



Cape Peninsula  
University of Technology

**PREDICTION OF THE INFLUENT WASTEWATER VARIABLES USING NEURAL  
NETWORK THEORY**

by

**CARL KRIGER**

**Thesis submitted in fulfilment of the requirements for the degree**

**Master of Technology: Discipline Electrical Engineering**

**in the Faculty of Engineering**

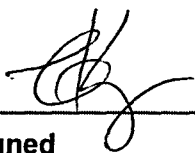
**at the Cape Peninsula University of Technology**

**Supervisor: Professor Raynitchka Tzoneva**

**Bellville**  
December 2007

## DECLARATION

I, Carl Kriger, declare that the contents of this thesis represent my own unaided work, and that the 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

MARCH 2008

Date

## ABSTRACT

In order to develop an effective control strategy for the activated sludge process of a wastewater treatment plant, an understanding of the nature of the influent load disturbances to the wastewater treatment plant is necessary. Biological systems are among the most difficult to control and predict. Due to the complex biological reaction mechanisms, the highly time-varying, and multivariable aspects of the wastewater treatment plant (WWTP), the diagnosis of the WWTP are still difficult in practice. The application of intelligent techniques, which can analyse the multi-dimensional nonlinear process data using a visualisation technique, can be useful for analysing and diagnosing the activated-sludge process in the WWTP. This complex capability for nonlinearity representation combined with the fact that no model exists for the WWTP influent dynamics to a WWTP, makes neural networks an ideal choice for a solution.

Forecasting the behaviour of complex systems has been a broad application area for neural networks. Applications such as economic forecasting, electricity load / demand forecasting, and forecasting natural and physical phenomena have been extensively studied, hence the numerous papers presented at annual conferences in this focus area. The cognitive ability of artificial neural networks to map nonlinear complex input-output relationships, which would allow for better prediction and corrective control of processes, make them particularly attractive.

The values of the influent disturbances are usually measured off-line in a laboratory, as there are still no reliable on-line sensors available. This work presents the development of a neural network model for prediction of the values of the influent disturbances based on historical plant and weather data, which ultimately affect the activated sludge process. Three different neural networks including the multilayer perceptron, recurrent and radial basis functions neural network are developed for the prediction of the influent disturbances of Chemical Oxygen Demand (COD), Total Kjeldahl Nitrogen (TKN) and influent flow rate respectively. The application area is the prediction of the influent variables at a local municipal wastewater treatment plant. The forecast result is used for the determination of the setpoint to a controller, in order to optimize plant performance. The results are first applied to a pilot wastewater treatment plant.

Much hype exists surrounding the subject of neural networks, and they are sometimes described as 'computers that think'. This sort of definition creates unrealistic expectations and is one of the reasons why it is discredited. The results obtained will hopefully present helpful insights as to the scope and possibilities as to the application for neural networks, but

also present the practical challenges which neural network practitioners and designers of intelligent systems face.

The solution of the problem for development of the mathematical model for dynamic behaviour of the influent disturbances according to the influence of the weather conditions and the season of the year is the first attempt in the scientific and research literature so far.

## ACKNOWLEDGEMENTS

**I wish to express my heartfelt thanks to:**

- My heavenly Father without whose strength and wisdom I would not have accomplished anything.
- My wife Linda, Jessè, and Anica my wonderful kids for their tremendous support and understanding during the extremely long nights and sacrifices they made. (I'll definitely make it up to you guys)
- Professor Raynitchka Tzoneva without whose supervision, guidance, assistance, commitment, patience, understanding and dedication this work would not have been possible.
- Mr S. Behardien in the Electrical Engineering Department, Cape Peninsula University of Technology for his insightful comments, guidance, and support.
- Mr T.van Breda and all staff in the Electrical Engineering Department Cape Peninsula University of Technology for their support.
- Mr J. Bruiners and Mr H. Rus of the Athlone Wastewater Plant for the wastewater plant data supplied.
- Mrs Wendy at the South African Weather Services in Cape Town for the weather data supplied.
- Mr P. Mapisa for his assistance on the commissioning and testing of the pilot plant.

**"It is a good thing to give thanks unto the Lord." Psalm 92:1**

**The financial assistance of the National Research Foundation towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the National Research Foundation.**

## TABLE OF CONTENTS

Declaration	ii
Abstract	iii
Acknowledgements	v
Glossary	xxv
Mathematical Notation	xxvi

### CHAPTER ONE: INTRODUCTION

1.1	Awareness of the problem	1
1.2	Legislation and current situation	3
1.3	Mathematical Modelling and Simulations of Wastewater Treatment Plants	4
1.3.1	Simulation of the Wastewater Treatment Process	4
1.4	Control of Wastewater Treatment Plants	5
1.4.1	Reasons why Advanced Control is not implemented at WWT Plants	6
1.4.2	Reasons why Automatic Control should be implemented in a WWTP	6
1.5	Dissolved Oxygen Control	7
1.6	Adaptive Control Strategy	8
1.7	Problem statement	10
1.8	Neural networks	12
1.8.1	Training of Neural Networks	12
1.8.2	Data preparation	13
1.8.3	Types of Neural Networks	14
1.8.4	Application areas for Neural Networks	14
1.8.5	Modelling of the Influent variables model behaviour using Neural Networks	15
1.8.6	Neural Networks implementation	16
1.9	Research Aim	16
1.10	Research Objectives	16
1.10.1	Scientific objectives	16
1.10.2	Application result objectives	17
1.11	Literature survey	18
1.12	Outline of thesis	18
1.13	List of publications	19

### CHAPTER TWO: WASTEWATER TREATMENT PROCESSES AND ITS DISTURBANCES

2.1	Introduction	21
2.2	Legislation	22
2.3	Current process control techniques	23
2.4	The Wastewater Treatment Process	24
2.4.1	Preliminary Treatment	24
2.4.2	Primary Treatment	24
2.4.3	Secondary Treatment	25
2.4.4	Biochemical Oxygen Demand (BOD)	25
2.4.5	Chemical Oxygen Demand (COD)	26
2.4.6	Total Suspended Solids (TSS)	26
2.4.7	Fecal and Chlorine Residual	26
2.4.8	The Activated Sludge Process (ASP)	26

2.4.8.1	The Aeration Tank	27
2.4.8.2	The Sedimentation Tank	27
2.4.8.3	Mixed Liquor Suspended Solids (MLSS)	27
2.4.8.4	Mixed Liquor Volatile Suspended Solids (MLVSS)	28
2.4.8.5	Food-to-microorganism ratio	28
2.4.8.6	Hydraulic Retention Time	28
2.4.8.7	Sludge Age	29
2.4.8.8	Biological Nutrient Removal	29
2.5	Disturbances to the Wastewater Treatment Process	31
2.6	Influent Wastewater Characterization	33
2.6.1	Chemical oxygen demand (COD) components in the UCT model	33
2.6.2	Nitrogen components in the UCT model	35
2.6.3	Influent Wastewater Characterization Summary	37
2.7	Summary of the chapter	37

### CHAPTER THREE: NEURAL NETWORKS BACKGROUND

3.1	Introduction	39
3.2	Artificial Neural Networks	40
3.3	A brief history of neural networks	40
3.4	Network Architecture (Topology)	43
3.5	Activation functions	44
3.6	Neural Network Training / Learning	48
3.7	The Perceptron Learning Rule	49
3.8	The Least Mean Square Algorithm	49
3.9	The Backpropagation Training Algorithm	50
3.9.1	Limitations of the Backpropagation Training Algorithm	51
3.9.2	Heuristic Improvements to the Backpropagation Algorithm	52
3.10	Generalization	53
3.10.1	Overfitting and underfitting	53
3.10.2	Early stopping	56
3.10.3	Regularization	57
3.11	Neural Network Positive Characteristics	57
3.12	Application Areas of Neural Networks	58
3.13	Neural Networks structures for use in prediction	58
3.14	Summary of the Chapter	59

### CHAPTER FOUR: DATA PRE-PROCESSING FOR SUPERVISED LEARNING

4.1	Introduction	61
4.2	Data description	62
4.3	Data selection	66
4.4	Data preprocessing	67
4.4.1	Missing data	67
4.4.2	The curse of dimensionality	69
4.4.3	Correlation	70
4.4.4	Principal Component Analysis	73
4.4.5	Normalization (scaling)	73
4.4.6	Trends	74
4.4.7	Seasonality	74
4.4.8	Circular discontinuity	75
4.4.9	Outliers and data rejection	77
4.5	Data post-processing	81
4.6	Summary of the chapter	81

## CHAPTER FIVE: APPLICATION OF NEURAL NETWORKS FOR INFLUENT DISTURBANCES PREDICTION

5.1	Introduction	83
5.2	The Multilayer Feed-forward Model	84
5.2.1	Feed-forward Multilayer Neural Network Implementation Criteria	86
5.2.1.1	Input selection	86
5.2.1.2	Length and selection of training and validation sets of data	87
5.2.1.3	Activation Function	88
5.2.1.4	Error criterion	89
5.2.1.5	Number of hidden layers and hidden neurons	90
5.2.1.6	Number of epochs	92
5.2.1.7	Weight and bias initialisation	92
5.2.1.8	Training Method	93
5.2.1.9	Learning rate	93
5.2.1.10	Gradient method	93
5.2.1.11	Training Stopping criterion	96
5.2.1.12	Performance measurement	97
5.3	The Elman Recurrent Neural Network Model	98
5.3.1	Elman Recurrent Neural Network Implementation Criteria	99
5.3.1.1	Input selection	99
5.3.1.2	Length and selection of training and validation sets	100
5.3.1.3	Activation Function	100
5.3.1.4	Error criterion	100
5.3.1.5	Number of hidden layers and hidden neurons	100
5.3.1.6	Number of epochs	100
5.3.1.7	Weight and bias initialisation	100
5.3.1.8	Training Method	100
5.3.1.9	Learning rate	100
5.3.1.10	Gradient method	101
5.3.1.11	Training Stopping criterion	101
5.3.1.12	Performance measurement	101
5.4.	The Radial Basis Function Neural Network Model	101
5.4.1.	Radial Basis Function Neural Network Implementation Criteria	103
5.4.1.1	Input selection	103
5.4.1.2	Length and selection of training and validation sets	103
5.4.1.3	Activation Function	103
5.4.1.4	Error criterion	103
5.4.1.5	Number of hidden layers	103
5.4.1.6	Number of epochs and hidden neurons	104
5.4.1.7	Weight and bias initialisation	104
5.4.1.8	Error goal	104
5.4.1.9	Spread constant	104
5.4.1.10	Training Stopping criterion	104
5.4.1.11	Performance measurement	104
5.5	Comparison of the advantages and disadvantages of the MLP and RBFNN	104
5.6	Software implementation	105
5.7	Summary of the chapter	105



## CHAPTER SIX: RESULTS

6.1	Introduction	106
6.2	COD Neural Network Results	106
6.2.1	COD Neural Network MODEL 1	106
6.2.2	COD Neural Network MODEL 2	117
6.2.3	COD Neural Network MODEL 3	120
6.2.4	COD Neural Network MODEL 4	123
6.2.5	COD Neural Network MODEL 5	126
6.2.6	COD Neural Network MODEL 6	129
6.2.7	COD Neural Network MODEL 7	132
6.2.8	COD Neural Network MODEL 8	135
6.2.9	COD Neural Network MODEL 9	138
6.2.10	COD Neural Network MODEL 10	141
6.2.11	COD Neural Network MODEL 11	144
6.2.12	COD Neural Network MODEL 12	147
6.2.13	COD Neural Network MODEL 13	150
6.3	TKN Neural Network Results	153
6.3.1	TKN Neural Network MODEL 1	153
6.3.2	TKN Neural Network MODEL 2	162
6.3.3	TKN Neural Network MODEL 3	165
6.3.4	TKN Neural Network MODEL 4	168
6.3.5	TKN Neural Network MODEL 5	171
6.3.6	TKN Neural Network MODEL 6	174
6.3.7	TKN Neural Network MODEL 7	177
6.3.8	TKN Neural Network MODEL 8	180
6.3.9	TKN Neural Network MODEL 9	183
6.2.10	TKN Neural Network MODEL 10	186
6.3.11	TKN Neural Network MODEL 11	189
6.3.12	TKN Neural Network MODEL 12	192
6.3.13	TKN Neural Network MODEL 13	195
6.4	FLOW Neural Network Results	198
6.4.1	FLOW Neural Network MODEL 1	198
6.4.2	FLOW Neural Network MODEL 2	207
6.4.3	FLOW Neural Network MODEL 3	210
6.4.4	FLOW Neural Network MODEL 4	213
6.4.5	FLOW Neural Network MODEL 5	216
6.4.6	FLOW Neural Network MODEL 6	219
6.4.7	FLOW Neural Network MODEL 7	222
6.4.8	FLOW Neural Network MODEL 8	225
6.4.9	FLOW Neural Network MODEL 9	228
6.4.10	FLOW Neural Network MODEL 10	231
6.4.11	FLOW Neural Network MODEL 11	234
6.4.12	FLOW Neural Network MODEL 12	237
6.4.13	FLOW Neural Network MODEL 13	240
6.5	Investigation of the variation of the structure of the NNs and the training process	243
6.5.1	The effect of increasing the number of hidden neurons	243
6.5.2	The effect of increasing the number of epochs	244
6.5.3	The effect of increasing the number of hidden layers	246
6.5.4	The effect of training to convergence	248
6.5.5	The training time when using different training algorithms	249
6.6	Discussion of results	251
6.6.1	COD NN results	251
6.6.2	TKN NN results	252
6.6.3	FLOW NN results	253
6.6.4	Variation of the NN structures results	254

6.7	Summary of the chapter	255
-----	------------------------	-----

## **CHAPTER SEVEN: REAL-TIME IMPLEMENTATION**

7.1	Introduction	256
7.2	Pilot plant and control system	256
7.3	Raw plant and weather data conversion	256
7.4	SCADA system	258
7.5	Programmable Logic Controller (PLC) system	260
7.6	MySQL Database	261
7.7	Neural Network Software algorithms	263
7.8	Real-time implementation algorithm	268
7.8.1	NN retuning and calculation of the new predicted values if COD, TKN, Flow rate	268
7.8.2	Calculation of the inflow COD and TKN components	268
7.9	Summary of the chapter	269

## **CHAPTER EIGHT: CONCLUSION AND FUTURE DIRECTION OF RESEARCH**

8.1	Conclusion	270
8.2	Thesis results and application of the results	272
8.3	Application of the obtained results	273
8.4	Future research	274
8.4.1	Publications related to the thesis	274

<b>REFERENCES</b>	<b>321</b>
-------------------	------------

## LIST OF FIGURES

<b>Figure 1.1: A Block Diagram of the Biological Wastewater Treatment Process</b>	<b>3</b>
<b>Figure 1.2: Three-layer control strategy for adaptive optimal control of the ASP</b>	<b>9</b>
<b>Figure 1. 1:Feed-forward-feedback control system for DO control</b>	<b>10</b>
<b>Figure 1. 2: Figure illustrating that the dependence between the weather and season (month) is a type of dynamic “black box”</b>	<b>11</b>
<b>Figure 1. 3: Connection between the process model and the influent disturbances model</b>	<b>11</b>
<b>Figure 2. 1: A The Activated Sludge Process with Extended Aeration</b>	<b>27</b>
<b>Figure 2. 2: The Activated Sludge Process with Pre-Denitrification</b>	<b>31</b>
<b>Figure 2. 3: The technological structure for the Athlone WWTP based on the UCT model</b>	<b>31</b>
<b>Figure 2. 4: Dynamics of the influent disturbances for the Athlone WWT Plant</b>	<b>32</b>
<b>Figure 2. 5: COD components in the influent for the UCT model</b>	<b>34</b>
<b>Figure 2. 6: Nitrogen components in the influent for the UCT model</b>	<b>36</b>
<b>Figure 3. 1: Structure of the brain and the biological neuron</b>	<b>39</b>
<b>Figure 3. 2: Structure of an artificial neuron</b>	<b>41</b>
<b>Figure 3. 3: A classification problem illustrating linear separable classes</b>	<b>42</b>
<b>Figure 3. 4: Feed-forward Neural Network</b>	<b>44</b>
<b>Figure 3. 5: Recurrent Neural Network</b>	<b>44</b>
<b>Figure 3. 6: Binary (Threshold) Activation function</b>	<b>45</b>
<b>Figure 3. 7: Bipolar Binary (Threshold) Activation function</b>	<b>45</b>
<b>Figure 3. 8: Logistic Sigmoid Activation function</b>	<b>46</b>
<b>Figure 3. 9: Hyperbolic Tangent Activation function</b>	<b>46</b>
<b>Figure 3. 10: Pure linear Activation function</b>	<b>47</b>
<b>Figure 3. 11: Radial Basis Activation function</b>	<b>47</b>
<b>Figure 3. 12: The Learning algorithm could get stuck in a local minima</b>	<b>52</b>
<b>Figure 3. 13: An example of overfitting of a noisy sine wave (Demuth, et al, 2004)</b>	<b>54</b>
<b>Figure 3. 14: Training error and generalization error</b>	<b>55</b>
<b>Figure 3. 15: Early Stopping Method plotting the training, validation and test sets</b>	<b>56</b>
<b>Figure 3. 16: Neural network structure for prediction applications where the inputs are time-delayed a number of steps</b>	<b>59</b>
<b>Figure 4. 1: Block diagram illustration of the use of data pre-processing and post-processing with a neural network (Adapted from Bishop (1995))</b>	<b>62</b>
<b>Figure 4. 2: An aerial view of the Athlone Wastewater Treatment Plant</b>	<b>63</b>
<b>Figure 4. 3: Chemical Oxygen Demand (COD) time series data: 1992- 2005</b>	<b>64</b>
<b>Figure 4. 4: Total Kjeldahl Nitrogen (TKN) time series data: 1992- 2005</b>	<b>64</b>
<b>Figure 4. 5: Flow rate time series data: 1992- 2005</b>	<b>65</b>
<b>Figure 4. 6: Maximum Temperature time series data: 1992- 2005</b>	<b>65</b>
<b>Figure 4. 7: Minimum Temperature time series data: 1992- 2005</b>	<b>65</b>
<b>Figure 4. 8: Rainfall time series data: 1992- 2005</b>	<b>66</b>
<b>Figure 4. 9: Wind Speed time series data: 1992- 2005</b>	<b>66</b>
<b>Figure 4. 10: Circularity discontinuity: Diagram illustrating the approach used in presenting data of a circular nature to the neural network</b>	<b>76</b>
<b>Figure 4. 11: Histogram plots are useful for data rejection rather than automated outlier rejection algorithms. The above figure shows the histogram plot of the COD variable</b>	<b>78</b>
<b>Figure 4. 12: Histogram plot of the TKN</b>	<b>78</b>
<b>Figure 4. 13: Histogram plot of the Flow rate</b>	<b>78</b>
<b>Figure 4. 14: Histogram plot of the Minimum temperature</b>	<b>79</b>
<b>Figure 4. 15: Histogram plot of the Maximum temperature</b>	<b>79</b>
<b>Figure 4.15: Histogram plot of the Rainfall</b>	<b>79</b>
<b>Figure 4. 16: Histogram plot of the Wind speed</b>	<b>80</b>

<b>Figure 4. 17: A summary of the pre-processing techniques implemented on the plant and weather data</b>	<b>81</b>
<b>Figure 5. 1: The Multilayer Perceptron Neural Network for prediction of an input disturbance</b>	<b>85</b>
<b>Figure 5. 2: The model of a neuron for the Multilayer Feed-forward Neural Network</b>	<b>85</b>
<b>Figure 5. 3: Neural Network model describing the inflow characteristics on the basis of environmental factors</b>	<b>87</b>
<b>Figure 5. 4: Calculating the number of hidden layer neurons using a pyramid approach (Masters, 1993)</b>	<b>91</b>
<b>Figure 5. 5: The Engine benchmark problem illustrating the convergence speed for the different training algorithms</b>	<b>95</b>
<b>Figure 5. 6: The Elman Recurrent Neural Network for prediction of an input disturbance</b>	<b>99</b>
<b>Figure 5. 7: The model of a neuron for the Elman Recurrent Neural Network</b>	<b>99</b>
<b>Figure 5. 8: The Radial Basis Neural Network for prediction of an input disturbance</b>	<b>102</b>
<b>Figure 5. 9: The model of a neuron for the Radial Basis Neural Network</b>	<b>102</b>
<b>Figure 6. 1: (a) MLP Feed-forward NN Model 1 with 1 input, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 1 with 1 input, 4 hidden neurons</b>	<b>107</b>
<b>Figure 6. 2: Training error versus the number of epochs for COD: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs</b>	<b>110</b>
<b>Figure 6. 3: A scatter-plot of the training input data and the NN response with the application of the training set for the COD: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs</b>	<b>111</b>
<b>Figure 6. 4: A scatter-plot of the test (validation) data and the NN response with the application of the test set for the COD: (a) FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs</b>	<b>112</b>
<b>Figure 6. 5: Network performance showing the linear regression (correlation) coefficients for training data (normalized) on the left, and the validation data set (denormalized) on the right for the COD: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs, where A and T are the NN output and desired target output respectively</b>	<b>113</b>
<b>Figure 6. 6: The instantaneous error across the entire validation data set for the COD: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs</b>	<b>114</b>
<b>Figure 6. 7: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs</b>	<b>116</b>
<b>Figure 6. 8: (a) A MLP Feed-forward neural network Model 2 with 2 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; (b) Block Diagram for the COD RBFNN Model 2 with 34 hidden neurons</b>	<b>117</b>
<b>Figure 6. 9: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 2 with 1 hidden neuron and 500 epochs; (b) ERNN Model 2 with 10 hidden neurons, 500 epochs; and (c) RBFNN Model 2 with 34 hidden neurons, 34 epochs</b>	<b>118</b>
<b>Figure 6. 10: (a) A MLP Feed-forward neural network Model 3 with 3 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 3 with 2 hidden neurons, 2 epochs</b>	<b>120</b>

<b>Figure 6. 11:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 3 with 1 hidden neuron and 500 epochs; (b) ERNN Model 3 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 3 with 2 hidden neurons, 2 epochs	121
<b>Figure 6. 12:</b> (a) A MLP Feed-forward neural network Model 4 with 4 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 4 with 2 hidden neurons, 2 epochs	123
<b>Figure 6. 13:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 4 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 4 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 4 with 2 hidden neurons, 2 epochs	124
<b>Figure 6. 14:</b> (a) A MLP Feed-forward neural network Model 5 with 5 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 5 with 3 hidden neurons, 3 epochs	126
<b>Figure 6. 15:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 5 with 5 hidden neurons and 500 epochs; (b) ERNN Model 5 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 5 with 3 hidden neurons, 3 epochs	127
<b>Figure 6. 16:</b> (a) A MLP Feed-forward neural network Model 6 with 6 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 6 with 4 hidden neurons, 4 epochs	129
<b>Figure 6. 17:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 6 with 1 hidden neuron and 500 epochs; (b) ERNN Model 6 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 6 with 4 hidden neurons, 4 epochs	130
<b>Figure 6. 18:</b> (a) A MLP Feed-forward neural network Model 7 with 8 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 7 with 19 hidden neurons, 19 epochs	132
<b>Figure 6. 19:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 7 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 7 with 5 hidden neurons, 1000 epochs; and (c) RBFNN Model 7 with 19 hidden neurons, 19 epochs	133
<b>Figure 6. 20:</b> (a) A MLP Feed-forward neural network Model 8 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 8 with 25 hidden neurons, 25 epochs	135
<b>Figure 6. 21:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 8 with 1 hidden neuron and 500 epochs; (b) ERNN Model 8 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 8 with 25 hidden neurons, 25 epochs	136
<b>Figure 6. 22:</b> (a) A MLP Feed-forward neural network Model 9 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 9 with 18 hidden neurons, 18 epochs	138
<b>Figure 6. 23:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 9 with 1 hidden neuron and 520 epochs; (b) ERNN Model 9 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 9 with 18 hidden neurons, 18 epochs	139
<b>Figure 6. 24:</b> (a) A MLP Feed-forward neural network Model 10 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 10 with 20 hidden neurons, 20 epochs	141
<b>Figure 6. 25:</b> The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 10 with 5 hidden neuron and 1000 epochs; (b) ERNN Model 10 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 10 with 20 hidden neurons, 20 epochs	142
<b>Figure 6. 26:</b> (a) A MLP Feed-forward neural network Model 11 with 10 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 11 with 19 hidden neurons, 19 epochs	144
<b>Figure 6. 27:</b> The NN response to the application of the test (validation) data	145

set for COD: (a) MLP FFNN Model 11 with 10 hidden neurons and 500 epochs; (b) ERNN Model 11 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 11 with 19 hidden neurons, 19 epochs	
Figure 6. 28: (a) A MLP Feed-forward neural network Model 12 with 11 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 12 with 25 hidden neurons, 25 epochs	147
Figure 6. 29: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 12 with 1 hidden neuron and 500 epochs; (b) ERNN Model 12 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 12 with 25 hidden neurons, 25 epochs	148
Figure 6. 30: (a) A MLP Feed-forward neural network Model 13 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 13 with 4 hidden neurons, 4 epochs	150
Figure 6. 31: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 13 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 13 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 13 with 4 hidden neurons, 4 epochs	151
Figure 6. 32: (a) MLP Feed-forward NN Model 1 with 1 input, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 1 with 1 input, 6 hidden neurons	153
Figure 6. 33: Training error versus the number of epochs for TKN: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 5 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs	155
Figure 6. 34: A scatter-plot of the training input data and the NN response with the application of the training set for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs	156
Figure 6. 35: A scatter-plot of the test (validation) data and the NN response with the application of the test set for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs	157
Figure 6. 36: Network performance showing the linear regression (correlation) coefficients for training data (normalized) on the left, and the validation data set (denormalized) on the right for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs, where A and T are the NN output and desired target output respectively	158
Figure 6. 37: The instantaneous error across the entire validation data set for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs	159
Figure 6. 38: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs	161
Figure 6. 39: (a) A MLP Feed-forward neural network Model 2 with 2 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; (b) Block Diagram for the TKN RBFNN Model 2 with 2 hidden neurons	162
Figure 6. 40: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 2 with 1 hidden neuron and 500 epochs; (b) ERNN Model 2 with 10 hidden neurons, 500 epochs; and (c) RBFNN Model 2 with 2 hidden neurons, 2 epochs	163
Figure 6. 41: (a) A MLP Feed-forward neural network Model 3 with 3 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 3 with 2 hidden neurons, 2 epochs	165

<b>Figure 6. 42:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 3 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 3 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 3 with 2 hidden neurons, 2 epochs	166
<b>Figure 6. 43:</b> (a) A MLP Feed-forward neural network Model 4 with 4 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 4 with 2 hidden neurons, 2 epochs	168
<b>Figure 6. 44:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 4 with 10 hidden neurons and 500 epochs; (b) ERNN Model 4 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 4 with 2 hidden neurons, 2 epochs	169
<b>Figure 6. 45:</b> (a) A MLP Feed-forward neural network Model 5 with 5 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 5 with 5 hidden neurons, 5 epochs	171
<b>Figure 6. 46:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 5 with 1 hidden neuron and 500 epochs; (b) ERNN Model 5 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 5 with 5 hidden neurons, 5 epochs	172
<b>Figure 6. 47:</b> (a) A MLP Feed-forward neural network Model 6 with 6 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 6 with 9 hidden neurons, 9 epochs	174
<b>Figure 6. 48:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 6 with 1 hidden neuron and 500 epochs; (b) ERNN Model 6 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 6 with 9 hidden neurons, 9 epochs	175
<b>Figure 6. 49:</b> (a) A MLP Feed-forward neural network Model 7 with 8 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 7 with 17 hidden neurons, 17 epochs	177
<b>Figure 6. 50:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 7 with 1 hidden neuron and 500 epochs; (b) ERNN Model 7 with 5 hidden neurons, 1000 epochs; and (c) RBFNN Model 7 with 17 hidden neurons, 17 epochs	178
<b>Figure 6. 51:</b> (a) A MLP Feed-forward neural network Model 8 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 8 with 17 hidden neurons, 17 epochs	180
<b>Figure 6. 52:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 8 with 1 hidden neuron and 500 epochs; (b) ERNN Model 8 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 8 with 17 hidden neurons, 17 epochs	181
<b>Figure 6. 53:</b> (a) A MLP Feed-forward neural network Model 9 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 9 with 19 hidden neurons, 19 epochs	183
<b>Figure 6. 54:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 9 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 9 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 9 with 19 hidden neurons, 19 epochs	184
<b>Figure 6. 55:</b> (a) A MLP Feed-forward neural network Model 10 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 10 with 16 hidden neurons, 16 epochs	186
<b>Figure 6. 56:</b> The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 10 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 10 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 10 with 16 hidden neurons, 16 epochs	187
<b>Figure 6. 57:</b> (a) A MLP Feed-forward neural network Model 11 with 10 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 11 with 24 hidden neurons, 24 epochs	189
<b>Figure 6. 58:</b> The NN response to the application of the test (validation) data	190

set for TKN: (a) MLP FFNN Model 11 with 1 hidden neuron and 1000 epochs;	
193(b) ERNN Model 11 with 5 hidden neurons, 500 epochs; and (c) RBFNN	
Model 11 with 24 hidden neurons, 24 epochs	
Figure 6. 59: (a) A MLP Feed-forward neural network Model 12 with 11	192
inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block	
Diagram for the TKN RBFNN Model 12 with 18 hidden neurons, 18 epochs	
Figure 6. 60: The NN response to the application of the test (validation) data	193
set for TKN: (a) MLP FFNN Model 12 with 5 hidden neurons and 500 epochs;	
(b) ERNN Model 12 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model	
12 with 18 hidden neurons, 18 epochs	
Figure 6. 61: (a) A MLP Feed-forward neural network Model 13 with 9 inputs,	195
1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram	
for the TKN RBFNN Model 13 with 14 hidden neurons, 14 epochs	
Figure 6. 62: The NN response to the application of the test (validation) data	196
set for TKN: (a) MLP FFNN Model 13 with 1 hidden neuron and 1000 epochs;	
(b) ERNN Model 13 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model	
13 with 14 hidden neurons, 14 epochs	
Figure 6. 63: (a) MLP Feed-forward NN Model 1 with 1 input, 1 hidden layer, 1	198
hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW	
RBFNN Model 1 with 1 input, 6 hidden neurons	
Figure 6. 64: Training error versus the number of epochs for FLOW: (a) MLP	200
FFNN Model 1 with 1 hidden neuron, 309 epochs; (b) ERNN Model 1 with 5	
hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7	
epochs	
Figure 6. 65: A scatter-plot of the training input data and the NN response	201
with the application of the training set for the FLOW: (a) MLP FFNN Model 1	
with 1 hidden neuron, 309 epochs; (b) ERNN Model 1 with 5 hidden neurons,	
500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs	
Figure 6. 66: A scatter-plot of the test (validation) data and the NN response	202
with the application of the test set for the FLOW: (a) MLP FFNN Model 1 with	
1 hidden neuron, 309 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500	
epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs	
Figure 6. 67: Network performance showing the linear regression	203
(correlation) coefficients for training data (normalized) on the left, and the	
validation data set (denormalized) on the right for the FLOW: (a) MLP FFNN	
Model 1 with 1 hidden neuron and 309 epochs; (b) ERNN Model 1 with 5	
hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons,	
7 epochs, where A and T are the NN output and desired target output	
respectively	
Figure 6. 68: The instantaneous error across the entire validation data set for	204
the FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b)	
ERNN Model 1 with 5 hidden neurons, 309 epochs; and (c) RBFNN Model 1	
with 7 hidden neurons, 7 epochs	
Figure 6. 69: The NN response to the application of the test (validation) data	206
set for FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron and 309 epochs;	
(b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model	
1 with 7 hidden neurons, 7 epochs	
Figure 6. 70: (a) A MLP Feed-forward neural network Model 2 with 2 inputs, 1	207
hidden layer, 1 hidden neuron and 1 output neuron; (b) Block Diagram for the	
FLOW RBFNN Model 2 with 3 hidden neurons	
Figure 6. 71: The NN response to the application of the test (validation) data	208
set for FLOW: (a) MLP FFNN Model 2 with 5 hidden neurons and 1000	
epochs; (b) ERNN Model 2 with 10 hidden neurons, 1000 epochs; and (c)	
RBFNN Model 2 with 3 hidden neurons, 3 epochs	
Figure 6. 72: (a) A MLP Feed-forward neural network Model 3 with 3 inputs, 1	210
hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram	
for the FLOW RBFNN Model 3 with 17 hidden neurons, 17 epochs	



<b>Figure 6. 73:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 3 with 10 hidden neurons and 1000 epochs; (b) ERNN Model 3 with 10 hidden neurons, 1000 epochs; and (c) RBFNN Model 3 with 17 hidden neurons, 17 epochs	211
<b>Figure 6. 74:</b> (a) A MLP Feed-forward neural network Model 4 with 4 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 4 with 35 hidden neurons, 35 epochs	213
<b>Figure 6. 75:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 4 with 5 hidden neurons and 500 epochs; (b) ERNN Model 4 with 5 hidden neurons, 1000 epochs; and (c) RBFNN Model 4 with 35 hidden neurons, 35 epochs	214
<b>Figure 6. 76:</b> (a) A MLP Feed-forward neural network Model 5 with 5 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 5 with 22 hidden neurons, 22 epochs	216
<b>Figure 6. 77:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 5 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 5 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 5 with 22 hidden neurons, 22 epochs	217
<b>Figure 6. 78:</b> (a) A MLP Feed-forward neural network Model 6 with 6 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 6 with 16 hidden neurons, 16 epochs	219
<b>Figure 6. 79:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 6 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 6 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 6 with 16 hidden neurons, 16 epochs	220
<b>Figure 6. 80:</b> (a) A MLP Feed-forward neural network Model 7 with 8 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 7 with 35 hidden neurons, 35 epochs	222
<b>Figure 6. 81:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 7 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 7 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 7 with 35 hidden neurons, 35 epochs	223
<b>Figure 6. 82:</b> (a) A MLP Feed-forward neural network Model 8 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 8 with 35 hidden neurons, 35 epochs	225
<b>Figure 6. 83:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 8 with 5 hidden neurons and 500 epochs; (b) ERNN Model 8 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 8 with 35 hidden neurons, 35 epochs	226
<b>Figure 6. 84:</b> (a) A MLP Feed-forward neural network Model 9 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 9 with 27 hidden neurons, 27 epochs	228
<b>Figure 6. 85:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 9 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 9 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 9 with 27 hidden neurons, 27 epochs	229
<b>Figure 6. 86:</b> (a) A MLP Feed-forward neural network Model 10 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 10 with 16 hidden neurons, 16 epochs	231
<b>Figure 6. 87:</b> The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 10 with 1 hidden neuron and 500 epochs; (b) ERNN Model 10 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 10 with 27 hidden neurons, 27 epochs	232
<b>Figure 6. 88:</b> (a) A MLP Feed-forward neural network Model 11 with 10 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 11 with 26 hidden neurons, 26 epochs	234
<b>Figure 6. 89:</b> The NN response to the application of the test (validation) data	235

set for FLOW: (a) MLP FFNN Model 11 with 1 hidden neuron and 500 epochs; (b) ERNN Model 11 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 11 with 26 hidden neurons, 26 epochs	
Figure 6. 90: (a) A MLP Feed-forward neural network Model 12 with 11 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 12 with 31 hidden neurons, 31 epochs	237
Figure 6. 91: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 12 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 12 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 12 with 31 hidden neurons, 31 epochs	238
Figure 6. 92: (a) A MLP Feed-forward neural network Model 13 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 13 with 24 hidden neurons, 24 epochs	240
Figure 6. 93: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 13 with 1 hidden neuron and 500 epochs; (b) ERNN Model 13 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 13 with 24 hidden neurons, 24 epochs	241
Figure 6. 94: The NN response to the application of the test (validation) data set for TKN ERNN Model 12 with 40 hidden neurons, 1000 epochs, where the number of neurons is increased: (a) Training phase; (b) Regression coefficients for training (left) and test phases (right); and (c) NN response to the test set	244
Figure 6. 95: The NN response to the application of the test (validation) data set for FLOW MLP FFNN Model 6 with 10 hidden neurons, 5000 epochs, where the epochs is increased: (a) Training; (b) Regression coefficients for training (left) and test phases (right); and (c) NN response to the test set	245
Figure 6. 96: The NN response to the application of the test (validation) data set for COD MLP FFNN Model 12 with 5_5 hidden neurons, 1000 epochs, where there are 2 hidden layers: (a) Training; (b) Regression coefficients for training (left) and test phases (right); and (c) NN response to the test set	247
Figure 6. 97: The NN response to the application of the test (validation) data set for COD RBFNN Model 7 with 414 hidden neurons, 414 epochs, where the network is trained to convergence: (a) Training; (b) Performance for training and test sets; (c) NN response	249
Figure 7. 1: The WWT Pilot plant together with the Modicon PLC and transmitters in the Electrical Engineering Department, CPUT	257
Figure 7.2: The Dissolved Oxygen transmitter	257
Figure 7. 3: The structure for the real-time implementation of the influent disturbances prediction showing the pilot plant, PLC, SCADA system, MySQL database with plant and weather data, and MATLAB NN algorithms	257
Figure 7. 4: The SCADA system graphical user interface allows the user to view real-time plant variables, trend information, enter data in manual mode, run the neural network software for training and validation, and view and enter data in the database	259
Figure 7. 5: The PLC program to move sensory data from the Conductivity and DO sensors, from one analogue input register to another holding register	260
Figure 7. 6: The MySQL database is shown with the tables with the data for the Athlone WWTP influent variables and the weather data	263
Figure 7. 7: The MATLAB GUI for the Neural Network program for prediction of the influent disturbances	264
Figure 7. 8: The plots illustrate the training, validation, and NN response to training validation data, and network performance	265
Figure 7. 9: The software flow diagram	267
Figure D. 1: The Modicon PLC I/O Base wiring diagram	307
Figure D. 1: The Dissolved Oxygen sensor wiring diagram	308
Figure D. 2: The Dissolved Oxygen transmitter wiring connector	309

Figure D. 3: The pH sensor wiring diagram	309
Figure D. 5: The Conductivity sensor wiring diagram	310
Figure E. 1: Load the text file into MS Excel	312
Figure E. 2: Text Import Wizard step 1	313
Figure E. 3: Text Import Wizard step 2	313
Figure E. 4: Weather file in MS Excel format	314
Figure E. 5: Creating a new database	315
Figure E. 6: Selecting the menu item for importing the Excel data	315
Figure E. 7: Selecting the MS Excel option from the Data Wizard window	316
Figure E. 8: Selecting the MS Excel file that is to be imported	316
Figure E. 9: Selecting the database where the MS Excel file is to be imported	317
Figure E. 10: The Windows Control Panel	318
Figure E. 11: The Administrative Tools window	318
Figure E. 12: The ODBC Data Source Administrator Window	319
Figure E. 13: The Create New Data Source Window	319
Figure E. 14: The Connector/ODBCAdd Data New Data Source Window	320
Figure E. 15: The ODBC Data Source Administrator Window showing the new MySQL driver	320

#### LIST OF TABLES (these are numbered according to the chapter in which they appear)

Table 4. 1: The information for the process variables for the Athlone wastewater treatment plant and weather data	64
Table 4. 2: The correlation coefficients for the Athlone wastewater treatment plant data and weather data at the NN input	71
Table 4. 3: The correlation coefficients for the correlation between the inputs and outputs for the three NNs for the different influent variables of COD, TKN and influent Flow rate	72
Table 4. 4: The monthly data represented as a 4-bit binary number instead of a decimal value	75
Table 4. 5: The monthly data represented as a sine / cosine pair	77
Table 4. 6: Summary of implemented pre-processing techniques	80
Table 5. 1: Input variable combinations to the NN with COD as NN output	88
Table 5. 2: Input variable combinations to the NN with TKN as NN output	89
Table 5. 3: Input variable combinations to the NN with influent FLOW rate as NN output	90
Table 5. 4: Training algorithms for use with the MATLAB Neural Network Toolbox	94
Table 6. 1: Input variables for COD Model 1	106
Table 6. 2: Results for the MLP Feed-forward Neural Network Model 1 with COD as NN output	109
Table 6. 3: Results for the Elman Recurrent Neural Network Model 1 with COD as NN output	109
Table 6.4: Results for the Radial Basis Function Neural Network Model 1 with COD as NN output	109
Table 6. 5: Input variables for COD Model 2	117
Table 6. 6: Results for the Feed-forward Neural Network Model 2 with COD as NN output	119
Table 6. 7: Results for the Elman Recurrent Neural Network Model 2 with COD as NN output	119
Table 6. 8: Results for the Radial Basis Function Neural Network Model 2 with COD as NN output	119
Table 6. 9: Input variables for COD Model 3	120
Table 6. 10: Results for the Feed-forward Neural Network Model 3 with COD as NN output	122

<b>Table 6. 11: Results for the Elman Recurrent Neural Network Model 3 with COD as NN output</b>	<b>122</b>
<b>Table 6. 12: Results for the Radial Basis Function Neural Network Model 3 with COD as NN output</b>	<b>122</b>
<b>Table 6. 13: Input variables for COD Model 4</b>	<b>123</b>
<b>Table 6. 14: Results for the Feed-forward Neural Network Model 4 with COD as output</b>	<b>125</b>
<b>Table 6. 15: Results for the Elman Recurrent Neural Network Model 4 with COD as output</b>	<b>125</b>
<b>Table 6. 16: Results for the Radial Basis Function Neural Network Model 4 with COD as output</b>	<b>125</b>
<b>Table 6. 17: Input variables for COD Model 5</b>	<b>126</b>
<b>Table 6. 18: Results for the Feed-forward Neural Network Model 5 with COD as output</b>	<b>128</b>
<b>Table 6. 19: Results for the Elman Recurrent Neural Network Model 5 with COD as output</b>	<b>128</b>
<b>Table 6. 20: Results for the Radial Basis Function Neural Network Model 5 with COD as output</b>	<b>128</b>
<b>Table 6. 21: Input variables for COD Model 6</b>	<b>129</b>
<b>Table 6. 22: Results for the Feed-forward Neural Network Model 6 with COD as output</b>	<b>131</b>
<b>Table 6. 23: Results for the Elman Recurrent Neural Network Model 6 with COD as output</b>	<b>131</b>
<b>Table 6. 24: Results for the Radial Basis Function Neural Network Model 6 with COD as output</b>	<b>131</b>
<b>Table 6. 25: Input variables for COD Model 7</b>	<b>132</b>
<b>Table 6. 26: Results for the Feed-forward Neural Network Model 7 with COD as output</b>	<b>134</b>
<b>Table 6. 27: Results for the Radial Basis Function Neural Network Model 7 with COD as output</b>	<b>134</b>
<b>Table 6. 28: Results for the Elman Recurrent Neural Network Model 7 with COD as output</b>	<b>134</b>
<b>Table 6. 29: Input variables for COD Model 8</b>	<b>135</b>
<b>Table 6. 30: Results for the Feed-forward Neural Network Model 8 with COD as output</b>	<b>137</b>
<b>Table 6. 31: Results for the Elman Recurrent Neural Network Model 8 with COD as output</b>	<b>137</b>
<b>Table 6. 32: : Results for the Radial Basis Function Neural Network Model 8 with COD as output</b>	<b>137</b>
<b>Table 6. 33: Input variables for COD Model 9</b>	<b>138</b>
<b>Table 6. 34: Results for the Feed-forward Neural Network Model 9 with COD as output</b>	<b>140</b>
<b>Table 6. 35: Results for the Elman Recurrent Neural Network Model 9 with COD as output</b>	<b>140</b>
<b>Table 6. 36: Results for the Radial Basis Function Neural Network Model 9 with COD as output</b>	<b>140</b>
<b>Table 6. 37: Input variables for COD Model 10</b>	<b>141</b>
<b>Table 6. 38: Results for the Feed-forward Neural Network Model 10 with COD as output</b>	<b>143</b>
<b>Table 6. 39: Results for the Elman Recurrent Neural Network Model 10 with COD as output</b>	<b>143</b>
<b>Table 6. 40: Results for the Radial Basis Function Neural Network Model 10 with COD as output</b>	<b>143</b>
<b>Table 6. 41: Input variables for COD Model 11</b>	<b>144</b>
<b>Table 6. 42: Results for the Feed-forward Neural Network Model 11 with COD as output</b>	<b>146</b>
<b>Table 6. 43: Results for the Elman Recurrent Neural Network Model 11 with</b>	<b>146</b>

COD as output	
Table 6. 44: Results for the Radial Basis Function Neural Network Model 11 with COD as output	146
Table 6. 45: Input variables for COD Model 12	147
Table 6. 46: Results for the Feed-forward Neural Network Model 12 with COD as output	149
Table 6. 47: Results for the Elman Recurrent Neural Network Model 12 with COD as output	149
Table 6. 48: Results for the Radial Basis Function Neural Network Model 12 with COD as output	149
Table 6. 49: Input variables for COD Model 13	150
Table 6. 50: Results for the Feed-forward Neural Network Model 13 with COD as output	152
Table 6. 51: Results for the Elman Recurrent Neural Network Model 13 with COD as output	152
Table 6. 52: Results for the Radial Basis Function Neural Network Model 13 with COD as output	152
Table 6.53: Input variables for TKN Model 1	153
Table 6.54: Results for the Feed-forward Neural Network Model 1 with TKN as output	154
Table 6.55: Results for the Elman Recurrent Neural Network Model 1 with TKN as output	154
Table 6. 56: Results for the Radial Basis Function Neural Network Model 1 with TKN as output	154
Table 6. 57: Input variables for TKN Model 2	162
Table 6. 58: Results for the Feed-forward Neural Network Model 2 with TKN as output	164
Table 6. 59: Results for the Elman Recurrent Neural Network Model 2 with TKN as output	164
Table 6. 60: Results for the Radial Basis Function Neural Network Model 2 with TKN as output	164
Table 6. 61: Input variables for TKN Model 3	165
Table 6. 62: Results for the Feed-forward Neural Network Model 3 with TKN as output	167
Table 6. 63: Results for Elman Recurrent Neural Network Model 3 with TKN as output	167
Table 6. 64: Results for the Radial Basis Function Neural Network Model 3 with TKN as output	167
Table 6. 65: Input variables for TKN Model 4	168
Table 6. 66: Results for the Feed-forward Neural Network Model 4 with TKN as output	170
Table 6. 67: Results for the Elman Recurrent Neural Network Model 4 with TKN as output	170
Table 6. 68: Results for the Radial Basis Function Neural Network Model 4 with TKN as output	170
Table 6. 69: Input variables for TKN Model 5	171
Table 6. 70: Results for the Feed-forward Neural Network Model 5 with TKN as output	173
Table 6. 71: Results for the Elman Recurrent Neural Network Model 5 with TKN as output	173
Table 6. 72: Results for the Radial Basis Function Neural Network Model 5 with TKN as output	173
Table 6. 73: Input variables for TKN Model 6	174
Table 6. 74: Results for the Feed-forward Neural Network Model 6 with TKN as output	176
Table 6. 75: Results for the Elman Recurrent Neural Network Model 6 with TKN as output	176

<b>Table 6. 76: Results for the Radial Basis Function Neural Network Model 6 with TKN as output</b>	<b>176</b>
<b>Table 6. 77: Input variables for TKN Model 7</b>	<b>177</b>
<b>Table 6. 78: Results for the Feed-forward Neural Network Model 7 with TKN as output</b>	<b>179</b>
<b>Table 6. 79: Results for the Elman Recurrent Neural Network Model 7 with TKN as output</b>	<b>179</b>
<b>Table 6. 80: Results for the Radial Basis Function Neural Network Model 7 with TKN as output 8</b>	<b>179</b>
<b>Table 6. 81: Input variables for TKN Model 8</b>	<b>180</b>
<b>Table 6. 82: Results for the Feed-forward Neural Network Model 8 with TKN as output</b>	<b>182</b>
<b>Table 6. 83: Results for the Elman Recurrent Neural Network Model 8 with TKN as output</b>	<b>182</b>
<b>Table 6. 84: Results for the Radial Basis Function Neural Network Model 8 with TKN as output</b>	<b>182</b>
<b>Table 6. 85: Input variables for TKN Model 9</b>	<b>183</b>
<b>Table 6. 86: Results for the Feed-forward Neural Network Model 9 with TKN as output</b>	<b>185</b>
<b>Table 6. 87: Results for the Elman Recurrent Neural Network Model 9 with TKN as output</b>	<b>185</b>
<b>Table 6. 88: Results for the Radial Basis Function Neural Network Model 9 with TKN as output</b>	<b>185</b>
<b>Table 6. 89: Input variables for TKN Model 10</b>	<b>186</b>
<b>Table 6. 90: Results for the Feed-forward Neural Network Model 10 with TKN as output</b>	<b>188</b>
<b>Table 6. 91: Results for the Elman Recurrent Neural Network Model 10 with TKN as output</b>	<b>188</b>
<b>Table 6. 92: Results for the Radial Basis Function Neural Network Model 10 with TKN as output</b>	<b>188</b>
<b>Table 6. 93: Input variables for TKN Model 11</b>	<b>189</b>
<b>Table 6. 94: Results for the Feed-forward Neural Network Model 11 with TKN as output</b>	<b>191</b>
<b>Table 6. 95: Results for the Elman Recurrent Neural Network Model 11 with TKN as output</b>	<b>191</b>
<b>Table 6. 96: Results for the Radial Basis Function Neural Network Model 11 with TKN as output</b>	<b>191</b>
<b>Table 6. 97: Input variables for TKN Model 12</b>	<b>192</b>
<b>Table 6. 98: Results for the Feed-forward Neural Network Model 12 with TKN as output</b>	<b>194</b>
<b>Table 6. 99: Results for the Elman Recurrent Neural Network Model 12 with TKN as output</b>	<b>194</b>
<b>Table 6. 100: Results for the Radial Basis Function Neural Network Model 12 with TKN as output</b>	<b>194</b>
<b>Table 6. 101: Input variables for TKN Model 13</b>	<b>195</b>
<b>Table 6. 102: Results for the Feed-forward Neural Network Model 13 with TKN as output</b>	<b>197</b>
<b>Table 6. 103: Results for the Elman Recurrent Neural Network Model 13 with TKN as output</b>	<b>197</b>
<b>Table 6.104: Results for the Radial Basis Function Neural Network Model 13 with TKN as output</b>	<b>197</b>
<b>Table 6.105: Input variables for FLOW Model 1</b>	<b>198</b>
<b>Table 6.106: Results for the Feed-forward Neural Network Model 1 with FLOW as output</b>	<b>199</b>
<b>Table 6.107: Results for the Elman Recurrent Neural Network Model 1 with FLOW as output</b>	<b>199</b>
<b>Table 6. 108: Results for the Radial Basis Function Neural Network Model 1</b>	<b>199</b>

with FLOW as output	
Table 6. 109: Input variables for FLOW Model 2	207
Table 6. 110: Results for the Feed-forward Neural Network Model 2 with FLOW as output	209
Table 6. 111: Results for the Elman Recurrent Neural Network Model 2 with FLOW as output	209
Table 6. 112: Results for the Radial Basis Function Neural Network Model 2 with FLOW as output	209
Table 6. 113: Input variables for FLOW Model 3	210
Table 6. 114: Results for the Feed-forward Neural Network Model 3 with FLOW as output	212
Table 6. 115: Results for the Elman Recurrent Neural Network Model 3 with FLOW as output	212
Table 6. 116: Results for the Radial Basis Function Neural Network Model 3 with FLOW as output	212
Table 6. 117: Input variables for FLOW Model 4	213
Table 6. 118: Results for the Feed-forward Neural Network Model 4 with FLOW as output	215
Table 6. 119: Results for the Elman Recurrent Neural Network Model 4 with FLOW as output	215
Table 6. 120: Results for the Radial Basis Function Neural Network Model 4 with FLOW as output	215
Table 6. 121: Input variables for FLOW Model 5	216
Table 6. 122: Results for the Feed-forward Neural Network Model 5 with FLOW as output	218
Table 6. 123: Results for the Elman Recurrent Neural Network Model 5 with FLOW as output	218
Table 6. 124: Results for the Radial Basis Function Neural Network Model 5 with FLOW as output	218
Table 6. 125: Input variables for FLOW Model 6	219
Table 6. 126: Results for the Feed-forward Neural Network Model 6 with FLOW as output	221
Table 6. 127: Results for the Elman Recurrent Neural Network Model 6 with FLOW as output	221
Table 6. 128: Results for the Radial Basis Function Neural Network Model 6 with FLOW as output	221
Table 6. 129: Input variables for FLOW Model 7	222
Table 6. 130: Results for the Feed-forward Neural Network Model 7 with FLOW as output	224
Table 6. 131: Results for the Elman Recurrent Neural Network Model 7 with FLOW as output	224
Table 6. 132: Results for the Radial Basis Function Neural Network Model 7 with FLOW as output	224
Table 6. 133: Input variables for FLOW Model 8	225
Table 6. 134: Results for the Feed-forward Neural Network Model 8 with FLOW as output	227
Table 6. 135: Results for the Elman Recurrent Neural Network Model 8 with FLOW as output	227
Table 6. 136: Results for the Radial Basis Function Neural Network Model 8 with FLOW as output	227
Table 6. 137: Input variables for FLOW Model 9	228
Table 6. 138: Results for the Feed-forward Neural Network Model 9 with FLOW as output	230
Table 6. 139: Results for the Elman Recurrent Neural Network Model 9 with FLOW as output	230
Table 6. 140: Results for the Radial Basis Function Neural Network Model 9 with FLOW as output	230

<b>Table 6. 141: Input variables for FLOW Model 10</b>	231
<b>Table 6. 142: Results for the Feed-forward Neural Network Model 10 with FLOW as output</b>	233
<b>Table 6. 143: Results for the Elman Recurrent Neural Network Model 10 with FLOW as output</b>	233
<b>Table 6. 144: Results for the Radial Basis Function Neural Network Model 10 with FLOW as output</b>	233
<b>Table 6. 145: Input variables for FLOW Model 11</b>	234
<b>Table 6. 146: Results for the Feed-forward Neural Network Model 11 with FLOW as output</b>	236
<b>Table 6. 147: Results for the Elman Recurrent Neural Network Model 11 with FLOW as output</b>	236
<b>Table 6. 148: Results for the Radial Basis Function Neural Network Model 11 with FLOW as output</b>	236
<b>Table 6. 149: Input variables for FLOW Model 12</b>	237
<b>Table 6. 150: Results for the Feed-forward Neural Network Model 12 with FLOW as output</b>	239
<b>Table 6. 151: Results for the Elman Recurrent Neural Network Model 12 with FLOW as output</b>	239
<b>Table 6. 152: Results for the Radial Basis Function Neural Network Model 12 with FLOW as output</b>	239
<b>Table 6. 153: Input variables for FLOW Model 13</b>	240
<b>Table 6. 154: Results for the Feed-forward Neural Network Model 13 with FLOW as output</b>	242
<b>Table 6. 155: Results for the Elman Recurrent Neural Network Model 13 with FLOW as output</b>	242
<b>Table 6. 156: Results for the Radial Basis Function Neural Network Model 13 with FLOW as output</b>	242
<b>Table 6. 157: Results for the ERNN Neural Network Model 12 with TKN as output, and where number of hidden neurons are increased to 40 compared to 1,5 and 10</b>	250
<b>Table 6. 158: Results for the MLP Neural Network Model 6 with FLOW as output, and where epochs are increased to 5000 compared to 500 and 1000</b>	250
<b>Table 6. 159: Results for the MLP Neural Network Model 12 with COD as output, and where the hidden layers are increased to two compared to one hidden layer</b>	250
<b>Table 6. 160: Results for the Radial Basis Function Neural Network Model 7 with COD as output, and where the network is trained until the error converges to a minimum compared with the previous considered case</b>	250
<b>Table 7. 1: MATLAB script files and the functional description in the development of the NN program for the prediction of the influent disturbances of COD, TKN and influent Flow rate</b>	266
<b>Table B. 1: The initial inputs weights and biases for COD Model 1</b>	280
<b>Table B. 2: The initial inputs weights and biases for COD Model 2</b>	281
<b>Table B. 3: The initial inputs weights and biases for COD Model 3</b>	282
<b>Table B. 4: The initial inputs weights and biases for COD Model 4</b>	283
<b>Table B. 5: The initial inputs weights and biases for COD Model 5</b>	284
<b>Table B. 6: The initial inputs weights and biases for COD Model 6</b>	285



## APPENDIX/APPENDICES

Appendix A: Types of Neural Networks	275
Appendix B: Selection of results of initial and final weights and biases	278
Appendix C: Software Routines	286
Appendix D: Technical wiring of the PLC and sensors	306
Appendix E: Database and conversion into MySQL format	311

## GLOSSARY

Terms/Acronyms/Abbreviations	Definition/Explanation
ANN	Artificial Neural Network
API	Application programming interface
ART	Adaptive Resonance Theory
ASCII	American Standard Code for Information Interchange
ASP	Activated sludge process
BAM	Bi-directional Associative Memory
BNR	Biological nutrient removal
BOD	Biochemical oxygen demand
BP	Backpropagation
BPNN	Backpropagation neural network
BSB	Brain State in a Box
CMAC	Cerebellar Model Articulation Controller
COD	Chemical oxygen demand
DB	Database
DCL	Differential Competitive Learning
DO	Dissolved Oxygen
DSN	Data Source Name
ERNN	Elman Recurrent Neural Network
FFNN	Feed-forward neural network
FIR	Finite Impulse Response
GNN	General regression neural network
GTM	Generative topographic mapping
LMS	Least Mean Square
LVQ	Learning vector quantization
MATLAB	Matrix Laboratory
MLP	Multi-layer perceptron
MLSS	Mixed liquor suspended solids
MLVSS	Mixed liquor volatile suspended solids
MS	Microsoft
NN	Neural network
ODBC	Open database connectivity
OLS	Ordinary Least Squares
PC	Personal computer
PCA	Principal component analysis
PLC	Programmable logic controller
PNN	Probabilistic neural network
RBF	Radial basis function
RBFNN	Radial basis function neural network

RNN  
RTRNN  
SCADA  
SOM  
SQL  
TDNN  
TKN  
WWT  
WWTP

Recurrent Neural Network  
Real-time Recurrent Neural Network  
Supervisory Control and Data Acquisition  
Self-organising map  
Structured Query Language  
Time delay neural network  
Total Kjeldahl nitrogen  
Wastewater treatment  
Wastewater treatment plant

## MATHEMATICAL NOTATION

$S_o$	concentration of the DO
$S_{o,in}$	input concentration of the DO
$S_{osat}$	concentration of the saturation of the wastewater with oxygen
$Q_{in}$	input flow rate for the aerobic tank
$Q_{out}$	Output flow rate for the aerobic tank
$K_{La}$	function for transfer of oxygen to wastewater
$V$	aeration tank volume
$\Gamma_{So}$	oxygen uptake rate
$\hat{y}(k)$	output of the neuron
$y_i$	input variables
$w_i$	synapse weights
$b_i$	bias term
$\phi$	nonlinear activation function
$u(k) \in R^r$	model inputs
$y(k) \in R^m$	model outputs
$f: R^r \rightarrow R^m$	nonlinear function of inputs and outputs
$F: M$	food-to-microorganism ratio
$BOD$	5 day bio-chemical oxygen demand (mg/l)
$MLVSS$	mixed liquor volatile suspended solids (mg/l)
$V_a$	volume of aeration tank (litres).
$HRT$	Hydraulic retention time
$D$	dilution rate
$SS_e$	suspended solids in wastewater effluent (mg/l)
$SS_w$	suspended solids in wasted sludge (mg/l)

$Q_w$	flow rate of wasted sludge ( $m^3/day$ )
$NH_4^+$	ammonium
$NO_2^-$	nitrite
$NO_3^-$	nitrate
$S_{ii}$	total COD
$S_{bi}$	biodegradable COD
$S_{ui}$	non- biodegradable COD (inert mass)
$S_{usi}$	soluble unbiodegradable COD
$S_{upi}$	unbiodegradable particulate COD
$S_{bsi}$	readily biodegradable COD
$S_{bpi}$	slowly biodegradable COD
$f_{us}$	fraction of the total influent COD which is unbiodegradable soluble
$f_{s,up}$	fraction of the total influent COD which is unbiodegradable particulate
$f_{bs}$	fraction of the biodegradable influent COD which is readily biodegradable
$S_{bi}$	biodegradable influent COD
$S_{zbi}$	COD of active aerobic organisms present in the influent
$N$	nitrogen
$N_{ii}$	TKN
$N_{ai}$	saline ammonia
$N_{oi}$	organically bound nitrogen
$N_{oui}$	unbiodegradable organically bound nitrogen
$N_{obi}$	biodegradable organically bound nitrogen
$N_{ousi}$	unbiodegradable soluble organically bound nitrogen
$N_{oupi}$	unbiodegradable particulate organically bound nitrogen

$N_{obsi}$	soluble influent biodegradable organic nitrogen
$N_{obpi}$	particulate influent biodegradable organic nitrogen
$f_{n,a}$	fraction of the total influent TKN which is free and saline ammonia
$f_{n,ous}$	fraction of the total influent TKN which is organic unbiodegradable soluble
$f_{n,obp}$	fraction of the organic biodegradable nitrogen which is particulate
$N_{zbi}$	influent biological (active) mass nitrogen fraction
$y_d$	desired model output
$y_m$	actual model output
$W$	weight vector
$\eta$	learning rate
$e(n)$	error signal
$\alpha$	momentum term
$\gamma$	performance ratio
$Cov(X,Y)$	covariance matrix of X and Y
$\sigma$	standard deviation
$N_{l,i}$	output of the neuron
$\hat{y}_l(k+1)$	output of the NN
$a_{l,i} = [a_{l,i,1} \dots a_{l,i,p}] \in R^p$	vector of the NN coefficients introducing the delayed values
$b_{l,i} = [b_{l,i,1} \dots b_{l,i,m_2}] \in R^{m_2}$	vector of the NN coefficients introducing all considered weather conditions
$c_{l,i} = [c_{l,i,1} \dots c_{l,i,m_1}] \in R^{m_1}$	vector of the NN coefficients introducing the inflow disturbances
$b_{l,i,h}$	bias coefficient
$p$	number of delays
$m_1 = 3$	number of disturbances

$y_1$	COD
$y_2$	TKN
$y_3$	Flowrate
$m_2 = 5$	number of weather conditions
$w_1$	month (sin)
$w_2$	month (cos)
$w_3$	min temp
$w_4$	rain
$w_5$	wind speed
$\varphi$	output layer activation function
$\phi_{i,j}$	hidden layer activation function
$mse$	mean squared error
$y(k)$	desired output
$\hat{y}(k)$	NN output
$r$	correlation coefficient
$r^2$	coefficient of determination
$APE$	absolute percentage error
$g_{i,j} \in R^{n_1}$	vector of the NN weight which introduces the delayed output of the first hidden layers
$\theta \in R^{n_1}$	vector of unit delays
$\phi(x; t_i)$	radial basis function
$\ x - t_i\ $	norm of the distance vector between the current vector and the centre point
$t_i$	center point of the cluster
$\sigma_i$	variance parameter

$r_j$

Euclidian distance to the center of the cluster

# CHAPTER ONE

## INTRODUCTION

### 1.1 Awareness of the problem

South Africa's conventional water resources are rapidly being depleted. During these times of water restrictions, the country is also faced with deteriorating water quality. This could be accredited amongst others, to the enormous discharge of treated effluents into rivers and streams. The effluent when discharged should not pollute the stream, be a danger to public health, or cause a local nuisance. Wastewater contains nutrients that stimulate plant growth and it may contain toxic compounds. Effective removal of pollutants such as nitrogen, phosphorous and carbon, from wastewater is of the utmost importance. The removal of total dissolved solids is also important.

If untreated wastewater is allowed to accumulate, the decomposition of the organic material can lead to the production of large quantities of malodorous gases (Sánchez-Marrè *et al.*, 1995). Wastewater treatment (WWT) plants are important to the economic and social development of any country. They protect the public's health from disease-causing bacteria and viruses. For these reasons, the immediate and nuisance-free removal of wastewater from its sources of generation, followed by treatment and disposal, is not only desirable but is also necessary in an industrialized society (Metcalf and Eddy, 1991).

The original aim of sewage disposal is the removal of the waterborne waste from domestic and industrial communities, and the removal of as much of the solids content as is practical and economical, and then to oxidise (and subsequently remove) the colloidal and dissolved solids without causing any danger to public health. The method of treatment chosen for any particular installation depends on the quality of effluent required.

Wastewater treatment plants are developed in order to remove unwanted nutrients that are deposited in the water. The nutrients that are prevalent in wastewater include Carbon, Nitrogen and Phosphorous. When these nutrients are discharged into water it leads to what is known as eutrophication. Large blooms of algae are a visible indication of eutrophication, which cause difficulties when it comes to water purification. Nutrients can be removed from wastewater effluents either by chemical or biological means.



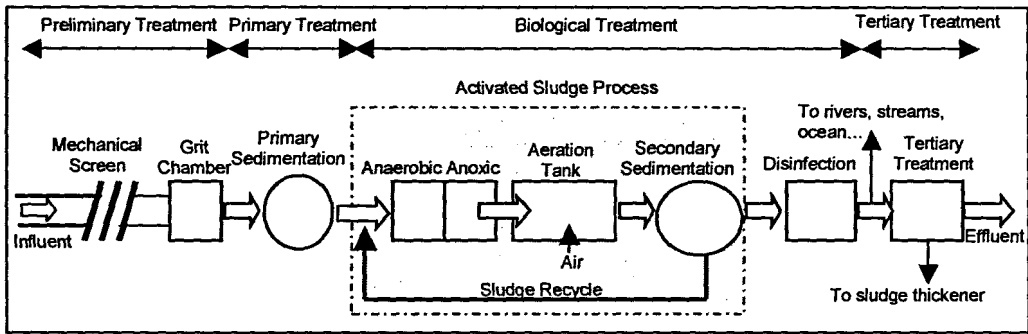
Wastewater treatment plants (WWTPs) can be divided into two types, biological and physical/chemical. Biological plants are commonly used to treat either domestic or combined domestic and industrial wastewater from a municipality. Wastewater contains many microorganisms and different bacteria. WWTPs use basically the same processes that occur naturally in the receiving water, where microorganisms in the wastewater break down the solids, but provide controlled conditions favourable for the growth of the bacteria, so that the biological reactions are completed before the water is discharged back into the environment. Physical/chemical plants are often used to treat industrial wastewaters directly, because they often contain pollutants that cannot be removed efficiently by microorganisms. Wastewater treatment plants that actually disinfect wastewater eliminate many harmful organisms.

Biological treatment processes are where microorganisms convert non-settleable solids to settleable solids. Sedimentation typically follows, where the settleable solids are allowed to settle out. There are usually three possible options including:

1. Activated Sludge - The most common option uses microorganisms in the treatment process to break down organic material with aeration and agitation, and then allows solids to settle out. Activated sludge containing bacteria is continually recycled back to the aeration basin to increase the rate of organic decomposition.
2. Trickling Filters - These are beds of coarse media (often stones or plastic) Wastewater is sprayed into the air (aeration), and is allowed to trickle through the media. Microorganisms attached to and growing on the media, break down organic material in the wastewater. Trickling filters drain at the bottom; the wastewater is collected and then undergoes sedimentation.
3. 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).

Activated sludge is defined as sludge particles produced by the growth of microorganisms in aerated tanks as a part of the process to treat wastewater. The activated sludge process is the biological process where these organisms oxidize and mineralise organic waste matter (Nadri, Hammouri and Fick, 2002). A typical biological wastewater treatment plant process is depicted in Figure 1.1. The biological WWT process includes preliminary, primary, biological and tertiary (secondary) treatment. These treatment processes as applied to the activated sludge process are discussed at length in Chapter 2 of this thesis.

Preliminary treatment includes the mechanical screens for removal of all large objects present in the wastewater influent. The grit chamber allows sand and small organic material to sink to the bottom. Primary treatment provides a zone where the sludge is allowed to settle and grease and other floating material is skimmed off the top.



**Figure 1. 1: A Block Diagram of the Biological Wastewater Treatment Process**

The activated sludge process is a widely used system of biological WWT. The wastewater enters a zone that is aerated where the sludge particles are brought into contact with the organic matter in the wastewater. The organic matter is converted in cell tissue and oxidised end products consisting mainly of carbon dioxide, CO<sub>2</sub>. The contents of this aerated zone are referred to as mixed liquor. The biological mass known as either mixed liquor suspended solids (MLSS) or mixed liquor volatile suspended solids (MLVSS), consists mostly of microorganisms, inert matter and non-biodegradable suspended matter. The secondary sedimentation (settling) tanks follow the aeration zone, and the suspended solids (SS) are separated from the treated wastewater. The biological solids are recycled back to the aeration zone in order to maintain a population of microorganisms to treat the wastewater. The excess biological solids are withdrawn from the secondary settling tanks. Some WWTPs disinfect the treated wastewater effluent before discharging it into streams or rivers or the ocean.

The activated sludge process is considered in the thesis.

## 1.2 Legislation and current situation

In South Africa legislation regarding pollution control and indirect reuse of treated wastewater is regulated by the Water Act (54 of 1956). The removal of carbonaceous material and the transformation of ammonia to nitrate are stipulated in the General and Special Standards Act of 1962 (Ekama, Marais *et al.*, 1984). The current situation in the Cape Town region is that there are 22 municipal wastewater treatment plants in operation. However it should be noted that not all of them are fully equipped to treat wastewater using the activated sludge process. Only 14 plants are staffed with personnel equipped to effectively operate the wastewater treatment plant and associated equipment. The reality of the situation is that many of the plants are not operating efficiently and produce effluent standards that are below acceptable levels.

Thus the need for monitoring and effective control of these wastewater treatment plants may remedy this situation. The effluent standards are tabled in Chapter 2 of this work.

### **1.3 Mathematical Modelling and Simulations of Wastewater Treatment Plants**

Many wastewater treatment processes are analysed and researched in order to understand their complex dynamic behaviour. This is the reason that mathematical models are developed describing the reactions between the organisms and the organic material present in the wastewater. Mathematical models are an important tool in wastewater treatment (Vargas, Moreno and Zeitz, 2001) and are used for:

- Design of the process. Modelling is useful in evaluating the impact of changes in parameters.
- Process control. Efficient control strategies are often model based.
- Forecasting. Predictions can be made of future plant performance through modelling.
- Education. Simulation models can be used for education and training.
- Optimisation. Plant performance can be enhanced.
- Research. Development and testing of hypothesis.

There are a number of commonly used models for the activated sludge process like the IAWQ (International Association on Water Quality) activated sludge model number 1 (ASM1) (Henze *et al.* 1987), number 2 (ASM2) (Henze *et al.* 1995), and the University of Cape Town (UCT) model, to name but a few. These models describe the removal of organic matter, nitrification and denitrification. Institutions throughout the world have developed various mathematical models describing the different wastewater processes. Alex, Ogurek and Jumar (2005) discuss the problems with the development and standardisation of models. The availability of reliable models is a fundamental pre-requisite for simulation of the wastewater treatment processes and systems. The work by Alex, Ogurek and Jumar, (2005) address the formulation of model representation methods for model development, exchange, and analysis.

#### **1.3.1 Simulation of the Wastewater Treatment Process**

Simulations are an additional tool in the study and understanding of the behaviour of these complex wastewater treatment processes. A simulation is an imitation of the behaviour of the real plant or system. If the mathematical model has been fairly well developed and calibrated, it is possible to predict the future behaviour of the plant using simulations (Yoo, Lee and Lee, 2002). It should be noted that any conclusions drawn from simulation results should be approached with caution.

Simulators are often equipped with a graphical user interface (GUI). The GUI gives the operator a graphical display of the simulated data together with the data presentation. A well-designed GUI can increase understanding of the process. There are many commercial and non-commercial simulation platforms available. Some of these include Matlab / Simulink and SIMBA.

#### 1.4 Control of Wastewater Treatment Plants

The control of the biological process is important to maintain high levels of treatment performance under a wide range of operating conditions. Successful process control consists of reviewing present and historical data and laboratory test results to select the proper operational parameters that provide the best performance at the least cost (Sánchez-Marrè *et al.*, 1995).

According to surveys conducted in the United States of America, more than 50% of all newly constructed plants are discharging effluents that do not comply with the definition of secondary treatment. The leading problem causing poor plant performance is found to be the lack of treatment concepts and laboratory testing to process control (Arthur, 1982).

The wastewater treatment process is a fairly complex, dynamic process. According to Theilliol *et al.*, (2003), certain factors need to be considered when attempting to implement advanced real-time control of the process:

- The process is time varying (Carlsson and Lindberg, 1996). Parameters such as temperature, composition of the influent water, amount of biomass, flows, pH, etc, vary over a period of time.
- The process is non-linear and dynamic.
- The process has stiff dynamics. The time constants involved in the process, range from seconds to months.
- The process is multivariable. There are several inputs and outputs.
- Sensors available in the plant are not very reliable although much progress has been made in the development of sensors. Problems encountered are often that sensors are very noisy, they have long response times, they require frequent maintenance and they drift.
- Large disturbances affect the process. The influent flow and concentration are subject to large variations particularly after heavy rains.

The chemical industry has demonstrated that automation improves the cost-effectiveness of chemical processes while enhancing product quality. It is reasonable therefore to conclude that automation should also improve the performance of wastewater treatment systems with less cost and less energy consumption (Arthur, 1982).

### **1.4.1 Reasons why Advanced Control is not implemented at WWT Plants**

The major impediment to automating the wastewater treatment process is that many of the processes have long response times, and it appears as if manual control is effective. Most operators and engineers at wastewater treatment plants have limited experience or training when it comes to plant automation that no efforts are made to automate. In those rare cases when a control automation system is present, most operators switch from automatic to manual mode (Arthur, 1982).

Listed below are some of the reasons why it is necessary to implement control of the wastewater treatment process and also why this is not always the case:

- Legislation regarding effluent quality has in many cases not been strictly monitored and enforced.
- There is a lack of suitable understanding of engineering of many of the control processes.
- On-line sensors require a lot of maintenance and are not very reliable.
- Plant workers do not understand how to use and maintain plant equipment properly.
- Pumps, valves and actuators are manually switched on or off by an operator in many plants.
- Processes at wastewater plants are fairly complex and it is not very easy to implement automatic control of the plant or optimise plant performance.
- Traditionally plant managers and staff do not appreciate the benefits of automatically controlling the plant and thus improving overall plant efficiency.
- Operator resistance to learning new technology and resistance to change.
- The use of daily average type data, not on-line data to control the process.
- The activated sludge process has not been studied completely as an object of control. It is not clear what models are convenient for real-time control implementation, what process variables are the most significant and in what way the quality of the process and its product is affected, and what the optimal combination of settings are for these significant variables.
- There is a lack of methods for measuring of important variables, for modelling and parameter estimation and for optimisation and control calculation.

### **1.4.2 Reasons why Automatic Control should be implemented in a WWTP**

- Stricter laws governing the effluent quality.
- General public awareness of environmental issues and focusing on sustainability and energy consumption issues.
- Taxes and costs associated with the effluent quality are increasing.
- The design reliability of on-line sensors and actuators are improving.
- New processes at plants are becoming more and more difficult to control manually.
- Various models for the control of wastewater plants are developed which are suitable for real-time control of existing plants.

An important step in the design of an automatic control strategy is to determine which signals to control, which signals to measure or estimate, and which control signals to use (Garvey, 1997). Some of the variables affecting carbon and nitrogen removal, which could be controlled, are:

- Dissolved Oxygen (DO) concentration;

- the aeration volume;
- ammonium concentration by manipulating the DO setpoint;
- internal recirculation control;
- the influent and excess sludge flow rates;
- the recycle sludge flow rate; and
- if necessary the chemical feed rates.

The type of control that the attention is focused on for this work is the control of the DO concentration.

## 1.5 Dissolved Oxygen Control

The Cape Peninsula University of Technology (CPUT) has initiated a collaborative partnership with the local municipal wastewater treatment plant authorities. The Athlone WWTP is a municipal plant in the Cape Town region in South Africa. At the Athlone wastewater treatment plant one of the parameters being monitored is DO (dissolved oxygen) in the aerated zone.

Automatic control techniques are successfully applied to control the DO concentration for many years, with tremendous cost and energy savings, and improved plant performance (Arthur, 1982). According to Carlsson and Lindberg (1996) this could be due to the fact that:

- On-line DO sensors have existed for several years.
- Energy conservation has to be considered in the control the DO, since it is very expensive to blow air into the large tanks.
- The effluent quality and process efficiency depends on a suitable dissolved oxygen level.

An excessively high dissolved oxygen concentration results in increased energy consumption, and deterioration in process performance as there is a reduction in sludge quality and the denitrification process is less efficient. On the other hand low DO concentrations lead to bad quality sludge and less efficient pollution removal (Gutierrez and Vega, 2002).

The DO concentration should be high enough to support the growth of adequate organisms, and low enough to conserve energy. Excessive mixing should be avoided as the low DO levels contribute to nitrate recirculation in the advanced nutrient process.

Different controllers are developed in the literature based on the DO mass balance model (Olssen and Newell, 1999).

$$S_o(t) = \frac{1}{V} [Q_{in} S_{o,in}(t) - Q_{out} S_o(t)] + K_{La} [S_{o,sat}(t) - S_o(t)] - \Gamma_{So}(t) \quad 1.1$$

where  $S_o$  is the concentration of the DO,  $S_{osat}$  is the concentration of the saturation of the wastewater with oxygen,  $Q_{in}$  and  $Q_{out}$  are the input and output flow rates for the aerobic tank,  $S_{o,in}$  is the input concentration of the DO,  $K_{La}$  is the function for transfer of oxygen to wastewater  $V$  is the aeration tank volume and  $\Gamma_{so}$  is the oxygen uptake rate. The oxygen uptake rate is an indicator for the dynamic behaviour of the microorganisms and the influent load of COD and TKN, because it is function of the microorganisms' growth rate and the concentrations of some of the components of influent COD and TKN.  $K_{La}$  is a function of the air flow rate and is used to introduce the control input into the model.

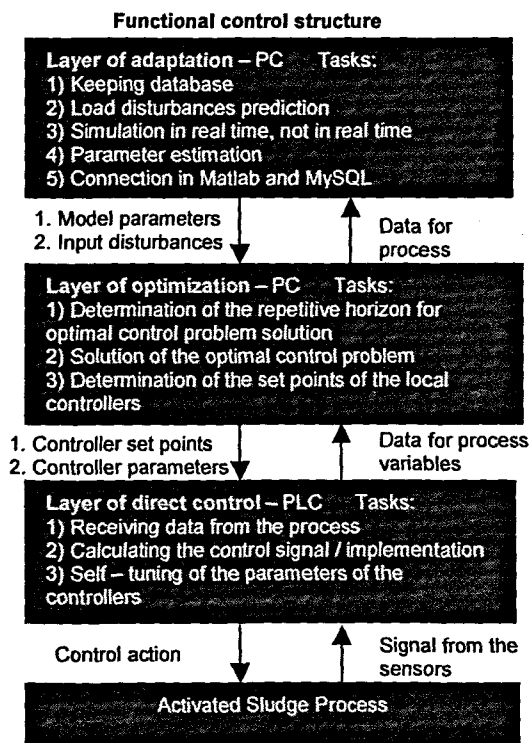
From the components of the model 1.1, it can be seen that the controller designed according to this model will depend on the values of the influent disturbances. But these disturbances cannot be measured on-line. Then the only possibility to apply modern control techniques to the DO control is to incorporate in the control strategy a software sensor for these variables. The prediction of the influent disturbances in real-time will make the control strategy adaptive and close to the real operating conditions of the WWTP.

## 1.6 Adaptive Control Strategy

Research in the area of wastewater treatment systems has focused largely on the dynamic behaviour of the activated sludge process of the wastewater treatment plants. The uncertainty in sludge activity and composition are caused by unpredictable change in the load disturbances represented by the input flow rate and waste concentrations. The latter consists mainly of carbonaceous and nitrogenous compositions that are described by the chemical oxygen demand (COD) and the total Kjeldahl nitrogen (TKN) (Olssen and Newell, 1999). The values of these load disturbances are dependant on the seasonal variations in temperature and rainfall. The control of the ASP depends strongly on the variations of the load disturbances because they determine different operating conditions for this process. This means that the control strategy has to be capable of generating control action according to the change in the operating conditions. This type of control can be achieved if the values of the disturbances are known or can be measured on-line. This is not possible for COD and TKN as these are measured off-line in a laboratory, as there are no reliable online sensors yet. The available off-line measurements are therefore not available immediately for real-time control implementation.

Another approach is to determine the expected values of the disturbance by prediction using historical data from lab measurements and soft computing (neural network) techniques (Kriger and Tzoneva, 2007a and 2007b). This approach is adopted in this work as a first step in implementation of an adaptive optimal control strategy for the ASP. This strategy determines the control of the dissolved oxygen into the aeration tank of the process using hierarchical layers of adaptation, optimization and direct control as depicted by Figure 1.2, and is implemented using a personal computer (PC), programmable logic controller (PLC) with Adroit SCADA (Supervisory Control and Data Acquisition) system, MySQL database and MATLAB.

The disturbance prediction is a part of a 3-layer control strategy developed for adaptive optimal control of the activated sludge process (ASP) through repetitive optimisation of the model parameters, controller set points and the controllers' parameters according to the diurnal variation of the input flow rate and concentrations of COD and TKN.

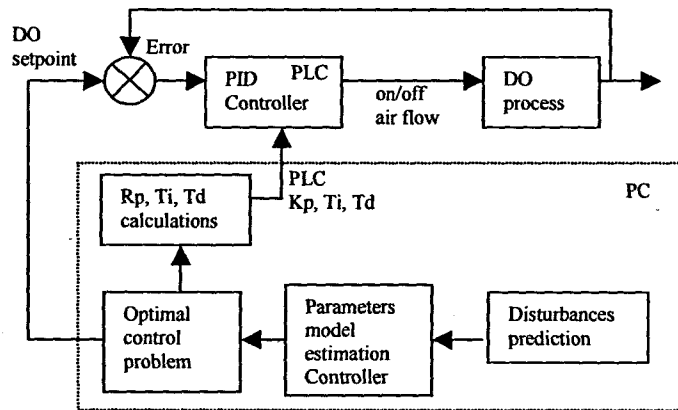


**Figure 1. 2:** Three-layer control strategy for adaptive optimal control of the ASP

This strategy is applied for development of a real-time control of a lab-scale wastewater treatment plant (WWTP) modelling a real process at the Athlone WWTP in Cape Town. Realisation of the control strategy for dissolved oxygen (DO) control is implemented in the Electrical Engineering Department at the Cape Peninsula University of Technology and is shown in Figure 1.3. The DO concentration should be



high enough to support the growth of adequate organisms, and low enough to conserve energy. This means that the controller has to send a control signal according to a set point that corresponds to the responding at the real-time disturbances.



**Figure 1. 3: Feed-forward-feedback control system for DO control**

### 1.7 Problem statement

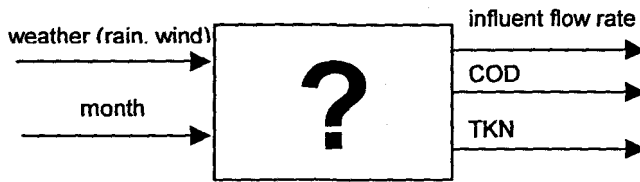
On the basis of the discussion above the problem considered in the thesis can be stated in the following way – To develop a mathematical model and technology for prediction of the influent flow COD and TKN concentrations on the basis of their dynamic behaviour caused by the weather conditions and the season (month) of the year.

There are no known investigations in this field. Different authors emphasize on the influence of the weather conditions, season, day of the week, time of the day over the influent waste concentrations and flow rate (Bechmann *et al.*, 1998), but no mathematical model is proposed. Olsson and Newell (1999) propose a formula for diurnal behaviour of the influent concentrations based on their average values for some period of time and combination of sinusoidal functions with different period. This formula is not based on the natural laws; it is constructed according to the observation of the process influent and the expert knowledge.

A more scientific approach to the solution of the problem can be developed (applied) using the capabilities of the latest developments of the methods of the Artificial Intelligence and especially of the neural network (NN) theory, based on the information that is available for the dynamic behaviour of the influent variables:

- the dependence between the weather conditions, season of the year is of the type of a dynamic “black box” one, (Figure 1.4);

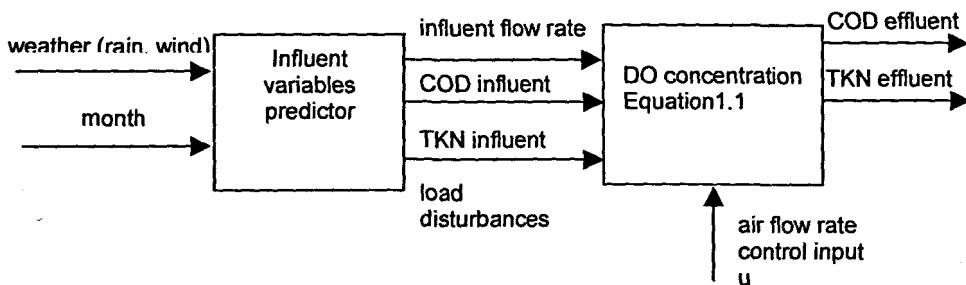
- knowledge for the relationships between the input and output variables of this model is not available at all;
- it can be assumed that the dependence between the input and output variables is deterministic;



**Figure 1. 4:** Figure illustrating that the dependence between the weather and season (month) is a type of dynamic “black box”

- there are historical data available for both input and output variables. The data for the influent are measured once every week in the scientific laboratory by exact methods and could be accepted as equal to the true values of the variables with some small error.
- The black box model is multivariable – it has many inputs and outputs. The number of outputs can be accepted to be three because the organic waste material in the influent can be grouped in two main groups, COD and TKN. The number and the type of the input variables has to be determined though the process of research in such a way that the errors between the predicted and the measured values of the disturbances are minimized.
- The black box model is a nonlinear one.

The black box model has to be used as a part of the adaptive control strategy in real time in order to act as a software sensor for the influent waste components. The outputs of this model are the load disturbances input for the considered model for the control of the DO (Figure 1.5).



**Figure 1. 5:** Connection between the process model and the influent disturbances model

The structure of the black box model can be obtained in two ways:

- assuming some type of mathematical connection between the input and output and fitting the parameters of this mathematical representation to the input-output data;
- using the NN theory approach where the structure could be supposed at the beginning and adjusted during the training in a more natural way than in the first approach.

The first approach is not very applicable because the model is multivariable and nonlinear. It is very difficult to judge the type of nonlinearity and the connections between the variables. That is why many of the difficulties of the multivariable, nonlinear fitting (regression) could be overcome by using the NN as predictors. The problem is further what exactly NN type, architecture could be the most convenient for the considered deterministic, dynamic, nonlinear multivariable model. The answer to this question determines the objectives of the thesis.

## 1.8 Neural networks

As is described in the previous section, use is made of neural networks for prediction of the inflow disturbances of COD, TKN and influent flow rate, by using historical data obtained from the Athlone WWTP and weather data obtained from the South African Weather Service.

Hecht-Nielsen (1987) defines neural networks as follows:

A neural network (NN) is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and can carry out localised information processing operations) interconnected via unidirectional signal channels called connections. Each processing element has a single output connection that branches in to as many collateral connections as desired; each carries to the same signal – the processing element output signal.

Artificial (so called to distinguish it from its biological counterpart) neural networks (ANN) have developed as a result of man's interest in the working of the human brain with the aim of achieving human-like performance. Please note that in this work the term *neural network* refers to the ANN.

### 1.8.1 Training of Neural Networks

McCulloch and Pitts proposed the first artificial neuron model in the 1940's. They described the behaviour of the neuron with the model:

$$\hat{y}(k) = \phi \left( \sum_{i=0}^N y_i w_i + b_i \right) \quad 1.2$$

where  $\hat{y}(k)$  is the output of the neuron,  $N$  is the number of inputs,  $y_i$  are the input variables,  $w_i$  are called the synapse weights and  $b_i$  is a bias term. The function  $\phi$  is a nonlinear activation function. The activation function is typically a saturating function, which normalizes the input signal to the standard values of the output signal (Papliński, 2004). These activations are typical unipolar or bipolar signals. Examples

of activation functions are the logarithmic sigmoid, hyperbolic tangent, hard limit threshold, radial basis function and the linear activation function to name but a few.

From the point of view of their active or decoding phase, NNs can be classified into feed-forward and feedback systems (Papliński, 2004). Feed-forward (static) NNs are neural networks where the network is ordered into layers with no feedback paths, while recurrent (dynamic) networks on the other hand have feedback connections from the output of one neuron back to the input of a previous layer (Mandic and Chambers, 2001).

Learning is defined as a dynamic process that modifies the weights of the network in some desirable manner. Learning can be supervised or unsupervised. For supervised learning the network is given a training set of data with a corresponding target output. Unsupervised learning also known as self-organizing is where a network develops its own classification rules using information present at the inputs presented to the network.

Neural networks (NNs) gives one the flexibility of modelling arbitrary input data by adjusting the internal network connections (weights) in such a way, that for a given input the difference between the output and the desired response, the error is minimised (Wang and Hu, 2002). In order to achieve this input-output mapping the NN has to be "trained" in order for it to model data efficiently. This training usually takes the form of supervised learning, where the network error is iteratively reduced over a training set of matching input-output vectors. Methods of training include gradient descent methods such as the very common backpropagation algorithm (Haykin, 1999).

After the network is trained, the performance has to be measured by introducing a set of data that is used to validate the model. This gives an indication of the generalisation ability of the NN and determines whether the NN has not just memorised the training data set.

### **1.8.2 Data preparation**

Data preparation is fundamental to the success of a NN to solve a given problem. If NN are presented with meaningless data, they usually produce meaningless results. A careful examination of the input data is required, and usually some knowledge of the relevance of including or excluding certain input data is advisable. Chapter 4

discusses the pre-processing techniques applied to the input data presented to the NN.

### 1.8.3 Types of Neural Networks

Broadly speaking NNs can be divided into 2 application categories, namely networks performing function approximation or regression (Masters, 1993) and those performing classification (Ripley, 1996) tasks.

Feed-forward supervised networks are mostly used for function approximation tasks and some types include:

- Linear recursive least mean square (LMS) networks;
- Backpropagation networks; and
- Radial basis networks

Feed-forward unsupervised networks are used to extract important properties of the input data and to map it into a "representation" domain. These include:

- Hebbian networks performing principal component analysis (PCA); and
- Competitive networks are used to perform learning vector quantization (LVQ).

Feedback dynamic networks are used to process the temporal features of the input data evolving over time. These include:

- Recurrent backpropagation networks;
- Associative memories; and
- Adaptive resonance networks.

(Papliński, 2004).

The feed-forward supervised networks with delayed inputs and the recurrent backpropagation network capabilities are investigated in the thesis to solve the dynamic regression problem for prediction of the influent variables.

A description of the different types of neural networks, together with all the relevant reference information, is presented in Appendix A.

### 1.8.4 Application areas for Neural Networks

Neural networks are widely applied in various disciplines for solving many real-world problems. Application areas include:

Misuse detection on computer networks (Cannady, 1998), Flood forecasting (Chang, *et al.*, 2001), River flow forecasting (Danh, Phien and Gupta, 1998), Time series (Malasri and Malasri, 2002), (Vahed, 2004), Wireless sensor networks (Wu and Jagannathan, 2007), Prediction of wind speed (Pérez-Llera, *et al.*, 2002), Nonlinear Control (Cheng, *et al.*, 2007), (Ahmida, Charef and

Becerra, 2007), and Short-term load forecasting (Vermaak and Botha, 1998, Khan and Ondrůšek, 2000).

The potential of neural networks for monitoring and control of dynamic systems was discovered a decade ago (Narendra and Parthasarathy, 1990, Bectroft et al, 1991) and many applications in industrial processes exist.

The feed-forward networks (Rumelhart et al, 1986) and recurrent networks (Atkeson et al, 1997) are most commonly used in system identification / model estimation and for control design and implementation.

These are only a few areas where NNs have be utilized to solve a particular problem.

### 1.8.5 Modelling of the Influent variables behaviour using Neural Networks

Neural networks offer the advantage of being able to model almost any system. The real problem results as one tries to apply a NN model to a specific process. The reasons for this being that considerable expertise is required in determining the number of processing elements in the network, the number of training iterations required, and the representation of data in the network

The resultant network model should be able to generalise for future unseen input vectors, provided the training set represents the process detailing the relationship between the input and output, accurately (Sakai, Homma and Abe, 2002).

The considered black box model process can be described as finite dimensional, discrete time, time invariant process by the equations

$$y(k) = f[y(k), y(k-1)...y(k-\tau), u(k), u(k-1)...u(k-\theta)] \quad 1.3$$

where  $u(k) \in R^r$  are the model inputs and  $y(k) \in R^m$  are the model outputs,  $f: R^r \rightarrow R^m$  is a nonlinear function of inputs and outputs,  $k \in 0, K$ .

The problem for modelling and parameter estimation is stated when the function  $f$  is unknown. The problem is to construct a model that displays the same input/output behaviour as the original system. This problem has to be solved using only input and output measurements (Potocnik, 2000).

For linear systems the solution of the problem is easier as the structure of the linear model is assumed previously. For the nonlinear system (ASP) the structure of the

model cannot be selected arbitrary. Then some assumptions need to be made to make the problem meaningful and tractable. Following Levin and Nerendra, 1995 the following assumptions can be made:

- 1)  $f$  is a smooth function
- 2) An upper bound of the order of the system is given.

The existence of the input/output model relies on the observability of the system which in this case cannot be checked previously and can be proved by investigation on the possibilities to make good prediction using the techniques of the NN theory.

### **1.8.6 Neural Networks implementation**

The prediction of the influent variables of COD, TKN and influent flow rate is a regression or function approximation problem. Three different neural network topologies (architectures) are investigated to solve this particular problem, namely the multilayer perceptron (MLP), the Elman recurrent neural network (ERNN) and the radial basis function neural network (RBFNN). The structures best suited for prediction problems are investigated and based on the finite impulse response (FIR) and infinite impulse response (IIR) filter concepts from signal processing (Mandic and Chambers, 2001). This is referred to as the autoregressive (AR), moving average (MA) and autoregressive moving average (ARMA) concepts from statistical theory. The practical issues that the neural network practitioner faces are investigated in this work, including the number of hidden layers, the number of hidden layer neurons, the number of training iterations, the choice of architecture, and training algorithm.

### **1.9 Research Aim**

The research aim is to develop a model predictor for the dynamic behaviour of the wastewater treatment plant influent variables (COD, TKN, Flow rate) as a result of influence of the weather conditions (temperature, wind, rainfall) and the season of the year (month) using NN theory.

### **1.10 Research Objectives**

The research objectives are divided into categories, namely the scientific objectives and the application results objectives.

#### **1.10.1 Scientific objectives**

In order to achieve the aforementioned aim, the following scientific objectives have been identified:

1. The role of the influent variables as the WWTP process main disturbances to be investigated.
2. The problem for the development of the mathematical model for the dynamic behaviour of the wastewater treatment plant influent variables to be analysed, and according to its characteristics a problem for prediction of the influent variables to be formulated as a problem for development of a neural network type and its structure.
3. Analysis of the existing types and structures of the NNs to be done according to the requirements of the formulated problem and 3 types of NN to be selected to be used:
  - Multilayer perceptron network with delayed inputs
  - Elman recurrent network
  - Radial basis function with delayed inputs
4. A procedure for preprocessing of data for the NN training and validation is to be developed on the basis of analysis of the existing approaches in the literature.
5. MLP with delayed data to be developed for one-step ahead prediction of the influent COD, TKN and Flow rate. Different variants / structures of the network to be investigated on the basis of variation of the type and the number of inputs.
6. Elman recurrent network to be developed for one-step ahead prediction of the influent COD, TKN and Flow rate. Different variants of the network to be investigated on the basis of the same as in point 5 variants of the input data.
7. Radial basis function network to be developed for one-step ahead prediction of the influent COD, TKN and Flow rate. Different variants of the network to be investigated on the basis of the same as in point 5 variants of the input data.
8. The generalized conjugate gradient method application to be investigated in order to improve the convergence and to reduce the time of the training process.
9. Different variants of the above NNs to be built and compared with those previously developed according to the number of the hidden layers, number of hidden layer neurons, number of epochs, and training to convergence (over-training).
10. MATLAB software to be written to implement all tasks for pre-processing of data, training, validation, analysis of the solutions and graphical user interface.

### **1.10.2 Application result objectives**

In order to achieve the aforementioned aim, the following application result objectives have been identified:



1. Data from the Athlone wastewater treatment plant for the influent variables and from the weather bureau to be gathered and processed.
2. Hardware structure of the control system for pilot wastewater treatment plant to be developed where the physical connections between the components of the system (plant, sensors, transmitters, PLC, PC) and their calibration to be done.
3. Data acquisition system for DO, pH, conductivity, temperature and turbidity to be developed using the sensors, transmitters, PLC, PC and MySQL database.
4. Adroit SCADA system to be developed for data acquisition, monitoring and calculation of the predicted values of the influent disturbances.
5. Communication between the software platforms Adroit, MATLAB and MySQL to be established in order to solve the problems for the influent COD, TKN and Flow rate prediction in real time. An Adroit GUI to be developed to implement the tasks for communication and prediction.

### **1.11 Literature survey**

Judging by the number of publications in journals and at conferences, there are many researchers working in the areas of neural networks and wastewater treatment processes. This information was gleaned from consulting books, journal publications, the Internet, and conference proceedings. Many authors have differing views on many issues and many 'rules of thumb' have been proposed to solve different practical implementation problems. The experience of the author has been to use these rules as guidelines, but to solve many of the practical problems with rigorous experimentation.

### **1.12 Outline of thesis**

The thesis consists of eight chapters detailing the background information and developed methods, algorithms, techniques applied, and results of the research project.

Chapter 2 briefly discusses the biological wastewater treatment process, particularly the activated sludge process for the removal of Nitrogen and Carbon. The importance of the nature of the influent disturbances of COD, TKN and inflow rate, and the effects that they have on the activated sludge process are described. The influent disturbances are characterized and the decomposition of Carbon and Nitrogen into fractional components is discussed. These fractional components are used in the mathematical model of the ASP.

Chapter 3 provides a background to the theory of neural networks, including a brief historical overview detailing the developments in the field. The process of supervised training of the neural network and the practical considerations are detailed. Here factors leading to problems such as over-training and overfitting the network are defined.

Chapter 4 deals with the important aspect of data preparation and pre-processing. Concepts such as missing data, data rejection, outlier detection, correlation, principal component analysis, trends, 'the curse of dimensionality' and circular discontinuity are addressed and the methods and techniques adopted in this work are tabled.

Chapter 5 discusses the design and practical implementation of the neural networks. The three architectures considered are the multilayer perceptron, the Elman recurrent and the radial basis function neural networks. All these architectures are applied to the problem of prediction of influent disturbances of COD, TKN and influent flow rate, one by one.

Chapter 6 presents the results of the implemented neural networks of the multilayer perceptron, the Elman recurrent and the radial basis function neural networks.

Chapter 7 describes the real-time hardware implementation and describes the different software algorithms applied throughout the work. The pilot lab scale plant, sensors and actuators, the Modicon PLC, Adroit SCADA system, MySQL database, and implemented neural network programs in MATLAB are presented.

Chapter 8 provides detailed conclusions and the future direction for research in the field of neural networks.

### **1.13 List of publications**

1. Kriger, C., Tzoneva, R. 2005. "Neural Network Structures for Prediction". Paper presented at the Postgraduate Control Systems Day, Pretoria University, Pretoria.
2. Kriger, C., Tzoneva, R. 2007. " A Neural Network model for control of wastewater treatment processes, Proc of the Int. Conf. NOLCOS 2007 DVD ROM.
3. Kriger, C., Tzoneva, R. 2007. "Neural Networks for Prediction of Wastewater Treatment Plant Influent Disturbances", Conference Proceedings of 'IEEE AFRICON' Windhoek, Namibia DVD ROM.

4. Kriger, C., Tzoneva. R. 2008. "Development of a NN software sensor for prediction of the wastewater treatment process variables", IEEE Control Systems Technology (submitted to journal).

## **CHAPTER TWO**

### **WASTEWATER TREATMENT PROCESS AND ITS DISTURBANCES**

#### **2.1 Introduction**

Clean water plays a crucial role in our daily lives and is an important resource particularly in Southern Africa where it is also a very scarce resource. Clean water is essential for plants and animals that live in water. This is important to the fishing industry, sport fishing enthusiasts, and future generations. Our rivers and ocean waters teem with life that depends on shoreline, beaches and marshes. They are critical habitats for hundreds of species of fish and other aquatic life. Migratory water birds use the areas for resting and feeding.

Wastewater is not just sewage, but used water. It includes substances such as human waste, food scraps, oils, soaps and chemicals. In homes, this includes water from sinks, showers, bathtubs, toilets, washing machines and dishwashers. Businesses and industries also contribute their share of used water that must be cleaned. Sewage is therefore created by residences, institutions, and commercial and industrial establishments. In combined municipal sewage systems, wastewater also includes storm water runoff. Although some people assume that the rain that runs down the street during a storm is fairly clean, it isn't. Harmful substances that wash off roads, parking lots, and rooftops can harm our rivers and lakes. If it is not properly cleaned, water can carry disease. People live, work and play so close to water, therefore harmful bacteria have to be removed to make water safe (USGS, 2006).

In the past people lived in relatively small population units, often in rural areas and the disposal of human waste was not a major problem (Lindberg, 1997). As rural populations expanded the accumulation of human waste became a hygienic problem. Urbanisation plays a major role in the increased production of large amounts of wastewater. This is particularly true in the Southern African region where people move from rural areas and from the northern and central regions of Africa into the major cities.

Wastewater treatment (WWT) plants are important to the economic and social development of any country. Municipal wastewater treatment plants provide an important buffer between the natural environment and the concentrated wastewaters from urban

areas. WWT plants protect the public's health from disease-causing bacteria and viruses. Treatment plants that actually disinfect wastewater eliminate many harmful organisms. WWT plants also protect water quality so that clean oceans, lakes, streams and rivers can be enjoyed.

The original aim of sewage disposal is the removal of the waterborne waste from domestic and industrial communities without causing any danger to public health. The principal aim of sewage treatment is to remove as much of the solids content as is practical and economical, and then to oxidise (and subsequently remove) the colloidal and dissolved solids before the remaining water, called effluent, is discharged back to the environment. Wastewater also contains nutrients, which can stimulate the growth of aquatic plants, and it may contain toxic compounds. Excessive amounts of organic matter in water deplete dissolved oxygen (by micro organic metabolism). This is due to the fact that as solid material decays, it uses up oxygen that is needed by the plants, animals and other aquatic life forms living in the water to survive.

The effluent when discharged should not pollute the receiving body of water, be a danger to public health, or cause a local nuisance. Nutrients can be removed from wastewater effluents either by chemical or biological means. The latter is more economical and is suitable for various forms of activated sludge systems (Lilley, Pybus and Power, 1997). The method of treatment chosen for any particular installation depends on the quality of effluent required.

This chapter discusses the wastewater treatment processes, the influent disturbances, and also give information for some of the existing mathematical models describing the activated sludge processes. Reasons are discussed for the development of an effective control strategy. Furthermore the motivation for using neural networks as a tool for developing a model to predict the influent disturbances to the wastewater plant is discussed.

## **2.2 Legislation**

The Department of Water Affairs and Forestry is the custodian of South Africa's water and forestry resources and is responsible for formulating and implementing policies governing these sectors. This is the body that establishes effluent limits for all point-source discharges. Effluents should comply with uniform standards, which are set at technologically, and economically attainable levels (Lilley, Pybus and Power, 1997).

Since 1973 the Water Research Commission in South Africa has been involved in coordinating and financing research and development in the field of nutrient removal from the activated sludge process.

The limits imposed on each institution are determined by the receiving water's capacity to assimilate the waste, as well as the use for which the water is intended. Water intended for drinking requires more protection than water used only for boating. However, all water quality should be suitable for swimming and fishing.

Legislation regarding pollution control and indirect reuse of treated wastewater is expressed in the Water Act (54 of 1956). The removal of carbonaceous material and the transformation of ammonia to nitrate are stipulated in the General and Special Standards Act of 1962 (Ekama, Marais *et al.*, 1984).

### **2.3 Current process control techniques**

Currently the technology available in South Africa to remove pollutants from wastewater has developed to a level that is in accordance with international standards. However the construction of facilities is only part of the solution. Good operation, maintenance and administrative programs must be in place, or the best of facilities produces unacceptable effluent (Bartlett, 1971).

The majority of wastewater plants in South Africa, particularly in the Western Cape region, are however manually controlled. This means that pumps and valves are switched on and off, and opened and closed, manually. Automation then, should improve the performance of wastewater treatment plants for less cost and with less energy consumption.

Currently there are a few wastewater treatment plants in South Africa that are automatically controlled to a certain degree, and monitored using some form of supervisory control. These plants are however not optimised for suitable effluent quality, or for conservation of energy (Vanrolleghem, 2000). There are at present twenty-two recognized municipal wastewater treatment plants in the Western Cape region. However of these only 14 plants are manned. The unmanned plants sometimes only have small oxygenation tanks, but these "plants" are also serviced by the Western Cape Municipality. There are six WWT plants that are known to have SCADA systems to some degree. Not all the SCADA systems are utilized throughout the entire plant, but are

sometimes limited to certain reactors only. Another fact to bear in mind is that many of the SCADA systems are used for mostly monitoring instead of controlling the plant processes. The plants that the Cape Peninsula University of Technology have been actively involved in, and that have SCADA systems, include the Cape Flats and Athlone WWT plants. The Cape Flats plant is one of the first to have a SCADA system for monitoring plant variables and performance, and to a very limited degree, controlling of certain parameters and processes. The other plants in the Western Cape with SCADA systems are Wildevoëlveifontein, Kraaifontein, Macassar and Zandvliet WWT plants. Given this perspective about 43% of plants in the Western Cape region have a SCADA system of sorts.

## **2.4 The Wastewater Treatment Process**

The biological wastewater treatment process depicted in Figure 1.1 in Chapter 1 can be divided into four stages, namely preliminary, primary, biological and secondary treatment. The importance of these phases in the biological treatment process is briefly discussed below.

### **2.4.1 Preliminary Treatment**

Preliminary treatment of wastewater is the stage in the process where sticks, bottles, cans, stone, grit, rags and other large objects are removed to protect the equipment in the plant from physical damage. This is achieved by the use of mechanical screens.

Grit removal is the next step in the process. Grit is sand and small hard inorganic material that, much like sand paper wears down expensive equipment if not removed at the beginning of the plant process. Grit chambers slow down the flow to allow grit to fall out.

### **2.4.2 Primary Treatment**

The wastewater flows from the preliminary treatment grit chambers to the primary treatment process that removes floating solid materials from the wastewater. The primary clarifiers provide a quiescent zone for the sludge to settle and be collected into hoppers by a travelling flight and chain system. Grease and other floatable material is skimmed from the surface and collected during the primary clarification process. The grease that is collected from the primary clarifiers is concentrated, and then fed into the incinerators.

### **2.4.3 Secondary Treatment**

Pollutants dissolved in the wastewater or that do not settle in the primary clarifiers flow on in the wastewater to the secondary treatment process. Secondary treatment further reduces organic matter through the addition of oxygen to the wastewater that provides an aerobic environment for the microorganisms to biologically break down this remaining organic matter.

