dissoved oxygen uptake rate
rso11=rso1./rso1_1max;%divide by maximum value
rso11=rso11';% the normalized dissoved oxygen uptake rate into column
vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u1_11max=max(u1);
u1_11=u1./u1_11max;%Normalize flow rate u1 by dividing with the max
value

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Vo_1max=max(Vo);%Normalize desired DO output by dividing with the max
value
Vo_11=Vo./Vo_1max;%the normalized desired DO output into column vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V1_1max=max(V1);%Normalize linear controller output by dividing with
the
%max value
V1_11=V1./V1_1max;%the normalized linear controller output into column
vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create delayed values of the input variables and output variables
N=u1_11(1:960);%nomalized flowrate u1
N0=[0;N(1:959)];%u1 with one delay
N1=[0;0;N(1:958)];%u1 with two delays
N2=[0;0;0;N(1:957)];%u1 with three delays

Ni=Vo_11(1:960);%nomalized linear controller v
N0i=[0;Ni(1:959)];%v with one delay
N1i=[0;0;Ni(1:958)];%v with two delays
N2i=[0;0;0;Ni(1:957)];%v with three delays

Ki=V1_11(1:960);%nomalized deired DO output So
K0i=[0;Ni(1:959)];%So with one delay
K1i=[0;0;Ni(1:958)];%So with two delays
K2i=[0;0;0;Ni(1:957)];%So with three delays
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Create input and target variables to the elman network
Pc=[Vo_11 V1_11 K0i rso11 Soin11]';%normalized inputs
Tc=[u1_11]';%normalized targets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Divide the data up into training, validation and test sets.
% The testing set will start with the second point and take
% every fourth point.  The validation set will start with the
% fourth point and take every fourth point.  The training set
% will take the remaining points.
[Rc,Qc] = size(Pc);
iitst = 2:4:Qc;
iival = 4:4:Qc;
iitr = 1:2:Qc;
validation.P = Pc(:,iival);
validation.T = Tc(:,iival)
testing.P = Pc(:,iitst)
testing.T = Tc(:,iitst)
ptrc = Pc(:,iitr)
ttrc = Tc(:,iitr)
pause % Strike any key to continue...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Training the network with scaled conjugate algorithm
netcf = newff(minmax(ptrc),[5 1],{'tansig' 'purelin'},'trainscg')
%Create a feedforward network with 5 hidden neurons, 1 output
% neurons, Tansig hidden neurons and linear output neurons.
[netcf,tr]=train(netcf,ptrc,ttrc,[],[],validation,testing);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

netcf.trainParam.show=5;% Show intermediate results every five
iterations.
%    Training begins...please wait...
% Train the network.  We use early stopping, so we are passing the
% validation data.  We also want the errors computed on a test
% set, so we are passing the testing data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

errc=Tc-sim(netcf,Pc)
%bar(err)
gc=input('Stike any key....');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
saveas(gcf, 'training_ err','fig')

% Simulate the trained network.
acf=sim(netcf,Pc);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Network weights and biases
netcf;
wc=netcf.IW{1,1}
bwc=netcf.b{1}
vc=netcf.LW{2,1}
bvc=netcf.b{2}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Plot the training, validation and test errors.
figure(1)
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-
.b')
legend('Training','Validation','Test',3);
xlabel('Epoch')
```

```
ylabel('Squared Error')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pause % Strike any key to display the regression analysis..
figure(2)
%     display plots showing regression analyses between the
%     network outputs and the corresponding targets (in original
units).
[mcf,bcf,rcf] = postreg(acf,Tc)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Plot the target,NN simulated target and the error
figure(3)
hold on
plot(Z,Tc,'b','markersize',3)
plot(Z,acf,'r','markersize',3);
plot(Z,errc,'g','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the network and error difference');
title('Output of the linearizing nonlinear controller');
legend('Training-Data','NN output','Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Denormalize the target in original units and plot
ycf = acf'.*ul_llmax;
figure(4)
hold on
plot(Z,ycf,'r','markersize',3);
plot(Z,errc,'g','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the network and error difference');
title('Output of the linearizing nonlinear controller');
legend('denormalized-Data(ul)')
hold off;
%end
```

C.10:MATLAB script-nonlinear linearizing controller Elman model nonlinearconelm.m

```
%Design of nonlinear linearizing controller Elman NN with 5 inputs, 1
%hidden layer with 7 neurons and 1 target output neuron
%Use tan-sigmoid activation function for the hidden layer and purelin
%transfer function for the output layer
%Train the network using the scaled conjugate gradient algorithm
%Use early stoppage training to prevent over fitting by dividing the
data
%up into training, validation and test sets.