The primary effluent enters the head end of the tanks where it mixes with return activated sludge that consists of microorganisms "activated" by the organic matter and oxygen. This combination of primary effluent and return sludge forms a mixture known as "Mixed Liquor". The mechanical mixer in each tank continuously stirs and thoroughly mixes the mixed liquor.

Once the mixed liquor goes through the complete oxygenation process, it flows to the secondary clarifiers where part of the biological solids produced during the oxygenation process are allowed to settle and be pumped back to the first tank in the process. These settled solids being pumped, called return activated sludge (RAS), mix with the primary effluent to become mixed liquor (Yu *et al.*, 2002). Since the population of microorganisms is growing some microorganisms in the return activated sludge are removed from the system. This solids waste stream is called waste activated sludge (WAS) and flows to the secondary gravity thickener for solids processing. The cleaned wastewater flows over the weir of the secondary clarifier and on to the disinfection (chlorination) process.

### **2.4.4 Biochemical Oxygen Demand (BOD)**

BOD is defined as the amount of dissolved oxygen (DO) consumed by microorganisms for the chemical oxidation of organic and inorganic matter (Bitton, 1999). BOD is also defined as a measure of the organic strength of the wastewater in parts per million or mg/l. It is determined by a simple laboratory test that measures oxygen depletion on a series of bottles of buffered distilled water seeded with active bacterial cultures. In addition, different dilutions of the wastewater being tested are added to the bottles. The organic content in the wastewater causes a depletion of oxygen that is measured after a period of five days. This is reported in parts per million or mg/l of BOD<sub>5</sub>.



#### **2.4.5 Chemical Oxygen Demand (COD)**

Chemical oxygen demand (COD) is the amount of oxygen required to oxidize the organic carbon completely to carbon dioxide (CO<sub>2</sub>), water (H<sub>2</sub>O) and ammonia. COD is therefore a measure of the oxygen equivalent of the organic matter as well as microorganisms in the wastewater. The discussion for the composition of COD follows in section 2.6.

#### **2.4.6 Total Suspended Solids (TSS)**

Also known as non-filterable solids, this is a measure of the suspended solids content of the wastewater also in parts per million or mg/l. This test is performed by filtering a known volume of the wastewater through a pre-weighed filter paper, drying it to remove the water content and reweighing it to determine the suspended solids content. These solids include both organic and inorganic matter. To determine the organic portion or volatile suspended solids (VSS), the filtered solids must be heated in a muffle furnace and the remaining portion weighed (Pons and Potier, 2002).

#### **2.4.7 Fecal and Chlorine Residual**

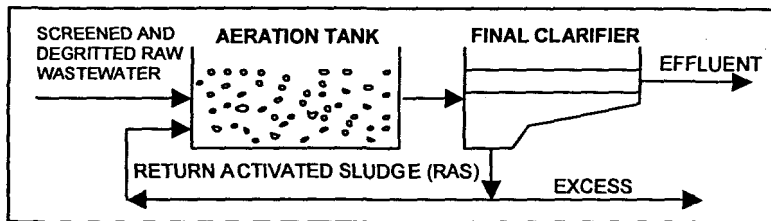
The purpose of chlorinating the effluent is to kill potentially pathogenic bacteria during the summer months when the receiving waters are most susceptible due to recreational use and assimilative capacity of the waters. Prior to final discharge, effluent chlorine residual must be reduced to less than 0.06 mg/l.

The Athlone wastewater treatment plant, which supplies the plant data for this project, does not have any chlorination stage for the treatment of the effluent so this topic is not discussed further. After secondary sedimentation, the effluent passes to maturation ponds and is finally discharged into the Liesbeeck River (Athlone plant).

#### **2.4.8 The Activated Sludge Process (ASP)**

The activated sludge process is an aerobic, suspended growth, biological treatment method. It employs the metabolic reactions of microorganisms to produce a high quality effluent by oxidation and conversion of organics to carbon dioxide, water and bio solids (sludge) (Gernaey, 2000).

Refer to Figure 2.1 for a diagram of the conventional activated sludge process using an extended aeration system. The microorganisms are kept suspended either by blowing air in the tank or by use of agitators.



**Figure 2. 1: A The Activated Sludge Process with Extended Aeration**

A basic description is that the system speeds up nature and supplies oxygen instead of the aquatic environment. High concentrations of microorganisms (compared to a natural aquatic environment) in the activated sludge use the pollutants in the primary treated wastewater as food and remove the dissolved and non-settleable pollutants from the wastewater. These pollutants are incorporated into the microorganisms' bodies and then settle in the secondary clarifiers. Oxygen needs to be supplied for the microorganisms to survive and consume the pollutants.

#### **2.4.8.1 The Aeration Tank**

In this tank aerobic oxidation of the organic matter is performed. The primary influent is introduced and mixed with the return activated sludge (RAS) to form the mixed liquor, containing 1,5 – 2,5 mg/l of suspended solids. Mechanical devices such as blowers provide aeration. The most important characteristic of the activated sludge process is the recycling of a large proportion of the biomass. The sludge age (see below) is much greater than the hydraulic retention time. The time that the organisms remain in this tank varies from between 4 to 8 hours.

#### **2.4.8.2 The Sedimentation Tank**

The tank is used for the sedimentation of sludge (microbial flocs) produced during the oxidation phase in the aeration tank. Only a portion of the sludge in the sedimentation tank is recycled back to the aeration basin, the rest is removed as excess sludge in order to maintain a constant sludge concentration despite the growth of the microorganisms.

#### **2.4.8.3 Mixed liquor Suspended Solids (MLSS)**

The mixed liquor suspended solids (MLSS) is the total amount of organic and mineral suspended solids, including microorganisms, in the mixed liquor. MLSS is calculated by filtering an aliquot of mixed liquor, drying the filter at 105 °C, and determining the weight of solids in the sample.

#### 2.4.8.4 Mixed liquor Volatile Suspended Solids (MLVSS)

The organic portion of MLSS is represented by mixed liquor volatile suspended solids (MLVSS), which comprises of non-bacterial organic matter as well as dead and live microorganisms and cellular debris. MLVSS is determined by heating of dried filtered samples at 600 –650 °C, and represents approximately 65-75% of MLSS.

#### 2.4.8.5 Food-to-microorganism ratio

One of the strategies used to control the ASP is the food-to-microorganism ratio (F:M). This ratio (F:M) indicates the organic load into the activated sludge system. It is expressed in kilograms BOD per kilogram of MLSS per day. F:M is controlled by the rate of activated sludge wasting. The higher the wasting rate, the higher the F/M rate.

$$F : M = \frac{Q \times BOD}{MLVSS \times V_a} \quad 2.1$$

where  $Q$  is the flow rate of the influent in litres per day (l/d),  $BOD$  is the 5 day biochemical oxygen demand (mg/l),  $MLVSS$  are the mixed liquor volatile suspended solids (mg/l), and  $V_a$  is the volume of aeration tank (litres).

A low F:M ratio indicates that the microorganisms in the aeration tank are starved, thus leading to a more efficient wastewater treatment (Fuente, Sainz Palmero and Pindado, 2002).

The F:M control strategy attempts to maintain a specific balance between the mass organic contaminant (F) applied to the aeration basin, and the mass of microorganisms (M) contained within the basin.

#### 2.4.8.6 Hydraulic Retention Time

The hydraulic retention time (HRT) is the time that the influent spends in the aeration tank. It is expressed as:

$$HRT = 1 / D = V_a / Q \quad 2.2$$

where  $V_a$  is the volume of aeration tank,  $Q$  is the flow rate of the influent into the aeration tank and  $D$  is the dilution rate.

#### 2.4.8.7 Sludge Age

Sludge age (also known as the sludge retention time) is the mean residence time of microorganisms in the system. This time could be in the order of days (not hours). It is expressed as:

$$\text{Sludge\_age} = \frac{MLSS \times V_a}{SS_e \times Q_e \times SS_w \times Q_w} \quad 2.3$$

where  $MLSS$  is the mixed liquor suspended solids (mg/l),  $V_a$  is the volume of the aeration tank(L),  $SS_e$  are the suspended solids in wastewater effluent (mg/l),  $Q_e$  is the flow rate of wastewater effluent ( $m^3/day$ ),  $SS_w$  are the suspended solids in wasted sludge (mg/l), and  $Q_w$  is the flow rate of wasted sludge ( $m^3/day$ ) (Lubenova, Simeonov and Queinnec, 2002).

Sludge age may vary from 5 to 15 days in conventional activated sludge process. It is usually higher in winter than summer.

The important parameters controlling the operation of an activated sludge process are organic loading rates, oxygen supply, and control and operation of the final settling tank. This tank is used for clarification and thickening of the sludge.

#### 2.4.8.8 Biological Nitrogen Removal (BNR)

Nitrogen is an essential nutrient for biological growth, and is one of the main constituents in all living organisms (Carlsson and Lindberg, 1996). The discharging of nitrogen into lakes, river, dams and other quiescent bodies of water has seen acceleration in the growth of algae. This process is known as eutrophication (Lukasse, 2001), and can be visibly noticed as large blooms of algae. Excessive algae growth is detrimental to the aquatic ecosystem. The removal of nitrogen in the wastewater effluent is therefore of the utmost importance. The biological removal of nitrogen is one of the most important and most intensely studied topics (Gernaey, 2000). This is due to the more stringent effluent criteria for nitrogen that have imposed for new and existing treatment plants (Nyberg et al., 1992).

Nitrogen in wastewater can be subdivided into two main forms: free and saline ammonia and organically bound nitrogen. The organically bound nitrogen can be subdivided further into non-biodegradable and biodegradable.

Nitrogen is characterised by the Total Kjeldahl Nitrogen (TKN) and the free and saline ammonia tests. Nitrogen is present in several forms, e.g. ammonia (NH<sub>3</sub>), ammonium (NH<sub>4</sub><sup>+</sup>), nitrate (NO<sub>3</sub><sup>-</sup>), nitrite (NO<sub>2</sub><sup>-</sup>) and as organic compounds. In rivers, lakes and dams the presence of nitrogen gives rise to some problems:

- Ammonia is toxic to aquatic organisms especially fish. The dissolved oxygen concentration may be severely depleted when ammonium is oxidized to nitrate, as there is a significant oxygen demand from the receiving water.
- Nitrite in drinking water is toxic, especially for infants. The chlorine dosage in drinking water is increased due to the presence of ammonia.

The biological nitrogen removal is performed in the anoxic and aerobic basins (tanks) of the ASP. The aerobic zone has dissolved oxygen present, while the anaerobic zone has neither dissolved oxygen or nitrate. The anoxic zone is characterised by its lack of oxygen, but not nitrate (Figure 2.3). The anaerobic zone is required to effect the removal of phosphorus, while the anoxic zone is required for the removal of nitrogen (Lilley, Pybus and Power, 1997).

The ammonium can be removed by oxidizing it to nitrate during aeration (dissolved oxygen is present). This process is known as *nitrification* (Fruchard, Bernard and Gouzè, 2002) and is described by the following equations:



From the chemical reaction simplified in equations (2.4) and (2.5) it can be seen that, ammonium (NH<sub>4</sub><sup>+</sup>) is first oxidized to nitrite (NO<sub>2</sub><sup>-</sup>) and then to nitrate (NO<sub>3</sub><sup>-</sup>).

After nitrification, the nitrate can now be converted to nitrogen gas by *denitrification*. This occurs in anoxic conditions (oxygenation is performed using nitrate).



For denitrification, anoxic zones are required in the activated sludge process. These zones can be either before the aeration zone (pre-denitrification), or after the aeration zone (post-denitrification). For pre-denitrification, an extra recirculation flow is introduced to transport the nitrified water back to the first zone (Figure 2.2). Denitrification is limited by the amount of carbon entering the system (Carlsson and Lindberg, 1996).

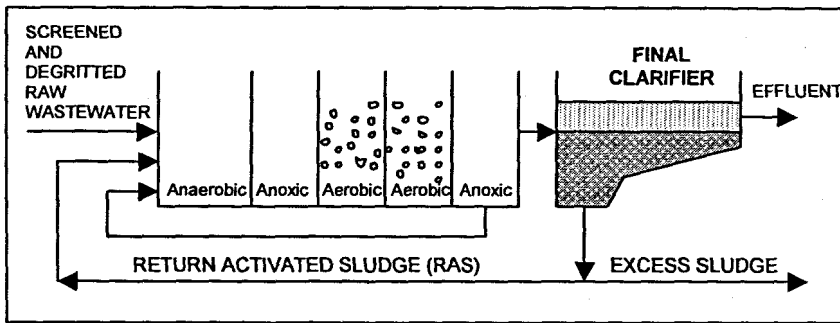


Figure 2. 2: The Activated Sludge Process with Pre-Denitrification

### 2.5 Disturbances to the Wastewater Treatment Process

The technological structure of the Athlone plant ASP is based on the UCT physical wastewater treatment plant (WWTP) model structure and has three tanks: anaerobic, anoxic and aerobic, and one secondary settling tank as depicted in Figure 2.4. Disturbances can be either *internal* originating from the plant itself, or *external* imposed on the plant by the surrounding environment. One important reason why there is an interest in dynamics when dealing with wastewater treatment, is the nature of the influent disturbances, which are characterized by varying values of the influent flow rate and waste concentrations mainly represented by the chemical oxygen demand (COD) and total Kjeldahl nitrogen (TKN), Figure 2.3, (Olssen and Newell, 1999).

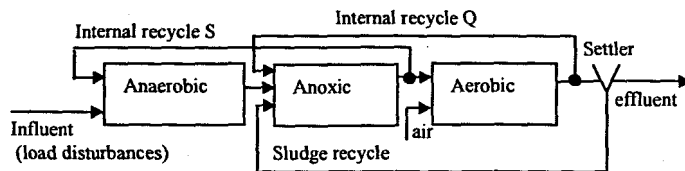
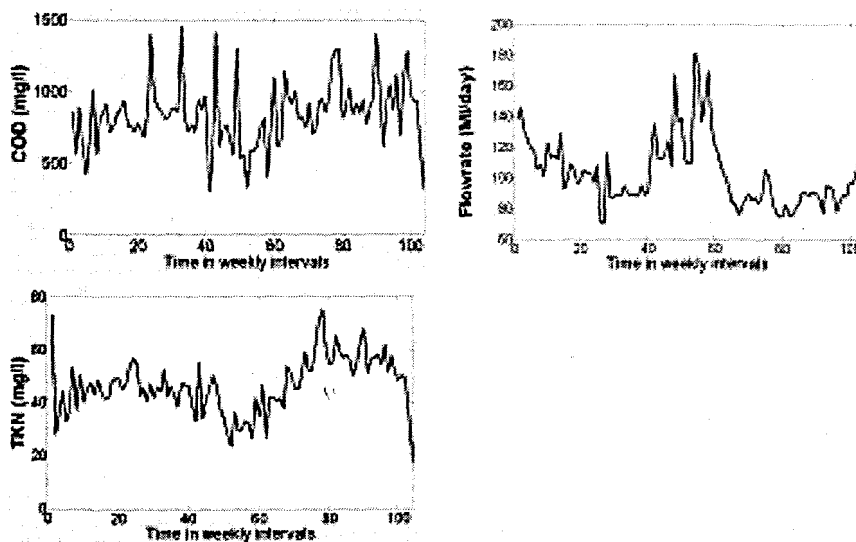


Figure 2. 3: The technological structure for the Athlone WWTP based on the UCT model

The variations of the inflow characteristics for the Athlone plant are presented in Figure 2.4. The uncertainty in sludge activity and composition caused by unpredictable change in the external disturbances due to seasonal variations in temperature and rainfall requires an adaptive control strategy to be developed and applied to the ASP. The variation of the values of the input disturbances determine different operating conditions for the process, which means that the control strategy has to be capable of generating control action according to the change in operating conditions. Such type of control can be achieved if the values of the disturbances are known or can be measured on-line. This

is not possible for COD and TKN as there is still a lack of reliable on-line sensors. The measurement of the load disturbances is done at the influent where the flow rate of the wastewater is measured by a flow meter and the COD and TKN concentrations are measured by sampling. The available off-line measurements are calculated in a laboratory and are not available immediately for real-time control implementation. These data can be used on the basis of the methods of the Artificial Intelligence to develop predictors for the future values of the inflow concentrations of the COD and TKN. The predicted values can then be used for building of the considered adaptive control strategy in Chapter 1.



**Figure 2. 4: Dynamics of the influent disturbances for the Athlone WWT Plant**

The problem for prediction of the inflow disturbances is based on the analysis of the environmental factors that could influence the input flow characteristics. The literature review and plant operator experience showed that such factors include:

- weather conditions - temperature, wind velocity, rainfall, etc.;
- season, day of the week, time of day;
- human activities; and
- industrial activities.

The first two groups of factors are selected as a basis for disturbance prediction as some pattern in the behaviour of the process can be observed when they are changed. The weather data was obtained from the South African Weather Service.

Then the problem for input disturbances prediction can be formulated by finding a NN model describing the behaviour of the inflow characteristics on the basis of variation of

the considered inflow characteristics, the weather conditions and month of the year, in such a way that the mean squared error between the real data for the inflow characteristics and the NN prediction is minimum.

## 2.6 Influent Wastewater Characterization

The characteristics of the external disturbances COD and TKN and their influences on the wastewater composition and microorganisms are discussed. The discussion is based on the UCT (University of Cape Town) biological model Dold *et al.*, (1991); and on the biological Activated Sludge Model 1 (ASM1) of the IAWQ (International Association on Water Quality), (Jeppson 1996 and Henze *et al.*, 1987).

### 2.6.1 Chemical oxygen demand (COD) components in the UCT model

The carbonaceous material representation in the UCT model is in terms of the chemical oxygen demand (COD). The total COD ( $S_{ii}$ ) is divided into biodegradable COD ( $S_{bi}$ ), non- biodegradable COD (inert mass) ( $S_{ui}$ ) and biomass (Figure 2.5). The influent biodegradable and non- biodegradable COD are further subdivided into:

*Unbiodegradable sub-fractions:* The influent unbiodegradable COD is divided into two fractions, namely soluble COD ( $S_{usi}$ ) and unbiodegradable particulate COD ( $S_{upi}$ ). Both are assumed to be unaffected by the biological action in the ASP. The ( $S_{usi}$ ) passes out in the secondary settling tank overflow while the ( $S_{upi}$ ) is enmeshed in the sludge mass and accumulates as inert VSS (volatile suspended solids).

*Biodegradable sub-fractions:* The influent COD is divided into two fractions, namely readily biodegradable COD (RBCOD,  $S_{bsi}$ ) and slowly biodegradable COD (SBCOD,  $S_{bpi}$ ). RBCOD consists of molecules that can be readily absorbed by the microorganisms and metabolized for energy and synthesis, while the SBCOD is made up of particulate organic molecules that require extracellular enzymatic breakdown prior to absorption and utilisation.

The division of the influent wastewater COD into fractions is defined by the fractional constants,  $f_{s,us}$ ,  $f_{s,up}$  and  $f_{bs}$  where,  $f_{s,us}$  is the fraction of the total influent COD which is unbiodegradable soluble;  $f_{s,up}$  is the fraction of the total influent COD which is



unbiodegradable particulate;  $f_{bs}$  is the fraction of the biodegradable influent COD which is readily biodegradable. The division of the influent COD into the various sub-fractions, using these fractional constants, can be expressed as:

$$S_{usi} = f_{s,us} \cdot S_{ti} \quad 2.7$$

$$S_{upi} = f_{s,up} \cdot S_{ti} \quad 2.8$$

The biodegradable influent COD ( $S_{bi}$ ) is given by the difference of the total influent COD  $S_{ti}$  and the sum of  $S_{usi}$  and  $S_{upi}$ .

$$\begin{aligned} S_{bi} &= S_{ti} - (S_{usi} + S_{upi}) \\ &= S_{ti} - (S_{ti} \cdot f_{s,us} + S_{ti} \cdot f_{s,up}) \\ &= S_{ti}(1 - f_{s,us} - f_{s,up}) \\ &= S_{bsi} + S_{bpi} \end{aligned} \quad 2.9$$

where,

$$S_{bsi} = f_{bs} \cdot S_{bi} \quad 2.10$$

$$S_{bpi} = (1 - f_{bs}) \cdot S_{bi} \quad 2.11$$

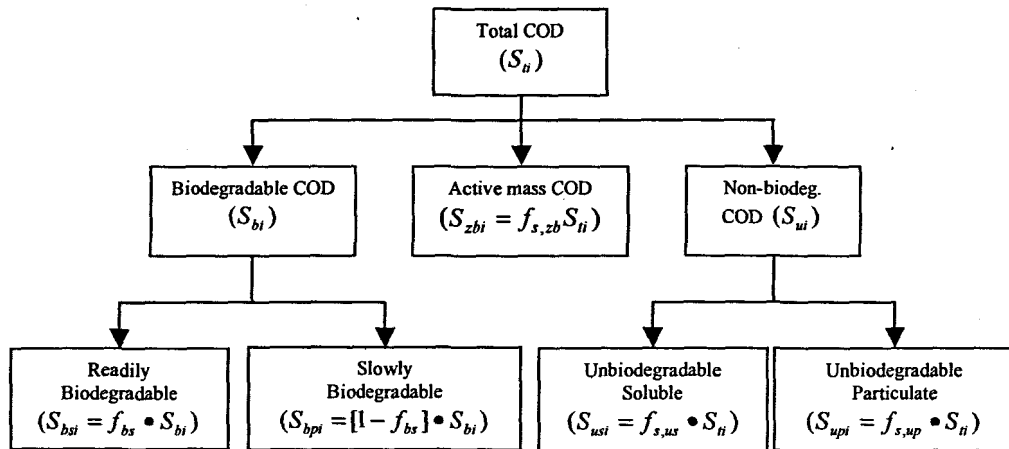


Figure 2. 5: COD components in the influent for the UCT model

The IAWQ Task group proposed another influent COD fraction, the COD of active aerobic organisms present in the influent. In both models (UCT and ASM 1), this COD fraction ( $S_{zbi}$ ) is with fractional constant  $f_{s,zb}$  where  $S_{zbi} = f_{s,zb} \cdot S_{ti}$ .

For South African municipal wastewaters, it appears that the fractional constant  $f_{s,zb}$  can be assumed to be zero as the sewers are generally anaerobic.

### 2.6.2 Nitrogen components in the UCT model

The division of nitrogenous material in the influent wastewater is illustrated in Figure 2.6. Based on the measurements of total Kjeldahl nitrogen (TKN), the nitrogen ( $N$ ) is divided into ammonia nitrogen, organically bound nitrogen and active mass nitrogen, that is, a fraction of the biomass that is assumed to be nitrogen.

#### *Free and saline ammonia and organically bound fractions*

The TKN ( $N_{ii}$ ) is divided into free and saline ammonia ( $N_{ai}$ ) and organically bound  $N$  ( $N_{oi}$ ). The influent organically bound nitrogen ( $N_{oi}$ ) is further divided into unbiodegradable ( $N_{oui}$ ) and biodegradable ( $N_{obi}$ ).

#### *Unbiodegradable organic nitrogen ( $N_{oui}$ )*

This nitrogen has two forms, unbiodegradable soluble ( $N_{ousi}$ ) and unbiodegradable particulate ( $N_{oupi}$ ). The unbiodegradable organic nitrogen is unaffected by biological action in the system.

#### *Biodegradable organic nitrogen ( $N_{obi}$ )*

The influent biodegradable organic nitrogen has two forms, soluble ( $N_{obsi}$ ) and particulate ( $N_{obpi}$ ). The division of the influent wastewater TKN into fractions is defined by the fractional constants,  $f_{n,a}$ ,  $f_{n,ous}$  and  $f_{n,obp}$  where,  $f_{n,a}$  is the fraction of the total influent TKN which is free and saline ammonia;

$f_{n,ous}$  is the fraction of the total influent TKN which is organic unbiodegradable soluble;

$f_{n,obp}$  is the fraction of the organic biodegradable  $N$  which is particulate. The division of the influent total TKN ( $N_{ii}$ ) into the various sub-fractions, using these fractional constants, can be expressed as:

$$N_{ai} = f_{n,a} \cdot N_{ii} \quad 2.12$$

$$N_{ousi} = f_{n,ous} \cdot N_{ii} \quad 2.13$$

The influent unbiodegradable particulate organic nitrogen ( $N_{oupi}$ ) is associated with the influent unbiodegradable particulate COD ( $S_{upi}$ ) and is expressed as follows:

$$N_{oupi} = f_{z,n} \cdot f_{s,up} \cdot S_{ti} \quad 2.14$$

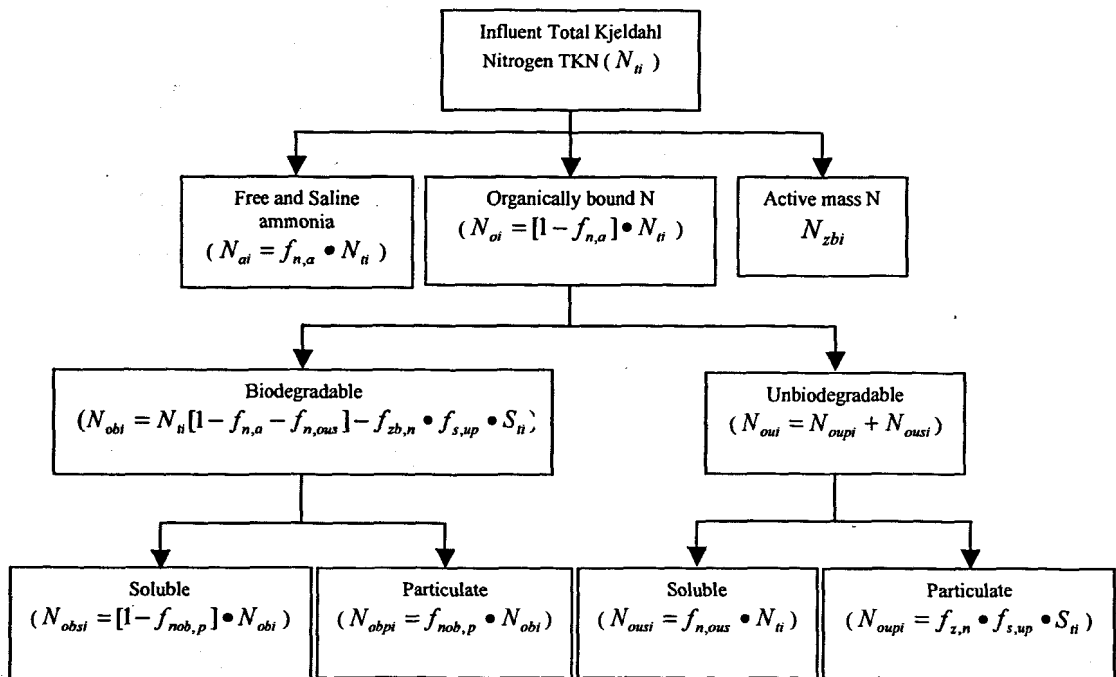
where  $f_{z,n}$  is the fraction of VSS (COD units) which is  $N$ . The influent biodegradable organic nitrogen ( $N_{obi}$ ) is given by the difference of the total influent TKN ( $N_{ti}$ ) and the sum of  $N_{ai}$ ,  $N_{ousi}$  and  $N_{oupi}$ .

$$\begin{aligned} N_{obi} &= N_{ti} - N_{ai} - N_{ousi} - N_{oupi} \\ &= N_{ti} - (f_{n,a} \cdot N_{ti} - f_{n,ous} \cdot N_{ti} - f_{z,n} \cdot f_{s,up} \cdot S_{ti}) \\ &= N_{ti}(1 - f_{n,a} - f_{n,ous}) - f_{z,n} \cdot f_{s,up} \cdot S_{ti} \end{aligned} \quad 2.15$$

The soluble ( $N_{obsi}$ ) and particulate ( $N_{obpi}$ ) influent biodegradable organic nitrogen concentrations are:

$$N_{obsi} = (1 - f_{n,obp}) \cdot N_{obi} \quad 2.16$$

$$N_{obpi} = f_{n,obp} \cdot N_{obi} \quad 2.17$$



**Figure 2. 6: Nitrogen components in the influent for the UCT model**

The IAWQ Task group recommended that an influent biological (active) mass nitrogen fraction ( $N_{zbi}$ ), is included in both models (UCT and ASM 1), this COD fraction ( $S_{zbi}$ ) is with fractional constant  $f_{z,zb}$  where  $N_{zbi} = f_{z,n} \cdot S_{zbi}$  and where  $f_{z,n}$  is the fraction of biological (active) mass which is nitrogen.

### **2.6.3 Influent Wastewater Characterization Summary**

Process variables and influent wastewater variables are correlated as can be seen on the basis of the above analysis of the components of the influent wastewater and of the state variables (components) of the ASP models from Figures 2.5 and Figure 2.6 and from the model equations. The current values of the process variables depend on the current values of their corresponding influent parts. This means that the influent values of the wastewater components have to be known by measurement in order to be capable to properly monitor and control the ASP. As stated previously, there are still not reliable on-line sensors to measure the carbonaceous and nitrogen components, currently only the flow rate can be measured on-line. One of the solutions is to apply the achievements of the modern control theory and to predict the values of the inflow components on the basis of the regression or soft computing techniques. The latter approach is applied in the thesis where only the total COD, TKN and influent flow rate are predicted. The values of the components of COD and TKN are determined as a second step on the basis of the obtained predictions using the fractional coefficients described above.

### **2.7 Summary of the chapter**

The chapter introduces the wastewater treatment process, and the motivation for WWT is discussed. Issues such as legislation and effluent requirements and compliance in South Africa specifically are addressed. The current situation in the Western Cape region in terms of municipal WWT plants and automation are tabled. The structure of the WWP is examined together with the processes such as preliminary, primary, biological, and secondary treatment that are involved in ensuring effective removal of nutrients. The biological treatment of wastewater specifically the activated sludge process is described. The influent process disturbances chemical oxygen demand (COD), total Kjeldahl nitrogen (TKN) and influent flow rate, are identified and characterized, and reasons as to why we need to predict these load disturbances are discussed. The total carbon and nitrogen inflow components and their various fractional compositions with relation to the UCT model are described.

In order to implement effective control of the ASP by controlling the dissolved oxygen, the load disturbances have to be predicted on the basis of past plant behaviour and environment conditions. This work utilizes this approach using neural networks to predict the expected values of the disturbances with historical data from a municipal plant, together with environmental variables such as temperature, rainfall and wind speed.

Chapter 3 provides a background to the theoretical aspects of neural networks and includes a historical overview of the important developments in this field. Various important topics such as learning (training), validation (testing), neural network architectures, and generalization aspects are also discussed.

# CHAPTER THREE

## NEURAL NETWORKS BACKGROUND

### 3.1 Introduction

A neural network is an information processing system that simulates the way the human brain works. The neural network is usually a computer-based software algorithm and it operates differently to a conventional computer.

Work in the area of artificial neural networks have been motivated by the fascination of mankind with the complexity of the human brain. The brain is a highly complex, nonlinear and parallel computer (information processing centre). It can perform certain computations many times faster than a digital computer in fields such as:

- Pattern recognition;
- Perception;
- Control; and
- Vision.

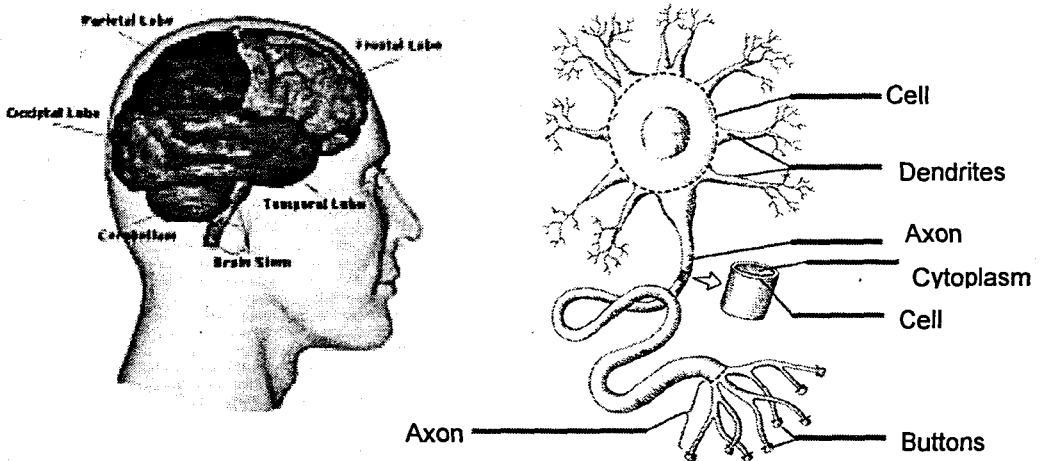


Figure 3. 1: Structure of the brain and the biological neuron

Biological neurons are the processing elements in the human brain. A neuron typically consists of 3 parts, namely the cell body, dendrites and axon (Figure 3.1). The cell body contains the nucleus and extends to the dendrites and axons. Dendrites receive nerve impulses from neighbouring neurons, the cell body processes them and the axon sends the resulting signals to other neurons. The point where the signals are exchanged is called the synapse.

The human brain consists of ten billion neurons and each neuron can interact with as few as a thousand or as many as two hundred thousand other neurons. Due to the

huge number of interconnections, the brain is able to adapt to external signals and make reasonable reactions (Park, 1996).

The brain is able to react to signals it has never experienced before because of the massive connections between neurons interacting with each other. The correct reactions are due to the genetic programming capability and the learning ability of the brain to respond to the events. This fact is the main focus of the development of an artificial neural network (ANN), or simply a neural network (NN) (Zupan 1993). The term "artificial" is used to differentiate between its biological counterparts. In this document the term "*neural network*" simply refers to the artificial NN.

### **3.2 Artificial Neural Networks**

A neural network is defined as an information processing system that simulates the human brain. The work on artificial neural networks has been motivated by the fact that the human brain computes entirely differently from the conventional digital computer (Haykin, 1999).

Haykin defines the neural network as a massively parallel, distributed processor which is made up of simple processing units, having the ability to store experiential knowledge and making it available for use. Neural networks resemble the brain in two respects:

- Knowledge is acquired from its environment through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the acquired knowledge.

The basic processing element of a neural network is a neuron. Neurons can have multiple inputs and a single output. Two basic components of neurons are the summing junction and activation functions (Figure 3.2).

### **3.3 A brief history of neural networks**

To provide the background to this particular work, and because there is such a wealth of material and applications for neural networks, a brief history has been included.

The history of neural networks can be traced back to the work of trying to model the neuron. The original biological motivation for the feed-forward neural network stems from McCulloch and Pitts (1943). The model they created had two inputs and a single output. The weights for the inputs were kept equal and the output was either logic 1 or logic 0. The output would remain zero until the weighted sum of the inputs reached a certain threshold. The McCulloch and Pitts neuron became known as a logic circuit.

The structure of an artificial neuron is shown in Figure 3.2.

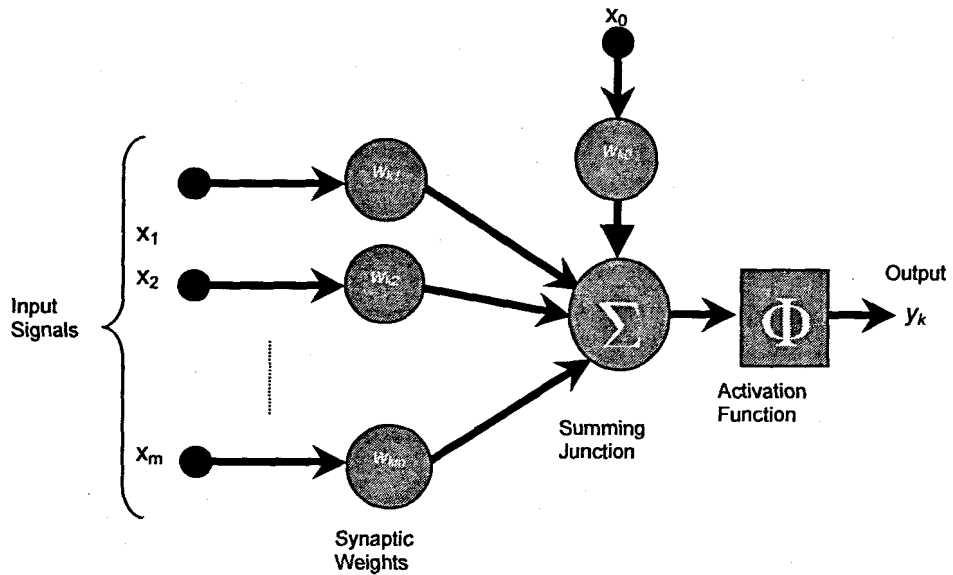


Figure 3. 2: Structure of an artificial neuron

The output of the neuron is given as:

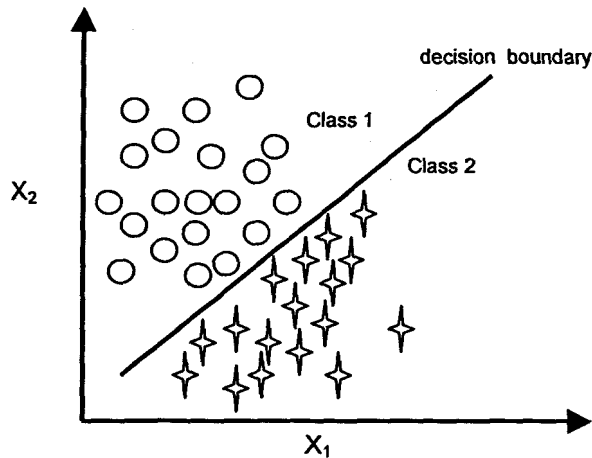
$$y_k = f(w_{k0}x_0 + \sum_{j=1}^m w_{kj}x_j) \quad 3.1$$

where,  $y_k$  is the neuron output,  $w_{k0}$  and  $x_0$  are the bias weight and magnitude,  $x_j$  is the input signal and  $w_{kj}$  are the weights.  $\phi$  is the activation function computed at the summation node.

Donald Hebb was concerned about the adaptation laws involved in neural systems, and in 1949 devised a learning rule for adapting the connections within artificial neurons (Hebb, 1949).

The Perceptron was the next neuron model developed by Rosenblatt (1958). Perceptrons are especially suited for simple problems in pattern classification. Rosenblatt randomly interconnected the perceptrons and used trial and error to randomly change the weights to achieve "learning". The McCulloch and Pitts' neuron was however a much better model for the electrochemical process that goes on inside the biological neuron than the perceptron, which is the basis for the modern day field of neural networks (Anderson and Rosenfeld, 1988). Unfortunately the major drawback of the perceptron is that it is only capable of classifying problems that are linearly separable at the output (Masters, 1993 and Demuth *et al.*, 2004).





**Figure 3. 3: A classification problem illustrating linear separable classes**

Figure 3.3 shows a hypothetical classification problem with two feature variables  $X_1$  and  $X_2$ . The decision boundary, denoted by the solid line, is able to provide good separation of the two classes.

Around 1954, Minsky introduced reinforcement learning. (Ng, 1997).

The next major development was the formulation of new learning rules by Widrow and Hoff in their application to a simple adaptive linear neuron (Adaline) (Widrow and Hoff, 1960). The rule was also known as the least mean square (LMS) algorithm. The extension of the ADALINE to multi-layer network is called MADALINE.

A significant breakthrough in neural networks was made in 1967 with the introduction of the smooth sigmoid activation function by Cowan. This function has the ability to approximate any nonlinear function. Instead of switching either on or off like the perceptron, this activation function turns the output on gradually when activated. For this and other common activation functions see section 3.5.

In 1969 Minsky and Papert published their book *Perceptrons* (Minsky and Papert, 1969) in which they showed the deficiencies of the perceptron models. Most neural network funding was redirected and many researchers left the field. Only a few researchers continued notably Teuvo Kohonen, Stephen Grossberg, James Anderson and Kunihiko Fukushima (Kröse and van der Smagt, 1996). The interest in neural networks re-emerged only after some important theoretical results were attained in the eighties (most notably backpropagation).

In 1974 Werbos published the backpropagation learning method, but this did not receive a lot of publicity. This learning method deals with propagation of the information about the errors back to the hidden layer.

Parker (1985) and Le Cun (1986) rejuvenated the interest in backpropagation. The error backpropagation method published by Rumelhart, Hinton and Williams (1986), could train multi-layered neural networks to perform tasks where the perceptron had failed. It was after this publication that the backpropagation was seen as a breakthrough in neural network literature with the introduction of the generalised delta rule. The backpropagation method has been used successfully in numerous applications such as filtering operations, handwritten digit recognition, control systems, etc. For a more detailed discussion on the history of NN see Ng (1997) and Tarassenko (1998).

### **3.4 Network Architecture (Topology)**

There are broadly speaking, essentially two types of NN topologies, namely the feed-forward neural network (FNN) and the recurrent neural network (RNN). The fully connected 3-layer FNN and RNN are illustrated in Figure 3.4 and 3.5 respectively. There are a large number of different neural network topologies. For a more comprehensive list of the different neural network topologies together with the sources consulted see Appendix A. This list is by no means the only types of NNs that are available.

Feed-forward NNs are restricted to finite-dimensional input and output spaces. RNNs can in theory process arbitrarily long strings of numbers or symbols. But training recurrent NNs has posed much more serious practical difficulties than training feed-forward networks.

Finding an optimal network configuration is one of the most important problems neural network designers face. The following neural network parameters should be selected:

- the number of hidden layers;
- the number of neurons in each hidden layer; and
- the type of activation function, which can vary in the same layer.

These parameters must be selected carefully to avoid overfitting and underfitting and, hopefully, to make the learning phase convergent (Khan and Ondrušek 2000). Due to a lack of information about the variables represented in the hidden layer, NNs are considered by most researchers to be a “black box” model (Kabundi, 2002).

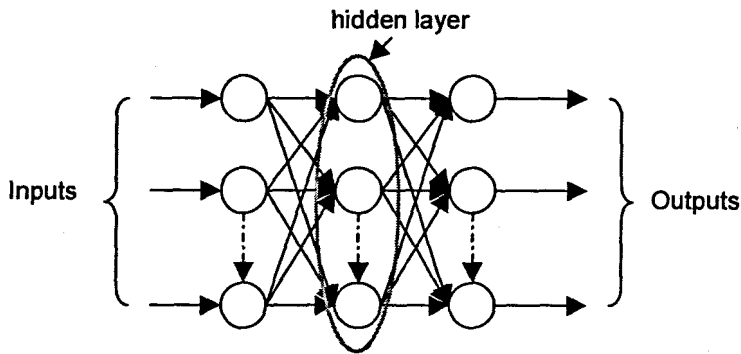


Figure 3. 4: Feed-forward Neural Network

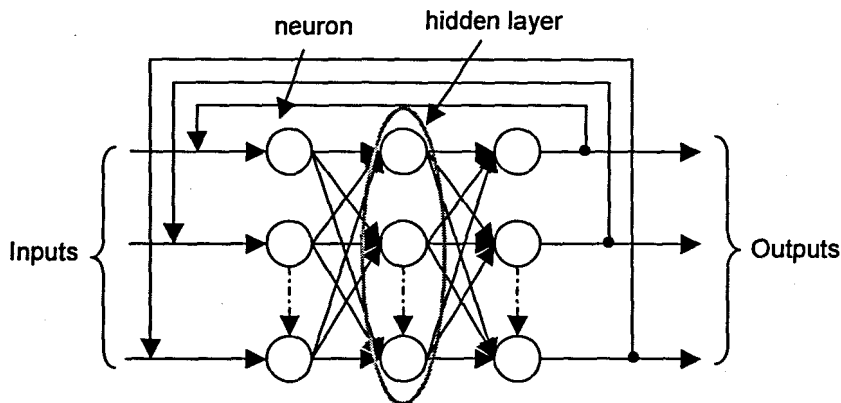


Figure 3. 5: Recurrent Neural Network

### 3.5 Activation functions

The output of the neuron is determined by the activation (or transfer) function of the neuron. The aim of the activation function is to introduce nonlinearity into the network. This nonlinearity combined with the interconnections of the neurons accomplishes proper mapping from input signals to the corresponding output activities (Zupan 1993). Without this nonlinearity the multilayer perceptron would not be functionally different from a linear filter and would not be able to perform nonlinear separation and trajectory learning for nonlinear and non-stationary signals (Mandic and Chambers, 2001).

The transfer function can be something as simple as a binary function also called a hard limiter or step function (Figure 3.6). The output depends upon whether the product is positive or negative (Equation 3.2 and 3.3). The output is '1' for a positive product and '0' for a negative product. The output can also be bipolar (between +1 or -1) as indicated in Figure 3.7.

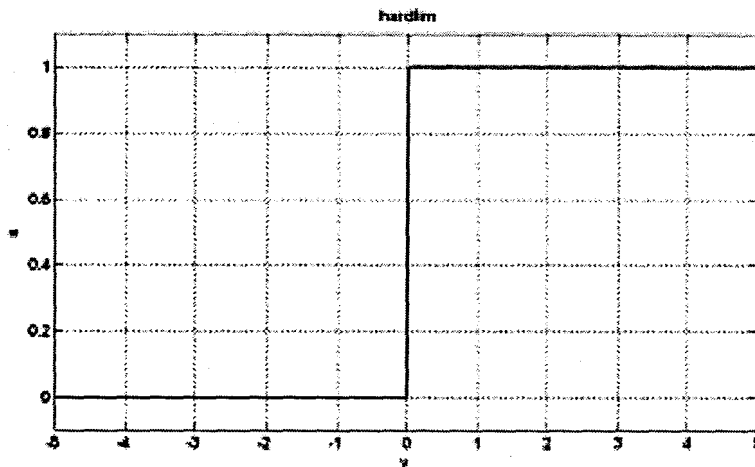


Figure 3. 6: Binary (Threshold) Activation function

$$a(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

3.2

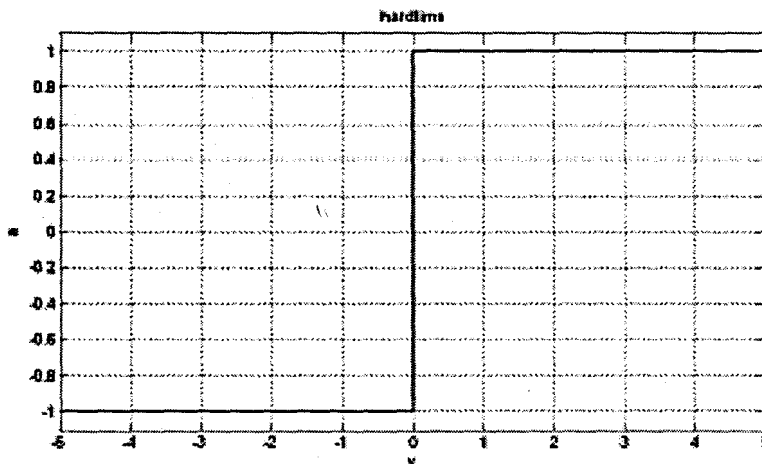


Figure 3. 7: Bipolar Binary (Threshold) Activation function

$$a(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$

3.3

Another very popular activation function is the sigmoid or 'S'-shaped curve (Figure 3.8). The sigmoid (Equation 3.4) has the ability to approximate to a certain degree of accuracy any nonlinear function. The sigmoid is also known as a "squashing" function, since the infinite input range is compressed into a finite output range. Here the curve asymptotically approaches a minimum and maximum. The difference between the sigmoid and the binary functions are that the logistic sigmoid turns on gradually when activated. Sigmoid functions are characterized by the fact that their

slope must approach zero, as the input gets large (Demuth *et al.*, 2004). This is also when the function is almost insensitive to input received from the input layer, and the output is inactive. Mathematically the function and its derivative are continuous.

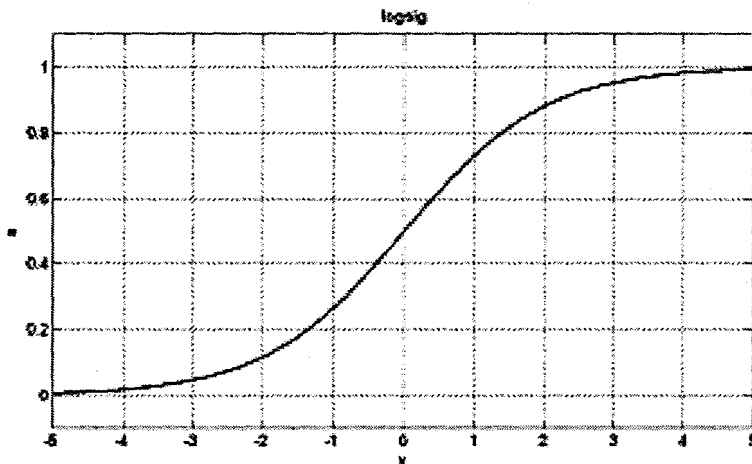


Figure 3. 8: Logistic Sigmoid Activation function

$$a(v) = \frac{1}{1 + e^{-av}} \quad 3.4$$

The hyperbolic tangent activation function (Figure 3.9) is similar to the sigmoid, but the output is bipolar (Equation 3.5). The tangent tends to yield faster training models than functions generating only positive values such as the sigmoid.

$$a(v) = \tanh(av) = \frac{e^{av} - e^{-av}}{e^{av} + e^{-av}} \quad 3.5$$

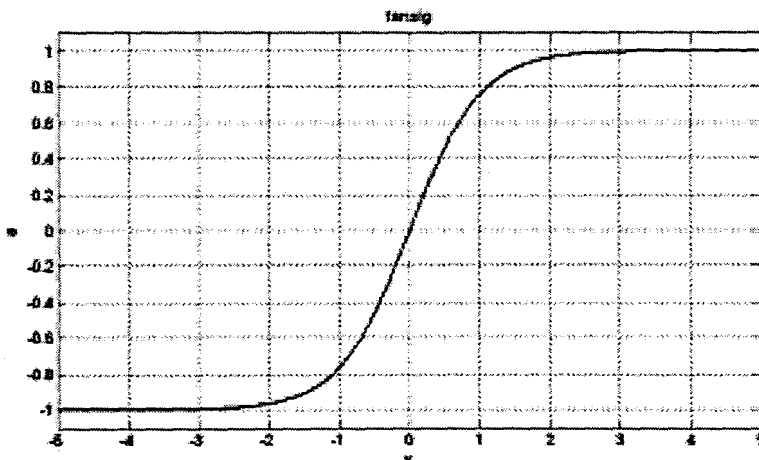


Figure 3. 9: Hyperbolic Tangent Activation function

3.5

The pure linear function (Equation 3.6) is a linear activation function and does no compression of the values. The linear transfer function calculates the neuron's output

by simply returning the value passed to it. The output layer for most function approximation and regression applications is usually a linear function. If the last layer of a multi-layer network has sigmoid neurons, then the outputs of the network are limited to a small range. If linear output neurons are used the network outputs can take on any value.

$$a = \text{purelin}(v) = v$$

3.6

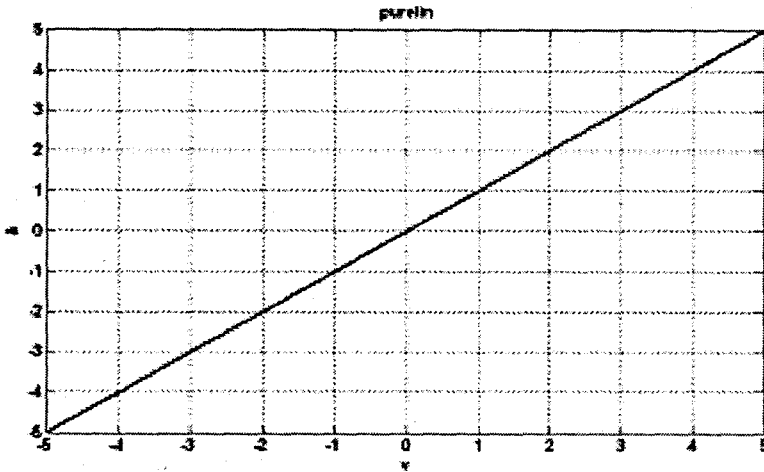


Figure 3. 10: Pure linear Activation function

The radial basis activation function (Equation 3.7) has a different neuron input structure to that of the other activation functions. The net input to the transfer function is the vector distance between its weight vector  $w$  and the input vector multiplied by the bias.

$$a = \text{radbas}(v) = e^{-v^2}$$

3.7

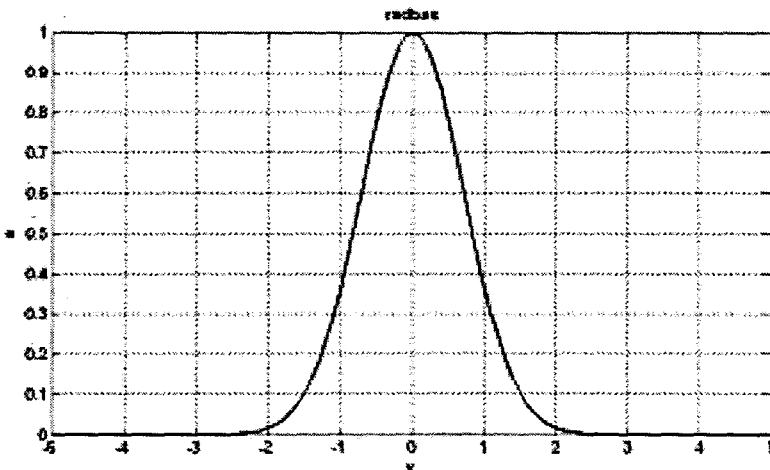


Figure 3. 11: Radial Basis Activation function

The radial basis function has a maximum of 1 when its input is zero (Figure 3.11). As the distance between the weight and input increases, the output decreases.

Therefore the radial basis function is said to act as a detector that produces 1 whenever the input and weight vectors are identical. The bias allows the sensitivity of the radial basis neuron to be adjusted.

There are other less popular activation functions such as the *satlin* (Saturating linear) function and the *poslin* (Positive linear) function (Burton, 2006). The success or failure of the learning process depends on whether the activation function is differentiable or not. Based on this requirement, the most popular nonlinear functions are the logistic sigmoid and the hyperbolic tangent.

### 3.6 Neural Network Training / Learning

Training is the way a NN learns. Learning or training algorithms are procedures for modifying the weights on the connections in a neural network structure. There are normally 3 types of learning algorithms (Ng, 1997):

- Supervised learning
- Unsupervised learning
- Reinforcement learning

For supervised learning, a set of inputs and correct outputs are used to train the network. The network then produces its own outputs. These outputs are compared with the correct outputs and the difference (error) is used to modify the weights. This type of learning is used in the thesis because of the nature of the available data.

Reinforcement learning is a special case of supervised learning where there is no set of inputs and correct outputs. The network is just told whether a produced output is 'good' or 'bad', according to a defined criterion.

Unsupervised learning or self-organizing is where a network develops its own classification rules by extracting information from the inputs presented to the network.

*Batch training* is where changes to the weights and biases are made based on the application of the entire input set of data vectors to the network.

*Incremental training* is where the changes to the weights and biases are made after the application of each individual input data vector. Incremental training is also referred to as on-line training or adaptive training.

Training algorithms use historical data (training data set) to automatically adjust the weights and thresholds in order to minimize the prediction error. The method used for

adjusting the weights on the connections is called a learning rule. There are many commonly used learning rules, but there are some that are proven to be efficient in solving certain types of problems.

### 3.7 The Perceptron Learning Rule

Rosenblatt's single layer perceptron is trained as follows:

1. Randomly initialise all the network weights.
2. Apply the inputs  $x$  and calculate the sum of each neuron  $S_j(k) = \sum_{i=1}^q x_i(k)w_{ij}(k)$ .
3. The outputs from each unit are:
4.  $O_j(k) = \begin{cases} 1 & S_j(k) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$
5. Compute the errors  $e_j(k) = O_{dj}(k) - O_j(k)$  where  $O_{dj}(k)$  is the known desired output value.
6. Update each weight as  $w_{ij}(k+1) = w_{ij}(k) + \eta x_i(k)e_j(k)$ , where  $\eta$  is the gradient procedure step (learning rate).
7. Repeat steps 2 to 4 until the errors reach a satisfactory level.

### 3.8 The Least Mean Square Algorithm

The Least mean square (LMS) learning algorithm is sometimes known as the delta rule or Widrow-Hoff learning rule. The correction of the weights is determined by:

$$\Delta w = \frac{\beta E x}{|x|^2} \quad 3.6$$

where  $E = y_d - y_m$  and where  $y_d$  is the desired output and  $y_m$  is the actual model output.

The error is based on the sum of the inputs to the unit  $S$ .

$$e_j(k) = O_{dj}(k) - S_j(k) \quad 3.7$$

The linear sum of the inputs  $S$  is passed through a tangent function that produces the output '+1' or '-1' depending on the polarity of the sum.

The LMS algorithm is based on the instantaneous values for the cost function, (Haykin, 1999):



$$\zeta(w) = \frac{1}{2}e^2(n) \quad 3.8$$

where  $e(n)$  is the error signal measured at time  $n$ . Differentiating  $\zeta(w)$  with respect to the weight vector  $w$  yields,

$$\frac{\partial \zeta(w)}{\partial w} = e(n) \frac{\partial e(n)}{\partial w} \quad 3.9$$

### 3.9 The Backpropagation Training Algorithm

Backpropagation is an example of supervised learning. In many applications neural networks are essentially function-mapping networks. The problem with mapping is to discover the relationship between the network inputs and target outputs. This relationship is in all likelihood probably nonlinear, as is the case with the activated sludge process. The backpropagation algorithm has been effectively utilized in many applications requiring nonlinear input-output mapping. With the ability to approximate any continuous nonlinear function, the backpropagation network has extraordinary mapping (forecasting) abilities (Rui and El-Keib 1995).

The term *backpropagation* refers to the manner in which the gradient is computed for nonlinear multilayer networks. Generalizing the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions created the Backpropagation algorithm.

The algorithm is based on steepest-descent (gradient descent) techniques extended to each of the layers in the network by the chain rule (Ng, 1997). The partial derivative of the error function with respect to the weights is computed as:

$$\nabla E(k) = \frac{\partial E(k)}{\partial W(k-1)} \quad 3.10$$

where  $W$  is the vector of the weights.

The error function is defined as:

$$E(k) = \frac{1}{2} [y_p(k) - y_m(k)]^2 \quad 3.11$$

where  $y_p$  is the plant output and  $e_m(k) = y_p(k) - y_m(k)$ . The constant  $\frac{1}{2}$  is chosen to facilitate the computation of a derivative for the cost function which is essential in the estimation of the parameters.

The objective is to minimize the error function  $E(k)$  by taking the error gradient with respect to the weight vector,  $W$  that is to be adapted. The weights are updated using:

$$W(k) = W(k-1) + \eta \frac{\partial E(k)}{\partial W(k-1)} \quad 3.12$$

where  $\eta$  is the learning rate and

$$\frac{\partial E(k)}{\partial W(k-1)} = -e_m(k) \frac{\partial y_m(k)}{\partial W(k-1)} = -e_m(k) \frac{\partial O_o(k)}{\partial W(k-1)}$$

It becomes evident that the success of the backpropagation technique depends on three factors: the learning rate  $\eta$ , the distance between the actual output and the predicted output, and the activation function. The learning rate determines how fast the network adjusts itself in response to the errors. If the learning rate is too high, the adjusted weights oscillate between points with similar heights on the error surface (Park, 1996) and the network does not converge to a minimum at all. If the learning rate is set too low, training is slow and the network is trapped in a local minimum (Figure 3.11). The learning rate should lie within the range  $0 \leq \eta \leq 1$ .

The backpropagation algorithm could be summarized as follows:

- A set of training inputs and corresponding target outputs are supplied to the network;
- The network calculates the error signals, and this is used to adjust the weights;
- After many forward and backward passes, the network error reaches a minimum on the training data;
- The network is tested on test data it has not seen before to measure the generalization ability of the network.

A single pass of the first two steps above is known as an epoch. Training usually lasts in some cases thousands of epochs until the network error is below a certain threshold. From this it is evident that modelling with neural networks is extremely time-consuming.

### 3.9.1 Limitations of the Backpropagation Training Algorithm

Limitations of the backpropagation training algorithm are given below:

- Long training times, therefore the speed of the convergence is relatively slow;
- Inappropriate training sets and learning strategy causes network paralysis. If the weights get too large, the changes in the weights become minimal (since for large weights, the derivative of the sigmoid is very small);
- No guarantee for convergence to a global minimum; sometimes the method can get stuck in a local minima (See Figure 3.12);

- There is no exact rule for setting the number of neurons and layers for the best performance; and
- Instability if the learning rate is too large.

Regardless of the above limitations to the backpropagation algorithm, it is still a very popular one especially in the control community. There are however numerous improvements to the common backpropagation algorithm. One improvement is adding a momentum term.

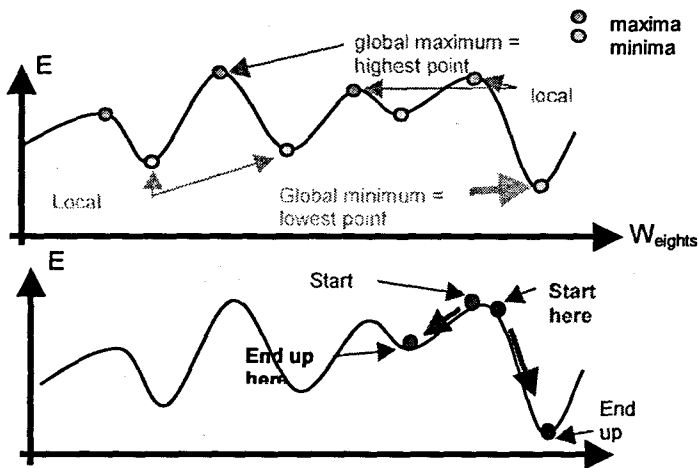


Figure 3.12: The Learning algorithm could get stuck in a local minima

### 3.9.2 Heuristic Improvements to the Backpropagation Algorithm

One method to avoid an error trajectory in the weight space being oscillatory is to add a momentum term (Papliński, 2004). Momentum is introduced to reduce the chance for oscillations for a given learning rate. Momentum allows a network to respond not only to the local gradient, but also to trends in the error surface (Demuth *et al.*, 2004). Momentum is a modification of the generalized delta rule by including a term in the weight updating equation (Equation 3.13). Including a momentum term can reduce the network training time and speed up the network convergence.

$$W(k) = W(k-1) + \left(-\eta \frac{\partial E(k)}{\partial W(k-1)}\right) + \alpha \Delta W(k-1) \quad 3.13$$

The momentum term parameter  $\alpha$  is selected between  $0 < \alpha < 1$ . If the momentum term is set to 0, the weight change is based only on the gradient. If the momentum term is set to 1, the new weight change is equal to the last weight change and the gradient is ignored. The gradient is calculated by summing the gradients calculated after each training example.

Such modification to the steepest descent learning law acts as a low-pass filter smoothing the error trajectory. As a result it is possible to apply a higher learning rate,  $\eta$ .

Modification to Equation 3.13 such as:

$$W(k) = W(k-1) + (-\eta \frac{\partial E(k)}{\partial W(k-1)}) + \alpha \Delta W(k-1) + \beta \Delta W(k-2) \quad 3.14$$

was proposed by Nagata *et al.* (1990) where  $\beta$  is a constant decided by the user. The claim was that this term reduces the possibility of the network being trapped in a local minimum. However, this is a repeat of the role of  $\alpha$ , and the advantage of adding  $\beta$  is not very clear (Ng, 1997).

Despite the attempts at improving the traditional backpropagation algorithm by adding momentum terms and other methods, there are other more efficient gradient methods such as the scaled conjugate gradient. This method is used in this work and is discussed in more detail in Chapter 5.

### 3.10 Generalization

It is rarely useful to have a NN simply memorize a set of data (function approximation), since numerous algorithms for table look-up can do memorization much more efficiently. Typically, it is required that the NN be able to perform accurately on new data, that is, to generalize.

Generalization can be defined as the ability of the network to make a hypothetical response to an input that has not been seen before, (Park, 1996) and make a reasonable response.

#### 3.10.1 Overfitting and underfitting

Overfitting or underfitting is essentially how well the NN make predictions for cases that are not in the training set. NNs, like other flexible nonlinear estimation methods such as kernel regression and smoothing splines, can suffer from either underfitting or overfitting. A network that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. A network that is too complex fits the noise also, not just the signal, leading to overfitting (Figure 3.13). Overfitting 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 NNs. Overfitting can also produce wild predictions in multilayer perceptrons even with noise-free data (NN FAQ, 2005).

In the following example in Figure 3.13, a NN has been trained to approximate a noisy sine function. The underlying sine function is shown by the dotted line, the noisy measurements are given by the '+' symbols, and the neural network response is given by the solid line. Clearly this network has overfit the data and does not generalize well (Demuth, Beale and Hagan, 2004).

There are generally three typical conditions necessary for good generalization:

The first condition is that the inputs to the network must contain sufficient information pertaining to the target, so that a mathematical function relating correct outputs to inputs with the desired degree of accuracy exists. A network cannot be expected to learn a non-existent function.

The second condition is that the function that is being approximated should be, in some sense, smooth. This means that a small change in the inputs should, most of the time, produce a small change in the outputs. For continuous inputs and targets, smoothness of the function implies continuity and restrictions on the first derivative over most of the input space. Some neural networks can learn discontinuities as long as the function consists of a finite number of continuous pieces. A nonlinear transformation of the input space can increase the smoothness of the function and improve generalization. A Boolean network with a small number of hidden units and small weights computes a "smoother" input-output function than a network with many hidden units and large weights.

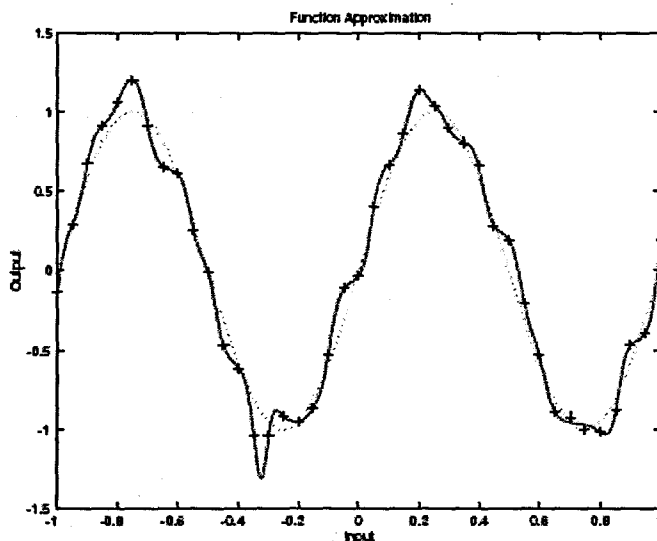


Figure 3. 13: An example of overfitting of a noisy sine wave (Demuth, et al, 2004)

The third requirement for good generalization is that the training cases be a sufficiently large and a representative subset of the set of all cases. This requirement

relates to the fact that there are two different types of generalization: interpolation and extrapolation. Interpolation applies to cases that are more or less surrounded by nearby training cases; everything else is extrapolation. In particular, cases that are outside the range of the training data require extrapolation. Cases inside large "holes" in the training data also effectively require extrapolation. Interpolation can often be done reliably, but extrapolation is notoriously unreliable. Therefore it is important to have sufficient training data to avoid the need for extrapolation. (NN FAQ, 2005).

If there are adequate samples for the training set, every case in the population is close to a sufficient number of training cases. Under these conditions and with proper training, a neural network does not generalize reliably well to the population.

If more information about the function, e.g. that the outputs should be linearly related to the inputs is available, one can often take advantage of this information by placing constraints on the network or by fitting a more specific model, such as a linear model, to improve generalization. Extrapolation is much more reliable in linear models than in flexible nonlinear models, although still not nearly as safe as interpolation. This information can be used to choose the training cases more efficiently. For example, with a linear model, training cases should be chosen at the outer limits of the input space instead of evenly distributing them throughout the input space.

Typically, the purpose of training is to make predictions for future cases in which only the inputs to the network are known. The result of conventional network training is a single set of weights that can be used to make such predictions. If a single-valued prediction is needed, one might use the mean of the predictive distribution, but the full predictive distribution also indicates how uncertain this prediction is.

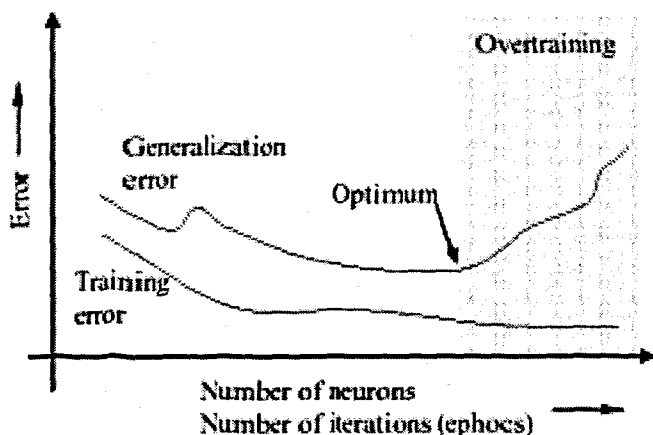


Figure 3. 14: Training error and generalization error

### 3.10.2 Early stopping

Another method for improving generalization is called *early stopping*. In this technique the available data is divided into three subsets. The first subset is the training set, which is used for computing the gradient and updating the network weights and biases. The error of this data set is minimized during training.

The second subset is the validation set. This set is used to determine the network performance on patterns which do not form part of the training set, but which determine when to terminate training. The error on the validation set is monitored during the training process. The validation error normally decreases during the initial phase of training, as does the training set error. However, when the network begins to overfit the data, the error on the validation set typically begins to rise. When the validation error increases for a specified number of iterations, the training is stopped, and the weights and biases at the minimum of the validation error are returned (Demuth, Beale and Hagan, 2004) (Figure 3.15).

The third subset is the test set which is not used during the training, but is used for obtaining an estimate of the true error rate of the network. It is also useful to plot the test set error during the training process (Figure 3.15). If the error in the test set reaches a minimum at a significantly different iteration number than the validation set error, this indicates a poor division of the data set.

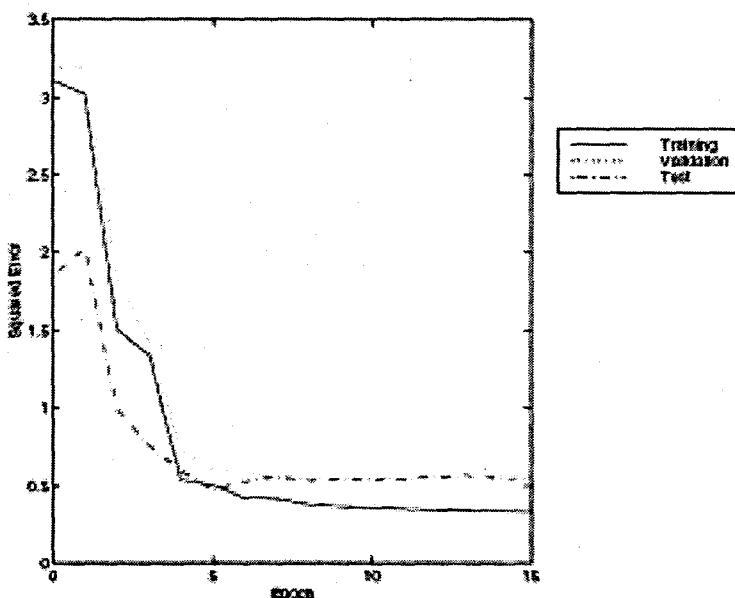


Figure 3. 15: Early Stopping Method plotting the training, validation and test sets

There are also many other important problems that are so difficult that a neural network is unable to learn them without memorizing the entire training set, such as:

- Predicting random or pseudo-random numbers.
- Factoring large integers.
- Determining whether a large integer is prime or composite.
- Decrypting anything encrypted by a good algorithm.

### 3.10.3 Regularization

Another method for improving the generalization ability of a NN is regularization. Statisticians use this technique to find the balance between the number of parameters and goodness of the fit by penalizing large models. The performance function is modified in such a way that the algorithm prunes the network by driving irrelevant estimates to zero (Rech, 2002). This involves modifying the performance function such as the sum of squares (mean square error) of the network error on the training set.

$$F = mse = \frac{1}{N} \sum_{i=1}^N (e_i)^2$$

It is possible to improve generalization by modifying the performance function by adding a term that consists of the mean of the sum of squares of the network weights and biases (Demuth, Beale and Hagan, 2004).

$$msereg = \gamma mse + (1 - \gamma) msw \quad 3.15$$

where  $\gamma$  is the performance ratio and  $msw = \frac{1}{n} \sum_{j=1}^n w_j^2$ .

Using this performance function causes the network to have smaller weights and biases, and this forces the network response to be smoother and less likely to overfit. The main advantage of using regularization is that even if the NN model is over-parameterized, the irrelevant parameter estimates are likely to be close to zero and the NN model behaves like a small network (Rech, 2002).

### 3.11 Neural Network Positive Characteristics

Neural networks are characterized by the following positive capabilities:

- To solve tasks where it is beneficial to use a machine, but where it is impractical to program responses to all possible outcomes (knowledge engineering bottleneck);
- Learning from limited examples – ‘human-like’ learning behaviour: neural networks are particularly suited to problems where the solution is complex and difficult to specify, but provides an abundance of data from which a response can be learnt (Tarassenko, 1998);
- Ability to generalize, i.e. map similar inputs to similar outputs: neural networks are able to interpolate from a previous learning experience. With careful design, a NN can be trained to give the correct response to data it has not previously encountered (generalization) (Tarassenko, 1998);
- Ability to map linear and nonlinear functions: nonlinear mapping often give neural networks the advantage of dealing with complex real-world problems;



- Computational efficiency: training a NN can be computationally intensive, but the computational requirements of a fully trained NN can be modest (Tarassenko, 1998). For larger problems speed is gained through parallel processing (Haykin, 1999);
- Robust in the presence of noise; and
- Multivariable capabilities.

### 3.12 Application Areas of Neural Networks

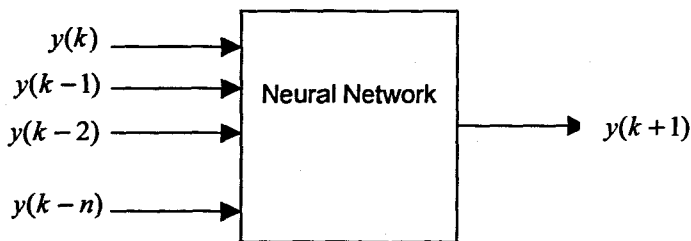
Some neural network application areas are listed in chapter 1, but also include (and are not limited to):

- Recognition and forecasting disturbances;
- Performing statistical quality control;
- Multiple model control for vehicles (Cavalletti, Ippoliti and Longhi, 2007);
- Adaptively tuning of process controllers; and
- Combining data from redundant sensors.

### 3.13 Neural Networks structures for use in prediction

The structure of a predictor underpins its ability to represent the dynamic properties of a statistically non-stationary discrete time input signal and hence its ability to predict some future value (Mandic and Chambers, 2001). Time whether continuous or discrete, is an important aspect of the neural network learning process (Elman, 1999). The most basic time-series predictions are made using lags from a single time series of as all predictors (Masters, 1993). For prediction applications a MLP NN can be compared to the FIR (finite impulse response) filter, or in terms of time series, moving average (MA).

Time series prediction has traditionally been performed by the use of linear parametric autoregressive (AR), moving average (MA) or autoregressive moving average (ARMA) models. The multilayer perceptron (MLP) is a static network, as it has no physical feedback connections. To transform the MLP into a dynamic network, time delays (memory) in the input data are introduced (Olssen and Newell, 1999), (Elman, 1990). The NN structure for prediction is shown in Figure 5.16 where the variable to be predicted is time delayed by a finite number of steps and used at the input. There are no physical feedback paths from the output back to the input in this structure, but it has been successfully applied in many applications such as time series prediction (Gao and Er, 2005), flood forecasting (Chang, Liang and Chen, 2001), short term load forecasting (Khan and Abraham, 2000), and time series forecasting (Malasri and Malasri, 2002) among others. It is used in the thesis to predict the time series data of the inflow disturbances. The other two structures are the Elman and RBF networks.



**Figure 3. 16:** Neural network structure for prediction applications where the inputs are time-delayed a number of steps

### 3.14 Summary of the Chapter

This chapter provides a brief theoretical introduction to neural networks and starts by examining the differences and similarities between the biological and artificial neuron. A brief historical background provides some perspective as to the developments and current research directions in the field of neural networks.

The development of the first perceptron, which ultimately led to the multilayer perceptron and many other different types of neural networks, are briefly discussed. The different types of activation functions such as the logarithmic sigmoid, hyperbolic tangent, pure linear, and hard limit (threshold binary) activations are investigated and proposed.

The manner in which a NN learns either by supervised or unsupervised learning is discussed. The gradient descent method of the backpropagation algorithm is examined and an improved algorithm with momentum is investigated. The use of a more efficient gradient method, the scaled conjugate gradient method is proposed. This is discussed in Chapter 5.

The next section that is a fundamental issue in the training and testing (validation) of NNs, is the generalisation ability of the NN. The definitions for overfitting and underfitting are tabled and two methods to improve generalisation are proposed. Overfitting occurs when the NN is in effect "over-trained". Underfitting occurs when there are insufficient neurons in the hidden layer to solve the given problem.

The first method to improve generalisation is the early stopping method where the data is divided in 3 sets, namely the training, validation and test sets. The error is monitored on the training and validation sets to avoid overfitting the data. The second method for improved generalisation is regularisation, where a modified performance

function is introduced. This ensures that the weights and biases in the network remain small.

Positive NN characteristics and application areas are described. Finally a brief look is taken at structures used in prediction or forecasting applications, where the output is delayed a finite number of steps and used at the input without any physical connections.

The success of a particular application may lie in the quality and quantity of the data presented to the neural network. Chapter 4 examines the importance of pre-processing the input data to a NN. The plant and weather data sets are discussed, and various techniques for transforming the data are investigated.

## CHAPTER FOUR

### DATA PRE-PROCESSING FOR SUPERVISED LEARNING

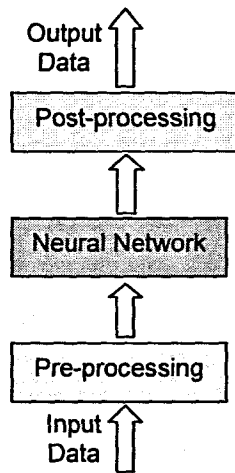
#### 4.1 Introduction

Our world is increasingly becoming more and more data-driven. Data in various formats surround us in almost sphere of our daily lives. This data is usually raw and it must be analyzed, processed, and maybe converted into another more meaningful format in order to inform, instruct and aid in our understanding and decision-making.

Neural networks can sometimes produce startlingly excellent results and offer an attractive paradigm for the design and analysis of adaptive intelligent systems for a broad range of applications (Garvey, 1997). They also handle seemingly impossible problems. From this point of view, neural network practitioners sometimes think that they can throw anything at a neural network and it will handle it (Masters, 1993). Neural networks when used wisely perform at least as well if not better in certain instances, than traditional methods. Many factors however contribute to the success of a neural network for a given problem. The representation and quality of the instance data is first and foremost (Kotsiantis *et al.*, 2006). If data is irrelevant, redundant, noisy or unreliable, then knowledge discovery and the learning of the relationships between cause and effect during the training phase are more difficult.

Data preparation and pre-processing although it takes a lot of time, is of utmost importance to the success of the neural network performance. Data preprocessing includes data cleaning, normalization, transformation, feature extraction and selection, etc. After pre-processing, the data is presented to the neural network, after which post-processing is performed to return the data in its original form (Figure 4.1). The network performance is then calculated on the post-processed data.

There is no one single recipe of data pre-processing algorithms to follow in order to guarantee the best performance for each data set. This chapter will present some well-known pre-processing steps that can be considered and applied so that the best performance is achieved for a given data set. These steps are applied to the data used in the thesis where the NNs are built using COD, TKN, flow, weather and month of the year as input data, and COD, TKN and flow as output data.



**Figure 4. 1: Block diagram illustration of the use of data pre-processing and post-processing with a neural network (Adapted from Bishop (1995))**

## 4.2 Data description

The plant data is obtained from the Athlone Wastewater Treatment Plant (Figure 4.2) in Cape Town, South Africa. The Athlone plant provides treatment of domestic and industrial waste. The plant has been designed with the following specifications:

- Plant capacity: 105 Ml;
- COD: 1125 mg/l;
- Suspended Solids (SS): 400 – 450 mg/l; and
- Sludge Age: 13 days

The effluent standards for the plant according to the Department of Water and Forestry is:

- Suspended solids: 25 mg/l
- NH<sub>3</sub> as N: 10 mg/l
- COD: 130 mg/l
- Ecoli: 1000/100 ml

The plant configuration is based on the UCT process model for Nitrogen and Phosphorous removal.

The influent variables of interest chosen are the Chemical Oxygen Demand (COD), Total Kjeldahl Nitrogen (TKN), and the influent flow rate. These variables are measured at the influent (input) of the plant and are off-line measurements except for the flow rate. These particular external disturbances are selected as they play a vital role in the activated sludge process (see Chapter 2), and are used in the mathematical models describing the activated sludge process. The historical data for the COD, TKN and Flow rate are sampled from July 1992 up until January 2005. Although data for COD and TKN are available from January 1985, the influent flow

rate data is only available from July 1992 onwards. The plant data is converted from the MS Excel format into the MySQL database format.



**Figure 4. 2: An aerial view of the Athlone Wastewater Treatment Plant.**

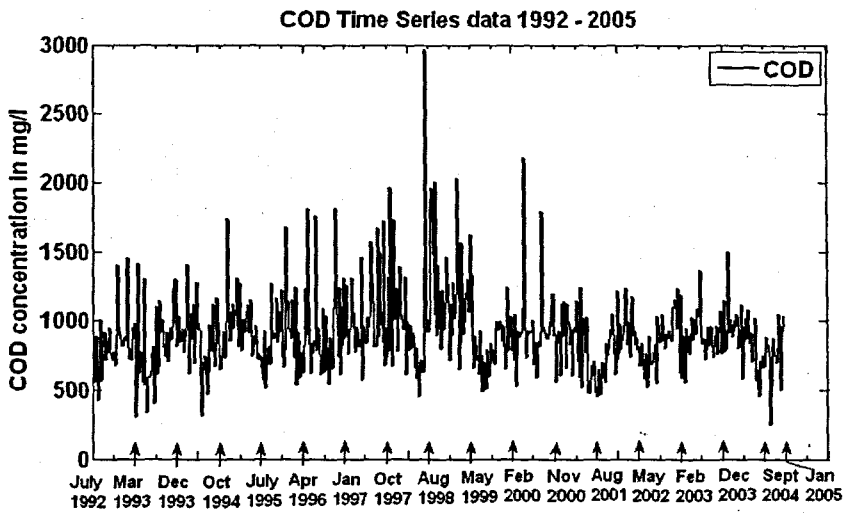
The weather data is obtained from the South African Weather Service Centre based at the Cape Town International Airport. This is the historical weather data for the Cape Town region between January 1985 and January 2005. The weather data used is from the same time period as the plant data, July 1992 up to January 2005. The data is converted from the ASCII text format to MS Excel and then transferred to the MySQL database. The variables chosen are minimum temperature, maximum temperature, rainfall, and wind speed.

The number of data points for both plant and weather data used for the training and test data sets together are 658 points. All the time series data for the specified time periods are displayed in Figures 4.3 to 4.9. The cyclic trends in the data variables are fairly evident upon visual examination of the raw plant and weather data. This is the case for particularly minimum and maximum temperature and the flow rate where changes occur as the seasons change. The Athlone WWT plant data are sampled daily but averaged to a weekly value. The daily weather data is also averaged to a

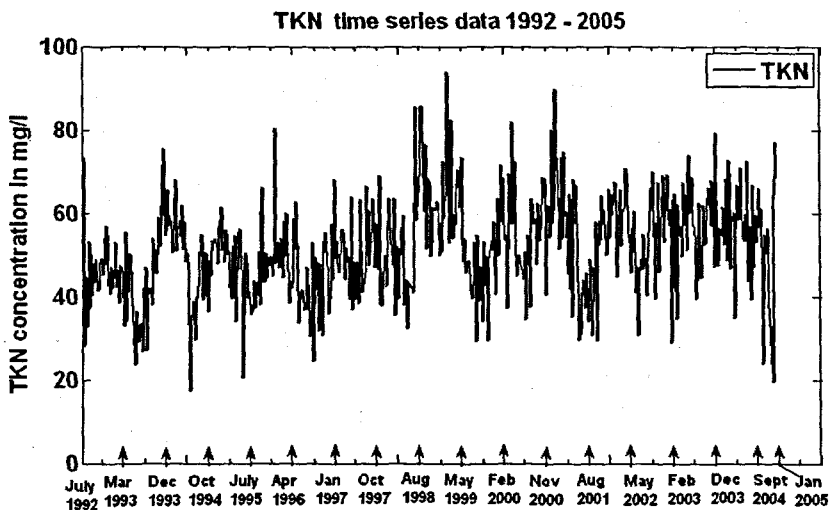
weekly sample value. The description of the variables, units of measure, range of the data, together with the mean and standard deviation of the plant and weather data are presented in Table 4.1. The monthly data representing the time aspect is another input to the NN and is reflected in Table 4.5 below.

**Table 4. 1: The information for the process variables for the Athlone wastewater treatment plant and weather data**

Variable	Description and unit of measure	Min	Max	Mean	Std Dev
COD	Chemical Oxygen Demand – mg/l	251.00	2030.00	898.09	249.14
TKN	Total Kjeldahl Nitrogen – mg/l	17.40	85.60	51.45	11.22
Flow rate	Influent flow rate to plant – m <sup>3</sup> /d	49.00	181.00	91.36	22.36
Maxtemp	Maximum ambient temperature - °C	14.29	32.64	22.36	3.98
Mintemp	Minimum ambient temperature - °C	3.17	18.59	11.79	3.50
Rain	Rainfall - mm/hr	0.00	21.44	1.42	2.33
Wind	Wind speed – km/hr	6.30	24.73	13.70	2.56



**Figure 4. 3: Chemical Oxygen Demand (COD) time series data: 1992- 2005**



**Figure 4. 4: Total Kjeldahl Nitrogen (TKN) time series data: 1992- 2005**

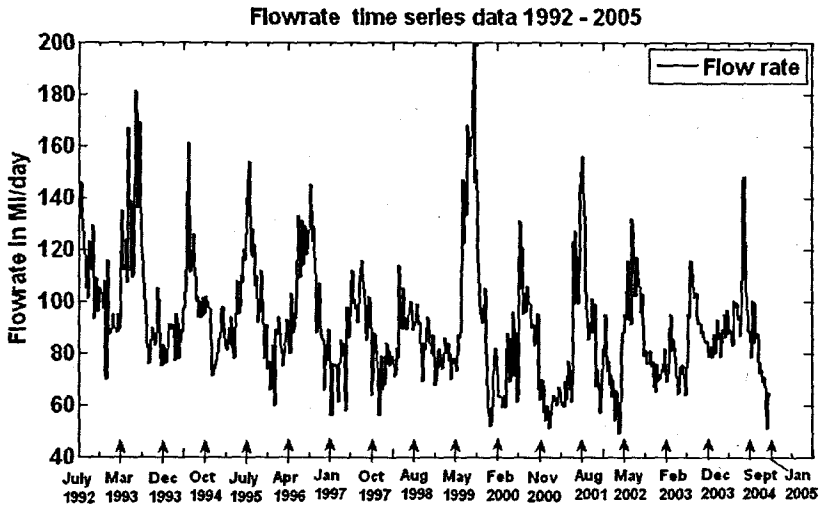


Figure 4. 5: Flow rate time series data: 1992- 2005

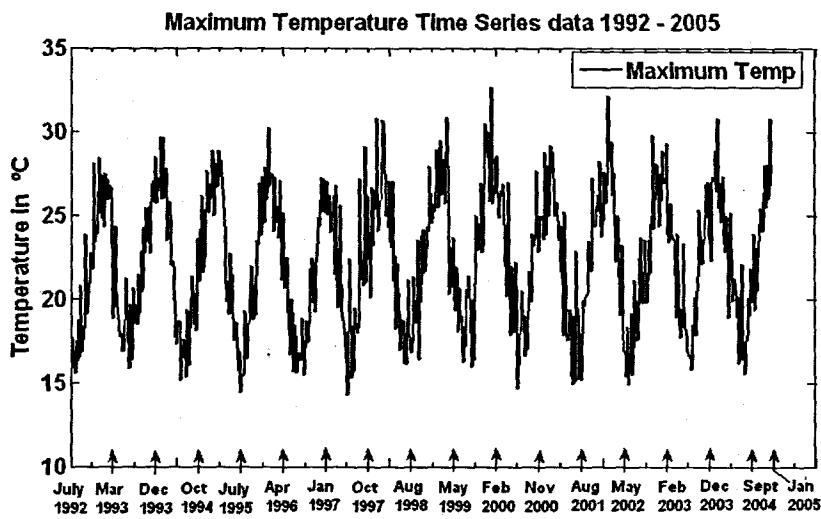


Figure 4. 6: Maximum Temperature time series data: 1992- 2005

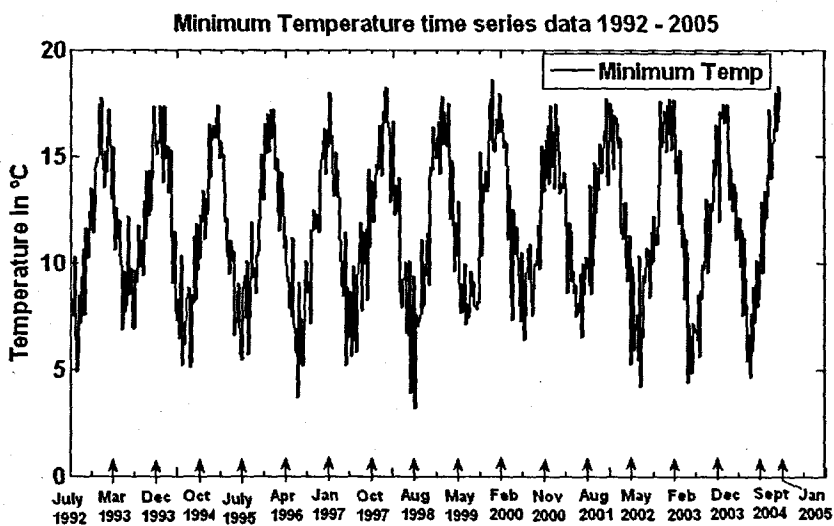


Figure 4. 7: Minimum Temperature time series data: 1992- 2005



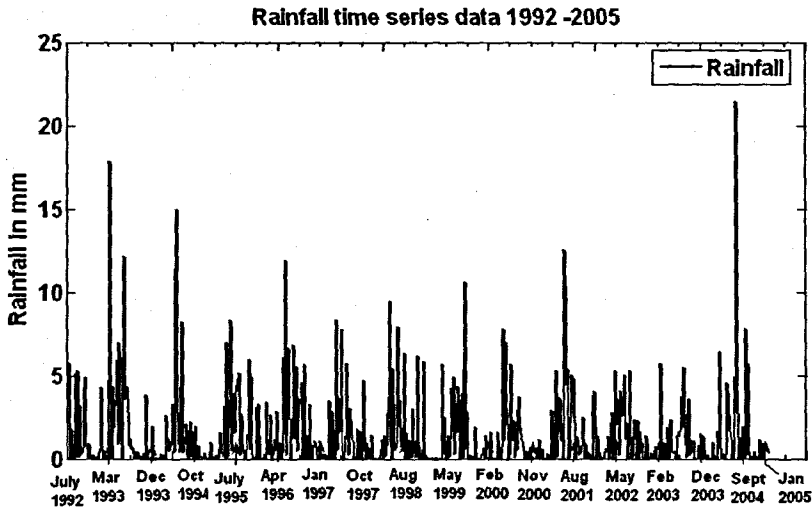


Figure 4. 8: Rainfall time series data: 1992- 2005

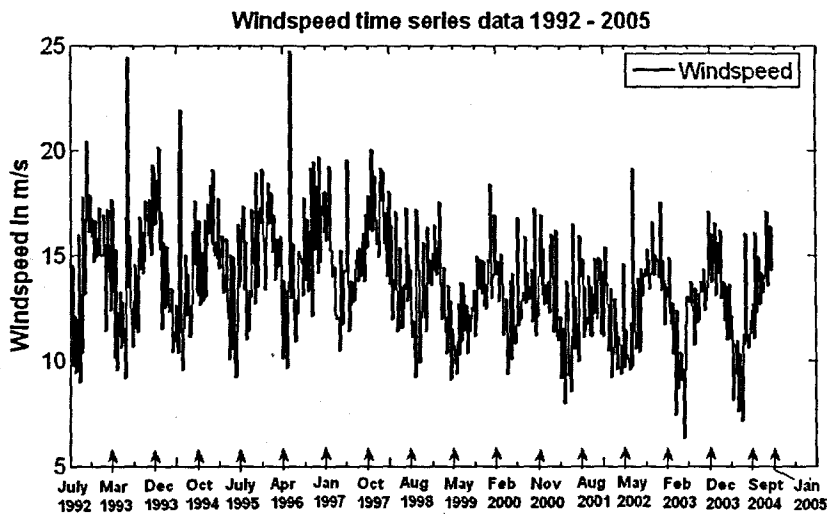


Figure 4. 9: Wind Speed time series data: 1992- 2005

### 4.3. Data selection

Neural networks are data-driven and often require large amounts of training data examples (Barnett 1999). The first step in selecting data is the data specification and collection, where variables of interest are identified and collected. The next step in the process would be the analysis and examination of the collected data. These two steps are usually performed iteratively and in parallel with each other (Casey, 2004). Finding good inputs for a NN and collecting enough training data often take far more time and effort than training the network.

Training data is essential to the success of neural computing applications development, and the availability and quality of data should be confirmed at the feasibility stage. The range of data available to train the network should be representative of the data that the network would have to process after it is trained.

There should be examples of extreme input conditions so as to ensure that the network will perform correctly at these extremes (Tarassenko, 1998).

If data is not available immediately the feasibility of collecting it has to be assessed, as there are usually practical problems and cost implications associated with collecting data (Tarassenko, 1998). Some expert knowledge regarding the types of data variables or different processes involved may assist in the selection of the input variables to the NN.

The influent wastewater disturbance variables of COD, TKN and flow were selected as an abundance of reliable data was available and these particular variables play an important role in the activated sludge process. For the other variables of interest there was not enough data available

#### **4.4 Data preprocessing**

Neural networks very rarely operate directly on the raw data, although this is possible. The disadvantage of using raw data values is that the training time for the neural network would be significantly longer as the various variables have very different ranges. Data has to be pre-processed before it is inputted to a neural network because:

- Neural networks learn faster and perform better if the input variables are pre-processed before it is used to train the network;
- Neural networks are able to interpolate well, but perform poorly when extrapolating data;
- The network should not be trained on one problem and tested on an entirely different problem. Here problems with generalization may occur.

Data pre-processing can have a significant effect on the generalization performance of a supervised neural network. Training a NN until the error converges to a minimum does not mean that the NN will respond positively to the application of a test data set. What usually occurs is that the NN has learned the training data set as well as the underlying noise and is not able to generalise. The elimination of noise from data is one of the most difficult problems in machine learning (Teng, 1999).

##### **4.4.1 Missing data**

Plant data is most times not very reliable and many problems can occur which can affect the reliability or integrity of the data. One of the most common problems is that of missing data. One of the many reasons for this could be as a result of operator failure to record data, but the more likely explanation could be instrumentation equipment failure. This is especially true of on-line sensors where all but the very

simplest of devices are prone to repeated failure (Arthur, 1982). There are various methods proposed to deal with the problem of missing data.

The first option should be to find the missing data. This step is almost always overlooked. If data is truly missing it may be worthwhile to enquire as to why the data is missing.

A single neuron can also be dedicated to be a "missing value" indicator. The neural network is given some value in place of the missing data. If the variable is nominal, the mode (most common value) could be used. If the data is ordinal the median is appropriate. For interval or ratio data, the mean is probably the best. The important point to consider is not to give it an oddball value that can be a misguided teacher, particularly for supervised learning (Masters, 1993).

Tarassenko (1998) proposes three strategies to deal with missing data. The first method consists of replacing the missing data value by its mean (Olssen and Newell, 1999) or median across the training set. The other method is to estimate the missing value for an  $n$ -dimensional input vector from knowledge of the other  $n-1$  input variables. The last method uses either a linear model or a NN network to predict the  $n$ th value given the set  $(n-1)$ -dimensional vectors as inputs. Olssen and Newell (1999) also propose using the previous good value, constant trend (replace using linear regression of points on either side of the missing data), or more complex relations can be used to interpolate (polynomials or splines).

The first approach that is used is that the missing data values are replaced by the average (mean) value for the particular variable for the entire year. The problem with this approach occurs if there are consecutive missing data values, as is the case with the Athlone plant data set. This means that for the same inputs the NN will have different output values. This can influence the response of the network to an entirely different data set (test data).

The approach adopted eventually is to use a linear interpolation method to replace the missing data values in the Athlone plant data set. There is no visible improvement in the predictive performance of the neural network for the case of using the mean value or the linear interpolation method in order to replace the missing data values. This could indicate that either method would work equally well for this data set.

The plant data for COD has 12 missing data points, for TKN there are 28 missing data values and for the influent flow rate there are 12 missing data points. In a few instances the missing data points were consecutive, but this did not extend to more than 3 consecutive missing points. There were 5 points at which all influent disturbance variables had missing data at the same moment in time. This could be due to the fact that no data was available either because of maintenance schedules, or no readings were captured. These influent data variables (influent disturbances) are the only ones where the linear interpolation method is applied. Missing data in the weather data set are replaced by using the monthly mean value for the particular variable.

After the problem of the missing data points have been dealt with, careful attention has to be paid as to which variables are selected as inputs to the NN. The first aspect to consider and investigate is referred to as the "curse of dimensionality".

#### 4.4.2 The curse of dimensionality

The number of inputs to a NN determines the network size. Therefore, as the number of inputs increase, the free parameters (weights) within that particular NN will also increase. Haykin (1999) proposes a "rule of thumb" for determining the number of data points that is required based on the number of inputs  $I$  to the NN:

$$m \geq I \left( \frac{W}{\varepsilon} \ln \left( \frac{N}{\varepsilon} \right) \right) \quad 4.1$$

where  $m$  is the number of training vectors to achieve a worst case generalisation error,  $\varepsilon$ , for the network with  $W$  weights and  $N$  nodes.

The "curse of dimensionality" (Bellman 1961) refers to the exponential growth of hyper-volume as a function of the input dimension. Many NNs can be thought of mapping from an input space to an output space. Every part of the input space needs to be represented by the NN in order to know how that part of the space should be mapped. Covering the input space takes resources, and the amount of resources needed is proportional to the hyper-volume of the input space. Neural networks with lots of unnecessary inputs generally perform relatively poorly. This is caused by the fact that the dimension of the input space is high, and the network uses almost all its resources to represent irrelevant portions of the input space. The Radial Basis Function NN is one type of NN that is particularly prone to this problem.

A partial remedy is to pre-process the input in the right way, for example by scaling the components according to their "importance". However, even if a network algorithm

were able to focus on important portions of the input space, the higher the dimensionality of the input space, more data would be needed to filter out irrelevant information.

A priori information can help with the curse of dimensionality. Careful feature selection and scaling of the inputs affects the severity of the problem, as well as the selection of the type of neural network model (Mandic and Chambers, 2001). One method used to aid in the selection of the inputs and to eliminate unnecessary inputs to the NN, is correlation.

### 4.4.3 Correlation

Several statistical methods are available for determining the significance of, and the relationship between variables. The problem is that many of these methods are linear techniques, where neural networks on the other hand are usually based on non-linear models. However certain linear methods such as correlation can be used to pre-process data before it is used as inputs to the NN.

Correlation can assist in determining which variables are closely correlated and removing any unnecessary inputs. Calculating the correlation coefficients for two variables will give an indication as to the strength of the relationship between them. Correlation measures the degree to which two variables move together. This is usually indicated by a value ranging from  $-1.0$  to  $+1.0$ :

- ◆ 0 indicates no correlation;
- ◆ 1.0 indicates a high degree of positive correlation (when  $x$  is high  $y$  is high); and
- ◆  $-1.0$  indicates a high negative correlation (when  $x$  is high  $y$  is low).

If two variables are highly correlated, they can be combined into one single input variable, or one can be discarded altogether. This can help reduce the network size. If we assume that  $X$  is the independent variable matrix and  $Y$  is the dependent variable matrix, then the correlation coefficients can be calculated as the covariance of the two variables divided by the product of their standard deviations:

$$\rho_{x,y} = \frac{Cov(X,Y)}{\sigma_x \cdot \sigma_y} \quad 4.2$$

where  $Cov(X,Y)$  is the covariance matrix of  $X$  and  $Y$  and is given as,

$$Cov(X,Y) = \frac{1}{n} \sum_{i=1}^n [(x - \mu_x) - (y - \mu_y)] \quad 4.3$$

and  $\sigma_x \cdot \sigma_y$  is the product of the standard deviations. The independent variable's standard deviation is denoted by:

$$\sigma_x = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad 4.4$$

where  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$  and the dependent variable's standard deviation is:

$$\sigma_y = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (y_i - \bar{y})^2} \quad 4.5$$

where  $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$

Two cases of correlation are considered:

- only between the input variables of the NN; and
- between the input and output variable, paying attention that one of the input variables is an output variable also, but considered at different instances in time.

The correlation coefficients for the plant process variables and weather data at the NN inputs are presented in Table 4.2 below where the dependent variable is the row (d) and the independent variable is the column (i):

**Table 4. 2: The correlation coefficients for the Athlone wastewater treatment plant data and weather data at the NN input**

i \ d	TKN	Flow	Maxtemp	Mintemp	Rain	Wind
COD	0.573	-0.319	0.2816	0.176	-0.278	0.053
TKN	1	-0.446	0.334	0.227	-0.2912	-0.135
Flow		1	-0.597	-0.521	0.396	-0.027
Maxtemp			1	0.854	-0.521	0.239
Mintemp				1	-0.313	0.442
Rain					1	0.144
Wind						1

The maximum temperature and minimum temperature have a correlation coefficient of about 0.85, which is close to 1. One of these two variables can therefore be eliminated. The maximum temperature correlation coefficient for TKN and COD are higher than that of the minimum temperature. This could indicate that the maximum temperature can therefore be discarded as an input.

The method of correlation discussed above is discussed and applied in the work of Khan and Ondrůšek (2000), where the input variables are correlated with one another and those that are highly correlated are eliminated. However another method of correlation is investigated where calculation of the correlation coefficient of the NN input with the NN output is done in order to determine the relationship between the input and output. The results of these correlation coefficients are displayed in Table 4.3 where three NNs are described with the influent disturbance variables of COD, TKN and Flow rate as the outputs of the NNs. Different combinations of inputs to

these NNs are examined. The type and the number of the inputs form different structures of the networks, which are called *models* in this thesis. There are 13 models for every one of the network types. Although a model number is presented in the table, this does not include the complete input set. The entire set of inputs for a specific model number is presented in detail in Chapter 5. The initial idea was to perform a correlation of the entire input vector with the output vector, but this is not possible as correlation can only be performed when the two vectors are of the same size. Therefore the individual inputs are correlated with the NN output. After a cursory examination of Table 4.3 it is fairly obvious that the NN with influent flow rate at the output has the highest correlation coefficients for almost all input variations. The first case of correlation is employed in this work although the latter is relevant when looking at the predictive performance of the NN.

**Table 4. 3: The correlation coefficients for the correlation between the inputs and outputs for the three NNs for the different influent variables of COD, TKN and influent Flow rate**

Input Model number	Input variable	Output	Correlation Coefficient
Model 1	COD	COD(k+1)	0.2820
Model 2	COD(k-1)	COD(k+1)	0.2482
Model 3	COD(k-2)	COD(k+1)	0.2274
Model 4	COD(k-3)	COD(k+1)	0.2699
Model 5	TKN	COD(k+1)	0.2618
Model 6	FLOW	COD(k+1)	-0.2579
Model 7	Month (2 input neurons)	COD(k+1)	0.1463 0.1832
Model 8	Minimum Temperature	COD(k+1)	0.1602
Model 9	Rain	COD(k+1)	-0.2088
Model 10	Wind	COD(k+1)	0.1152
Model 1	TKN	TKN(k+1)	0.4973
Model 2	TKN(k-1)	TKN(k+1)	0.4555
Model 3	TKN(k-2)	TKN(k+1)	0.3955
Model 4	TKN(k-3)	TKN(k+1)	0.3468
Model 5	COD	TKN(k+1)	0.2262
Model 6	FLOW	TKN(k+1)	-0.4198
Model 7	Month (2 input neurons)	TKN(k+1)	0.2137 0.2006
Model 8	Minimum Temperature	TKN(k+1)	0.2369
Model 9	Rain	TKN(k+1)	-0.2434
Model 10	Wind	TKN(k+1)	-0.0520
Model 1	FLOW	FLOW(k+1)	0.8587
Model 2	FLOW(k-1)	FLOW(k+1)	0.7575
Model 3	FLOW(k-2)	FLOW(k+1)	0.6894
Model 4	FLOW(k-3)	FLOW(k+1)	0.6299
Model 5	COD	FLOW(k+1)	-0.3213
Model 6	TKN	FLOW(k+1)	-0.4417
Model 7	Month (2 input neurons)	FLOW(k+1)	-0.4990 -0.3774
Model 8	Minimum Temperature	FLOW(k+1)	-0.5342
Model 9	Rain	FLOW(k+1)	0.4429
Model 10	Wind	FLOW(k+1)	-0.0478

Another method to investigate the relationship between input variables and for the elimination of any unnecessary inputs is principal component analysis (PCA).

#### 4.4.4 Principal Component Analysis

When the input data is high dimensional and no obvious features can be extracted from it, the input dimensionality must still be reduced in order to limit the number of free parameters in the network. The free parameters are the total number of biases and weights within the network and are determined mainly by the number of inputs (Tarassenko, 1998). Sometimes the situation arises that the number of inputs to the NN is large, but the components of the inputs in the input vector are highly correlated (redundant). It is useful in this situation to reduce the number of the inputs to the NN. A suitable technique for performing this dimensionality reduction is principal component analysis (PCA). This technique has three effects: it orthogonalizes the components of the input vector (so that they are uncorrelated with each other); it orders the resulting orthogonal components (principal components) so that those with the largest variation come first; and it eliminates those components that contribute the least to the variation in the data set (Demuth, Beale and Hagan, 2004).

In the initial phases of this study, PCA is performed on the data inputs to the NN using the plant and weather data, but there is no reduction of the dimensions to the input vector. The need for PCA in this study is therefore not required.

#### 4.4.5 Normalization (scaling)

Neural networks don't cope very well with a big range of values. If scaled data is presented to the network, the weights can remain in small, similar predictable ranges. Neural networks can be trained by using raw data as inputs, but the training time will be considerably longer.

Some typical normalization methods include the following:

$$\text{Normalized value} = \frac{\text{actual value} - \text{min imum value}}{\text{max imum value} - \text{min imum value}} \quad 4.6$$

$$\text{Normalized value} = \frac{\text{actual value}}{\text{sum of the weekly values}} \quad 4.7$$

$$\text{Normalized value} = \frac{\text{actual value}}{\text{max imum value of the week}} \quad 4.8$$

$$\text{Normalized value} = \frac{\text{actual value} - \text{average value}}{\text{max imum value} - \text{average value}} \quad 4.9$$

One of the main reasons for normalizing is to equalize the importance of variables. The scaling of the data is done in order to improve interpretability of network weights.



Also the uniform scaling equalizes initially the importance of variables (Khan and Ondrušek, 2000).

The normalization method chosen is represented by equation 4.8. This particular equation in essence scales the data so that the maximum normalized value would be  $\leq 1$ .

#### 4.4.6 Trends

Statistical and numerical measures of trends in historical data can prove to be fairly valuable. The most salient information can be extracted from the historical data and the NN is thus presented with only a summary measure. The elimination of trend and seasonal variation is a very complex subject, and is discussed practically by Kendall and Stuart, vol 3 (1976). The reasons, which might be considered before eliminating large-scale trends, according to Masters (1993) include:

- The NN will assume that the trend is important information and will attempt to use that information for prediction. By eliminating trends that are easily predicted and added back in later, the NN is free to concentrate on finer details.
- NN are as stated previously inherently nonlinear. This is one of major advantages of the NN, but much of the activity takes place in fairly linear regions of the neuron's activation functions. Thus the large variations are superimposed on important subtle information. In the learning process these smaller variation in the information may be scaled down or even neglected.

#### 4.4.7 Seasonality

Some data sets have a seasonal or time-lagged aspect associated with it, and there is often a natural cyclical phenomenon present. If a particular variable has this form, it may not be appropriate to compare values if they are taken from different phases of a cycle (e.g. summer, winter, autumn or spring).

To solve this problem of seasonality, many analysts compare quarterly data on a lagged basis (Casey, 2004). This lagging of the data simply compares the current period's data with the previous corresponding periods in the cycle. Fast Fourier transforms or wavelets are used as a more sophisticated curve fitting technique, for higher frequency time series.

The wastewater variables most definitely have a seasonality aspect associated with it. This is evident as more people drink more fluids during the summer months and

therefore use the toilet more frequently. Also during public holidays the concentrations of the variables to the plant are higher than on days without any holidays. The seasonality and circularity problem (below) is solved using sine and cosine pairs to represent the month in which the sample is taken, thus taking the season into account as well.

#### 4.4.8 Circular discontinuity

Certain variables such as dates in the year may be fundamentally circular. With these types of variables a problem arises when the date passes from 31<sup>st</sup> December to 1<sup>st</sup> January (day 365 to day 1). The neural network may interpret this to mean that these days are extremely far apart, and may even classify day 365 as being more important than day 1.

If the position of a rotating object is defined as being an angle measured from 0 to 360 degrees, a serious problem occurs when the object passes from 360 to 0 degrees. If a single neuron is used to decode this value, it is found that two extremely close values, 359 and 1 degrees are represented by two extremely different activations. Therefore variables with circular discontinuity are very difficult to learn (Masters, 1993).

The same problem arises if the months of the year are used as inputs to a neural network (in this particular project). Month 1 may be deemed less important by the network than say, month 12.

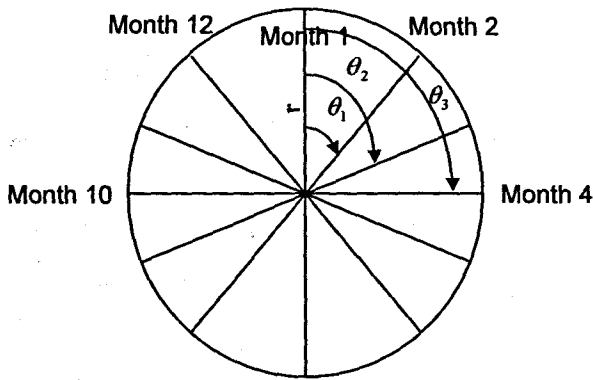
One proposed method to get around this type of problem, is to present the data in a different form to the network. The monthly data was initially presented as a 4-bit binary value as an input to the network. This is illustrated in Table 4.3.

**Table 4. 4: The monthly data represented as a 4-bit binary number instead of a decimal value**

Month	4-bit Binary representation			
January	0	0	0	1
February	0	0	1	0
March	0	0	1	1
April	0	1	0	0
May	0	1	0	1
June	0	1	1	0
July	0	1	1	1
August	1	0	0	0
September	1	0	0	1
October	1	0	1	0
November	1	0	1	1
December	1	1	0	0

The above approach was investigated and implemented without much success in terms of the network response to different inputs. This approach does not really address the circularity problem, but does indeed transform the way that the data is presented.

Another more practical approach to this type of problem is to represent the data as an angle around the circumference of a circle. The circle is divided into 12 sectors with angles  $\theta = \frac{360^\circ}{12} = 30^\circ$ . Month 1 is represented with angle  $\theta_1 = 30^\circ$ . Month 2 is represented with angle  $\theta_2$  equal to  $2\theta = 60^\circ$ . Then the sine and cosine of the angle is representative of the month. This is illustrated in Figure 4.10 and the result is presented in Table 4.5. As stated previously, the daily plant and weather data is averaged to a weekly sample value. The month in which the data is sampled is presented as the time varying variable.



**Figure 4. 10: Circularity discontinuity: Diagram illustrating the approach used in presenting data of a circular nature to the neural network**

The angle is therefore calculated as being:

$$\theta_m = \frac{360^\circ}{12} \cdot m = 30^\circ m \tag{4.10}$$

where  $m = \overline{1,12}$ . Therefore every month is represented by two inputs to the neural network. From equation 4.10 the sine and cosine pairs are determined. The values for the month of year are represented by two inputs and these are indicated in Table 4.5 below.

**Table 4. 5: The monthly data represented as a sine / cosine pair**

Month	No.	Angle	Sine	Cosine
January	1	30°	0.5	0.866
February	2	60°	0.866	0.5
March	3	90°	1	0
April	4	120°	0.866	-0.5
May	5	150°	0.5	-0.866
June	6	180°	0	-1
July	7	210°	-0.5	-0.866
August	8	240°	-0.866	-0.5
September	9	270°	-1	0
October	10	300°	-0.866	0.5
November	11	330°	-0.5	0.866
December	12	360°	0	1

The best way to handle circular variables is to encode them using two or three neurons. Usually two are sufficient (Masters, 1993).

#### **4.4.9 Outliers and data rejection**

Data that appear to be very far away from the normal data distribution may be classified as being outliers. In certain instances however, this outlying value may be correct and is a natural product of the variable's distribution (Masters, 1993).

There are some important considerations to bear in mind. Firstly, in any given data set there will be outliers. Also some abnormality in the data is normal. A rule of thumb to follow is that no data point should be rejected unless it is really far out. Masters (1993) remarks, "Unless we can satisfy ourselves beyond a reasonable doubt that the observed value is erroneous, we should endeavour to find an explanation for the value and perhaps modify our collection procedure so as to include its relatives".

Another important fact to consider is that when a learning algorithm that minimizes the mean error across the training set is used, rare outliers will have a comparatively small impact on the learned weights. This is due to the fact that it is first "tamed" by the squashing (activation) function, and then swamped out by the mass of correct data.

Discarding any data from a relevant variable should only be considered as a last resort. The recommended approach for data rejection is to plot a histogram of the data distribution and then carefully scrutinize the data which appear as outliers. An example of a histogram plot for the Chemical Oxygen Demand (COD) variable is given in Figure 4.12. Possible outliers could be to the right of the histogram plot (close to 3000 mg/l) as the rest of the distribution of data is in a similar region. The

histogram plots for the rest of the inputs are presented from Figure 4.13 up to and including Figure 4.19.

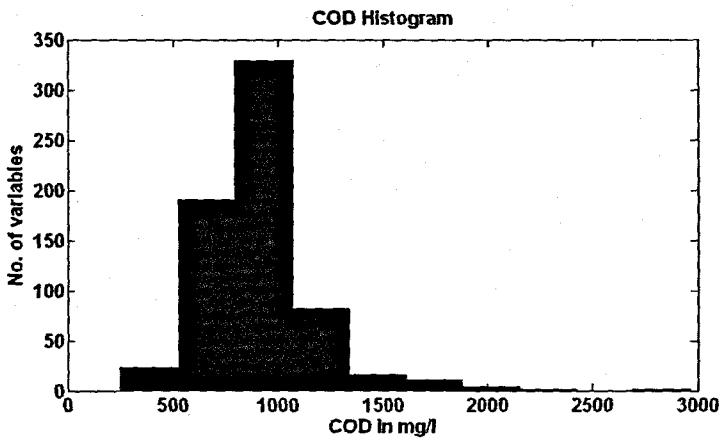


Figure 4. 11: Histogram plots are useful for data rejection rather than automated outlier rejection algorithms. The above figure shows the histogram plot of the COD variable

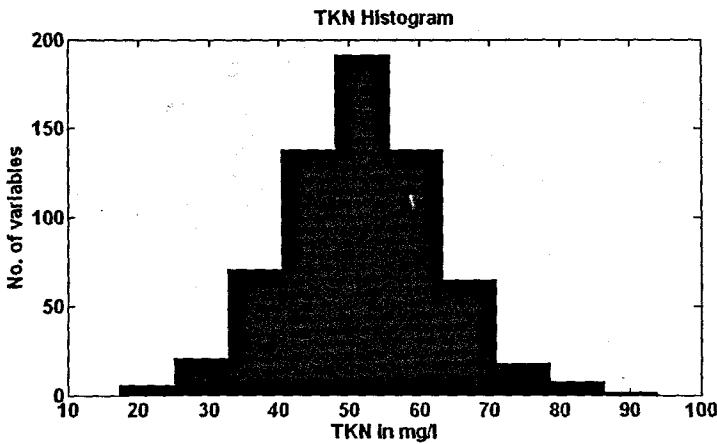


Figure 4. 12: Histogram plot of the TKN

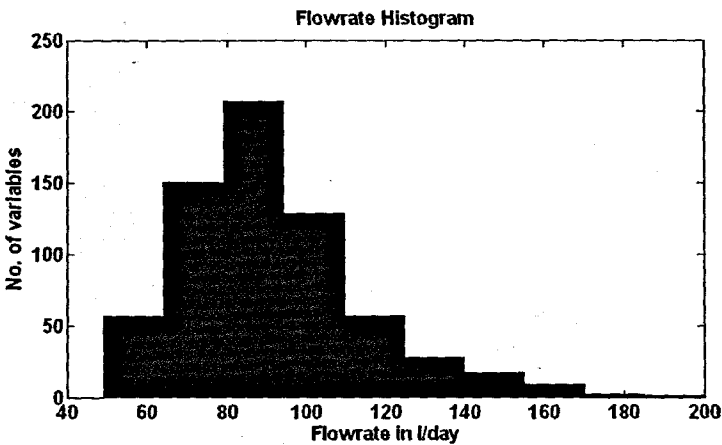
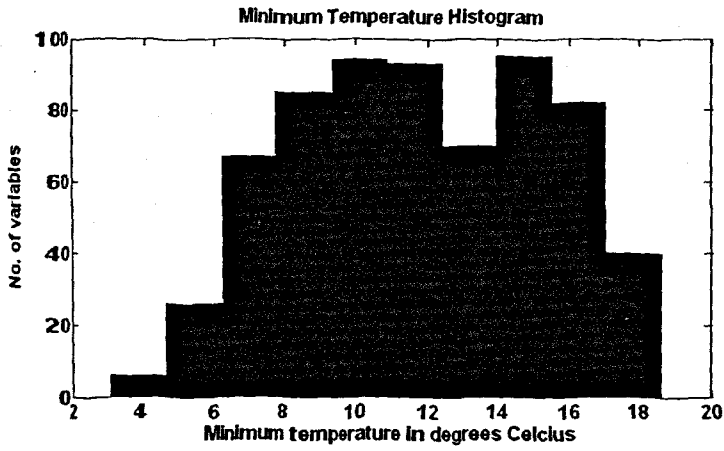
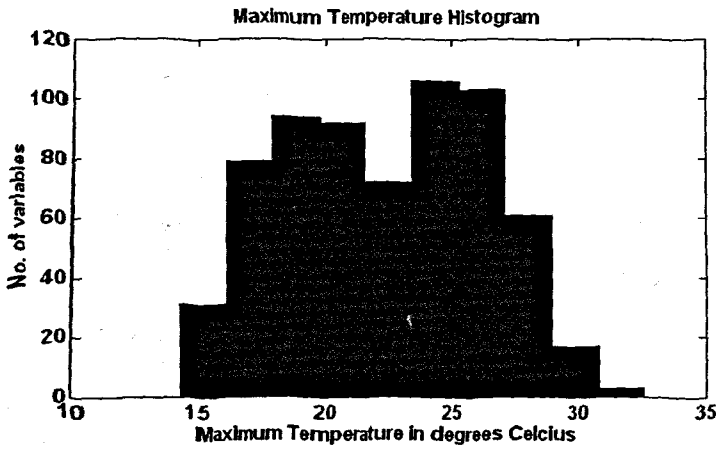


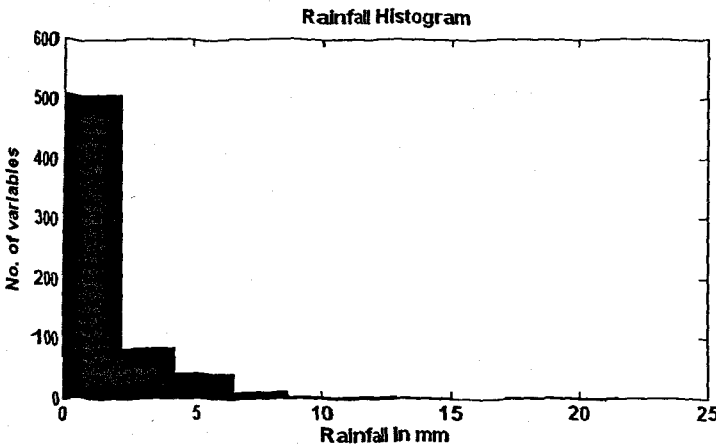
Figure 4. 13: Histogram plot of the Flow rate



**Figure 4. 14: Histogram plot of the Minimum temperature**



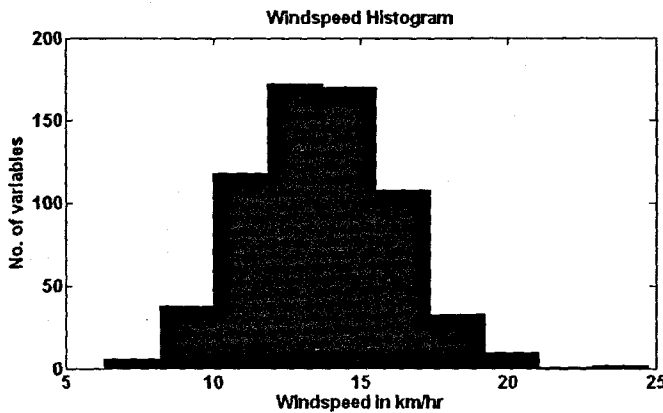
**Figure 4. 15: Histogram plot of the Maximum temperature**



**Figure 4.15: Histogram plot of the Rainfall**

An examination of the rainfall histogram (Figure 4.15) reveals that the data from about 10mm up to 22mm cannot be classified as outliers as this could be the low rainfall

periods and therefore this data is then most definitely valid. Therefore it is important to have some prior knowledge as to the significance, importance and relevance of the variables. This is the case for the wind speed variable (Figure 4.16) as well. The data between 20 – 25 km/hr may be regarded as outliers, if enough knowledge about the wind conditions in the area is not known.



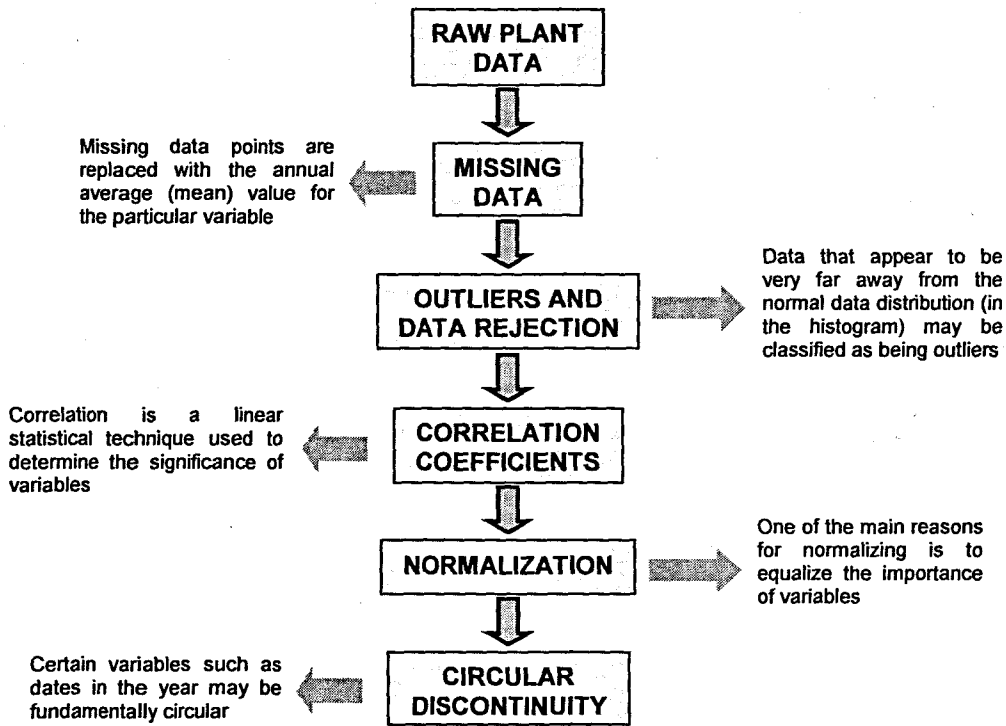
**Figure 4. 16: Histogram plot of the Wind speed**

In the Cape region the wind speed may for the most time be very low, but may be strong for a limited period of time.

Initially the possible outliers are not discarded and the different networks are trained and tested on the original raw data with no missing points. For the influent variables of COD and TKN, the predictive performance of the neural network is quite far from the desired target test set, therefore the possible outliers are discarded and replaced with values determined by linear interpolation, and the networks are re-examined. After extensive experimentation the conclusion is reached that the predictive performance results of the neural networks with or without the possible outliers appeared to be unchanged. The disturbance variables of interest are the only variables where the outliers are replaced using linear interpolation. The weather data variables are not changed, as there does not appear to be any problems with outliers.

**Table 4. 6: Summary of implemented pre-processing techniques**

Description of the preprocessing technique	Implemented in this work?
Handling of missing data	YES
Outliers and data rejection	YES
Correlation of input variables with each other	YES
Normalization / scaling	YES
Trends	NO
Seasonality	YES
Circular discontinuity	YES
Curse of dimensionality	NO
Principal component analysis	NO



**Figure 4. 17:** A summary of the pre-processing techniques implemented on the plant and weather data

After the neural network has been trained, the network has to be tested with data it has not seen before to determine the generalisation ability of the network. This data is subjected to the same processing techniques as for the training data.

#### 4.5 Data post-processing

Post-processing tasks include denormalizing the data back by its original scaled value. The maximum value used in denormalizing the data is used to transform the data into its original range.

#### 4.6 Summary of the chapter

This chapter discusses the importance of careful selection of input data variables to a neural network. The plant and weather input selection criteria are examined based on operator experience and the literature survey. The raw plant and weather data are available as two formats, namely MS Excel and ASCII text format. All data variables are imported into the MySQL database before being processed by the MATLAB NN algorithms.

Various pre-processing techniques are examined and proposed:

- the handling of missing data;



- outliers and data rejection;
- correlation;
- normalization;
- trends;
- seasonality;
- circular discontinuity;
- the curse of dimensionality; and
- principal component analysis.

A few of these techniques are summarised in Figure 4.18. These techniques are then applied to the raw plant and weather data before being used as inputs to the NN algorithm. Table 4.6 summarizes the techniques that are implemented in this work.

Chapter 5 discusses the actual development of the three different architectures (topologies) including the multilayer perceptron, recurrent Elman and radial basis functions NN. The various input combinations, the effect of varying the number of neurons and epochs, training methods and performance measurements are considered among other topics.

# CHAPTER FIVE

## APPLICATION OF NEURAL NETWORKS FOR INFLUENT DISTURBANCES PREDICTION

### 5.1 Introduction

Neural networks offer the ability to model arbitrary input data by means of adjusting the internal network connections, such that for a given input the difference between the network output and the desired response, the error, is minimized. This is the process of 'training' the network by means of supervised learning, whereby the network error is iteratively reduced over a training set of matching input-output vectors. Provided the training set is representative of the process, the resultant NN model captures the inherent relationship between input and output and with the ability to generalise for future unseen inputs (Garvey, 1997).

The main objective of this research is to find a NN model to predict one by one the influent disturbances COD, TKN, inflow rate to a wastewater treatment plant on the basis of the predicted and other two influent disturbances and weather conditions. One of the challenges is to find an optimal network topology by varying the types of inputs to the network and so solve the particular input-output mapping problem. The correct network architecture for a given application is highly problem dependent, a diversity of the training set and the complexity of the underlying function (Garvey, 1997).

Endeavouring to find the correct architecture involves the time-consuming task of investigating the effect varying certain parameters has on the NN learning, generalisation, and ultimately prediction capability. This chapter presents the implementation strategy for the three different NN architectures (topologies), namely the Multilayer Perceptron (MLP), the Elman Recurrent NN (ERNN) and the Radial Basis Function NN (RBFNN). Each topology is implemented having three neural networks, one for each influent disturbance namely, COD, TKN and influent flow rate. The common factor for the three topologies is that the input variations remain the same for all. Other factors such as the effect of the number of neurons, hidden layers and epochs are considered. The training algorithms that are investigated and used are also shown. The training and validation process is presented, and finally the performance criterion is discussed.

All the neural network algorithms presented in this work are implemented using the Neural Network Toolbox for use with MATLAB®, Version 4.0.6, (R14SP3). The MATLAB version used is MATLAB Version 7.1.0.246 (R14) Service Pack 3. All references to the software can be cross-referenced in the work of Demuth, Beale and Hagan (2004)

The first NN presented is the MLP where different combinations of WWTP influent disturbances and weather inputs are presented to the NN, the number of hidden layers and neurons are increased, training conditions are changed, and the network response is observed. The second NN is the ERNN and finally the RBF NN is presented.

## 5.2 The Multilayer Feed-forward Model

The limitations of the perceptron presented in Chapter 2, are overcome by arranging the individual neurons in a multilayer configuration (Figure 5.1). The output of the one neuron becomes the input of the neuron in the following layer. The network consists of an *input layer*, followed by a *hidden layer* and lastly, the *output layer*. The *input layer* does no processing on the arbitrary number of input nodes, but supplies the input signals to the next layer of neurons. The *hidden layer* is a layer that consists of an arbitrary number of neurons having a continuous activation function such as the tangent-sigmoid function. The final *output layer* consists of neuron elements the same as the hidden layer, however each output of the activation function represents the network output. Therefore the MLP consists of 3 layers:

- an input layer with no weights, summation or activation function;
- a hidden layer with weights, summation and a hyperbolic tangent function; and
- output layer with weights, summation and a linear activation.

However it should be noted that certain references refer to the above MLP network as a **TWO-layer** network as the input is not considered because no processing takes place here.

Three different MLP NNs are implemented, one NN for each influent disturbance variable, namely COD, TKN and Flow rate respectively. All the outputs are a prediction of the input disturbance variable at one time interval ahead; in this case the interval is one week ahead. The time horizon for the predicted output is one week ahead.

The structure of the MLP for prediction of the  $l^{\text{th}}$  disturbance  $l = \overline{1, m_1}$  is given in Figure 5.1 and 5.2.

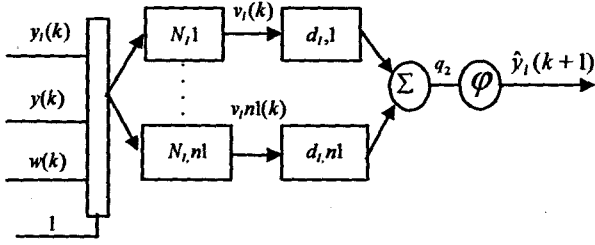


Figure 5. 1: The Multilayer Perceptron Neural Network for prediction of an input disturbance

The  $N_{i,i}$  structure is given as

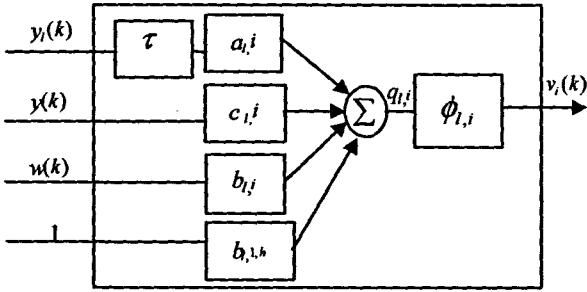


Figure 5. 2: The model of a neuron for the Multilayer Feed-forward Neural Network

The output of the neuron  $N_{i,i}$  is:

$$v_{i,i}(k) = \phi_{i,i}(a_{i,i}y_1(k - \tau) + c_{i,i}y(k) + b_{i,i}w(k) + b_{i,i,h}) \quad 5.1$$

where  $i = \overline{1, n_1}$ , and the output of the NN is:

$$\hat{y}_1(k+1) = \phi_l(d_l v_l(k) + b_{l,0}) \quad 5.2$$

where  $a_{i,i} = [a_{i,i,1} \dots a_{i,i,p}] \in R^p$  is a vector of the NN coefficients introducing the delayed values of the  $l^{\text{th}}$  disturbance;  $b_{i,i} = [b_{i,i,1} \dots b_{i,i,m_2}] \in R^{m_2}$  is the vector of the NN coefficients introducing all considered weather conditions;  $c_{i,i} = [c_{i,i,1} \dots c_{i,i,m_1}] \in R^{m_1}$  is the vector of the NN coefficients introducing the inflow disturbances;  $b_{i,i,h}$  is the bias coefficient;  $p$  is the number of delays,  $m_1 = 3$  is the number of disturbances where,  $y_1 = \text{COD}$ ,  $y_2 = \text{TKN}$ ,  $y_3 = \text{Flowrate}$ ;  $m_2 = 5$  is the number of weather conditions, where  $w_1 = \text{month (sin)}$ ,  $w_2 = \text{month (cos)}$ ,  $w_3 = \text{min temp}$ ,  $w_4 = \text{rain}$ ,  $w_5 = \text{wind speed}$ ;  $\phi$  is the activation function of the output layer; and  $\phi_{i,i}$  is the activation function of the hidden layer for the neuron  $N_{i,i}$ . The estimation of parameters  $a_{i,i}$ ,  $b_{i,i}$  and  $c_{i,i}$  is based on minimising some error function, the most common being the mean squared error or cost function given as:

$$\text{mse} = \frac{1}{N} \sum_{k=1}^N e(k)^2 = \frac{1}{N} \sum_{k=1}^N (y(k) - \hat{y}(k))^2 \quad 5.3$$

where  $\hat{y}(k)$  is the NN output and  $y(k)$  is the desired output.

### 5.2.1 Feed-forward Multilayer Neural Network Implementation Criteria

There are a number of factors to consider when attempting finding a solution to the prediction problem. These factors include:

- Input selection;
- The length and selection of training and validation sets;
- Activation function;
- Error criterion;
- Number of hidden layers;
- Number of neurons in the various hidden layers;
- Number of epochs;
- Weight and bias initialisation;
- Training method;
- Learning rate;
- Gradient method;
- Training stopping criterion; and
- Performance measurement.

#### 5.2.1.1 Input selection

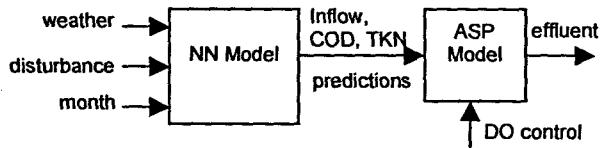
The multilayer perceptron (MLP) is a static network, as it has no physical feedback connections. To transform the MLP into a dynamic network, time delays (memory) in the input data are introduced (Olssen and Newell, 1999), (Elman, 1990). These delays are applied to the influent data of the predicted variable only and the period for the delay is 3 tapped unit steps. This approach has been successfully applied in short-term load forecasting (Khan, 2000) and flood forecasting (Chen, *et al.*, 2001) among others.

The problem for prediction of the inflow disturbances is based on the analysis of the environmental factors that could influence the input flow characteristics. The literature review and plant operator experience showed that such factors include:

- weather conditions - temperature, wind velocity, rainfall, etc.;
- season, day of the week, time of day;
- human activities; and
- industrial activities.

The first two groups of factors are selected as a basis for disturbance prediction as some pattern in the behaviour of the process can be observed when they are changed. Then the problem for input disturbances prediction can be formulated in the following way:

- find a NN model describing the behaviour of the inflow characteristics on the basis of the predicted disturbance delayed data, all inflow disturbances data, the weather conditions data and month of the year characteristics, as given in Figure 5.3, in such a way that the mean square error in Equation 5.3 between the real data for the inflow characteristics and the NN prediction is minimum.



**Figure 5. 3: Neural Network model describing the inflow characteristics on the basis of environmental factors**

The solution to this problem is based on the following steps:

- pre-processing of I/O data;
- selection of the NN architecture (topology);
- selection of the type of the predicting model – static or dynamic;
- calculation of the NN weights by training (learning); and
- validation of the model.

The method followed in order to determine the correct combination of disturbance and weather variables in order to form the predicting NN inputs, is an arduous and laborious task of empirical testing. The proposed combination of variables consists of the influent disturbance variables and the environmental variables together with delayed inputs of the disturbance variable to be predicted. The different combination of the input variables determines different models of the NN. The model numbers represent 13 different input variations which are summarized in Tables 5.1, 5.2 and 5.3, where the considered combinations of the NN inputs for the prediction of COD, TKN and influent FLOW rate, are given respectively.

After the inputs have been selected, various processing techniques are performed on the data before it is input to the NN. These preprocessing techniques are discussed in Chapter 4.

### **5.2.1.2 Length and selection of training and validation sets of data**

It is advisable to make the test set representative of the entire period of the training set. Caution should be exercised however to ensure the test set does not include data from the training set. If however the NN is trained with a particular set of data, yet tested with another completely different data set, the network is not be able to generalize. The total number of data points is 658 as is previously mentioned. One-third of the data is used for the testing (validation) set (Haykin, 1999). The method followed is to extract every 3<sup>rd</sup> sample from the entire 658 points as the test set. This results in a test set of 218 points and a training set of 436 points. This method is acceptable and is proposed in the literature consulted.

One other method considered is the early stopping method where the data is divided into three subsets, namely a training, validation and test set. The early stopping method is discussed in Chapter 3. This method did not however produce acceptable results and the first approach was adopted where the data is divided into two sets.

**Table 5. 1: Input variable combinations to the NN with COD as NN output**

No. of inputs	Model No.	Input description	Input – COD	Output
1	Model 1	Disturbance only	COD	COD(k+1)
2	Model 2	Disturbance, disturbance delayed 1 time interval	COD, COD(k-1)	COD(k+1)
3	Model 3	Disturbance, disturbance delayed 2 time intervals	COD, COD(k-1), COD(k-2)	COD(k+1)
4	Model 4	Disturbance, disturbance delayed 3 time intervals	COD, COD(k-1), COD(k-2), COD(k-3)	COD(k+1)
5	Model 5	Disturbance, delayed disturbance, one other disturbance	COD, COD(k-1), COD(k-2), COD(k-3), TKN	COD(k+1)
6	Model 6	Disturbance, delayed disturbance, all other disturbances	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW	COD(k+1)
8	Model 7	Disturbance, delayed disturbance, all other disturbances, month	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW, Month	COD(k+1)
9	Model 8	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW, Month, Minimum Temperature	COD(k+1)
9	Model 9	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW, Month, Rain	COD(k+1)
9	Model 10	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW, Month, Wind	COD(k+1)
10	Model 11	Disturbance, delayed disturbance, all other disturbances, month, 2 environmental	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW, Month, Minimum Temperature, Rain	COD(k+1)
11	Model 12	Disturbance, delayed disturbance, all other disturbances, month, all environmental	COD, COD(k-1), COD(k-2), COD(k-3), TKN, FLOW, Month, Minimum Temperature, Rain, Wind	COD(k+1)
9	Model 13	Disturbance, delayed disturbance, month, all environmental	COD, COD(k-1), COD(k-2), COD(k-3), Month, Minimum Temperature, Rain, Wind	COD(k+1)

### 5.2.1.3 Activation Function

The activation function chosen, is the hyperbolic tangent sigmoid activation function  $\phi$  in Equation 5.4 that states:

$$\phi(q_{i,i}) = \tanh(q_{i,i}) = \frac{e^{aq_{i,i}} - e^{-aq_{i,i}}}{e^{aq_{i,i}} + e^{-aq_{i,i}}}$$

5.4

where  $y$  is a the output of the neuron and the argument of the activation (transfer) function. The coefficient  $a > 0$  is set by the MATLAB initialisation function (see section 5.2.1.7). The reason that this (hyperbolic tangent) activation function is chosen is that

the month data input ranges between +1 and -1. Therefore the much more common *log-sigmoid* transfer function cannot suffice. The difference between the two activation functions being that the *log-sigmoid* compresses data in the range between 0 and 1, and the *hyperbolic tangent* compresses data between +1 and -1.

The output layer simply has a pure linear activation function,  $\varphi(q_2) = k_2 q_2$ , where  $k_2$  is the coefficient. This type of selection for the activation functions for the hidden and output layers is fairly common for regression or forecasting applications.

**Table 5. 2: Input variable combinations to the NN with TKN as NN output**

No. of inputs	Model No.	Input description	Input – TKN	Output
1	Model 1	Disturbance only	TKN	TKN(k+1)
2	Model 2	Disturbance, disturbance delayed 1 time interval	TKN, TKN(k-1)	TKN(k+1)
3	Model 3	Disturbance, disturbance delayed 2 time intervals	TKN, TKN(k-1), TKN(k-2)	TKN(k+1)
4	Model 4	Disturbance, disturbance delayed 3 time intervals	TKN, TKN(k-1), TKN(k-2), TKN(k-3)	TKN(k+1)
5	Model 5	Disturbance, delayed disturbance, one other disturbance	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD	TKN(k+1)
6	Model 6	Disturbance, delayed disturbance, all other disturbances	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW	TKN(k+1)
8	Model 7	Disturbance, delayed disturbance, all other disturbances, month	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW, Month	TKN(k+1)
9	Model 8	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW, Month, Minimum Temperature	TKN(k+1)
9	Model 9	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW, Month, Rain	TKN(k+1)
9	Model 10	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW, Month, Wind	TKN(k+1)
10	Model 11	Disturbance, delayed disturbance, all other disturbances, month, 2 environmental	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW, Month, Minimum Temperature, Rain	TKN(k+1)
11	Model 12	Disturbance, delayed disturbance, all other disturbances, month, all environmental	TKN, TKN(k-1), TKN(k-2), TKN(k-3), COD, FLOW, Month, Minimum Temperature, Rain, Wind	TKN(k+1)
9	Model 13	Disturbance, delayed disturbance, month, all environmental	TKN, TKN(k-1), TKN(k-2), TKN(k-3), Month, Minimum Temperature, Rain, Wind	TKN(k+1)

#### 5.2.1.4 Error criterion

The cost function used during training is the means squared error defined in Equation 5.3. The error is calculated as the difference between the target output and the network output. We want to minimize the average of the sum of these errors. This is accomplished by changing the weights and biases within the network. The neural



network is trained until either the error is a minimum, or the maximum number of epochs is reached.

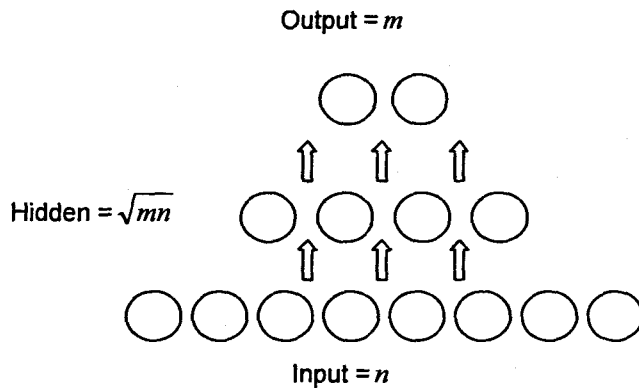
**Table 5. 3: Input variable combinations to the NN with influent FLOW rate as NN output**

No. of inputs	Model No.	Input description	Example Input - FLOW	Output
1	Model 1	Disturbance only	FLOW	FLOW(k+1)
2	Model 2	Disturbance, disturbance delayed 1 time interval	FLOW, FLOW(k-1)	FLOW(k+1)
3	Model 3	Disturbance, disturbance delayed 2 time intervals	FLOW, FLOW(k-1), FLOW(k-2)	FLOW(k+1)
4	Model 4	Disturbance, disturbance delayed 3 time intervals	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3)	FLOW(k+1)
5	Model 5	Disturbance, delayed disturbance, one other disturbance	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), TKN	FLOW(k+1)
6	Model 6	Disturbance, delayed disturbance, all other disturbances	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN	FLOW(k+1)
8	Model 7	Disturbance, delayed disturbance, all other disturbances, month	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN, Month	FLOW(k+1)
9	Model 8	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN, Month, Minimum Temperature	FLOW(k+1)
9	Model 9	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN, Month, Rain	FLOW(k+1)
9	Model 10	Disturbance, delayed disturbance, all other disturbances, month, 1 environmental	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN, Month, Wind	FLOW(k+1)
10	Model 11	Disturbance, delayed disturbance, all other disturbances, month, 2 environmental	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN, Month, Minimum Temperature, Rain	FLOW(k+1)
11	Model 12	Disturbance, delayed disturbance, all other disturbances, month, all environmental	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), COD, TKN, Month, Minimum Temperature, Rain, Wind	FLOW(k+1)
9	Model 13	Disturbance, delayed disturbance, month, all environmental	FLOW, FLOW(k-1), FLOW(k-2), FLOW(k-3), Month, Minimum Temperature, Rain, Wind	FLOW(k+1)

### 5.2.1.5 Number of hidden layers and hidden neurons

The multilayer perceptron is said to be able to approximate any function provided there are sufficient neurons (Haykin, 1999) to solve a complex problem. A major problem in designing a neural network is establishing the optimal number of layers and number of neurons on each of it. For the *input* and *output layers*, the solution is very easy, because for the input the number of neurons are equal to the numbers of weights and biases (Kröse, and van der Smagt, 1996), the problem being in establishing the number of hidden layers and the number of neurons on each layer. The literature research indicated that many authors propose rules of thumb to deal with specific practical issues. However the experience of the author is that sometimes these do not hold true as the problem at hand might warrant a different approach.

Masters (1993) proposes using the following method (Figure 5.4) to determine the number of hidden neurons.



**Figure 5.4:** Calculating the number of hidden layer neurons using a pyramid approach (Masters, 1993)

Choosing the number of hidden layers and the number of neurons is therefore difficult, because there are no generally acceptable theories, with solutions in the literature only for specific cases. At present there are no formal ways of determining the size of a neural network, according to complexity of the problem to solve. The solutions normally are found by means of empirical testing. For real-world problems with NNs the network tends to be of a large size. One practical consideration is to try and minimize the size of the network while maintaining good performance. Smaller NNs are less prone to noise in the training data set and generalize better.

The number of neurons must be high enough for generating a configuration of the decision areas and complex enough for a given problem. However, if the number of neurons is too big, then the number of connections becomes too big and there is a risk that the balance of these connections are not be calculated using only the available examples. In this case it is possible for the network to generate noise (Hornik, *et al.*, 1990); hence the phenomenon of overfitting the neural network. On the other hand if the number of neurons from the hidden layers is too small, the neural network can't learn the entire input-output mapping relationship, and therefore cannot solve the given problem. Thus there is a tradeoff between the number of neurons, training time, and problem solving ability (Park, 1996).

One method proposed for the determination of the optimal architecture of a neural network for solving the same problem is the choosing of a large number of networks with different architectures and training them on a set of common database, until the performance criteria is fulfilled and then selecting the structure with the smallest

dimensions. The disadvantage of this is that the results depend on the initial architecture selection and it takes a very long time to implement (Ostafe, 2005).

Haykin (1999) proposes two techniques to remedy this dilemma, namely *network growing* and *network pruning*. Network growing involves the recommendation to start with only one hidden layer, and if the results are not good, increase the number of hidden layers. (Hornik, et al., 1990) (Haykin, 1999). The second method of network pruning is where a large MLP is used that has adequate performance and is able to solve the problem at hand, and then prune it by eliminating certain weights in a selective and orderly manner (Haykin, 1999).

The approach adopted in this work is one of network growing, where a small network is grown. However it needs to be mentioned that the method of network pruning is attempted, but a large enough MLP able to generalize could not be successfully implemented. The number of neurons is varied from 1 to 5 up to 10. The number of hidden layers is determined experimentally to be one layer. Results for two layers are included for selected models in Chapter 6.

#### **5.2.1.6 Number of epochs**

The number of epochs is determined by experimentation. A low mean squared error (MSE) during training suggests that the network construction is of a good quality. However, this is not usually the case as overfitting occurs. The generalization ability of the network therefore has to be monitored as the number of epochs is increased. The number is varied from 500 to 1000 epochs. There are results where the epochs are varied as low as 100 and the performance of the network is similar to that where the number of epochs is high.

#### **5.2.1.7 Weight and bias initialisation**

The initial weights and bias values are critical to the success of the learning process. Incorrectly initialised weights cause the activation potentials to be large which saturates the neurons. In saturation the derivatives of the activation function is equal to zero and no learning takes place (Papliński, 2004).

All weights and biases are initialised randomly in the initial tests performed. Later initialisation of all the weights and biases was performed using the Nguyen-Widrow layer initialisation function (Demuth, Beale and Hagan, 2004). The Nguyen-Widrow method generates initial weight and bias values for a layer, so that the active regions of the layer's neurons are distributed approximately evenly over the input space. The advantages over purely random weights and biases are:

- Few neurons are wasted; and
- Training is faster.

Proper initialisation can significantly speed up the learning process.

#### **5.2.1.8 Training Method**

As discussed in Chapter 3, there are broadly speaking two different styles of training, namely *incremental* training and *batch* training. In *incremental* training the weights and biases of the network are updated each time an input is presented to the network. Incremental training can be applied to both static and dynamic networks, although it is more commonly used with dynamic networks, such as adaptive filters. For *batch* training the weights and biases are only updated after all the inputs are presented. The training method used in this work is the batch training method.

#### **5.2.1.9 Learning rate**

The least mean square (LMS) algorithm is an example of supervised learning where the learning rule is provided with a set of inputs and desired target outputs. The LMS (also referred to as the Widrow-Hoff) learning algorithm is based on an approximate steepest descent procedure. Upon application of each input to the network, the network output is compared to the target. The LMS algorithm minimizes the mean squared error.

When the learning rate (see Chapter 3) is large, learning occurs quickly. If the learning rate is too large, it leads to instability and the errors even increase. To ensure stable learning, the learning rate must be less than the reciprocal of the largest eigenvalue of the correlation matrix of the input vectors (Demuth, Beale and Hagan, 2004). The learning rate for the considered case is set to a value of 0.6; this is determined by experimentation.

#### **5.2.1.10 Gradient method**

Once the number of layers and the number of neurons in each layer has been selected, the weights and biases must be adjusted so as to minimize the prediction error made by the network. This is the role of training algorithms. This process is equivalent to fitting the model represented by the network to the available training data (Statsoft, 2003). The objective of training is to find the lowest point in the multi-dimensional error surface. NN error surfaces are complex and are characterized by local minima. It is not possible to analytically determine where the global minimum of the error surface is. NN training is thus an exploration of the error surface (Bishop, 1995). Typically the gradient (slope) of the error surface is calculated at the current

point, and used to make a downhill move (Statsoft, 2003). The algorithm stops at a low point, which is hopefully the global minimum.

In selecting a training algorithm the one crucial factor considered is the time taken for the mean square error to reach a minimum. This varies depending on factors such as the complexity of the given 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 in an application for pattern recognition (discriminant analysis) or function approximation (regression). The MATLAB Neural Network Toolbox contains many training algorithms as summarized in Table 5.4.

**Table 5. 4: Training algorithms for use with the MATLAB Neural Network Toolbox**

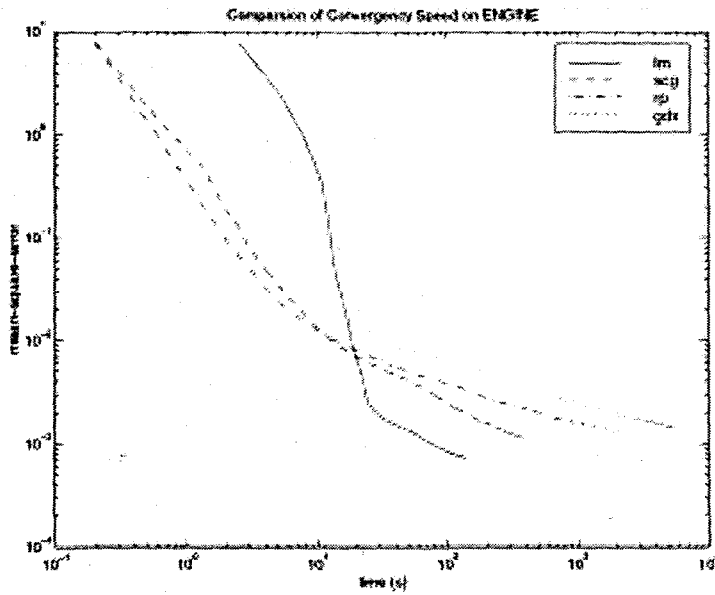
Acronym	Algorithm	Description
LM	trainlm	Levenberg-Marquardt
BFG	trainbfg	BFGS Quasi-Newton
RP	trainrp	Resilient Backpropagation
SCG	trainscg	Scaled Conjugate Gradient
CGB	traincgb	Conjugate Gradient with Powell/Beale Restarts
CGF	traincgf	Fletcher-Powell Conjugate Gradient
CGP	traincgp	Polak-Ribière Conjugate Gradient
OSS	trainoss	One-Step Secant
GDX	traingdx	Variable Learning Rate Backpropagation

Of all the algorithms presented in the table, only four are discussed and the advantages and disadvantages considered. In order to provide the proper perspective needed for this discussion, one of six benchmarks for a real world application (Engine operation) taken from Demuth, Beale and Hagan (2004), is presented in Figure 5.5. The results show the convergence time in seconds for the Levenberg-Marquardt, Scaled Conjugate Gradient, Resilient Backpropagation and Variable Learning Rate Propagation are 18.45, 36.02, 65.91 and 188.50 respectively. The discussion below pertains to the benchmark problem.

Generally speaking, on function approximation problems, for networks that contain up to a few hundred weights, the Levenberg-Marquardt (LM) algorithm has the fastest convergence and is especially suited to problems requiring very accurate training. Compared to the other algorithms, the LM is able to obtain lower mean square errors. The drawback to the LM algorithm is that an increase in the number of weights in the network causes the advantage of using this algorithm to decrease. Another disadvantage is that algorithm does not perform well for pattern recognition problems.

The memory storage requirements of the LM algorithm are larger than the other algorithms tested.

The resilient backpropagation (RP) algorithm is the fastest algorithm on pattern recognition problems, but performs poorly on function approximation problems. Reducing the error goal causes this algorithm to degrade. One advantage is that this algorithm is not as memory intensive compared to the other considered algorithms.



**Figure 5.5:** The Engine benchmark problem illustrating the convergence speed for the different training algorithms

The scaled conjugate gradient (SCG) algorithm appears to perform well over a wide variety of problems, particularly for networks with a large number of weights. This algorithm is faster than the LM algorithm for large networks for function approximation problems and almost as fast as the RP algorithm on pattern recognition problems. The performance of the SCG does not degrade as quickly as the RP algorithm does when the error is reduced. One major advantage is the conjugate gradient algorithms have relatively modest memory requirements.

The variable learning rate algorithm (GDX) is usually much slower than any of the other algorithms, but it can still be useful for some problems. However when using early stopping the training algorithm should not converge too quickly otherwise inconsistent results are obtained. For this benchmark application, the GDX algorithm is probably the best option to consider.

For the practical implementation of the MLP and ERNN in this work, all four training algorithms discussed in the previous section are considered. The Levenberg-Marquardt is the initial algorithm that exhibits promising performance for the given problem. The LM is an example of a backpropagation algorithm that is optimized using numerical methods. However for the larger networks, the LM algorithm is slow and prone to reaching a minimum gradient, thus stopping the training prematurely. The slowness of the algorithm is attributed to the fact that this algorithm is memory intensive. The RP and GDX exhibit similar performance to that of the LM.

The conclusions reached after the practical implementation of the LM, RP and GDX training algorithms are:

- that the standard backpropagation method is too slow;
- the faster modified backpropagation algorithm with a momentum term added, still did not perform satisfactorily in terms of training speed;
- that although the LM, RP and GDX are faster than the standard backpropagation algorithms, the execution speed was still slower than the SCG; and
- for larger networks the LM algorithm ran out of memory.

The scaled conjugate gradient algorithm compared to the above algorithms performs the best if faster training is required. The scaled conjugate gradient uses a search along the error surface based on the conjugate direction of the gradient and not the steepest descent, like the standard backpropagation method. This search in the conjugate direction gradient results in faster convergence.

Based on all of the above considerations and by extensive experimentation, the scaled conjugate gradient (SCG) algorithm is eventually selected as being the best for the given problem. The results for the time required for training the Levenberg-Marquardt versus the scaled conjugate gradient is reported in Chapter 6.

#### **5.2.1.11 Training Stopping criterion**

In order to ensure that the NN is able to generalize, it is sometimes necessary to stop the training (learning) before the network converges to a global minimum. The learning or training phase of a NN can be interrupted on three conditions:

- The maximum number of epochs has been reached;
- The mean squared error has converged to a minimum; or
- The learning algorithm has reached a minimum gradient.

The operator can also manually abort the learning algorithm if it is observed that the error stays constant for a sufficiently long number of training iterations (epochs).

### 5.2.1.12 Performance measurement

The prediction performance of the NN can be measured using a number of different factors. One has to be careful as to what criterion is used when deciding on the measurement to determine the NN performance. In the thesis three methods are employed to measure the predictive performance of the NN.

The first method is to determine the regression or correlation coefficient between the desired target output and the NN output. Regression analysis is used to describe techniques that help in deciding whether there is a relationship between particular variables. Correlation on the other hand is defined as the measure of the linear relationship between variables. The relationship between regression and correlation is discussed in Bettely, et al. 1994. In order to determine the relationship between the NN output and the desired target output a regression line is drawn. This can be seen graphically in Figure 6.5 in Chapter 6. Many of the calculations involved in finding the correlation coefficient are the same as those used in the calculation of regression line (Betteley, et al. 1994). Therefore one can use the correlation measurement,  $r$  to determine whether the NN output is close to the target output.

$$r = \frac{n \sum \hat{y} - (\sum y)(\sum \hat{y})}{\sqrt{(n \sum y^2 - (\sum y)^2) \cdot (n \sum \hat{y}^2 - (\sum \hat{y})^2)}} \quad 5.5$$

where  $y$  is the target output, and  $\hat{y}$  is the NN output and  $n$  is the number of data points.

The correlation coefficient always has a positive value between minus one and plus 1, which is expressed as  $-1 \leq r \leq +1$ .

- For  $r = +1$ , there is perfect positive correlation and all points lie on a straight line with a positive slope.
- For  $r = -1$ , there is perfect negative correlation and all points lie on a straight line with a negative slope.
- For  $r = 0$ , there is no correlation.

Therefore for  $r = +1$ , the NN has approximated the function (fitted the data) very well.

The second measure of performance is the coefficient of determination  $r^2 \times 100$ . This is stated as a percentage.

The third and final performance measurement used in this work is the absolute percentage error (APE) and is given by:



$$APE = 100 \times [|y - \hat{y}| / \hat{y}]$$

5.6

where  $y$  is the desired target output, and  $\hat{y}$  is the NN output.

The performance is measured during both the training and validation phases. The results of the NN performance are presented in Chapter 6.

### 5.3 The Elman Recurrent Neural Network Model

Recurrent NNs have feedback connections and are often used to represent dynamic systems. Two straightforward ways of including recurrent connections in NNs is the *activation feedback*, where the feedback connection is from the output of the hidden neuron back to the input, and *output feedback* where the feedback is from the output back to the input (Mandic and Chambers, 2001). These recurrent networks can be either locally recurrent such as the Elman and Jordan recurrent NN. The Elman network is also referred to as locally recurrent-globally feed-forward structure. Elman networks are two-layer (excluding the input) backpropagation networks, with the addition of a feedback connection from the output of the hidden layer to its input (Figure. 5.5). This feedback path allows Elman networks to learn to recognize and generate temporal patterns, as well as spatial patterns. These recurrent connections are what give the network memory (Elman, 1990). Every neuron in the hidden layer receives as input, in addition to the external inputs of the network, the outputs of the hidden layer neurons. Other recurrent networks such as the Williams-Zipser network that is a fully connected recurrent NN is presented in Mandic and Chambers (2001). The inputs to both the feed-forward and recurrent NNs are considered to be exactly the same, therefore Equations 5.1 and 5.2 can be modified as follows:

$$v_{i,i}(k) = \phi_{i,i}(a_{i,i}y_i(k-\tau) + c_{i,i}y(k) + b_{i,i}w(k) + g_{i,i}v_i(k-\theta) + b_{i,ih}), i = \overline{1, n_i} \quad 5.7$$

and the output of the RNN is:

$$\hat{y}_i(k+1) = \phi_i(d_i v_i(k) + b_{i,o}) \quad 5.8$$

where additionally to the previous notation  $g_{i,i} \in R^{n_i}$  is a vector of the NN weight which introduces the delayed output of the first hidden layers as an input to the first hidden layer where  $\theta \in R^{n_i}$  is the vector of unit delays.

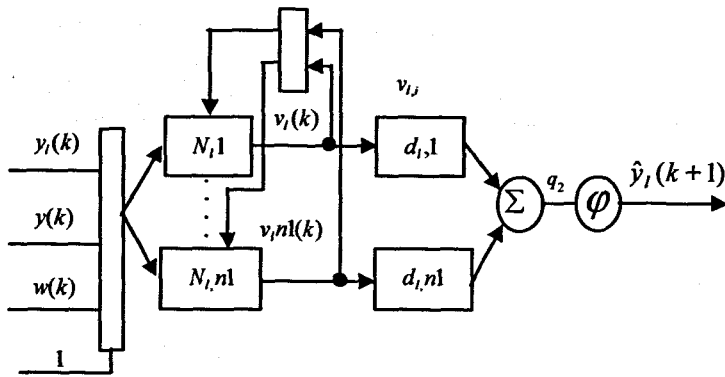


Figure 5. 6: The Elman Recurrent Neural Network for prediction of an input disturbance

The  $N_{i,l}$  structure is given as

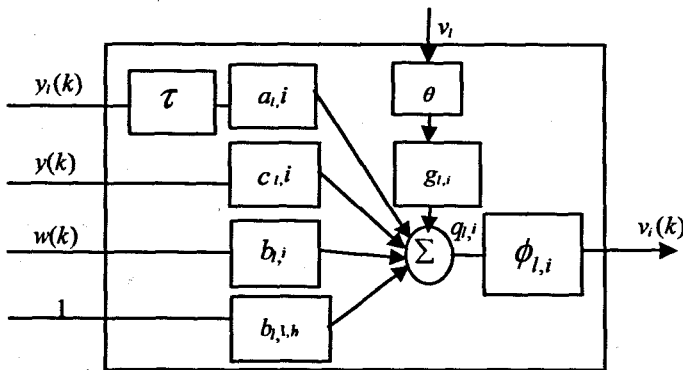


Figure 5. 7: The model of a neuron for the Elman Recurrent Neural Network

### 5.3.1 Elman Recurrent Neural Network Implementation Criteria

The factors considered for implementation of the ERNN included:

- Input selection;
- The length and selection of training and validation sets;
- Activation function;
- Error criterion;
- Number of hidden layers;
- Number of neurons in the various hidden layers;
- Number of epochs;
- Weight and bias initialisation;
- Training method;
- Learning rate;
- Gradient method;
- Training stopping criterion; and
- Performance measurement.

#### 5.3.1.1 Input selection

The inputs for the ERNN are the same as for the MLP and are described in Tables 5.1 – 5.3.

### **5.3.1.2 Length and selection of training and validation sets**

The partitioning of data into a training and test set for the ERNN are exactly the same as for the MLP. The training set has 436 data points and the test set has 218 data points. The test set is representative of the entire data set as every third sample is extracted.

### **5.3.1.3 Activation Function**

The activation function for the hidden layer for the ERNN is the hyperbolic tangent (Equation 5.4), and the linear function for the output layer.

### **5.3.1.4 Error criterion**

The performance function for the training phase is the minimising of the mean squared error (Equation 5.3).

### **5.3.1.5 Number of hidden layers and hidden neurons**

The number of hidden layers is found by experimentation to be one. The number neurons are varied from 1 to 5 to a maximum of 10. However this has been increased gradually to a maximum of 80.

### **5.3.1.6 Number of epochs**

The number of epochs is varied from 500 to 1000 epochs. The epochs are also varied from 100 epochs although these results are not included in this document due to space considerations.

### **5.3.1.7 Weight and bias initialisation**

All weights and biases are initialised using the Nguyen-Widrow layer initialisation function.

### **5.3.1.8 Training Method**

The NN is trained using the batch method where the entire input set is applied before the weights and biases are updated.

### **5.3.1.9 Learning rate**

The learning rate was kept at a value of 0.1. Changing the learning rate from 0.1 to 1.0 did not have a noticeable effect on the NN learning or predictive performance.

#### **5.3.1.10 Gradient method**

The error is minimised using the scaled gradient conjugate training algorithm for the considered case. This algorithm is faster than the Levenberg-Marquardt training algorithm.

#### **5.3.1.11 Training Stopping criterion**

Training stops when the error is at a minimum, or the algorithm encounters a minimum gradient, or when the operator manually stops the training phase.

#### **5.3.1.12 Performance measurement**

There are three performance measurements as indicated in section 5.2.1.12. These are the correlation or regression coefficient  $r$  described by Equation 5.5, the coefficient of determination  $r^2$  and the absolute percentage error  $APE$  (Equation 5.6). The ideal values for  $r$  is as close to 1 as possible,  $r^2$  should ideally be close to 1 or 100%, the  $APE$  should ideally be as small as possible.

### **5.4 The Radial Basis Function Neural Network Model**

The final topology considered is that of the Radial basis function NN. The RBFNN has a form similar to that of the MLP, as it is also a feed-forward network. The RBFNN is a three-layered feed-forward network (if the input is considered as a layer), comprising of the input, hidden layer and output layer respectively. The structure is represented in Figure 5.8 with the neuron structure in Figure 5.9. The difference between the MLP and the RBFNN is the activation for hidden neurons. For the RBFNN it is the radial basis function. The output layer is like the MLP a linear output which is the best layout for function approximation problems such as this.

Radial basis functions arise as optimal solutions to problems of interpolation, approximation and regularisation of functions (Haykin, 1999). Much of the inspiration for RBF neural networks comes from traditional statistical pattern classification techniques (Casey, 2004).

The unique feature of RBFNN is the process performed in the hidden layer. The idea behind this is that the patterns in the input space form clusters. If the centres of these clusters are known, then the distance from the cluster centre can be measured. This distance measurement is made nonlinear, so that if a pattern is in an area that is close to a cluster centre it gives a value close to 1 (Casey, 2004).

Radial basis functions  $\phi(x; t_i)$  form a family of functions of a  $p$ -dimensional vector,  $x$ , each function being centered at point  $t_i$ . A very common radial basis function is the Gaussian function that depends only on the distance between the current point  $x$ , and the center point,  $t_i$ , and the variance parameter  $\sigma_i$ :

$$\phi(x; t_i) = G(\|x - t_i\|) = \exp\left(-\frac{\|x - t_i\|^2}{2\sigma_i^2}\right) \quad 5.9$$

where  $\|x - t_i\|$  is the norm of the distance vector between the current vector  $x$  and the center point  $t_i$ , of the symmetrical multi-dimensional Gaussian surface. The spread of the surface is controlled by the variance parameter  $\sigma_i$ . The equation 5.9 represents a Gaussian bell-shaped curve as shown in Figure 3.11 in Chapter 3.

The distance measurement to the center of the cluster is referred to as the Euclidian distance and is represented by:

$$r_j = \sqrt{\sum_{i=1}^n (x_i - t_{ij})^2} \quad 5.10$$

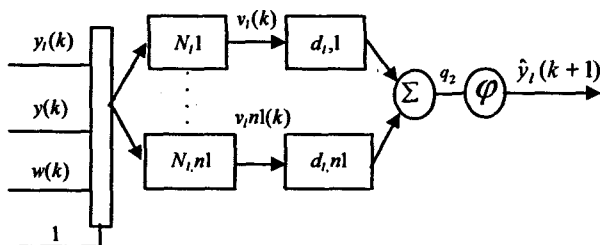


Figure 5. 8: The Radial Basis Neural Network for prediction of an input disturbance

The  $N_{i,i}$  structure is given as

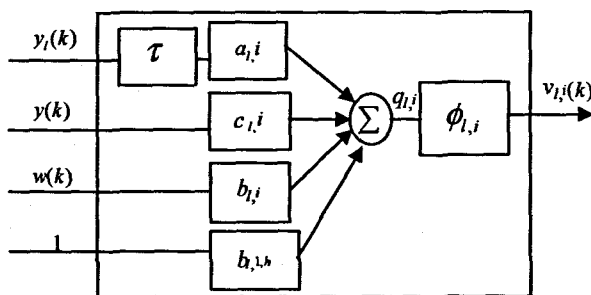


Figure 5. 9: The model of a neuron for the Radial Basis Neural Network

The equations for the radial basis function are exactly the same as for the MLP, the only difference being the activation, which is now represented by the radial basis function in Equation 5.9.

#### **5.4.1 Radial Basis Function Neural Network Implementation Criteria**

Radial basis networks often require more neurons than the MLP feed-forward NN, but it can be designed in a fraction of the time it takes to train the MLP NN. The RBFNN works best when many training vectors are available.

The factors considered for implementation of the ERNN included:

- Input selection;
- The length and selection of training and validation sets;
- Activation function;
- Error criterion;
- Number of hidden layers;
- Number of neurons in the various hidden layers;
- Number of epochs;
- Weight and bias initialisation;
- Training method;
- Error goal;
- Spread constant;
- Training stopping criterion; and
- Performance measurement.

##### **5.4.1.1 Input selection**

The inputs were selected as per the Tables 5.1 - 5.3.

##### **5.4.1.2 Length and selection of training and validation sets**

The partitioning of data into a training and test set for the RBFNN are exactly the same as for the MLP and ERNN.

##### **5.4.1.3 Activation Function**

The activation function for the RBFNN is different to the MLP and ERNN, and is the radial basis function for the hidden layer, and the linear function for the output layer.

The coefficient  $t_i$  is set by the MATLAB function for the initialisation of the RBF network.

##### **5.4.1.4 Error criterion**

The performance function for the training phase is the minimising of the sum of squares error (Demuth, Beale and Hagan, 2004).

##### **5.4.1.5 Number of hidden layers**

The number of hidden layers is kept at one layer.

#### **5.4.1.6 Number of epochs and hidden neurons**

A neuron is added to the hidden layer after each training epoch. The number of neurons is therefore much higher than that of the MLP.

#### **5.4.1.7 Weight and bias initialisation**

All weights and biases are initialised using the Nguyen-Widrow layer initialisation function.

#### **5.4.1.8 Error goal**

The NN is trained until the error goal is reached. There is a trade-off that has to be made between the training to convergence and the generalisation performance of the network. Therefore the error is not set to a minimum value such as zero.

#### **5.4.1.9 Spread constant**

The spread constant denoted by the variance parameter  $\sigma$ , is varied together with the error goal to find the best performance for the given set of inputs. The larger the spread, the smoother the function approximation is. Too large a spread means a lot of neurons are required to fit a fast-changing function. Too small a spread means many neurons are required to fit a smooth function, and the network does not generalize well. The spread constant is varied between 0.1 to 1.0 and the value giving the best predictive performance is chosen.

#### **5.4.1.10 Training Stopping criterion**

Training stops when the error is at a minimum, or the algorithm encounters a minimum gradient, or when the operator manually stops the training phase.

#### **5.4.1.11 Performance measurement**

There are three performance measurements as indicated in section 5.2.1.12. These are the correlation or regression coefficient  $r$ , the coefficient of determination  $r^2$  and the absolute percentage error  $APE$ . The ideal values for  $r$  is as close to 1 as possible,  $r^2$  should ideally be close to 1 or 100%, the  $APE$  should ideally be as small as possible.

### **5.5 Comparison of the advantages and disadvantages of the MLP and RBFNN**

Radial basis function neural networks have become very popular and are serious rivals to the multilayer perceptron. According to Casey (2004):

- RBF NNs train faster than the MLP FFNN;

- Another advantage that is claimed is that the hidden layer for the RBFNN is easier to interpret than the hidden layer for a MLP.
- RBF NNs although they are trained much quicker than a MLP, in actual use it is much slower. Therefore if speed is a consideration, the MLP is faster.

## **5.6 Software implementation**

All software algorithms are implemented using code developed by author, together with the Neural Network Toolbox for use with MATLAB®, Version 4.0.6, (R14SP3). The MATLAB™ version used is MATLAB Version 7.1.0.246 (R14) Service Pack 3.

## **5.7 Summary of the chapter**

This chapter describes the implementation of the three neural networks, namely the MLP, ERNN and the RBFNN. The combination of the variable selection to the input of the NNs are discussed including the conversion of the static properties of a MLP into a dynamic model by introducing time delays at the input. The partitioning of the data into two sets is discussed. The architecture, including the number of hidden layers and the number of hidden neurons are investigated and presented. The incorrect selection of these leads to the problems of overfitting the model and over-training, in the case of increasing the number of epochs. The training algorithms used and the methods for minimizing the mean squared error and the sum of squared error during training are presented. The validation process and the measurement of the NN performance using the regression coefficient, the coefficient of determination, and the absolute percentage error are described for each implemented neural network. The differences between the various types of NNs are investigated and presented.

Chapter 6 provides the results of the implemented neural networks for the influent disturbances of COD, TKN and flow rate. The results of all the different types of NNs are presented together so as to provide the reader with some visual form of comparing network performance.



# CHAPTER SIX

## RESULTS

### 6.1 Introduction

This chapter presents the implementation of the NNs for one step prediction of the influent disturbances with the influent disturbances, weather, and month of the year as input data. The results are presented for three different NN topologies namely, the multilayer perceptron neural network (MLPNN), Elman recurrent neural network (ERNN), and radial basis functions neural network (RBFNN).

There are 13 different input variations to the NNs consisting of the influent variable, delayed influent variable, weather and month of the year data as presented in Chapter 5, Table 5.1, Table 5.2 and Table 5.3. These input variations are specified from Model 1 up to Model 13 with the input variations being the same for the MLP, ERNN and the RBFNN.

Due to the sheer magnitude of the number of available results and the space considerations in this document, only selected results are included in the document. However it should be noted that all results are available for inspection and can be viewed on the enclosed CD.

### 6.2 COD Neural Network Results

The NN presented in this section is for the influent variable COD as the predicted output of the different NNs. The NN model numbers as per the inputs are specified in tables at the beginning of the results for the corresponding model. All the models in this section have only one hidden layer.

#### 6.2.1 COD Neural Network MODEL 1

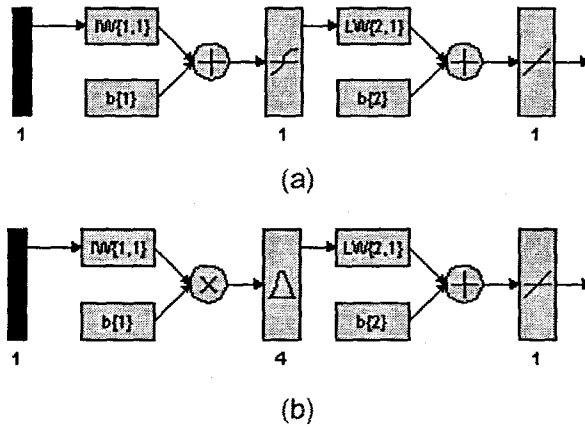
The input for model 1 is COD(k) and the output is COD(k+1) one time step ahead. These are tabled below:

**Table 6. 1: Input variables for COD Model 1**

Inputs	Output
COD(k)	COD(k+1)

The structure of the MLP and RBF NNs for Model 1 are presented in Figure 6.1 below. The figures are generated using the *nntool* command in MATLAB. The practitioner is able to design and view the created NNs using this graphical user

interface (GUI) development environment. This tool is however only used for visualisation of the created NNs. The only NN that cannot be visualised in this design environment is the Elman recurrent NN. The structure of the Elman and all the other NNs are however described in the previous chapter.



**Figure 6. 1:** (a) MLP Feed-forward NN Model 1 with 1 input, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 1 with 1 input, 4 hidden neurons

The results for Model 1 are presented in Table 6.2 up until Table 6.4 for the MLP, Elman and RBF NN respectively, where:

- **Hid layers** refer to the number of hidden layers;
- **Hid neurons** are the number of hidden neurons;
- **Train algorithm** is the training algorithm used;
- **trainscg** is the MATLAB training algorithm used, in this case - the Scaled Conjugate Gradient (Demuth *et al.*, 2004);
- **Hidden activation** is the hidden layer activation function;
- **tansig** is the MATLAB function for the hyperbolic tangent sigmoid activation function in Equation 5.3;
- **Output activation** is the activation function between the hidden layer and output layer;
- **purelin** is the Linear activation function;
- **Epochs** refer to the number of training iterations;
- **Performance** is the error function used;
- **MSE** is the mean squared error cost function in Equation 5.2;
- **SSE** is the sum of squared error function used for the RBF NN;
- **Min train err** is the minimum error value for the mean squared training error;
- **Learn rate** is the rate that determines the length of each step of weight change;
- **Momentum** is the momentum factor;
- **Error goal** is the minimum Sum of squares error goal;
- **Spread** is the constant for Gaussian function used in the RBF NN;
- **r\_train** is the training correlation coefficient;
- **r\_test** is the correlation coefficient after a test (validation) set of data is applied;
- **A** is the NN output;
- **T** is the desired target output;
- **R\_sq\_train** is the determination coefficient for the training phase;

- *R\_sq\_test* is the determination coefficient for the testing phase;
- *Avg\_test\_err* is the instantaneous error between the target output and the predicted output of the NN; and
- *APE* is the absolute percentage error

The definitions above pertain to all the other tables described later in this chapter. As is seen from the tables, for the MLP and the ERNN the number of neurons in the hidden layer are varied from 1 to 5 and finally to 10 neurons. The number of training iterations (epochs) is varied from 500 to a maximum of 1000. The activation function for the hidden layer is the hyperbolic tangent for the MLP and Elman RNN, and for the RBF NN the Gaussian radial basis function is used. Training for the MLP and ERNN stops when the mean squared error goal is reached, or when the maximum number of epochs is reached. For the RBF training stops when the sum of square error reaches a minimum or when the training iterations equal the maximum number of input data points.

For the NN having TKN as an output in section 6.3, all tables and abbreviations remain the same as per the section above for the NN with COD as output. The inputs are specified as per Table 5.2. For space considerations only for Model 1 all figures are included, the rest of the models only have the final NN response to the application of the test data set. However, please note that all results can be viewed using the program on the enclosed compact disc (CD).

From Figure 6.2 up to Figure 6.7 the training, the testing (validation), and performance measurement phases are displayed.

**Table 6. 2: Results for the MLP Feed-forward Neural Network Model 1 with COD as NN output.**

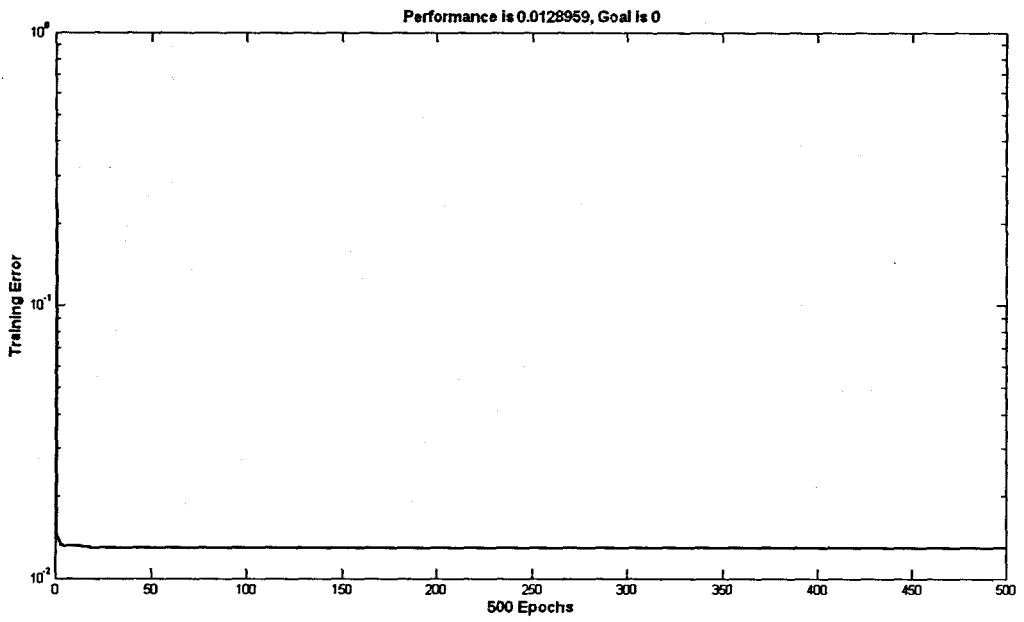
Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0129	0.6	0.5	0.331	0.243	0.110	0.059	-30.239	20.292
1	5	trainscg	tansig	purelin	500	MSE	0.0124	0.6	0.5	0.375	0.223	0.141	0.050	-32.589	20.938
1	10	trainscg	tansig	purelin	500	MSE	0.0119	0.6	0.5	0.419	0.249	0.176	0.062	-27.507	20.752
1	1	trainscg	tansig	purelin	993	MSE	0.0129	0.6	0.5	0.331	0.243	0.110	0.059	-30.452	20.299
1	5	trainscg	tansig	purelin	1000	MSE	0.0123	0.6	0.5	0.389	0.214	0.151	0.046	-29.936	21.029
1	10	trainscg	tansig	purelin	1000	MSE	0.0119	0.6	0.5	0.425	0.248	0.181	0.062	-27.081	20.691

**Table 6. 3: Results for the Elman Recurrent Neural Network Model 1 with COD as NN output.**

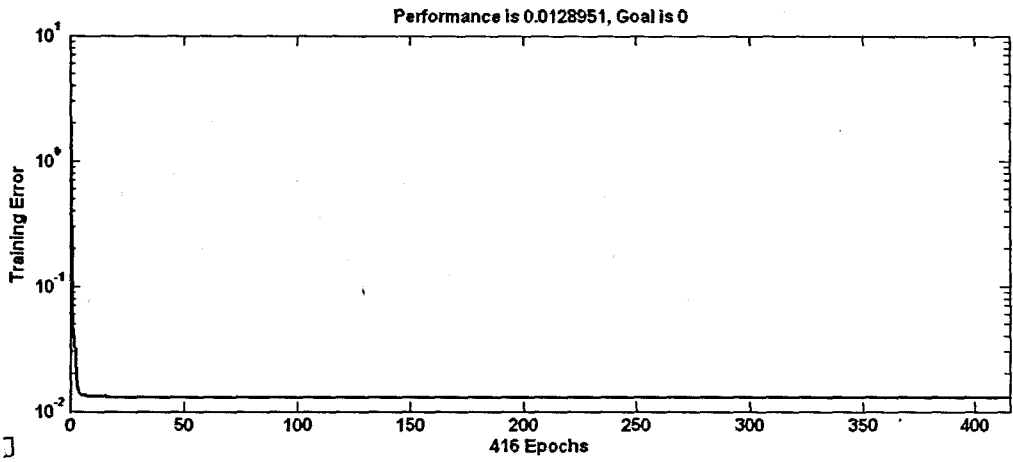
Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0129	0.01	0.5	0.330	0.242	0.109	0.058	-29.475	20.268
1	5	trainscg	tansig	purelin	500	MSE	0.0126	0.01	0.5	0.359	0.246	0.129	0.061	-33.176	20.430
1	10	trainscg	tansig	purelin	500	MSE	0.0126	0.01	0.5	0.357	0.246	0.128	0.060	-34.451	20.418
1	1	trainscg	tansig	purelin	158	MSE	0.0129	0.01	0.5	0.330	0.240	0.109	0.057	-28.830	20.262
1	5	trainscg	tansig	purelin	1000	MSE	0.0126	0.01	0.5	0.362	0.246	0.131	0.061	-34.307	20.554
1	10	trainscg	tansig	purelin	1000	MSE	0.0127	0.01	0.5	0.353	0.245	0.125	0.060	-31.400	20.353

**Table 6.4: Results for the Radial Basis Function Neural Network Model 1 with COD as NN output.**

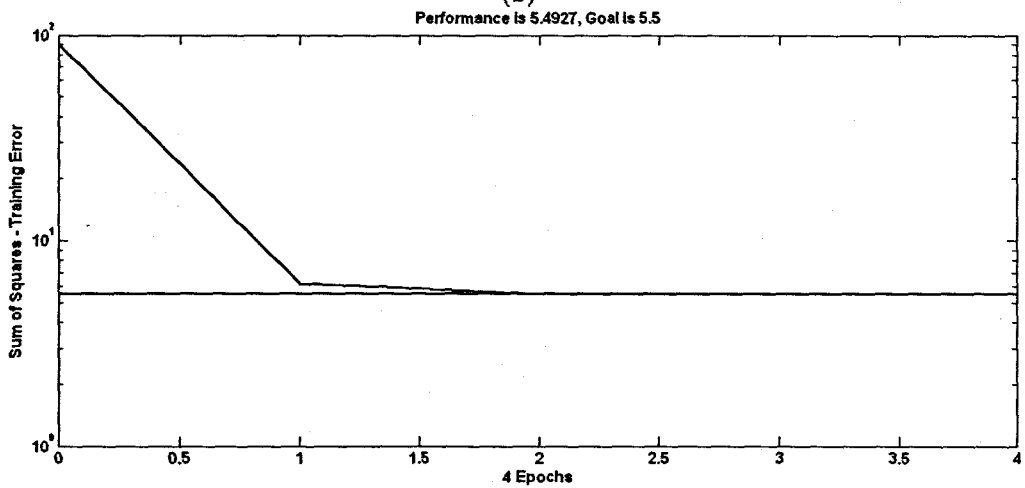
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	4	4	SSE	5.493	5.5	0.2	0.361	0.248	0.130	0.061	-33.642	20.584



(a)

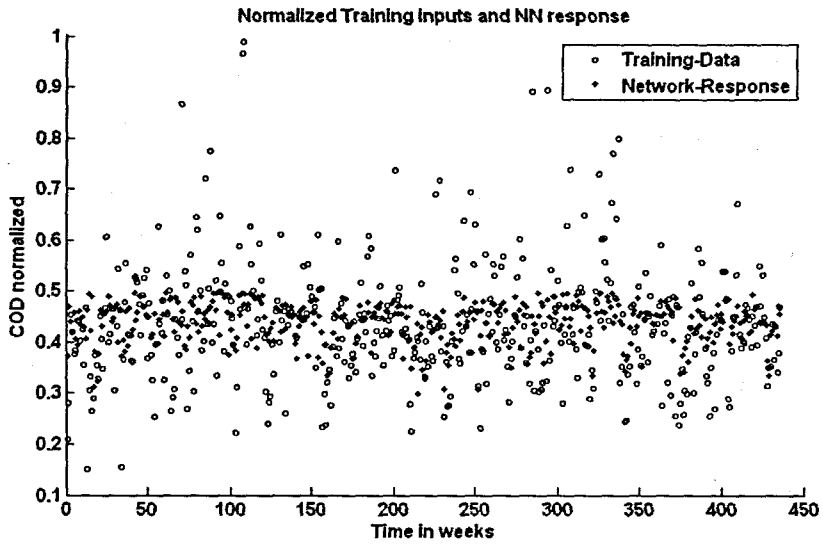


(b)

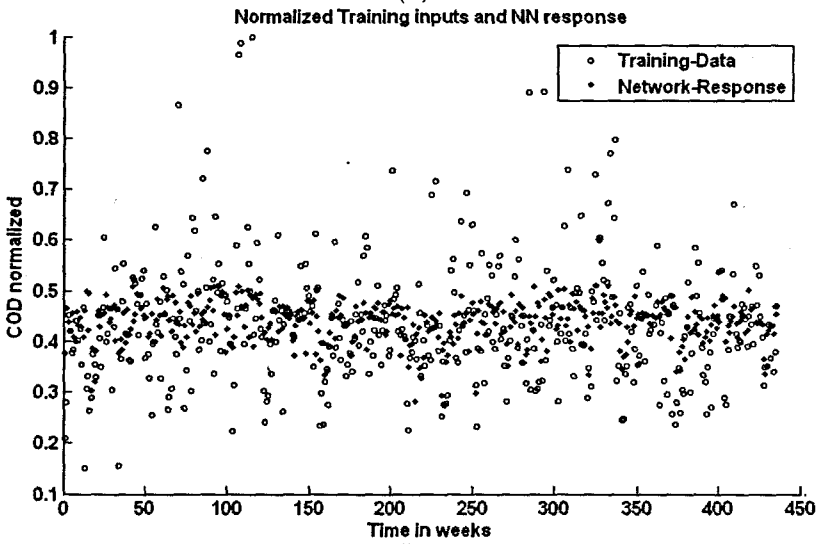


(c)

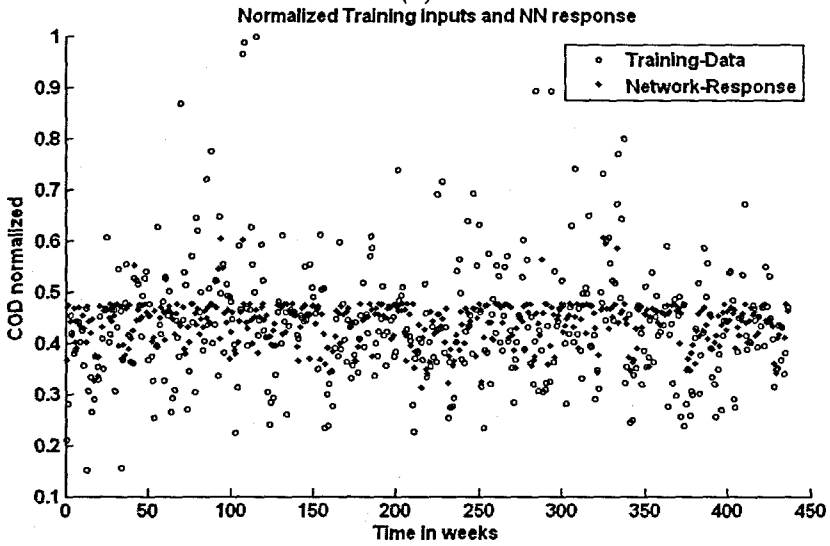
**Figure 6. 2:** Training error versus the number of epochs for COD: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs



(a)

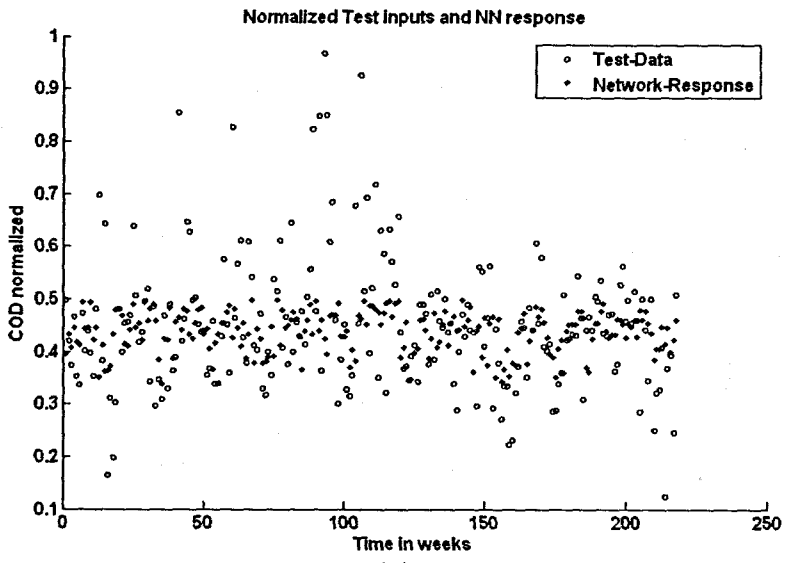


(b)

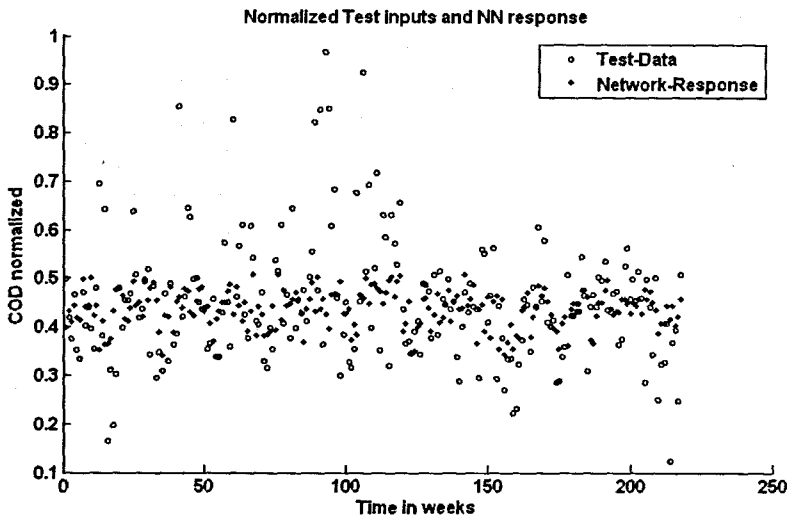


(c)

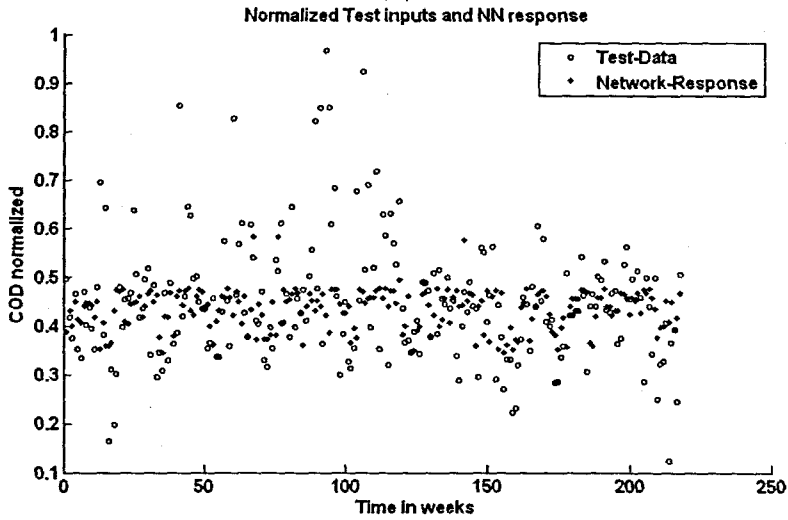
**Figure 6. 3:** A scatter-plot of the training input data and the NN response with the application of the training set for the COD: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs



(a)

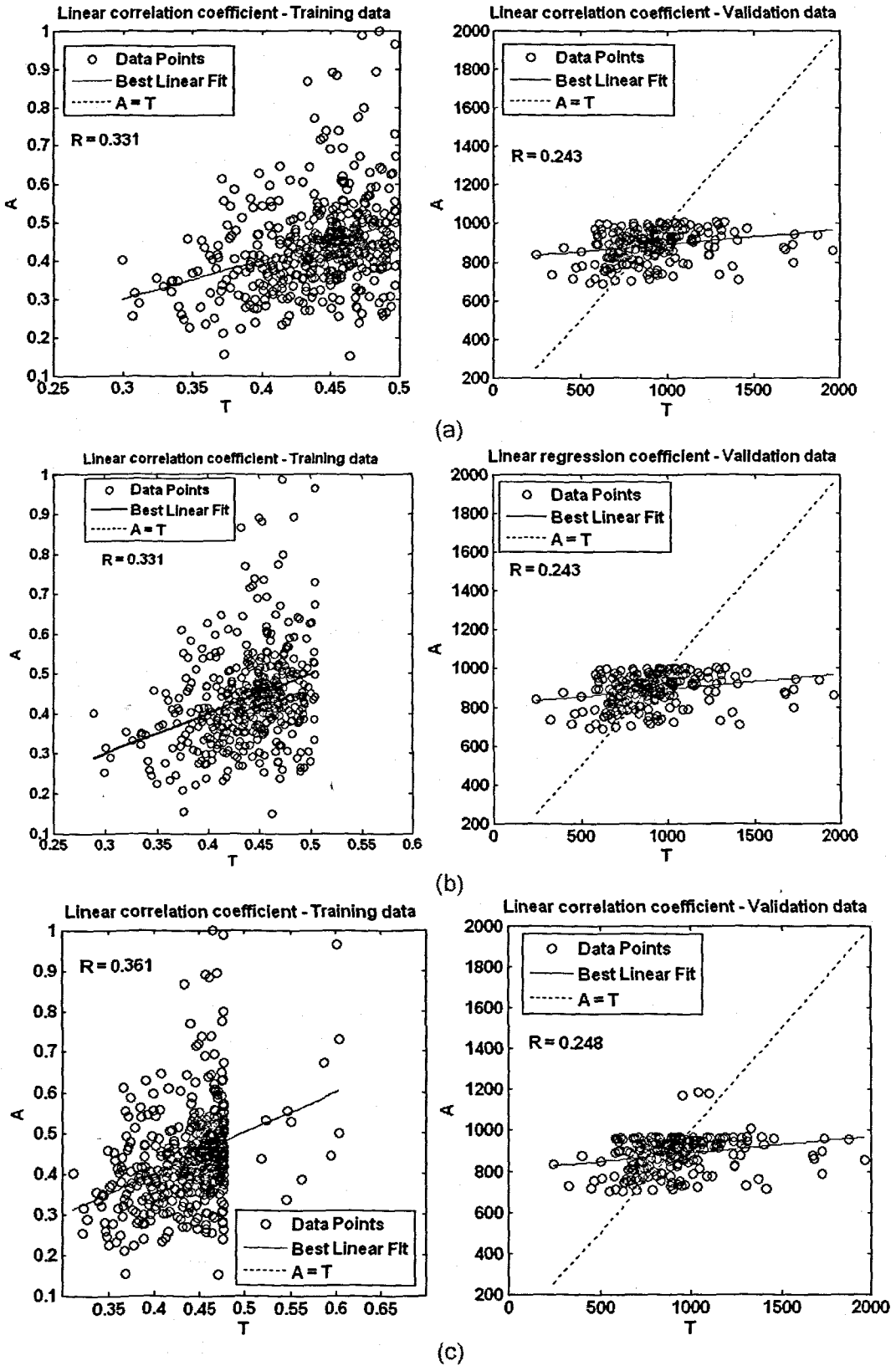


(b)



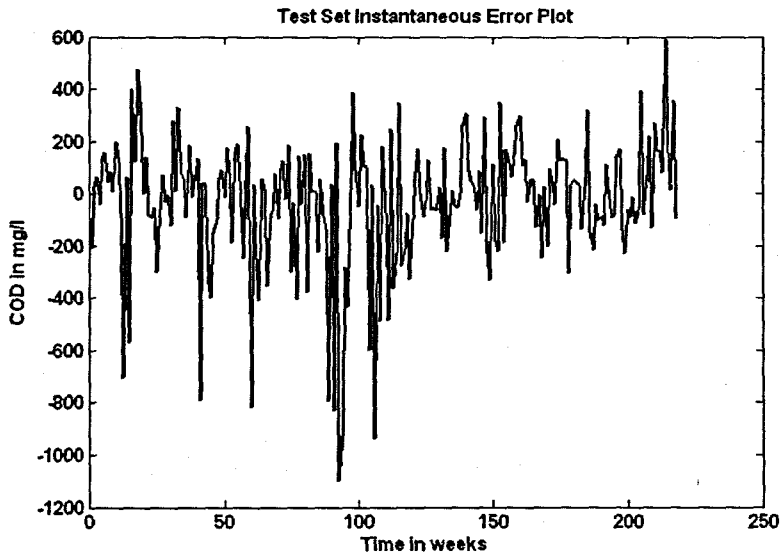
(c)

Figure 6. 4: A scatter-plot of the test (validation) data and the NN response with the application of the test set for the COD: (a) FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs

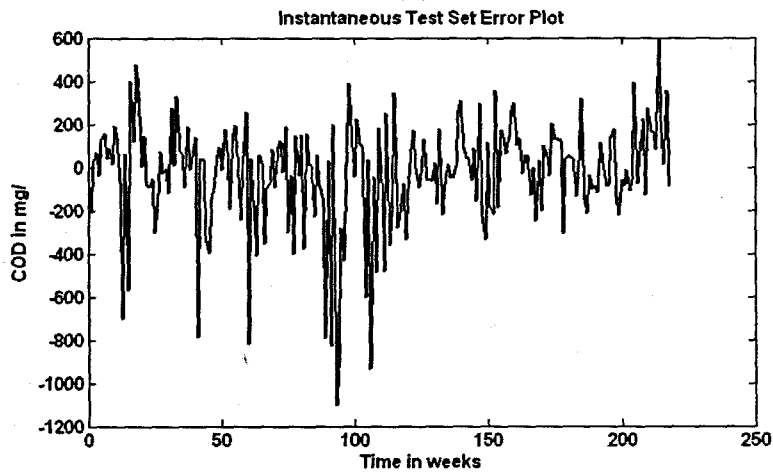


**Figure 6. 5:** Network performance showing the linear regression (correlation) coefficients for training data (normalized) on the left, and the validation data set (denormalized) on the right for the COD: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs, where A and T are the NN output and desired target output respectively

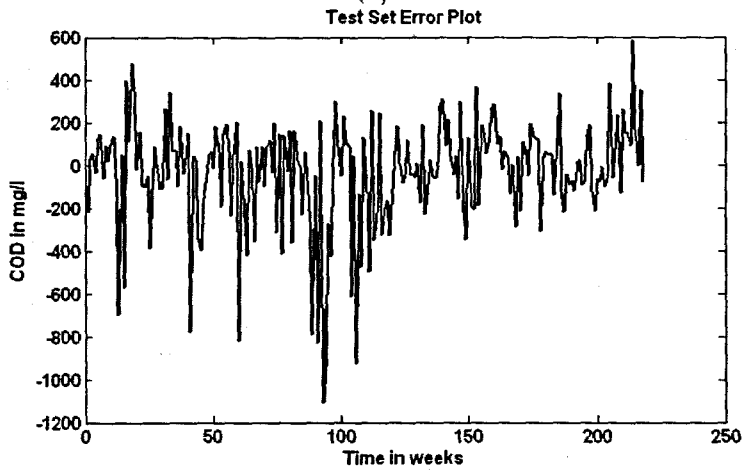




(a)



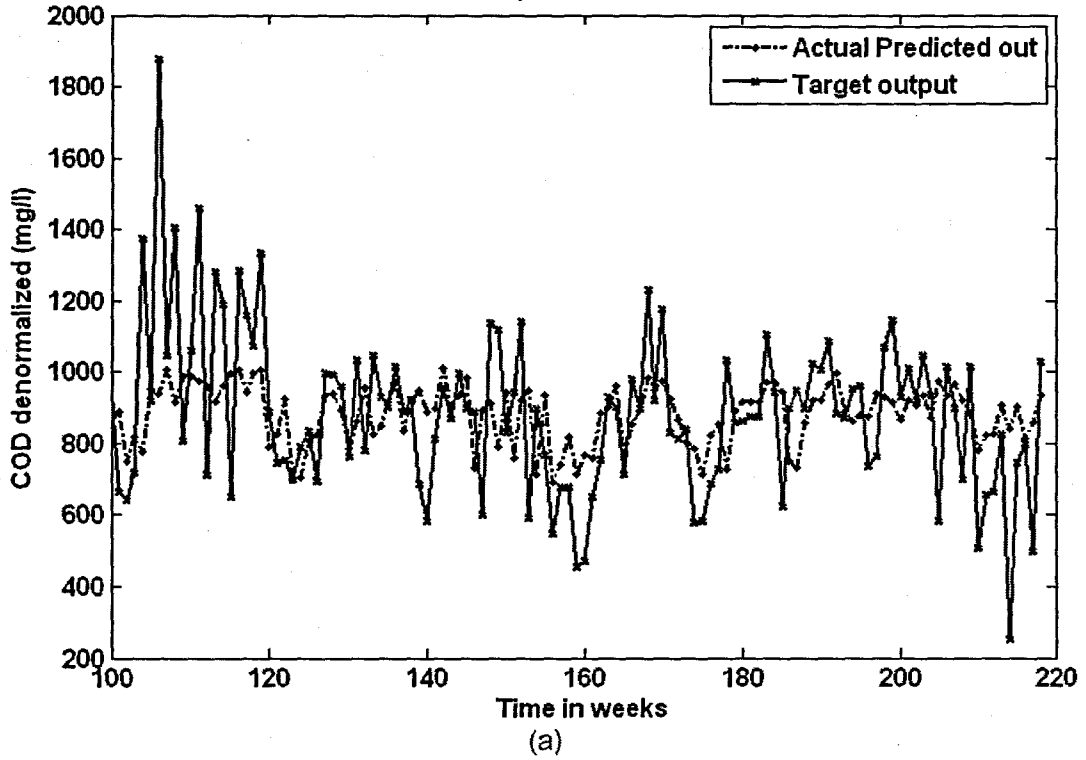
(b)



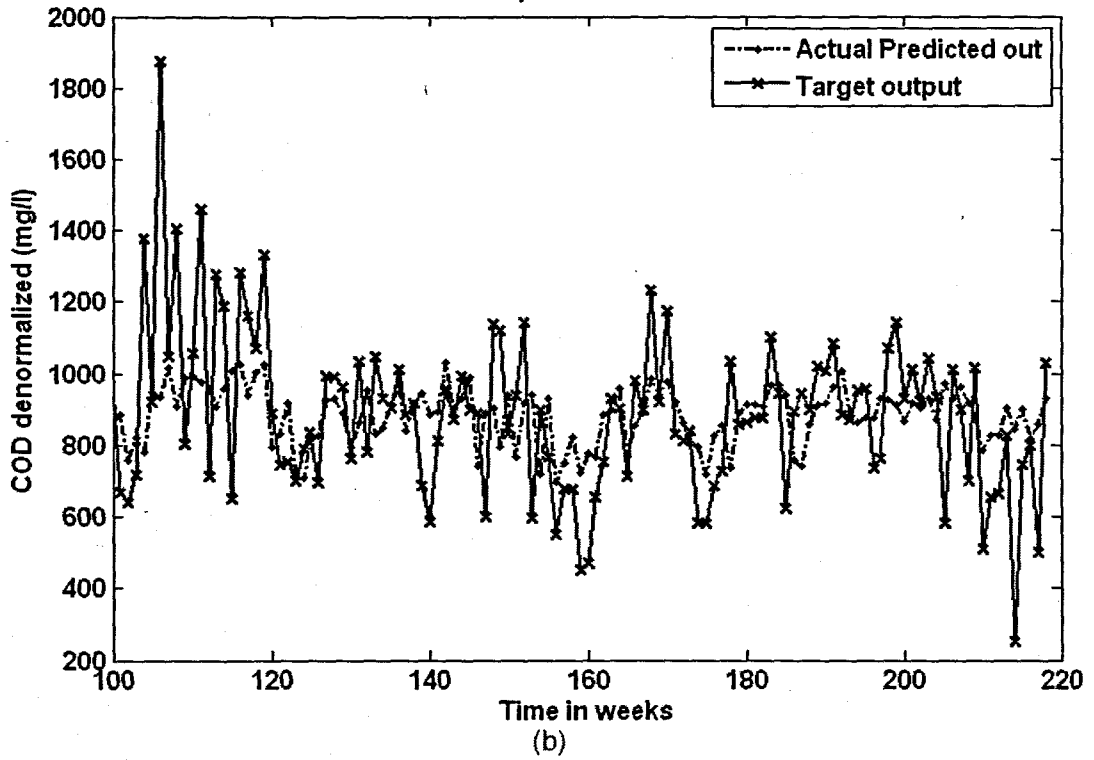
(c)

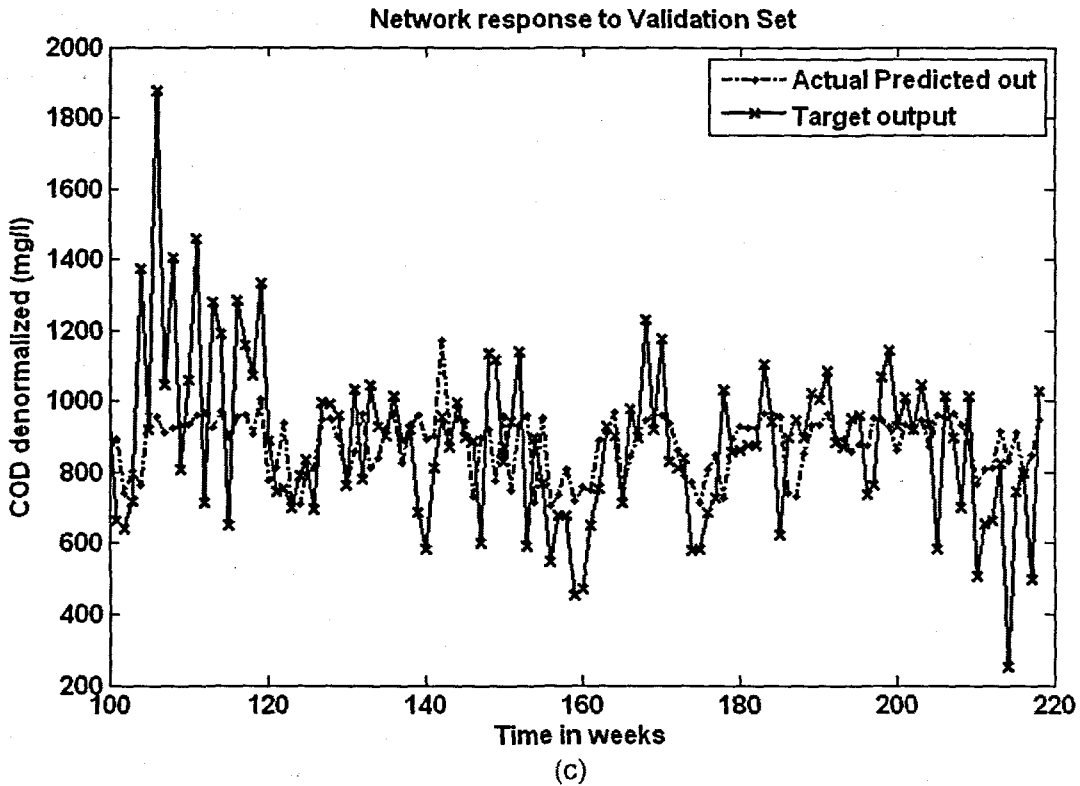
**Figure 6. 6:** The instantaneous error across the entire validation data set for the COD: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs

Network response to Validation Set



Network response to Validation Set





**Figure 6. 7:** The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 4 hidden neurons, 4 epochs

An examination of Figure 6.2 (a) and (b), shows the training error for the MLP and ERNN drops to a value of 0.01. However although the training error has converged to a minimum, the scatter plots for the training data and target output displayed in Figure 6.3 (a) and (b) reveal that the NN response to the application of the TRAINING data set is not good, and the regression analysis in Figure 6.5 (a) and (b) shows the training regression coefficient is only 0.331 for both the MLP and the ERNN. The same can be said about the RBFNN. The regression coefficient for the RBFNN is only slightly higher at 0.361. The linear regression coefficient for the TEST phase also shows that the NN is not able to generalize well, with the coefficients being 0.243, 0.243 and 0.248 for the MLP, ERNN and RBF respectively. The instantaneous error between the test data and the NN output for each point across the test set are displayed in Figure 6.6. Finally the plots for the target and NN output are displayed in Figure 6.7.

The results show that although three different architectures are used, the prediction performance of all three neural networks is very similar.