%Julius N'gon'ga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
close all;
close all;
clear all;
load important2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%diary nonlinear
%Normalize the inputs and outputs in the range (0,1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
rsol_1max=max(rsol);%Normalize dissoved oxygen uptake rate
rsol1=rsol./rsol_1max;%divide by maximum value
rsol11=rsol1';% the normalized dissoved oxygen uptake rate into column
vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ul_11max=max(ul);
ul_11=ul./ul_11max;%Normalize flow rate ul by dividing with the max
value

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Vo_1max=max(Vo);%Normalize desired DO output by dividing with the max
value
Vo_11=Vo./Vo_1max;%the normalized desired DO output into column vector

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V1_1max=max(V1);%Normalize linear controller output by dividing with
the
%max value
V1_11=V1./V1_1max;%the normalized linear controller output into column
vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create delayed values of the input variables and output variables
N=ul_11(1:960);%nomalized flowrate ul
N0=[0;N(1:959)];%ul with one delay
N1=[0;0;N(1:958)];%ul with two delays
N2=[0;0;0;N(1:957)];%ul with three delays

Ni=Vo_11(1:960);%nomalized linear controller v
N0i=[0;Ni(1:959)];%v with one delay
N1i=[0;0;Ni(1:958)];%v with two delays
N2i=[0;0;0;Ni(1:957)];%v with three delays

Ki=V1(1:960);%nomalized delred DO output So
K0i=[0;Ki(1:959)];%So with one delay
K1i=[0;0;Ki(1:958)];%So with two delays
K2i=[0;0;0;Ni(1:957)];%So with three delays
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create input and target variables to the elman network
Pc=[Vo_11 V1_11 K0i rsol1 Soin11]';%normalized inputs
Tc=[ul_11]';%normalized targets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Divide the data up into training, validation and test sets.
% The testing set will start with the second point and take
% every fourth point.  The validation set will start with the
% fourth point and take every fourth point.  The training set
% will take the remaining points.
[Rc,Qc] = size(Pc);
iitst = 2:4:Qc;
iival = 4:4:Qc;
iitr = 1:2:Qc;
validation.P = Pc(:,iival);
validation.T = Tc(:,iival)
testing.P = Pc(:,iitst)
```

```
testing.T = Tc(:,iitst)
ptrc = Pc(:,iitr)
ttrc = Tc(:,iitr)
pause % Strike any key to continue...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Training the network with scaled conjugate algorithm
netc = newelm(minmax(ptrc),[7 1],{'tansig' 'purelin'},'trainscg')
%Create an elman network with 6 hidden neurons, 1 output
% neurons, TANSIG hidden neurons and linear output neurons.
[netc,tr]=train(netc,ptrc,ttrc,[],[],validation,testing);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

netc.trainParam.show=5;% Show intermediate results every five
iterations.
%    Training begins...please wait...
% Train the network.  We use early stopping, so we are passing the
% validation data.  We also want the errors computed on a test
% set, so we are passing the testing data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

errc=Tc-sim(netc,Pc)
%bar(err)
gc=input('Stike any key....');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
saveas(gcf, 'training_ err','fig')

% Simulate the trained network.
ac=sim(netc,Pc);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Network weights and biases
netc;
wc=netc.IW{1,1}
bwc=netc.b{1}
vc=netc.LW{2,1}
bvc=netc.b{2}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Plot the training, validation and test errors.
figure(1)
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,'g',tr.epoch,tr.tperf,'-
.b')
legend('Training','Validation','Test',3);
xlabel('Epoch')
ylabel('Squared Error')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pause % Strike any key to display the regression analysis..
figure(2)
%    display plots showing regression analyses between the
%    network outputs and the corresponding targets (in original
units).
[mc,bc,rc] = postreg(ac,Tc)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the target,NN simulated target and the error
figure(3)
hold on
plot(Z,Tc,'b','markersize',3)
plot(Z,ac,'r','markersize',3);
plot(Z,errc,'g','markersize',3)
```

```
grid;
xlabel('time(days)');
ylabel('Output of the network and error difference');
title('Output of the linearizing nonlinear controller');
legend('Training-Data','NN output','Error')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Denormalize the target in original units and plot
yc = ac'.*u1_11max;
figure(4)
hold on
plot(Z,yc,'r','markersize',3);
plot(Z,errc,'g','markersize',3)
grid;
xlabel('time(days)');
ylabel('Output of the network and error difference');
title('Output of the linearizing nonlinear controller');
legend('denormalized-Data(u1)')
hold off;
%end
```

C.11: MATLAB script-nonlinear function-parametersforu1.m

```
%Simulate the trajectory of the nonlinear linearizing controller output
%derived from equation 5.35 namely, the model of the linearizing
%controller.

%Julius Ngonga Muga
%Cape Peninsula University of Technology
%2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
load important1;
a=-0.02;%constant
b=200;%constant
h=1.0141667e-004;%sampling time
g1=h*(Q/V);%constant
g2=h;%constant
g=h*(Sosat-Soo)*K1;
f=Soo*(1-g1)+h*K1*(Sosat-Soo);
u1=((a*V1+b*Vo)-f-g1-(g2*rso))/g;%V1 is the desired DO trajectory,
%Vo is the virtual control input from the PI controller
%end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

APPENDIX D:

DATABASE ANDCONVERSION INTO MYSQL FORMAT

This section presents the conversion of the ASCII text files and the MS Excel files into MySQL database format. The procedure for setting up the MySQL is also database is also discussed.

APPENDIX E:

COMMUNICATION BETWEEN THE SOFTWARE PLATFORMS ADROIT, MATLAB AND MySQL

This section discusses the development of a GUI for the implementation of the communication between various platforms.