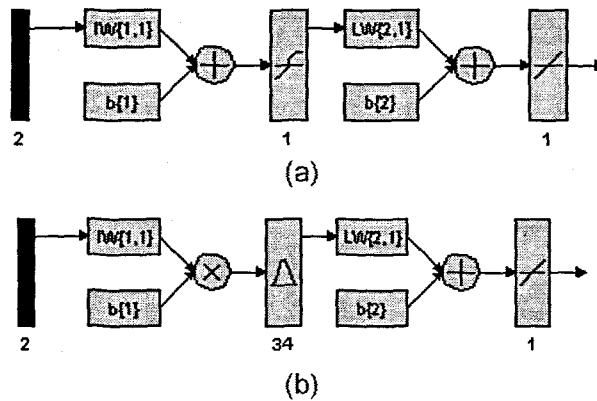
## 6.2.2 COD Neural Network MODEL 2

The inputs for Model 2 are COD(k) and COD(k-1) and the output is COD(k+1) one time step ahead. These are tabled below:

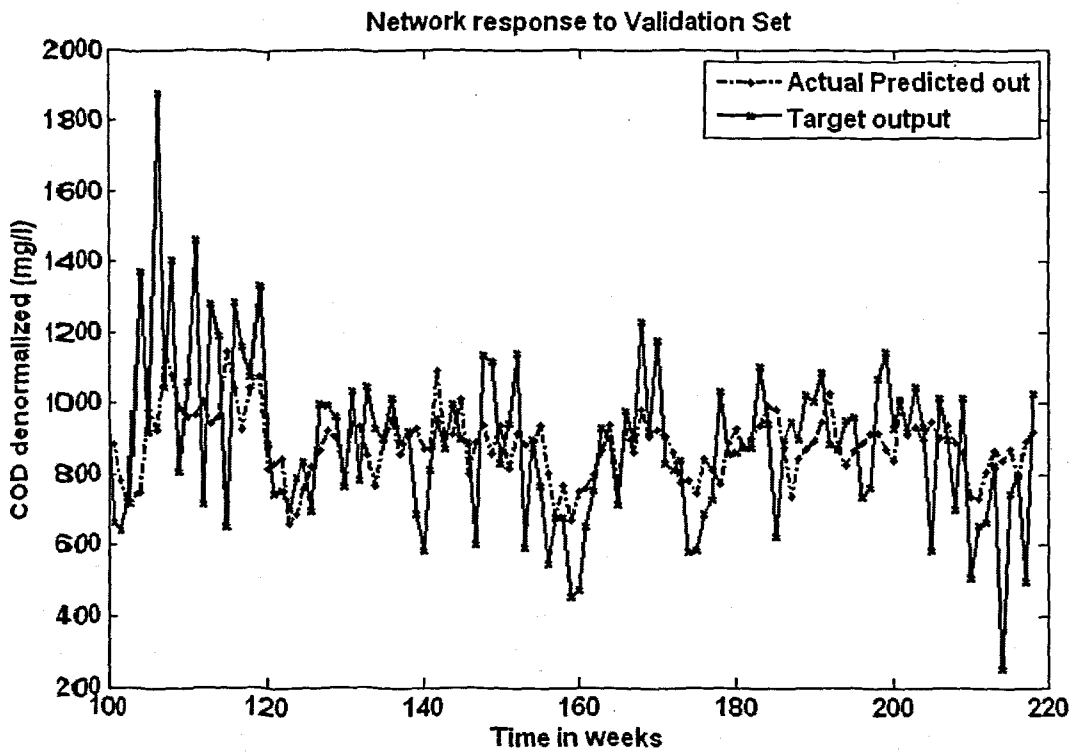
**Table 6. 5: Input variables for COD Model 2**

Inputs	Output
COD(k), COD(k-1)	COD(k+1)

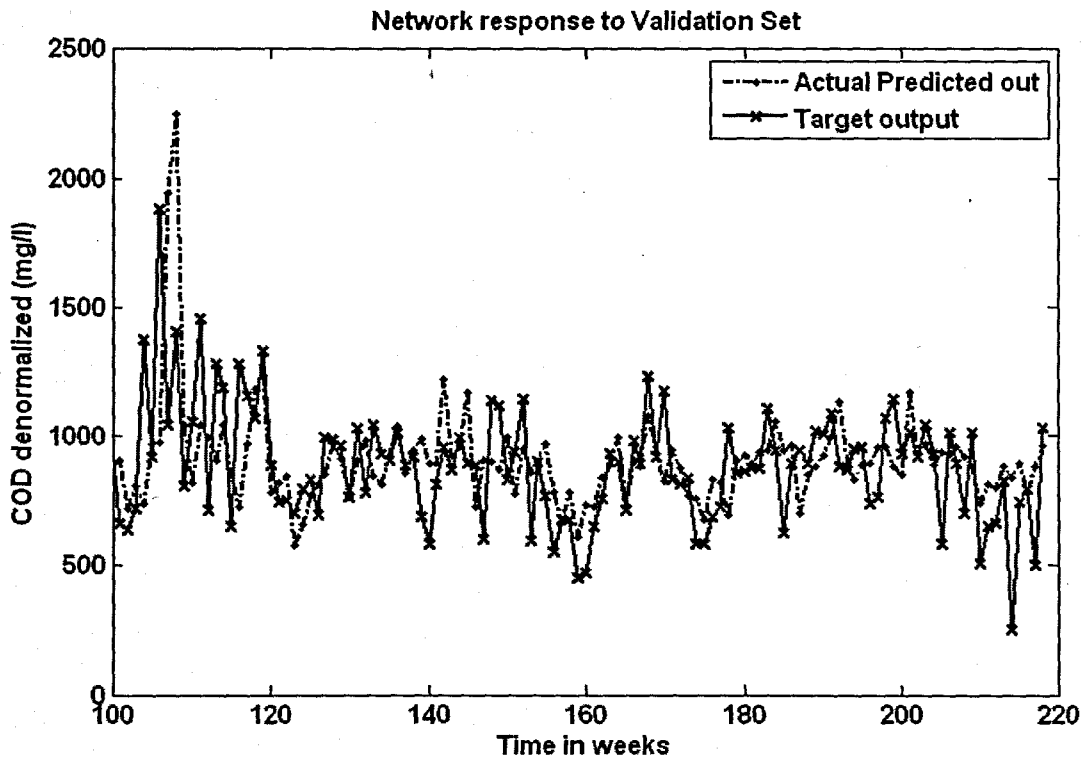
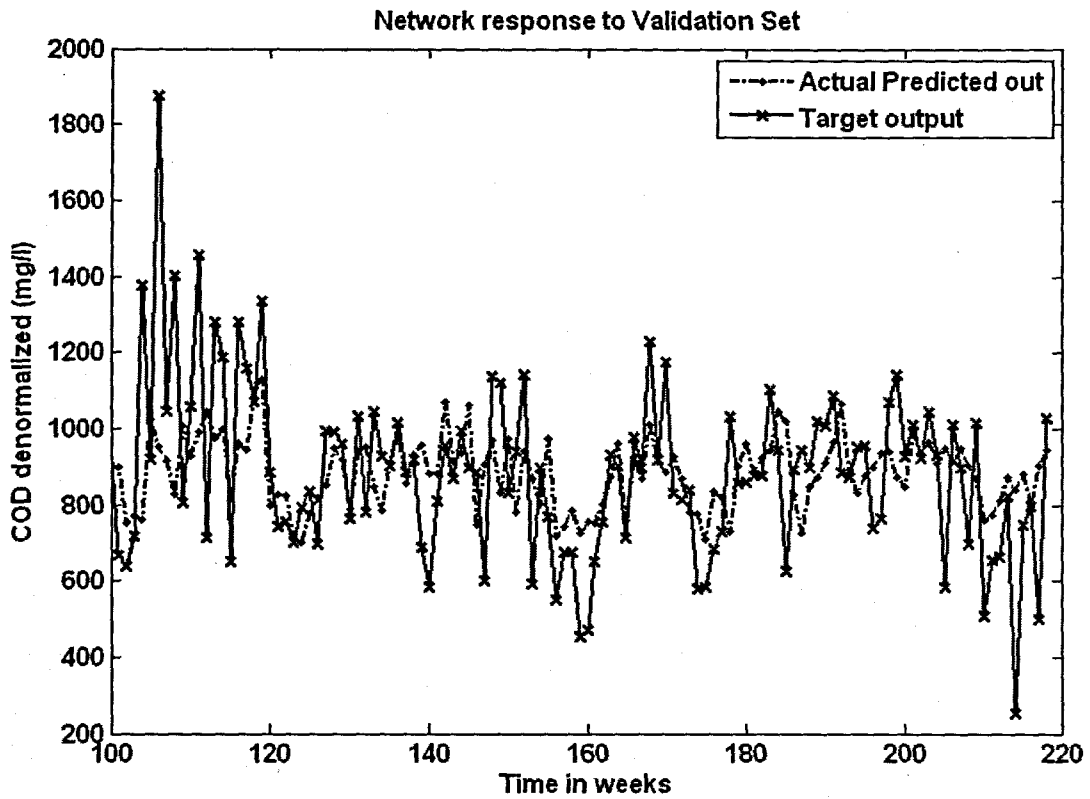
The structure of the MLP and RBF NNs for Model 2 are presented in Figure 6.8 below.



**Figure 6. 8:** (a) A MLP Feed-forward neural network Model 2 with 2 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; (b) Block Diagram for the COD RBFNN Model 2 with 34 hidden neurons



(a)



**Figure 6. 9:** The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 2 with 1 hidden neuron and 500 epochs; (b) ERNN Model 2 with 10 hidden neurons, 500 epochs; and (c) RBFNN Model 2 with 34 hidden neurons, 34 epochs

**Table 6. 6: Results for the Feed-forward Neural Network Model 2 with COD as NN output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0124	0.6	0.5	0.376	0.278	0.141	0.077	-31.271	19.936
1	5	trainscg	tansig	purelin	500	MSE	0.0119	0.6	0.5	0.420	0.253	0.177	0.064	-28.459	20.348
1	10	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.442	0.219	0.195	0.0482	-26.242	20.664
1	1	trainscg	tansig	purelin	1000	MSE	0.012	0.6	0.5	0.376	0.278	0.141	0.077	-31.259	19.940
1	5	trainscg	tansig	purelin	1000	MSE	0.0117	0.6	0.5	0.442	0.250	0.195	0.062	-32.254	20.723
1	10	trainscg	tansig	purelin	1000	MSE	0.0113	0.6	0.5	0.467	0.253	0.218	0.064	-24.468	20.853

**Table 6. 7: Results for the Elman Recurrent Neural Network Model 2 with COD as NN output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0124	0.01	0.5	0.376	0.278	0.141	0.077	-31.269	19.939
1	5	trainscg	tansig	purelin	500	MSE	0.0120	0.01	0.5	0.416	0.269	0.173	0.072	-25.393	20.281
1	10	trainscg	tansig	purelin	500	MSE	0.0119	0.01	0.5	0.424	0.281	0.180	0.079	-26.551	20.135
1	1	trainscg	tansig	purelin	1000	MSE	0.0120	0.01	0.5	0.376	0.278	0.141	0.077	-31.244	19.938
1	5	trainscg	tansig	purelin	1000	MSE	0.0120	0.01	0.5	0.418	0.266	0.175	0.071	-25.509	20.314
1	10	trainscg	tansig	purelin	1000	MSE	0.0119	0.01	0.5	0.422	0.280	0.178	0.078	-26.039	20.120

**Table 6. 8: Results for the Radial Basis Function Neural Network Model 2 with COD as NN output.**

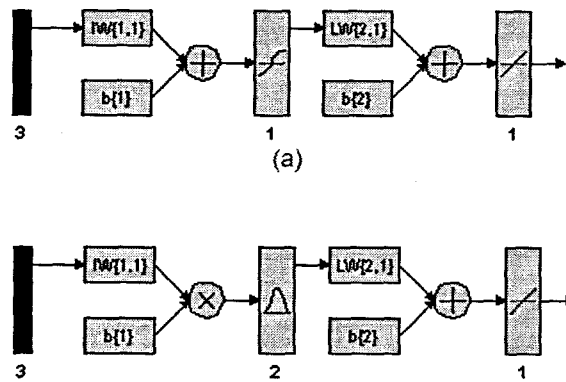
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	34	34	SSE	4.782	4.8	0.3	0.493	0.226	0.243	0.051	-14.390	22.633

### 6.2.3 COD Neural Network MODEL 3

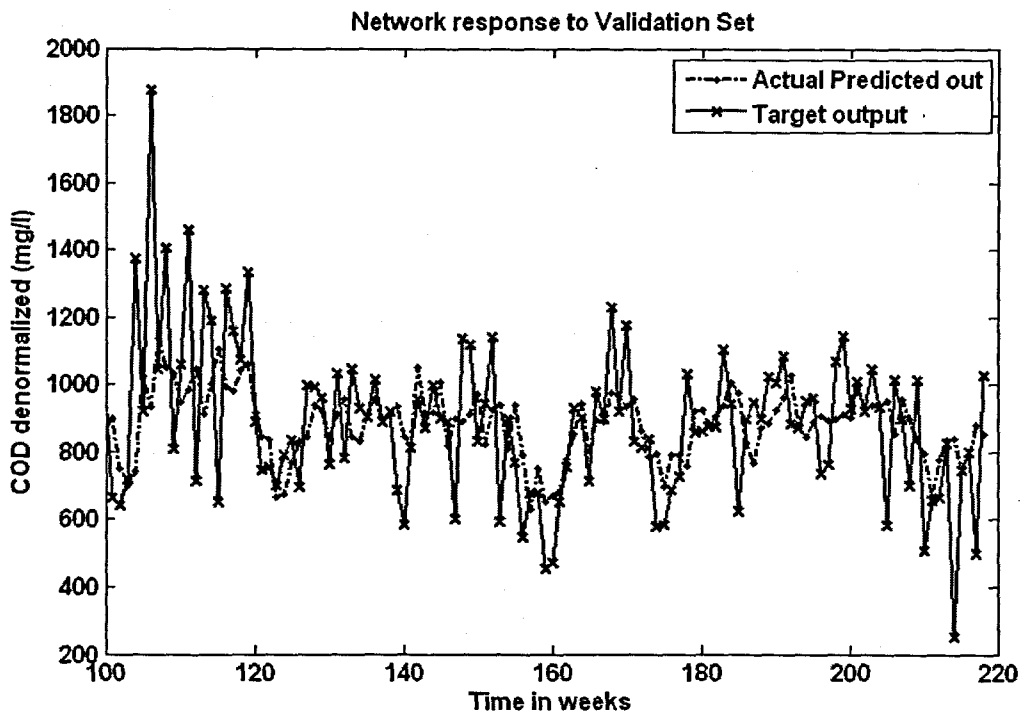
The inputs for Model 3 are  $COD(k)$ ,  $COD(k-1)$ ,  $COD(k-2)$ , and the output is  $COD(k+1)$  one time step ahead. These are tabled below:

**Table 6. 9: Input variables for COD Model 3**

Inputs	Output
$COD(k)$ , $COD(k-1)$ , $COD(k-2)$	$COD(k+1)$



**Figure 6. 10: (a) A MLP Feed-forward neural network Model 3 with 3 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 3 with 2 hidden neurons, 2 epochs**



(a)

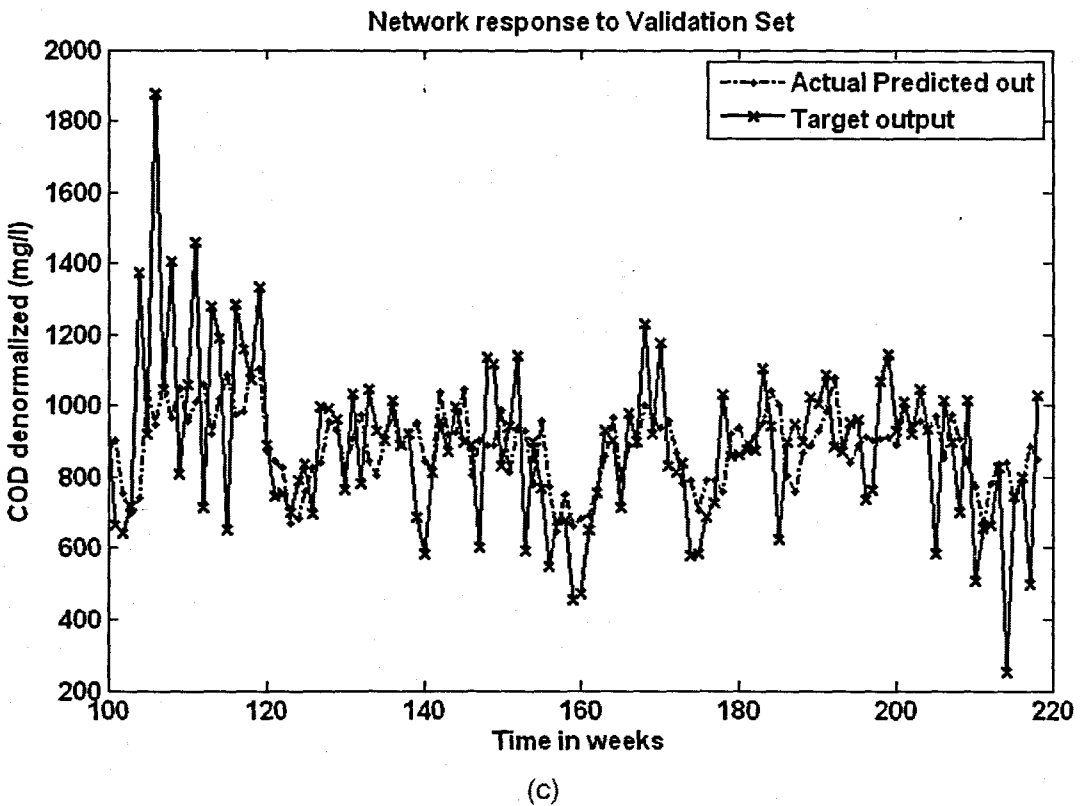
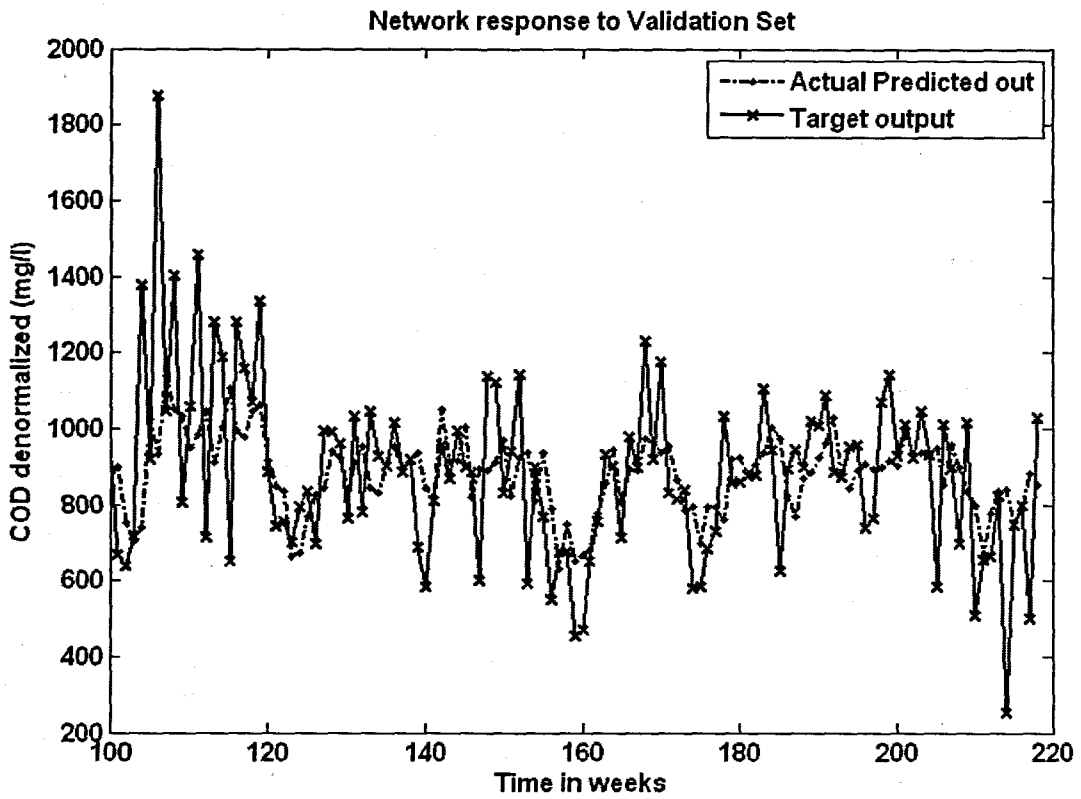


Figure 6. 11: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 3 with 1 hidden neuron and 500 epochs; (b) ERNN Model 3 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 3 with 2 hidden neurons, 2 epochs



**Table 6. 10: Results for the Feed-forward Neural Network Model 3 with COD as NN output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0122	0.6	0.5	0.400	0.315	0.160	0.099	-26.593	19.608
1	5	trainscg	tansig	purelin	500	MSE	0.0116	0.6	0.5	0.449	0.303	0.202	0.092	-29.764	19.785
1	10	trainscg	tansig	purelin	500	MSE	0.0110	0.6	0.5	0.489	0.211	0.239	0.045	-20.808	20.888
1	1	trainscg	tansig	purelin	953	MSE	0.0122	0.6	0.5	0.400	0.314	0.160	0.099	-26.587	19.616
1	5	trainscg	tansig	purelin	1000	MSE	0.0114	0.6	0.5	0.463	0.297	0.215	0.088	-21.007	20.000
1	10	trainscg	tansig	purelin	1000	MSE	0.0104	0.6	0.5	0.528	0.249	0.279	0.062	-16.545	20.506

**Table 6. 11: Results for the Elman Recurrent Neural Network Model 3 with COD as NN output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0122	0.01	0.5	0.400	0.314	0.160	0.099	-26.606	19.610
1	5	trainscg	tansig	purelin	500	MSE	0.0117	0.01	0.5	0.439	0.303	0.192	0.092	-25.077	19.814
1	10	trainscg	tansig	purelin	500	MSE	0.0114	0.01	0.5	0.458	0.245	0.210	0.060	-20.460	20.391
1	1	trainscg	tansig	purelin	159	MSE	0.0122	0.01	0.5	0.400	0.314	0.160	0.098	-26.603	19.620
1	5	trainscg	tansig	purelin	1000	MSE	0.0111	0.01	0.5	0.484	0.242	0.234	0.058	-16.592	20.366
1	10	trainscg	tansig	purelin	1000	MSE	0.0111	0.01	0.5	0.480	0.225	0.231	0.050	-19.581	20.483

**Table 6. 12: Results for the Radial Basis Function Neural Network Model 3 with COD as NN output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	2	2	SSE	5.248	5.8	1	0.411	0.289	0.169	0.084	-26.881	19.826

### 6.2.4 COD Neural Network MODEL 4

The inputs for Model 4 are COD(k), COD(k-1), COD(k-2), COD(k-3), and the output is COD(k+1) one time step ahead. These are tabled below:

Table 6. 13: Input variables for COD Model 4

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3)	COD(k+1)

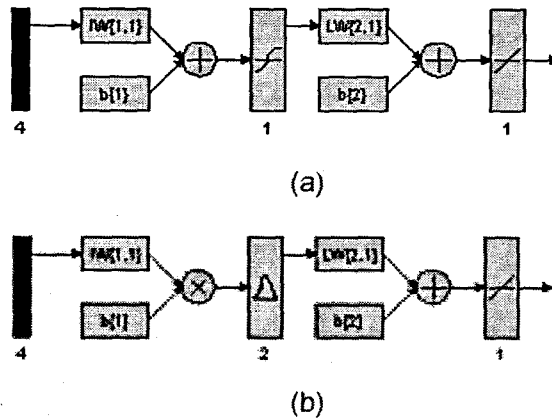
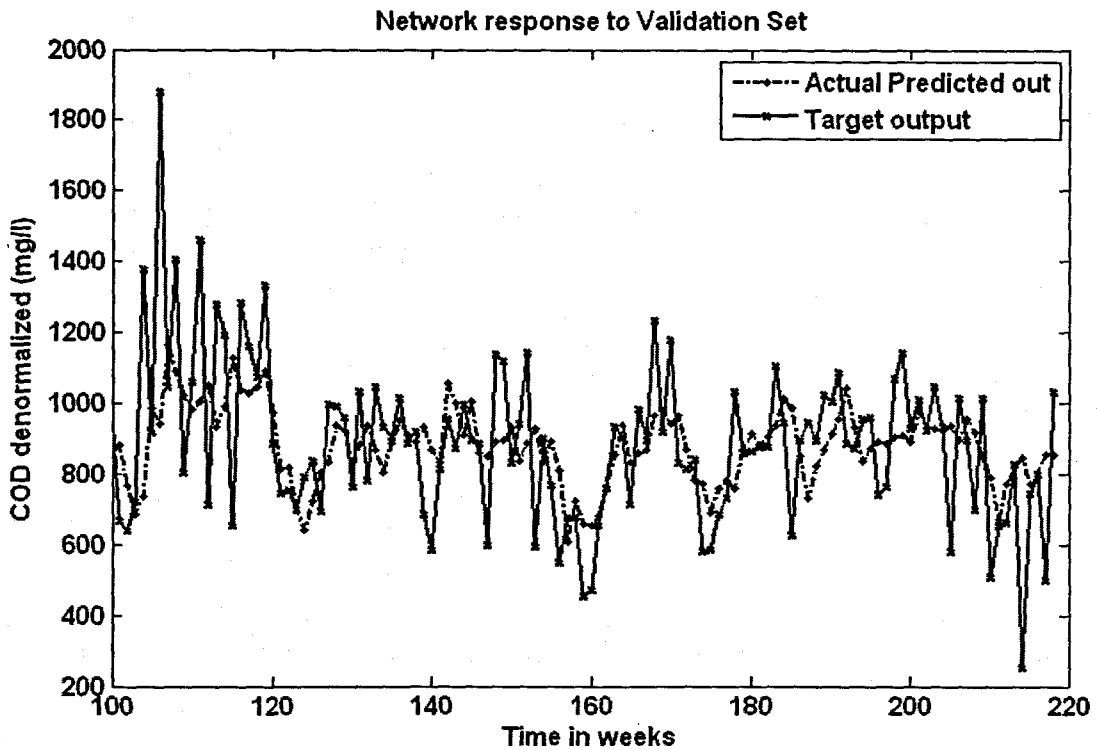


Figure 6. 12: (a) A MLP Feed-forward neural network Model 4 with 4 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 4 with 2 hidden neurons, 2 epochs



(a)

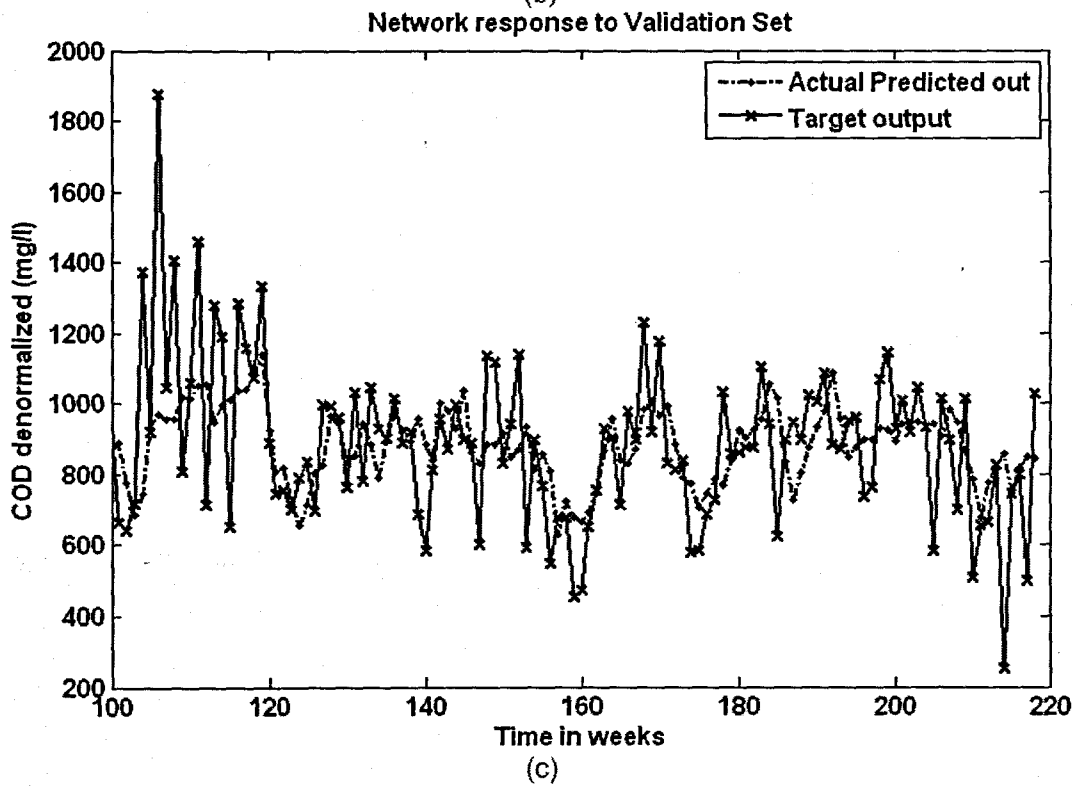
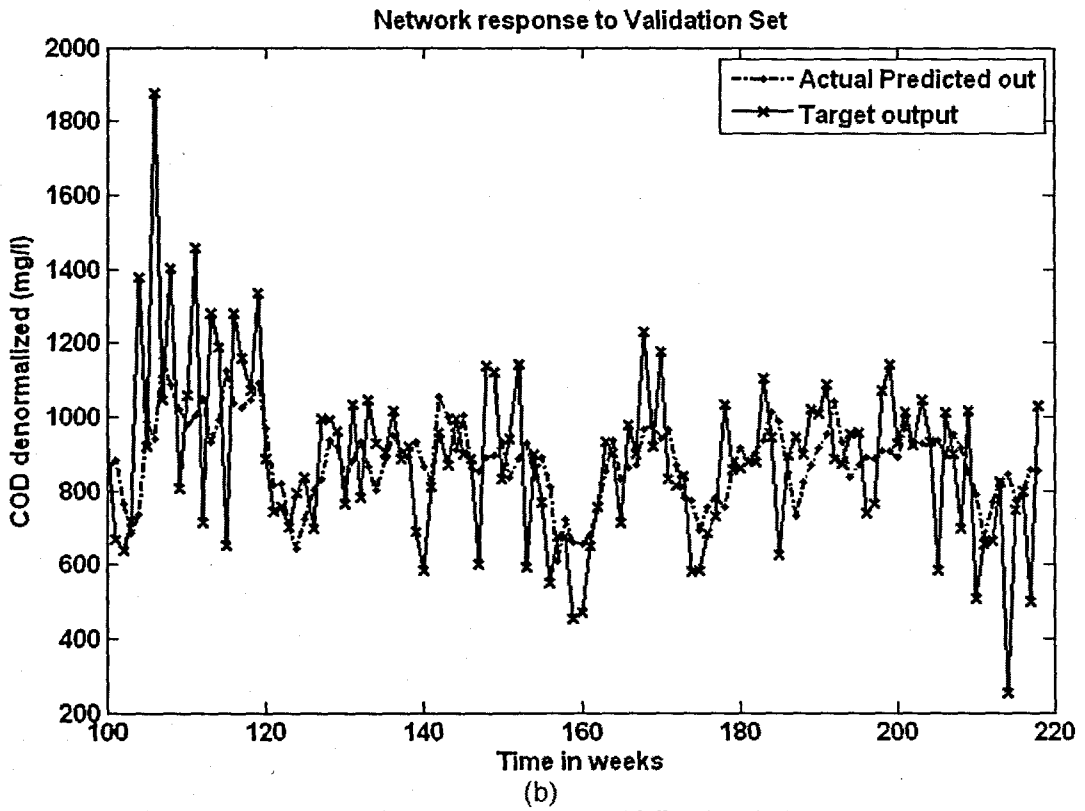


Figure 6. 13: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 4 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 4 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 4 with 2 hidden neurons, 2 epochs

**Table 6. 14: Results for the Feed-forward Neural Network Model 4 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0120	0.6	0.5	0.414	0.358	0.172	0.128	-28.174	19.361
1	5	trainscg	tansig	purelin	500	MSE	0.0111	0.6	0.5	0.486	0.290	0.236	0.084	-27.574	20.111
1	10	trainscg	tansig	purelin	500	MSE	0.0105	0.6	0.5	0.522	0.285	0.272	0.081	-24.997	19.835
1	1	trainscg	tansig	purelin	1000	MSE	0.0120	0.6	0.5	0.415	0.358	0.172	0.128	-28.169	19.365
1	5	trainscg	tansig	purelin	1000	MSE	0.0108	0.6	0.5	0.503	0.302	0.253	0.091	-32.342	19.607
1	10	trainscg	tansig	purelin	1000	MSE	0.0105	0.6	0.5	0.525	0.221	0.276	0.049	-24.614	20.502

**Table 6. 15: Results for the Elman Recurrent Neural Network Model 4 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0120	0.01	0.5	0.414	0.358	0.172	0.128	-28.178	19.363
1	5	trainscg	tansig	purelin	500	MSE	0.0114	0.01	0.5	0.461	0.328	0.213	0.108	-25.476	19.674
1	10	trainscg	tansig	purelin	500	MSE	0.0108	0.01	0.5	0.502	0.272	0.252	0.074	-29.024	20.405
1	1	trainscg	tansig	purelin	1000	MSE	0.0120	0.01	0.5	0.415	0.358	0.172	0.128	-28.153	19.365
1	5	trainscg	tansig	purelin	1000	MSE	0.0107	0.01	0.5	0.510	0.278	0.261	0.077	-28.513	20.118
1	10	trainscg	tansig	purelin	1000	MSE	0.0106	0.01	0.5	0.519	0.264	0.270	0.070	-27.675	20.179

**Table 6. 16: Results for the Radial Basis Function Neural Network Model 4 with COD as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	2	2	SSE	5.281	5.5	0.9	0.405	0.343	0.164	0.117	-28.891	19.404

### 6.2.5 COD Neural Network MODEL 5

The inputs for Model 5 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k) and the output is COD(k+1) one time step ahead. These are tabled below:

Table 6. 17: Input variables for COD Model 5

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k)	COD(k+1)

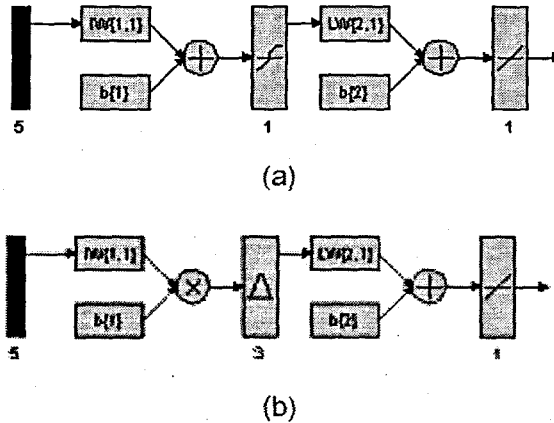
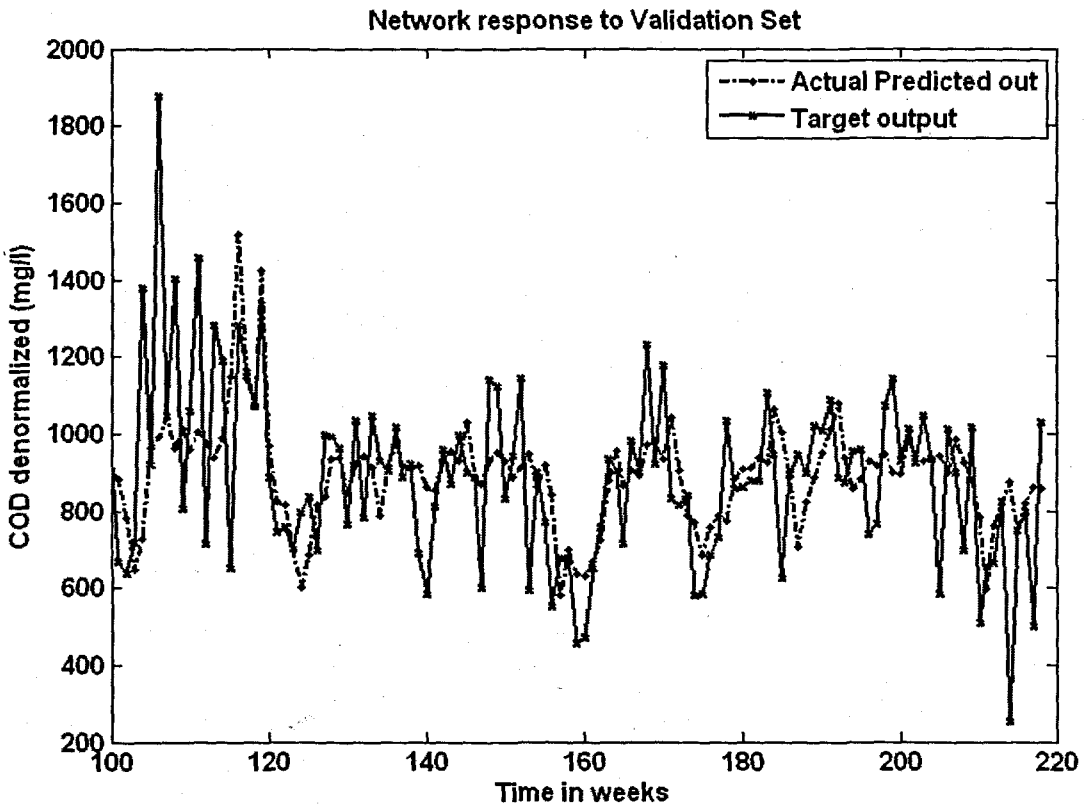


Figure 6. 14: (a) A MLP Feed-forward neural network Model 5 with 5 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 5 with 3 hidden neurons, 3 epochs



(a)

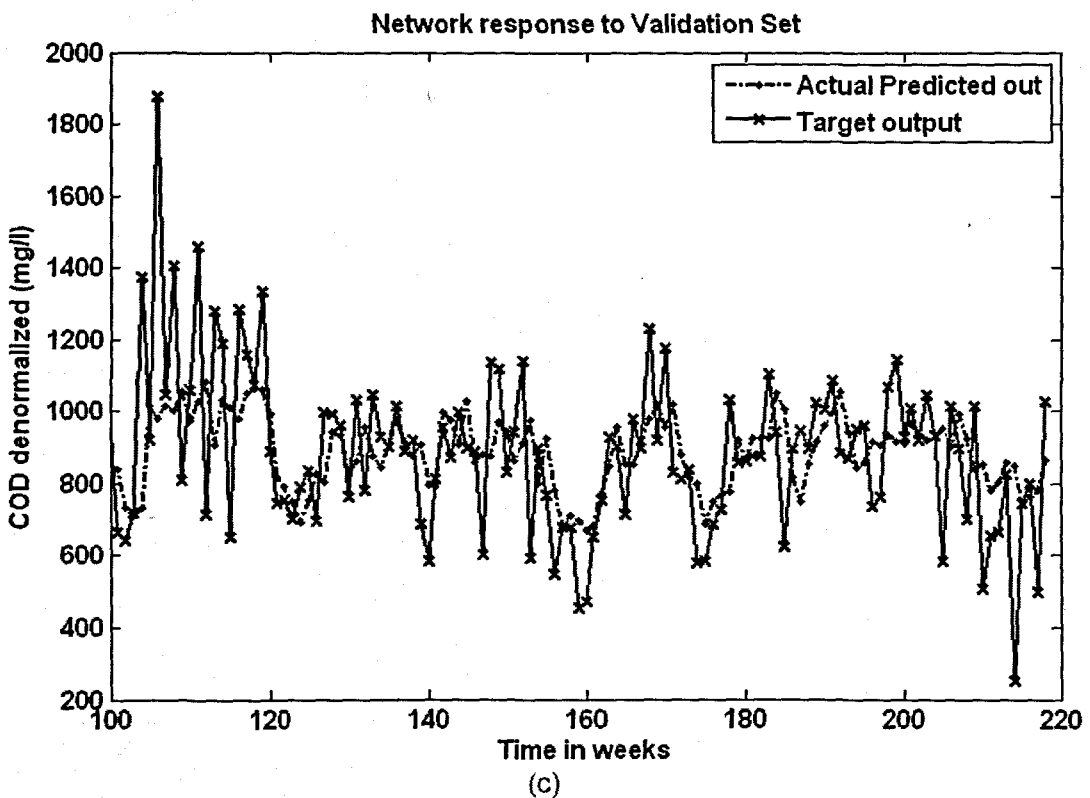
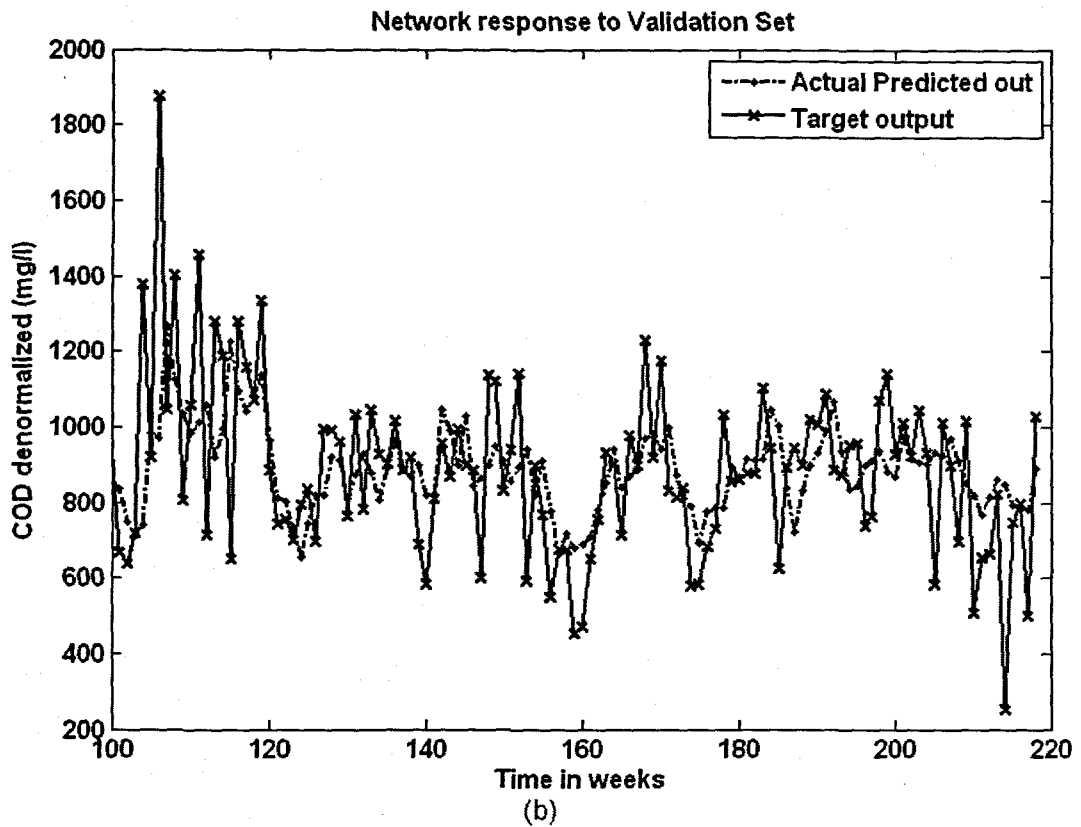


Figure 6. 15: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 5 with 5 hidden neurons and 500 epochs; (b) ERNN Model 5 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 5 with 3 hidden neurons, 3 epochs

**Table 6. 18: Results for the Feed-forward Neural Network Model 5 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r train	r test	R sq train	R sq test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0119	0.6	0.5	0.421	0.356	0.177	0.127	-30.832	19.456
1	5	trainscg	tansig	purelin	500	MSE	0.0107	0.6	0.5	0.512	0.358	0.262	0.128	-30.011	19.137
1	10	trainscg	tansig	purelin	500	MSE	0.0105	0.6	0.5	0.526	0.277	0.277	0.077	-32.669	19.947
1	1	trainscg	tansig	purelin	1000	MSE	0.0119	0.6	0.5	0.421	0.357	0.177	0.127	-30.636	19.447
1	5	trainscg	tansig	purelin	1000	MSE	0.0102	0.6	0.5	0.543	0.272	0.295	0.074	-29.848	20.617
1	10	trainscg	tansig	purelin	1000	MSE	0.0098	0.6	0.5	0.568	0.279	0.322	0.078	-28.624	20.401

**Table 6. 19: Results for the Elman Recurrent Neural Network Model 5 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r train	r test	R sq train	R sq test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0119	0.01	0.5	0.421	0.356	0.177	0.127	-30.699	19.453
1	5	trainscg	tansig	purelin	500	MSE	0.0110	0.01	0.5	0.492	0.289	0.242	0.084	-31.494	20.098
1	10	trainscg	tansig	purelin	500	MSE	0.0106	0.01	0.5	0.515	0.235	0.266	0.055	-33.228	20.593
1	1	trainscg	tansig	purelin	244	MSE	0.0119	0.01	0.5	0.421	0.357	0.177	0.127	-30.700	19.446
1	5	trainscg	tansig	purelin	1000	MSE	0.0112	0.01	0.5	0.476	0.306	0.227	0.094	-26.434	19.715
1	10	trainscg	tansig	purelin	1000	MSE	0.0105	0.01	0.5	0.526	0.235	0.277	0.055	-35.027	20.869

**Table 6. 20: Results for the Radial Basis Function Neural Network Model 5 with COD as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r train	r test	R sq train	R sq test	Avg_test error	APE
1	radbas	purelin	3	3	SSE	5.336	5.5	0.5	0.394	0.354	0.155	0.125	-27.709	19.544

### 6.2.6 COD Neural Network MODEL 6

The inputs for Model 6 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k) and the output is COD(k+1) one time step ahead. These are tabled below:

Table 6. 21: Input variables for COD Model 6

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k)	COD(k+1)

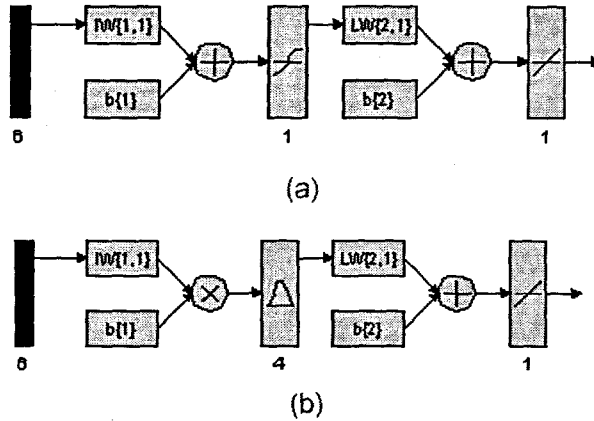
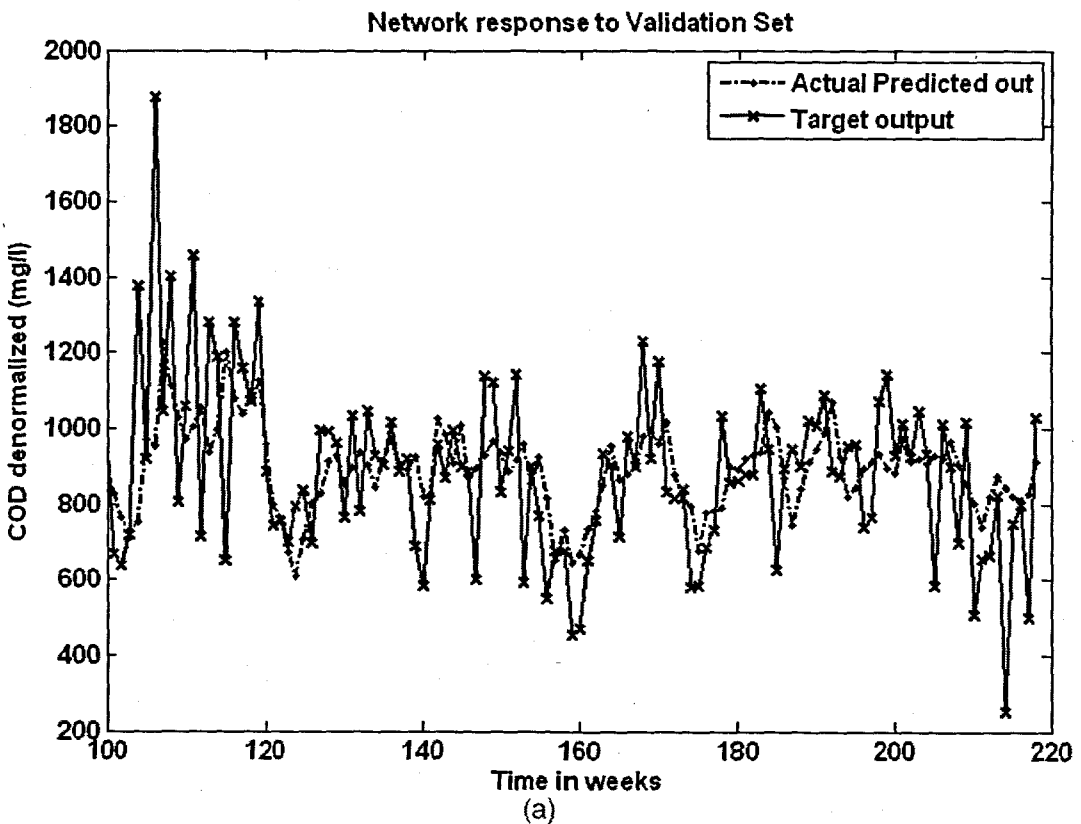


Figure 6. 16: (a) A MLP Feed-forward neural network Model 6 with 6 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 6 with 4 hidden neurons, 4 epochs





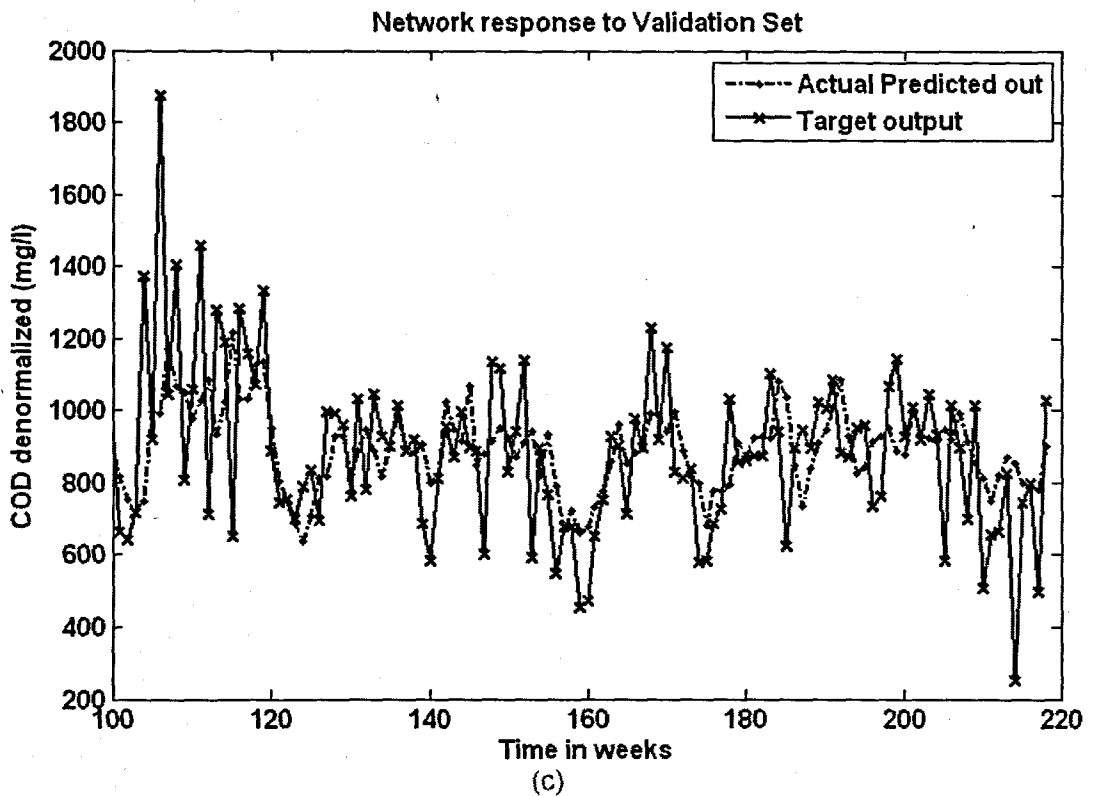
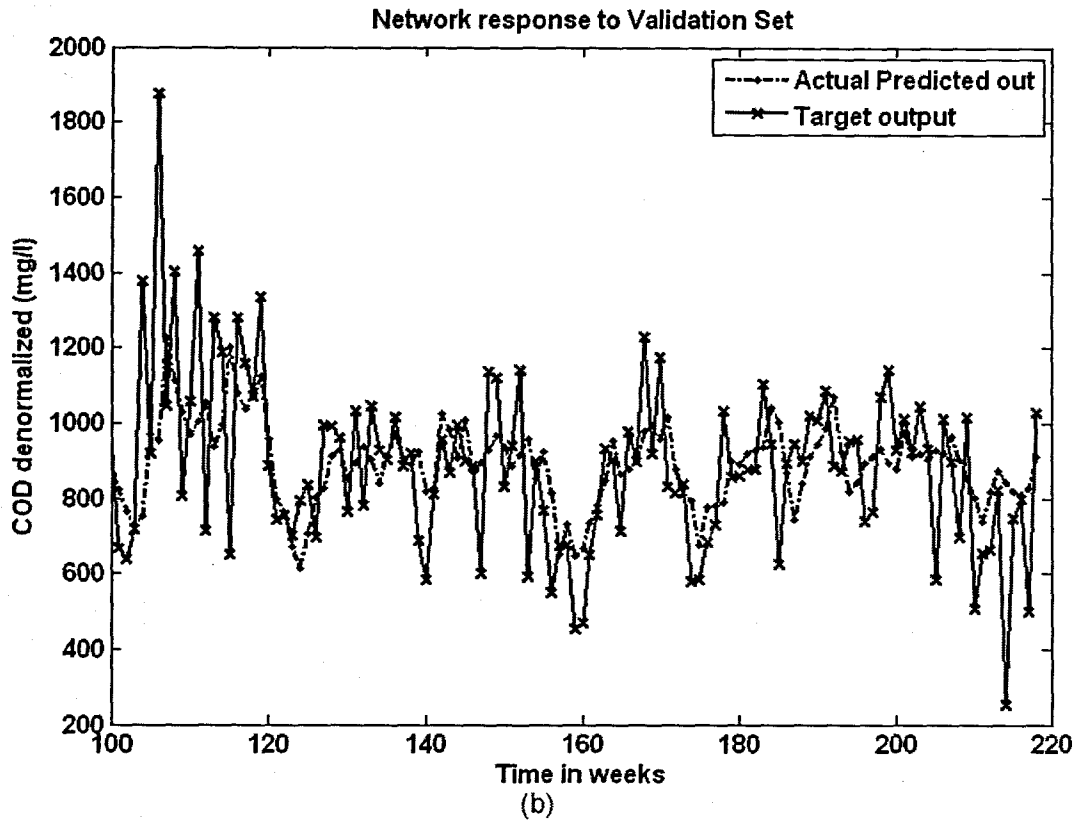


Figure 6. 17: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 6 with 1 hidden neuron and 500 epochs; (b) ERNN Model 6 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 6 with 4 hidden neurons, 4 epochs

**Table 6. 22: Results for the Feed-forward Neural Network Model 6 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0118	0.6	0.5	0.427	0.358	0.182	0.129	-30.027	19.370
1	5	trainscg	tansig	purelin	500	MSE	0.0112	0.6	0.5	0.475	0.330	0.226	0.109	-26.644	19.458
1	10	trainscg	tansig	purelin	500	MSE	0.0105	0.6	0.5	0.528	0.301	0.279	0.091	-33.317	19.663
1	1	trainscg	tansig	purelin	1000	MSE	0.0118	0.6	0.5	0.427	0.358	0.182	0.129	-30.030	19.372
1	5	trainscg	tansig	purelin	1000	MSE	0.0105	0.6	0.5	0.524	0.299	0.274	0.089	-29.089	20.160
1	10	trainscg	tansig	purelin	1000	MSE	0.0096	0.6	0.5	0.580	0.290	0.336	0.084	-19.360	20.100

**Table 6. 23: Results for the Elman Recurrent Neural Network Model 6 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0118	0.01	0.5	0.427	0.358	0.182	0.129	-30.066	19.370
1	5	trainscg	tansig	purelin	500	MSE	0.0107	0.01	0.5	0.509	0.317	0.259	0.100	-23.513	19.977
1	10	trainscg	tansig	purelin	500	MSE	0.0107	0.01	0.5	0.514	0.287	0.264	0.082	-29.797	20.249
1	1	trainscg	tansig	purelin	324	MSE	0.0118	0.01	0.5	0.427	0.359	0.182	0.129	-30.024	19.372
1	5	trainscg	tansig	purelin	1000	MSE	0.0118	0.01	0.5	0.507	0.324	0.257	0.105	-33.078	19.617
1	10	trainscg	tansig	purelin	1000	MSE	0.0100	0.01	0.5	0.555	0.244	0.308	0.059	-24.787	20.223

**Table 6. 24: Results for the Radial Basis Function Neural Network Model 6 with COD as output.**

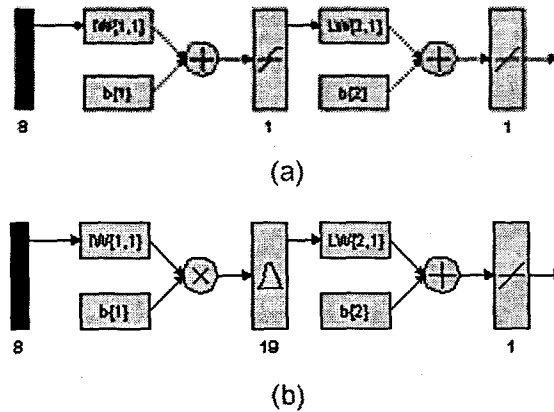
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	4	4	SSE	5.150	5.2	0.7	0.430	0.348	0.185	0.121	-29.662	19.470

### 6.2.7 COD Neural Network MODEL 7

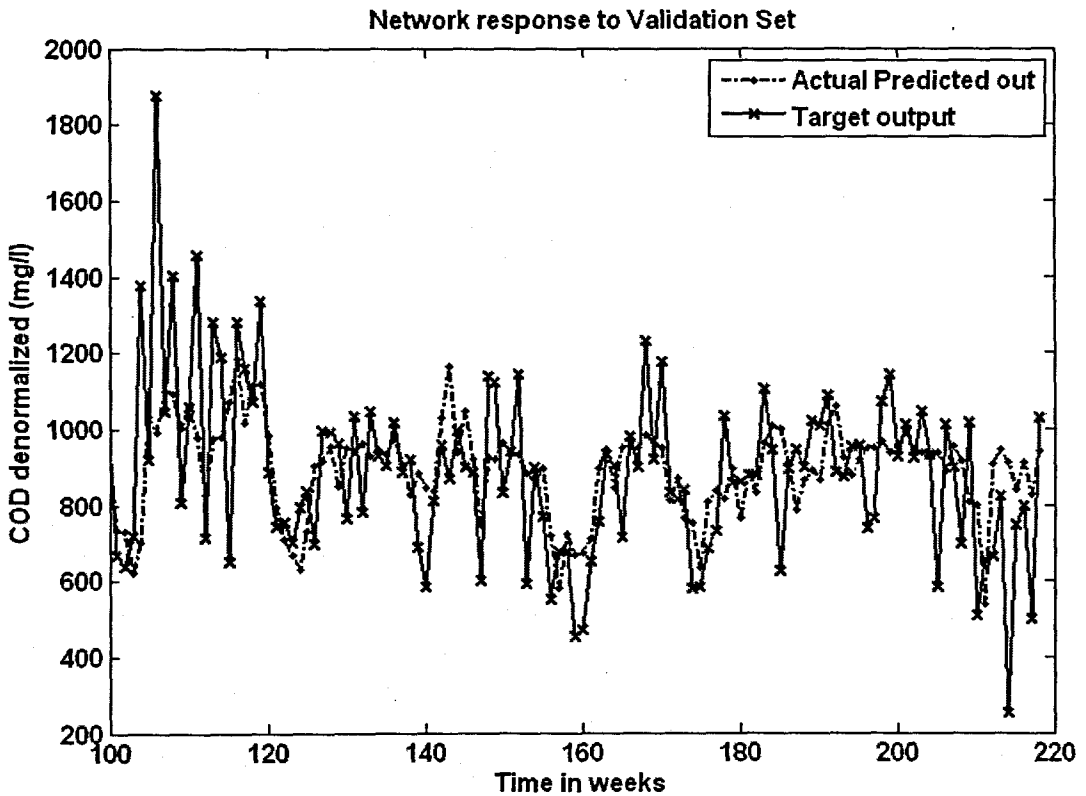
The inputs for Model 7 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), MONTH and the output is COD(k+1) one time step ahead. These are tabled below:

**Table 6. 25: Input variables for COD Model 7**

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), Month	COD(k+1)



**Figure 6. 18: (a) A MLP Feed-forward neural network Model 7 with 8 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 7 with 19 hidden neurons, 19 epochs**



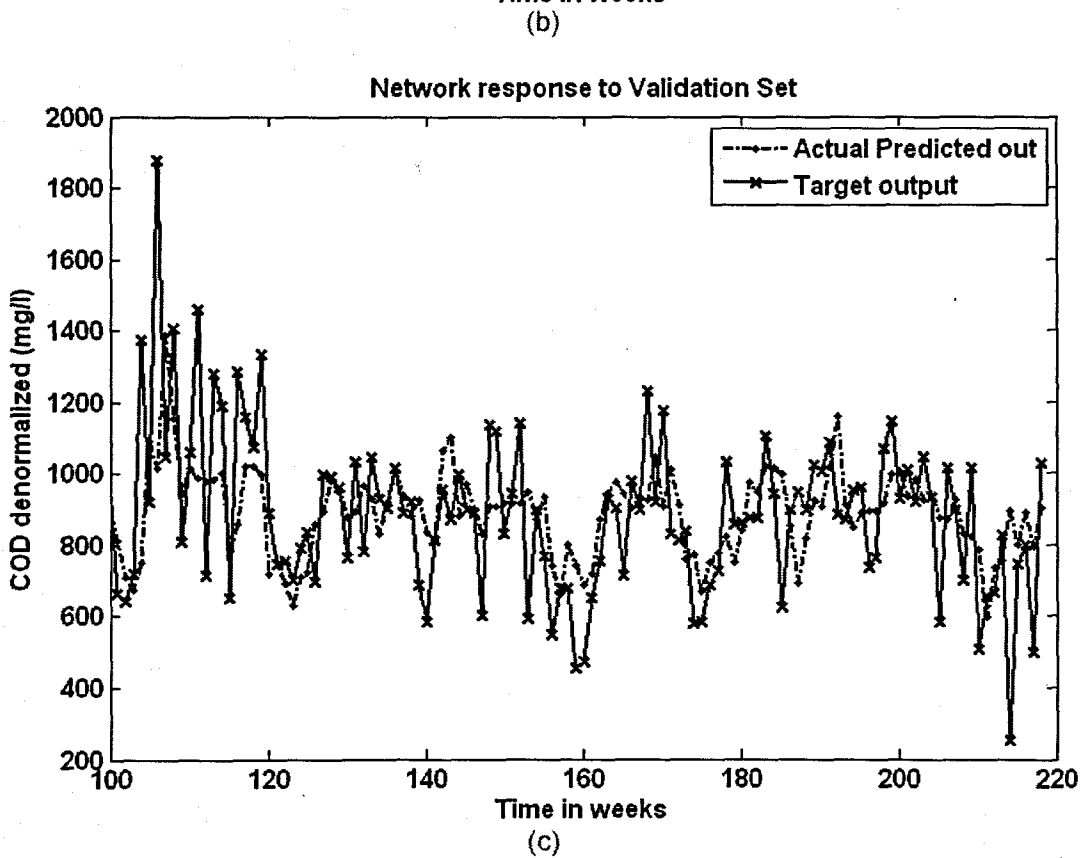
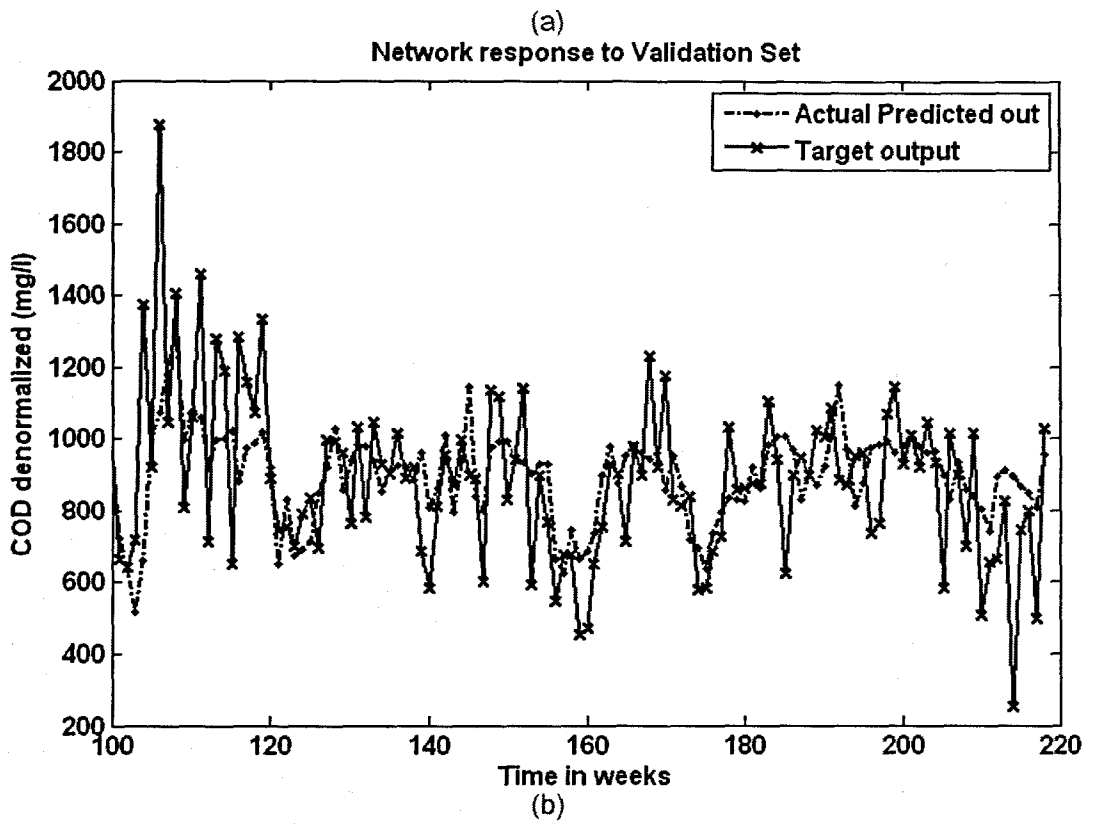


Figure 6. 19: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 7 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 7 with 5 hidden neurons, 1000 epochs; and (c) RBFNN Model 7 with 19 hidden neurons, 19 epochs

**Table 6. 26: Results for the Feed-forward Neural Network Model 7 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.436	0.329	0.190	0.108	-21.388	19.429
1	5	trainscg	tansig	purelin	500	MSE	0.0108	0.6	0.5	0.502	0.319	0.252	0.102	-25.797	19.812
1	10	trainscg	tansig	purelin	500	MSE	0.0093	0.6	0.5	0.597	0.314	0.356	0.099	-21.897	20.941
1	1	trainscg	tansig	purelin	645	MSE	0.0117	0.6	0.5	0.436	0.324	0.190	0.105	-20.799	19.459
1	5	trainscg	tansig	purelin	1000	MSE	0.0100	0.6	0.5	0.556	0.333	0.309	0.111	-19.122	19.525
1	10	trainscg	tansig	purelin	1000	MSE	0.0087	0.6	0.5	0.632	0.286	0.399	0.082	-26.473	21.058

**Table 6. 27: Results for the Elman Recurrent Neural Network Model 7 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.01	0.5	0.436	0.325	0.190	0.106	-20.950	19.449
1	5	trainscg	tansig	purelin	500	MSE	0.0103	0.01	0.5	0.535	0.337	0.286	0.114	-26.419	19.674
1	10	trainscg	tansig	purelin	500	MSE	0.0095	0.01	0.5	0.588	0.273	0.346	0.075	-21.980	21.045
1	1	trainscg	tansig	purelin	952	MSE	0.0117	0.01	0.5	0.436	0.324	0.190	0.105	-20.802	19.459
1	5	trainscg	tansig	purelin	1000	MSE	0.0100	0.01	0.5	0.555	0.342	0.308	0.117	-25.175	19.718
1	10	trainscg	tansig	purelin	1000	MSE	0.0084	0.01	0.5	0.646	0.269	0.418	0.072	-10.847	21.804

**Table 6. 28: Results for the Radial Basis Function Neural Network Model 7 with COD as output.**

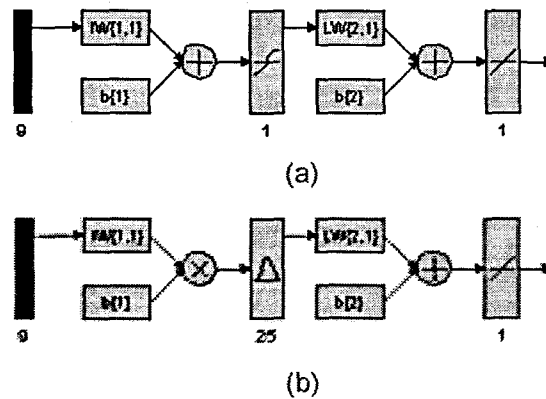
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	19	19	SSE	4.587	4.6	0.8	0.523	0.351	0.274	0.123	-31.742	20.257

### 6.2.8 COD Neural Network MODEL 8

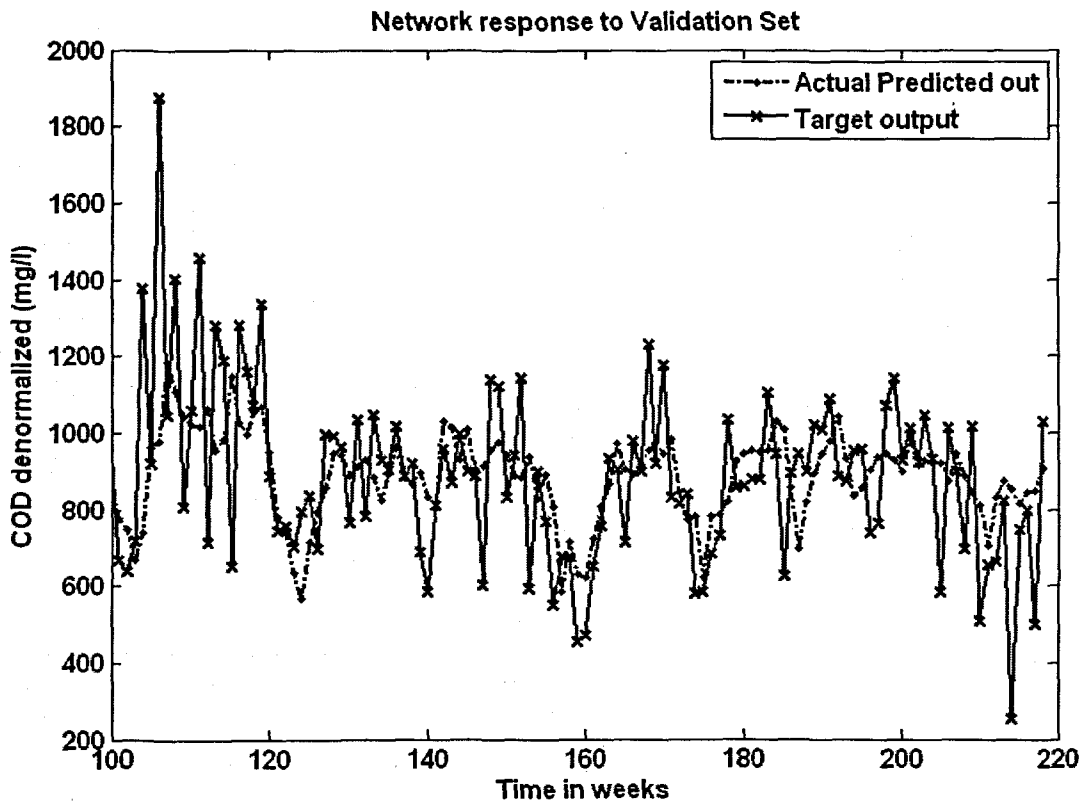
The inputs for Model 8 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), MONTH, Minimum Temperature, and the output is COD(k+1) one time step ahead. These are tabled below:

**Table 6. 29: Input variables for COD Model 8**

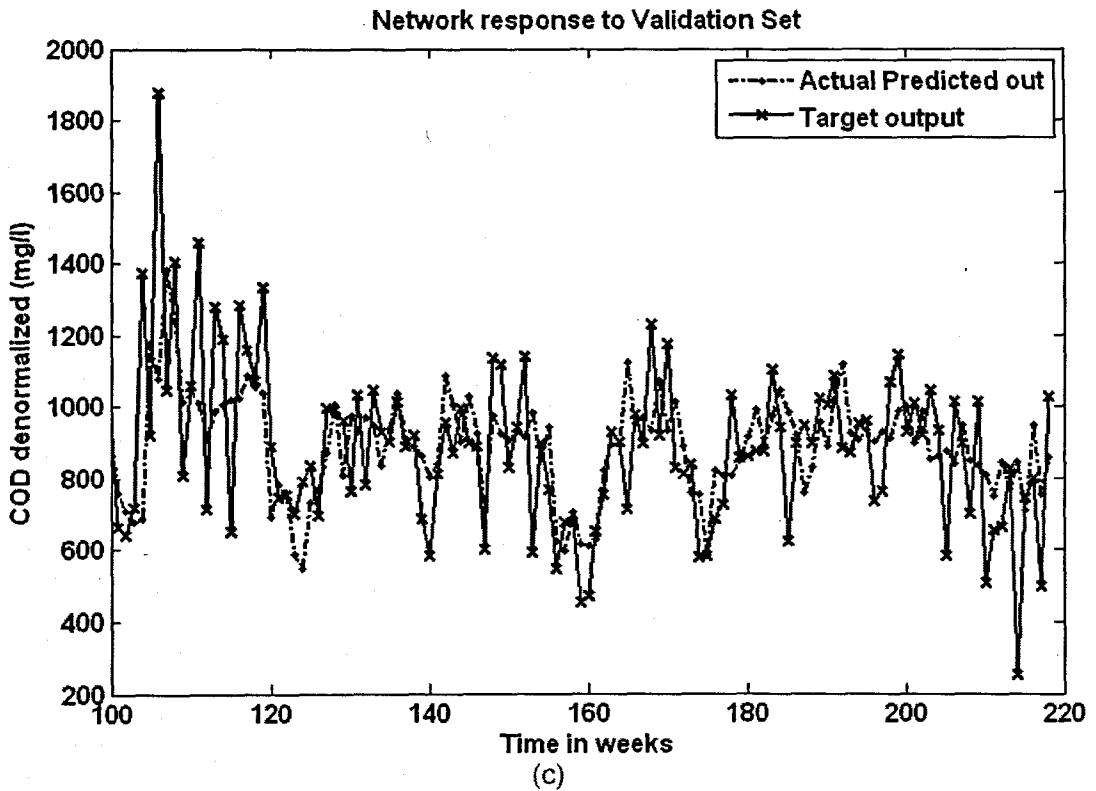
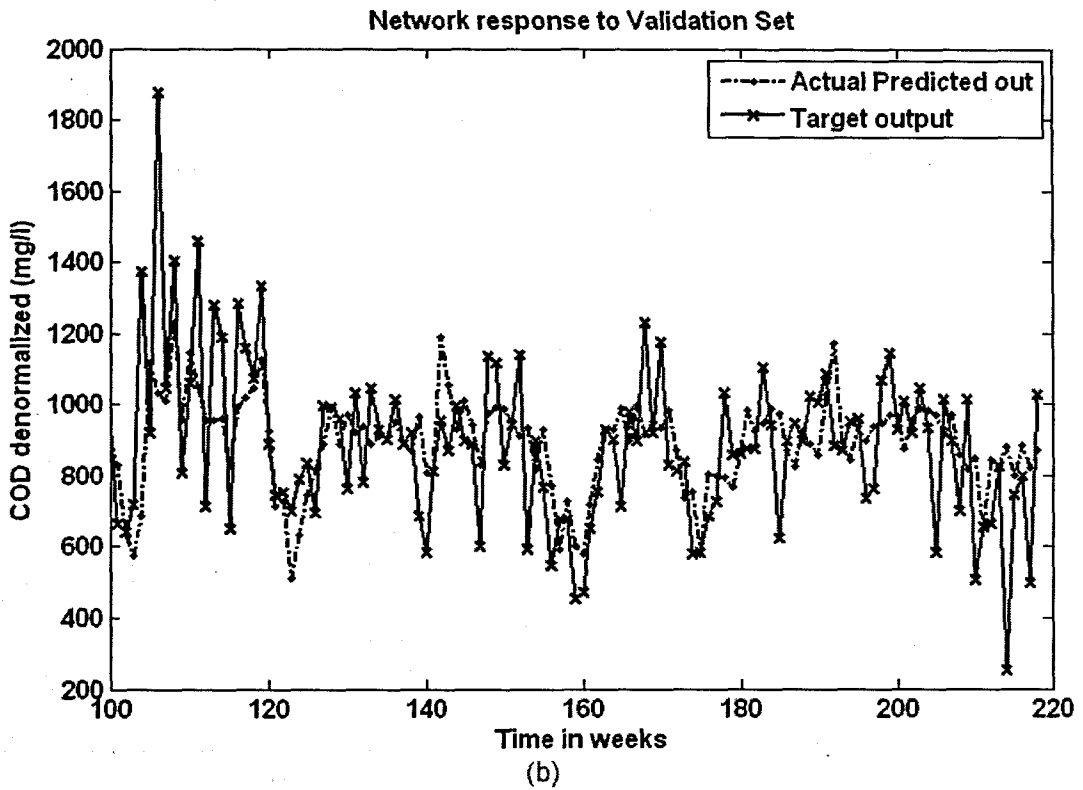
Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), Month, Minimum Temperature	COD(k+1)



**Figure 6. 20:** (a) A MLP Feed-forward neural network Model 8 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 8 with 25 hidden neurons, 25 epochs



(a)



**Figure 6. 21:** The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 8 with 1 hidden neuron and 500 epochs; (b) ERNN Model 8 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 8 with 25 hidden neurons, 25 epochs

**Table 6. 30: Results for the Feed-forward Neural Network Model 8 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.435	0.385	0.189	0.148	-30.519	19.323
1	1	trainscg	tansig	purelin	500	MSE	0.0106	0.6	0.5	0.518	0.250	0.268	0.062	-17.005	20.951
1	10	trainscg	tansig	purelin	500	MSE	0.0099	0.6	0.5	0.561	0.289	0.315	0.084	-33.171	21.328
1	1	trainscg	tansig	purelin	1000	MSE	0.0117	0.6	0.5	0.438	0.336	0.192	0.113	-22.411	19.331
1	5	trainscg	tansig	purelin	1000	MSE	0.0087	0.6	0.5	0.632	0.211	0.399	0.045	-13.123	21.731
1	10	trainscg	tansig	purelin	1000	MSE	0.0076	0.6	0.5	0.688	0.231	0.474	0.054	-35.579	23.268

**Table 6. 31: Results for the Elman Recurrent Neural Network Model 8 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	459	MSE	0.0117	0.01	0.5	0.438	0.337	0.192	0.114	-22.521	19.324
1	5	trainscg	tansig	purelin	500	MSE	0.0102	0.01	0.5	0.545	0.389	0.297	0.151	-21.235	19.152
1	10	trainscg	tansig	purelin	500	MSE	0.0095	0.01	0.5	0.584	0.225	0.341	0.051	-28.433	21.592
1	1	trainscg	tansig	purelin	618	MSE	0.0117	0.01	0.5	0.438	0.337	0.192	0.114	-22.502	19.325
1	5	trainscg	tansig	purelin	1000	MSE	0.0097	0.01	0.5	0.576	0.194	0.332	0.038	-23.752	21.546
1	10	trainscg	tansig	purelin	1000	MSE	0.0076	0.01	0.5	0.689	0.204	0.474	0.042	-6.577	22.959

**Table 6. 32: Results for the Radial Basis Function Neural Network Model 8 with COD as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	25	25	SSE	4.346	4.4	0.8	0.558	0.342	0.312	0.117	-31.523	20.682

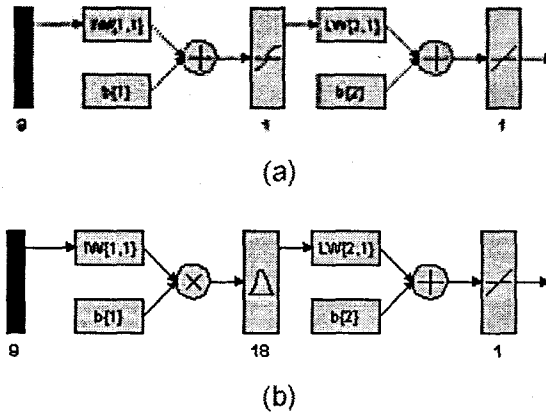


### 6.2.9 COD Neural Network MODEL 9

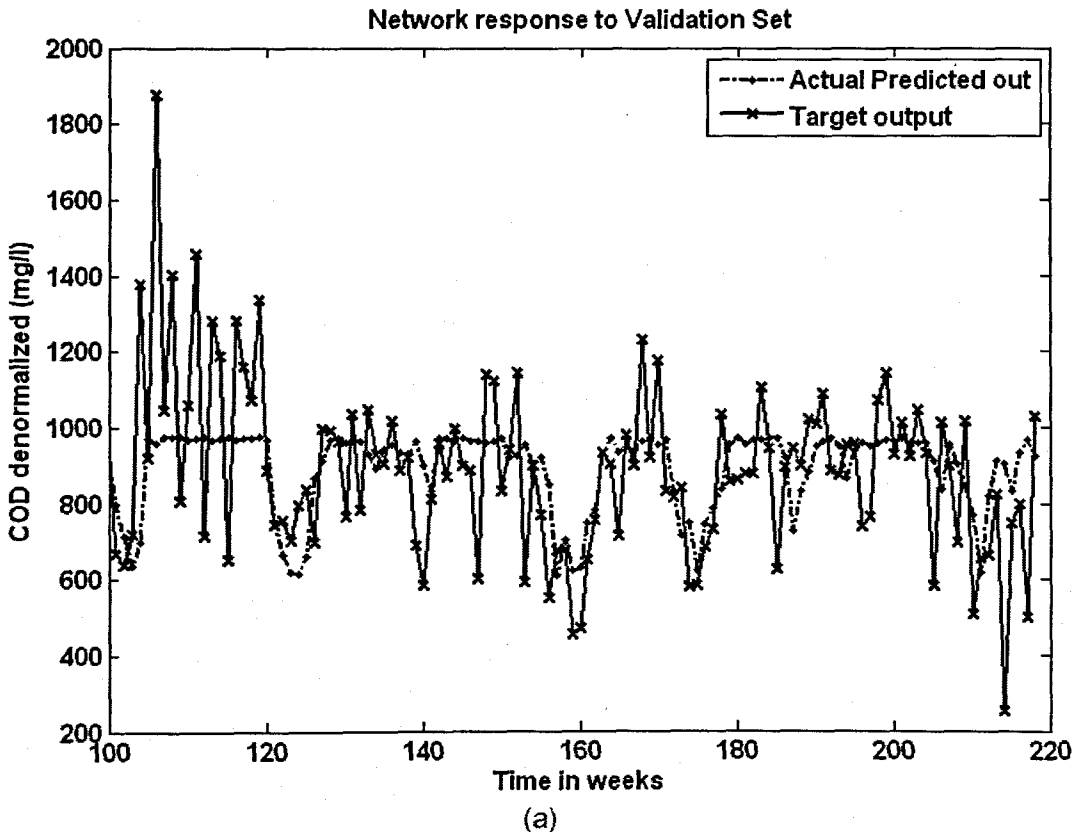
The inputs for Model 9 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), MONTH, Rain, and the output is COD(k+1) one time step ahead. These are tabled below:

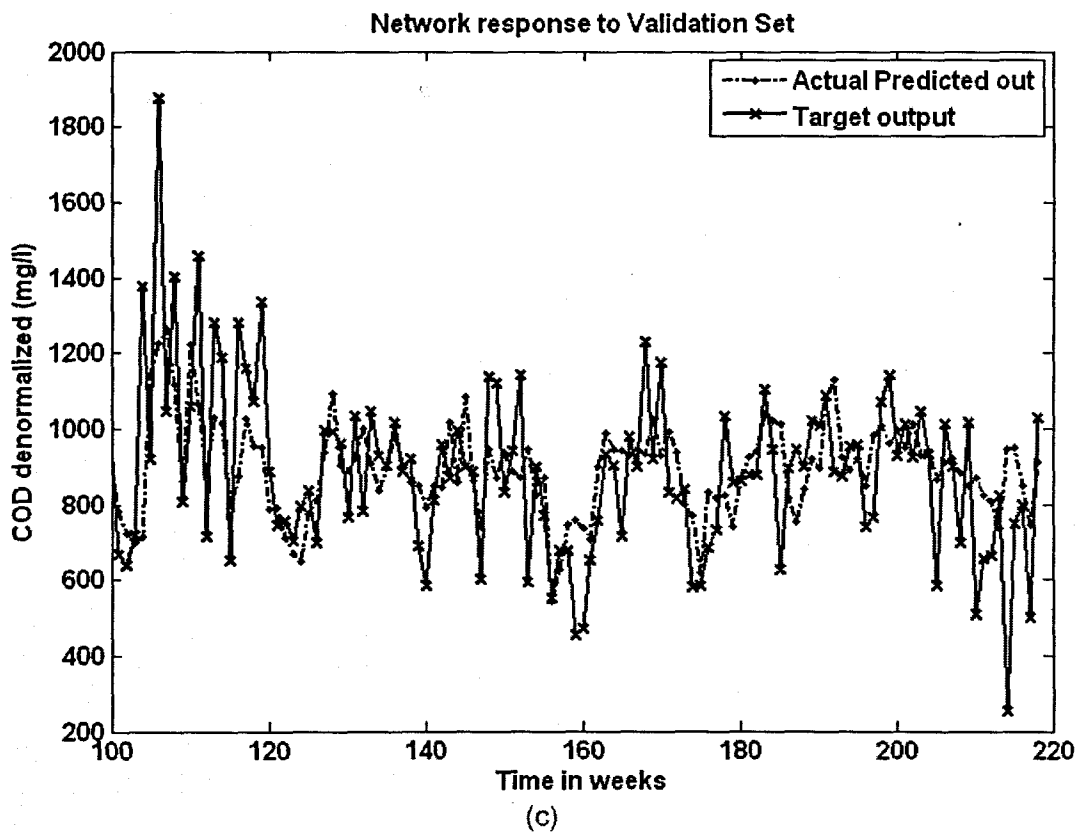
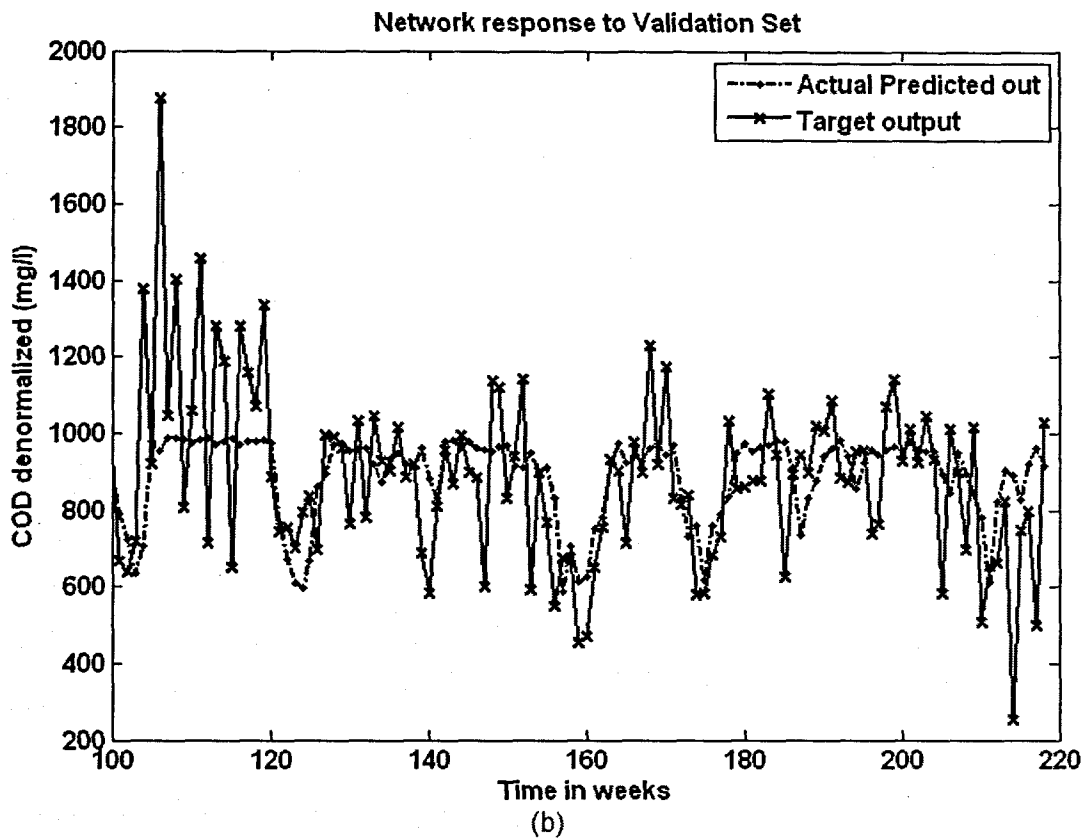
**Table 6. 33: Input variables for COD Model 9**

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), Month, Rain	COD(k+1)



**Figure 6. 22: (a) A MLP Feed-forward neural network Model 9 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 9 with 18 hidden neurons, 18 epochs**





**Figure 6. 23:** The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 9 with 1 hidden neuron and 520 epochs; (b) ERNN Model 9 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 9 with 18 hidden neurons, 18 epochs

**Table 6. 34: Results for the Feed-forward Neural Network Model 9 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.437	0.339	0.191	0.115	-21.874	19.372
1	5	trainscg	tansig	purelin	500	MSE	0.0100	0.6	0.5	0.555	0.326	0.308	0.106	-22.368	20.419
1	10	trainscg	tansig	purelin	500	MSE	0.0097	0.6	0.5	0.575	0.278	0.331	0.077	-26.739	20.298
1	1	trainscg	tansig	purelin	520	MSE	0.0117	0.6	0.5	0.437	0.340	0.191	0.115	-21.905	19.372
1	5	trainscg	tansig	purelin	1000	MSE	0.0090	0.6	0.5	0.617	0.241	0.380	0.058	-20.059	21.486
1	10	trainscg	tansig	purelin	1000	MSE	0.0081	0.6	0.5	0.663	0.203	0.439	0.041	-21.707	22.085

**Table 6. 35: Results for the Elman Recurrent Neural Network Model 9 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.01	0.5	0.437	0.349	0.191	0.122	-23.051	19.373
1	5	trainscg	tansig	purelin	500	MSE	0.0102	0.01	0.5	0.545	0.258	0.297	0.066	-23.875	20.496
1	10	trainscg	tansig	purelin	500	MSE	0.0093	0.01	0.5	0.601	0.202	0.361	0.041	-21.022	21.829
1	1	trainscg	tansig	purelin	813	MSE	0.0117	0.01	0.5	0.437	0.339	0.191	0.115	-21.868	19.371
1	5	trainscg	tansig	purelin	1000	MSE	0.0094	0.01	0.5	0.595	0.287	0.354	0.082	-24.913	20.212
1	10	trainscg	tansig	purelin	1000	MSE	0.0083	0.01	0.5	0.655	0.209	0.429	0.044	-21.373	22.603

**Table 6. 36: Results for the Radial Basis Function Neural Network Model 9 with COD as output.**

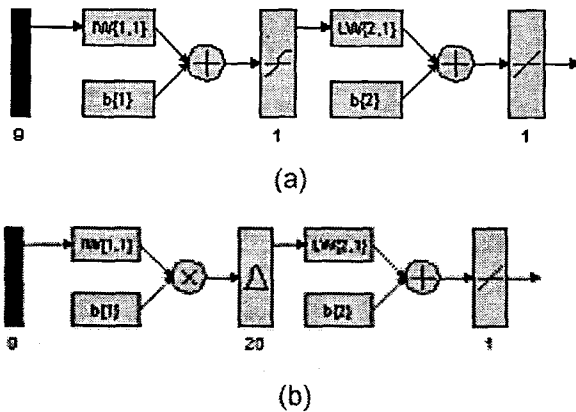
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	18	18	SSE	4.377	4.4	0.6	0.554	0.380	0.307	0.144	-31.069	20.150

### 6.2.10 COD Neural Network MODEL 10

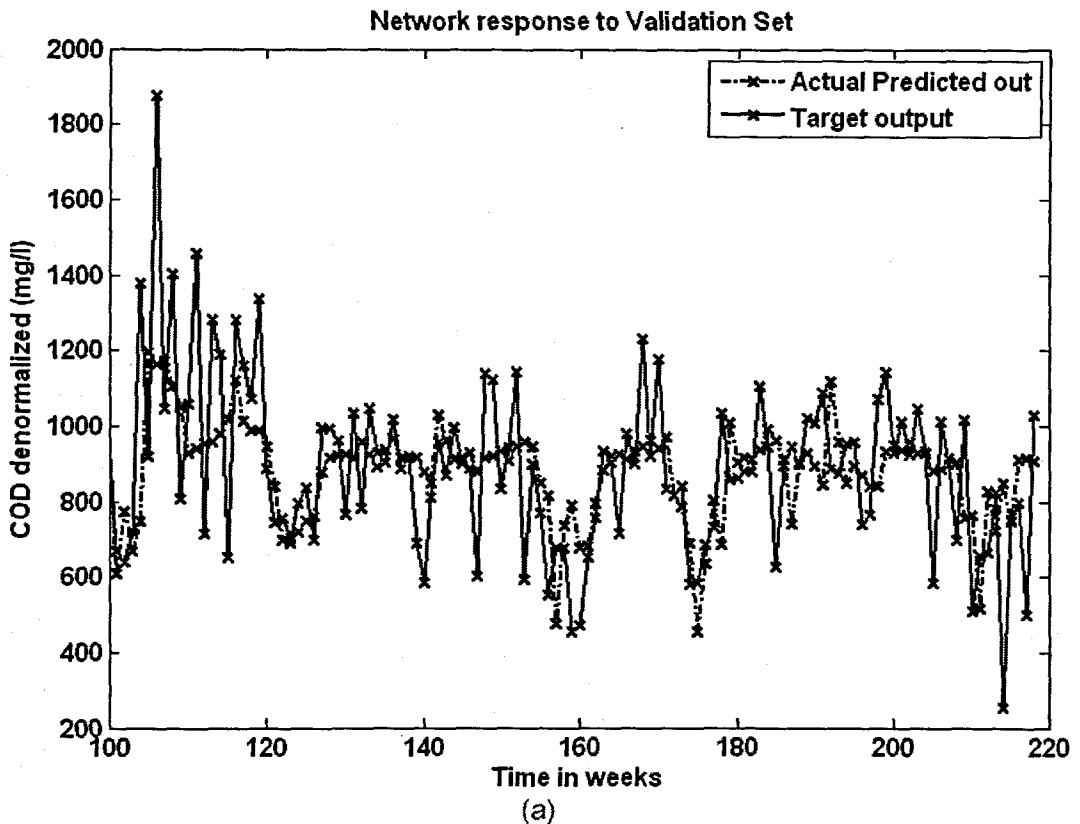
The inputs for Model 10 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), MONTH, Wind, and the output is COD(k+1) one time step ahead. These are tabled below:

**Table 6. 37: Input variables for COD Model 10**

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), Month, Wind	COD(k+1)



**Figure 6. 24: (a) A MLP Feed-forward neural network Model 10 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 10 with 20 hidden neurons, 20 epochs**



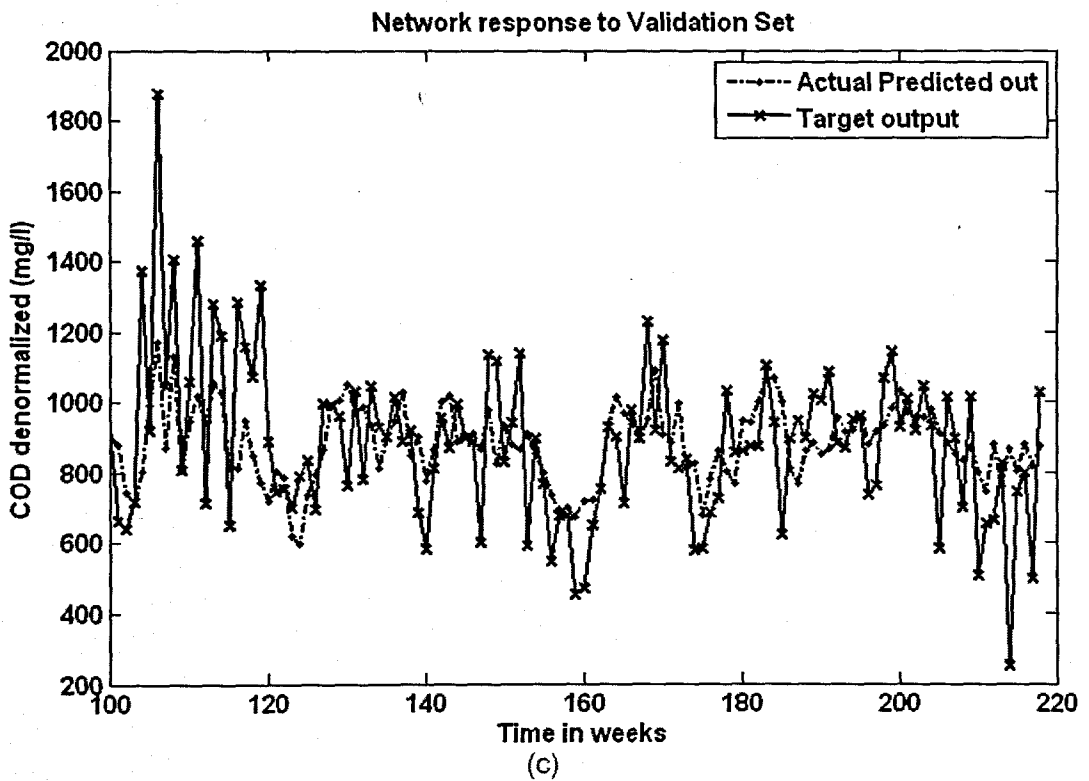
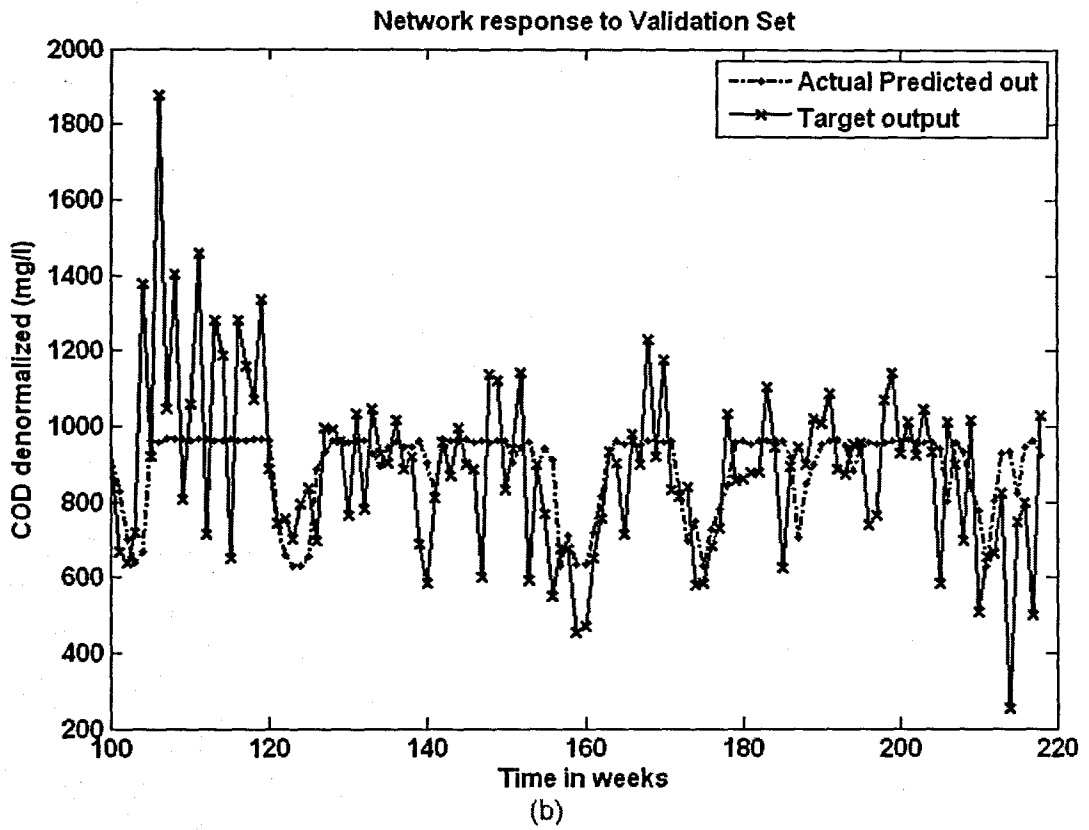


Figure 6. 25: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 10 with 5 hidden neuron and 1000 epochs; (b) ERNN Model 10 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 10 with 20 hidden neurons, 20 epochs

**Table 6. 38: Results for the Feed-forward Neural Network Model 10 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.437	0.320	0.191	0.102	-21.254	19.498
1	5	trainscg	tansig	purelin	500	MSE	0.0106	0.6	0.5	0.516	0.356	0.266	0.127	-29.691	19.575
1	10	trainscg	tansig	purelin	500	MSE	0.0093	0.6	0.5	0.596	0.185	0.355	0.034	-23.349	22.211
1	1	trainscg	tansig	purelin	943	MSE	0.0117	0.6	0.5	0.437	0.319	0.191	0.102	-21.234	19.500
1	5	trainscg	tansig	purelin	1000	MSE	0.0101	0.6	0.5	0.548	0.331	0.300	0.110	-31.542	19.961
1	10	trainscg	tansig	purelin	1000	MSE	0.0080	0.6	0.5	0.667	0.216	0.445	0.047	-36.008	22.105

**Table 6. 39: Results for the Elman Recurrent Neural Network Model 10 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	475	MSE	0.0117	0.01	0.5	0.437	0.319	0.191	0.102	-21.218	19.502
1	5	trainscg	tansig	purelin	500	MSE	0.0095	0.01	0.5	0.584	0.308	0.342	0.095	-32.925	21.024
1	10	trainscg	tansig	purelin	500	MSE	0.0090	0.01	0.5	0.618	0.232	0.382	0.054	-32.669	22.374
1	1	trainscg	tansig	purelin	756	MSE	0.0117	0.01	0.5	0.437	0.318	0.191	0.101	-21.139	19.505
1	5	trainscg	tansig	purelin	1000	MSE	0.0096	0.01	0.5	0.580	0.254	0.337	0.065	-27.514	20.393
1	10	trainscg	tansig	purelin	1000	MSE	0.0090	0.01	0.5	0.616	0.217	0.380	0.047	-21.110	22.249

**Table 6. 40: Results for the Radial Basis Function Neural Network Model 10 with COD as output.**

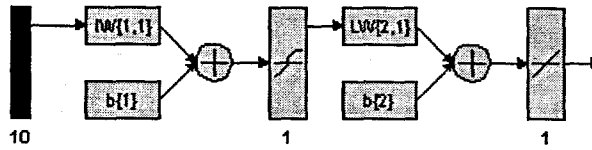
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	20	20	SSE	4.384	4.4	0.4	0.553	0.354	0.306	0.125	-37.670	20.129

### 6.2.11 COD Neural Network MODEL 11

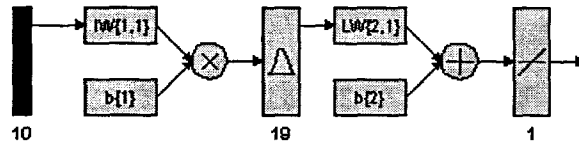
The inputs for Model 11 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), MONTH, Minimum Temperature, Rain, and the output is COD(k+1) one time step ahead. These are tabled below:

**Table 6. 41: Input variables for COD Model 11**

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), Month, Minimum Temperature, Rain	COD(k+1)

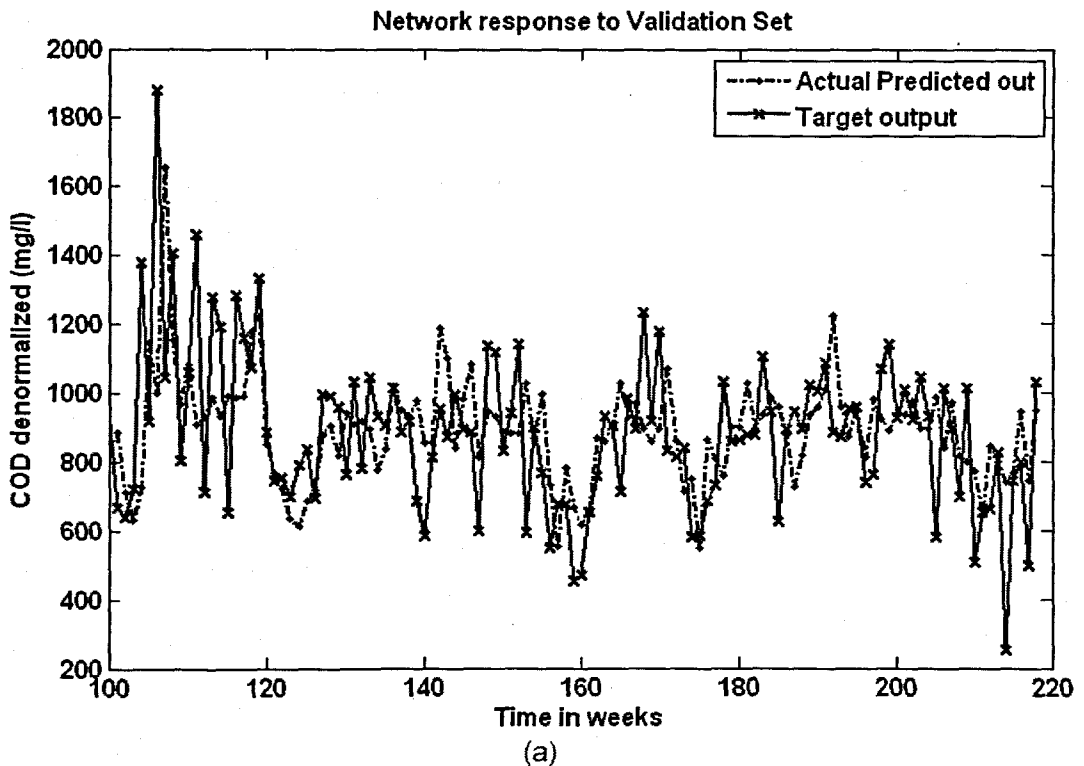


(a)



(b)

**Figure 6. 26: (a) A MLP Feed-forward neural network Model 11 with 10 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 11 with 19 hidden neurons, 19 epochs**



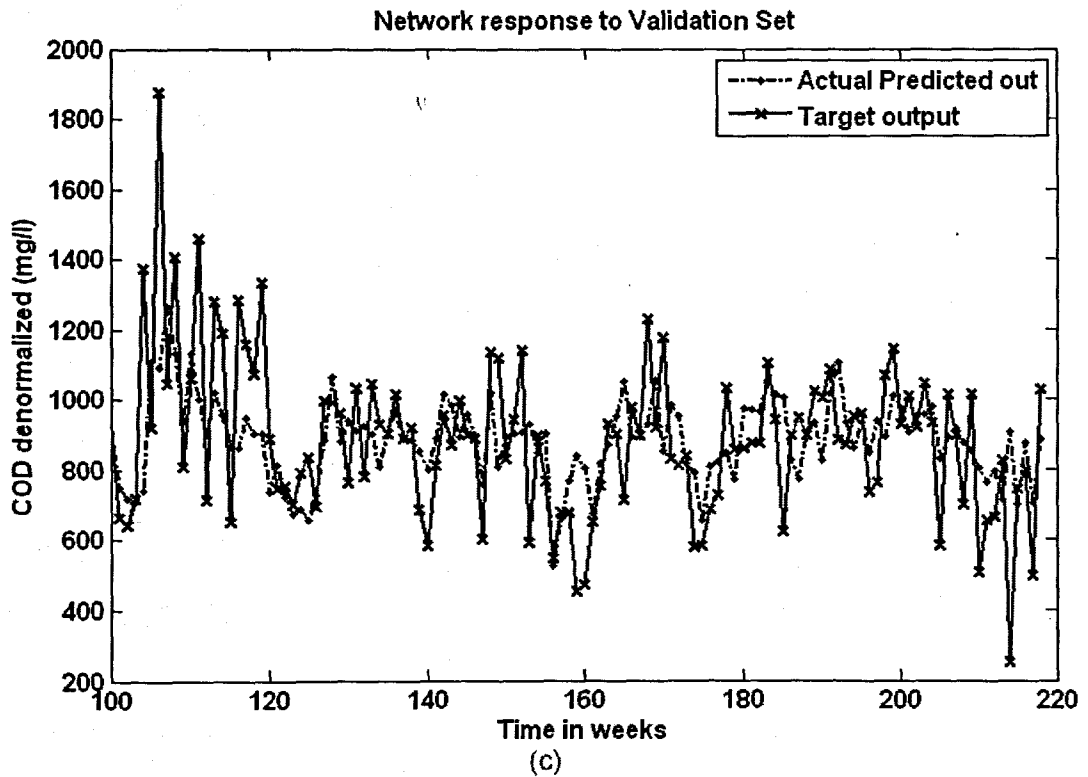
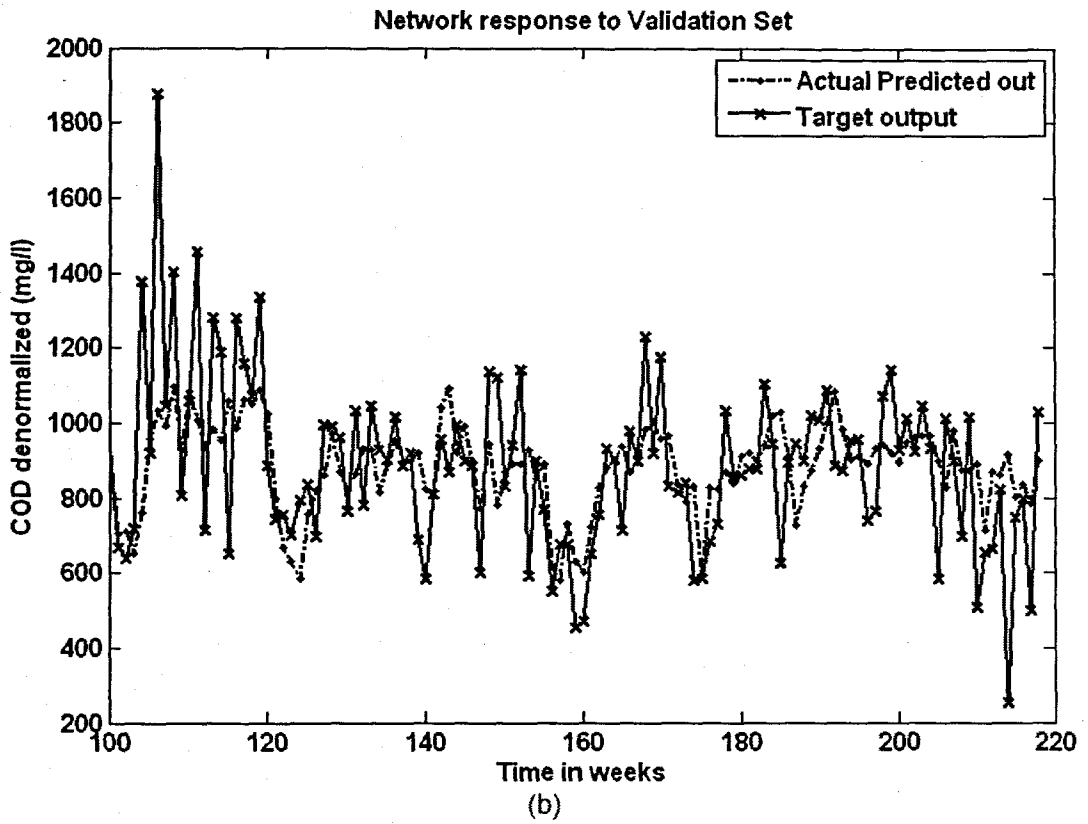


Figure 6. 27: The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 11 with 10 hidden neurons and 500 epochs; (b) ERNN Model 11 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 11 with 19 hidden neurons, 19 epochs



**Table 6. 42: Results for the Feed-forward Neural Network Model 11 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.439	0.352	0.193	0.124	-23.804	19.282
1	5	trainscg	tansig	purelin	500	MSE	0.0107	0.6	0.5	0.513	0.344	0.263	0.118	-27.965	20.242
1	10	trainscg	tansig	purelin	500	MSE	0.0098	0.6	0.5	0.566	0.354	0.320	0.125	-18.301	20.796
1	1	trainscg	tansig	purelin	537	MSE	0.0117	0.6	0.5	0.439	0.352	0.193	0.124	-23.742	19.281
1	5	trainscg	tansig	purelin	1000	MSE	0.0095	0.6	0.5	0.587	0.222	0.344	0.049	-26.683	20.514
1	10	trainscg	tansig	purelin	1000	MSE	0.0076	0.6	0.5	0.691	0.100	0.478	0.010	-34.433	24.988

**Table 6. 43: Results for the Elman Recurrent Neural Network Model 11 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.01	0.5	0.438	0.379	0.191	0.144	-27.040	19.228
1	5	trainscg	tansig	purelin	500	MSE	0.0111	0.01	0.5	0.486	0.379	0.236	0.144	-32.939	19.177
1	10	trainscg	tansig	purelin	500	MSE	0.0106	0.01	0.5	0.518	0.307	0.269	0.094	-32.689	20.004
1	1	trainscg	tansig	purelin	491	MSE	0.0117	0.01	0.5	0.439	0.352	0.193	0.124	-23.735	19.282
1	5	trainscg	tansig	purelin	1000	MSE	0.0098	0.01	0.5	0.569	0.256	0.324	0.066	-30.042	20.433
1	10	trainscg	tansig	purelin	1000	MSE	0.0076	0.01	0.5	0.691	0.133	0.477	0.018	-26.579	23.676

**Table 6. 44: Results for the Radial Basis Function Neural Network Model 11 with COD as output.**

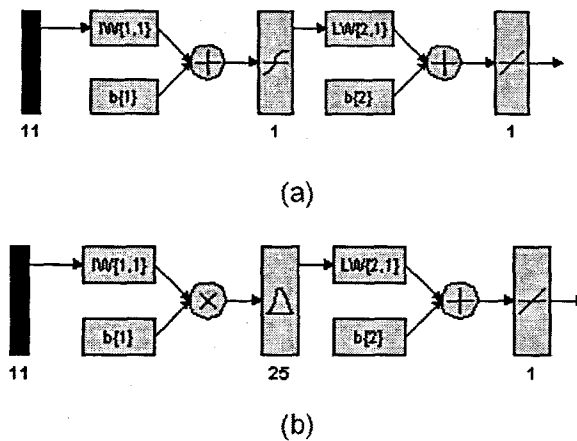
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	19	19	SSE	4.306	4.4	0.6	0.564	0.353	0.318	0.124	-36.968	20.417

### 6.2.12 COD Neural Network MODEL 12

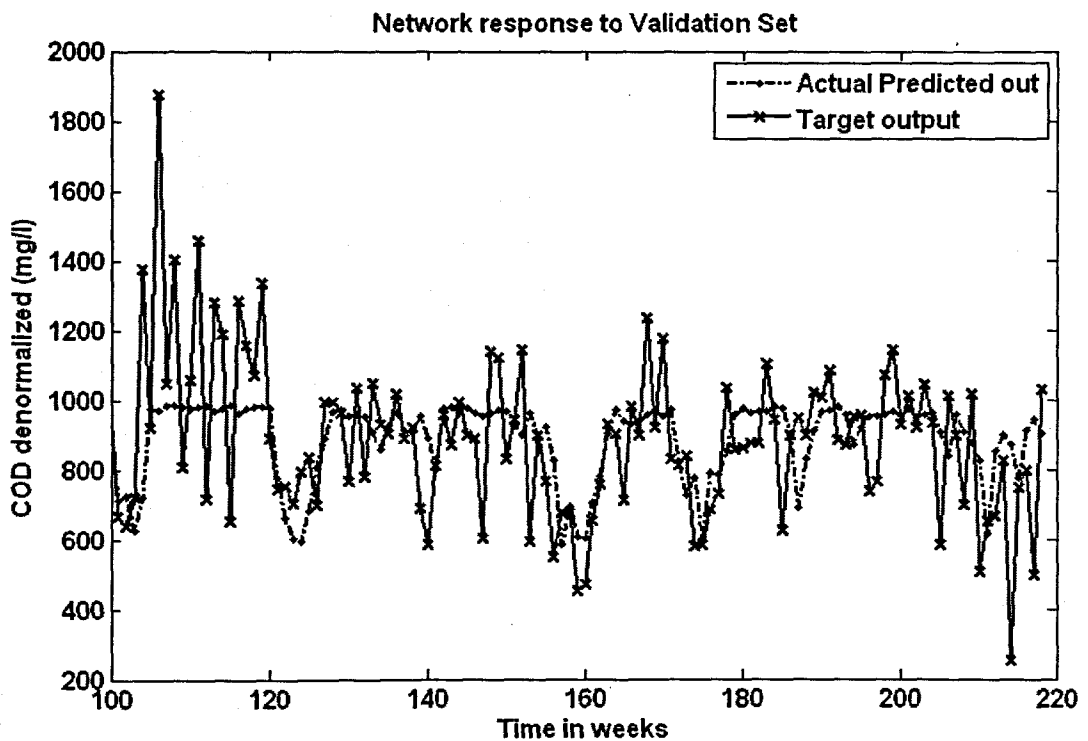
The inputs for Model 12 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), MONTH, Minimum Temperature, Rain, Wind, and the output is COD(k+1) one time step ahead. These are tabled below:

**Table 6. 45: Input variables for COD Model 12**

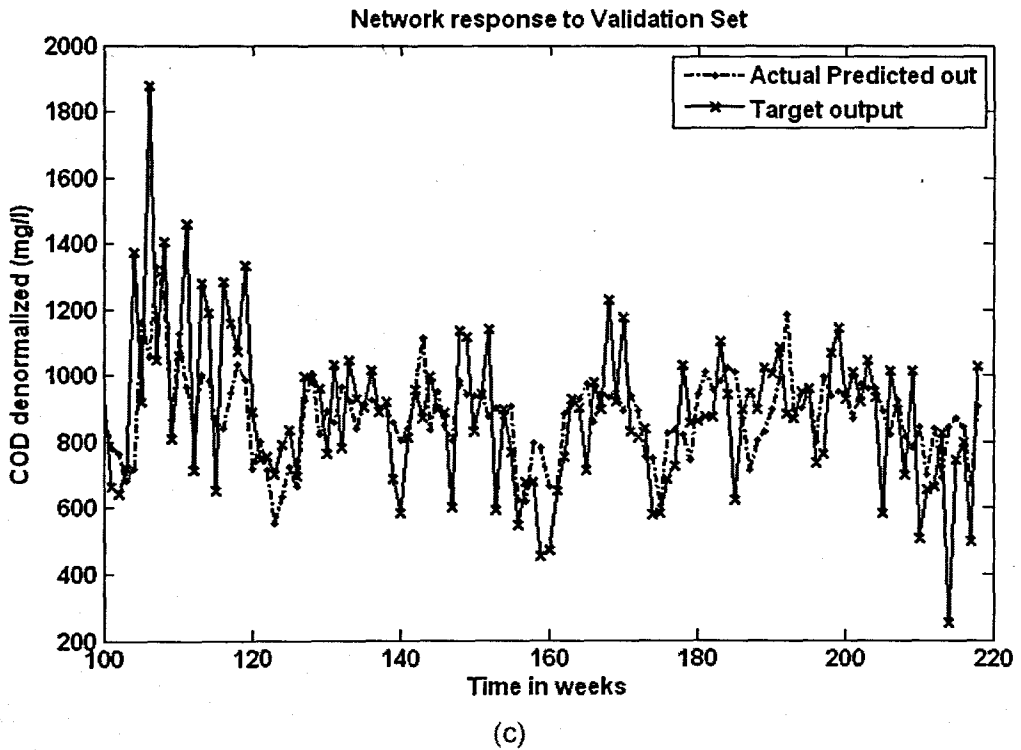
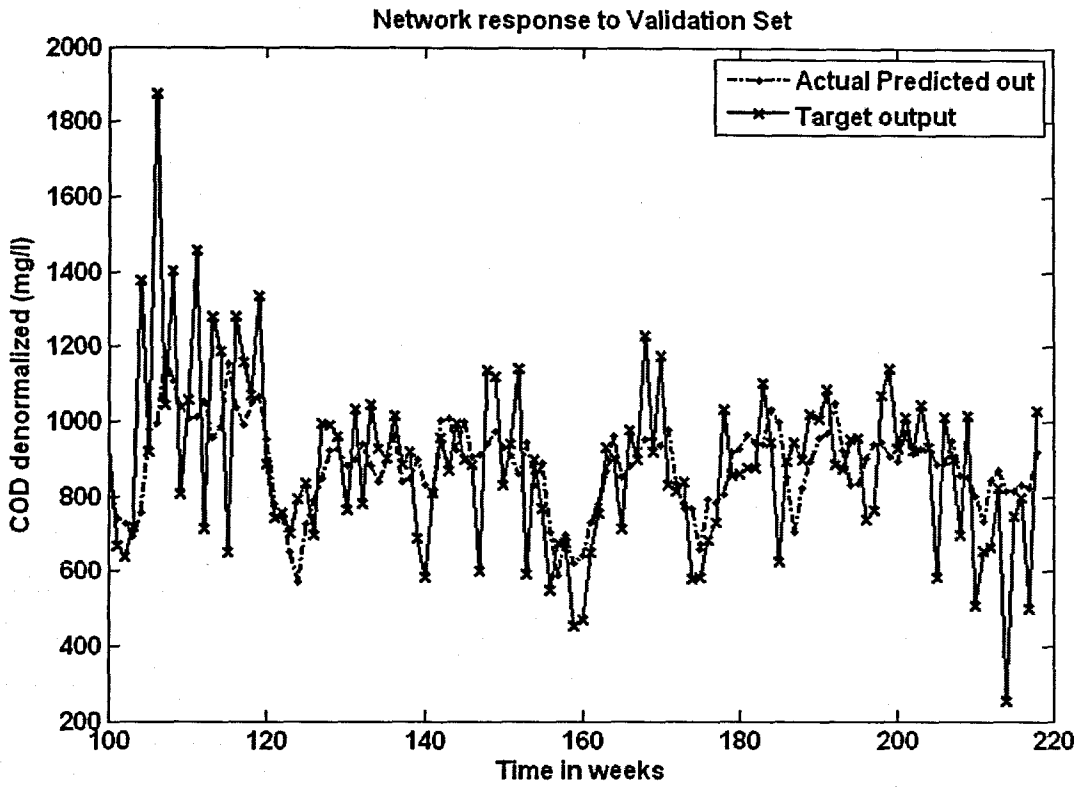
Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k), Month, Minimum Temperature, Rain, Wind	COD(k+1)



**Figure 6. 28: (a) A MLP Feed-forward neural network Model 12 with 11 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 12 with 25 hidden neurons, 25 epochs**



(a)



**Figure 6. 29:** The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 12 with 1 hidden neuron and 500 epochs; (b) ERNN Model 12 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 12 with 25 hidden neurons, 25 epochs

**Table 6. 46: Results for the Feed-forward Neural Network Model 12 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r train	r test	R sq train	R sq test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.439	0.360	0.193	0.130	-24.696	19.281
1	5	trainscg	tansig	purelin	500	MSE	0.0105	0.6	0.5	0.524	0.314	0.275	0.098	-36.501	20.793
1	10	trainscg	tansig	purelin	500	MSE	0.0098	0.6	0.5	0.566	0.295	0.320	0.087	-32.881	20.809
1	1	trainscg	tansig	purelin	1000	MSE	0.0117	0.6	0.5	0.439	0.349	0.193	0.122	-23.780	19.307
1	5	trainscg	tansig	purelin	1000	MSE	0.0098	0.6	0.5	0.568	0.288	0.322	0.083	-36.529	20.437
1	10	trainscg	tansig	purelin	1000	MSE	0.0085	0.6	0.5	0.643	0.172	0.414	0.030	-34.419	22.906

**Table 6. 47: Results for the Elman Recurrent Neural Network Model 12 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r train	r test	R sq train	R sq test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.01	0.5	0.441	0.408	0.194	0.166	-30.066	19.193
1	5	trainscg	tansig	purelin	500	MSE	0.0098	0.01	0.5	0.566	0.376	0.320	0.142	-33.320	19.992
1	10	trainscg	tansig	purelin	500	MSE	0.0096	0.01	0.5	0.581	0.333	0.337	0.111	-37.867	20.248
1	1	trainscg	tansig	purelin	823	MSE	0.0117	0.01	0.5	0.439	0.350	0.193	0.122	-23.789	19.306
1	5	trainscg	tansig	purelin	1000	MSE	0.0096	0.01	0.5	0.578	0.289	0.334	0.083	-30.547	20.349
1	10	trainscg	tansig	purelin	1000	MSE	0.0075	0.01	0.5	0.696	0.200	0.484	0.040	-42.493	23.375

**Table 6. 48: Results for the Radial Basis Function Neural Network Model 12 with COD as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r train	r test	R sq train	R sq test	Avg_test error	APE
1	radbas	purelin	25	25	SSE	4.392	4.4	0.9	0.552	0.360	0.305	0.129	-33.581	20.490

### 6.2.13 COD Neural Network MODEL 13

The inputs for Model 13 are COD(k), COD(k-1), COD(k-2), COD(k-3), MONTH, Minimum Temperature, Rain, Wind, and the output is COD(k+1) one time step ahead. These are tabled below:

Table 6. 49: Input variables for COD Model 13

Inputs	Output
COD(k), COD(k-1), COD(k-2), COD(k-3), Month, Minimum Temperature, Rain, Wind	COD(k+1)

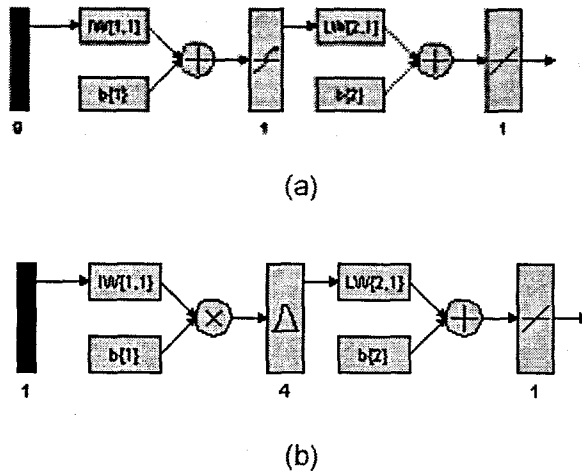
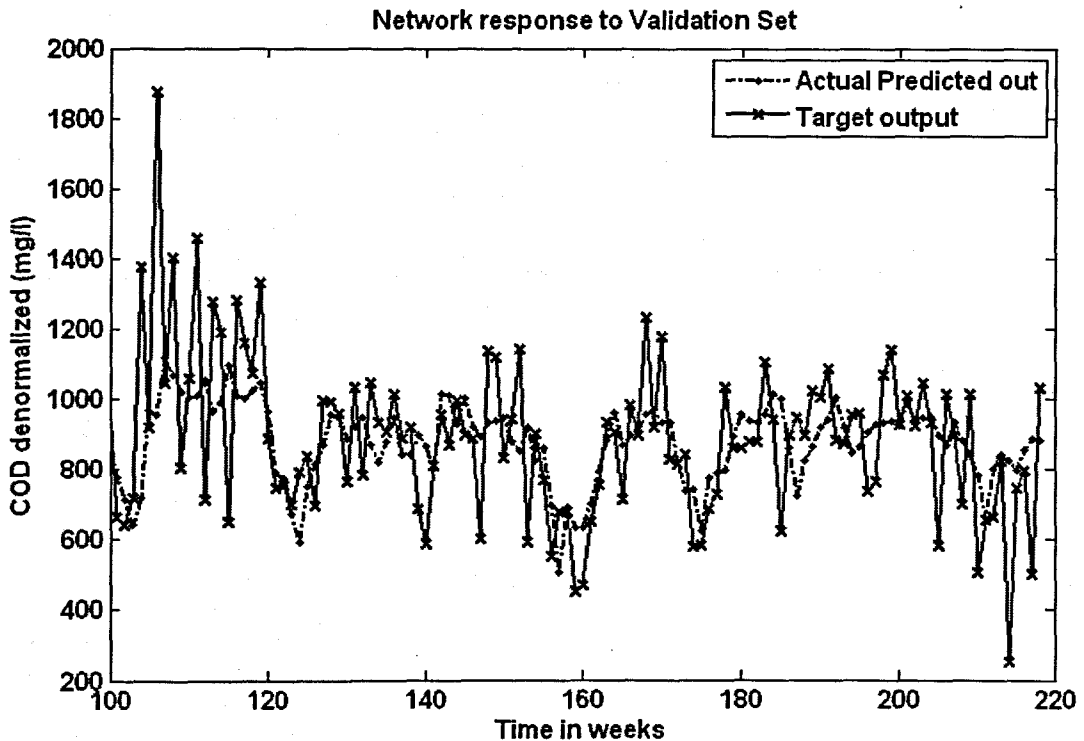
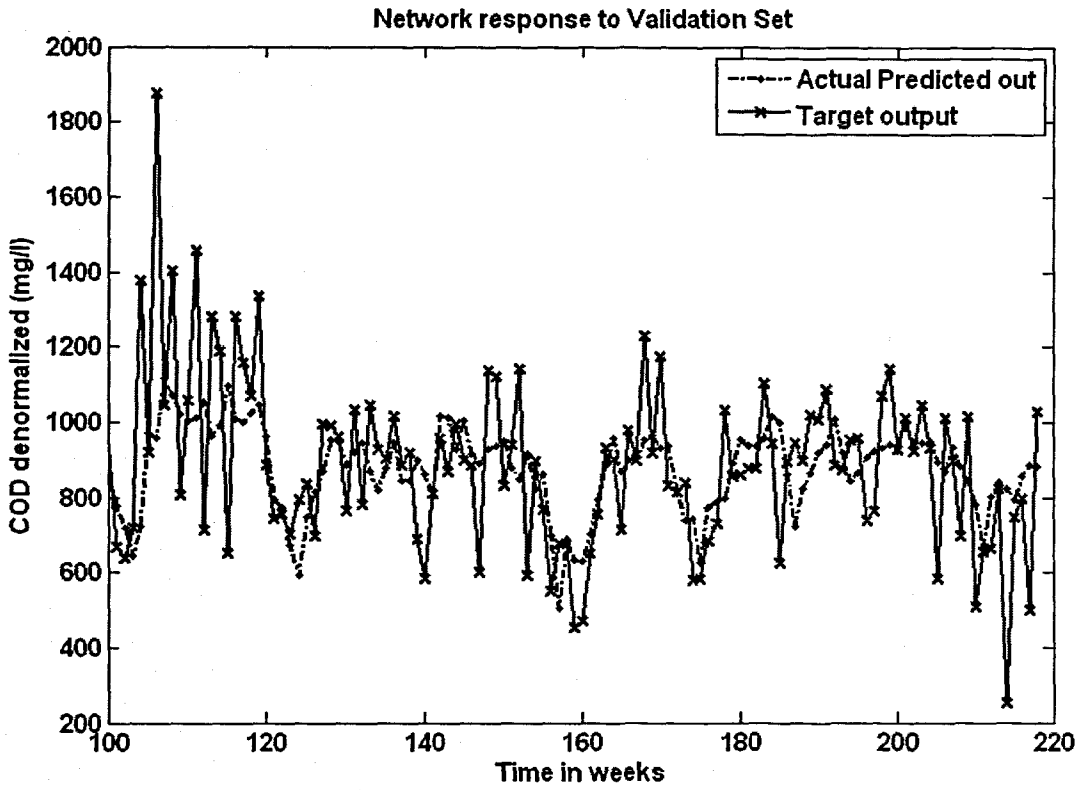


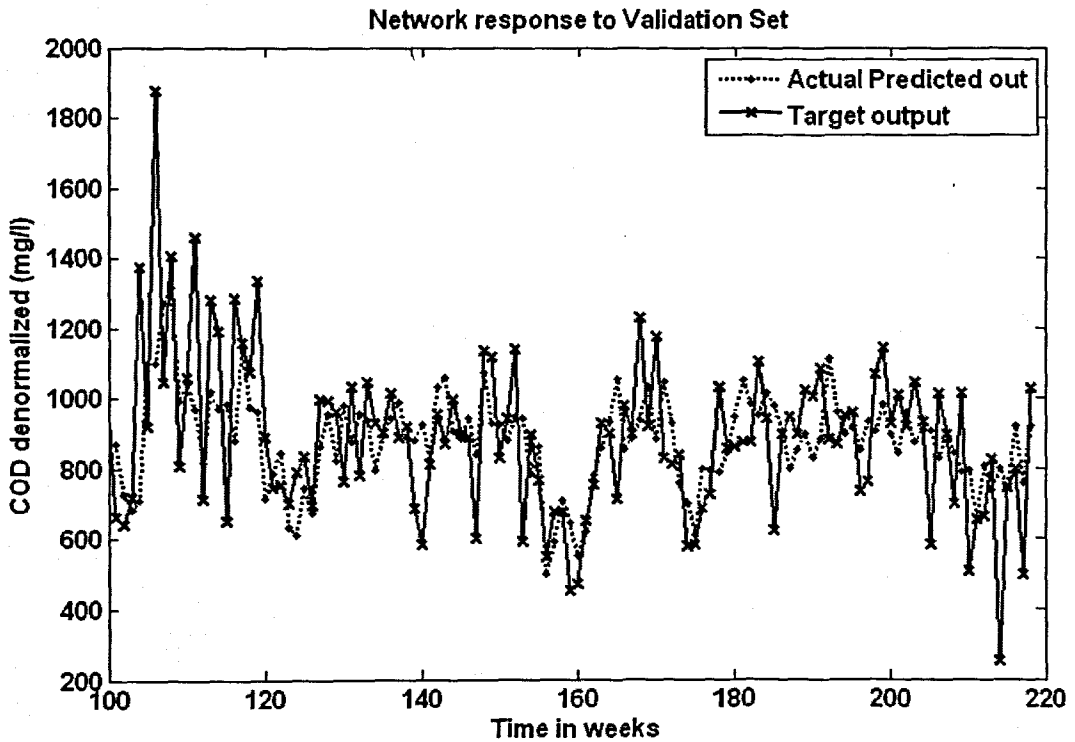
Figure 6. 30: (a) A MLP Feed-forward neural network Model 13 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 13 with 4 hidden neurons, 4 epochs



(a)



(b)



(c)

**Figure 6. 31:** The NN response to the application of the test (validation) data set for COD: (a) MLP FFNN Model 13 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 13 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 13 with 4 hidden neurons, 4 epochs

**Table 6. 50: Results for the Feed-forward Neural Network Model 13 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0117	0.6	0.5	0.435	0.400	0.189	0.160	-28.365	19.119
1	5	trainscg	tansig	purelin	500	MSE	0.0105	0.6	0.5	0.526	0.279	0.277	0.078	-31.638	20.473
1	10	trainscg	tansig	purelin	500	MSE	0.0090	0.6	0.5	0.615	0.291	0.379	0.085	-31.304	20.778
1	1	trainscg	tansig	purelin	1000	MSE	0.0117	0.6	0.5	0.435	0.400	0.190	0.160	-28.544	19.121
1	5	trainscg	tansig	purelin	1000	MSE	0.0098	0.6	0.5	0.567	0.285	0.321	0.081	-32.742	21.315
1	10	trainscg	tansig	purelin	1000	MSE	0.0084	0.6	0.5	0.640	0.274	0.410	0.075	-45.071	22.874

**Table 6. 51: Results for the Elman Recurrent Neural Network Model 13 with COD as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0118	0.01	0.5	0.433	0.393	0.188	0.154	-27.306	19.123
1	5	trainscg	tansig	purelin	500	MSE	0.0108	0.01	0.5	0.505	0.362	0.255	0.131	-27.215	19.835
1	10	trainscg	tansig	purelin	500	MSE	0.0096	0.01	0.5	0.582	0.361	0.339	0.131	-32.109	20.270
1	1	trainscg	tansig	purelin	1000	MSE	0.0117	0.01	0.5	0.435	0.400	0.190	0.160	-28.504	19.123
1	5	trainscg	tansig	purelin	1000	MSE	0.0098	0.01	0.5	0.568	0.230	0.323	0.053	-24.733	21.948
1	10	trainscg	tansig	purelin	1000	MSE	0.0081	0.01	0.5	0.662	0.229	0.438	0.052	-34.236	22.802

**Table 6. 52: Results for the Radial Basis Function Neural Network Model 13 with COD as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	4	4	SSE	5.493	5.5	0.2	0.361	0.248	0.130	0.061	-33.642	20.584

### 6.3 TKN Neural Network Results

The NN presented in this section is for the influent variable TKN as the predicted output of the different NNs. The NN model numbers as per the inputs are specified in tables at the beginning of the representation of the results for every model. All the models have only one hidden layer.

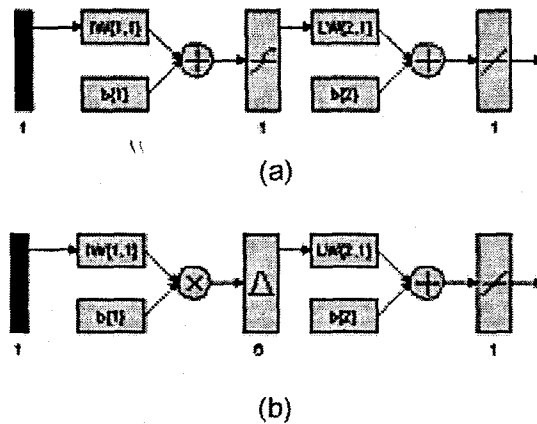
#### 6.3.1 TKN Neural Network MODEL 1

The input for model 1 is TKN(k) and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 53: Input variables for TKN Model 1**

Inputs	Output
TKN(k)	TKN(k+1)

The physical structure for the MLP FFNN and the RBF NN are represented in the following figure. Due to limitations on the MATLAB program for the visualisation of the physical NN, the Elman Recurrent NN is not displayed here.



**Figure 6. 32: (a) MLP Feed-forward NN Model 1 with 1 input, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 1 with 1 input, 6 hidden neurons**

For the NN with TKN as an output in this section 6.3, all tables and abbreviations remain the same as per the section in 6.2 for the NN with COD as output. The inputs are specified as per Table 5.2. For space considerations only for Model 1 all figures are included, the rest of the models only have the final NN response to the application of the test data set. However, please note that all results can be viewed using the program on the enclosed CD.



**Table 6.54: Results for the Feed-forward Neural Network Model 1 with TKN as output.**

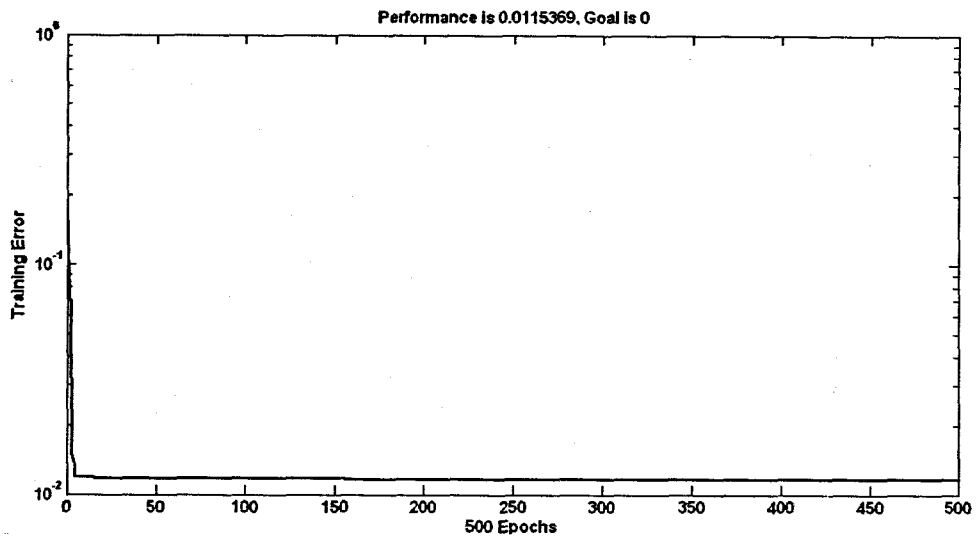
Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0115	0.6	0.5	0.548	0.440	0.301	0.194	-0.864	14.991
1	5	trainscg	tansig	purelin	500	MSE	0.0112	0.6	0.5	0.565	0.407	0.319	0.166	-1.123	15.344
1	10	trainscg	tansig	purelin	500	MSE	0.0110	0.6	0.5	0.576	0.379	0.332	0.144	-0.976	15.417
1	1	trainscg	tansig	purelin	1000	MSE	0.0115	0.6	0.5	0.551	0.435	0.304	0.189	-0.873	15.020
1	5	trainscg	tansig	purelin	1000	MSE	0.0112	0.6	0.5	0.565	0.405	0.319	0.164	-1.122	15.365
1	10	trainscg	tansig	purelin	1000	MSE	0.0110	0.6	0.5	0.577	0.382	0.333	0.146	-0.964	15.442

**Table 6.55: Results for the Elman Recurrent Neural Network Model 1 with TKN as output.**

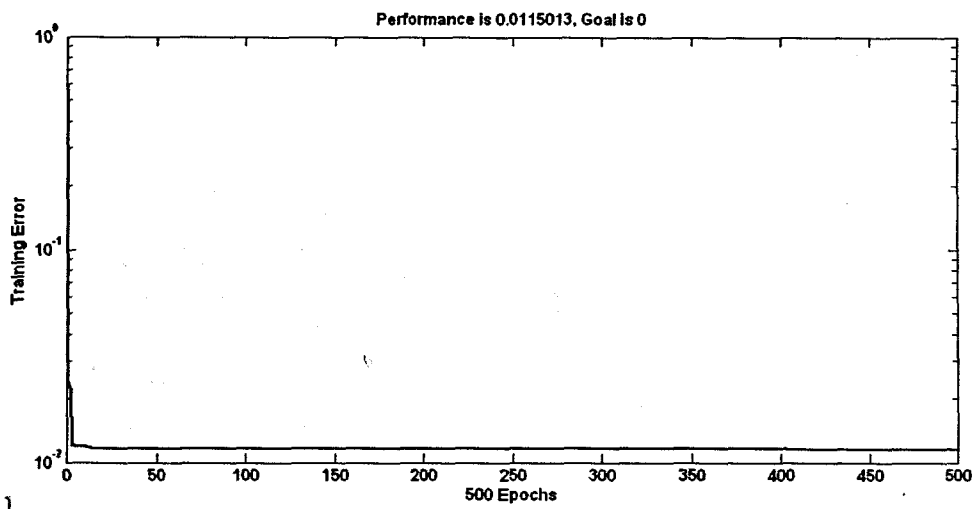
Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	414	MSE	0.0114	0.01	0.5	0.553	0.421	0.306	0.178	-0.899	15.112
1	5	trainscg	tansig	purelin	500	MSE	0.0115	0.01	0.5	0.550	0.424	0.303	0.180	-0.850	15.077
1	10	trainscg	tansig	purelin	500	MSE	0.0115	0.01	0.5	0.551	0.430	0.304	0.185	-0.882	15.027
1	1	trainscg	tansig	purelin	338	MSE	0.0114	0.01	0.5	0.553	0.421	0.306	0.177	-0.900	15.115
1	5	trainscg	tansig	purelin	1000	MSE	0.0113	0.01	0.5	0.562	0.405	0.316	0.164	-0.955	15.290
1	10	trainscg	tansig	purelin	1000	MSE	0.0115	0.01	0.5	0.549	0.426	0.302	0.182	-0.845	15.046

**Table 6.56: Results for the Radial Basis Function Neural Network Model 1 with TKN as output.**

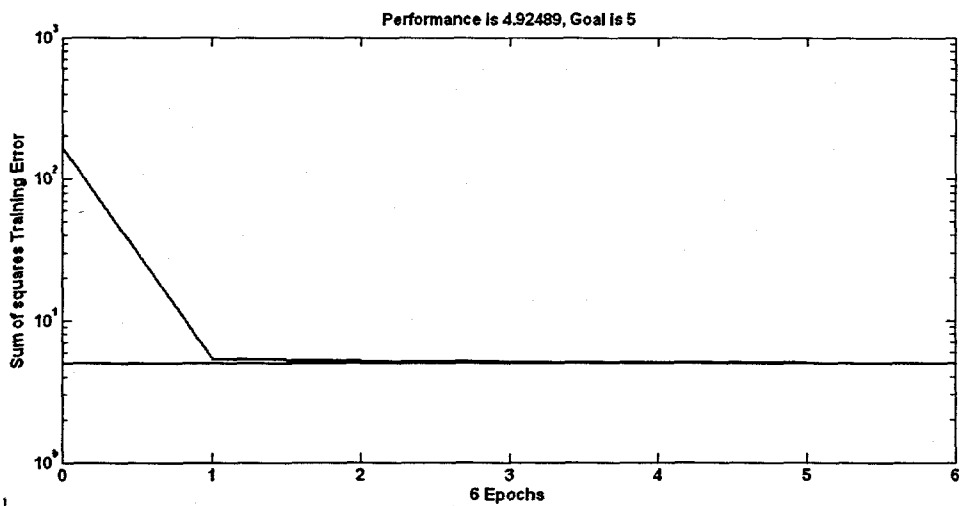
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	6	6	SSE	4.925	5	0.6	0.562	0.411	0.315	0.169	-0.962	15.235



(a)

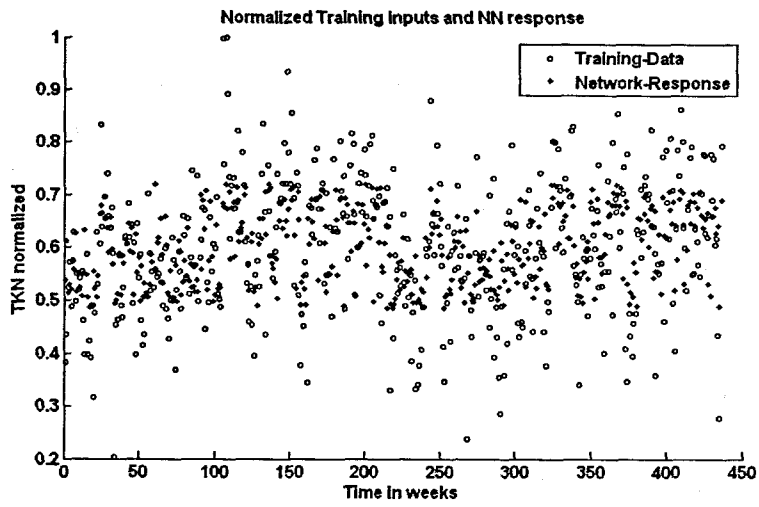


(b)

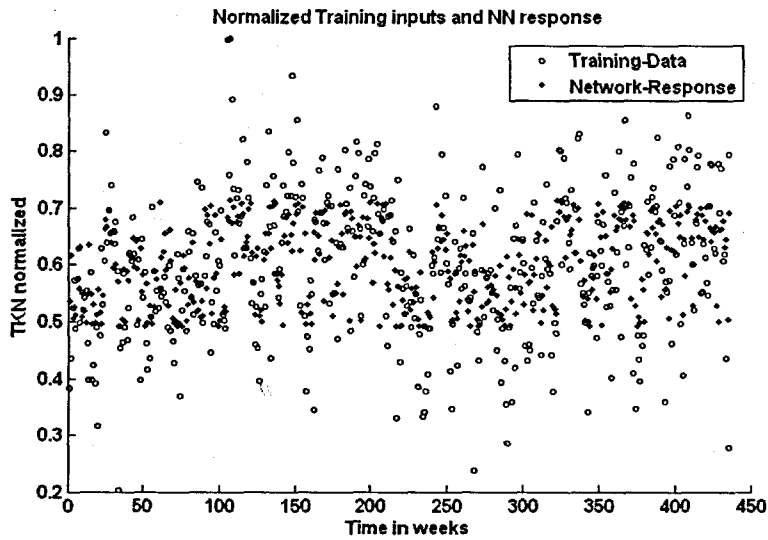


(c)

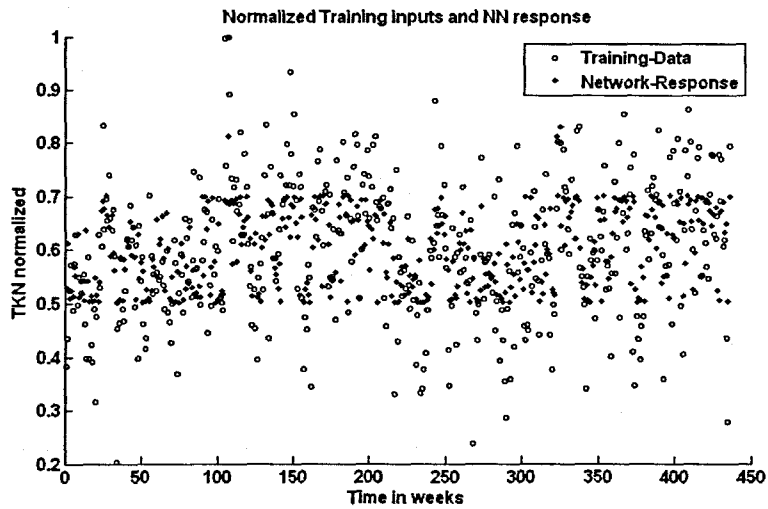
**Figure 6. 33: Training error versus the number of epochs for TKN: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 5 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs**



(a)

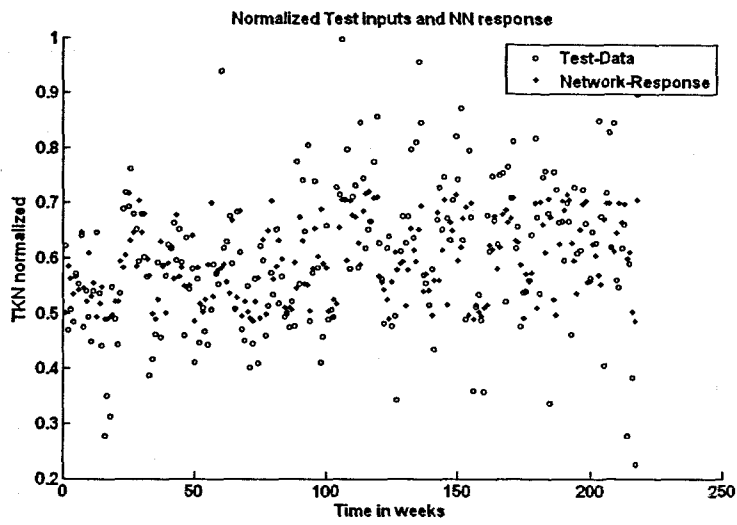


(b)

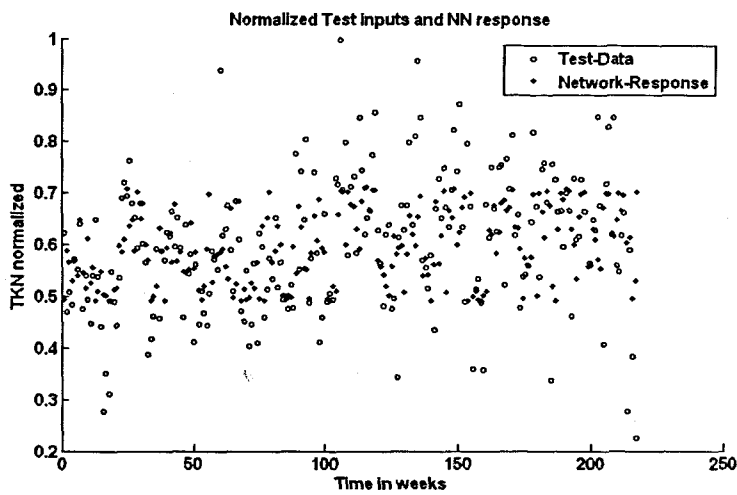


(c)

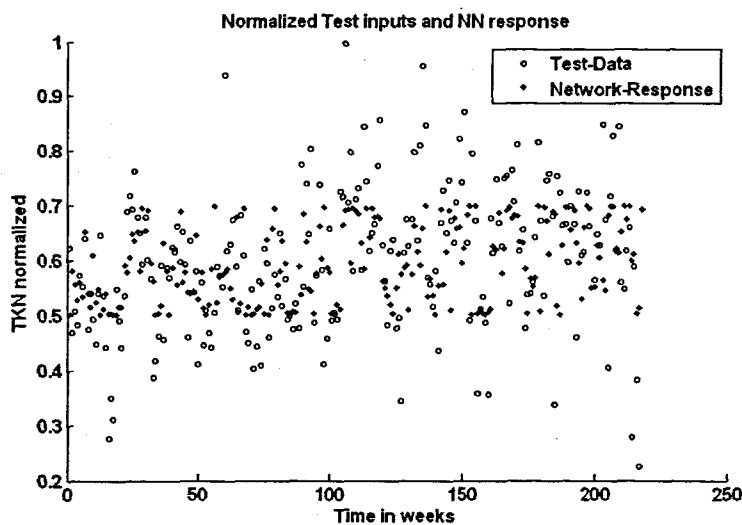
**Figure 6. 34:** A scatter-plot of the training input data and the NN response with the application of the training set for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs



(a)

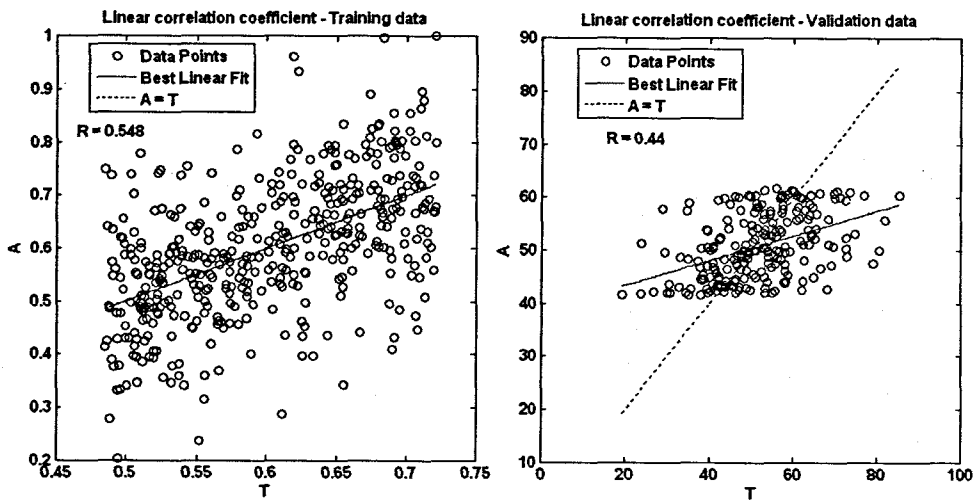


(b)

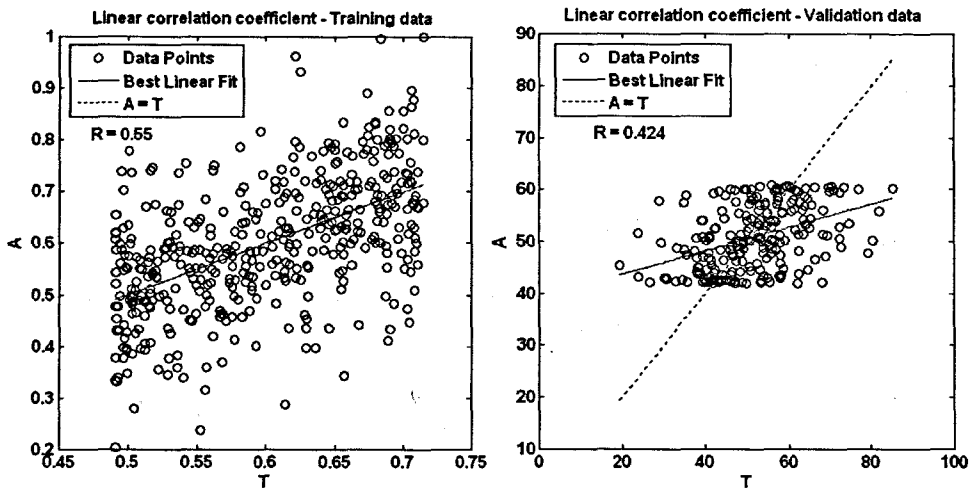


(c)

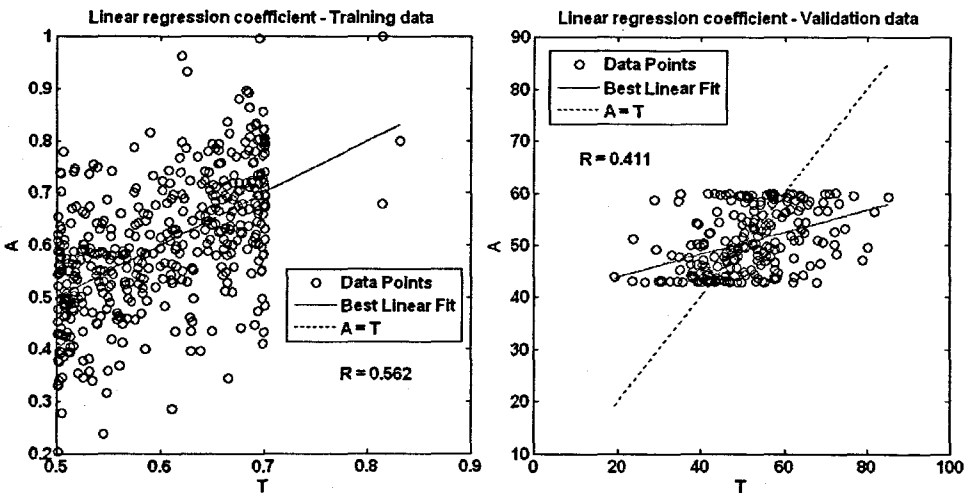
**Figure 6. 35:** A scatter-plot of the test (validation) data and the NN response with the application of the test set for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron, 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs



(a)

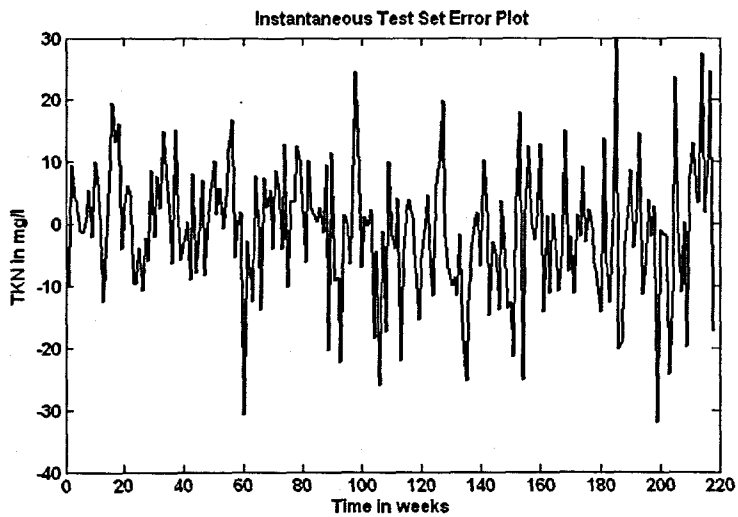
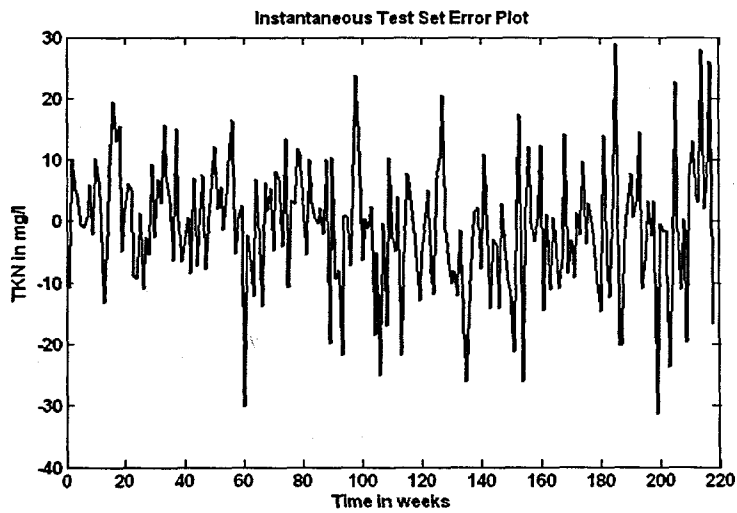
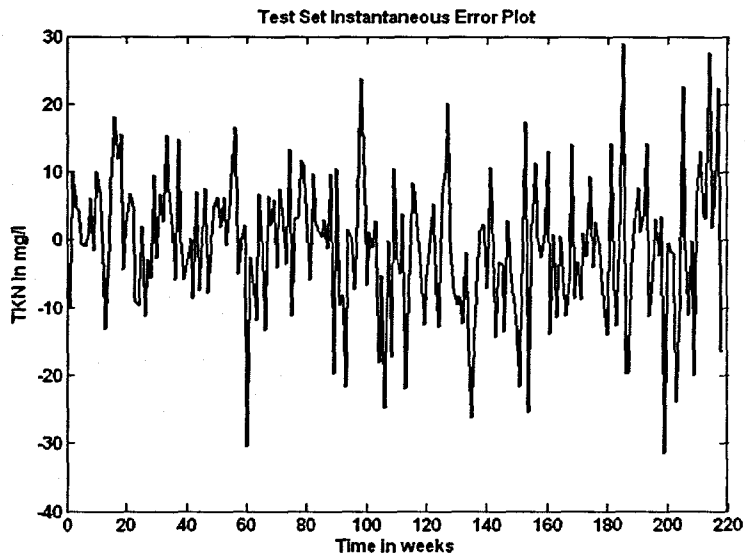


(b)

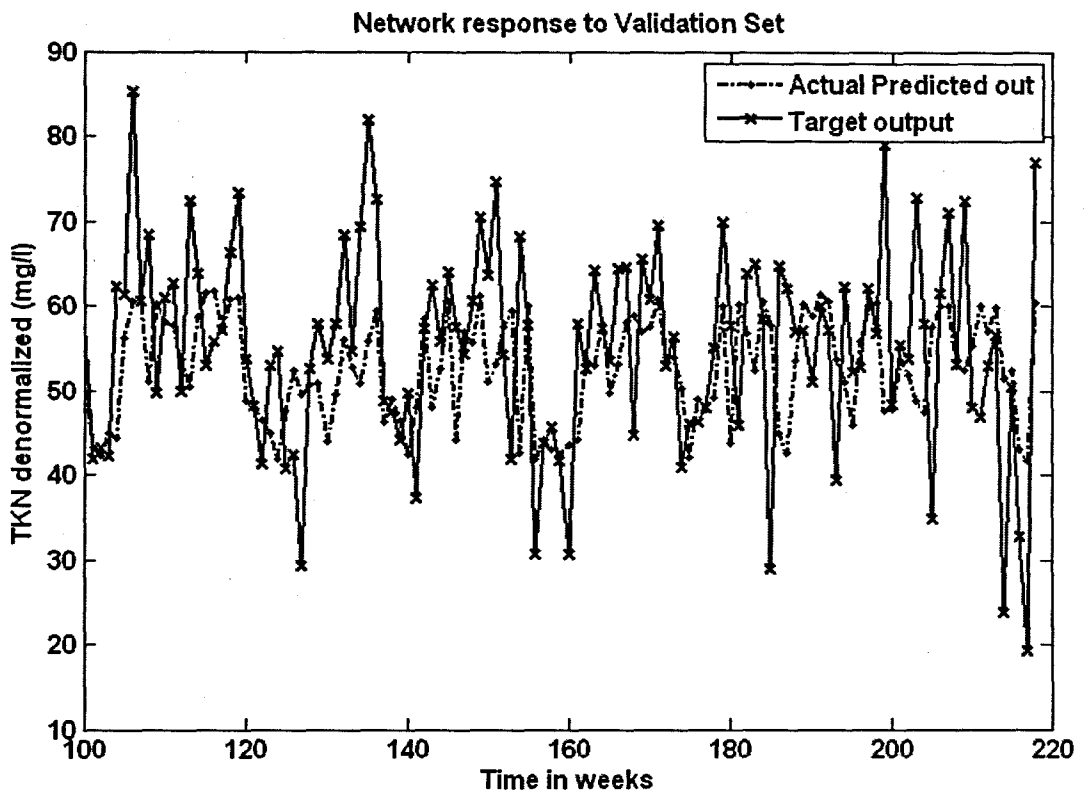


(c)

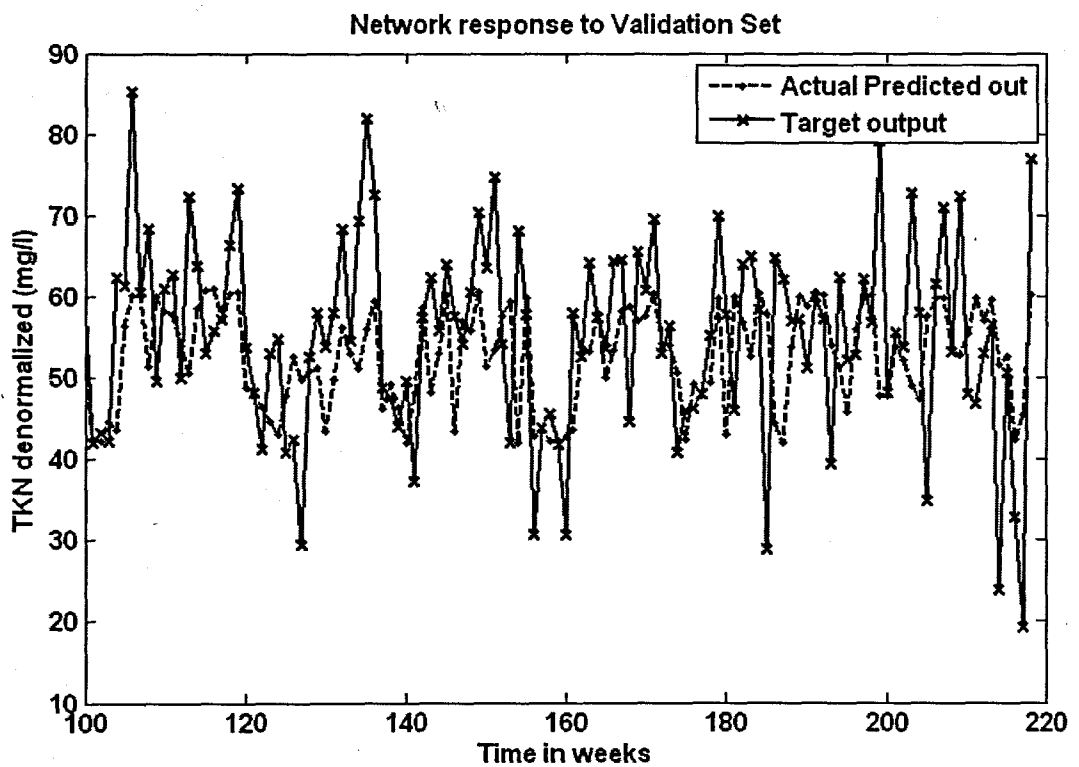
Figure 6. 36: Network performance showing the linear regression (correlation) coefficients for training data (normalized) on the left, and the validation data set (denormalized) on the right for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs, where A and T are the NN output and desired target output respectively



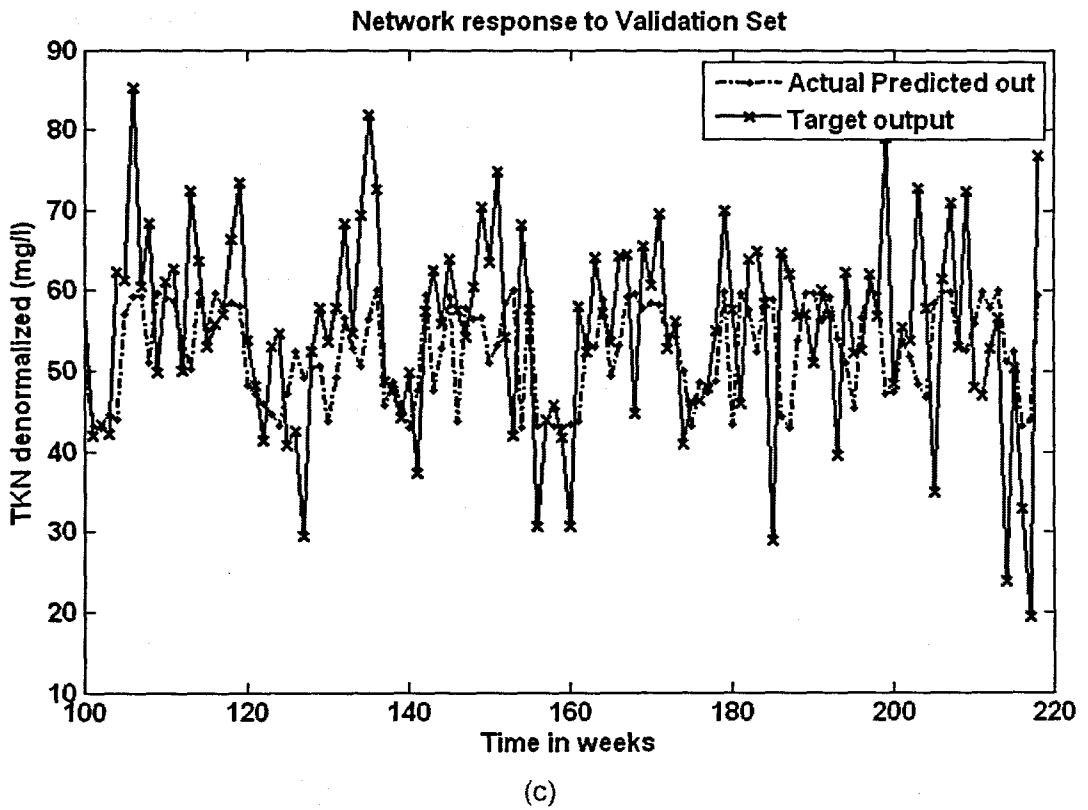
**Figure 6. 37:** The instantaneous error across the entire validation data set for the TKN: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs



(a)



(b)



**Figure 6. 38:** The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 6 hidden neurons, 6 epochs

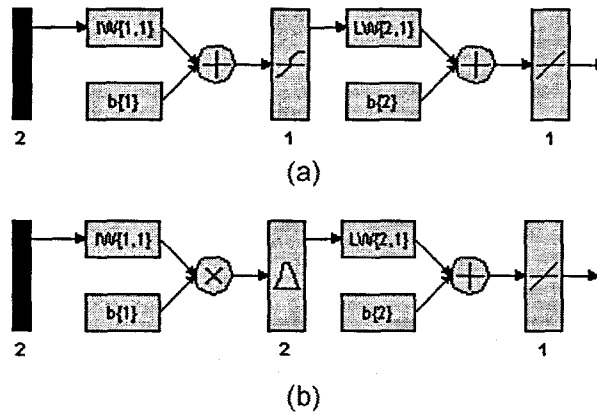


### 6.3.2 TKN Neural Network MODEL 2

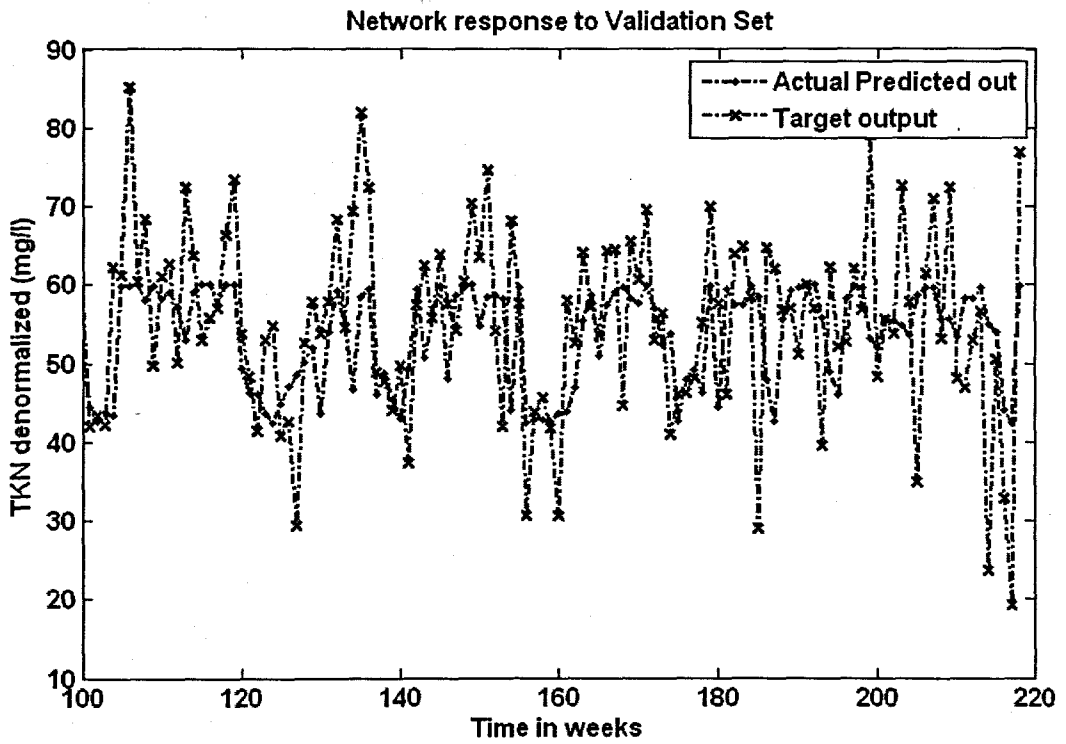
The inputs for Model 2 are TKN(k) and TKN(k-1) and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 57: Input variables for TKN Model 2**

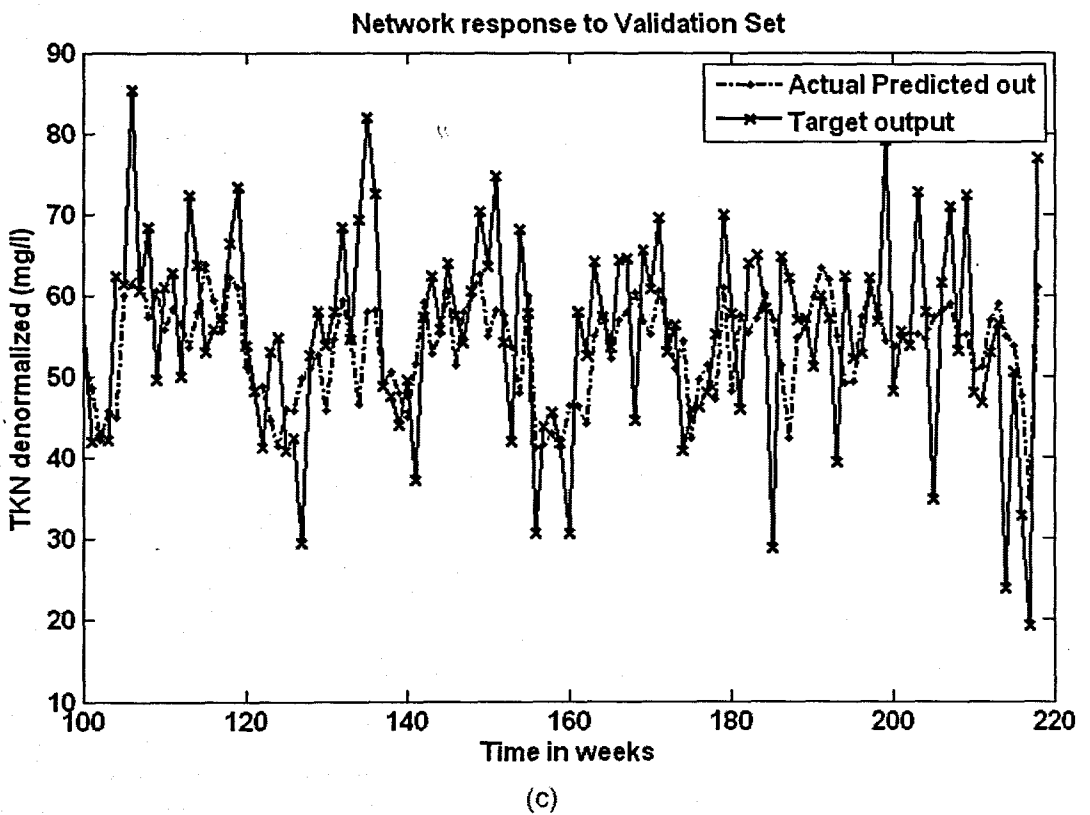
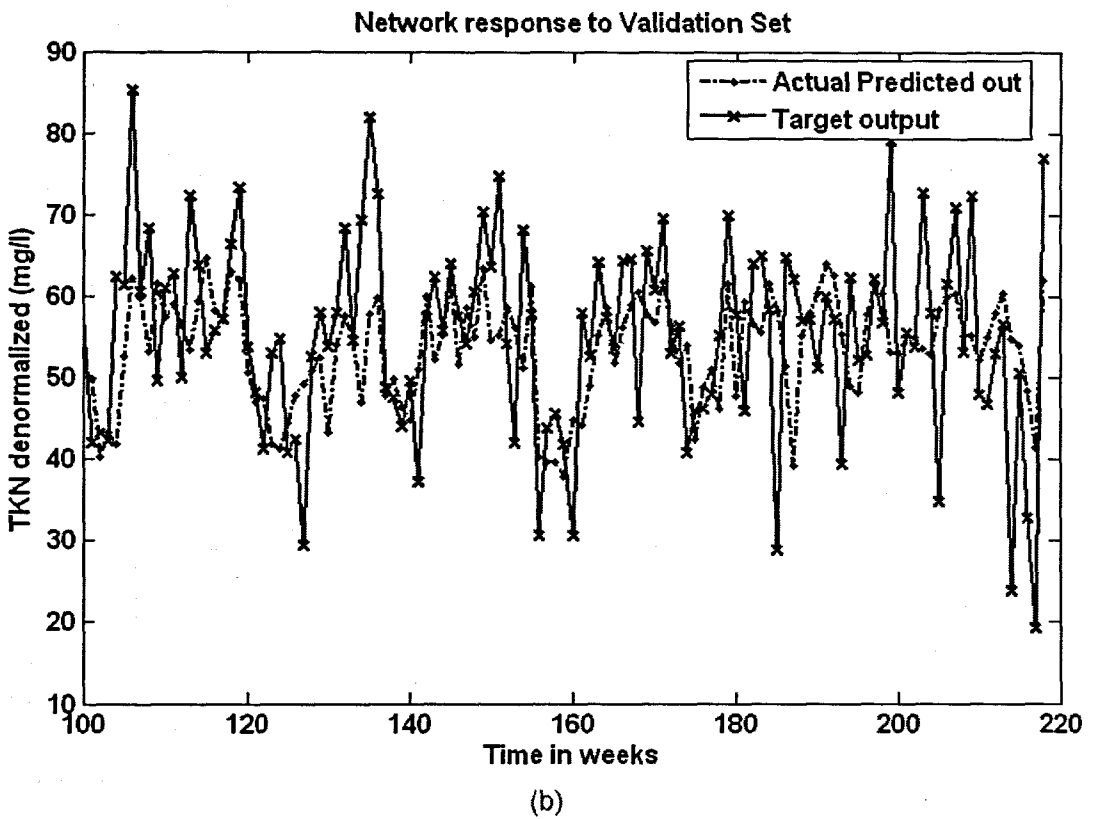
Inputs	Output
TKN(k), TKN(k-1)	TKN(k+1)



**Figure 6. 39: (a) A MLP Feed-forward neural network Model 2 with 2 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; (b) Block Diagram for the TKN RBFNN Model 2 with 2 hidden neurons**



(a)



**Figure 6. 40:** The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 2 with 1 hidden neuron and 500 epochs; (b) ERNN Model 2 with 10 hidden neurons, 500 epochs; and (c) RBFNN Model 2 with 2 hidden neurons, 2 epochs

**Table 6. 58: Results for the Feed-forward Neural Network Model 2 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	359	MSE	0.0109	0.6	0.5	0.581	0.491	0.338	0.241	-0.606	14.347
1	5	trainscg	tansig	purelin	500	MSE	0.0107	0.6	0.5	0.595	0.459	0.354	0.210	-0.599	14.776
1	10	trainscg	tansig	purelin	500	MSE	0.0104	0.6	0.5	0.609	0.484	0.371	0.234	-0.702	14.728
1	1	trainscg	tansig	purelin	324	MSE	0.011	0.6	0.5	0.581	0.491	0.338	0.241	-0.606	14.347
1	5	trainscg	tansig	purelin	1000	MSE	0.0104	0.6	0.5	0.609	0.482	0.371	0.232	-0.480	14.577
1	10	trainscg	tansig	purelin	1000	MSE	0.0101	0.6	0.5	0.623	0.471	0.388	0.221	-0.472	14.870

**Table 6. 59: Results for the Elman Recurrent Neural Network Model 2 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	328	MSE	0.0109	0.01	0.5	0.581	0.490	0.338	0.240	-0.607	14.352
1	5	trainscg	tansig	purelin	500	MSE	0.0107	0.01	0.5	0.595	0.494	0.354	0.244	-0.507	14.592
1	10	trainscg	tansig	purelin	500	MSE	0.0105	0.01	0.5	0.603	0.486	0.364	0.236	-0.559	14.700
1	1	trainscg	tansig	purelin	232	MSE	0.0110	0.01	0.5	0.581	0.491	0.338	0.241	-0.606	14.347
1	5	trainscg	tansig	purelin	1000	MSE	0.0106	0.01	0.5	0.599	0.481	0.358	0.232	-0.397	14.688
1	10	trainscg	tansig	purelin	1000	MSE	0.0105	0.01	0.5	0.603	0.482	0.364	0.232	-0.529	14.731

**Table 6. 60: Results for the Radial Basis Function Neural Network Model 2 with TKN as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	2	2	SSE	4.974	5	0.7	0.556	0.546	0.309	0.298	-0.340	13.873

### 6.3.3 TKN Neural Network MODEL 3

The inputs for Model 3 are  $TKN(k)$ ,  $TKN(k-1)$ ,  $TKN(k-2)$ , and the output is  $TKN(k+1)$  one time step ahead. These are tabled below:

Table 6. 61: Input variables for TKN Model 3

Inputs	Output
$TKN(k)$ , $TKN(k-1)$ , $TKN(k-2)$	$TKN(k+1)$

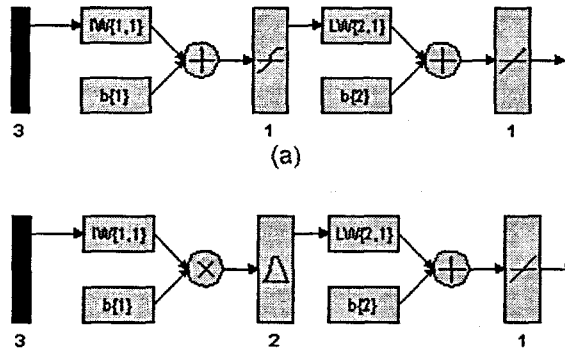
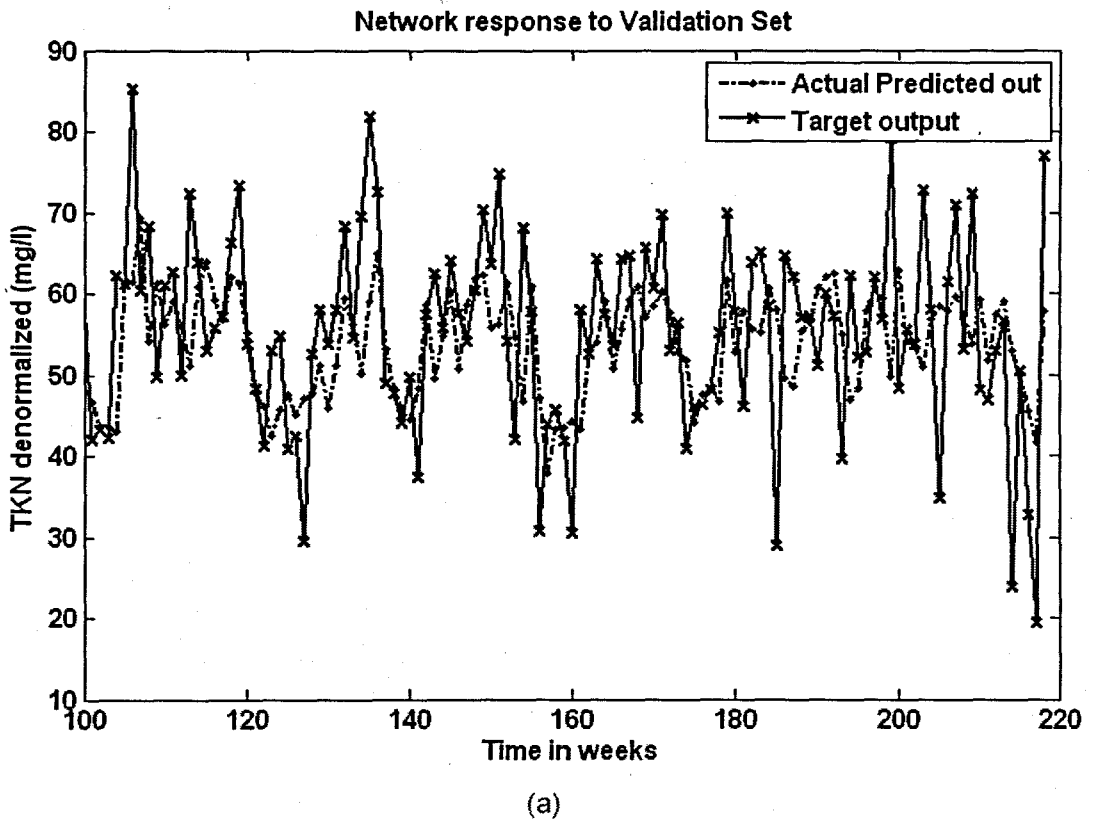
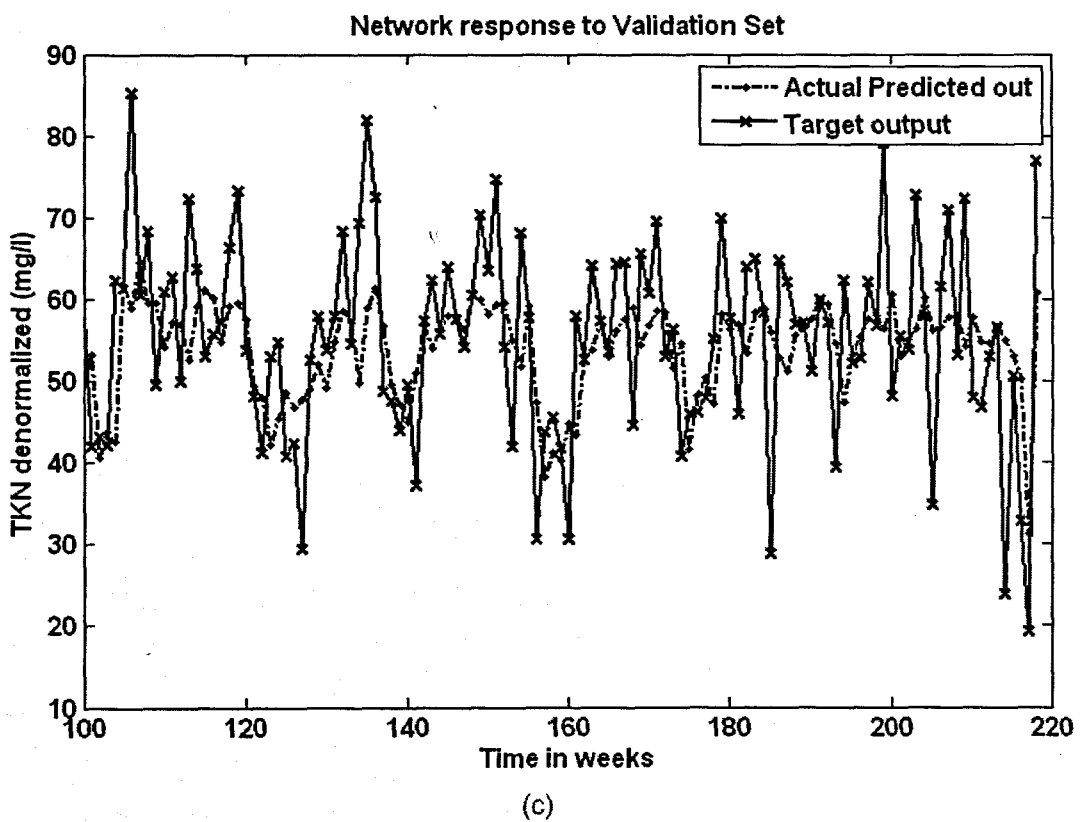
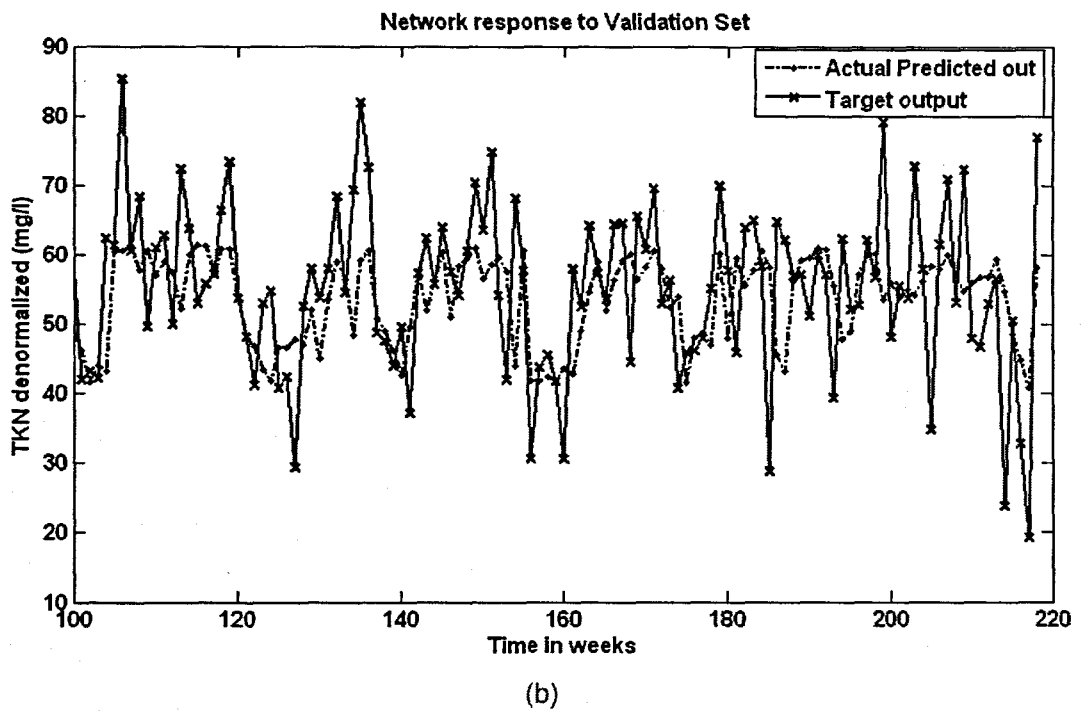


Figure 6. 41: (a) A MLP Feed-forward neural network Model 3 with 3 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 3 with 2 hidden neurons, 2 epochs





**Figure 6. 42:** The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 3 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 3 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 3 with 2 hidden neurons, 2 epochs

**Table 6. 62: Results for the Feed-forward Neural Network Model 3 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	271	MSE	0.0108	0.6	0.5	0.400	0.588	0.511	0.346	0.261	-0.547
1	5	trainscg	tansig	purelin	500	MSE	0.0102	0.6	0.5	0.619	0.485	0.383	0.235	-0.092	14.508
1	10	trainscg	tansig	purelin	500	MSE	0.0101	0.6	0.5	0.623	0.455	0.389	0.207	-0.294	14.994
1	1	trainscg	tansig	purelin	154	MSE	0.0108	0.6	0.5	0.400	0.588	0.511	0.346	0.261	-0.546
1	5	trainscg	tansig	purelin	1000	MSE	0.0103	0.6	0.5	0.463	0.614	0.495	0.376	0.245	-0.418
1	10	trainscg	tansig	purelin	1000	MSE	0.0095	0.6	0.5	0.651	0.426	0.424	0.181	-0.491	15.118

**Table 6. 63: Results for the Elman Recurrent Neural Network Model 3 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	166	MSE	0.0108	0.01	0.5	0.588	0.511	0.346	0.261	-0.546	14.156
1	5	trainscg	tansig	purelin	500	MSE	0.0105	0.01	0.5	0.601	0.514	0.361	0.264	-0.419	14.289
1	10	trainscg	tansig	purelin	500	MSE	0.0104	0.01	0.5	0.610	0.511	0.372	0.261	-0.416	14.453
1	1	trainscg	tansig	purelin	611	MSE	0.0108	0.01	0.5	0.588	0.511	0.346	0.261	-0.547	14.155
1	5	trainscg	tansig	purelin	1000	MSE	0.0105	0.01	0.5	0.605	0.517	0.366	0.267	-0.427	14.271
1	10	trainscg	tansig	purelin	1000	MSE	0.0101	0.01	0.5	0.621	0.503	0.386	0.253	-0.312	14.579

**Table 6. 64: Results for the Radial Basis Function Neural Network Model 3 with TKN as output.**

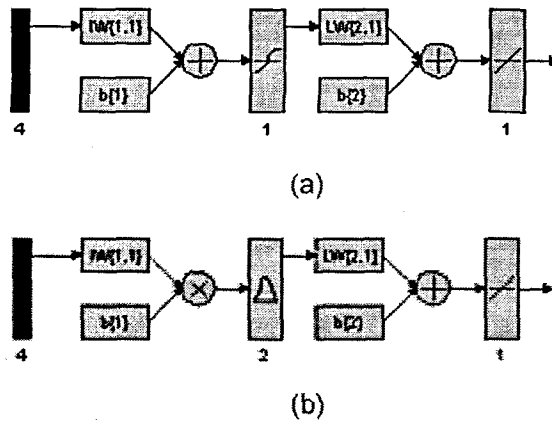
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	2	2	SSE	4.772	5	0.4	0.580	0.548	0.337	0.301	-0.391	13.963

### 6.3.4 TKN Neural Network MODEL 4

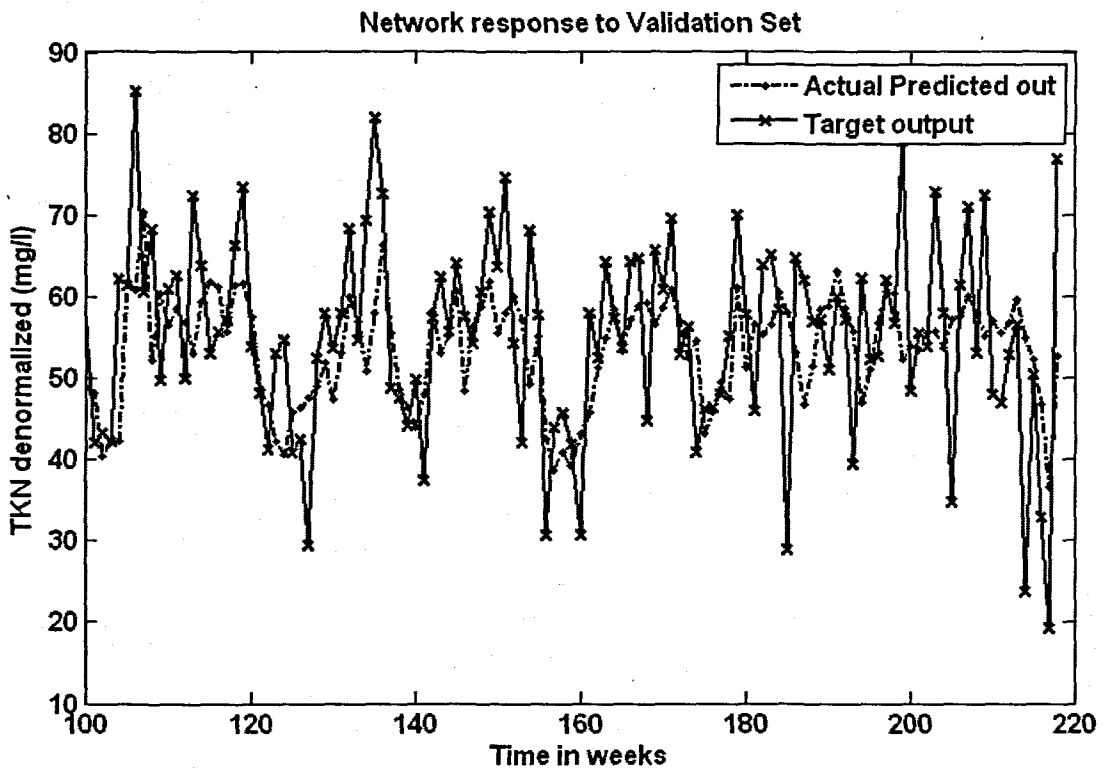
The inputs for Model 4 are  $TKN(k)$ ,  $TKN(k-1)$ ,  $TKN(k-2)$ ,  $TKN(k-3)$ , and the output is  $TKN(k+1)$  one time step ahead. These are tabled below:

**Table 6. 65: Input variables for TKN Model 4**

Inputs	Output
$TKN(k)$ , $TKN(k-1)$ , $TKN(k-2)$ , $TKN(k-3)$	$TKN(k+1)$



**Figure 6. 43: (a) A MLP Feed-forward neural network Model 4 with 4 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 4 with 2 hidden neurons, 2 epochs**



(a)

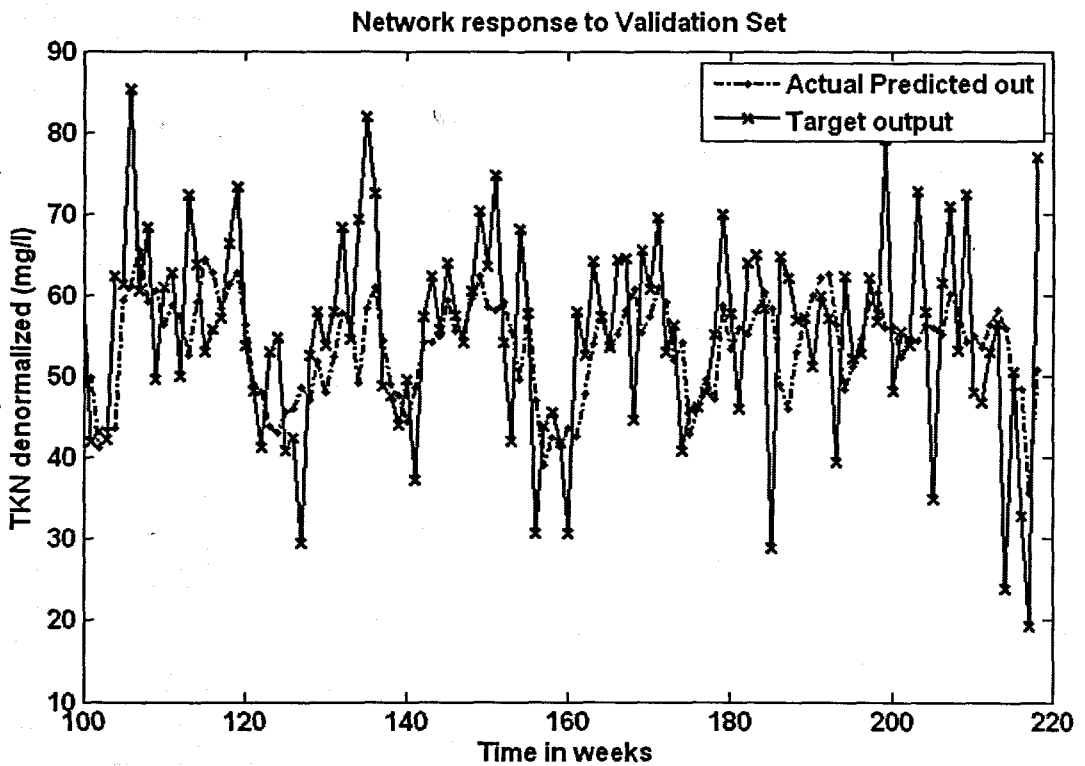
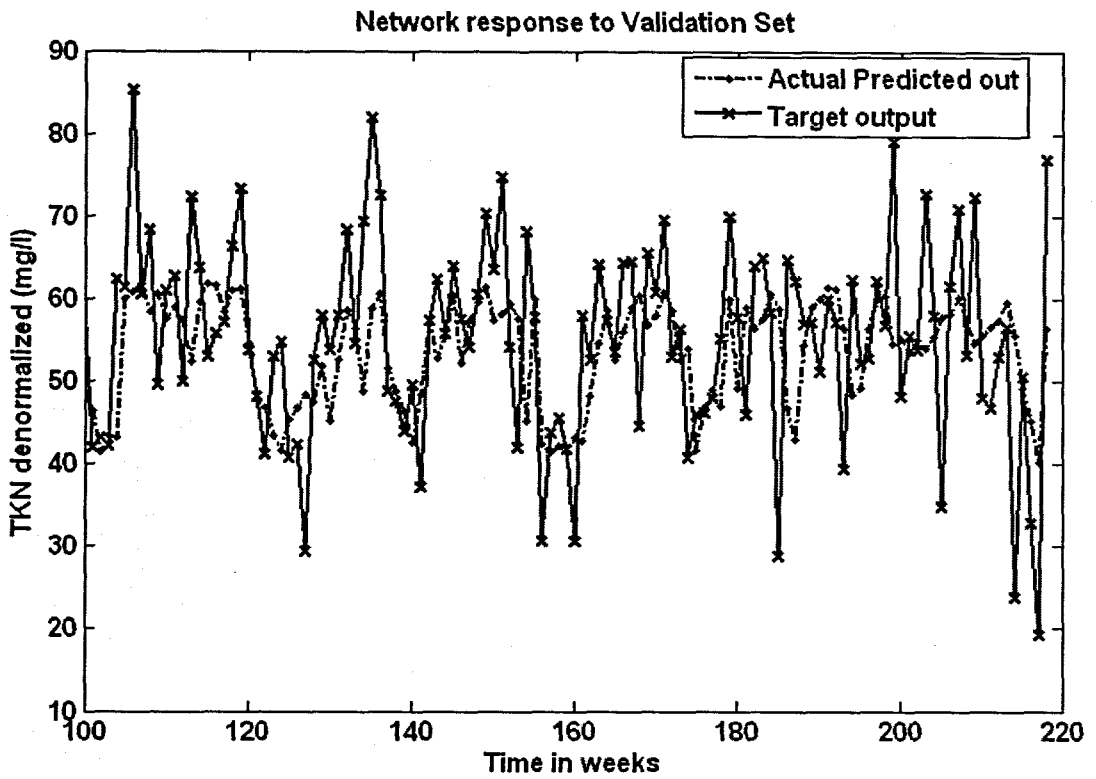


Figure 6. 44: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 4 with 10 hidden neurons and 500 epochs; (b) ERNN Model 4 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 4 with 2 hidden neurons, 2 epochs



**Table 6. 66: Results for the Feed-forward Neural Network Model 4 with-TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	102	MSE	0.0108	0.6	0.5	0.590	0.521	0.348	0.27	-0.588	14.052
1	5	trainscg	tansig	purelin	500	MSE	0.0103	0.6	0.5	0.613	0.513	0.376	0.263	-14.884	14.169
1	10	trainscg	tansig	purelin	500	MSE	0.0104	0.6	0.5	0.608	0.531	0.370	0.282	-0.593	14.153
1	1	trainscg	tansig	purelin	214	MSE	0.0108	0.6	0.5	0.608	0.531	0.370	0.282	-0.593	14.153
1	5	trainscg	tansig	purelin	1000	MSE	0.0101	0.6	0.5	0.624	0.384	0.390	0.147	-0.900	15.314
1	10	trainscg	tansig	purelin	1000	MSE	0.0096	0.6	0.5	0.648	0.488	0.420	0.238	-0.152	14.503

**Table 6. 67: Results for the Elman Recurrent Neural Network Model 4 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	379	MSE	0.0108	0.01	0.5	0.590	0.521	0.348	0.272	-0.588	14.052
1	5	trainscg	tansig	purelin	500	MSE	0.0107	0.01	0.5	0.593	0.543	0.351	0.295	-0.566	13.874
1	10	trainscg	tansig	purelin	500	MSE	0.0100	0.01	0.5	0.629	0.495	0.395	0.245	-0.226	14.613
1	1	trainscg	tansig	purelin	1000	MSE	0.0110	0.01	0.5	0.580	0.538	0.336	0.290	-0.577	14.119
1	5	trainscg	tansig	purelin	1000	MSE	0.0100	0.01	0.5	0.628	0.497	0.395	0.247	-0.604	14.575
1	10	trainscg	tansig	purelin	1000	MSE	0.0100	0.01	0.5	0.629	0.500	0.396	0.250	-0.701	14.536

**Table 6. 68: Results for the Radial Basis Function Neural Network Model 4 with TKN as output.**

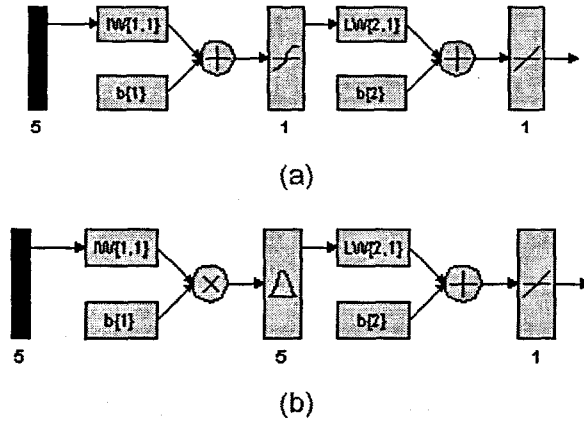
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	2	2	SSE	4.777	5	0.9	0.580	0.542	0.336	0.294	-0.538	13.949

### 6.3.5 TKN Neural Network MODEL 5

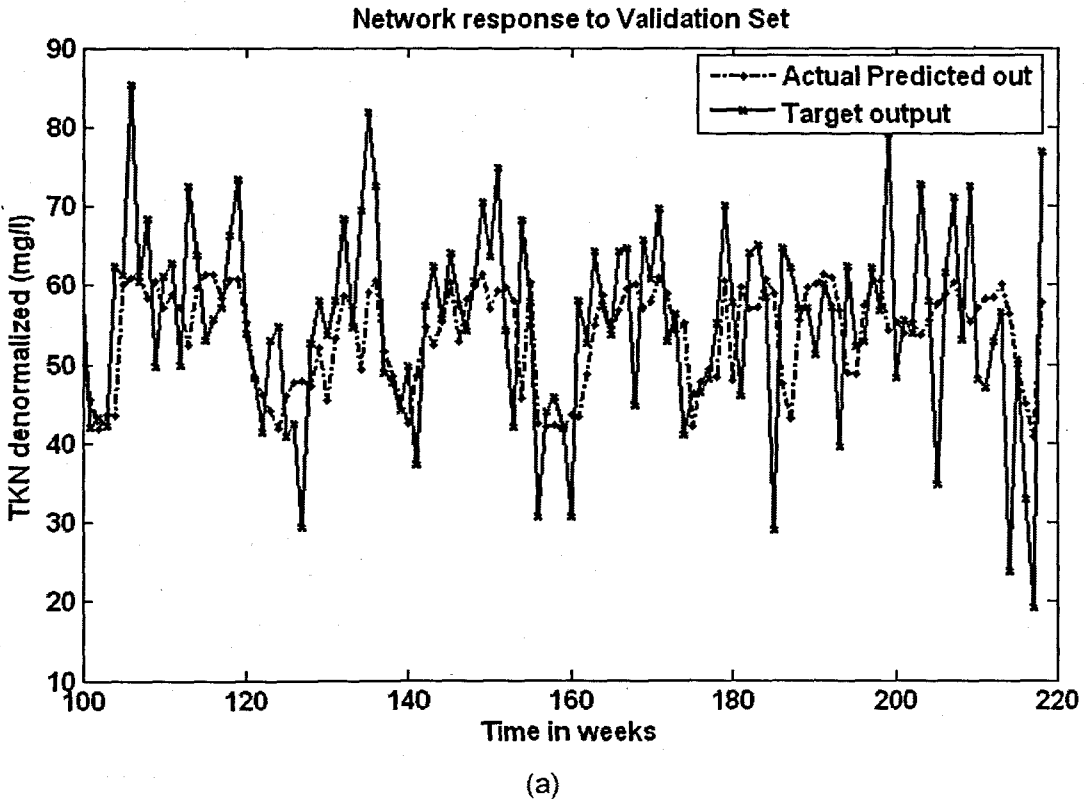
The inputs for Model 5 are TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 69: Input variables for TKN Model 5**

Inputs	Output
TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k)	TKN(k+1)



**Figure 6. 45:** (a) A MLP Feed-forward neural network Model 5 with 5 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 5 with 5 hidden neurons, 5 epochs



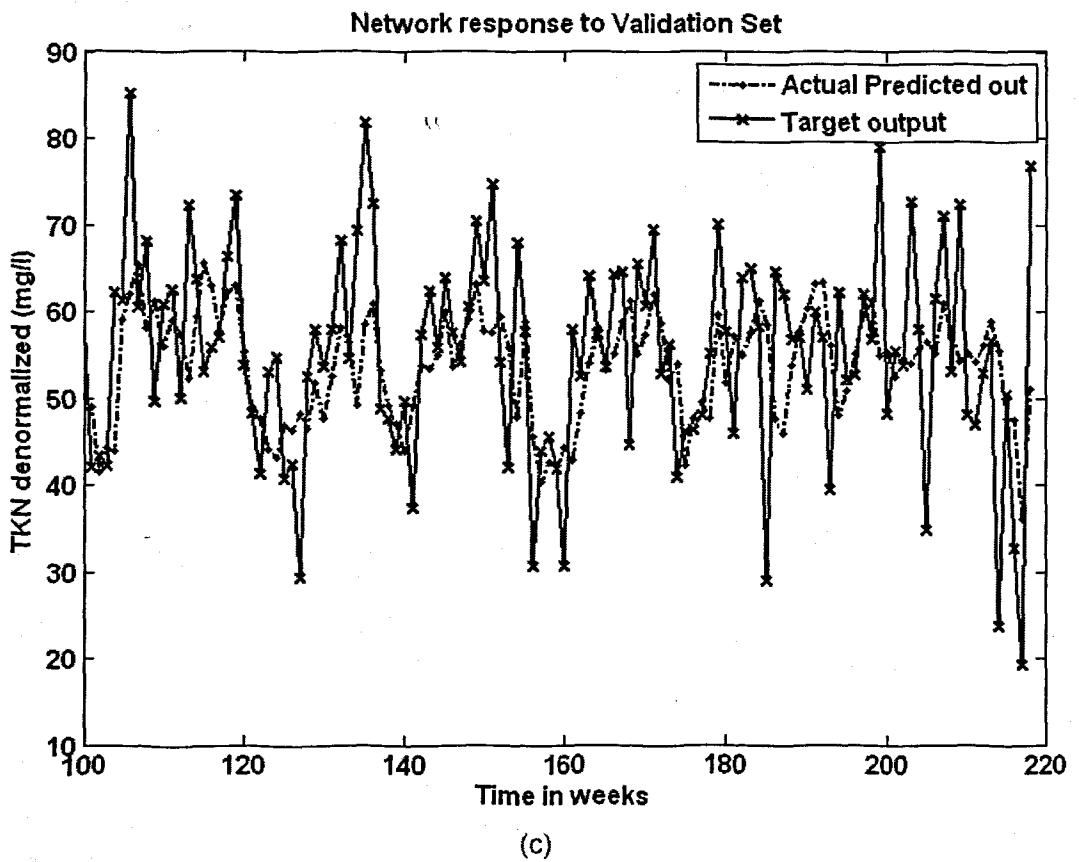
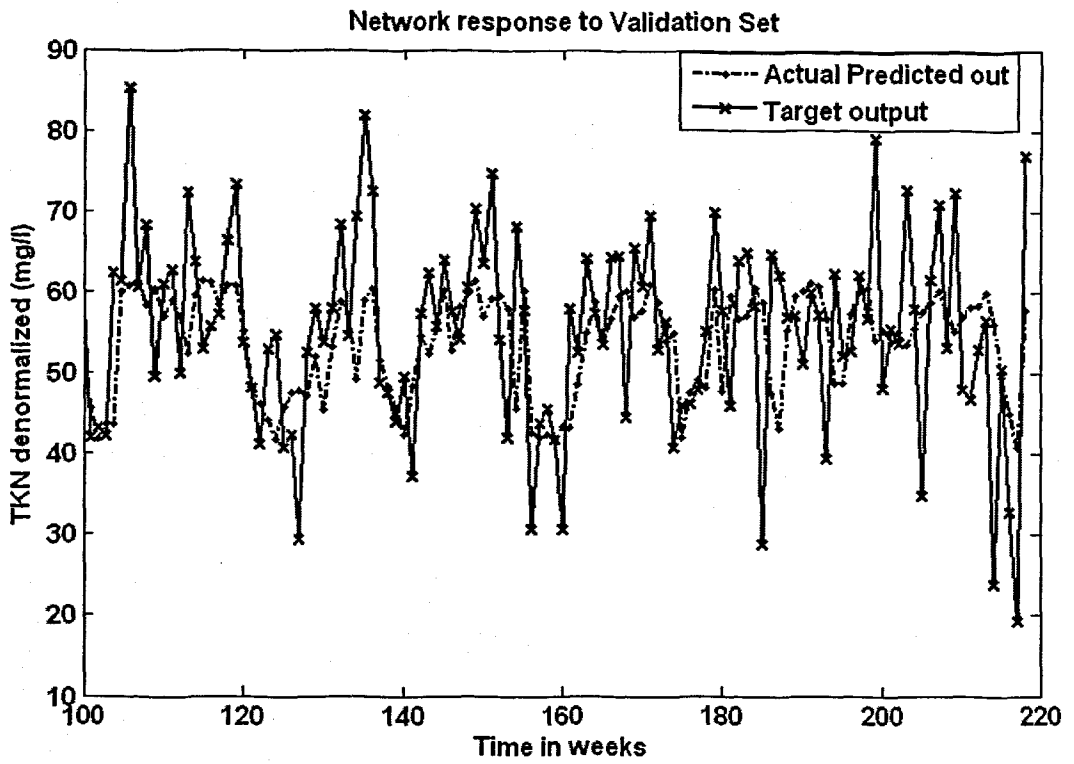


Figure 6. 46: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 5 with 1 hidden neuron and 500 epochs; (b) ERNN Model 5 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 5 with 5 hidden neurons, 5 epochs

**Table 6. 70: Results for the Feed-forward Neural Network Model 5 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	308	MSE	0.0107	0.6	0.5	0.593	0.524	0.352	0.275	-0.575	13.993
1	5	trainscg	tansig	purelin	500	MSE	0.0103	0.6	0.5	0.611	0.502	0.373	0.252	-0.388	14.370
1	10	trainscg	tansig	purelin	500	MSE	0.0098	0.6	0.5	0.635	0.481	0.404	0.231	-0.259	14.866
1	1	trainscg	tansig	purelin	271	MSE	0.0107	0.6	0.5	0.593	0.524	0.352	0.275	-0.575	13.993
1	5	trainscg	tansig	purelin	1000	MSE	0.0098	0.6	0.5	0.637	0.487	0.405	0.238	-0.371	14.464
1	10	trainscg	tansig	purelin	1000	MSE	0.0098	0.6	0.5	0.656	0.436	0.430	0.190	-0.400	15.206

**Table 6. 71: Results for the Elman Recurrent Neural Network Model 5 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	404	MSE	0.0107	0.01	0.5	0.593	0.524	0.352	0.275	-0.575	13.993
1	5	trainscg	tansig	purelin	500	MSE	0.0106	0.01	0.5	0.599	0.532	0.359	0.283	-0.457	14.051
1	10	trainscg	tansig	purelin	500	MSE	0.0104	0.01	0.5	0.644	0.474	0.415	0.224	-0.306	14.714
1	1	trainscg	tansig	purelin	303	MSE	0.0107	0.01	0.5	0.593	0.524	0.352	0.275	-0.575	13.993
1	5	trainscg	tansig	purelin	1000	MSE	0.0099	0.01	0.5	0.634	0.487	0.403	0.237	-0.488	14.515
1	10	trainscg	tansig	purelin	1000	MSE	0.0098	0.01	0.5	0.636	0.468	0.405	0.219	-0.716	14.795

**Table 6. 72: Results for the Radial Basis Function Neural Network Model 5 with TKN as output.**

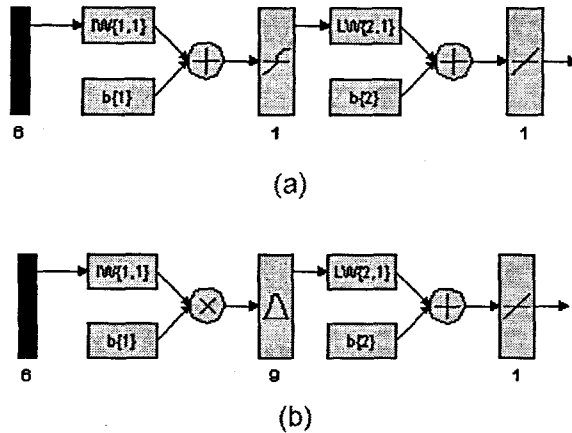
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	5	5	SSSE	4.752	4.8	0.7	0.583	0.537	0.339	0.289	-0.524	14.044

### 6.3.6 TKN Neural Network MODEL 6

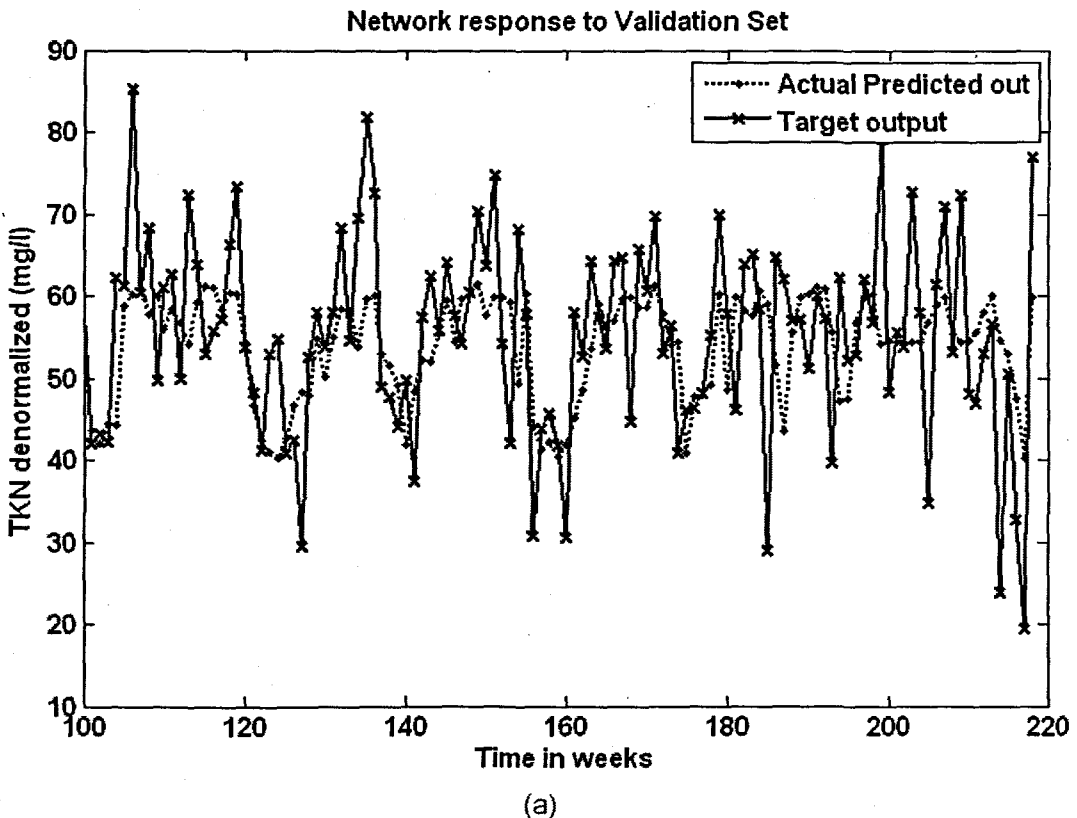
The inputs for Model 6 are  $TKN(k)$ ,  $TKN(k-1)$ ,  $TKN(k-2)$ ,  $TKN(k-3)$ ,  $COD(k)$ ,  $FLOW(k)$ , and the output is  $TKN(k+1)$  one time step ahead. These are tabled below:

**Table 6. 73: Input variables for TKN Model 6**

Inputs	Output
$TKN(k)$ , $TKN(k-1)$ , $TKN(k-2)$ , $TKN(k-3)$ , $COD(k)$ , $FLOW(k)$	$TKN(k+1)$



**Figure 6. 47: (a) A MLP Feed-forward neural network Model 6 with 6 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 6 with 9 hidden neurons, 9 epochs**



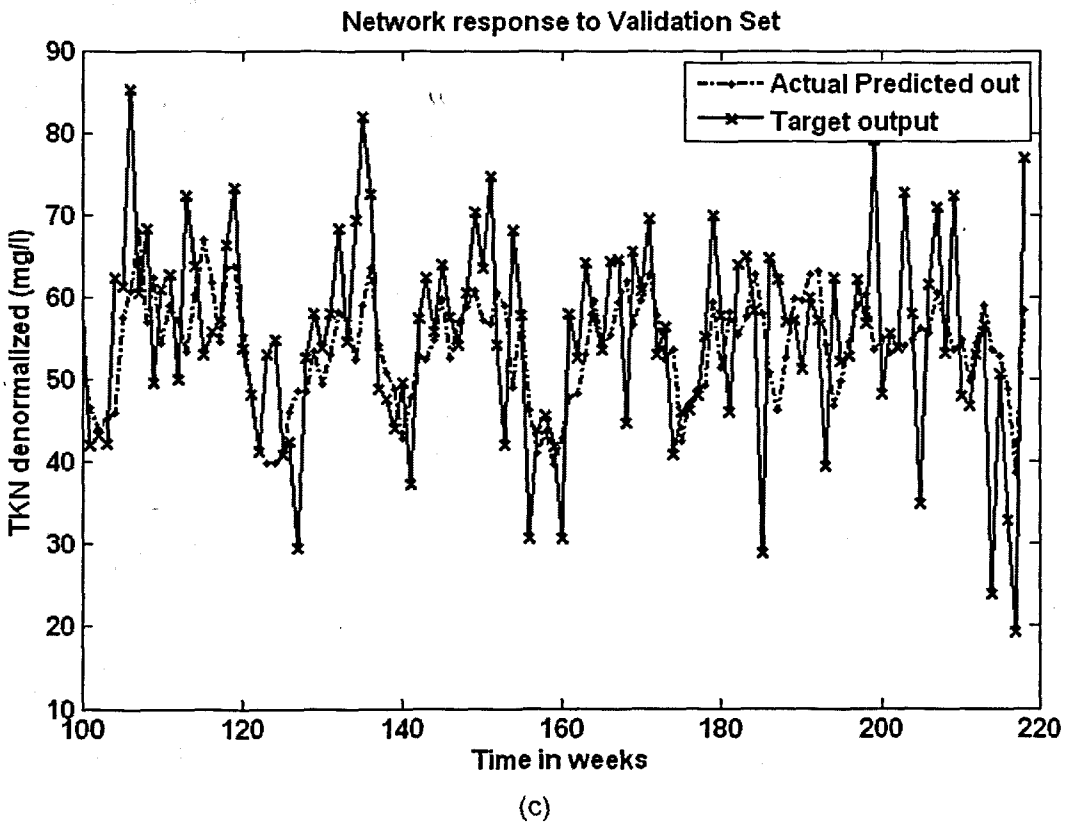
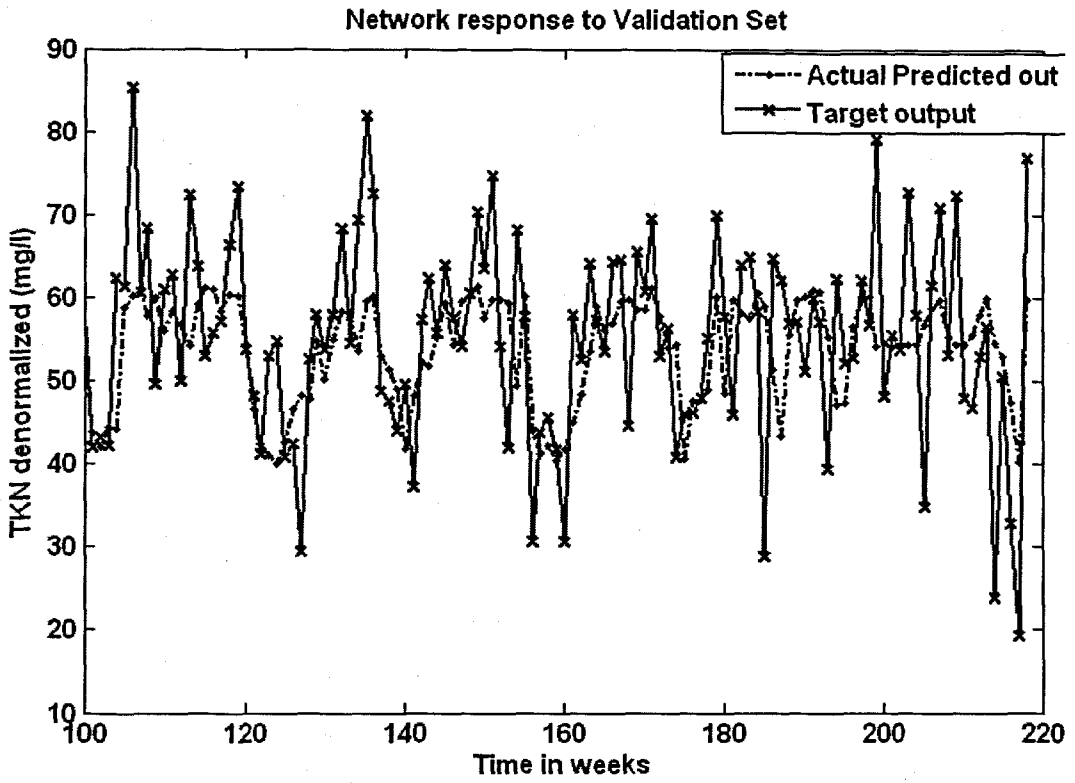


Figure 6. 48: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 6 with 1 hidden neuron and 500 epochs; (b) ERNN Model 6 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 6 with 9 hidden neurons, 9 epochs

**Table 6. 74: Results for the Feed-forward Neural Network Model 6 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	145	MSE	0.0104	0.6	0.5	0.606	0.556	0.367	0.309	-0.503	13.701
1	5	trainscg	tansig	purelin	500	MSE	0.0100	0.6	0.5	0.628	0.532	0.394	0.283	-0.396	14.190
1	10	trainscg	tansig	purelin	500	MSE	0.0096	0.6	0.5	0.648	0.549	0.420	0.302	-0.427	14.042
1	1	trainscg	tansig	purelin	178	MSE	0.0104	0.6	0.5	0.606	0.556	0.367	0.309	-0.503	13.701
1	5	trainscg	tansig	purelin	1000	MSE	0.0091	0.6	0.5	0.670	0.514	0.448	0.264	-0.096	14.415
1	10	trainscg	tansig	purelin	1000	MSE	0.0090	0.6	0.5	0.676	0.511	0.457	0.261	0.006	14.283

**Table 6. 75: Results for the Elman Recurrent Neural Network Model 6 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	147	MSE	0.0104	0.01	0.5	0.606	0.556	0.367	0.309	-0.502	13.700
1	5	trainscg	tansig	purelin	500	MSE	0.0097	0.01	0.5	0.643	0.505	0.413	0.255	-0.428	14.364
1	10	trainscg	tansig	purelin	500	MSE	0.0097	0.01	0.5	0.644	0.494	0.414	0.244	-0.264	14.743
1	1	trainscg	tansig	purelin	234	MSE	0.0104	0.01	0.5	0.606	0.556	0.367	0.309	-0.502	13.701
1	5	trainscg	tansig	purelin	1000	MSE	0.0093	0.01	0.5	0.660	0.497	0.435	0.247	-0.319	14.354
1	10	trainscg	tansig	purelin	1000	MSE	0.0089	0.01	0.5	0.677	0.530	0.458	0.281	-0.008	14.208

**Table 6. 76: Results for the Radial Basis Function Neural Network Model 6 with TKN as output.**

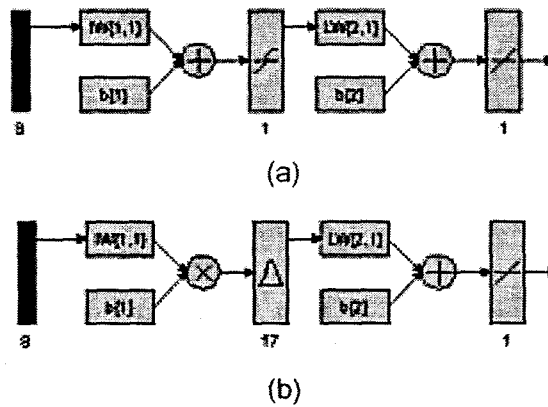
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	9	9	SSE	4.528	4.6	0.7	0.609	0.554	0.371	0.307	-0.517	13.944

### 6.3.7 TKN Neural Network MODEL 7

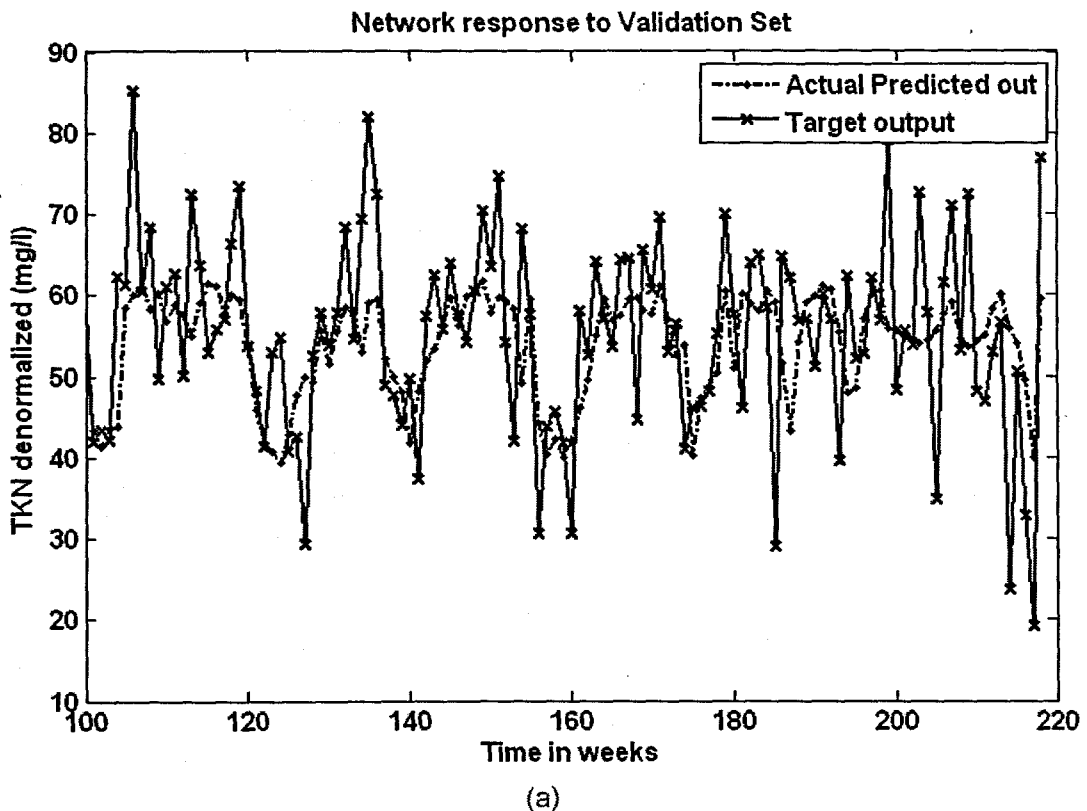
The inputs for Model 7 are TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), MONTH, and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 77: Input variables for TKN Model 7**

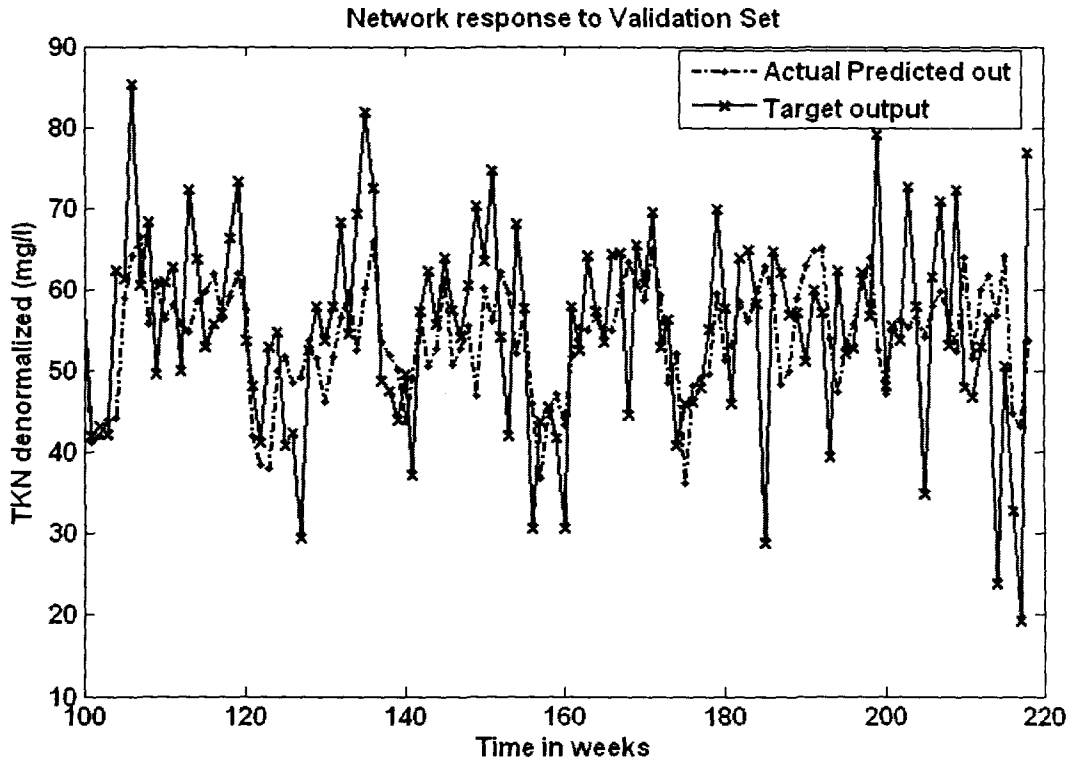
Inputs	Output
TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), Month	TKN(k+1)



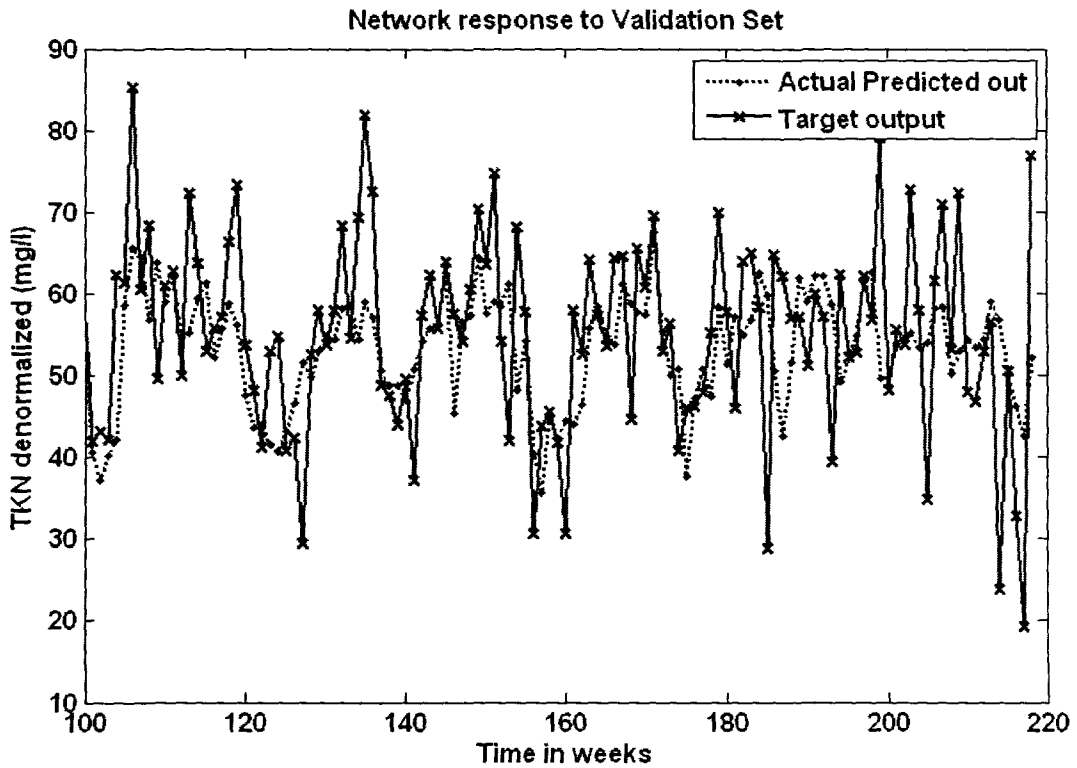
**Figure 6. 49: (a) A MLP Feed-forward neural network Model 7 with 8 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the COD RBFNN Model 7 with 17 hidden neurons, 17 epochs**







(b)



(c)

**Figure 6. 50:** The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 7 with 1 hidden neuron and 500 epochs; (b) ERNN Model 7 with 5 hidden neurons, 1000 epochs; and (c) RBFNN Model 7 with 17 hidden neurons, 17 epochs

**Table 6. 78: Results for the Feed-forward Neural Network Model 7 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	187	MSE	0.0117	0.6	0.5	0.609	0.566	0.371	0.320	-0.448	13.470
1	5	trainscg	tansig	purelin	500	MSE	0.0096	0.6	0.5	0.647	0.532	0.419	0.283	-0.346	14.022
1	10	trainscg	tansig	purelin	500	MSE	0.0094	0.6	0.5	0.654	0.558	0.428	0.311	-0.345	13.381
1	1	trainscg	tansig	purelin	407	MSE	0.0117	0.6	0.5	0.609	0.566	0.371	0.320	-0.449	13.471
1	5	trainscg	tansig	purelin	1000	MSE	0.0091	0.6	0.5	0.670	0.561	0.450	0.314	-0.322	13.463
1	10	trainscg	tansig	purelin	1000	MSE	0.0085	0.6	0.5	0.697	0.514	0.486	0.264	-0.007	14.448

**Table 6. 79: Results for the Elman Recurrent Neural Network Model 7 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	347	MSE	0.0104	0.01	0.5	0.609	0.566	0.371	0.320	-0.448	13.471
1	5	trainscg	tansig	purelin	500	MSE	0.0096	0.01	0.5	0.647	0.515	0.419	0.265	-0.688	14.233
1	10	trainscg	tansig	purelin	500	MSE	0.0088	0.01	0.5	0.682	0.509	0.466	0.259	-0.550	14.879
1	1	trainscg	tansig	purelin	1000	MSE	0.0105	0.01	0.5	0.602	0.565	0.362	0.319	-0.548	13.638
1	5	trainscg	tansig	purelin	1000	MSE	0.0089	0.01	0.5	0.677	0.511	0.458	0.261	-0.196	14.315
1	10	trainscg	tansig	purelin	1000	MSE	0.0085	0.01	0.5	0.697	0.507	0.486	0.257	-0.475	14.559

**Table 6. 80: Results for the Radial Basis Function Neural Network Model 7 with TKN as output.**

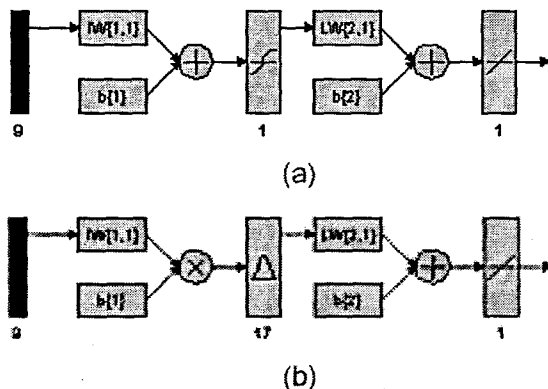
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	17	17	SSE	4.383	4.4	0.8	0.625	0.536	0.391	0.287	-0.755	13.963

### 6.3.8 TKN Neural Network MODEL 8

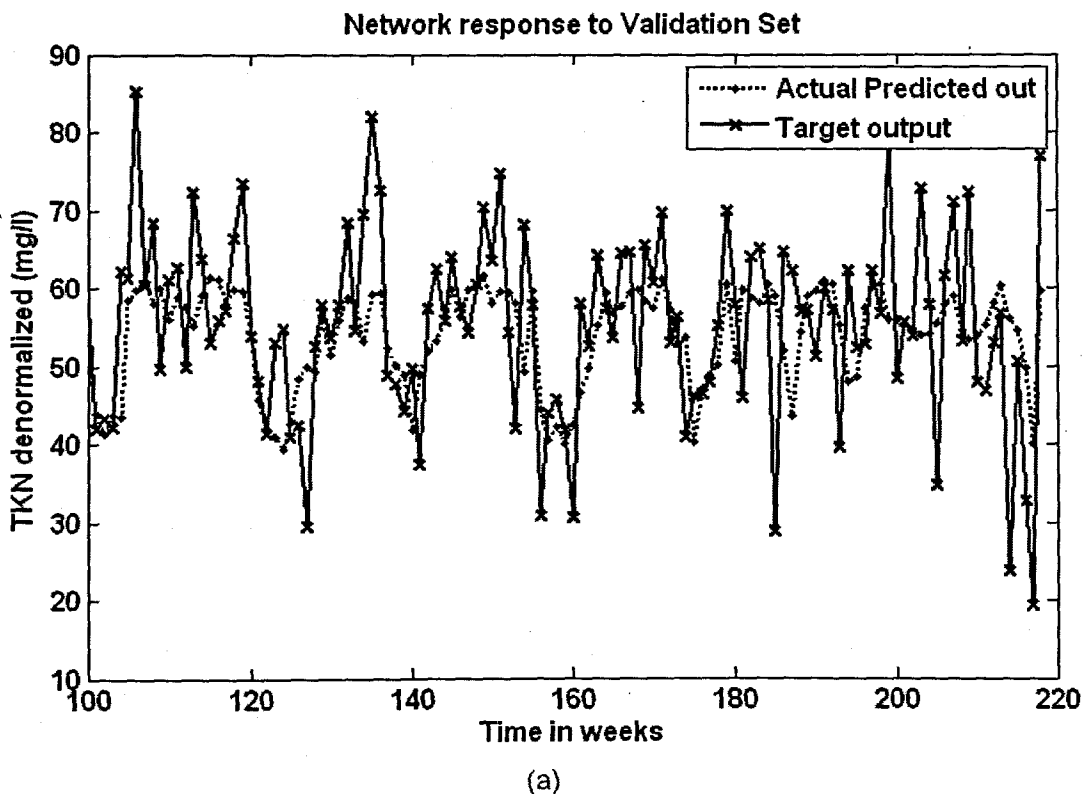
The inputs for Model 8 are TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), MONTH, Minimum Temperature, and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 81: Input variables for TKN Model 8**

Inputs	Output
TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), Month, Minimum Temperature	TKN(k+1)



**Figure 6. 51: (a) A MLP Feed-forward neural network Model 8 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 8 with 17 hidden neurons, 17 epochs**



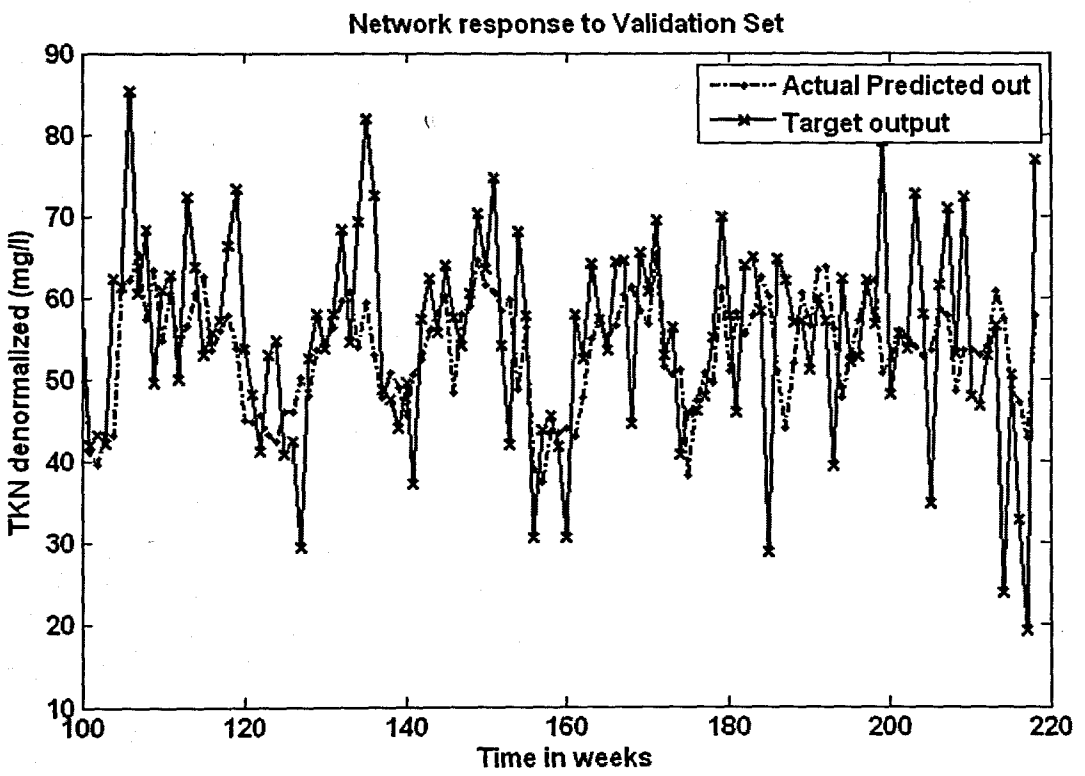
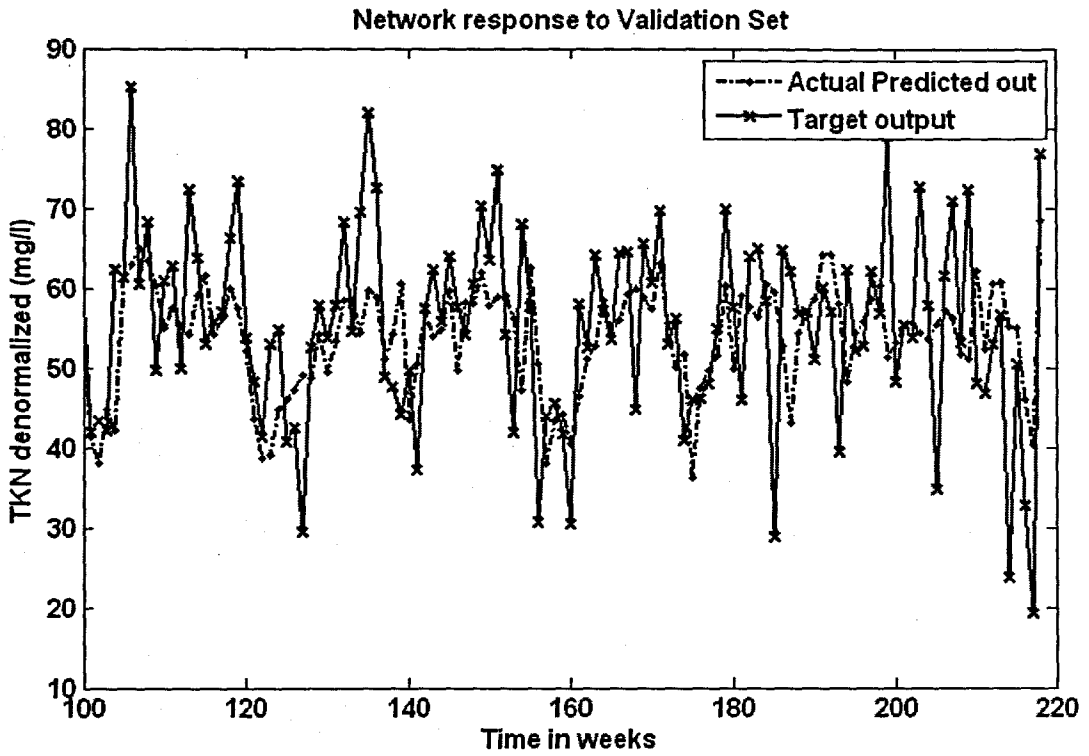


Figure 6. 52: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 8 with 1 hidden neuron and 500 epochs; (b) ERNN Model 8 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 8 with 17 hidden neurons, 17 epochs

**Table 6. 82: Results for the Feed-forward Neural Network Model 8 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	322	MSE	0.0104	0.6	0.5	0.609	0.563	0.371	0.317	-0.425	13.518
1	1	trainscg	tansig	purelin	500	MSE	0.0094	0.6	0.5	0.655	0.541	0.430	0.292	-0.497	13.937
1	10	trainscg	tansig	purelin	500	MSE	0.0089	0.6	0.5	0.679	0.564	0.461	0.318	-0.083	13.533
1	1	trainscg	tansig	purelin	287	MSE	0.0117	0.6	0.5	0.609	0.563	0.371	0.317	-0.425	13.518
1	5	trainscg	tansig	purelin	1000	MSE	0.0096	0.6	0.5	0.647	0.477	0.418	0.227	-0.553	14.807
1	10	trainscg	tansig	purelin	1000	MSE	0.0079	0.6	0.5	0.723	0.501	0.523	0.251	0.153	14.683

**Table 6. 83: Results for the Elman Recurrent Neural Network Model 8 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	322	MSE	0.0104	0.01	0.5	0.609	0.563	0.371	0.317	-0.425	13.518
1	1	trainscg	tansig	purelin	500	MSE	0.0094	0.01	0.5	0.655	0.541	0.430	0.292	-0.497	13.937
1	10	trainscg	tansig	purelin	500	MSE	0.0089	0.01	0.5	0.679	0.564	0.461	0.318	-0.083	13.533
1	1	trainscg	tansig	purelin	287	MSE	0.0104	0.01	0.5	0.609	0.563	0.371	0.317	-0.425	13.518
1	5	trainscg	tansig	purelin	1000	MSE	0.0096	0.01	0.5	0.647	0.477	0.418	0.227	-0.553	14.807
1	10	trainscg	tansig	purelin	1000	MSE	0.0079	0.01	0.5	0.723	0.501	0.523	0.251	0.153	14.683

**Table 6. 84: Results for the Radial Basis Function Neural Network Model 8 with TKN as output.**

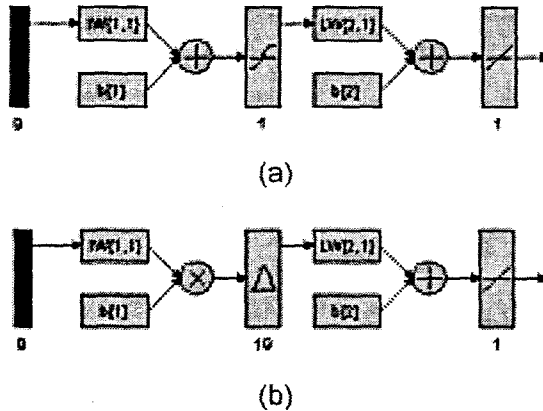
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	17	17	SSE	4.337	4.4	0.9	0.630	0.546	0.397	0.298	-0.664	13.832

### 6.3.9 TKN Neural Network MODEL 9

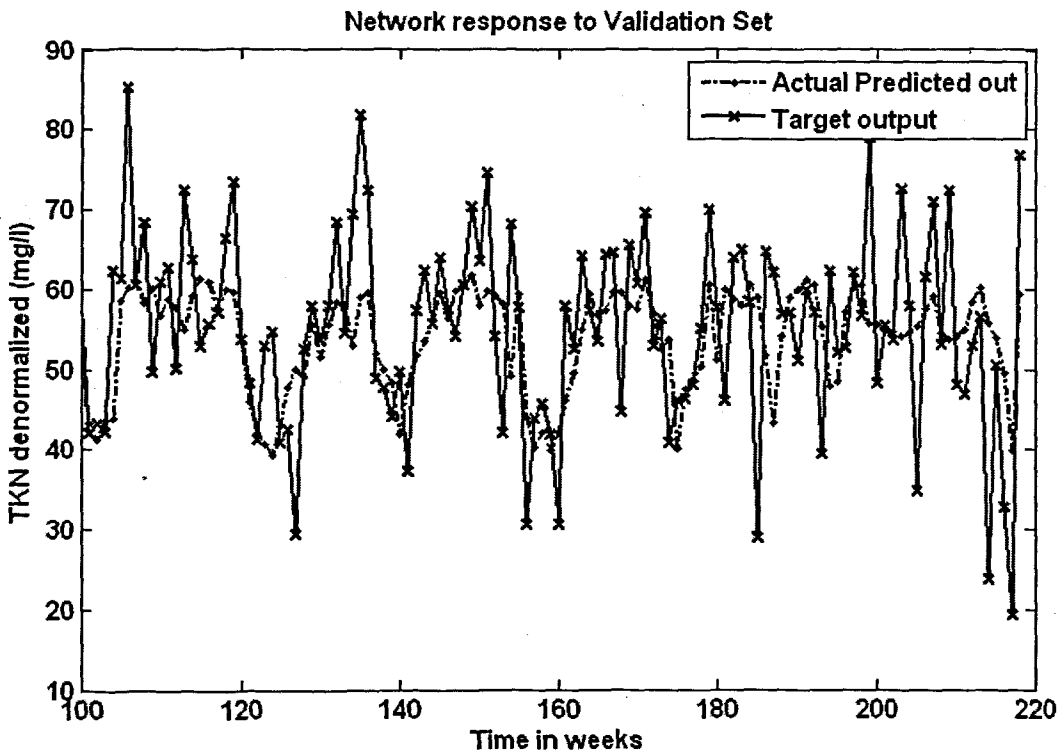
The inputs for Model 9 are  $TKN(k)$ ,  $TKN(k-1)$ ,  $TKN(k-2)$ ,  $TKN(k-3)$ ,  $COD(k)$ ,  $FLOW(k)$ ,  $MONTH$ ,  $Rain$ , and the output is  $TKN(k+1)$  one time step ahead. These are tabled below:

**Table 6. 85: Input variables for TKN Model 9**

Inputs	Output
$TKN(k)$ , $TKN(k-1)$ , $TKN(k-2)$ , $TKN(k-3)$ , $COD(k)$ , $FLOW(k)$ , $Month$ , $Rain$	$TKN(k+1)$



**Figure 6. 53: (a) A MLP Feed-forward neural network Model 9 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 9 with 19 hidden neurons, 19 epochs**



(a)

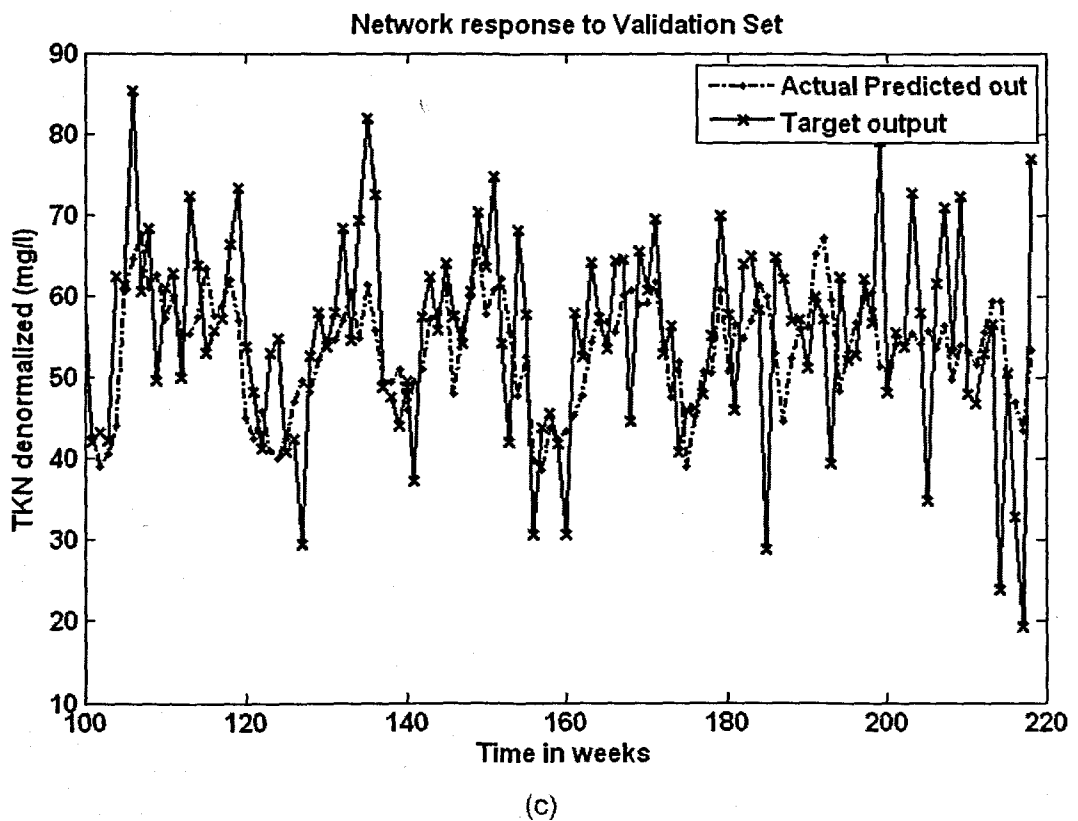
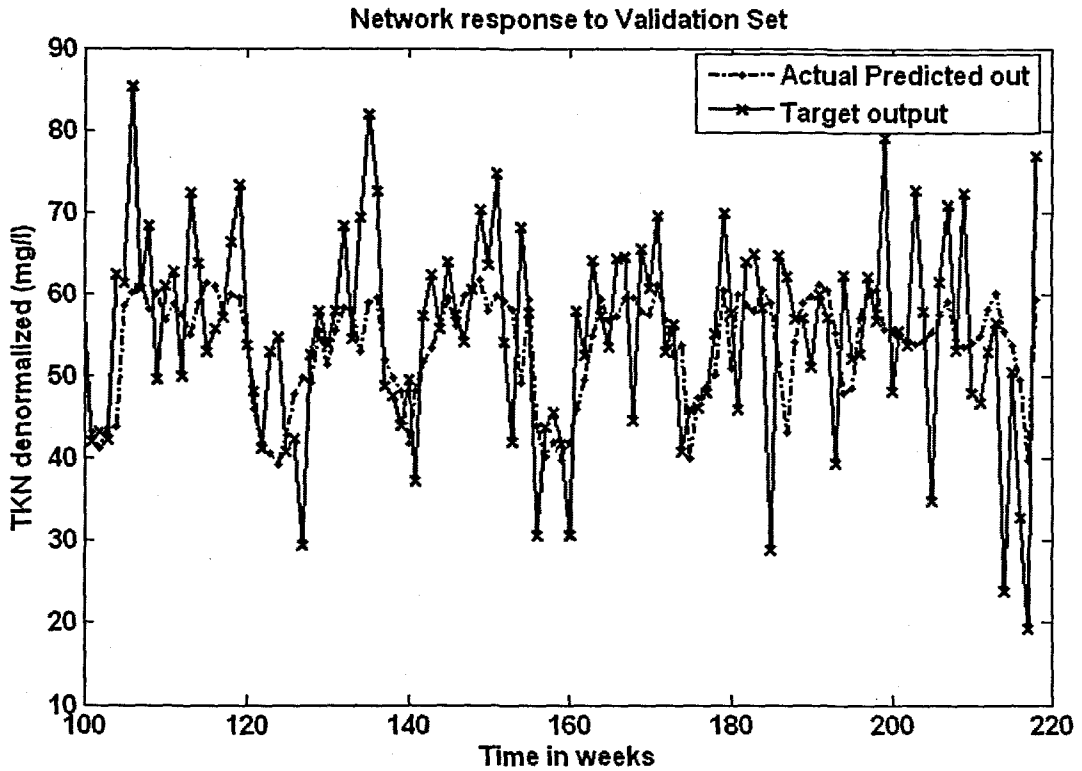


Figure 6. 54: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 9 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 9 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 9 with 19 hidden neurons, 19 epochs

**Table 6. 86: Results for the Feed-forward Neural Network Model 9 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	443	MSE	0.0104	0.6	0.5	0.609	0.567	0.371	0.322	-0.448	13.457
1	5	trainscg	tansig	purelin	500	MSE	0.0093	0.6	0.5	0.659	0.557	0.434	0.311	-0.441	13.721
1	10	trainscg	tansig	purelin	500	MSE	0.0090	0.6	0.5	0.676	0.523	0.457	0.273	-0.621	13.970
1	1	trainscg	tansig	purelin	376	MSE	0.0104	0.6	0.5	0.609	0.567	0.371	0.322	-0.448	13.456
1	5	trainscg	tansig	purelin	1000	MSE	0.0087	0.6	0.5	0.688	0.318	0.473	0.101	0.129	15.216
1	10	trainscg	tansig	purelin	1000	MSE	0.0077	0.6	0.5	0.730	0.487	0.533	0.237	-0.619	14.824

**Table 6. 87: Results for the Elman Recurrent Neural Network Model 9 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	294	MSE	0.0104	0.01	0.5	0.609	0.567	0.371	0.322	-0.448	13.458
1	5	trainscg	tansig	purelin	500	MSE	0.0098	0.01	0.5	0.639	0.547	0.408	0.299	-0.221	13.655
1	10	trainscg	tansig	purelin	500	MSE	0.0090	0.01	0.5	0.675	0.563	0.456	0.317	-0.662	13.771
1	1	trainscg	tansig	purelin	231	MSE	0.0104	0.01	0.5	0.609	0.567	0.371	0.322	-0.448	13.457
1	5	trainscg	tansig	purelin	1000	MSE	0.0092	0.01	0.5	0.665	0.548	0.442	0.300	-0.406	14.042
1	10	trainscg	tansig	purelin	1000	MSE	0.0075	0.01	0.5	0.738	0.483	0.544	0.233	-0.464	14.977

**Table 6. 88: Results for the Radial Basis Function Neural Network Model 9 with TKN as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	19	19	SSE	4,280	4.3	0.9	0.636	0.547	0.405	0.299	-0.724	13.935

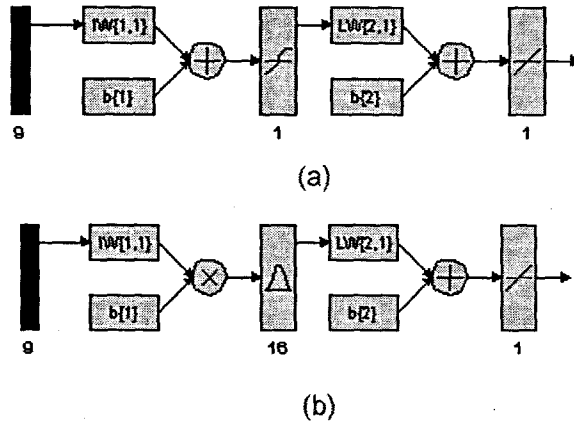


**6.3.10 TKN Neural Network MODEL 10**

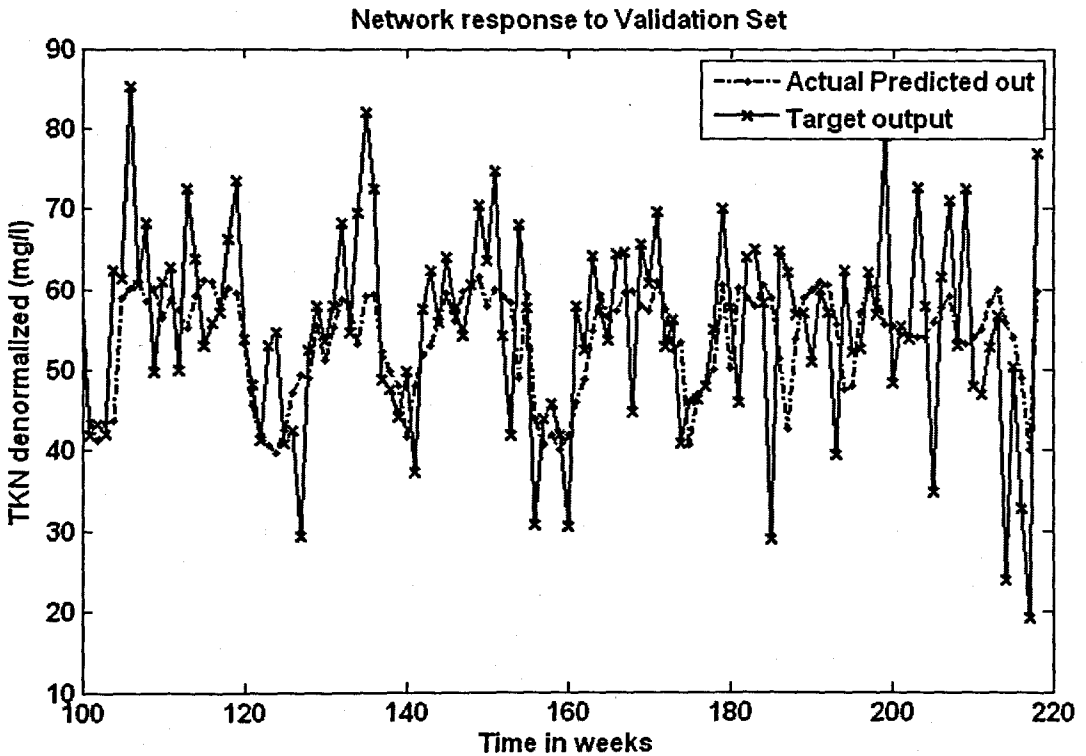
The inputs for Model 10 are TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), MONTH, Wind, and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 89: Input variables for TKN Model 10**

Inputs	Output
TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), Month, Wind	TKN(k+1)



**Figure 6. 55: (a) A MLP Feed-forward neural network Model 10 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 10 with 16 hidden neurons, 16 epochs**



(a)

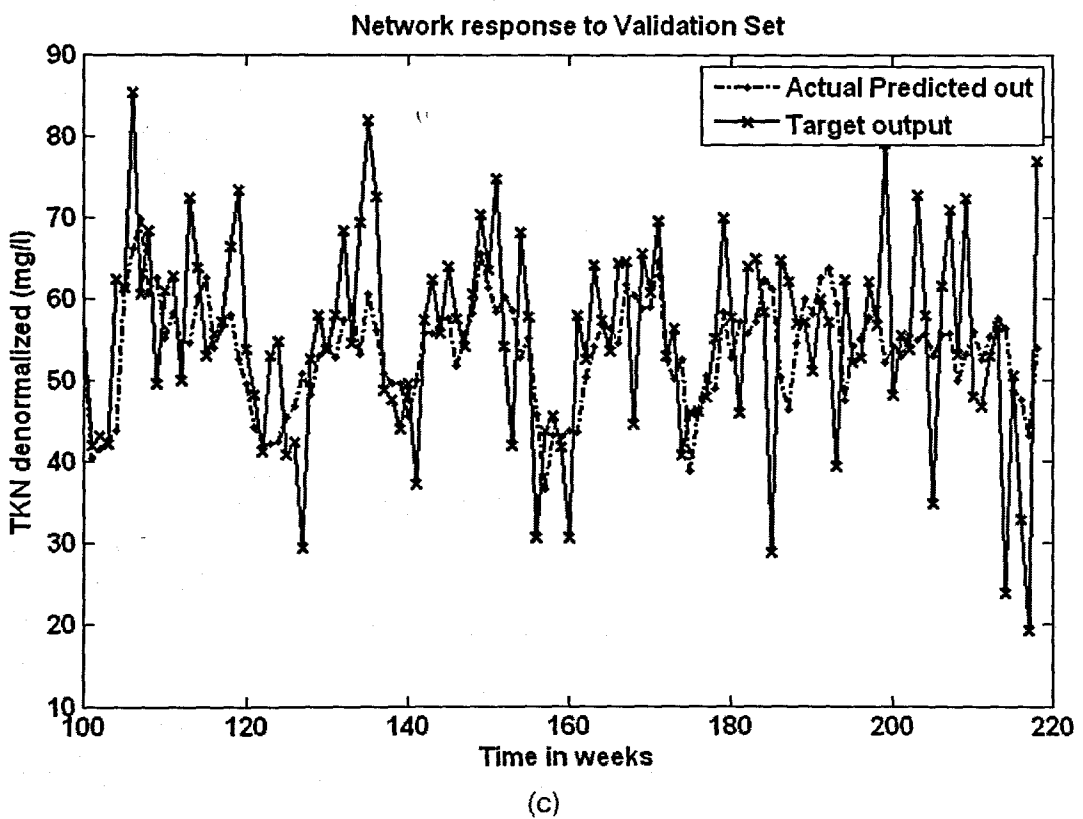
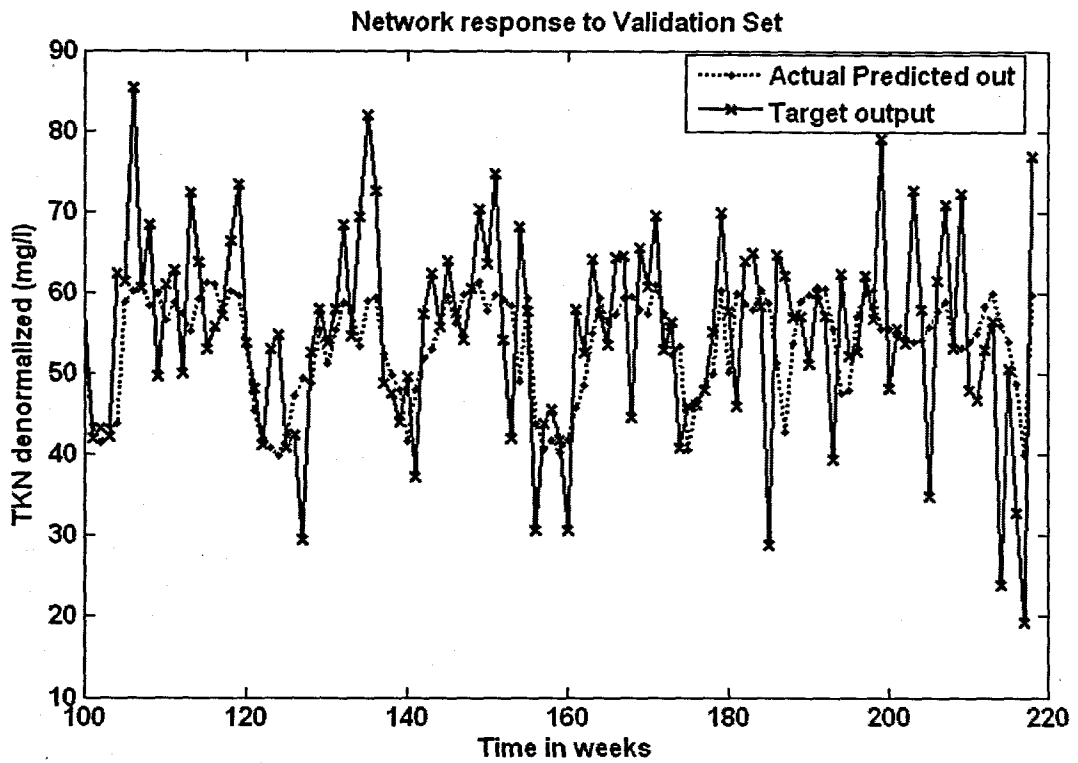


Figure 6. 56: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 10 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 10 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 10 with 16 hidden neurons, 16 epochs

**Table 6. 90: Results for the Feed-forward Neural Network Model 10 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	408	MSE	0.0104	0.6	0.5	0.610	0.562	0.372	0.316	-0.431	13.527
1	5	trainscg	tansig	purelin	500	MSE	0.0095	0.6	0.5	0.652	0.527	0.425	0.278	-0.257	13.819
1	10	trainscg	tansig	purelin	500	MSE	0.0089	0.6	0.5	0.679	0.514	0.461	0.264	-0.708	14.818
1	1	trainscg	tansig	purelin	328	MSE	0.0104	0.6	0.5	0.610	0.562	0.372	0.316	-0.432	13.527
1	5	trainscg	tansig	purelin	1000	MSE	0.0087	0.6	0.5	0.687	0.505	0.472	0.256	-0.879	14.670
1	10	trainscg	tansig	purelin	1000	MSE	0.0072	0.6	0.5	0.750	0.492	0.563	0.242	-0.980	14.992

**Table 6. 91: Results for the Elman Recurrent Neural Network Model 10 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	368	MSE	0.0104	0.01	0.5	0.610	0.562	0.372	0.316	-0.431	13.528
1	5	trainscg	tansig	purelin	500	MSE	0.0092	0.01	0.5	0.664	0.549	0.441	0.301	-0.642	13.762
1	10	trainscg	tansig	purelin	500	MSE	0.0086	0.01	0.5	0.694	0.509	0.481	0.259	-1.155	14.748
1	1	trainscg	tansig	purelin	324	MSE	0.0104	0.01	0.5	0.610	0.562	0.372	0.316	-0.431	13.528
1	5	trainscg	tansig	purelin	1000	MSE	0.0089	0.01	0.5	0.681	0.513	0.464	0.263	-1.001	14.674
1	10	trainscg	tansig	purelin	1000	MSE	0.0070	0.01	0.5	0.759	0.486	0.576	0.236	-0.926	15.402

**Table 6. 92: Results for the Radial Basis Function Neural Network Model 10 with TKN as output.**

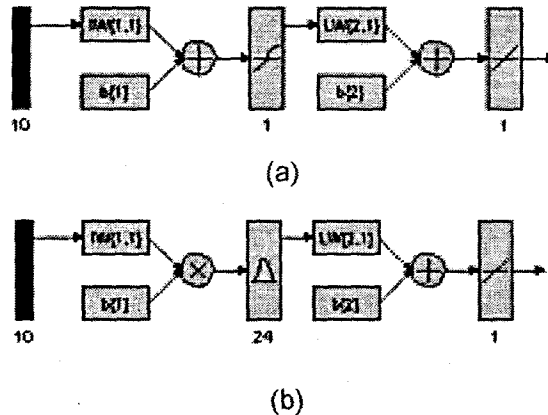
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	16	16	SSE	4.295	4.3	0.9	0.635	0.553	0.403	0.306	-0.686	13.662

### 6.3.11 TKN Neural Network MODEL 11

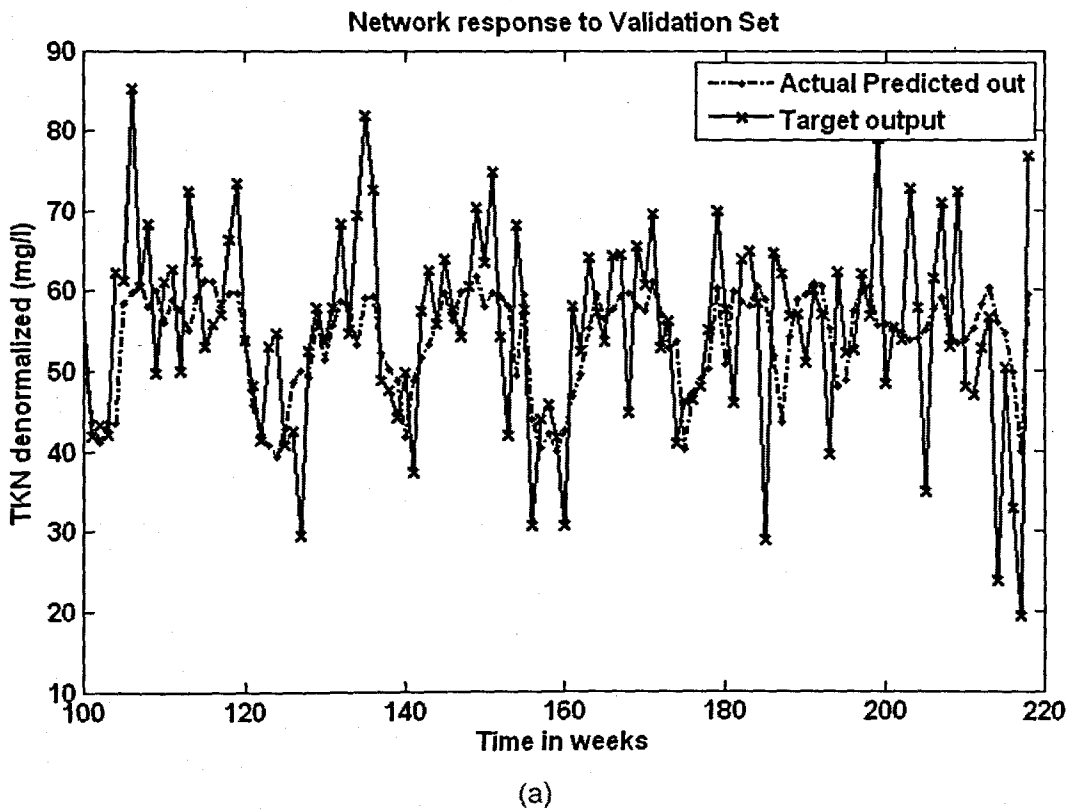
The inputs for Model 11 are TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), MONTH, Minimum Temperature, Rain, and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 93: Input variables for TKN Model 11**

Inputs	Output
TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), Month, Minimum Temperature, Rain	TKN(k+1)



**Figure 6. 57: (a) A MLP Feed-forward neural network Model 11 with 10 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 11 with 24 hidden neurons, 24 epochs**



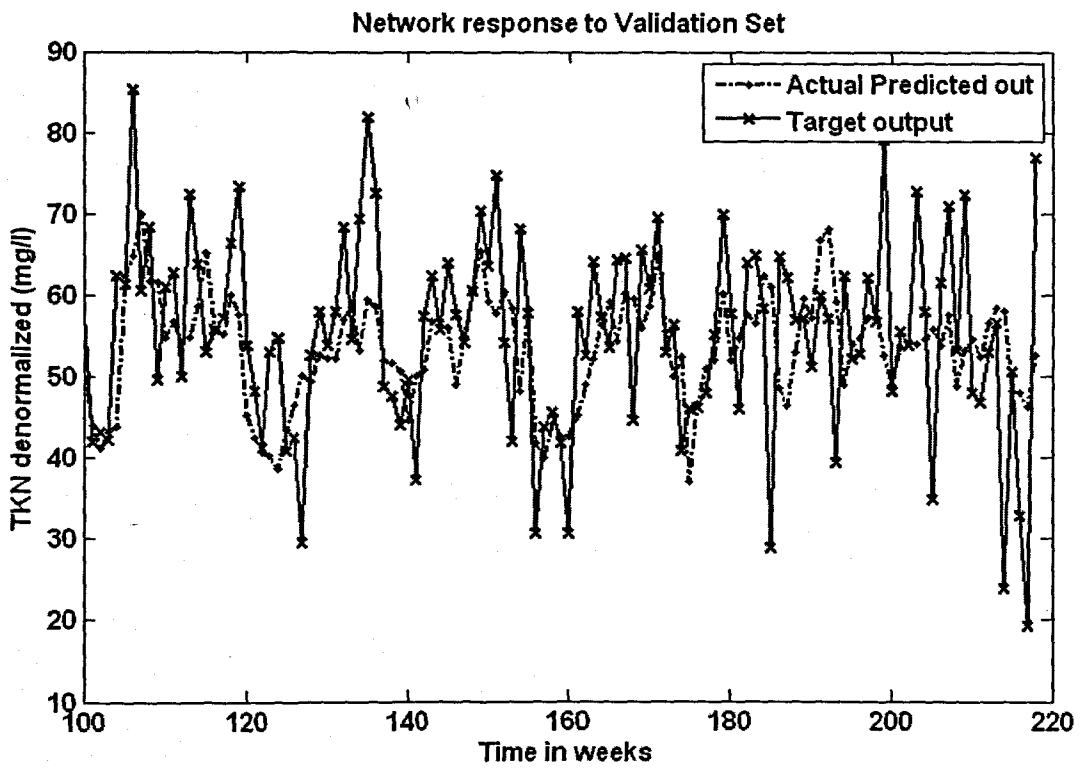
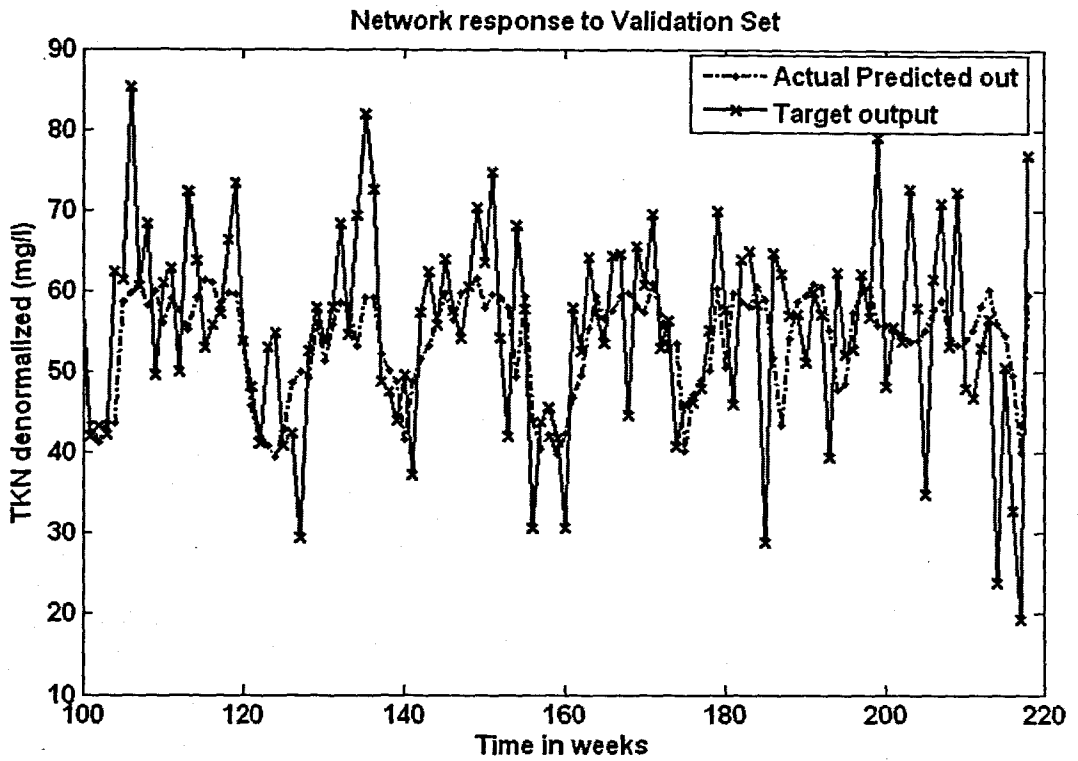


Figure 6. 58: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 11 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 11 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 11 with 24 hidden neurons, 24 epochs

**Table 6. 94: Results for the Feed-forward Neural Network Model 11 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	479	MSE	0.0104	0.6	0.5	0.609	0.564	0.371	0.318	-0.426	13.503
1	5	trainscg	tansig	purelin	500	MSE	0.0095	0.6	0.5	0.649	0.556	0.422	0.309	-0.942	13.744
1	10	trainscg	tansig	purelin	500	MSE	0.0085	0.6	0.5	0.695	0.502	0.483	0.252	-0.780	14.717
1	1	trainscg	tansig	purelin	368	MSE	0.0104	0.6	0.5	0.609	0.564	0.371	0.318	-0.426	13.503
1	5	trainscg	tansig	purelin	1000	MSE	0.0091	0.6	0.5	0.669	0.536	0.448	0.288	-0.669	13.906
1	10	trainscg	tansig	purelin	1000	MSE	0.0080	0.6	0.5	0.716	0.485	0.512	0.235	-0.205	14.829

**Table 6. 95: Results for the Feed-forward Neural Network Model 11 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	479	MSE	0.0104	0.6	0.5	0.609	0.564	0.371	0.318	-0.426	13.503
1	5	trainscg	tansig	purelin	500	MSE	0.0095	0.6	0.5	0.649	0.556	0.422	0.309	-0.942	13.744
1	10	trainscg	tansig	purelin	500	MSE	0.0085	0.6	0.5	0.695	0.502	0.483	0.252	-0.780	14.717
1	1	trainscg	tansig	purelin	368	MSE	0.0104	0.6	0.5	0.609	0.564	0.371	0.318	-0.426	13.503
1	5	trainscg	tansig	purelin	1000	MSE	0.0091	0.6	0.5	0.669	0.536	0.448	0.288	-0.669	13.906
1	10	trainscg	tansig	purelin	1000	MSE	0.0080	0.6	0.5	0.716	0.485	0.512	0.235	-0.205	14.829

**Table 6. 96: Results for the Radial Basis Function Neural Network Model 11 with TKN as output.**

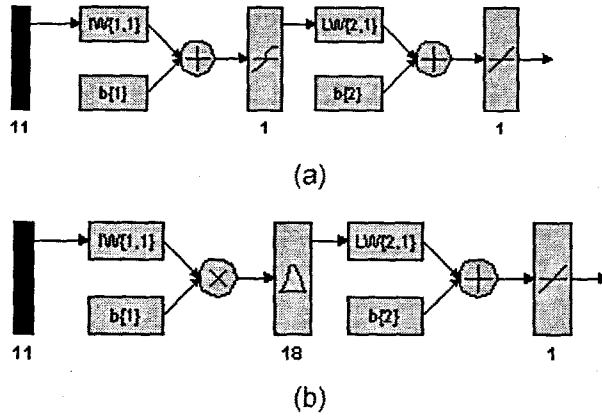
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	24	24	SSE	4.252	4.3	0.9	0.640	0.537	0.409	0.289	-0.733	14.012

### 6.3.12 TKN Neural Network MODEL 12

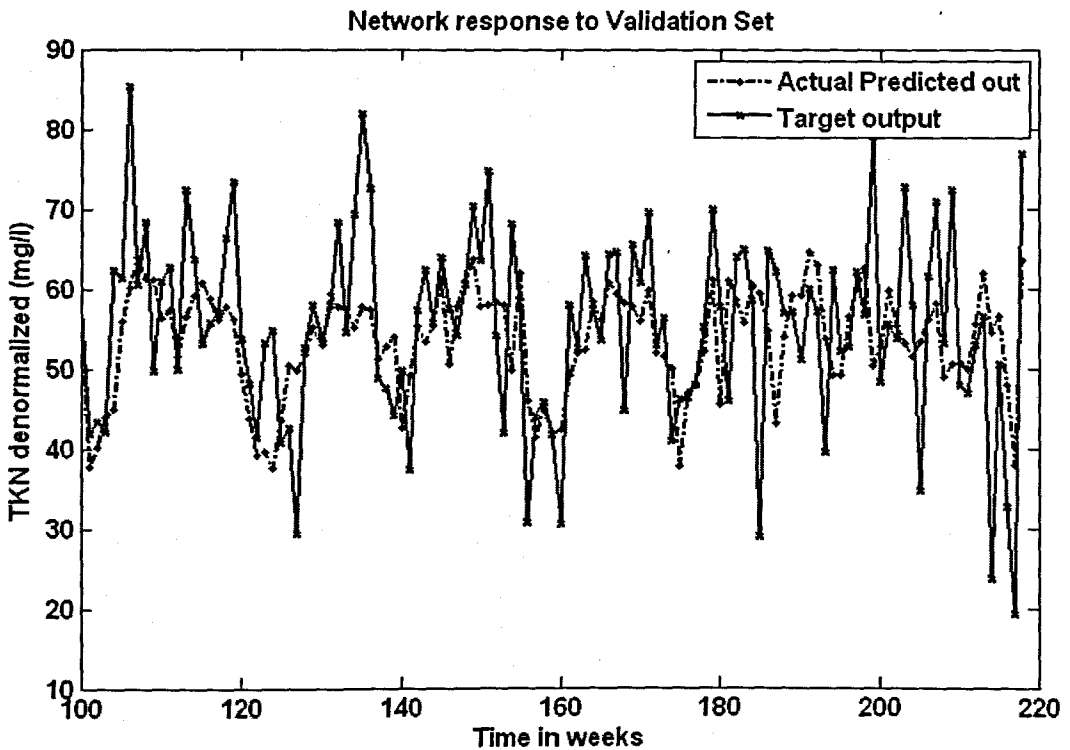
The inputs for Model 12 are TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), MONTH, Minimum Temperature, Rain, Wind, and the output is TKN(k+1) one time step ahead. These are tabled below:

**Table 6. 97: Input variables for TKN Model 12**

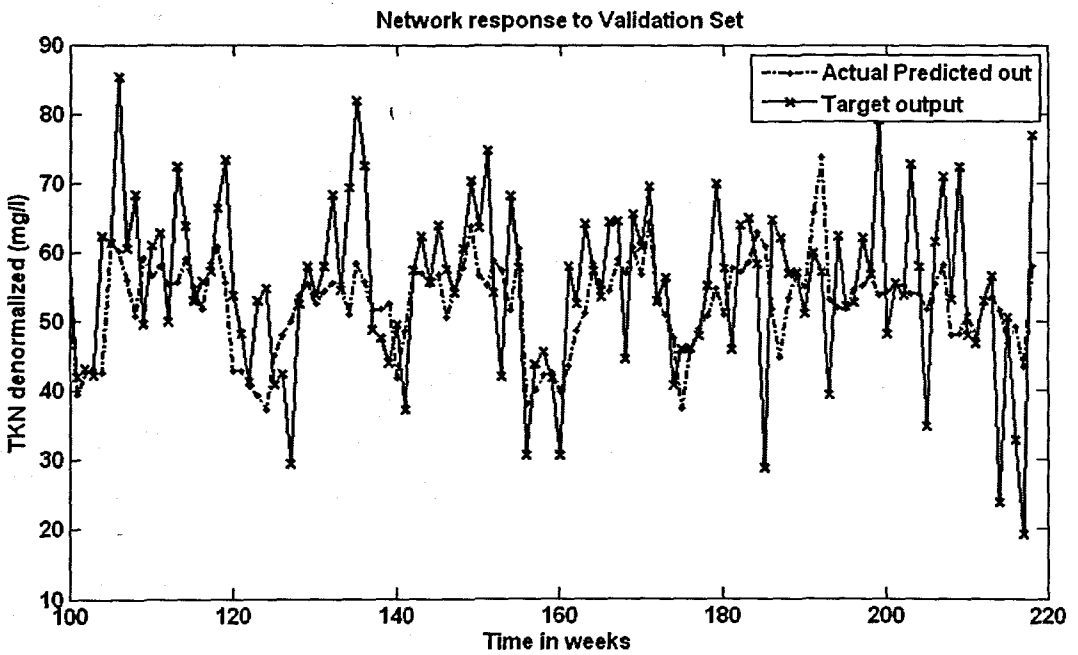
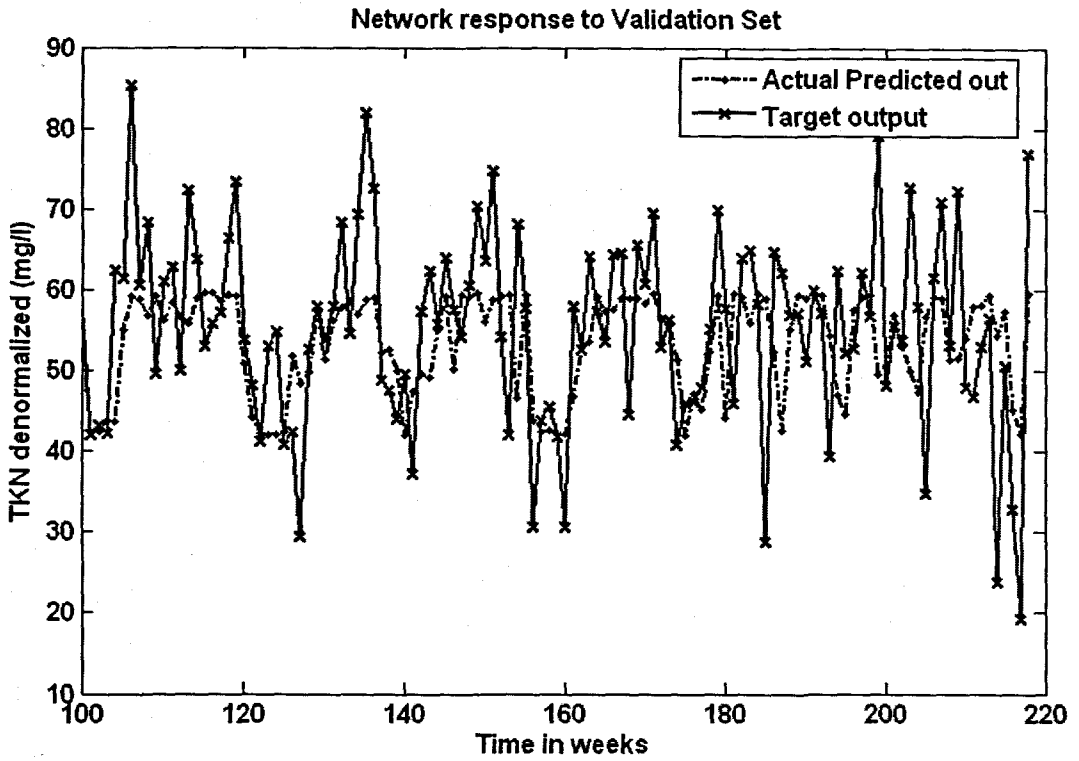
Inputs	Output
TKN(k), TKN(k-1), TKN(k-2), TKN(k-3), COD(k), FLOW(k), Month, Minimum Temperature, Rain, Wind	TKN(k+1)



**Figure 6. 59: (a) A MLP Feed-forward neural network Model 12 with 11 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 12 with 18 hidden neurons, 18 epochs**



(a)



**Figure 6. 60:** The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 12 with 5 hidden neurons and 500 epochs; (b) ERNN Model 12 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 12 with 18 hidden neurons, 18 epochs



**Table 6. 98: Results for the Feed-forward Neural Network Model 12 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r train	r test	R sq train	R sq test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0106	0.6	0.5	0.597	0.511	0.356	0.261	-0.595	14.160
1	5	trainscg	tansig	purelin	500	MSE	0.0101	0.6	0.5	0.624	0.523	0.389	0.274	-0.651	14.155
1	10	trainscg	tansig	purelin	500	MSE	0.0087	0.6	0.5	0.686	0.463	0.471	0.215	-1.011	14.893
1	1	trainscg	tansig	purelin	782	MSE	0.0106	0.6	0.5	0.597	0.504	0.356	0.254	-0.578	14.194
1	5	trainscg	tansig	purelin	1000	MSE	0.0087	0.6	0.5	0.687	0.475	0.472	0.226	-1.035	14.944
1	10	trainscg	tansig	purelin	1000	MSE	0.0075	0.6	0.5	0.738	0.485	0.545	0.235	-1.093	15.373

**Table 6. 99: Results for the Elman Recurrent Neural Network Model 12 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r train	r test	R sq train	R sq test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0106	0.01	0.5	0.597	0.514	0.356	0.264	-0.599	14.148
1	5	trainscg	tansig	purelin	500	MSE	0.0096	0.01	0.5	0.646	0.495	0.418	0.245	-0.878	14.325
1	10	trainscg	tansig	purelin	500	MSE	0.0085	0.01	0.5	0.699	0.477	0.488	0.228	-0.678	15.090
1	1	trainscg	tansig	purelin	904	MSE	0.0106	0.01	0.5	0.597	0.504	0.356	0.254	-0.578	14.195
1	5	trainscg	tansig	purelin	1000	MSE	0.0087	0.01	0.5	0.686	0.465	0.470	0.216	-1.092	15.058
1	10	trainscg	tansig	purelin	1000	MSE	0.0086	0.01	0.5	0.690	0.502	0.477	0.252	-0.703	14.445

**Table 6. 100: Results for the Radial Basis Function Neural Network Model 12 with TKN as output.**

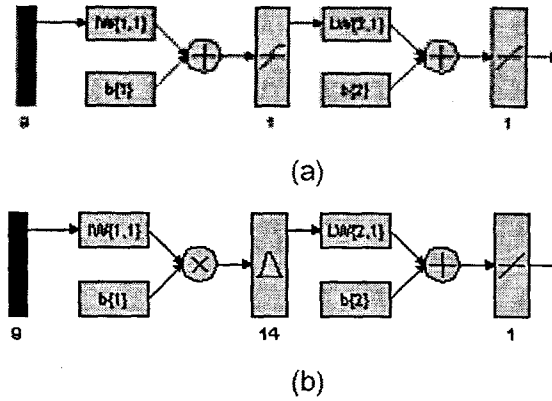
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r train	r test	R sq train	R sq test	Avg_test error	APE
1	radbas	purelin	18	18	SSE	4.465	4.5	1	0.616	0.529	0.379	0.280	-0.953	14.021

### 6.3.13 TKN Neural Network MODEL 13

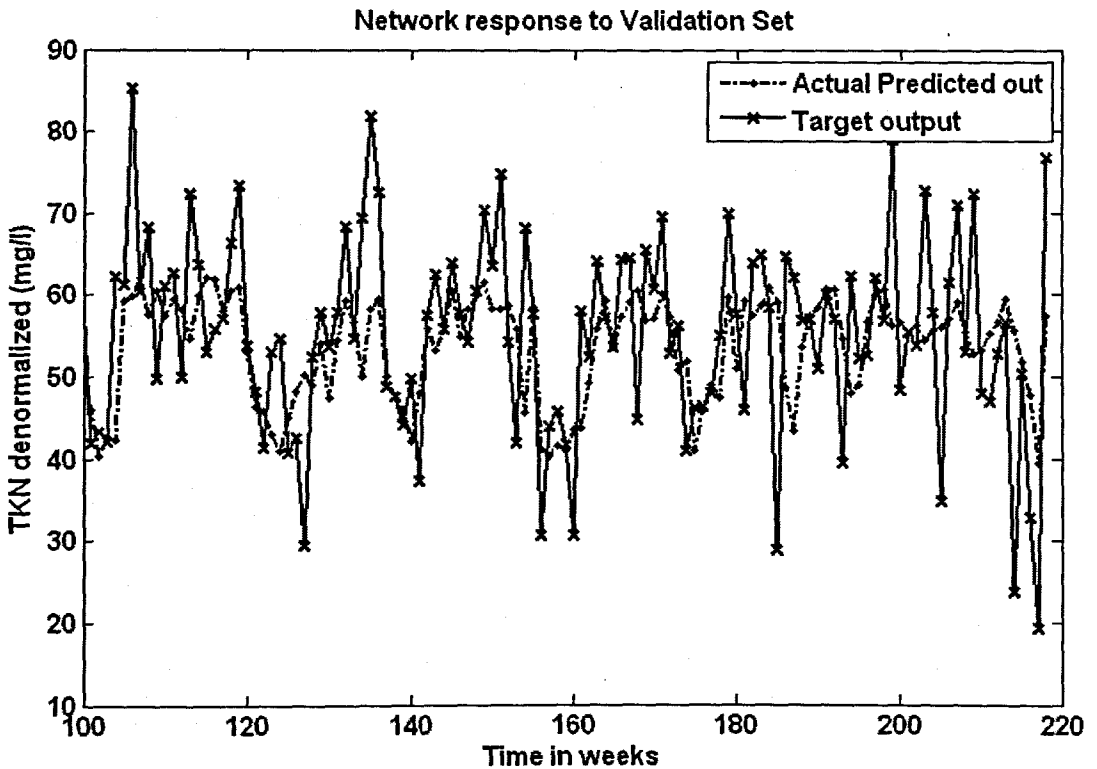
The inputs for Model 13 are  $TKN(k)$ ,  $TKN(k-1)$ ,  $TKN(k-2)$ ,  $TKN(k-3)$ , MONTH, Minimum Temperature, Rain, Wind, and the output is  $TKN(k+1)$  one time step ahead. These are tabled below:

**Table 6. 101: Input variables for TKN Model 13**

Inputs	Output
$TKN(k)$ , $TKN(k-1)$ , $TKN(k-2)$ , $TKN(k-3)$ , Month, Minimum Temperature, Rain, Wind	$TKN(k+1)$



**Figure 6. 61: (a) A MLP Feed-forward neural network Model 13 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the TKN RBFNN Model 13 with 14 hidden neurons, 14 epochs**



(a)

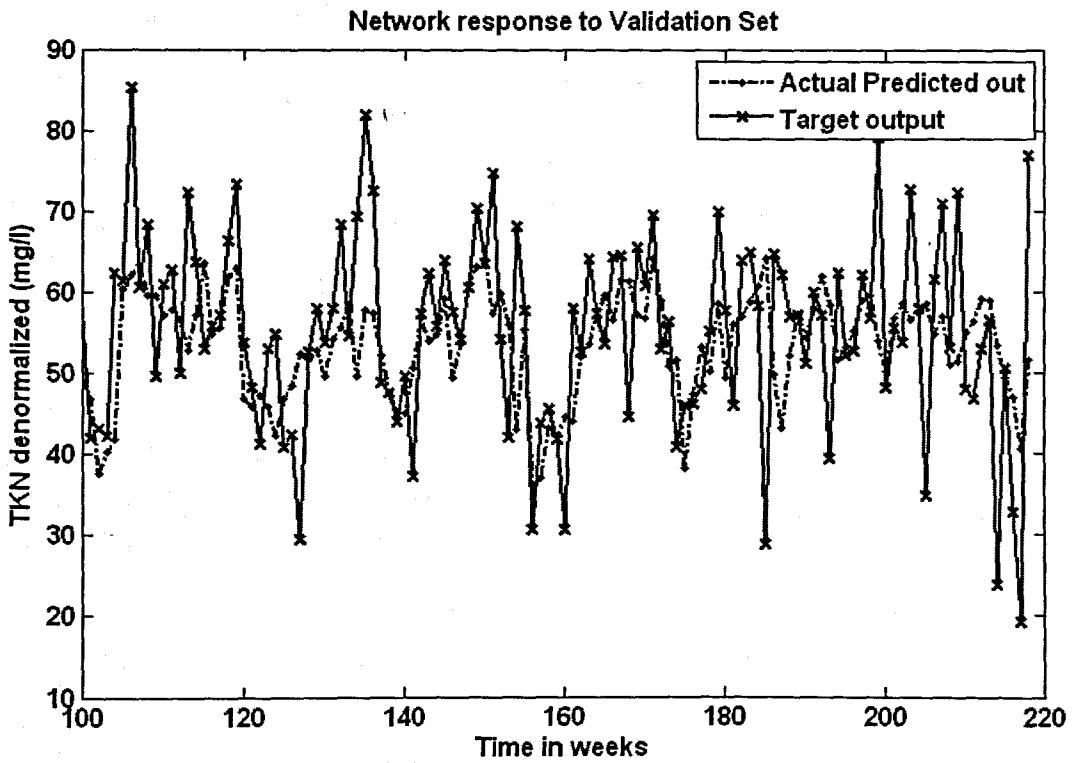
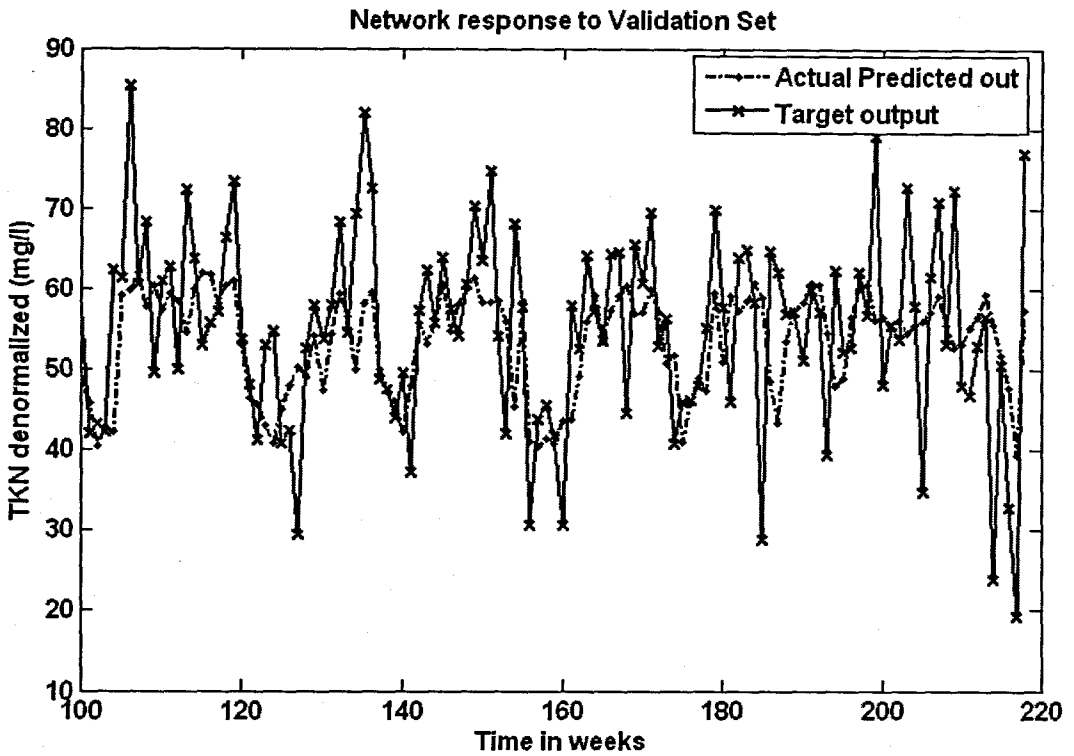


Figure 6. 62: The NN response to the application of the test (validation) data set for TKN: (a) MLP FFNN Model 13 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 13 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 13 with 14 hidden neurons, 14 epochs

**Table 6. 102: Results for the Feed-forward Neural Network Model 13 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0106	0.6	0.5	0.596	0.539	0.356	0.290	-0.445	13.684
1	5	trainscg	tansig	purelin	500	MSE	0.0097	0.6	0.5	0.643	0.544	0.414	0.296	-0.819	13.889
1	10	trainscg	tansig	purelin	500	MSE	0.0088	0.6	0.5	0.683	0.526	0.467	0.277	-0.565	14.244
1	1	trainscg	tansig	purelin	408	MSE	0.0106	0.6	0.5	0.597	0.548	0.357	0.300	-0.472	13.617
1	5	trainscg	tansig	purelin	1000	MSE	0.0092	0.6	0.5	0.667	0.500	0.445	0.250	-0.756	14.480
1	10	trainscg	tansig	purelin	1000	MSE	0.0087	0.6	0.5	0.686	0.519	0.471	0.270	-0.823	14.375

**Table 6. 103: Results for the Elman Recurrent Neural Network Model 13 with TKN as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	384	MSE	0.0106	0.01	0.5	0.597	0.547	0.357	0.300	-0.471	13.617
1	5	trainscg	tansig	purelin	500	MSE	0.0092	0.01	0.5	0.667	0.462	0.444	0.214	-0.850	14.882
1	10	trainscg	tansig	purelin	500	MSE	0.0085	0.01	0.5	0.698	0.508	0.487	0.258	-1.141	14.979
1	1	trainscg	tansig	purelin	337	MSE	0.0106	0.01	0.5	0.597	0.547	0.357	0.300	-0.471	13.617
1	5	trainscg	tansig	purelin	1000	MSE	0.0096	0.01	0.5	0.648	0.498	0.420	0.248	-0.939	14.557
1	10	trainscg	tansig	purelin	1000	MSE	0.0077	0.01	0.5	0.732	0.423	0.536	0.179	-0.523	15.470

**Table 6. 104: Results for the Radial Basis Function Neural Network Model 13 with TKN as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	14	14	SSE	4.389	4.4	1	0.624	0.535	0.390	0.286	-0.866	13.978

## 6.4 Flow Neural Network Results

The final set of NN results presented in this section is for the influent variable input FLOW rate as the predicted output of the 3 different NN architectures. The NN model numbers as per the inputs are specified in tables at the beginning of the results for the corresponding model. All the models have only one hidden layer.

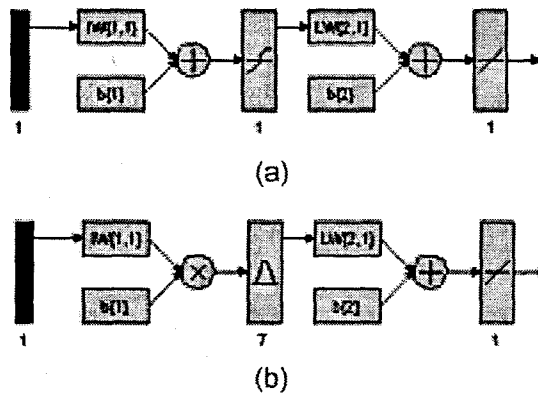
### 6.4.1 FLOW Neural Network MODEL 1

The input for model 1 is FLOW(k), and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6.105: Input variables for FLOW Model 1**

Inputs	Output
FLOW(k)	FLOW(k+1)

The physical structure for the MLP FFNN and the RBF NN are represented in the following figure. Due to limitations on the MATLAB program for the visualisation of the physical NN, the Elman Recurrent NN is not displayed here.



**Figure 6.63: (a) MLP Feed-forward NN Model 1 with 1 input, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 1 with 1 input, 6 hidden neurons**

For the NN with FLOW as an output in this section 6.4, all tables and abbreviations remain the same as per the section in 6.2 for the NN with COD as output. The inputs are specified as per Table 5.3. For space considerations only for Model 1 all figures are included, the rest of the models only have the final NN response to the application of the test data set. However, please note that all results can be viewed using the program on the enclosed CD.

**Table 6.106: Results for the Feed-forward Neural Network Model 1 with FLOW as output.**

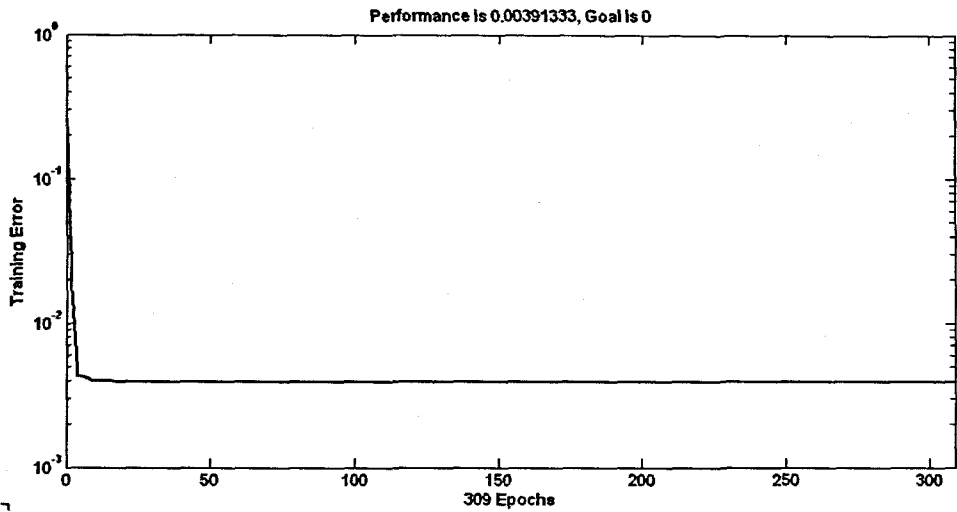
Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	309	MSE	0.0039	0.6	0.5	0.855	0.868	0.731	0.753	-0.424	8.695
1	5	trainscg	tansig	purelin	500	MSE	0.0038	0.6	0.5	0.858	0.859	0.736	0.737	-0.701	9.031
1	10	trainscg	tansig	purelin	500	MSE	0.0037	0.6	0.5	0.863	0.854	0.744	0.729	-0.453	9.190
1	1	trainscg	tansig	purelin	477	MSE	0.0039	0.6	0.5	0.855	0.868	0.731	0.753	-0.424	8.696
1	5	trainscg	tansig	purelin	1000	MSE	0.0038	0.6	0.5	0.858	0.856	0.737	0.733	-0.702	9.057
1	10	trainscg	tansig	purelin	1000	MSE	0.0037	0.6	0.5	0.863	0.856	0.745	0.732	-0.452	9.127

**Table 6.107: Results for the Elman Recurrent Neural Network Model 1 with FLOW as output.**

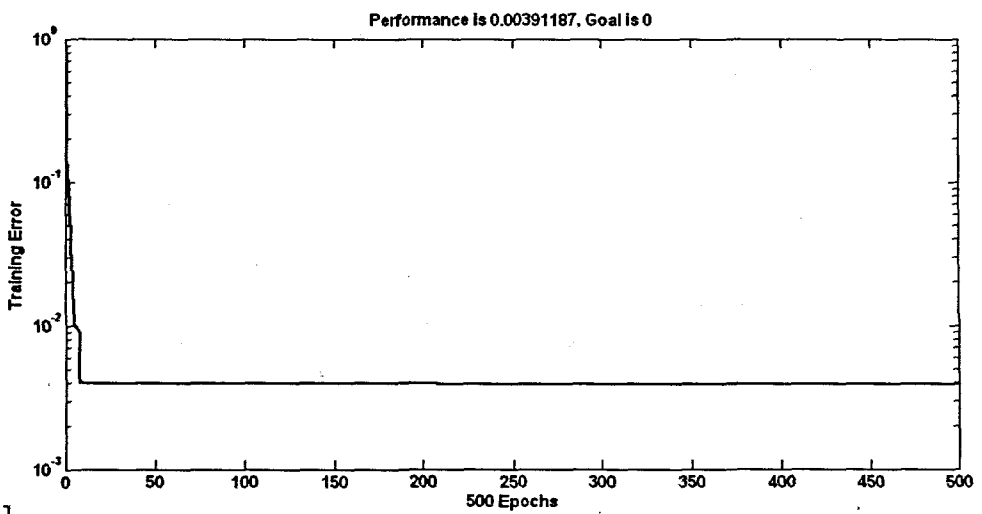
Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	119	MSE	0.0039	0.01	0.5	0.855	0.868	0.731	0.753	-0.421	8.690
1	5	trainscg	tansig	purelin	500	MSE	0.0039	0.01	0.5	0.855	0.867	0.731	0.752	-0.440	8.708
1	10	trainscg	tansig	purelin	500	MSE	0.0039	0.01	0.5	0.856	0.866	0.732	0.749	-0.457	8.782
1	1	trainscg	tansig	purelin	279	MSE	0.0039	0.01	0.5	0.855	0.855	0.868	0.731	0.753	-0.423
1	5	trainscg	tansig	purelin	1000	MSE	0.0039	0.01	0.5	0.856	0.864	0.732	0.746	-0.467	8.833
1	10	trainscg	tansig	purelin	1000	MSE	0.0039	0.01	0.5	0.856	0.864	0.732	0.746	-0.472	8.823

**Table 6.108: Results for the Radial Basis Function Neural Network Model 1 with FLOW as output.**

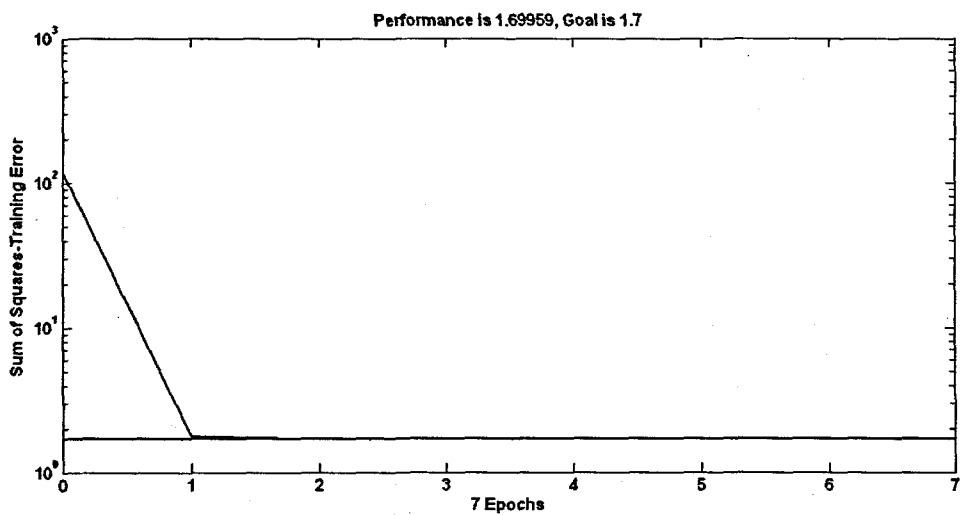
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	7	7	SSE	1.700	1.7	0.5	0.856	0.863	0.732	0.745	-0.461	8.878



(a)

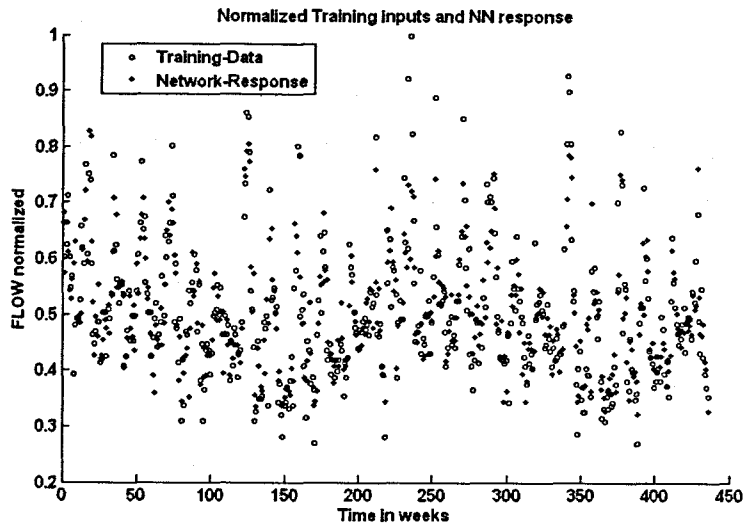


(b)

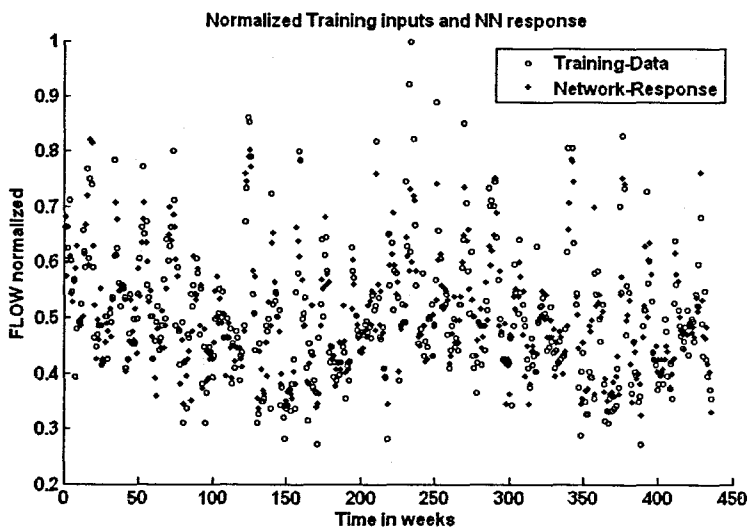


(c)

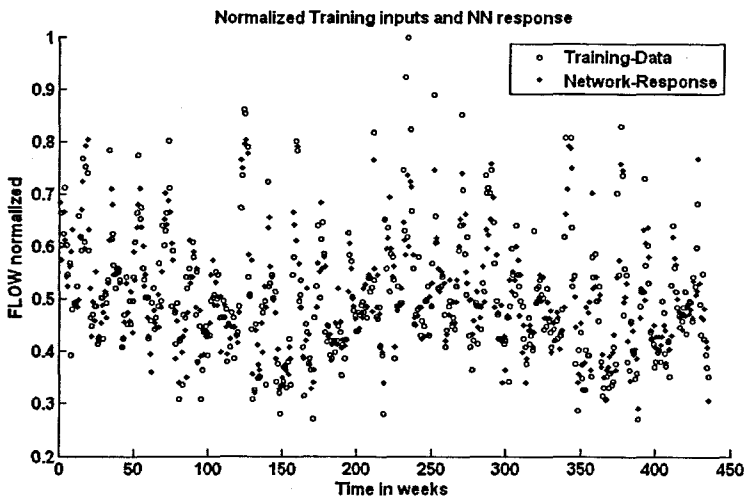
**Figure 6. 64:** Training error versus the number of epochs for FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron, 309 epochs; (b) ERNN Model 1 with 5 hidden neuron, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs



(a)



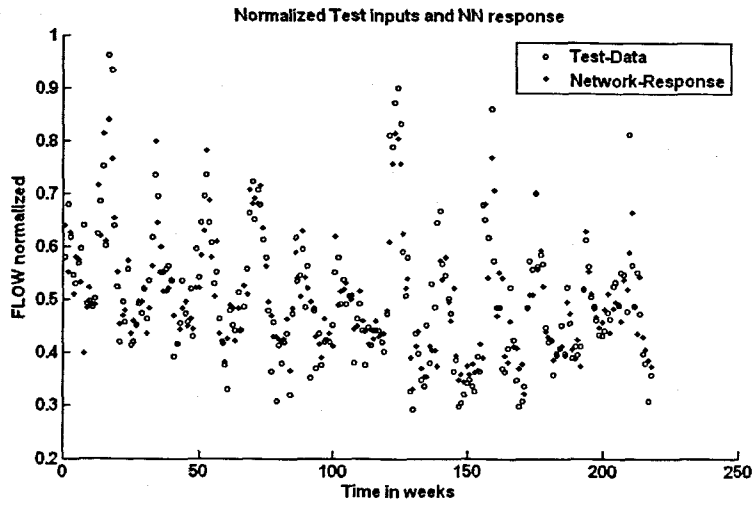
(b)



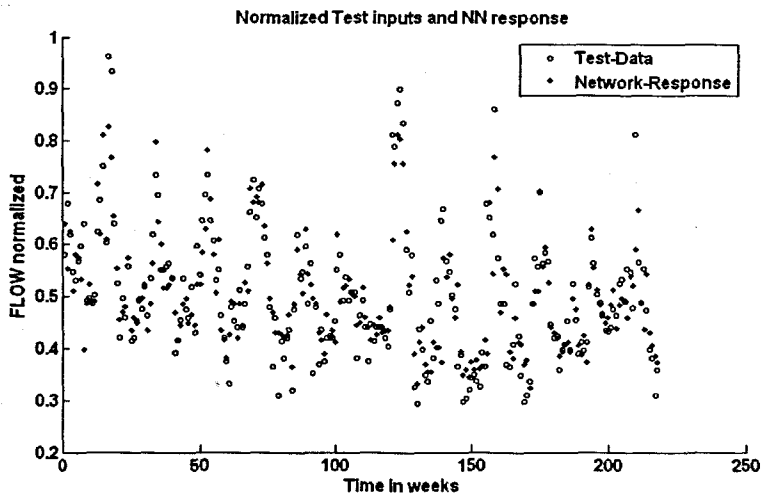
(c)

**Figure 6. 65:** A scatter-plot of the training input data and the NN response with the application of the training set for the FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron, 309 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs

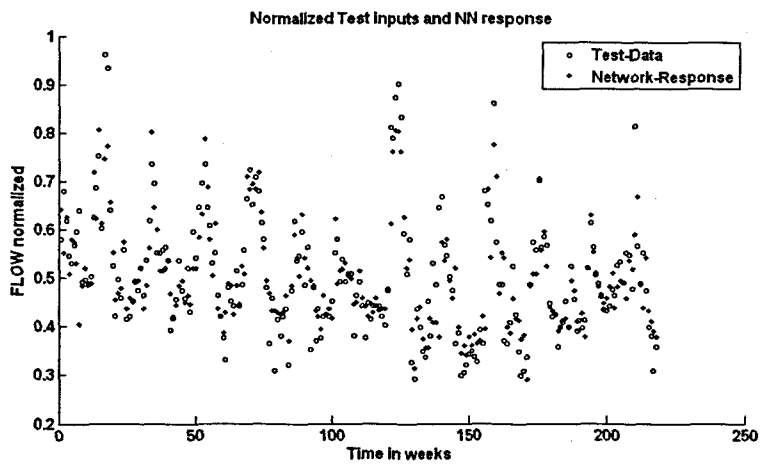




(a)



(b)



(c)

Figure 6. 66: A scatter-plot of the test (validation) data and the NN response with the application of the test set for the FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron, 309 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs

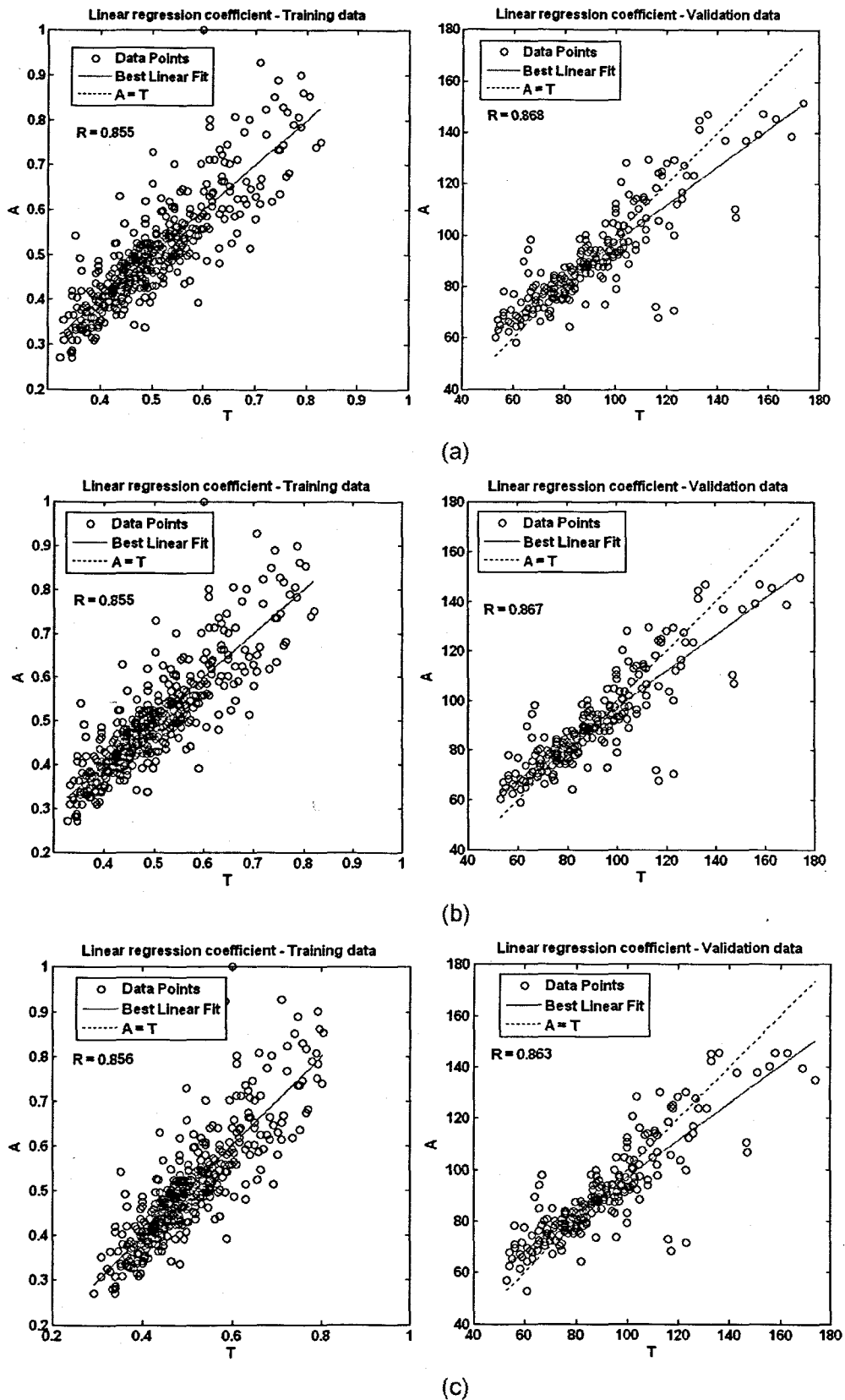
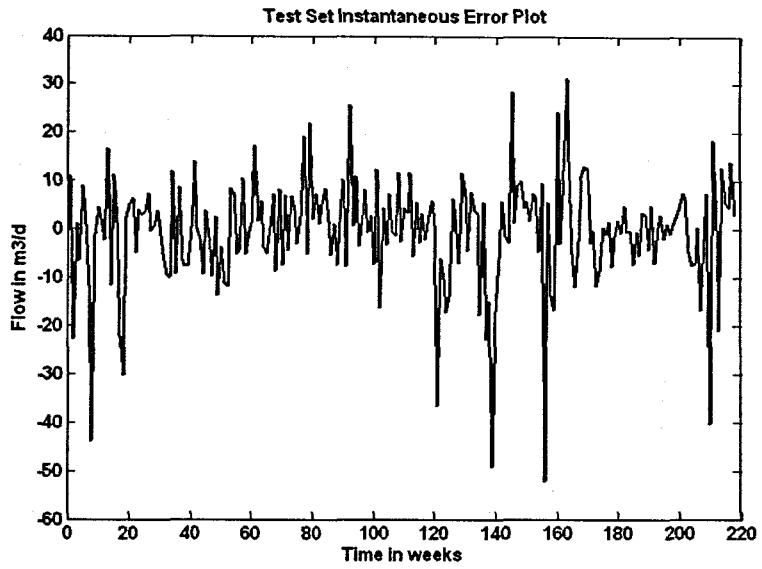
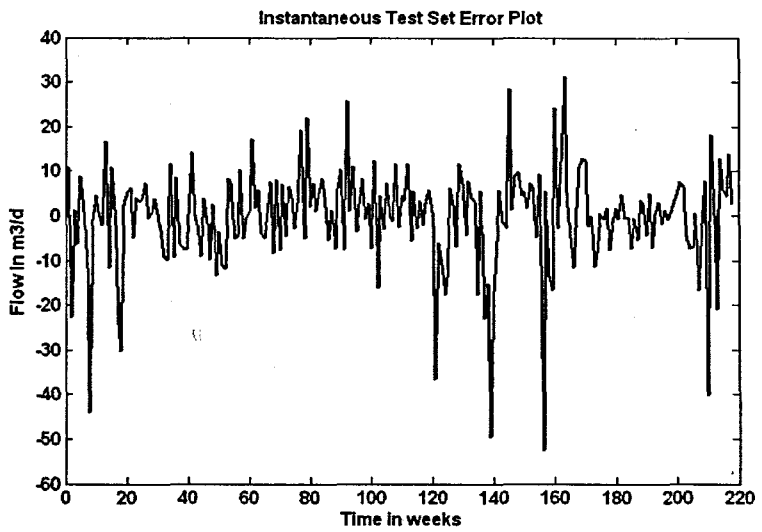


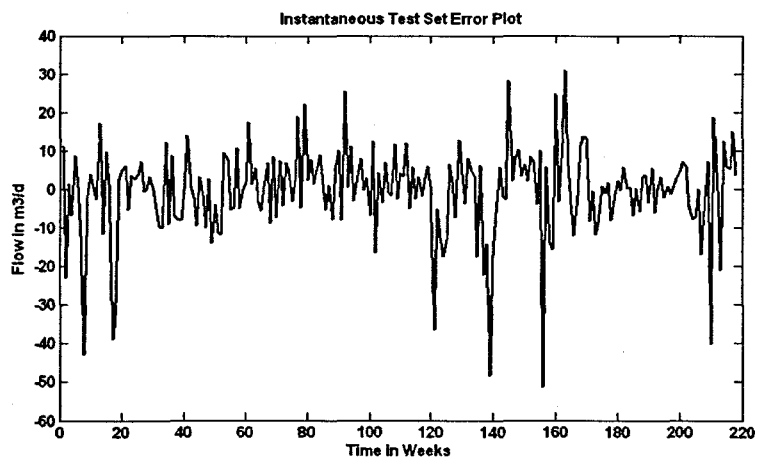
Figure 6. 67: Network performance showing the linear regression (correlation) coefficients for training data (normalized) on the left, and the validation data set (denormalized) on the right for the FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron and 309 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs, where A and T are the NN output and desired target output respectively



(a)



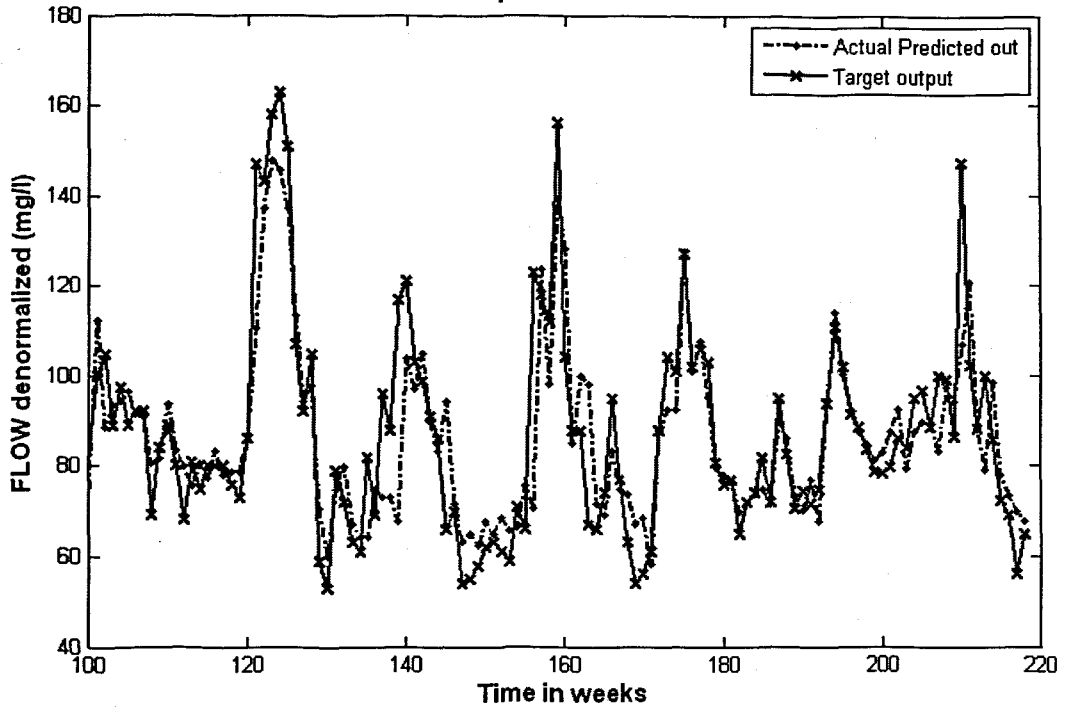
(b)



(c)

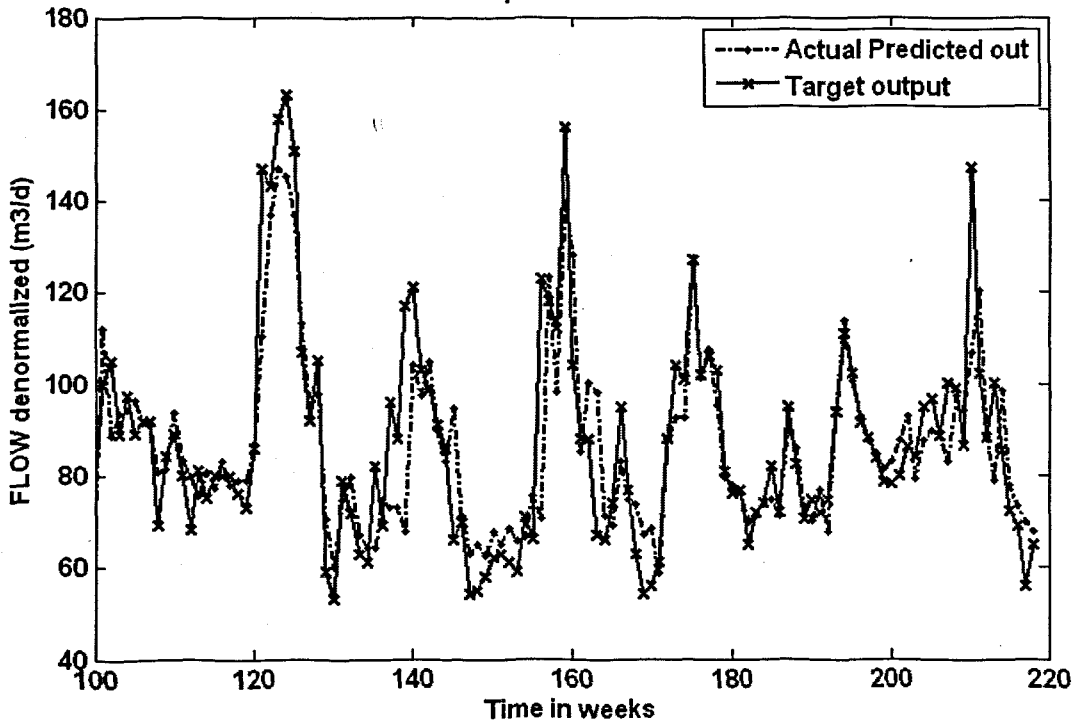
**Figure 6. 68:** The instantaneous error across the entire validation data set for the FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron and 500 epochs; (b) ERNN Model 1 with 5 hidden neurons, 309 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs

Network response to Validation Set

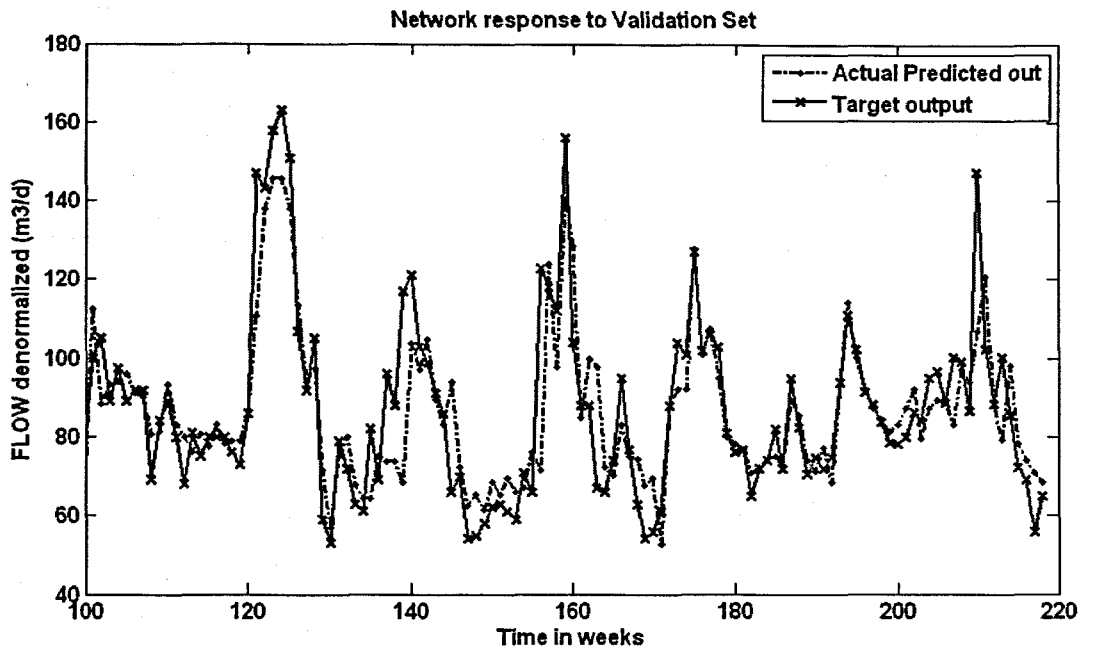


(a)

Network response to Validation Set



(b)



(c)

**Figure 6. 69:** The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 1 with 1 hidden neuron and 309 epochs; (b) ERNN Model 1 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 1 with 7 hidden neurons, 7 epochs

### 6.4.2 FLOW Neural Network MODEL 2

The inputs for Model 2 are  $FLOW(k)$ ,  $FLOW(k-1)$ , and the output is  $FLOW(k+1)$  one time step ahead. These are tabled below:

Table 6. 109: Input variables for FLOW Model 2

Inputs	Output
$FLOW(k)$ , $FLOW(k-1)$	$FLOW(k+1)$

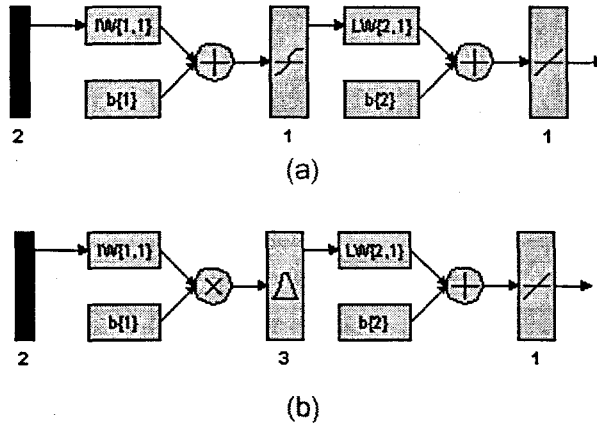
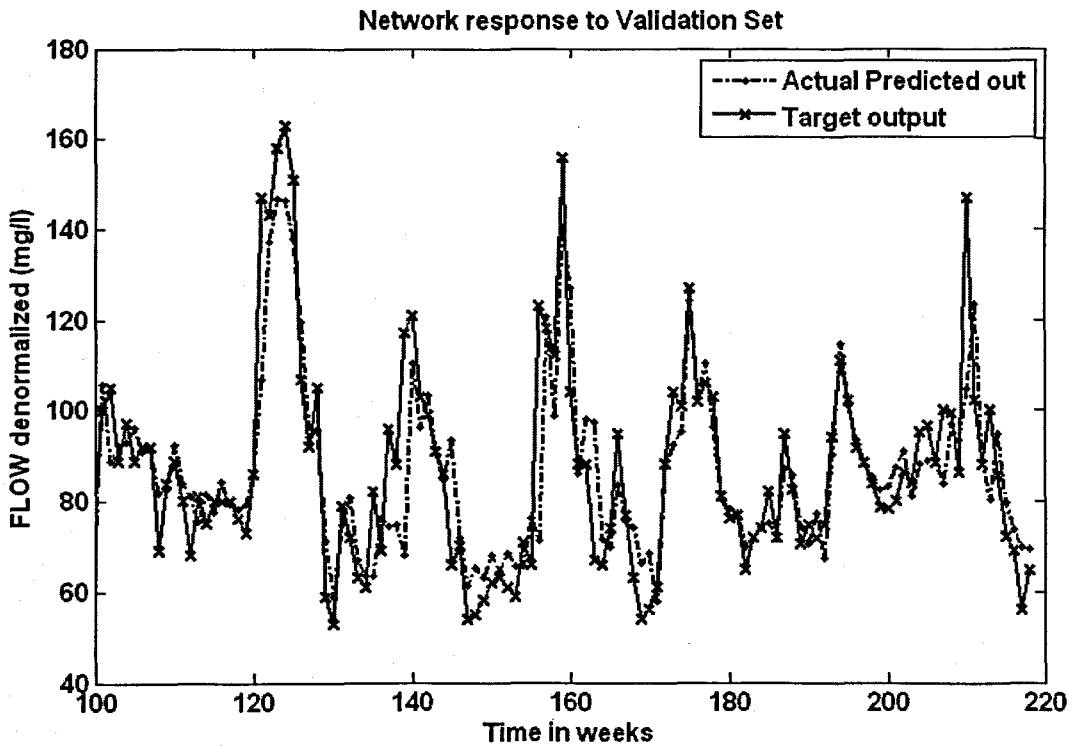
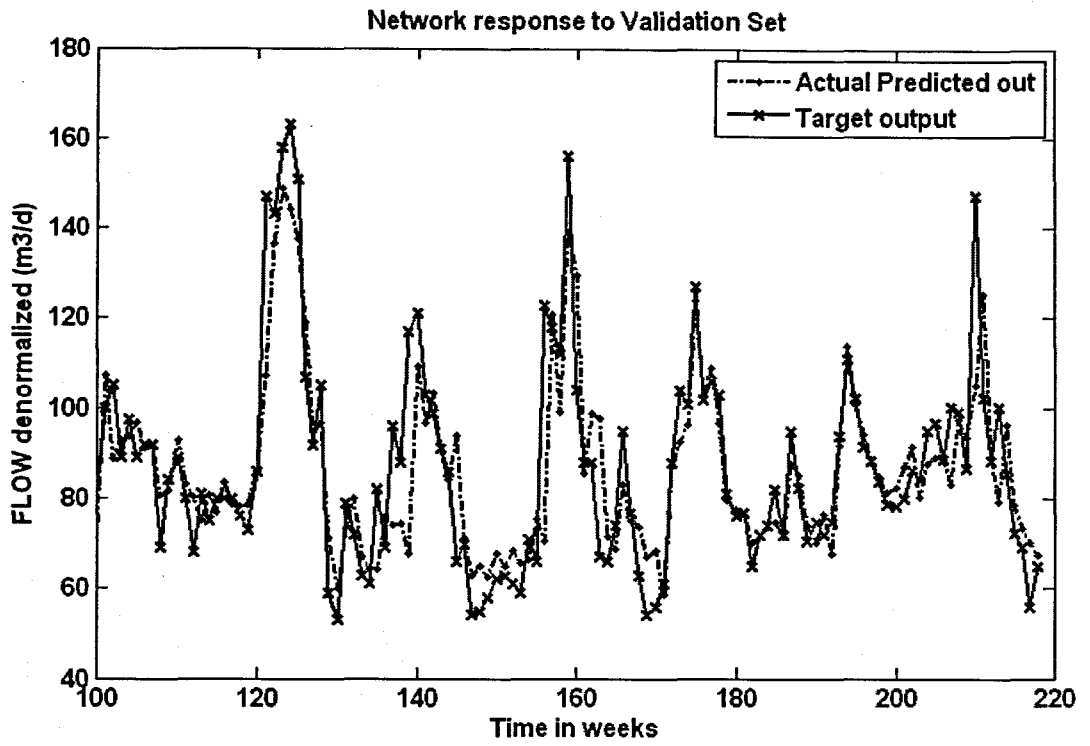


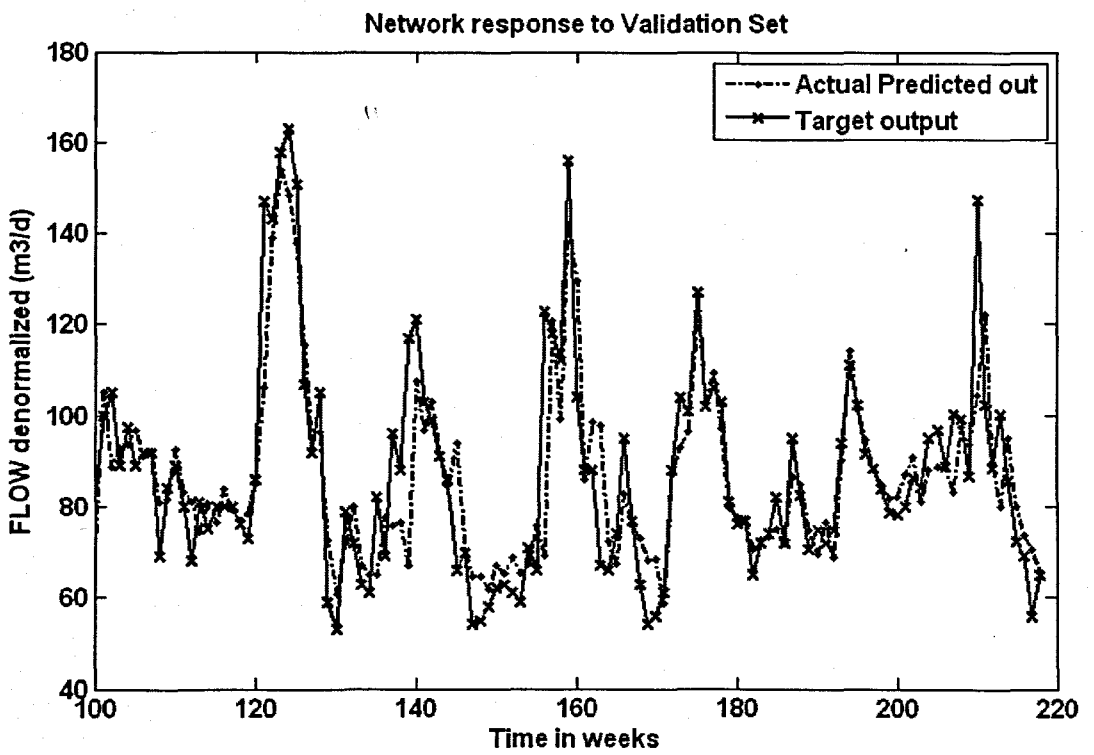
Figure 6. 70: (a) A MLP Feed-forward neural network Model 2 with 2 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; (b) Block Diagram for the FLOW RBFNN Model 2 with 3 hidden neurons



(a)



(b)



(c)

Figure 6. 71: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 2 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 2 with 10 hidden neurons, 1000 epochs; and (c) RBFNN Model 2 with 3 hidden neurons, 3 epochs

**Table 6. 110: Results for the Feed-forward Neural Network Model 2 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	269	MSE	0.0038	0.6	0.5	0.858	0.864	0.736	0.746	-0.442	8.774
1	5	trainscg	tansig	purelin	500	MSE	0.0039	0.6	0.5	0.858	0.859	0.735	0.737	-0.630	8.960
1	10	trainscg	tansig	purelin	500	MSE	0.0038	0.6	0.5	0.861	0.856	0.7416	0.733	-0.5211	8.980
1	1	trainscg	tansig	purelin	403	MSE	0.0040	0.6	0.5	0.858	0.864	0.736	0.746	-0.443	8.775
1	5	trainscg	tansig	purelin	1000	MSE	0.0038	0.6	0.5	0.860	0.866	0.740	0.750	-0.322	8.754
1	10	trainscg	tansig	purelin	1000	MSE	0.0037	0.6	0.5	0.863	0.859	0.744	0.738	-0.684	9.064

**Table 6. 111: Results for the Elman Recurrent Neural Network Model 2 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	206	MSE	0.0038	0.01	0.5	0.858	0.864	0.736	0.746	-0.443	8.775
1	5	trainscg	tansig	purelin	500	MSE	0.0038	0.01	0.5	0.858	0.865	0.737	0.749	-0.324	8.707
1	10	trainscg	tansig	purelin	500	MSE	0.0038	0.01	0.5	0.858	0.864	0.737	0.747	-0.377	8.753
1	1	trainscg	tansig	purelin	234	MSE	0.0040	0.01	0.5	0.858	0.864	0.736	0.746	-0.442	8.775
1	5	trainscg	tansig	purelin	1000	MSE	0.0038	0.01	0.5	0.858	0.865	0.737	0.748	-0.380	8.761
1	10	trainscg	tansig	purelin	1000	MSE	0.0038	0.01	0.5	0.859	0.865	0.737	0.749	-0.303	8.700

**Table 6. 112: Results for the Radial Basis Function Neural Network Model 2 with FLOW as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	3	3	SSE	1.691	1.7	0.7	0.857	0.864	0.734	0.747	-0.178	8.694



### 6.4.3 FLOW Neural Network MODEL 3

The inputs for Model 3 are FLOW, FLOW(k-1), FLOW(k-2), and the output is FLOW(k+1) one time step ahead. These are tabled below:

Table 6. 113: Input variables for FLOW Model 3

Inputs	Output
FLOW, FLOW(k-1), FLOW(k-2)	FLOW(k+1)

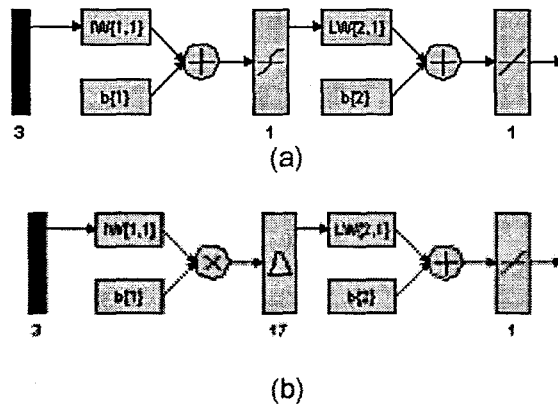
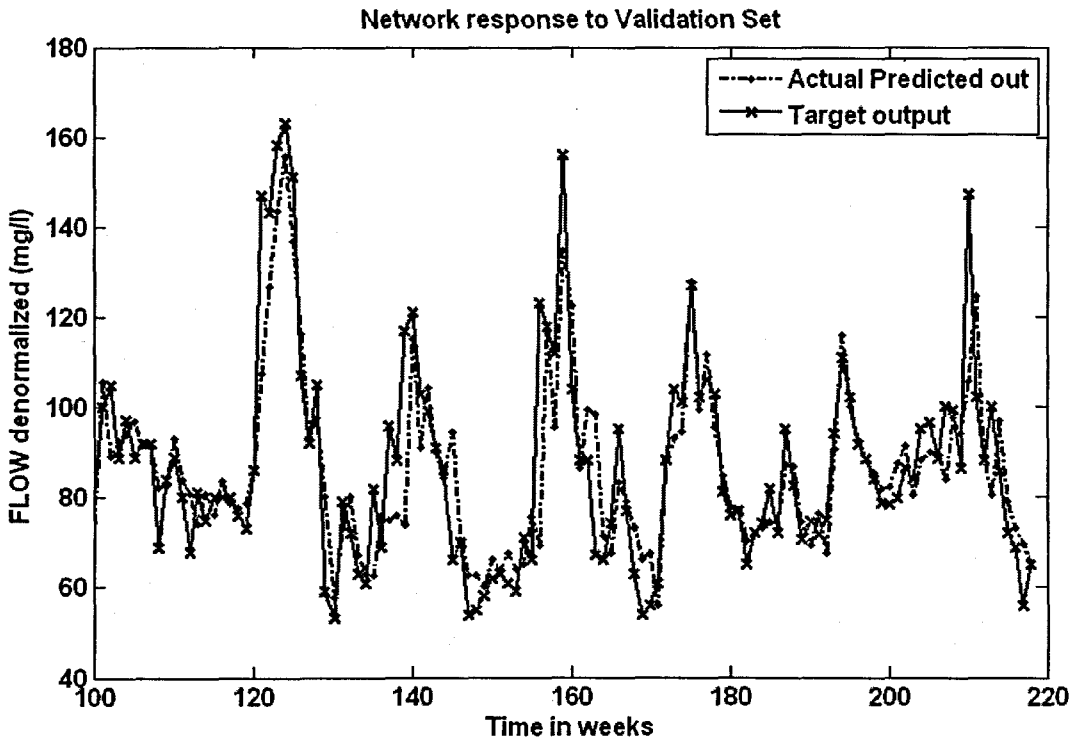
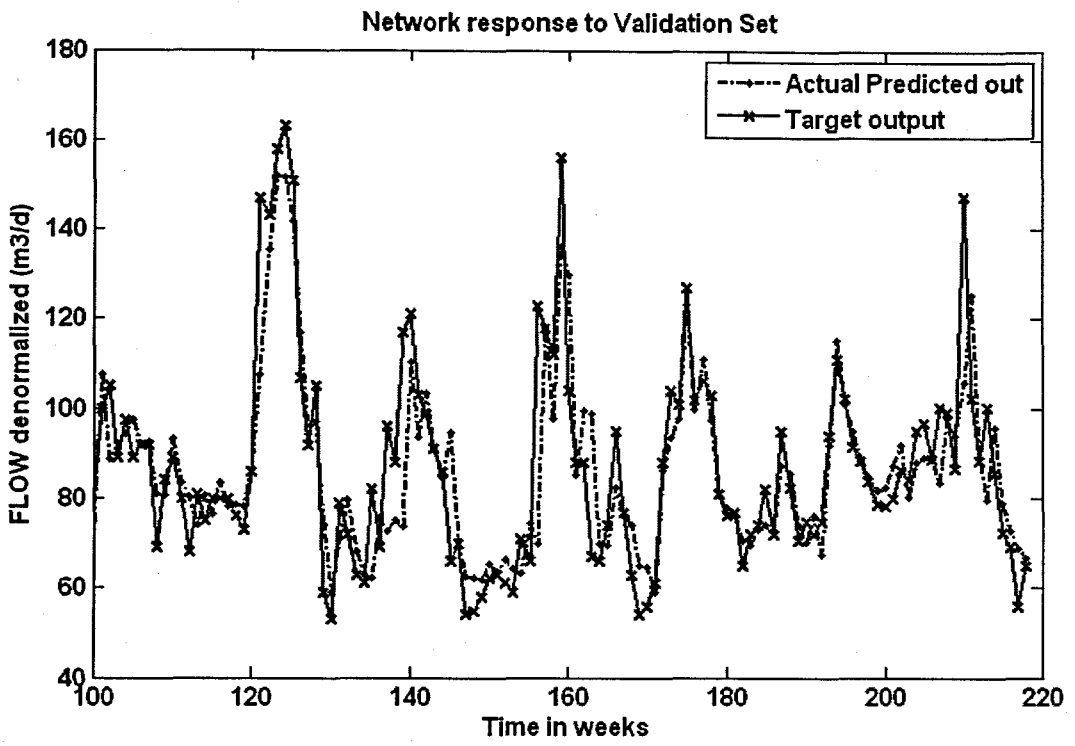


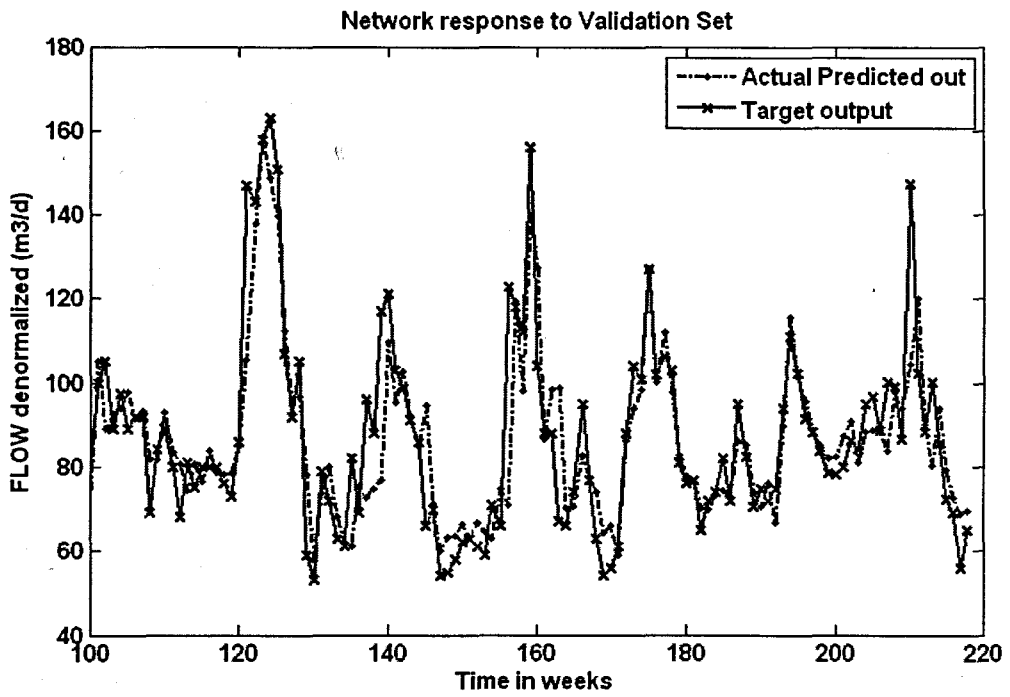
Figure 6. 72: (a) A MLP Feed-forward neural network Model 3 with 3 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 3 with 17 hidden neurons, 17 epochs



(a)



(b)



(c)

Figure 6. 73: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 3 with 10 hidden neurons and 1000 epochs; (b) ERNN Model 3 with 10 hidden neurons, 1000 epochs; and (c) RBFNN Model 3 with 17 hidden neurons, 17 epochs

**Table 6. 114: Results for the Feed-forward Neural Network Model 3 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	330	MSE	0.0038	0.6	0.5	0.858	0.864	0.736	0.746	-0.439	8.769
1	5	trainscg	tansig	purelin	500	MSE	0.0038	0.6	0.5	0.860	0.863	0.739	0.745	-0.370	8.790
1	10	trainscg	tansig	purelin	500	MSE	0.0038	0.6	0.5	0.860	0.868	0.740	0.753	-0.343	8.672
1	1	trainscg	tansig	purelin	263	MSE	0.0038	0.6	0.5	0.858	0.864	0.736	0.746	-0.438	8.769
1	5	trainscg	tansig	purelin	1000	MSE	0.0038	0.6	0.5	0.861	0.864	0.741	0.747	-0.387	8.777
1	10	trainscg	tansig	purelin	1000	MSE	0.0037	0.6	0.5	0.864	0.872	0.747	0.760	-0.446	8.677

**Table 6. 115: Results for the Elman Recurrent Neural Network Model 3 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	277	MSE	0.0038	0.01	0.5	0.858	0.864	0.736	0.746	-0.437	8.767
1	5	trainscg	tansig	purelin	500	MSE	0.0038	0.01	0.5	0.858	0.862	0.736	0.742	-0.287	8.750
1	10	trainscg	tansig	purelin	500	MSE	0.0039	0.01	0.5	0.858	0.863	0.735	0.745	-0.302	8.630
1	1	trainscg	tansig	purelin	433	MSE	0.0038	0.01	0.5	0.858	0.864	0.736	0.747	-0.436	8.763
1	5	trainscg	tansig	purelin	1000	MSE	0.0037	0.01	0.5	0.863	0.870	0.745	0.756	-0.398	8.550
1	10	trainscg	tansig	purelin	1000	MSE	0.0037	0.01	0.5	0.862	0.872	0.743	0.761	-0.401	8.527

**Table 6. 116: Results for the Radial Basis Function Neural Network Model 3 with FLOW as output.**

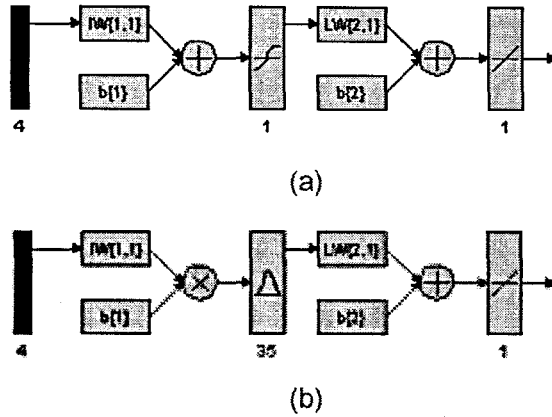
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	17	17	SSE	1.594	1.6	0.7	0.865	0.873	0.749	0.762	0.125	8.469

### 6.4.4 FLOW Neural Network MODEL 4

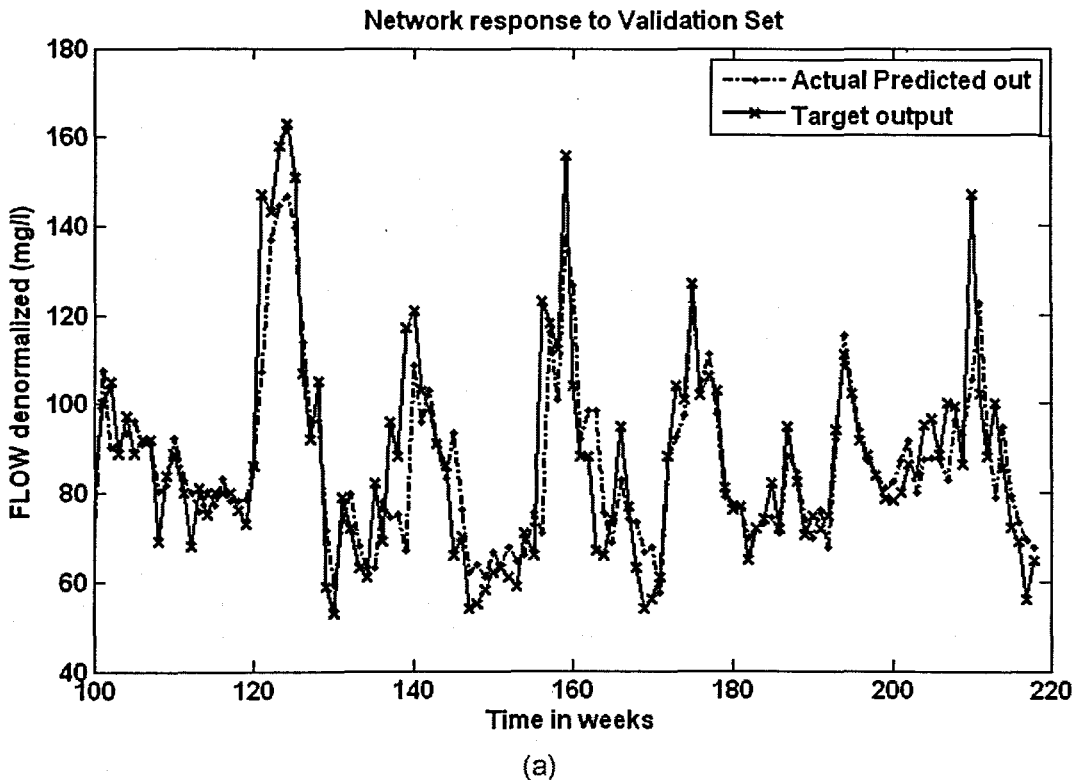
The inputs for Model 4 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 117: Input variables for FLOW Model 4**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3)	FLOW(k+1)



**Figure 6. 74: (a) A MLP Feed-forward neural network Model 4 with 4 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 4 with 35 hidden neurons, 35 epochs**



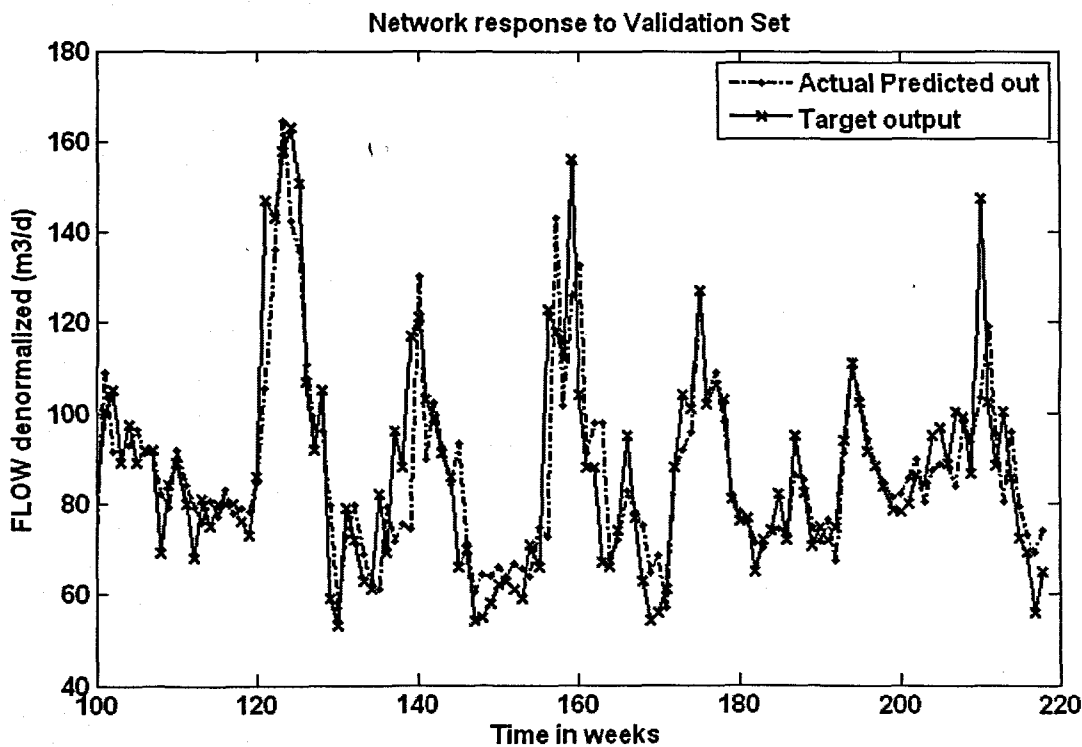
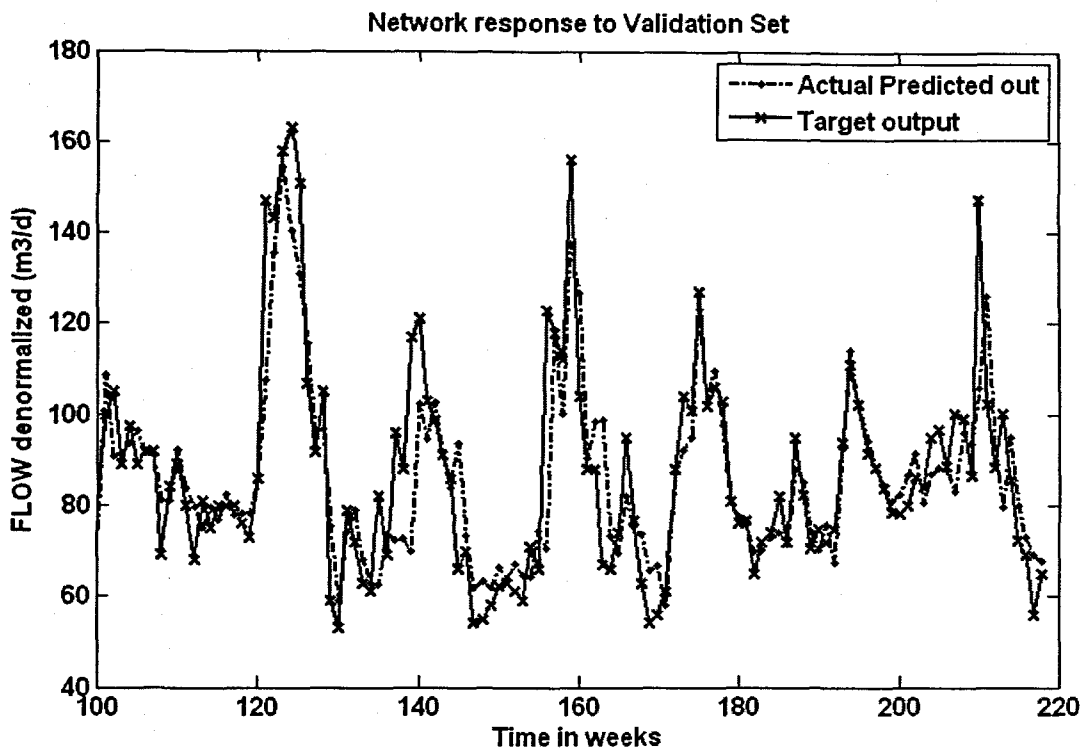


Figure 6. 75: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 4 with 5 hidden neurons and 500 epochs; (b) ERNN Model 4 with 5 hidden neurons, 1000 epochs; and (c) RBFNN Model 4 with 35 hidden neurons, 35 epochs

**Table 6. 118: Results for the Feed-forward Neural Network Model 4 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	244	MSE	0.0038	0.6	0.5	0.858	0.864	0.736	0.747	-0.444	8.754
1	5	trainscg	tansig	purelin	500	MSE	0.0037	0.6	0.5	0.863	0.865	0.744	0.748	-0.525	8.765
1	10	trainscg	tansig	purelin	500	MSE	0.0038	0.6	0.5	0.861	0.861	0.741	0.741	-0.352	8.744
1	1	trainscg	tansig	purelin	347	MSE	0.0038	0.6	0.5	0.858	0.864	0.736	0.747	-0.444	8.755
1	5	trainscg	tansig	purelin	1000	MSE	0.0036	0.6	0.5	0.868	0.854	0.754	0.730	-0.440	9.059
1	10	trainscg	tansig	purelin	1000	MSE	0.0035	0.6	0.5	0.868	0.854	0.754	0.730	-0.440	9.059

**Table 6. 119: Results for the Elman Recurrent Neural Network Model 4 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	392	MSE	0.0038	0.01	0.5	0.858	0.864	0.736	0.747	-0.442	8.751
1	5	trainscg	tansig	purelin	500	MSE	0.0038	0.01	0.5	0.859	0.865	0.737	0.748	-0.292	8.769
1	10	trainscg	tansig	purelin	500	MSE	0.0038	0.01	0.5	0.860	0.860	0.740	0.739	-0.499	8.818
1	1	trainscg	tansig	purelin	344	MSE	0.0038	0.01	0.5	0.858	0.864	0.736	0.747	-0.442	8.750
1	5	trainscg	tansig	purelin	1000	MSE	0.0037	0.01	0.5	0.868	0.864	0.753	0.747	-0.425	8.675
1	10	trainscg	tansig	purelin	1000	MSE	0.0037	0.01	0.5	0.863	0.865	0.744	0.748	-0.438	8.686

**Table 6. 120: Results for the Radial Basis Function Neural Network Model 4 with FLOW as output.**

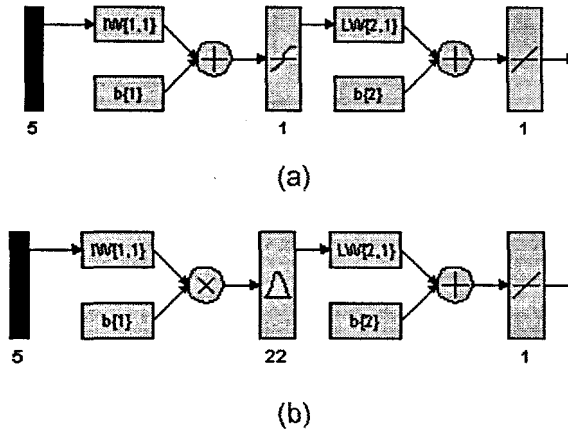
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	35	35	SSE	1.496	1.5	0.8	0.874	0.854	0.764	0.729	-0.205	9.150

### 6.4.5 FLOW Neural Network MODEL 5

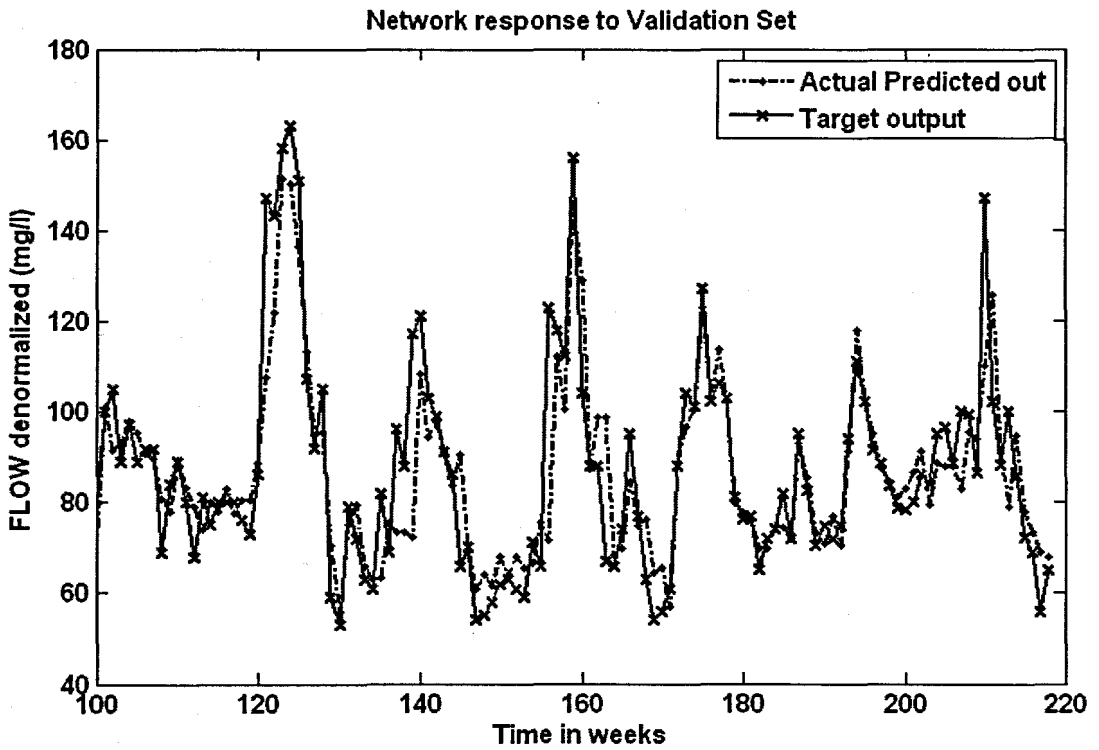
The inputs for Model 5 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 121: Input variables for FLOW Model 5**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k)	FLOW(k+1)



**Figure 6. 76:**(a) A MLP Feed-forward neural network Model 5 with 5 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 5 with 22 hidden neurons, 22 epochs



(a)

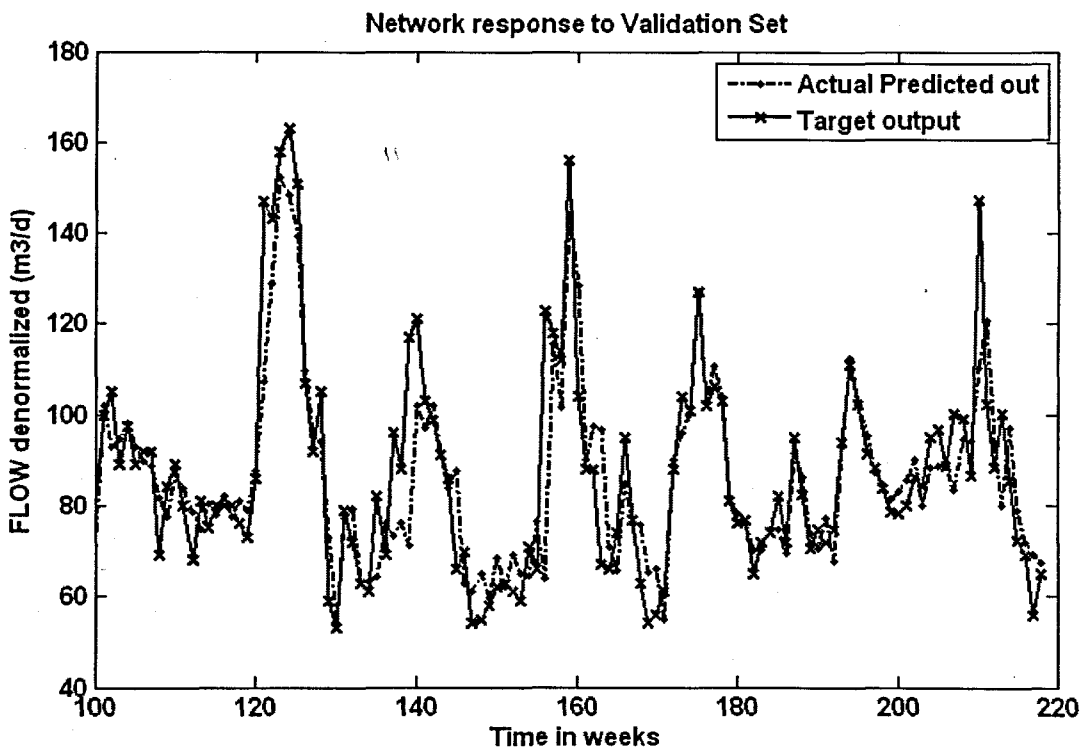
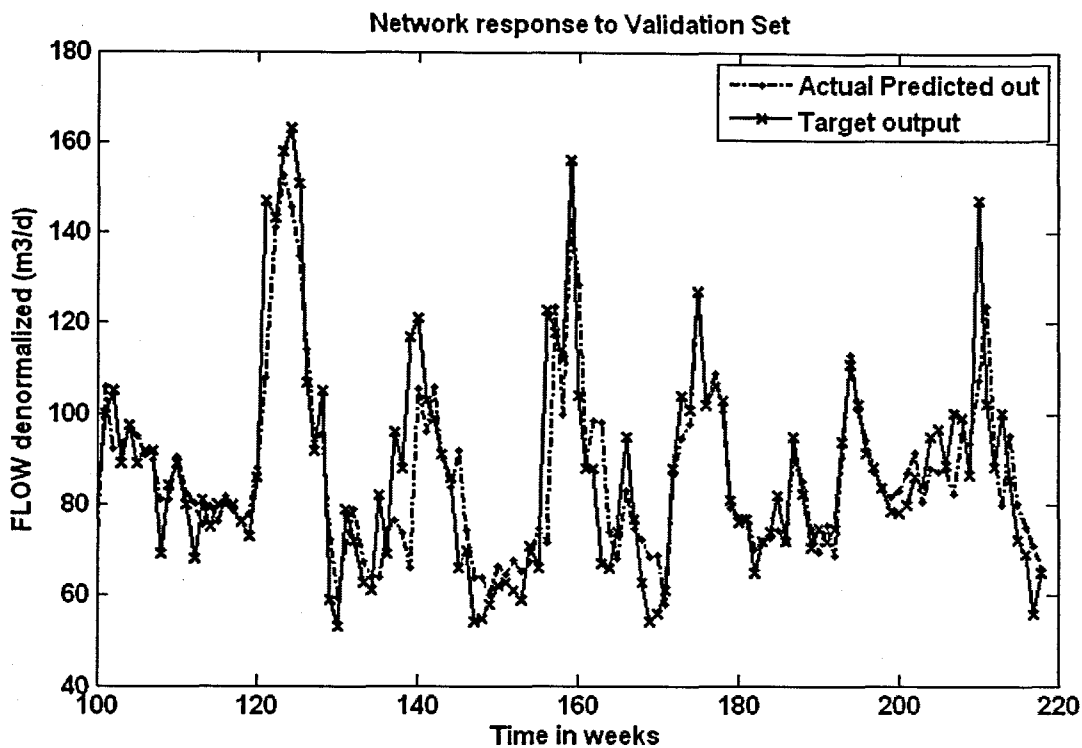


Figure 6. 77: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 5 with 5 hidden neurons and 1000 epochs; (b) ERNN Model 5 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 5 with 22 hidden neurons, 22 epochs



**Table 6. 122: Results for the Feed-forward Neural Network Model 5 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	272	MSE	0.0038	0.6	0.5	0.859	0.866	0.738	0.751	-0.411	8.653
1	5	trainscg	tansig	purelin	500	MSE	0.0037	0.6	0.5	0.864	0.867	0.747	0.751	-0.750	8.844
1	10	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.871	0.847	0.759	0.717	-0.753	9.184
1	1	trainscg	tansig	purelin	244	MSE	0.0038	0.6	0.5	0.859	0.866	0.738	0.751	-0.412	8.655
1	5	trainscg	tansig	purelin	1000	MSE	0.0038	0.6	0.5	0.868	0.866	0.754	0.751	-0.579	8.640
1	10	trainscg	tansig	purelin	1000	MSE	0.0034	0.6	0.5	0.877	0.855	0.769	0.730	-0.809	9.200

**Table 6. 123: Results for the Elman Recurrent Neural Network Model 5 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	261	MSE	0.0038	0.01	0.5	0.859	0.866	0.738	0.751	-0.412	8.655
1	5	trainscg	tansig	purelin	500	MSE	0.0037	0.01	0.5	0.864	0.868	0.747	0.753	-0.375	8.569
1	10	trainscg	tansig	purelin	500	MSE	0.0035	0.01	0.5	0.870	0.865	0.757	0.749	-0.635	8.794
1	1	trainscg	tansig	purelin	184	MSE	0.0038	0.01	0.5	0.859	0.866	0.738	0.751	-0.412	8.655
1	5	trainscg	tansig	purelin	1000	MSE	0.0036	0.01	0.5	0.866	0.861	0.750	0.742	-0.690	8.960
1	10	trainscg	tansig	purelin	1000	MSE	0.0034	0.01	0.5	0.874	0.864	0.763	0.747	-0.459	8.750

**Table 6. 124: Results for the Radial Basis Function Neural Network Model 5 with FLOW as output.**

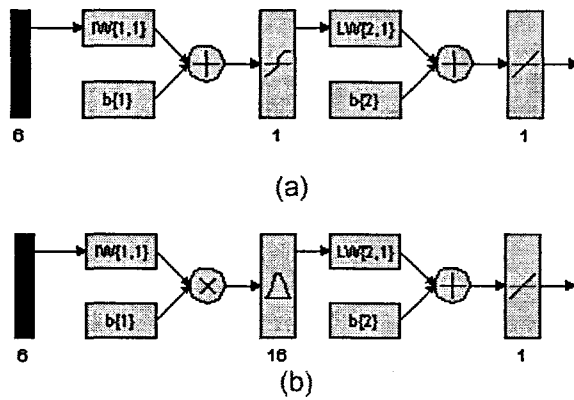
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	22	22	SSE	1.486	1.5	0.5	0.875	0.857	0.766	0.734	-0.740	8.766

### 6.4.6 FLOW Neural Network MODEL 6

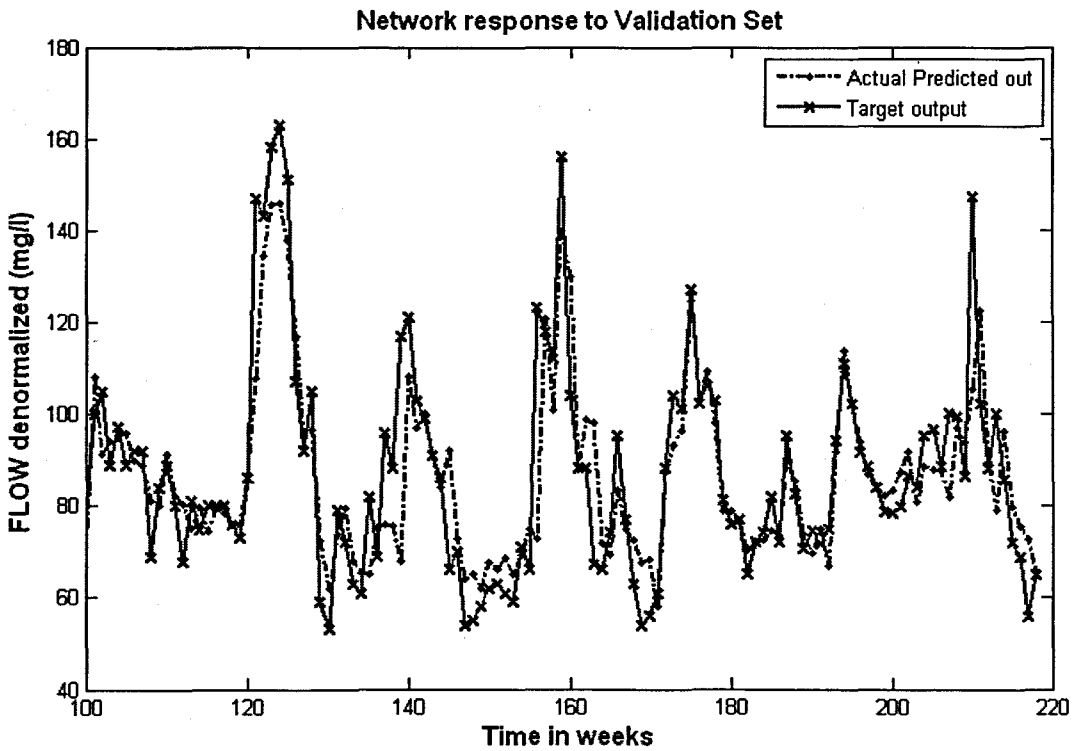
The inputs for Model 6 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 125: Input variables for FLOW Model 6**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k)	FLOW(k+1)



**Figure 6. 78: (a) A MLP Feed-forward neural network Model 6 with 6 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 6 with 16 hidden neurons, 16 epochs**



(a)

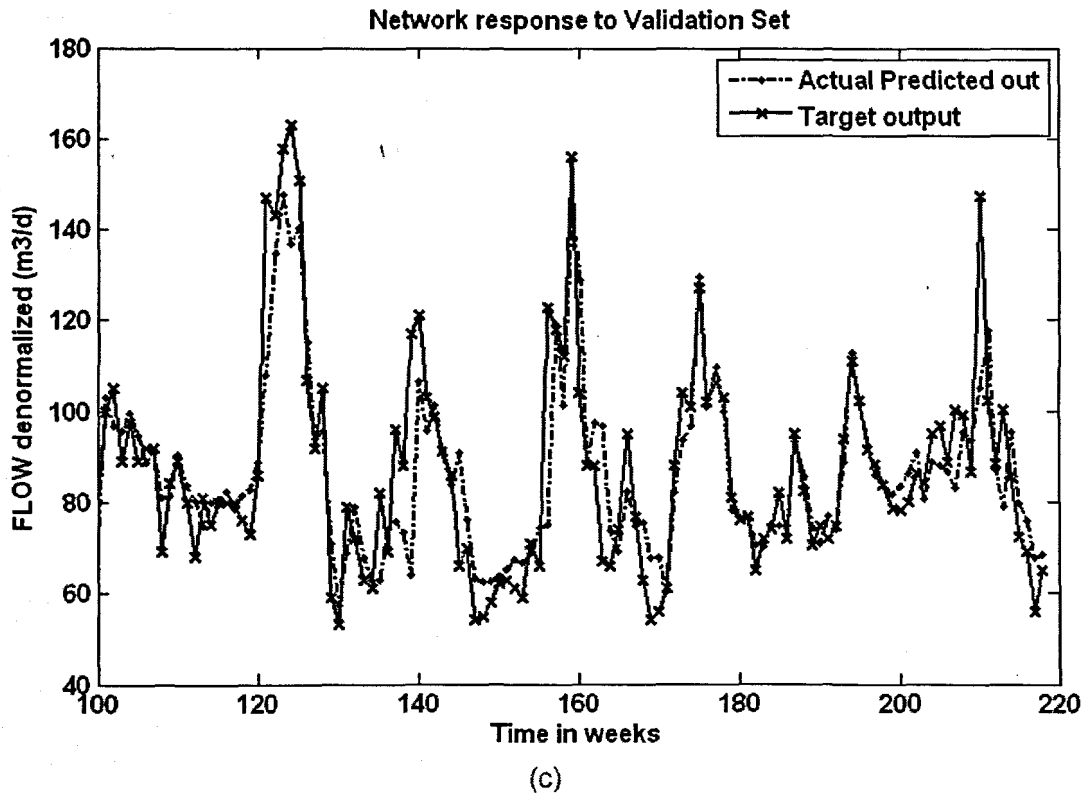
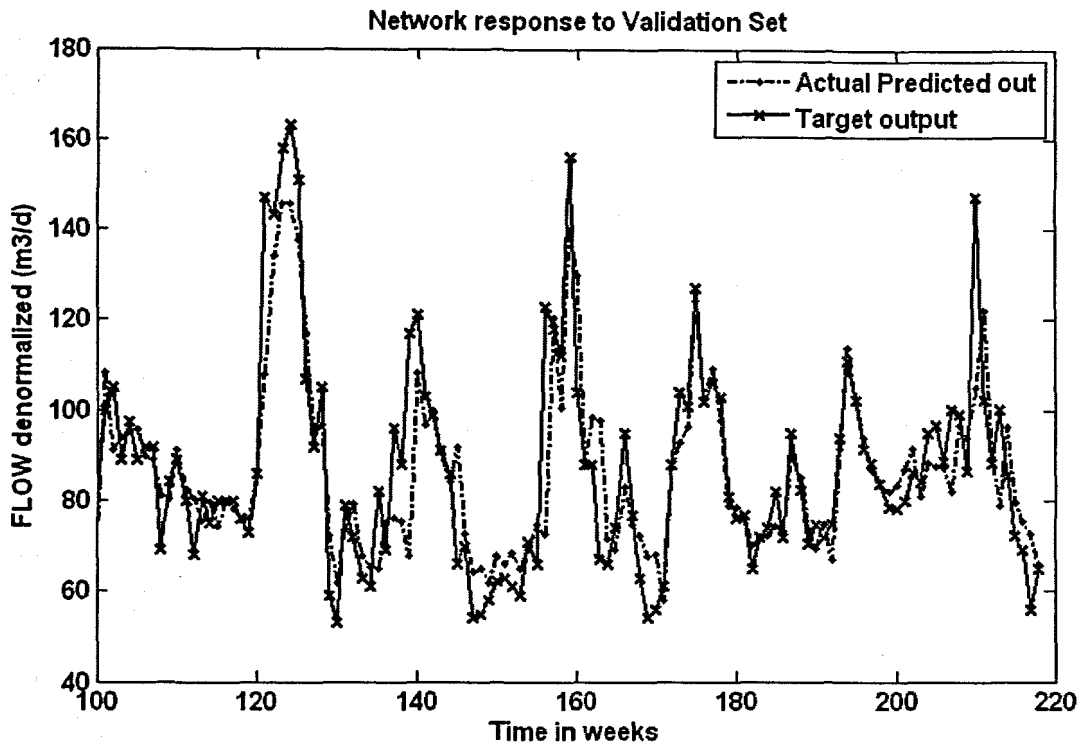


Figure 6. 79: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 6 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 6 with 1 hidden neuron, 500 epochs; and (c) RBFNN Model 6 with 16 hidden neurons, 16 epochs

**Table 6. 126: Results for the Feed-forward Neural Network Model 6 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	310	MSE	0.0038	0.6	0.5	0.859	0.868	0.738	0.753	-0.354	8.637
1	5	trainscg	tansig	purelin	500	MSE	0.0037	0.6	0.5	0.865	0.859	0.748	0.737	-0.122	8.794
1	10	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.871	0.846	0.759	0.716	-0.903	9.285
1	1	trainscg	tansig	purelin	324	MSE	0.0038	0.6	0.5	0.859	0.868	0.738	0.753	-0.353	8.635
1	5	trainscg	tansig	purelin	1000	MSE	0.0038	0.6	0.5	0.867	0.861	0.751	0.742	-0.428	8.878
1	10	trainscg	tansig	purelin	1000	MSE	0.0035	0.6	0.5	0.873	0.854	0.762	0.729	-0.204	8.955

**Table 6. 127: Results for the Elman Recurrent Neural Network Model 6 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	279	MSE	0.0038	0.01	0.5	0.859	0.868	0.738	0.753	-0.356	8.639
1	5	trainscg	tansig	purelin	500	MSE	0.0038	0.01	0.5	0.862	0.859	0.742	0.739	-0.378	8.863
1	10	trainscg	tansig	purelin	500	MSE	0.0037	0.01	0.5	0.865	0.865	0.749	0.748	-0.506	8.703
1	1	trainscg	tansig	purelin	240	MSE	0.0038	0.01	0.5	0.859	0.868	0.738	0.753	-0.353	8.635
1	5	trainscg	tansig	purelin	1000	MSE	0.0036	0.01	0.5	0.869	0.863	0.755	0.746	-0.360	8.736
1	10	trainscg	tansig	purelin	1000	MSE	0.0034	0.01	0.5	0.875	0.850	0.765	0.722	-0.617	9.052

**Table 6. 128: Results for the Radial Basis Function Neural Network Model 6 with FLOW as output.**

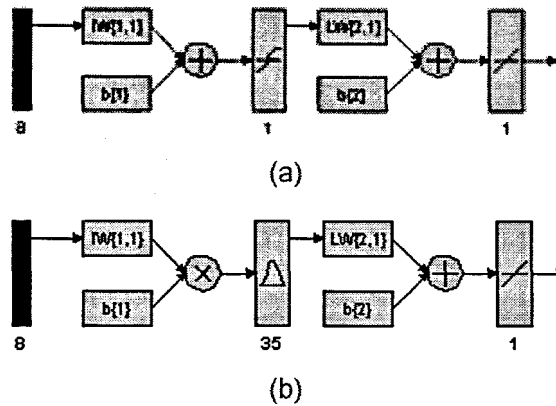
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	16	16	SSE	1.594	1.6	0.8	0.865	0.860	0.749	0.739	-0.338	8.807

### 6.4.7 FLOW Neural Network MODEL 7

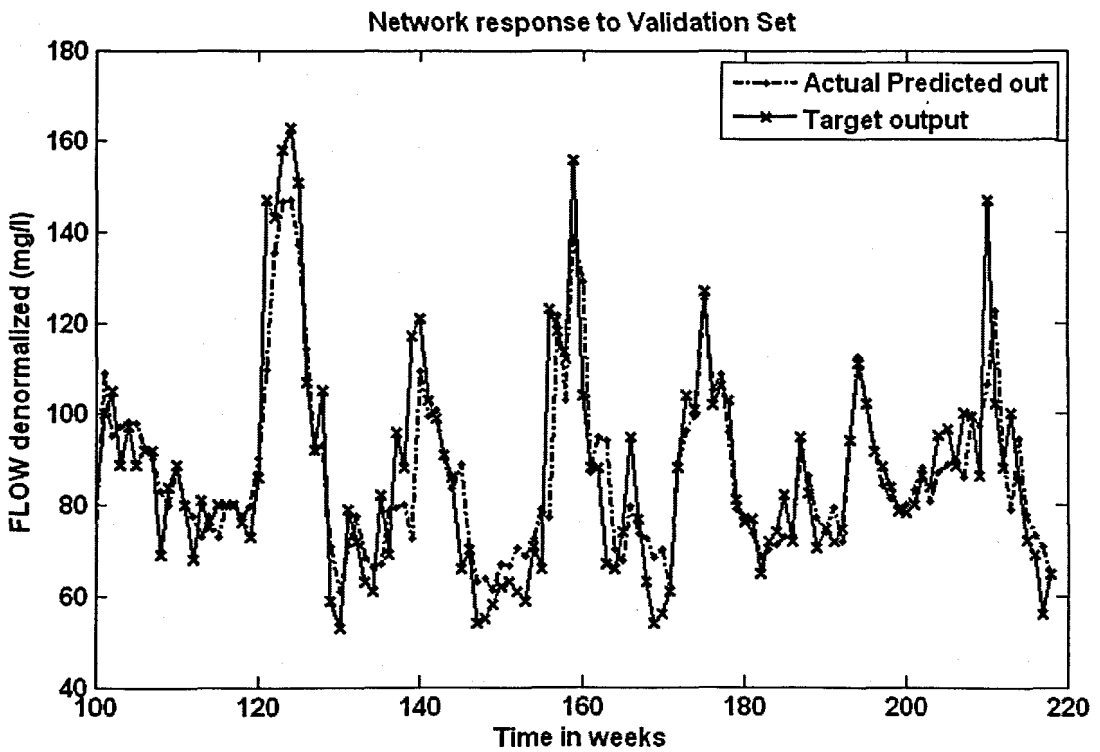
The inputs for Model 7 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), MONTH, and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 129: Input variables for FLOW Model 7**

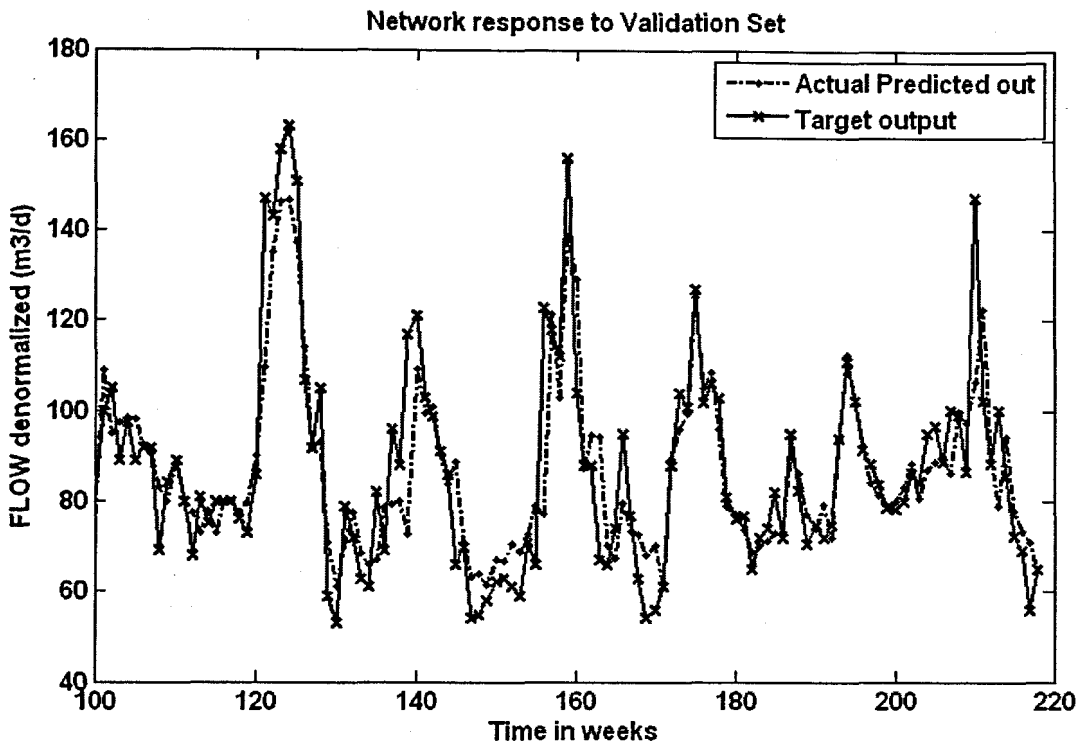
Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), Month	FLOW(k+1)



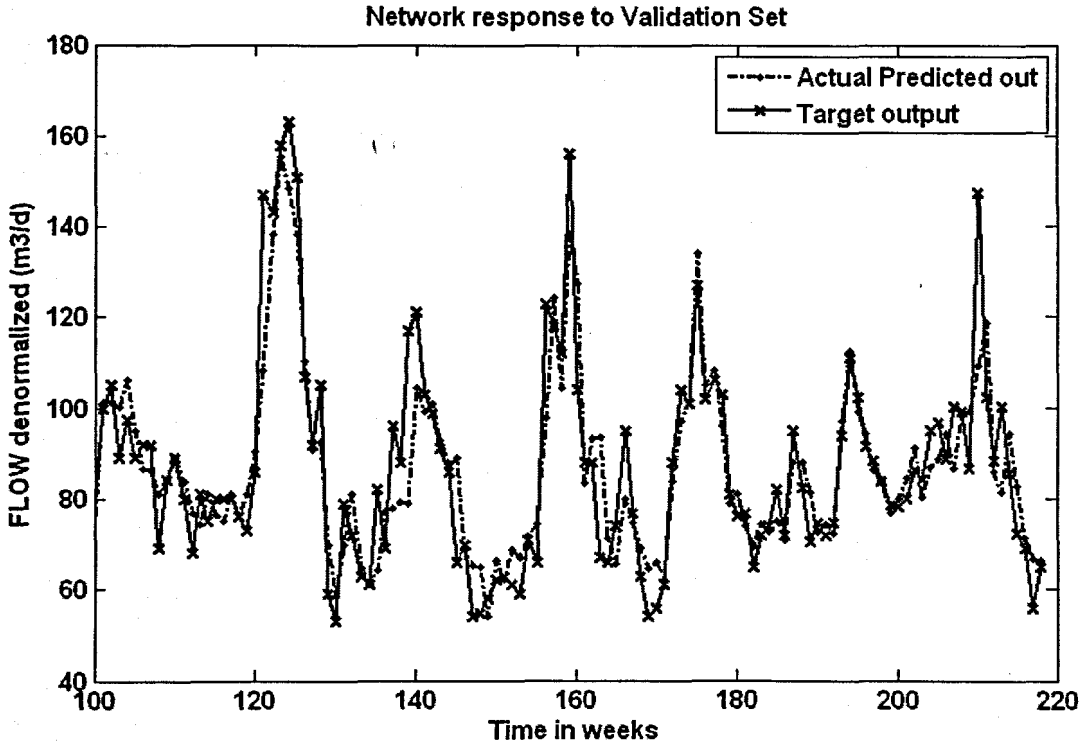
**Figure 6. 80:** (a) A MLP Feed-forward neural network Model 7 with 8 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 7 with 35 hidden neurons, 35 epochs



(a)

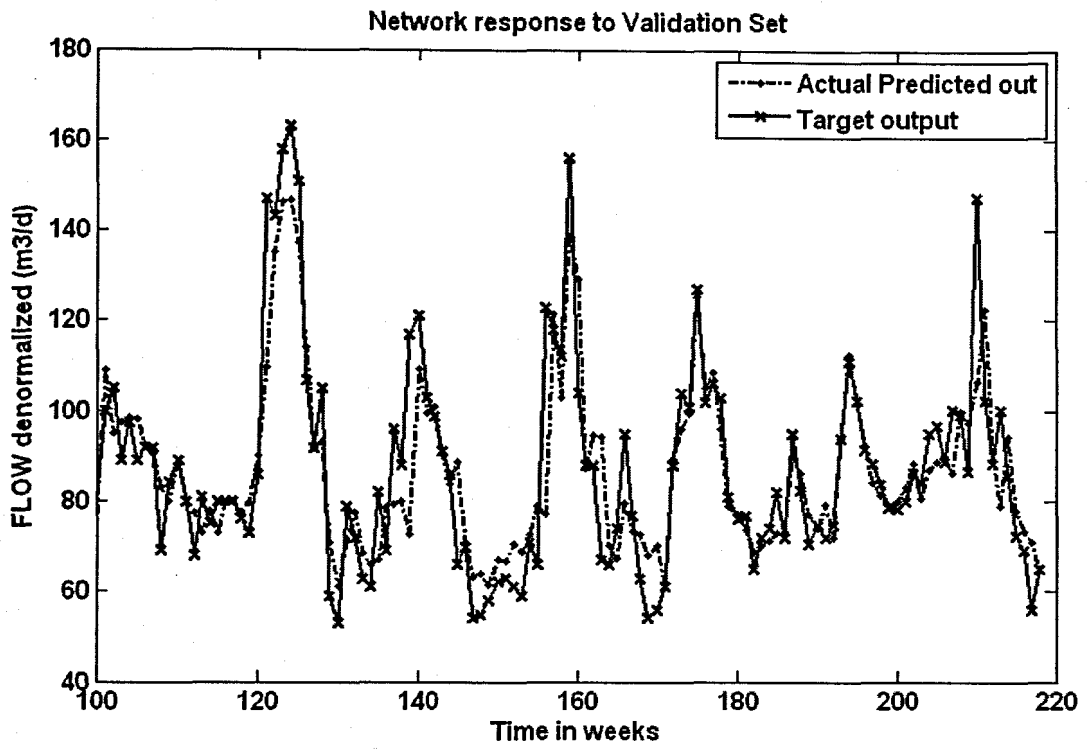


(b)

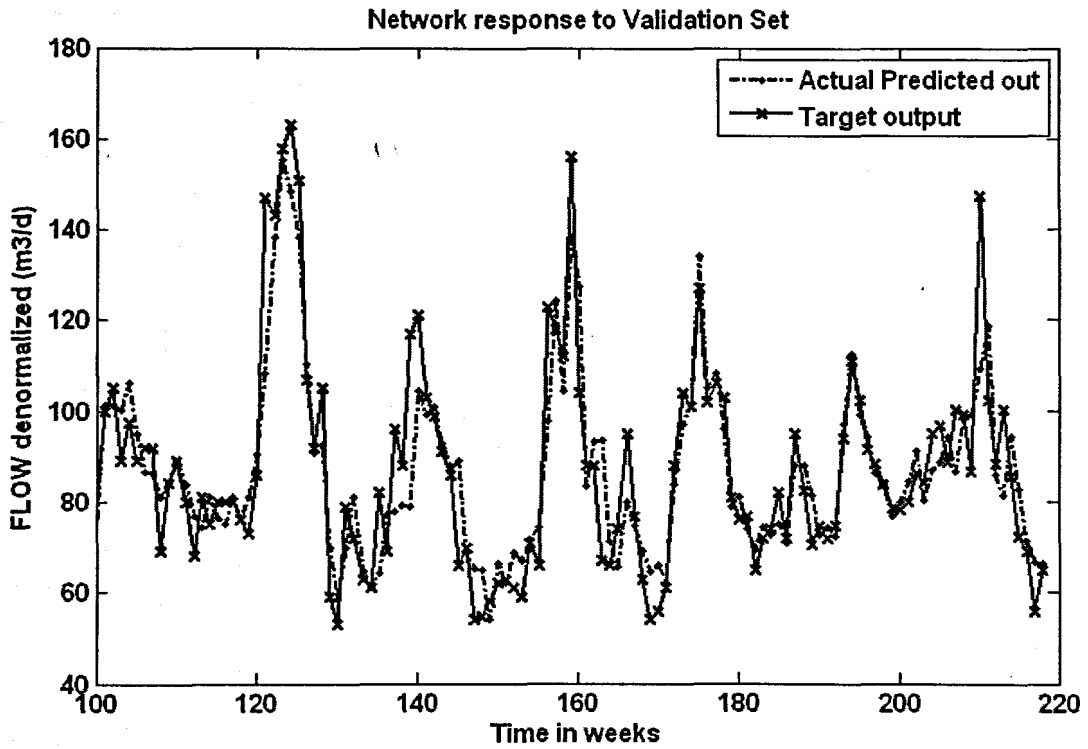


(c)

Figure 6. 81: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 7 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 7 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 7 with 35 hidden neurons, 35 epochs



(b)



(c)

Figure 6. 81: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 7 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 7 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 7 with 35 hidden neurons, 35 epochs

**Table 6. 130: Results for the Feed-forward Neural Network Model 7 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0037	0.6	0.5	0.866	0.880	0.749	0.774	-0.412	8.361
1	5	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.872	0.873	0.761	0.762	-0.423	8.474
1	10	trainscg	tansig	purelin	500	MSE	0.0032	0.6	0.5	0.885	0.856	0.783	0.734	-0.252	9.130
1	1	trainscg	tansig	purelin	810	MSE	0.0039	0.6	0.5	0.866	0.880	0.749	0.774	-0.407	8.353
1	5	trainscg	tansig	purelin	1000	MSE	0.0033	0.6	0.5	0.878	0.867	0.770	0.751	-0.512	8.599
1	10	trainscg	tansig	purelin	1000	MSE	0.0031	0.6	0.5	0.888	0.869	0.788	0.755	-0.236	8.669

**Table 6. 131: Results for the Elman Recurrent Neural Network Model 7 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	439	MSE	0.0037	0.01	0.5	0.866	0.880	0.749	0.774	-0.411	8.359
1	5	trainscg	tansig	purelin	500	MSE	0.0034	0.01	0.5	0.877	0.879	0.770	0.772	-0.105	8.443
1	10	trainscg	tansig	purelin	500	MSE	0.0033	0.01	0.5	0.881	0.866	0.775	0.751	-0.497	8.686
1	1	trainscg	tansig	purelin	511	MSE	0.0037	0.01	0.5	0.866	0.880	0.749	0.774	-0.410	8.357
1	5	trainscg	tansig	purelin	1000	MSE	0.0034	0.01	0.5	0.876	0.881	0.767	0.776	-0.171	8.506
1	10	trainscg	tansig	purelin	1000	MSE	0.0032	0.01	0.5	0.885	0.865	0.783	0.749	-0.084	8.652

**Table 6. 132: Results for the Radial Basis Function Neural Network Model 7 with FLOW as output.**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	35	35	SSE	1.394	1.4	0.9	0.883	0.886	0.780	0.785	-0.364	8.419

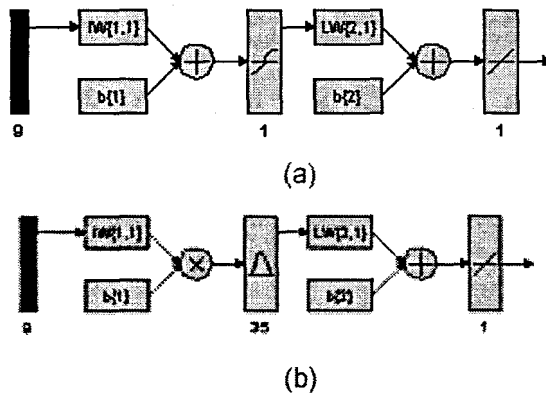


### 6.4.8 FLOW Neural Network MODEL 8

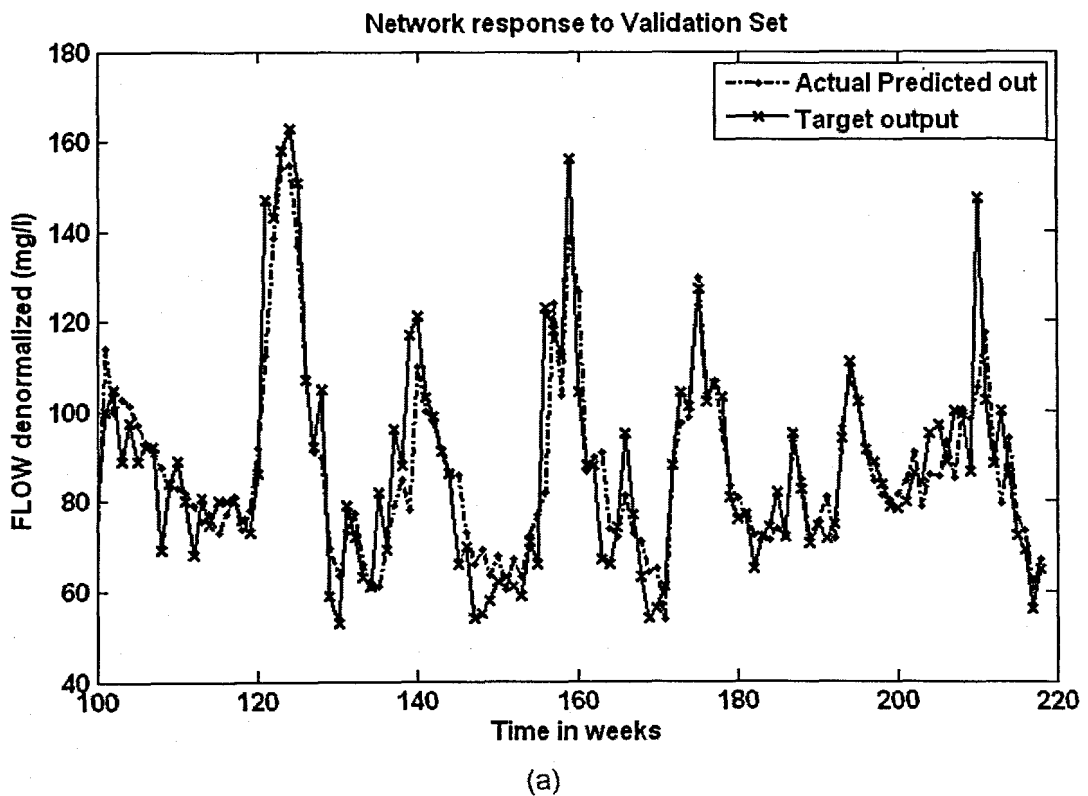
The inputs for Model 8 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), MONTH, Minimum Temperature, and the output is FLOW(k+1) one time step ahead. These are tabled below:

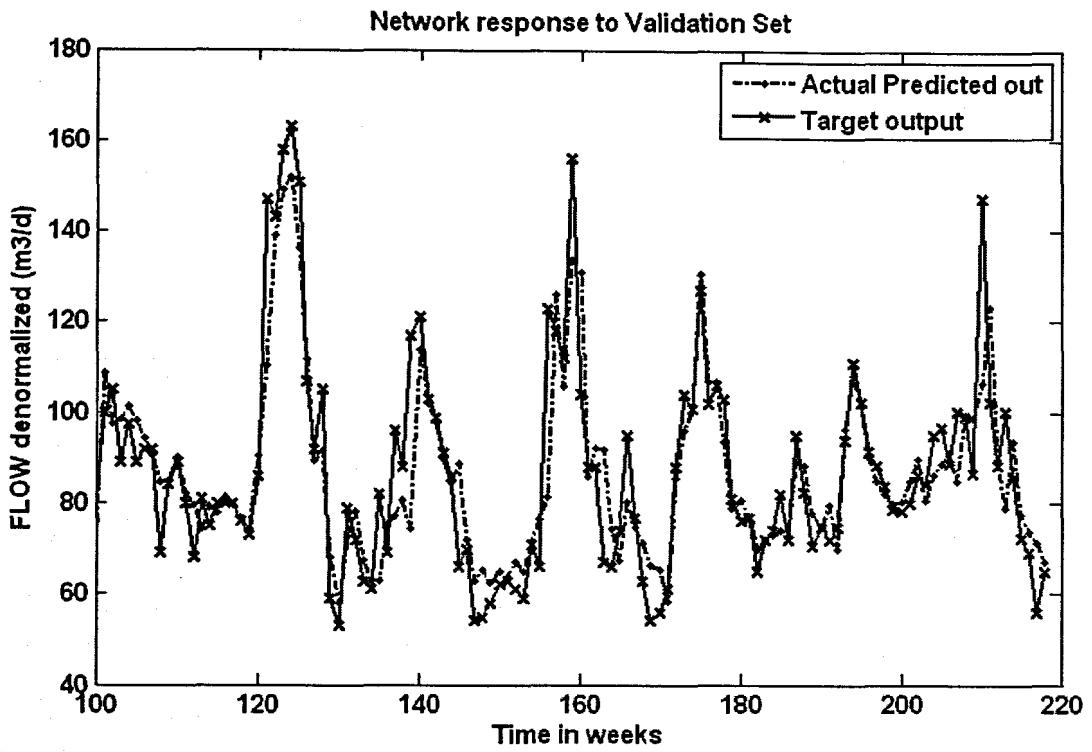
**Table 6. 133: Input variables for FLOW Model 8**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), Month, Minimum Temperature	FLOW(k+1)

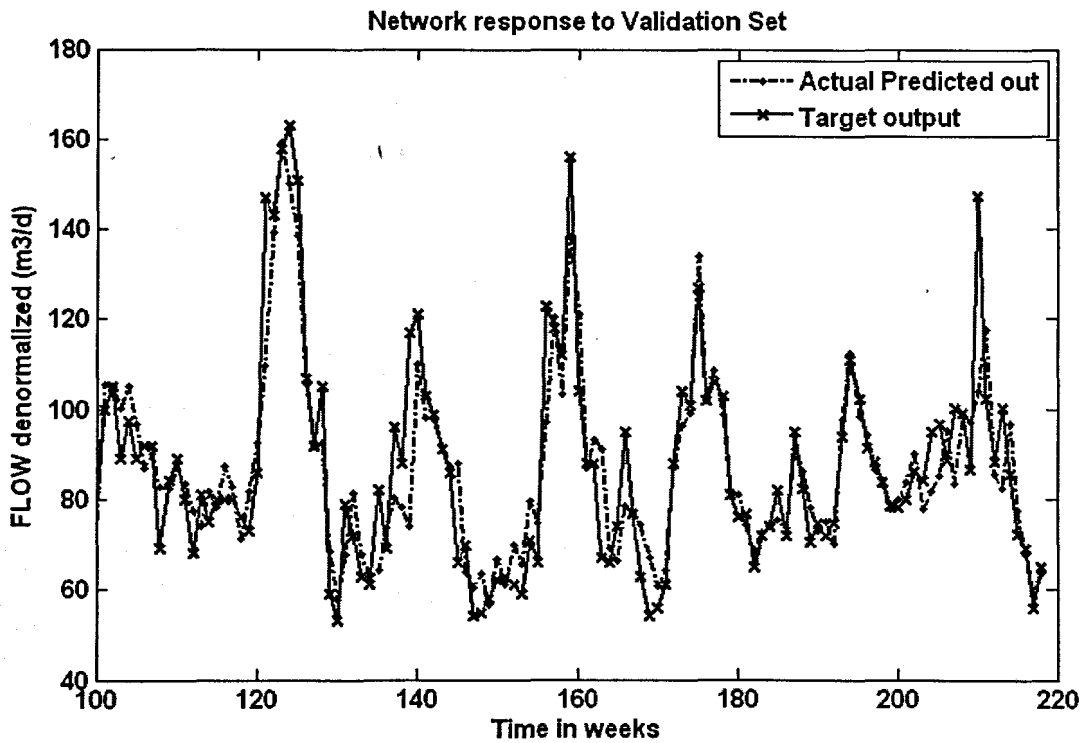


**Figure 6. 82:** (a) A MLP Feed-forward neural network Model 8 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 8 with 35 hidden neurons, 35 epochs





(b)



(c)

Figure 6. 83: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 8 with 5 hidden neurons and 500 epochs; (b) ERNN Model 8 with 5 hidden neurons, 500 epochs; and (c) RBFNN Model 8 with 35 hidden neurons, 35 epochs

**Table 6. 134: Results for the Feed-forward Neural Network Model 8 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	454	MSE	0.0037	0.6	0.5	0.866	0.880	0.749	0.774	-0.395	8.369
1	5	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.873	0.884	0.762	0.781	-0.312	8.251
1	10	trainscg	tansig	purelin	500	MSE	0.0033	0.6	0.5	0.881	0.866	0.776	0.749	-0.606	8.837
1	1	trainscg	tansig	purelin	418	MSE	0.0039	0.6	0.5	0.866	0.880	0.749	0.774	-0.396	8.371
1	5	trainscg	tansig	purelin	1000	MSE	0.0033	0.6	0.5	0.880	0.876	0.774	0.767	-0.439	8.581
1	10	trainscg	tansig	purelin	1000	MSE	0.0028	0.6	0.5	0.898	0.860	0.806	0.740	-0.607	9.041

**Table 6. 135: Results for the Elman Recurrent Neural Network Model 8 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0037	0.01	0.5	0.866	0.880	0.749	0.774	-0.396	8.370
1	5	trainscg	tansig	purelin	500	MSE	0.0035	0.01	0.5	0.874	0.881	0.763	0.776	-0.229	8.288
1	10	trainscg	tansig	purelin	500	MSE	0.0031	0.01	0.5	0.886	0.873	0.784	0.763	-0.326	8.290
1	1	trainscg	tansig	purelin	493	MSE	0.0037	0.01	0.5	0.866	0.880	0.749	0.774	-0.395	8.369
1	5	trainscg	tansig	purelin	1000	MSE	0.0032	0.01	0.5	0.882	0.874	0.779	0.763	-0.733	8.551
1	10	trainscg	tansig	purelin	1000	MSE	0.0030	0.01	0.5	0.892	0.869	0.795	0.756	-0.272	8.613

**Table 6. 136: Results for the Radial Basis Function Neural Network Model 8 with FLOW as output.**

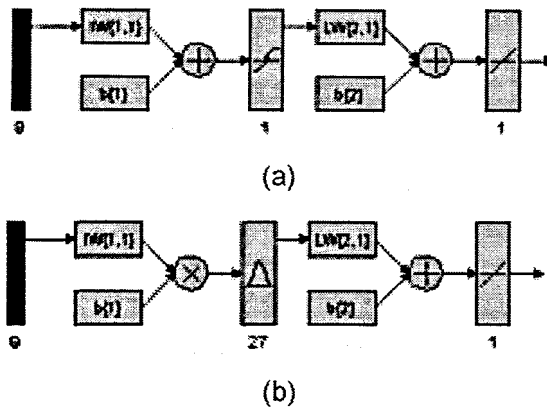
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	35	35	SSE	1.394	1.4	1	0.883	0.889	0.780	0.790	-0.523	8.150

### 6.4.9 FLOW Neural Network MODEL 9

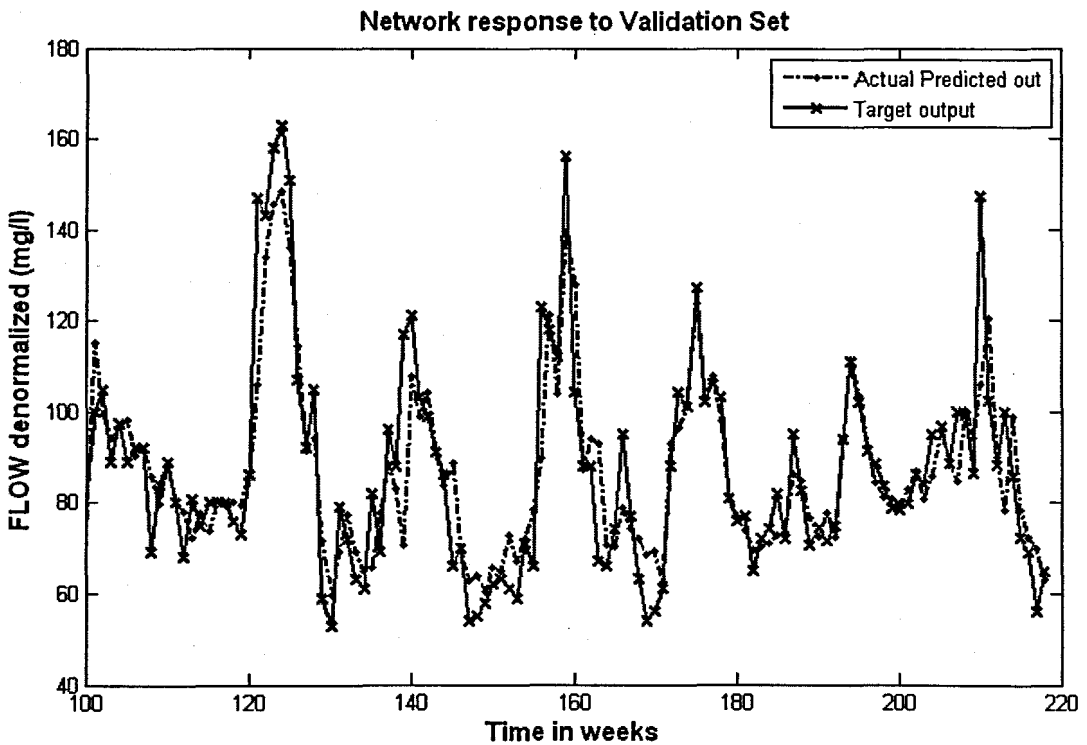
The inputs for Model 9 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), MONTH, Rain, and the output is FLOW(k+1) one time step ahead. These are tabled below:

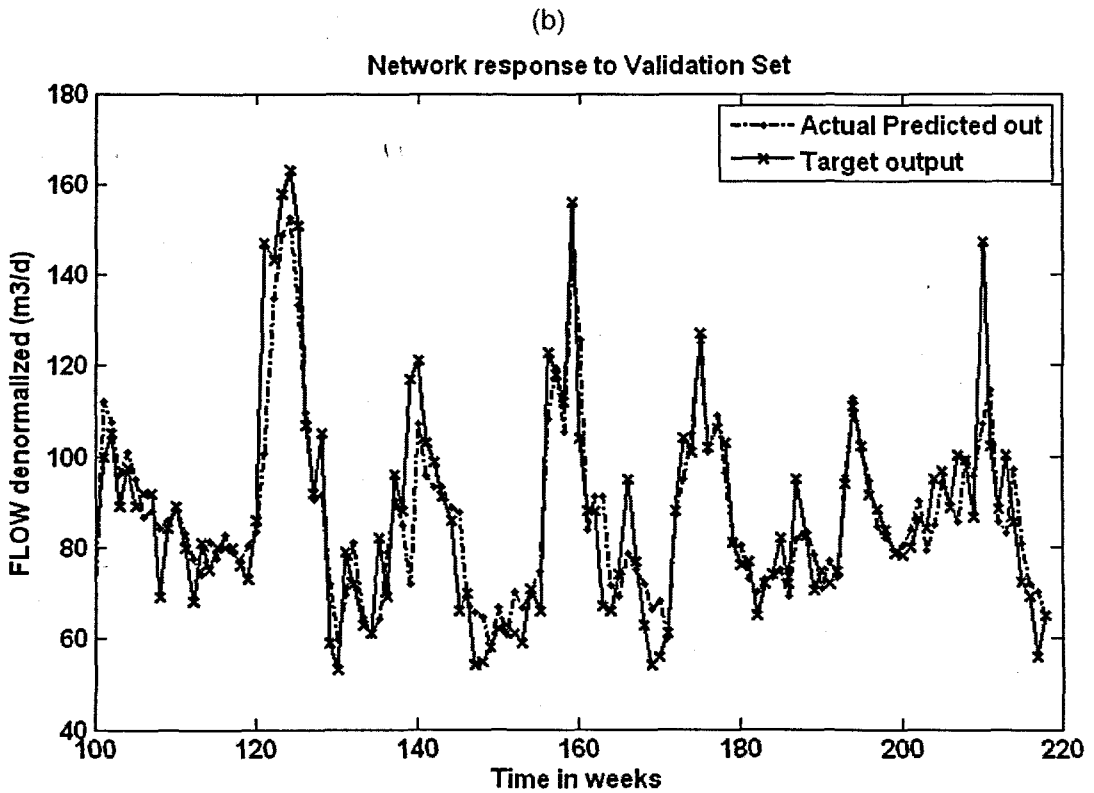
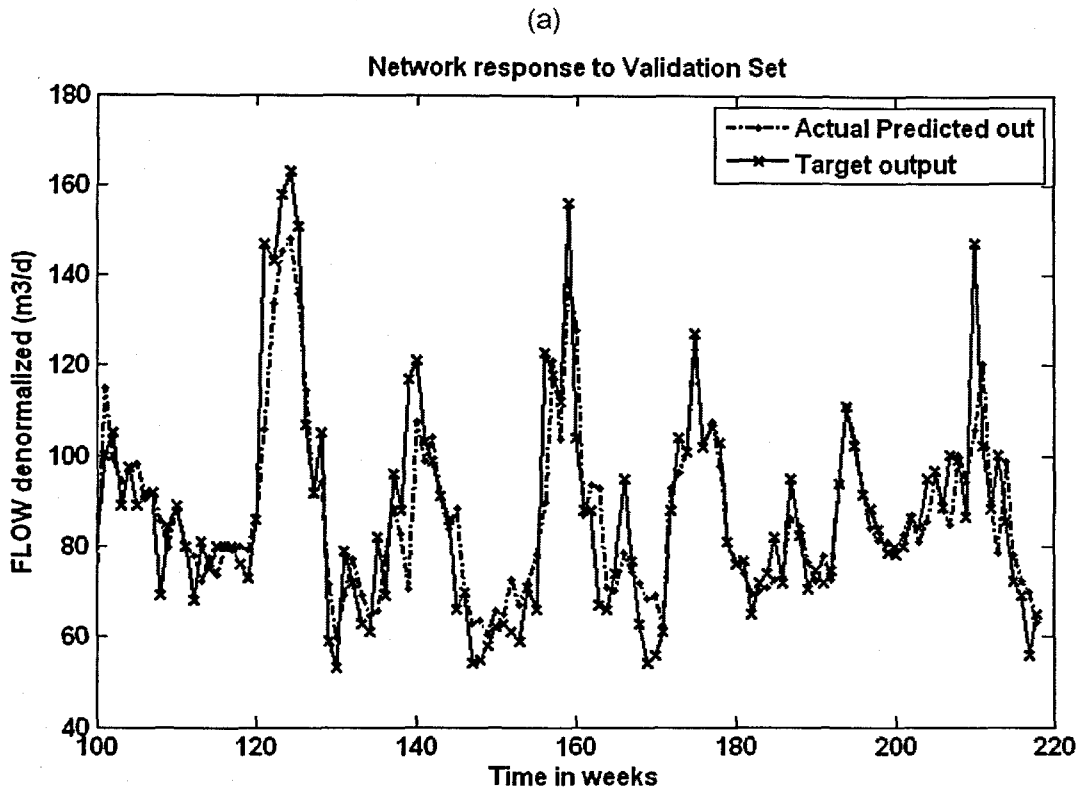
**Table 6. 137: Input variables for FLOW Model 9**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), Month, Rain	FLOW(k+1)



**Figure 6. 84: (a) A MLP Feed-forward neural network Model 9 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 9 with 27 hidden neurons, 27 epochs**





(c)

Figure 6. 85: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 9 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 9 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 9 with 27 hidden neurons, 27 epochs

**Table 6. 138: Results for the Feed-forward Neural Network Model 9 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.873	0.882	0.762	0.777	-0.371	8.339
1	5	trainscg	tansig	purelin	500	MSE	0.0032	0.6	0.5	0.885	0.874	0.783	0.764	-0.407	8.488
1	10	trainscg	tansig	purelin	500	MSE	0.0028	0.6	0.5	0.897	0.868	0.805	0.753	0.156	8.881
1	1	trainscg	tansig	purelin	481	MSE	0.0035	0.6	0.5	0.873	0.882	0.762	0.777	-0.373	8.337
1	5	trainscg	tansig	purelin	1000	MSE	0.0031	0.6	0.5	0.886	0.862	0.786	0.744	-0.372	8.856
1	10	trainscg	tansig	purelin	1000	MSE	0.0030	0.6	0.5	0.889	0.866	0.791	0.751	-0.286	8.810

**Table 6. 139: Results for the Elman Recurrent Neural Network Model 9 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0035	0.01	0.5	0.873	0.882	0.762	0.777	-0.374	8.339
1	5	trainscg	tansig	purelin	500	MSE	0.0032	0.01	0.5	0.882	0.881	0.777	0.776	-0.504	8.332
1	10	trainscg	tansig	purelin	500	MSE	0.0033	0.01	0.5	0.881	0.878	0.776	0.771	-0.372	8.429
1	1	trainscg	tansig	purelin	501	MSE	0.0035	0.01	0.5	0.873	0.882	0.762	0.777	-0.373	8.337
1	5	trainscg	tansig	purelin	1000	MSE	0.0032	0.01	0.5	0.882	0.881	0.778	0.776	-0.500	8.396
1	10	trainscg	tansig	purelin	1000	MSE	0.0029	0.01	0.5	0.894	0.862	0.799	0.743	-0.213	8.930

**Table 6. 140: Results for the Radial Basis Function Neural Network Model 9 with FLOW as output.**

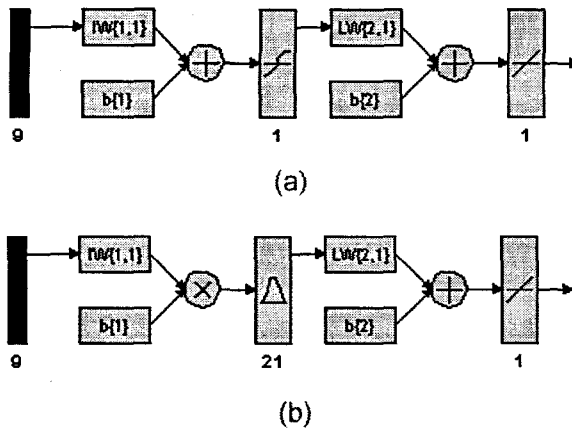
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	27	27	SSE	1.388	1.4	0.9	0.884	0.883	0.781	0.779	-0.340	8.410

### 6.4.10 FLOW Neural Network MODEL 10

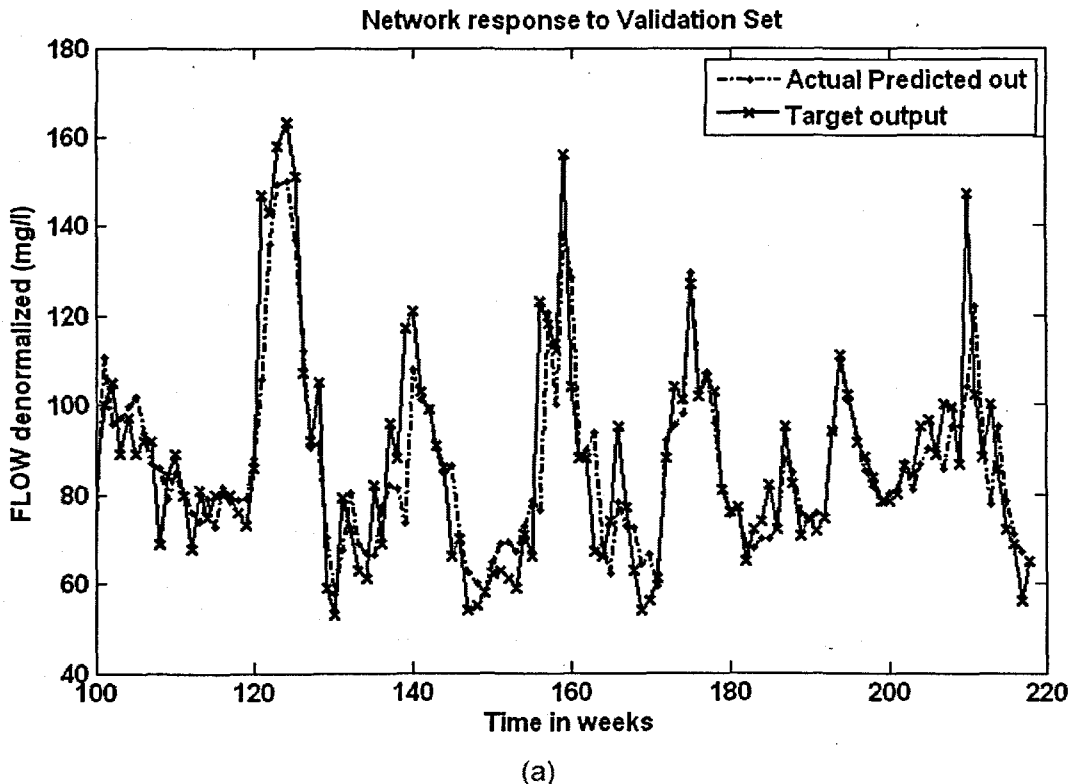
The inputs for Model 10 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), MONTH, Wind and the output is FLOW(k+1) one time step ahead. These are tabled below:

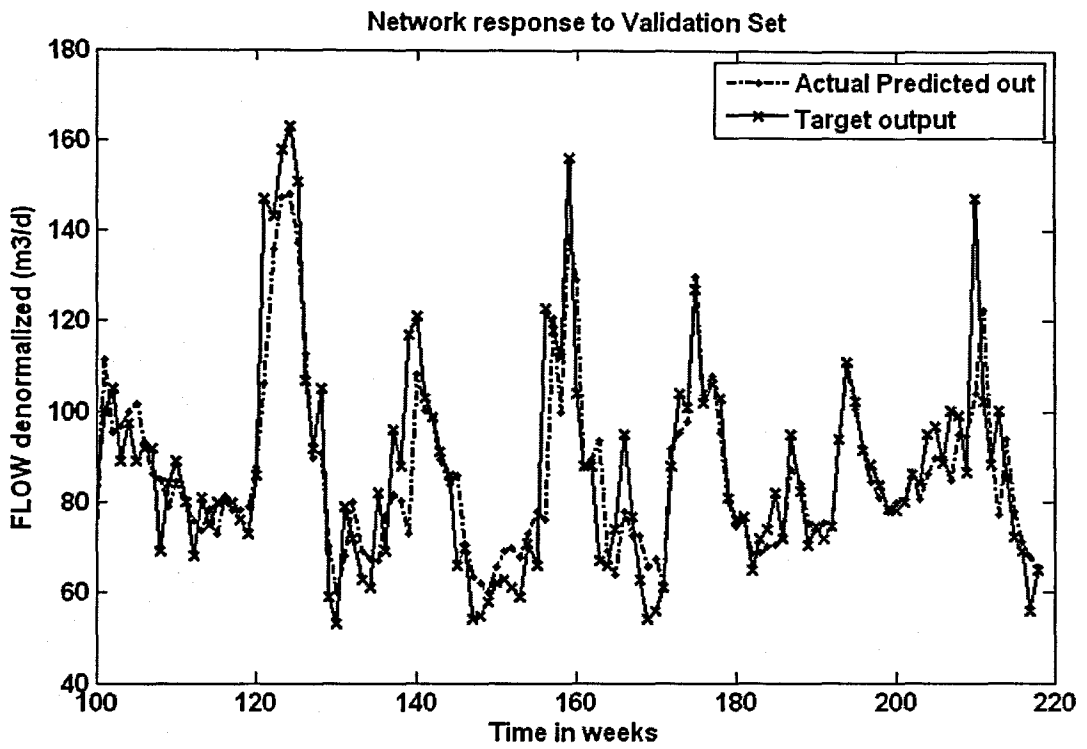
**Table 6. 141: Input variables for FLOW Model 10**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), Month, Wind	FLOW(k+1)

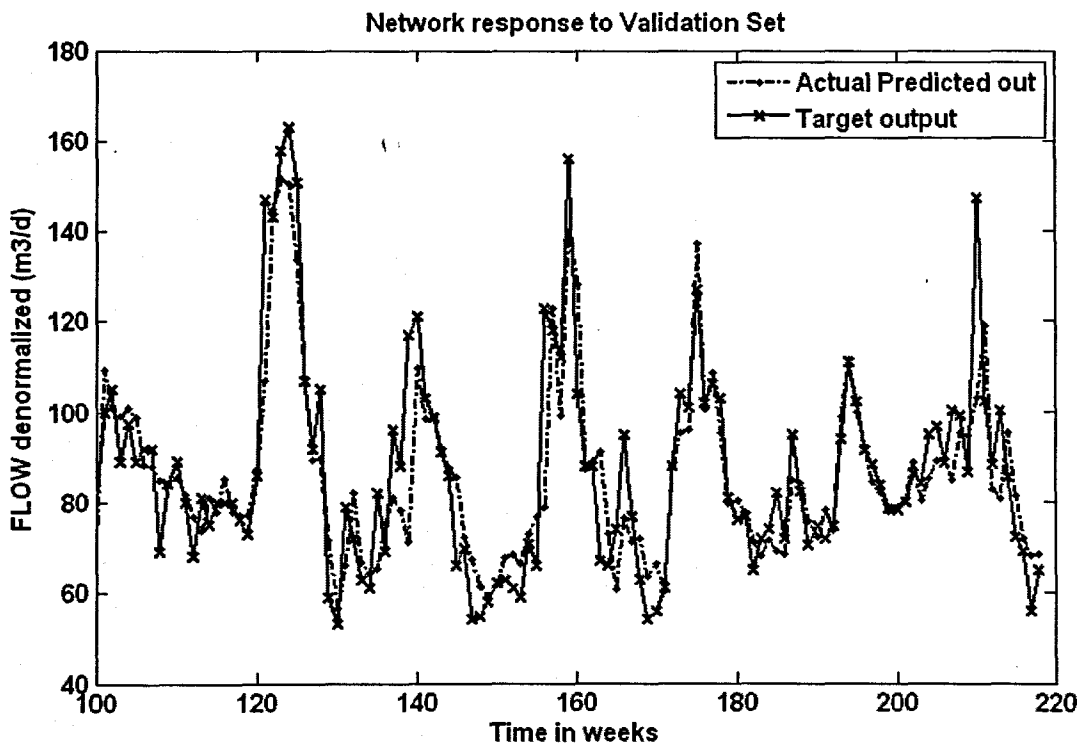


**Figure 6. 86: (a) A MLP Feed-forward neural network Model 10 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 10 with 16 hidden neurons, 16 epochs**





(b)



(c)

Figure 6. 87: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 10 with 1 hidden neuron and 500 epochs; (b) ERNN Model 10 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 10 with 27 hidden neurons, 27 epochs



**Table 6. 142: Results for the Feed-forward Neural Network Model 10 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err.	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.870	0.872	0.757	0.761	-0.164	8.485
1	5	trainscg	tansig	purelin	500	MSE	0.0034	0.6	0.5	0.875	0.865	0.766	0.748	-0.052	8.830
1	10	trainscg	tansig	purelin	500	MSE	0.0031	0.6	0.5	0.886	0.856	0.785	0.733	-0.067	9.087
1	1	trainscg	tansig	purelin	634	MSE	0.0035	0.6	0.5	0.870	0.871	0.758	0.758	-0.263	8.651
1	5	trainscg	tansig	purelin	1000	MSE	0.0031	0.6	0.5	0.886	0.859	0.786	0.739	-0.569	8.715
1	10	trainscg	tansig	purelin	1000	MSE	0.0028	0.6	0.5	0.899	0.846	0.808	0.716	0.001	9.510

**Table 6. 143: Results for the Elman Recurrent Neural Network Model 10 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err.	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	436	MSE	0.0035	0.01	0.5	0.870	0.871	0.758	0.758	-0.263	8.652
1	5	trainscg	tansig	purelin	500	MSE	0.0033	0.01	0.5	0.878	0.856	0.770	0.734	-0.117	9.205
1	10	trainscg	tansig	purelin	500	MSE	0.0032	0.01	0.5	0.884	0.862	0.781	0.744	0.187	9.046
1	1	trainscg	tansig	purelin	544	MSE	0.0035	0.01	0.5	0.870	0.871	0.758	0.758	-0.263	8.651
1	5	trainscg	tansig	purelin	1000	MSE	0.0030	0.01	0.5	0.890	0.854	0.791	0.730	-0.511	9.247
1	10	trainscg	tansig	purelin	1000	MSE	0.0030	0.01	0.5	0.889	0.854	0.791	0.729	0.001	9.363

**Table 6. 144: Results for the Radial Basis Function Neural Network Model 10 with FLOW as output.**

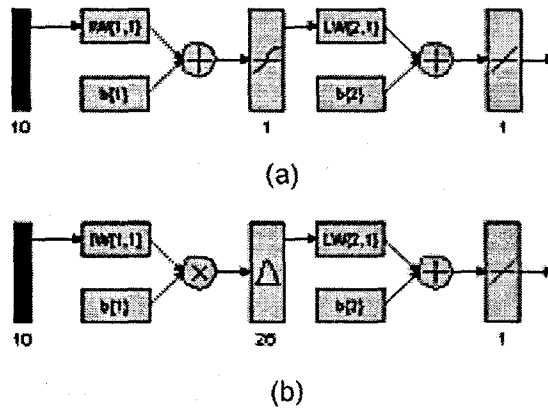
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	21	21	SSE	1.459	1.5	1	0.878	0.865	0.770	0.748	-0.069	8.910

### 6.4.11 FLOW Neural Network MODEL 11

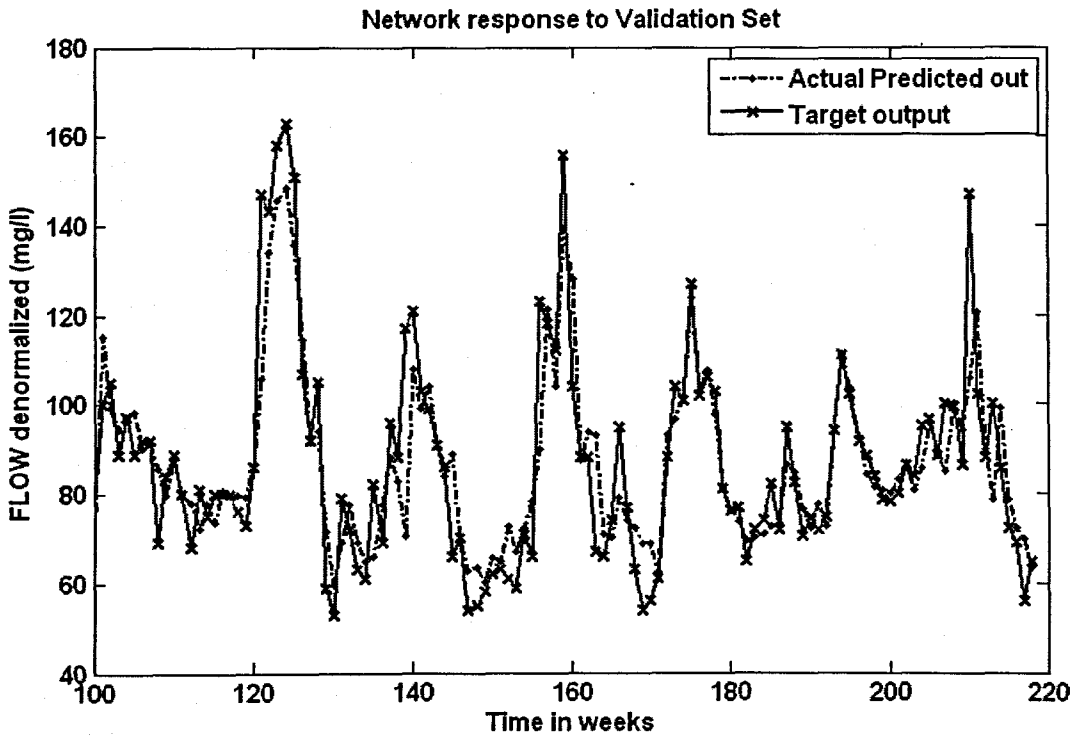
The inputs for Model 11 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), MONTH, Minimum Temperature, Rain, and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 145: Input variables for FLOW Model 11**

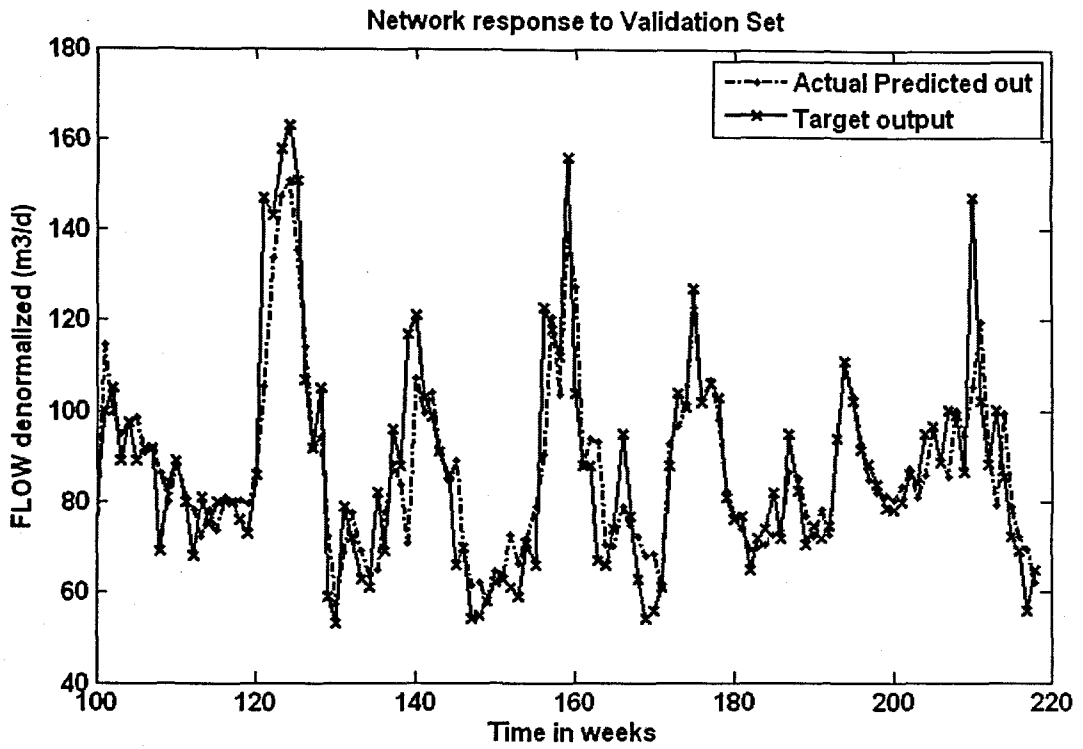
Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), Month, Minimum Temperature, Rain	FLOW(k+1)



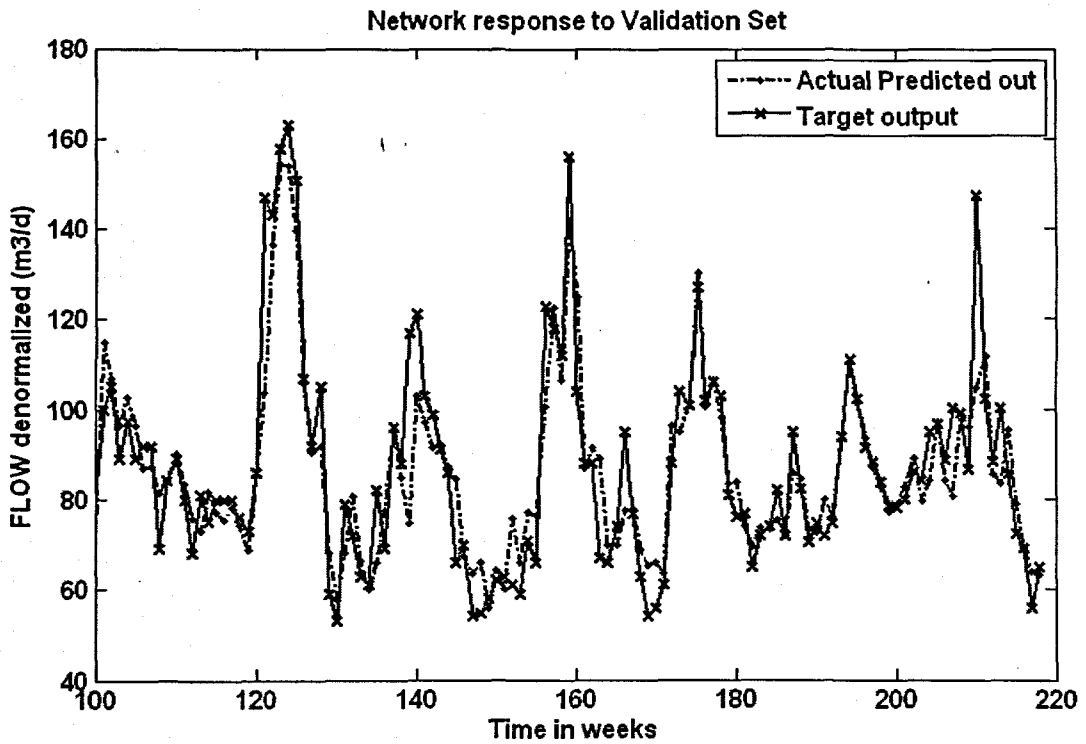
**Figure 6. 88: (a) A MLP Feed-forward neural network Model 11 with 10 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 11 with 26 hidden neurons, 26 epochs**



(a)



(b)



(c)

Figure 6. 89: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 11 with 1 hidden neuron and 500 epochs; (b) ERNN Model 11 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 11 with 26 hidden neurons, 26 epochs

**Table 6. 146: Results for the Feed-forward Neural Network Model 11 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.873	0.881	0.762	0.777	-0.363	8.345
1	5	trainscg	tansig	purelin	500	MSE	0.0031	0.6	0.5	0.885	0.870	0.784	0.756	-0.490	8.825
1	10	trainscg	tansig	purelin	500	MSE	0.0031	0.6	0.5	0.887	0.878	0.787	0.771	-0.043	8.303
1	1	trainscg	tansig	purelin	517	MSE	0.0035	0.6	0.5	0.873	0.881	0.762	0.777	-0.363	8.345
1	5	trainscg	tansig	purelin	1000	MSE	0.0031	0.6	0.5	0.886	0.874	0.784	0.763	-0.444	8.509
1	10	trainscg	tansig	purelin	1000	MSE	0.0028	0.6	0.5	0.899	0.848	0.808	0.720	-0.643	9.251

**Table 6. 147: Results for the Elman Recurrent Neural Network Model 11 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	432	MSE	0.0035	0.01	0.5	0.873	0.881	0.762	0.777	-0.363	8.345
1	5	trainscg	tansig	purelin	500	MSE	0.0032	0.01	0.5	0.884	0.877	0.781	0.768	0.061	8.593
1	10	trainscg	tansig	purelin	500	MSE	0.0030	0.01	0.5	0.890	0.869	0.792	0.756	-0.466	8.496
1	1	trainscg	tansig	purelin	1000	MSE	0.0035	0.01	0.5	0.872	0.884	0.761	0.782	-0.296	8.177
1	5	trainscg	tansig	purelin	1000	MSE	0.0030	0.01	0.5	0.891	0.853	0.793	0.727	-0.402	8.734
1	10	trainscg	tansig	purelin	1000	MSE	0.0029	0.01	0.5	0.893	0.861	0.798	0.741	-0.278	8.876

**Table 6. 148: Results for the Radial Basis Function Neural Network Model 11 with FLOW as output.**

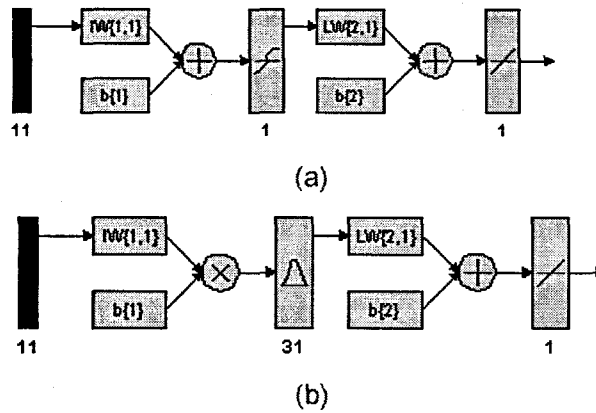
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	26	26	SSE	1.379	1.4	0.9	0.885	0.882	0.783	0.779	-0.521	8.366

### 6.4.12 FLOW Neural Network MODEL 12

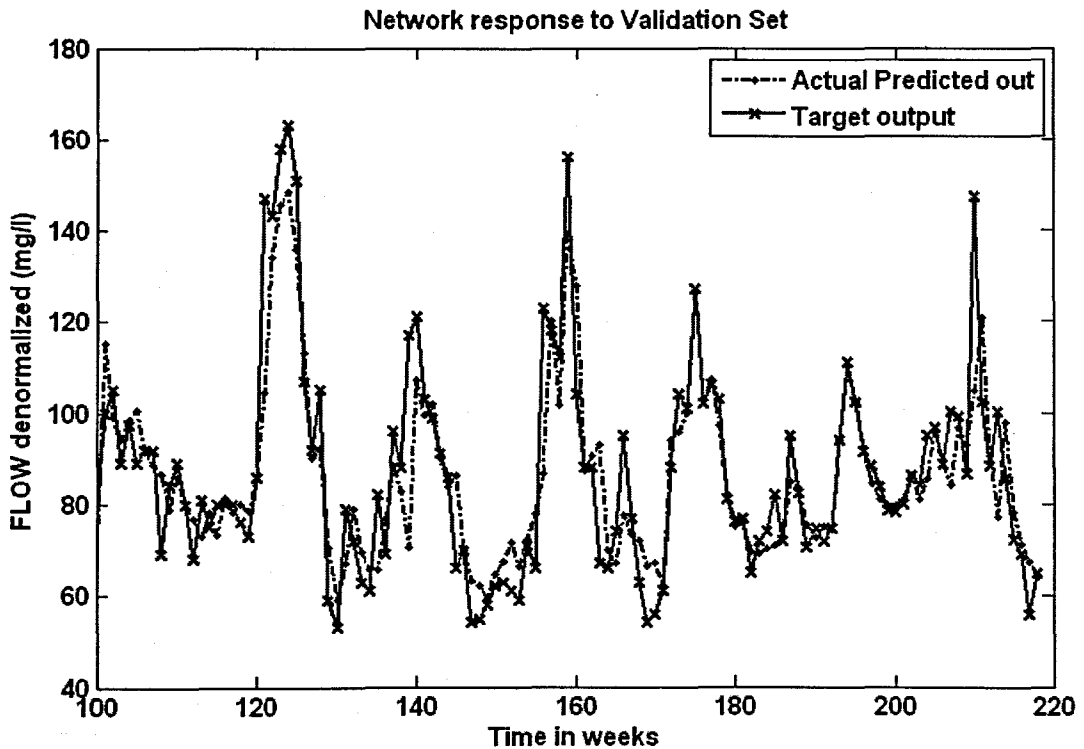
The inputs for Model 12 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), MONTH, Minimum Temperature, Rain, Wind, and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 149: Input variables for FLOW Model 12**

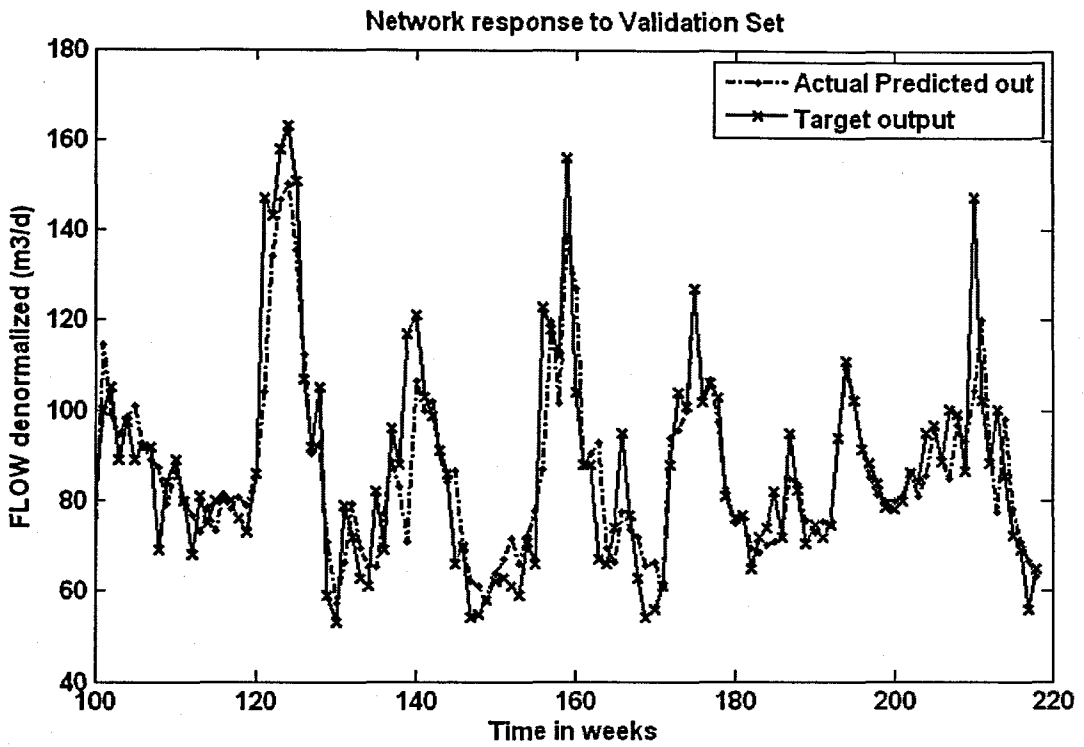
Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), COD(k), TKN(k), Month, Minimum Temperature, Rain, Wind	FLOW(k+1)



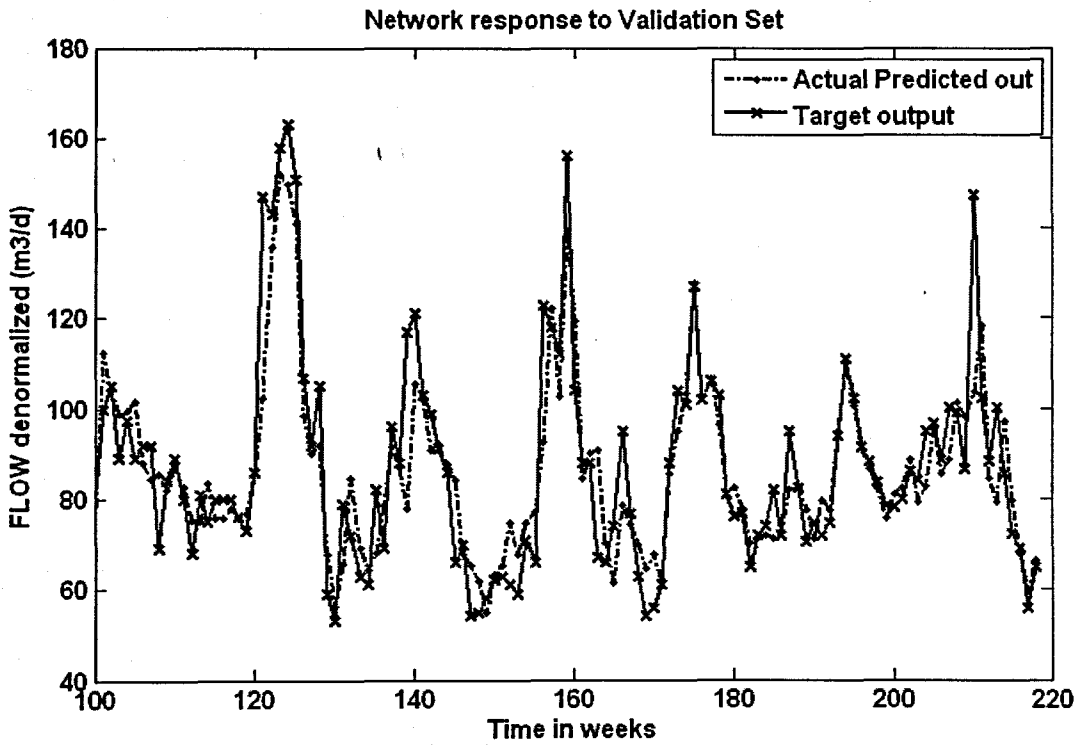
**Figure 6. 90: (a) A MLP Feed-forward neural network Model 12 with 11 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 12 with 31 hidden neurons, 31 epochs**



(a)



(b)



(c)

Figure 6. 91: The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 12 with 1 hidden neuron and 1000 epochs; (b) ERNN Model 12 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 12 with 31 hidden neurons, 31 epochs

**Table 6. 150: Results for the Feed-forward Neural Network Model 12 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0034	0.6	0.5	0.875	0.874	0.766	0.764	-0.298	8.566
1	5	trainscg	tansig	purelin	500	MSE	0.0032	0.6	0.5	0.885	0.871	0.782	0.759	0.208	8.895
1	10	trainscg	tansig	purelin	500	MSE	0.0032	0.6	0.5	0.884	0.840	0.781	0.706	0.158	9.128
1	1	trainscg	tansig	purelin	722	MSE	0.0034	0.6	0.5	0.875	0.874	0.766	0.764	-0.287	8.542
1	5	trainscg	tansig	purelin	1000	MSE	0.0030	0.6	0.5	0.891	0.861	0.793	0.741	-0.412	9.035
1	10	trainscg	tansig	purelin	1000	MSE	0.0026	0.6	0.5	0.908	0.801	0.824	0.641	-0.601	10.791

**Table 6. 151: Results for the Elman Recurrent Neural Network Model 12 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0034	0.01	0.5	0.875	0.874	0.766	0.764	-0.291	8.555
1	5	trainscg	tansig	purelin	500	MSE	0.0032	0.01	0.5	0.884	0.860	0.781	0.739	-0.054	8.987
1	10	trainscg	tansig	purelin	500	MSE	0.0030	0.01	0.5	0.890	0.860	0.791	0.740	-0.131	8.906
1	1	trainscg	tansig	purelin	1000	MSE	0.0034	0.01	0.5	0.875	0.876	0.766	0.767	-0.228	8.419
1	5	trainscg	tansig	purelin	1000	MSE	0.0031	0.01	0.5	0.887	0.852	0.786	0.726	-0.328	9.029
1	10	trainscg	tansig	purelin	1000	MSE	0.0027	0.01	0.5	0.901	0.828	0.812	0.686	-0.216	9.958

**Table 6. 152: Results for the Radial Basis Function Neural Network Model 12 with FLOW as output.**

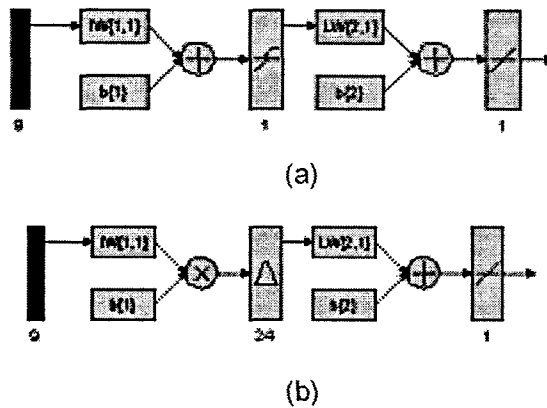
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	31	31	SSE	1.392	1.4	0.8	0.884	0.874	0.781	0.765	-0.440	8.616

### 6.4.13 FLOW Neural Network MODEL 13

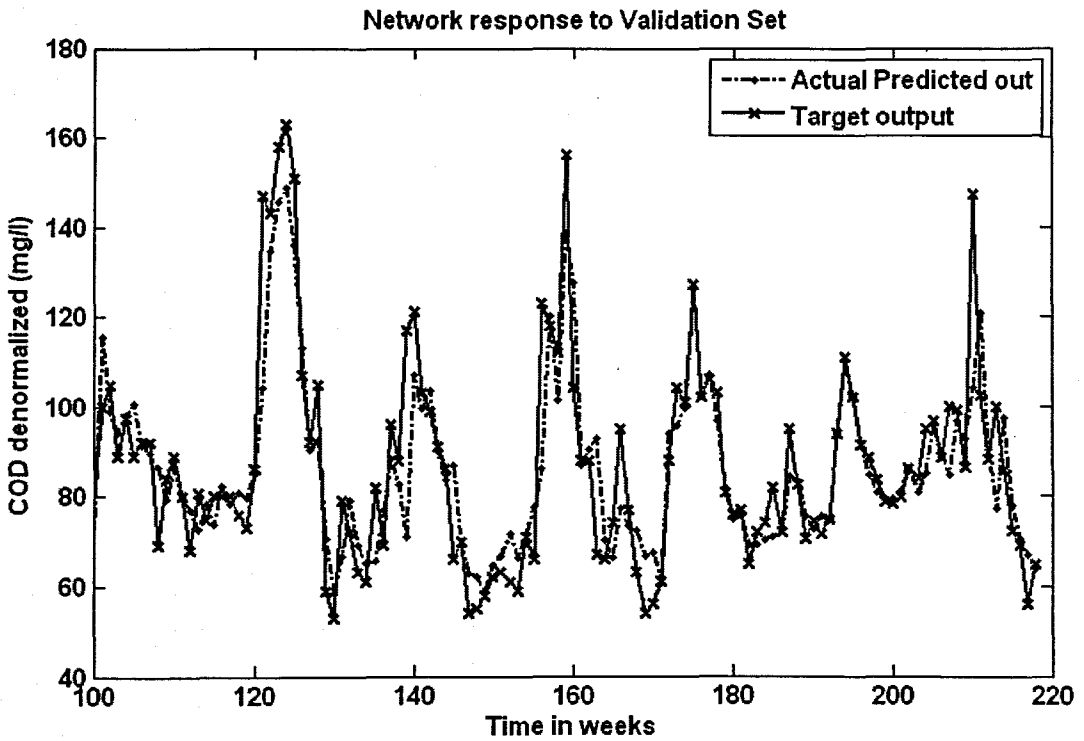
The inputs for Model 13 are FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), MONTH, Minimum Temperature, Rain, Wind, and the output is FLOW(k+1) one time step ahead. These are tabled below:

**Table 6. 153: Input variables for FLOW Model 13**

Inputs	Output
FLOW(k), FLOW(k-1), FLOW(k-2), FLOW(k-3), Month, Minimum Temperature, Rain, Wind	FLOW(k+1)

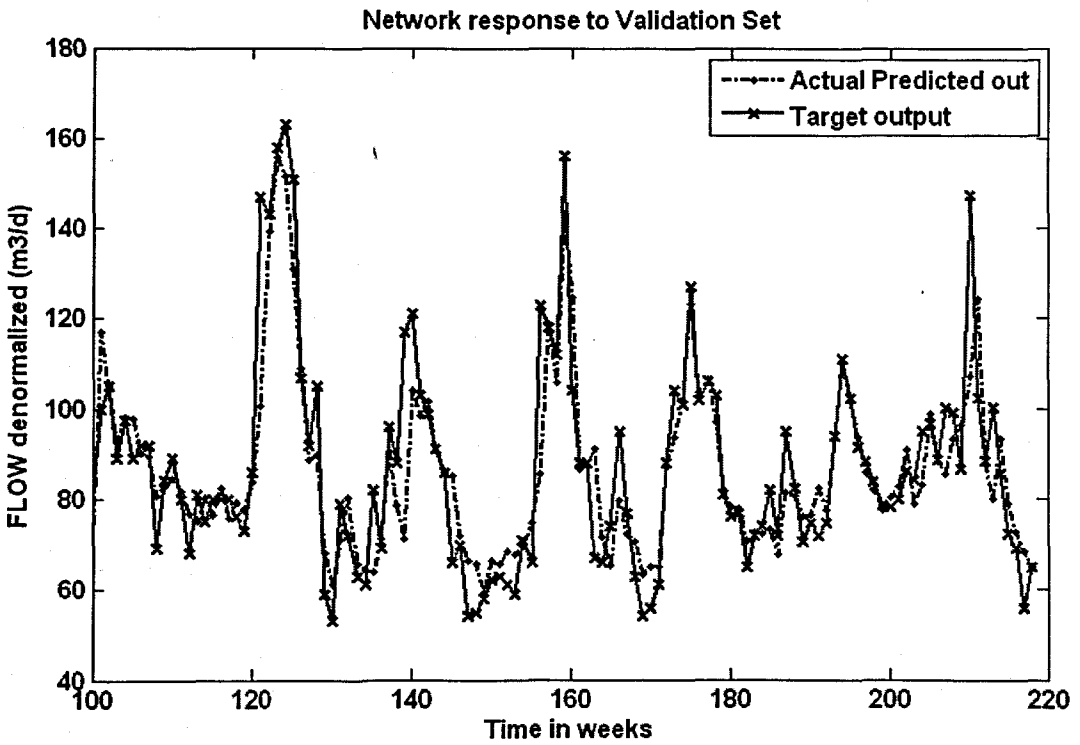
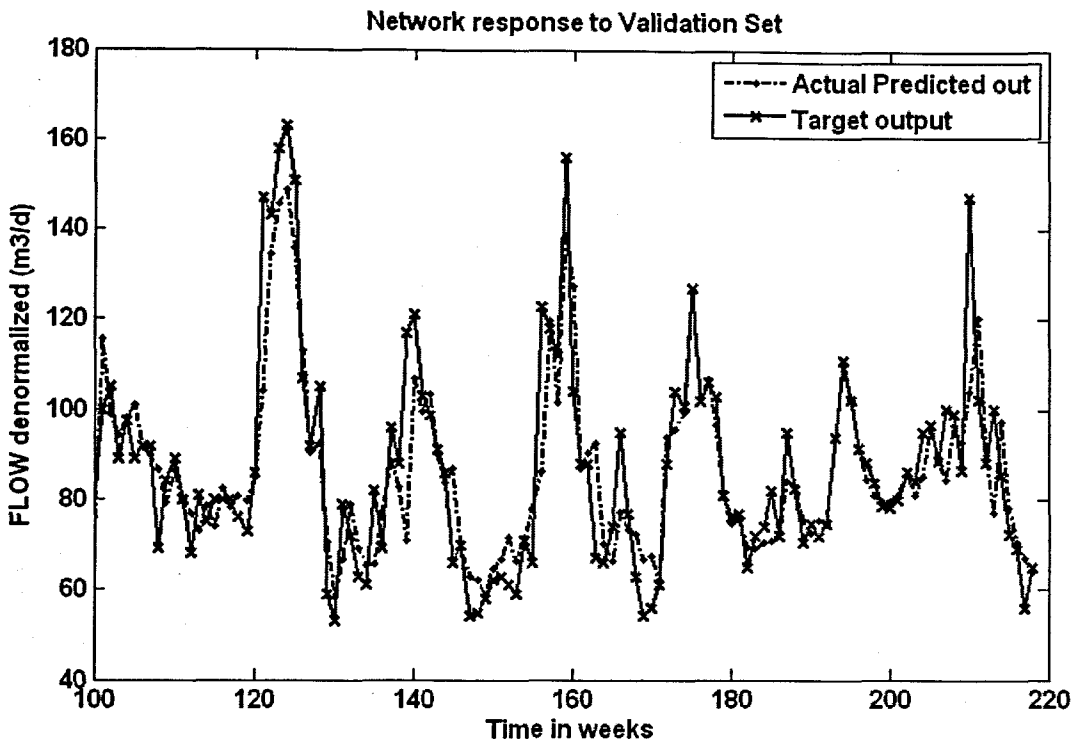


**Figure 6. 92: (a) A MLP Feed-forward neural network Model 13 with 9 inputs, 1 hidden layer, 1 hidden neuron and 1 output neuron; and (b) Block Diagram for the FLOW RBFNN Model 13 with 24 hidden neurons, 24 epochs**



(a)





**Figure 6. 93:** The NN response to the application of the test (validation) data set for FLOW: (a) MLP FFNN Model 13 with 1 hidden neuron and 500 epochs; (b) ERNN Model 13 with 1 hidden neuron, 1000 epochs; and (c) RBFNN Model 13 with 24 hidden neurons, 24 epochs

**Table 6. 154: Results for the Feed-forward Neural Network Model 13 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	247	MSE	0.0034	0.6	0.5	0.875	0.874	0.766	0.764	-0.304	8.568
1	5	trainscg	tansig	purelin	500	MSE	0.0032	0.6	0.5	0.882	0.860	0.778	0.740	-0.080	8.897
1	10	trainscg	tansig	purelin	500	MSE	0.0031	0.6	0.5	0.888	0.852	0.788	0.726	-0.195	8.720
1	1	trainscg	tansig	purelin	688	MSE	0.0034	0.6	0.5	0.875	0.874	0.766	0.764	-0.304	8.570
1	5	trainscg	tansig	purelin	1000	MSE	0.0029	0.6	0.5	0.895	0.819	0.801	0.671	0.299	9.867
1	10	trainscg	tansig	purelin	1000	MSE	0.0029	0.6	0.5	0.896	0.833	0.804	0.694	-0.583	9.777

**Table 6. 155: Results for the Elman Recurrent Neural Network Model 13 with FLOW as output.**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	1	trainscg	tansig	purelin	500	MSE	0.0034	0.01	0.5	0.875	0.873	0.766	0.763	-0.315	8.606
1	5	trainscg	tansig	purelin	500	MSE	0.0031	0.01	0.5	0.887	0.849	0.787	0.721	-0.154	9.075
1	10	trainscg	tansig	purelin	500	MSE	0.0031	0.01	0.5	0.887	0.866	0.788	0.751	0.058	8.843
1	1	trainscg	tansig	purelin	625	MSE	0.0034	0.01	0.5	0.875	0.874	0.766	0.764	-0.305	8.570
1	5	trainscg	tansig	purelin	1000	MSE	0.0030	0.01	0.5	0.892	0.840	0.795	0.706	-0.018	9.064
1	10	trainscg	tansig	purelin	1000	MSE	0.0027	0.01	0.5	0.903	0.844	0.816	0.712	-0.279	9.369

**Table 6. 156: Results for the Radial Basis Function Neural Network Model 13 with FLOW as output.**

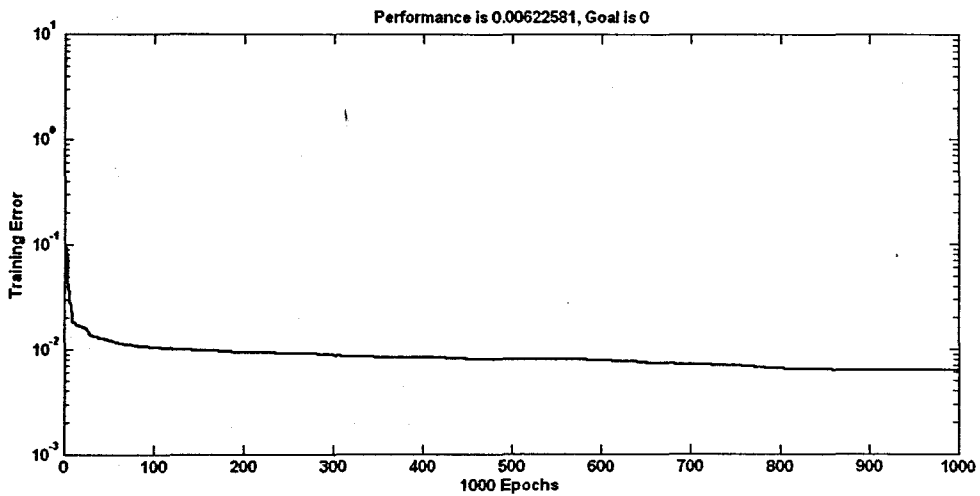
Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	24	24	SSE	1.400	1.4	0.9	0.883	0.871	0.780	0.758	-0.422	8.519

## 6.5 Investigation of the variation of the structure of the NNs and the training process

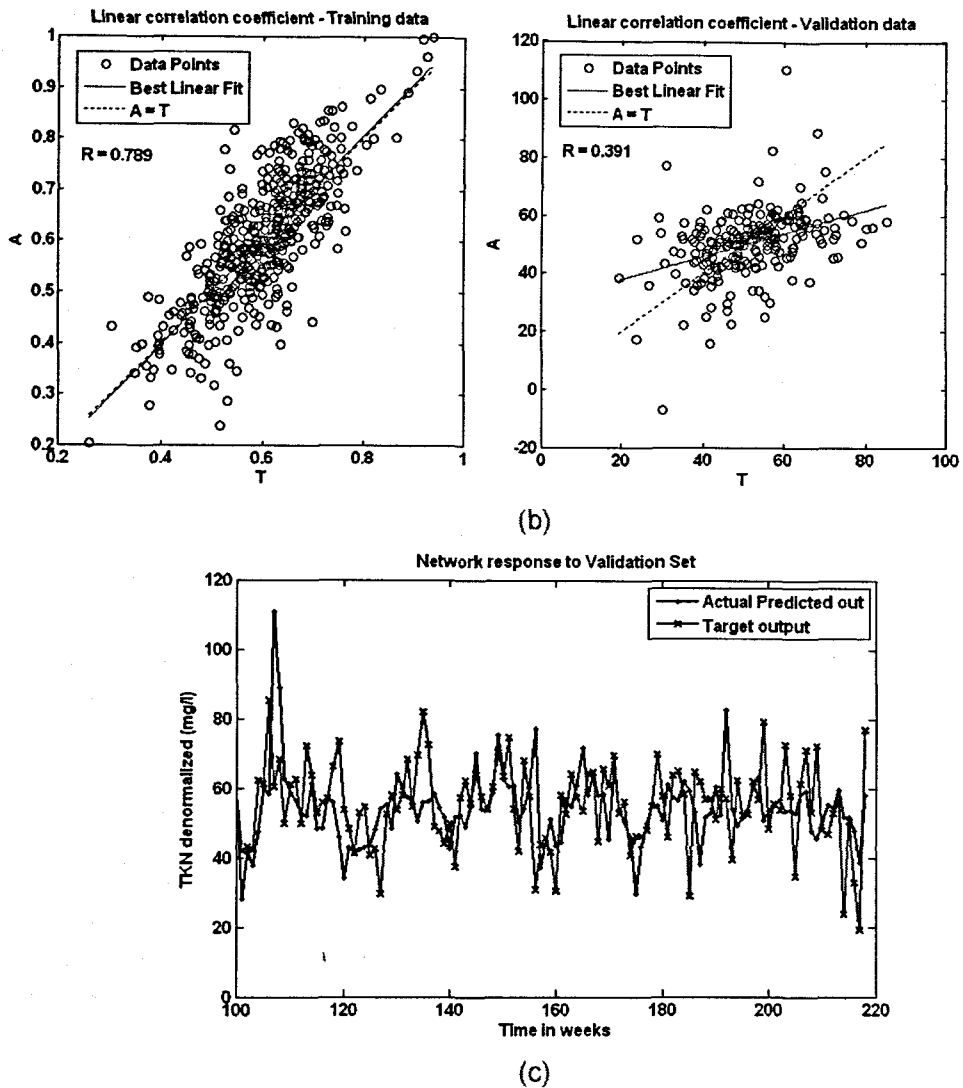
The following section investigates the effect that varying the structure of the NN by increasing the number of hidden layers, neurons and epochs, has on the predictive and ultimately generalisation performance of the network.

### 6.5.1 The effect of increasing the number of hidden neurons

The first set of results in section 6.5.1 examines effect of increasing the number of hidden neurons. The type of NN is the ERNN for prediction of TKN. The inputs are  $TKN(k)$ ,  $TKN(k-1)$ ,  $TKN(k-2)$ ,  $TKN(k-3)$ ,  $COD(k)$ ,  $FLOW(k)$ ,  $MONTH$ , Minimum Temperature, Rain, Wind, and the output is  $TKN(k+1)$  one time step ahead. The epochs are kept at 1000 while the neurons are increased to 40. Figure 6.94 displays the: (a) training error versus the epochs, (b) regression analysis with the linear correlation coefficients for the training (left) and test (right) phases, and (c) NN response to the application of the test (validation) set. Table 6.157 summarises the results by comparing it to the previous case where the neurons are varied from 1 to 5 to 10 (lower part of the table).



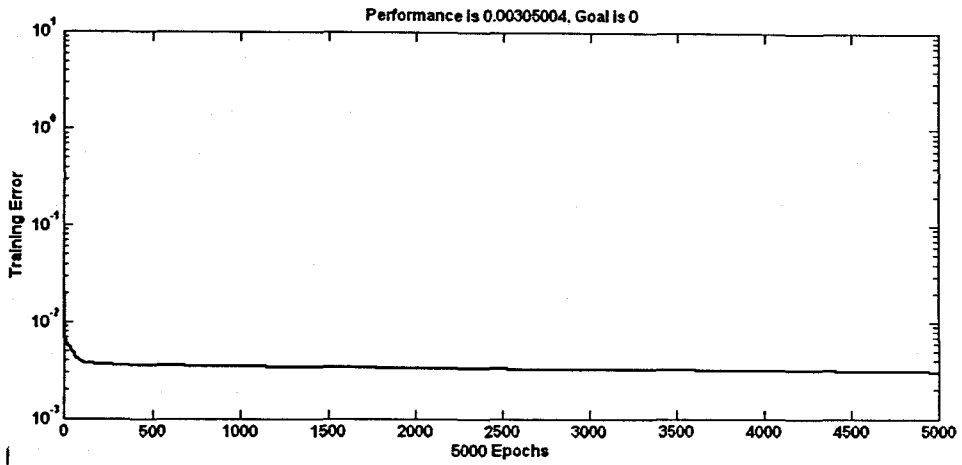
(a)



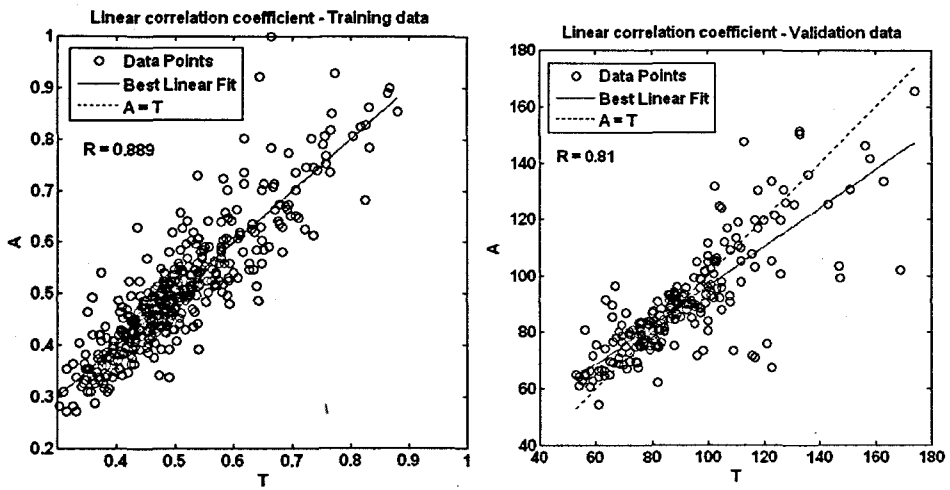
**Figure 6.94:** The NN response to the application of the test (validation) data set for TKN ERNN Model 12 with 40 hidden neurons, 1000 epochs, where the number of neurons is increased: (a) Training phase; (b) Regression coefficients for training (left) and test phases (right); and (c) NN response to the test set

### 6.5.2 The effect of increasing the number of epochs

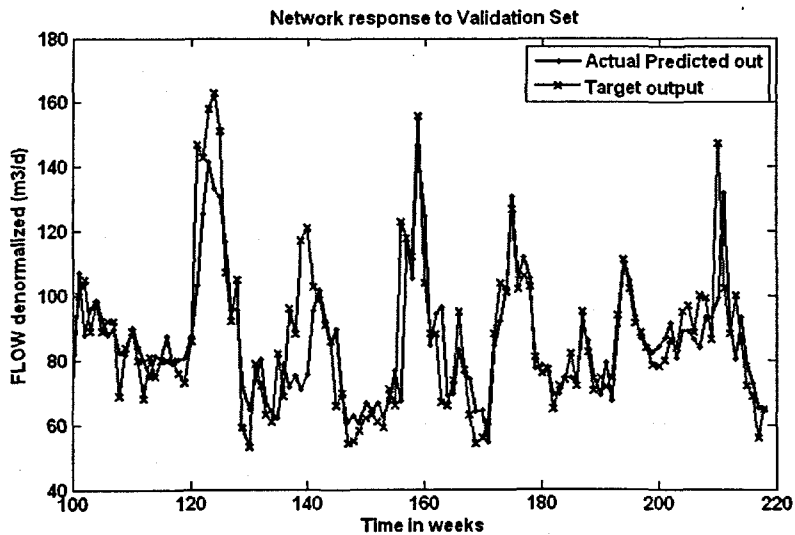
This section examines the effect that increasing the number of training iterations (epochs) has on the network performance. The neural network is the MLP with  $FLOW(k)$ ,  $FLOW(k-1)$ ,  $FLOW(k-2)$ ,  $FLOW(k-3)$ ,  $COD(k)$ ,  $TKN(k)$ , as inputs and the output is  $FLOW(k+1)$  one time step ahead. The hidden neurons are kept at 10 while the epochs are increased to 5000. Figure 6.95 displays the: (a) training error versus the epochs, (b) regression analysis with the linear correlation coefficients for the training (left) and test (right) phases, and (c) NN response to the application of the test (validation) set. The results are summarised in Table 6.158 that provides a comparison of the previously obtained results for this same model.



(a)



(b)

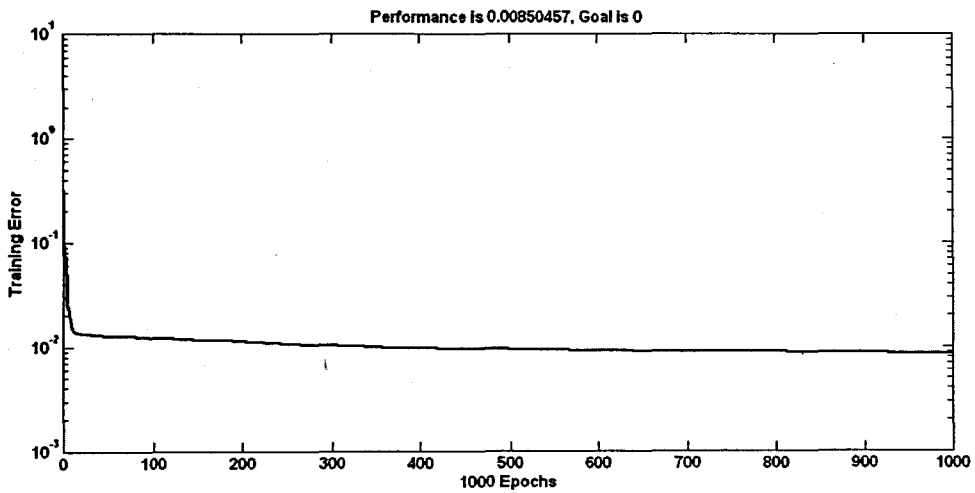


(c)

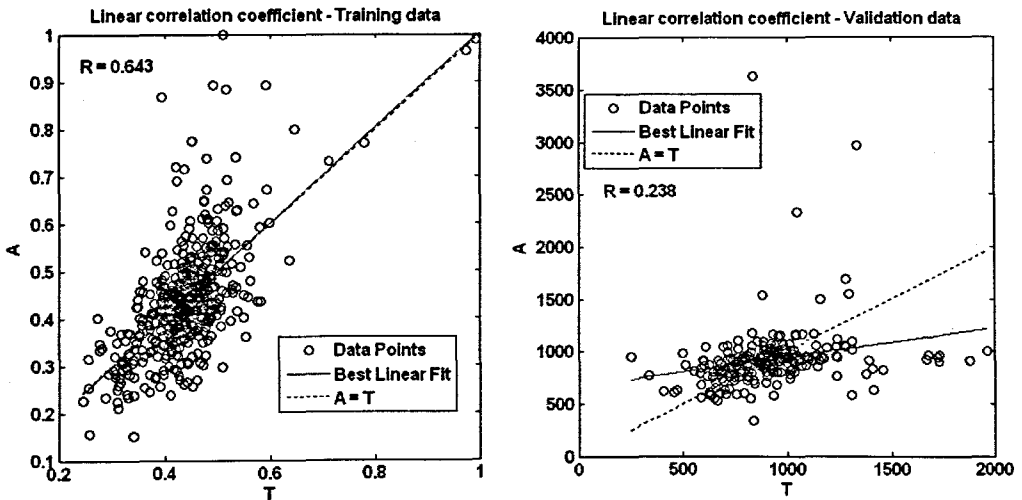
Figure 6. 95: The NN response to the application of the test (validation) data set for FLOW MLP FFNN Model 6 with 10 hidden neurons, 5000 epochs, where the epochs is increased: (a) Training; (b) Regression coefficients for training (left) and test phases (right); and (c) NN response to the test set

### 6.5.3 The effect of increasing the number of hidden layers

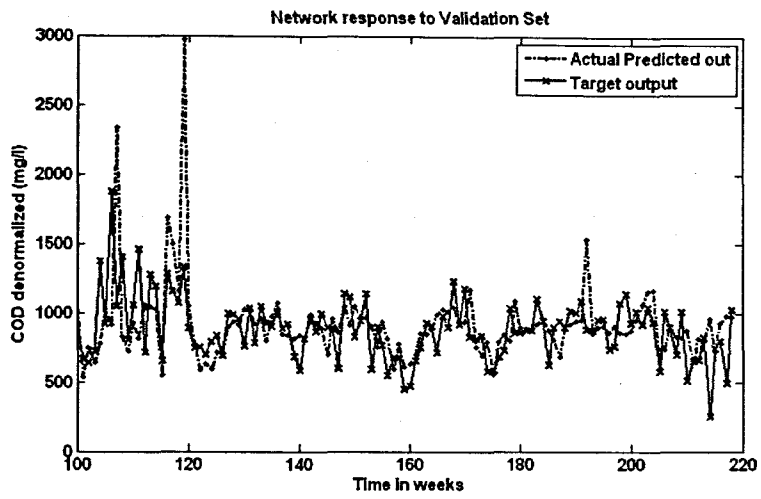
This section examines what effect increasing the number of hidden layers to two has on the network performance. The considered network is the MLP with inputs COD(k), COD(k-1), COD(k-1) COD(k1), TKN(k), FLOW(k), MONTH, Minimum Temperature, Rain, Wind, and the output is COD(k+1) one time step ahead. The hidden neurons for the first and second hidden layers are set to 5. The network is trained for 1000 epochs. Figure 6.96 displays the: (a) training error versus the epochs, (b) regression analysis with the linear correlation coefficients for the training (left) and test (right) phases, and (c) NN response to the application of the test (validation) set. The results are tabled in Table 6.159 where the networks with one hidden layer are compared with the network having two hidden layers.



(a)



(b)

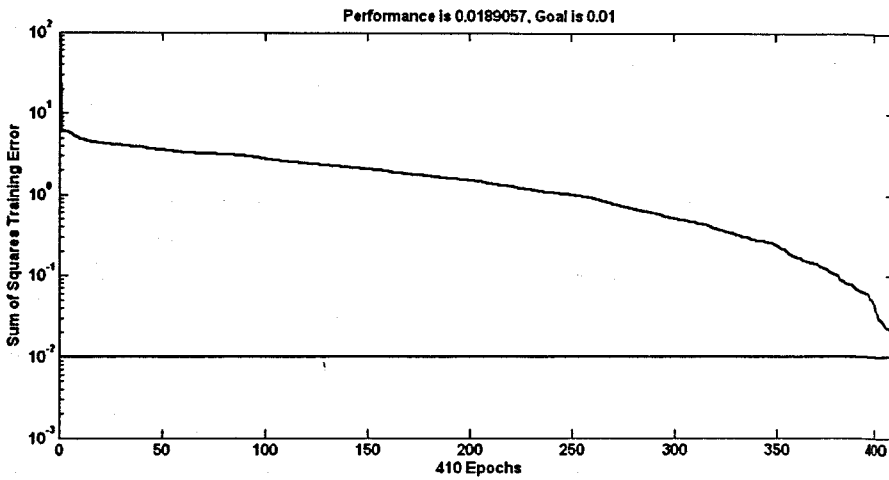


(c)

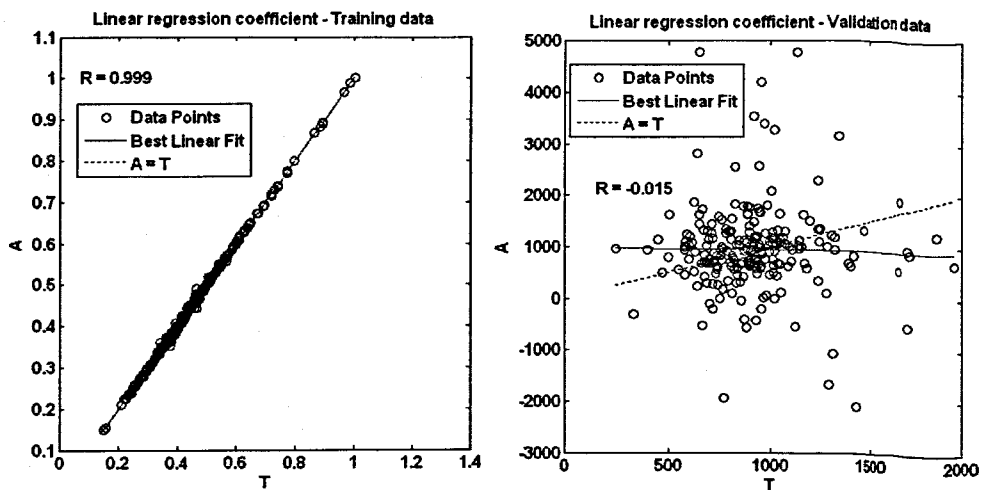
**Figure 6. 96:** The NN response to the application of the test (validation) data set for COD MLP FFNN Model 12 with 5\_5 hidden neurons, 1000 epochs, where there are 2 hidden layers: (a) Training; (b) Regression coefficients for training (left) and test phases (right); and (c) NN response to the test set

### 6.5.4 The effect of training to convergence

This section examines what the effect is when the network is trained until the error converges to a minimum. The considered network is the RBF with inputs COD(k), COD(k-1), COD(k-2) COD(k-3), TKN(k), FLOW(k), MONTH, and the output is COD(k+1) one time step ahead. The spread constant is set to a value of 0.6 and the error goal is set at 0.01. Figure 6.97 displays the: (a) training error versus the epochs, (b) regression analysis with the linear correlation coefficients for the training (left) and test (right) phases, and (c) NN response to the application of the test (validation) set. The results are tabled in Table 6.160 where it is compared to the previously considered case.

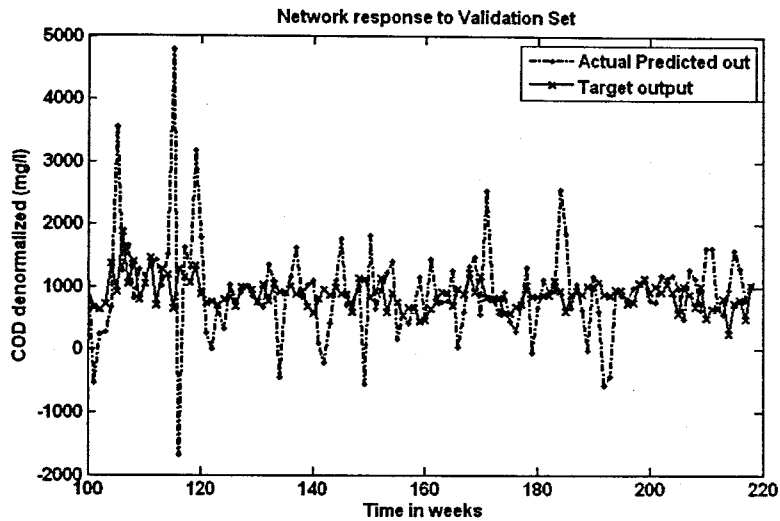


(a)



(b)





(c)

**Figure 6. 97:** The NN response to the application of the test (validation) data set for COD RBFNN Model 7 with 414 hidden neurons, 414 epochs, where the network is trained to convergence: (a) Training; (b) Performance for training and test sets; (c) NN response

### 6.5.5 The training time when using different training algorithms

The different training algorithms execute at different speeds as presented in Chapter 5. The training algorithms are changed from initially the Levenberg-Marquardt to the scaled conjugate gradient algorithm. The NN is the MLP for COD with inputs COD(k), COD(k-1), COD(k-2) COD(k-3), MONTH, Minimum Temperature, Rain, Wind, and the output is COD(k+1) one time step ahead. The times are measured with the *tic* and *toc* MATLAB functions. The number of hidden neurons is 10 and the network is trained for 1000 epochs.

The training execution time result for the scaled gradient is 8.9755 seconds. The time taken using the Levenberg-Marquardt is 25.5834 seconds. This shows the considerable difference in training speeds.

**Table 6. 157: Results for the ERNN Neural Network Model 12 with TKN as output, and where number of hidden neurons are increased to 40 compared to 1,5 and 10**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	40	trainscg	tansig	purelin	1000	MSE	0.0062	0.01	0.5	0.789	0.391	0.623	0.153	-1.321	17.687
1	1	trainscg	tansig	purelin	904	MSE	0.0106	0.01	0.5	0.597	0.504	0.356	0.254	-0.578	14.195
1	5	trainscg	tansig	purelin	1000	MSE	0.0087	0.01	0.5	0.686	0.465	0.470	0.216	-1.092	15.058
1	10	trainscg	tansig	purelin	1000	MSE	0.0086	0.01	0.5	0.690	0.502	0.477	0.252	-0.703	14.445

**Table 6. 158: Results for the MLP Neural Network Model 6 with FLOW as output, and where epochs are increased to 5000 compared to 500 and 1000**

Hid layers	Hid neurons	Train algorithm	Hidden activation	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
1	10	trainscg	tansig	purelin	5000	MSE	0.0031	0.6	0.5	0.889	0.810	0.791	0.657	-1.008	9.405
1	10	trainscg	tansig	purelin	500	MSE	0.0035	0.6	0.5	0.871	0.846	0.759	0.716	-0.903	9.285
1	10	trainscg	tansig	purelin	1000	MSE	0.0035	0.6	0.5	0.873	0.854	0.762	0.729	-0.204	8.955

**Table 6. 159: Results for the MLP Neural Network Model 12 with COD as output, and where the hidden layers are increased to two compared to one hidden layer**

Hid layers	Hid neurons	Hid neurons 2 layer	Train algorithm	Hidden activation1	Hidden activation2	Output activation	Epochs	Performance	Min train err	Learn rate	Momentum	r_train	r_test	R_sq_train	R_sq_test	Avg_test err	APE
2	5	5	trainscg	tansig	tansig	purelin	1000	MSE	0.0088	0.6	0.5	0.625	0.206	0.391	0.042	-27.912	21.857
1	1	N/A	trainscg	tansig	N/A	purelin	1000	MSE	0.0117	0.6	0.5	0.439	0.349	0.193	0.122	-23.780	19.307
1	5	N/A	trainscg	tansig	N/A	purelin	1000	MSE	0.0098	0.6	0.5	0.568	0.288	0.322	0.083	-36.529	20.437
1	10	N/A	trainscg	tansig	N/A	purelin	1000	MSE	0.0085	0.6	0.5	0.643	0.172	0.414	0.030	-34.419	22.906

**Table 6. 160: Results for the Radial Basis Function Neural Network Model 7 with COD as output, and where the network is trained until the error converges to a minimum compared with the previous considered case**

Hidden layers	Hidden activation	Output activation	Epochs	Hidden neurons	Performance Function	Min train error	Error Goal	Spread	r_train	r_test	R_sq_train	R_sq_test	Avg_test error	APE
1	radbas	purelin	414	414	SSE	0.008	0.01	0.6	0.999	-0.015	0.999	0.000	30.284	41.327
1	radbas	purelin	19	19	SSE	4.587	4.6	0.8	0.523	0.351	0.274	0.123	-31.742	20.257

## 6.6 Discussion of results

The three topologies examined included the MLP, ERNN and the RBFNN. Generally speaking the results for particularly the multilayer perceptron and the recurrent NN were very close, with the radial basis function NN not very far off. Thirteen models are presented where the inputs are varied to include combinations of the influent variable, delayed influent variable, other influent variables, environmental variables of temperature, rainfall and wind speed, and the month of the year. The physical structure for all the considered NNs in this section only has ONE hidden layer. The hidden activation functions for both the MLP and the ERNN are the hyperbolic tangent, and the output has a linear activation function. The RBF NN has a Gaussian radial basis function for the hidden layer activation function and a linear activation at the output. The epochs are increased for the MLP and the ERNN from 500 to 1000 epochs. The number of neurons in the hidden layer are increased from 1, 5 up to 10 neurons for the both the MLP and ERNN. The parameters for the RBF that are varied are the sum of squares error goal and the spread of the Gaussian function. The only model where the entire set of figures are presented is the case for Model 1. However the plots and all software algorithms for all other models are available for examination on the enclosed CD. The plots for the instantaneous error indicate the difference between the NN output and the desired target output for the entire test data set.

### 6.6.1 COD NN results

The first set of results is for the COD NN and is presented in section 6.2. The input combinations for COD are presented in Table 5.1. The figures for COD Model 1 are:

- Figure 6.2 - the training versus number of epochs plots for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.3 – the scatterplot for the NN response to the training data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.4 - the scatterplot for the NN response to the test (validation) data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.5 – the network performance showing the linear correlation coefficient during the training phase (left) and the test phase (right) for the (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.6 – the instantaneous error showing the difference between the NN output and the desired target output for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.7 – the NN response to the application of the test (validation) data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.

The results for the COD neural networks show that although the training error (mean squared error) is a minimum (0.01 on average), the NNs are not able to respond well to the input training data. Hence the generalisation performance of the networks are not very good. The regression coefficient  $r$  for the validation phase upon application of the test set, ranges from 0.2 till 0.4 on average. The coefficient of determination  $r^2$

for the COD NN ranges from 0.5% to 0.19% on average. The absolute percentage error for the difference between the desired target output and the NN output range between 19% and 22% on average. These figures indicate that the NN for COD at the output is therefore not able to solve this particular input-output mapping problem.

The results could not be improved despite investigating additional networks where the neurons are gradually increased up to a maximum of 80, and the epochs are increased to a maximum of 10 000. The problem when the number of hidden neurons is increased is that the training error is minimised, and the correlation coefficients for the training phase increase, while the test phase correlation coefficient decreases. The problem with increasing the number of epochs, is that the network eventually converges, but this does not mean that the NN has learned the training data. This is what is meant by the trade-off between generalization (performance) and training to convergence. This is best illustrated in Figure 6.97 where a RBF NN having 414 neurons is able to approximate the training data, but is unable to generalize. The correlation coefficient for the test phase is -0.015 and the coefficient of determination is 0. This indicates that the network has memorized the training data but cannot make any accurate predictions with the test data. The APE is unacceptably high at a value of 41.327%.

The best result of the 13 models considered for COD is the MLP FFNN, Model 13 with inputs COD(k), COD(k-1), COD(k-2) COD(k-3), MONTH, Minimum Temperature, Rain, Wind, and the output is COD(k+1) one time step ahead. The results are tabled in Table 6.50. The number of hidden neurons is 1, the training error 0.0117, the linear correlation coefficient is 0.4 for the test phase, and the APE is 19.119%. This result shows that the NNs are unable to solve this particular input-output mapping problem.

### 6.6.2 TKN NN results

The second set of results is for the TKN NN and is presented in section 6.3. The figures for TKN Model 1 are:

- Figure 6.33 - the training versus number of epochs plots for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.34 – the scatterplot for the NN response to the training data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.35 - the scatterplot for the NN response to the test (validation) data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.36 – the network performance showing the linear correlation coefficient during the training phase (left) and the test phase (right) for the (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.37 – the instantaneous error showing the difference between the NN output and the desired target output for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.

- Figure 6.38 – the NN response to the application of the test (validation) data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.

The results for Model 3 in Table 6.62 are interesting as the APE falls for the MLP is – 0.547 and the regression coefficient is 0.588 for the test phase. This indicates that this model has the best result throughout. However all other results are not too far off. The regression coefficient ranges from 0.38 up to a maximum of 0.588. The determination coefficient ranges from 0.241 to 0.346. The APE ranges from 15.290% in Model 1 to 13.457% in Model 9. For many of the models the results for the MLP and the ERNN were very similar.

The NNs for TKN performed better than the one for COD when comparing the performance figures. However the NNs for TKN could not learn the training data either although the training error reached an average value of 0.01, therefore the generalization performance of the networks are not very good either.

### 6.6.3 FLOW NN results

The third set of results examined the NNs with the flow rate as the influent disturbance at the NN output. These results are presented in section 6.4. The figures for FLOW Model 1 are:

- Figure 6.64 - the training versus number of epochs plots for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.65 – the scatterplot for the NN response to the training data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.66 - the scatterplot for the NN response to the test (validation) data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.67 – the network performance showing the linear correlation coefficient during the training phase (left) and the test phase (right) for the (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.68 – the instantaneous error showing the difference between the NN output and the desired target output for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.
- Figure 6.69 – the NN response to the application of the test (validation) data set for the, (a) MLP FFNN, (b) ERNN and the (c) RBFNN.

These set of results are by far the best. In general all the types of NNs are able to solve this problem to a certain degree. The generalisation ability of the FLOW NN has increased considerably when compared to that of COD and TKN. The regression coefficients for Model 8 in Table 6.136, for the RBF are 0.889 and the APE at 8.150 and the determination coefficient is at 0.790. This result is the best of all the results for the flow rate NNs. However additional experimentation could not improve this result significantly. The range for the regression coefficient is between 0.846 and 0.889 for the test phase. The APE ranged from 8.150% in Model 8, to 10.791 in

Model 12. As is seen from the table, the training regression coefficient has increased drastically to an average value of 0.9. The coefficient for the test phase on average is also around 0.9. Although the results are not ideal, they are good, as the generalization performance of the network has improved considerably compared to that of the neural networks for COD and TKN.

#### 6.6.4 Variation of the NN structures results

For the first case in section 6.5.1 the hidden neurons are increased to a value of 40. When compared to that of the previous cases where the the neurons are varied from 1 to 5 to 10, it is obvious that increasing the number of hidden neurons:

- does not guarantee that the network will converge;
- does not guarantee that the generalisation ability of the network will improve; in fact it deteriorates.

The results in Table 6.157 for regression value for the training phase has increased from about 0.66 for the previous case to 0.789. However the regression value for the test (validation) phase has now decreased from an average value of 0.490 for the previous case, to 0.391. The APE has increased from an average value of about 14.566 for the previous case to 17.687. Therefore the neurons have to be increased gradually and a trade-off in terms of the number of hidden neurons versus the generalisation ability of the network, has to be established.

In the next case in section 6.5.2, the number of training iterations are increased dramatically to a value of 5000. When the graph in Figure 6.95 (a) is examined, it is obvious that the error has not changed drastically from around 500 epochs. The results in Table 6.158 show that the training error has decreased from 0.0035 for the previously considered case, to 0.0031. There is therefore not much difference between the error for these two cases presented. The regression coefficient has increased slightly from 0.872 for the training phase of the previous case, to 0.889 once the epochs are increase. The regression value for the test phase shows a decrease from 0.85 for the previous case to 0.810 when the epochs is increased. The APE has increased to value of 9.405% when the training iterations are increased.

In section 6.5.3 the number of hidden layers are increase to two. The number of hidden neurons are selected to be 5 for both the first and second hidden layers. Although the training error has decreased to a value (0.0088) that is lower than the previous considered cases, the generalisation ability of the network is compromised. This is seen from the low value 0.206 for the regression coefficient in the test (validation) phase.

The variation considered in section 6.5.4 is where the NN is trained until the error converges to a minimum. The number of neurons in the hidden layer for the RBF has increased from a modest 19 to 414 neurons. The regression analysis value in Figure 6.97 (b) on the left shows a perfect fit for the training phase, but not good at all for the validation phase in the figure on the right. Figure 6.97 (c) shows the network is unable to generalise and the predicted values are pretty 'wild' with the APE at 41.327%.

## **6.7 Summary of the chapter**

The chapter presents the results of the NNs for COD, TKN and flow rate for the MLP FFNN, ERNN and the RBFNN. The models presented are variations on the input data combinations to the NNs. Thirteen models are presented for each considered case.

Overall the results show that NNs are a promising alternative to traditional linear prediction methods. However additional improvements to the NN structures have to be investigated.

Chapter 7 presents the practical implementation of this work as part of the control system for a lab-scale pilot plant at the University. All SCADA and PLC programs are presented and the NN GUI for the operation of the NN programs for the influent disturbances are shown.

## CHAPTER SEVEN

### REAL-TIME IMPLEMENTATION

#### 7.1 Introduction

The project under study is practically implemented using a pilot plant located in the Electrical Engineering Department at the Cape Peninsula University of Technology (CPUT) (Figure 7.1). This chapter presents the real-time implementation and describes the hardware configuration of the various components in the project together with a description of all software used. All software algorithms are presented in Appendix B. The technical information pertaining to the sensor connections, transmitters, calibration procedures, wiring and the PLC are presented in Appendix C.

#### 7.2 Pilot plant and control system

The pilot plant is a lab-scale version of an actual wastewater treatment plant. The plant is used to simulate conditions on a smaller scale to that of an actual WWTP. There are tanks (zones) that can be connected for various plant configurations such as those used for the UCT, Johannesburg and the Phoredox process structures, among others (Lilley, Pybus and Power, 1997). There are sensors placed inside the tanks and they measure pH, turbidity, conductivity, temperature and dissolved oxygen (DO). These sensors are connected to transmitters (Figure 7.2), which in turn are connected to the PLC. The PLC is in turn connected to the personal computer (PC) via the serial (RS232) port. The entire plant structure together with the software requirements for real-time implementation is displayed in Figure 7.3. The data acquisition was implemented with the assistance of a student whose document describes the technical aspects of the acquisition system (Mapisa, 2004) and can be viewed in Appendix C as is previously stated. A detailed description of all facets of the system follows.

#### 7.3 Raw plant and weather data conversion

The raw weather data received from the South African Weather Service ([www.weathersa.co.za](http://www.weathersa.co.za)) as described in Chapter 4, is in the ASCII (American Standard Code for Information Interchange) text format. The data is converted from this format into the MS (Microsoft) Excel format. The data is then converted once more from MS Excel into the format used by the MySQL database. Similarly the plant data received from the Athlone Municipal Wastewater Treatment Plant is converted from MS Excel into the MySQL database format. This conversion procedure for the raw data is illustrated in a step-by-step procedure in Appendix E.



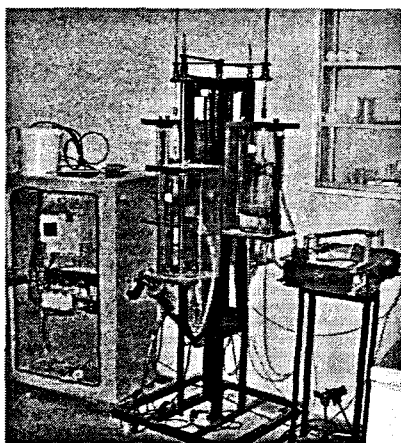


Figure 7. 1: The WWT Pilot plant together with the Modicon PLC and transmitters in the Electrical Engineering Department, CPUT

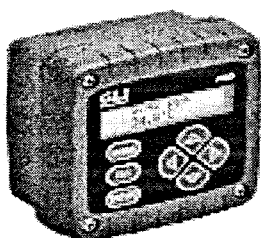


Figure 7.2: The Dissolved Oxygen transmitter

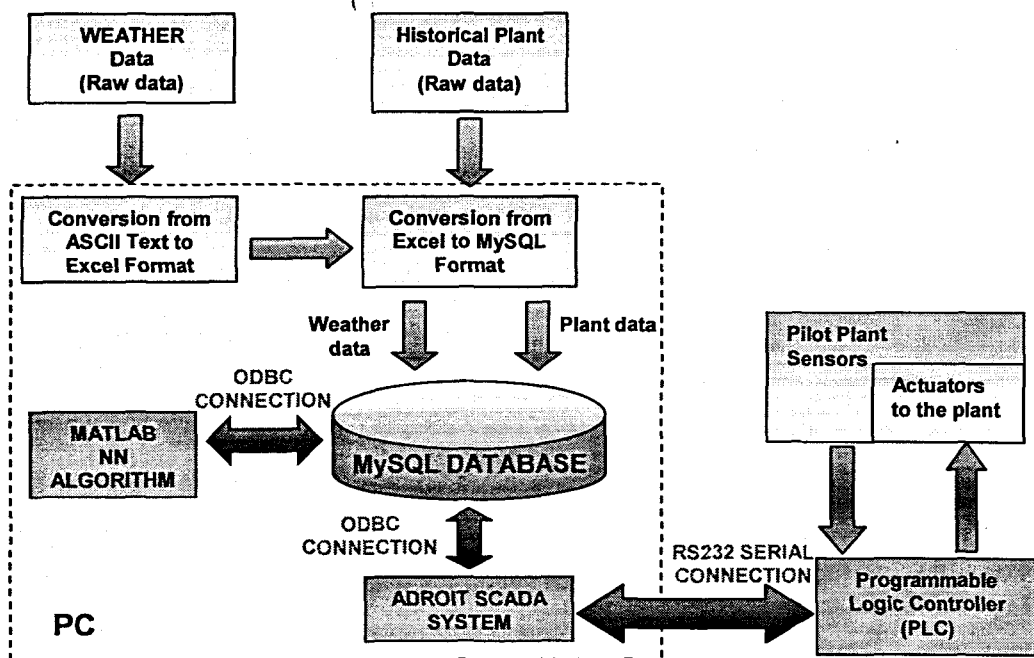


Figure 7. 3: The structure for the real-time implementation of the influent disturbances prediction showing the pilot plant, PLC, SCADA system, MySQL database with plant and weather data, and MATLAB NN algorithms

## 7.4 SCADA system

The plant sensory information is displayed on a graphical user interface (GUI) known as a SCADA (Supervisory Control and Data Acquisition) system. The SCADA system software utilised for the project is *Adroit Version 5.0*. The sensor data is transmitted to the PLC and then to the PC via an RS232 cable, where *Adroit* can retrieve data sent from the PLC on certain specified channels. The SCADA system is linked using the ODBC (open database connectivity) protocol, to the MySQL database that is the repository for all data utilized in this work. The database is at the heart of the communication between all the different software platforms used.

*Adroit* is an acronym for **A**dvanced, **D**istributed, **R**eal-time, **O**bject-oriented, **I**ntelligent, **T**oolkit. Not only does *Adroit* compete successfully, it has become the product of choice for users in mainstream as well as lesser-known industrial sectors. *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.

In *Adroit*, the server part of the client-server architecture is called an *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 embody "behavior". The definition of an object in object-oriented terms is precisely the same as an agent: namely, an entity that has identity, state, and behavior.

*Adroit's User Interface*, consisting of several different types of windows, makes up the client portion of the client-server architecture. The pictorial display windows within the user interface are themselves made up of objects, in this case graphical objects known as picture elements. In addition to state information, such as the properties of size, location, color, etc., In *Adroit* graphical objects may also be configured to have a wide variety of animation behaviors. By connecting these behaviors to data values within agents, many different animated display effects can be configured which, in real-time.

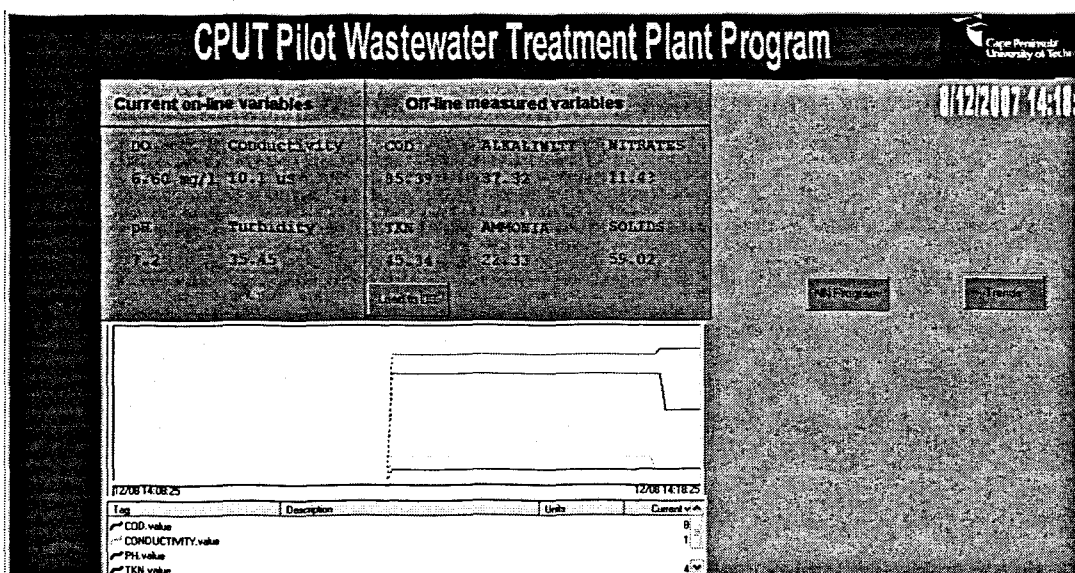
*Adroit's* two main components, the user-interface and the agent server are, therefore, founded on sound object-oriented principles. It is now widely recognized that such use of object-oriented principles is the most effective way of designing and implementing software that is robust and flexible enough to meet future, as yet

unforeseen, requirements. Exploitation of the object-oriented paradigm is, furthermore, fully carried through into Adroit's implementation, by use of C++, the programming language that is increasingly becoming recognized as the most effective way of delivering production grade object-oriented applications.

Typical Adroit configurations will in fact involve many of these object-oriented agent servers and user-interfaces running on several different physical computers connected in a local - or even wide-area network - with client-server communications happening transparently and seamlessly across the network. Adroit may therefore be succinctly described as an open, portable, object-oriented, industrial application toolkit, structured around a distributed client-server architecture, and hosted on operating systems from the industry standard Microsoft® 32 bit Windows™ family (Adroit User Manual, 1990-2000).

The developed Adroit SCADA GUI (Figure 7.4) allows the operator:

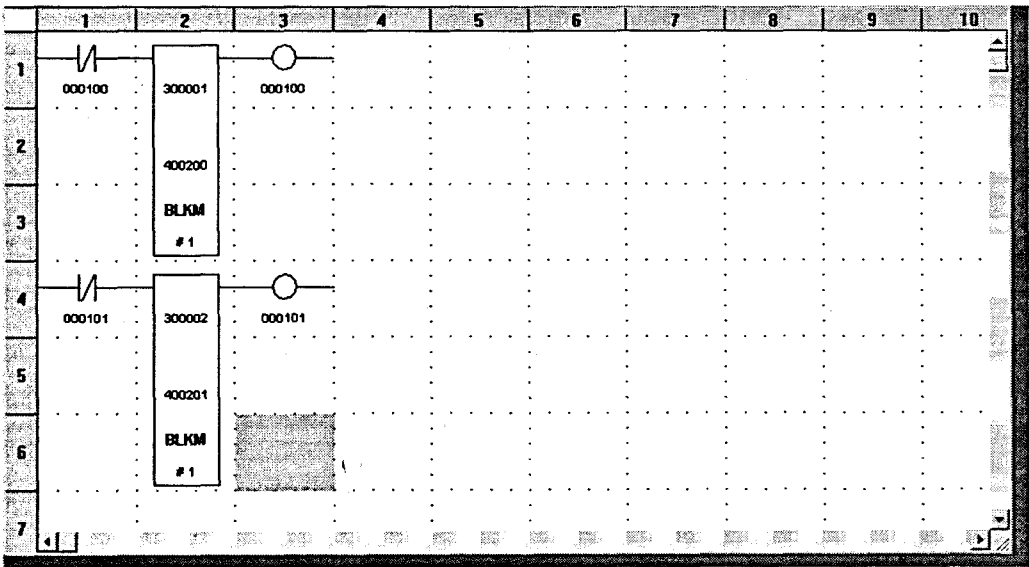
- to view all current variables of pH, turbidity, conductivity, and DO;
- to view current and historical trend data for these variables;
- to manually enter off-line laboratory variables manually in the database
- to execute the neural network program for training and validation of the NN for prediction of the influent disturbances of COD, TKN and flow rate;
- to use the predicted values for the program for the optimal control strategy;
- to implement the algorithm for real-time prediction of the inflow



**Figure 7. 4:** The SCADA system graphical user interface allows the user to view real-time plant variables, trend information, enter data in manual mode, run the neural network software for training and validation, and view and enter data in the database

## 7.5 Programmable Logic Controller (PLC) system

The Modicon Momentum PLC is programmed using software called Concept version 2.2. Concept is designed for use in any system configuration and it can be interfaced to powerful operators and monitoring systems. In this project it is used to acquire data from the wastewater treatment pilot plant in order to monitor and control process variables through the use of corresponding sensors and actuators and it is interfaced with the SCADA software package, Adroit. The type of programming language is ladder logic. It consists of schematic block diagram functions with unique functionality. The PLC program below consists of relays, coils and a function block.



**Figure 7. 5:** The PLC program to move sensory data from the Conductivity and DO sensors, from one analogue input register to another holding register

This PLC program is used to acquire data from the sensors (Figure 7.5). The **BLKM** or block move function is used to move the data from the analogue input 300001 to the holding register 400200. The address 000100 is for the output coil, which is also used to latch the normally closed contact, which is the input to the **BLKM**. The analogue input 300001 is used to acquire data from the Conductivity sensor. The Conductivity sensor signal is physically connected to this input 300001.

The second **BLKM** is also configured in the same way as the first one except that it is used to acquire data from the DO sensor, which is physically connected to the PLC analogue input 300002. The **BLKM** moves this data to the holding register 400201.

For additional PLC programs and connection information please consult Appendix D.

## 7.6 MySQL Database

A database is a structured, searchable collection of data or records. Databases are designed to offer an organized mechanism for storing, managing and retrieving information (Chapple 2005). Databases are actually much more powerful than spreadsheets in the way they are able to manipulate data. Databases could be used for:

- Retrieval of all records that match certain criteria
- Updating records in bulk
- Cross-referencing records in different tables
- Performing complex aggregate calculations

There are several types of databases in use today, with some of the most common in order of popularity, being:

1. Relational databases;
2. Object databases; and
3. Flat file databases.

The latter two types are not considered, however some popular relational databases include *MySQL*, *Oracle* and *Microsoft SQL Server*. Relational databases use tables to store information. The fields are represented as columns and records are represented as rows in a table, and they have a specific *relationship* between them; hence the term, '*relational database*'. The relational database stores and tracks data by establishing relationships between different pieces of information. It is therefore possible to find specific information very easily, and sort this information based on any field or record.

Data is requested from the database by sending it a *query*. This is usually accomplished by using a special programming language called SQL (Structured Query Language), which has been designed for the retrieval and management of data in databases. Queries allow one to select, insert, update, and locate data, etc in a database. For a particular query, the database returns a result set listing the rows with the requested information. The resulting rows can also be filtered to return the exact information required. SQL is the foundation for all popular database applications available today. For an introductory tutorial on SQL, please consult <http://www.w3schools.com/sql/default.asp>. An example of the SQL script for extracting data from the database into the MATLAB workspace is found in the `Open_all_data.m` script file (Appendix C).

As is stated previously the database is at the core of the entire research project. The database forms the interface between the plant with the PLC and SCADA system on one hand, and the neural network processing software, MATLAB (*Matrix Laboratory*)

on the other (Figure 7.3). The database contains the historical plant data from the Athlone wastewater treatment plant and the weather data from the Cape Town Weather Service. The outputs of the neural network are also stored in the database. The software used for the database is MySQL® version 5.0.4.

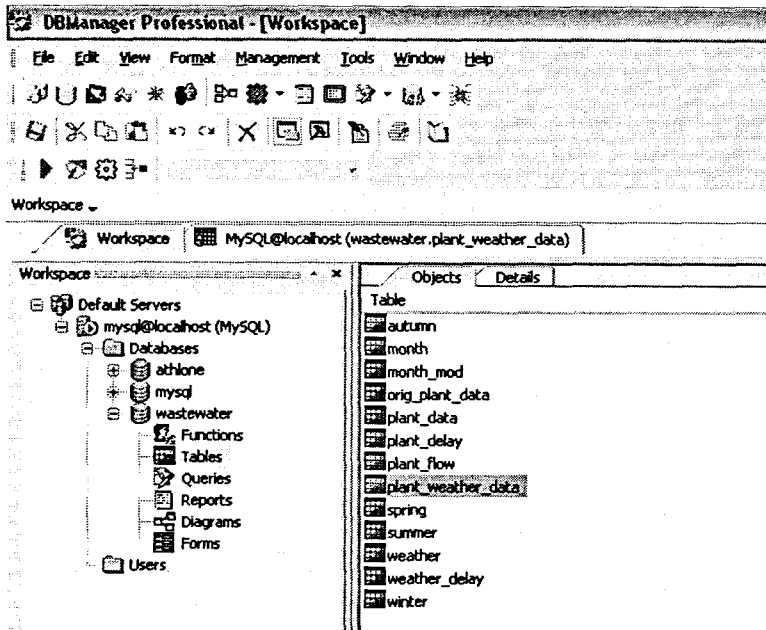
The MySQL® database software falls into the category of those programs that are regarded as *freeware*. This means that one is able to operate and use the software freely without the need of a license. The MySQL® database software has become one of the most popular open source databases due to its consistent fast performance, high reliability and ease of use. The MySQL database suite of products consists of :

- MySQL Query Browser 1.1.10
- MySQL Administrator 1.0.21
- MySQL Server 4.1.11
- DB Manager Professional 3.0.3a

The DB Manager Professional is one of the most powerful applications for data management and is used to set up and manage the database functions.

The connection between the MySQL database, MATLAB and the Adroit SCADA system, is made using the ODBC protocol. ODBC is a standard Application Programming Interface (API) developed by the Microsoft Corporation that allows one to connect to a data source (logical name for a data repository), for example, a MySQL database. This API allows databases created by various database management programs to be accessed using a common interface independent of the database file format. With an ODBC connection, one is able to connect to any ODBC compliant database from any ODBC compliant software platform. ODBC defines a set of function calls, error codes and data types that can be used to develop database independent applications. ODBC is usually used when database independence or simultaneous access to different data sources is required. Please refer to Appendix E.3 for the procedure to set up the ODBC connection.

The MySQL database window displayed in Figure 7.7 is the DBManager Professional application that allows the operator to inspect the various databases and tables that are configured here. The user is able to extract certain information from the fields within the different tables, or import or export data, or perform other management functions, should it be required.



**Figure 7. 6:** The MySQL database is shown with the tables with the data for the Athlone WWTP influent variables and the weather data

## 7.7 Neural Network Software algorithms

As stated previously all neural network algorithms presented in this work are implemented using the Neural Network Toolbox for use with MATLAB®, Version 4.0.6, (R14SP3). The MATLAB™ version used is MATLAB Version 7.1.0.246 (R14) Service Pack 3. (Demuth *et al.*, 2004). A standard desktop computer with Windows XP as an operating system is used to implement the programs.

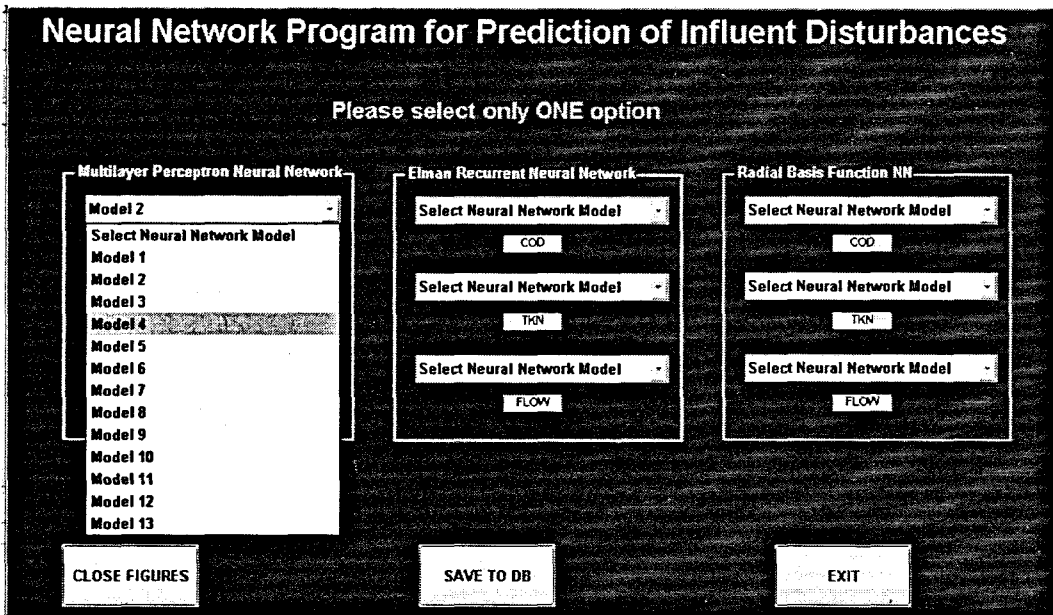
*MATLAB™/Simulink™* is a general high-level language for technical computing that includes a large library of predefined mathematical functions. There are approximately 40 toolboxes for use with MATLAB available that extend the MATLAB environment to solve particular classes of problems.

Simulink is an add-on software product to Matlab for modelling, simulating and analyzing any type of dynamic system. Matlab and Simulink are fully integrated, meaning that all functionalities of the MATLAB toolboxes are available in the Simulink environment as well. Simulink provides a graphical user interface for building models as block diagrams and manipulating these blocks dynamically. A large number of predefined building blocks are included and it is easy to extend the functionality by creating custom blocks or new ones.

*The Neural Network Toolbox* facilitates the implementation of neural networks in MATLAB. The toolbox consists of a set of functions and structures that handle neural

networks, so it is not necessary to write programs for all activation functions, training algorithms, etc. that we want to use. The *nntool* command shows a graphical representation of the neural network structure. This interface is similar to the Simulink interface.

The neural network program is executed when the operator clicks on the NN program button in the Adroit SCADA GUI, or types the '*NN\_GUI*' command in the MATLAB workspace. The user can select the type of NN required and the influent disturbance that needs to be predicted (Figure 7).

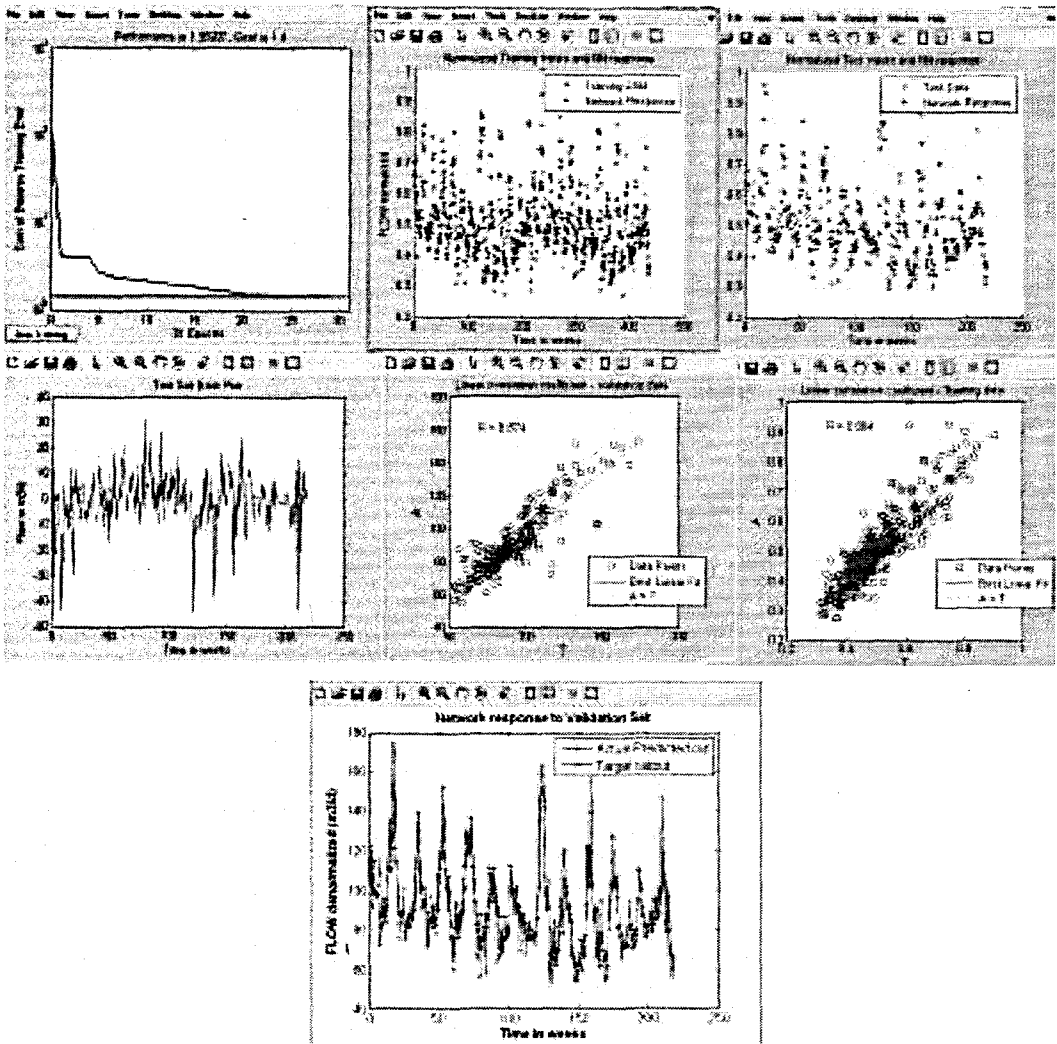


**Figure 7. 7:** The MATLAB GUI for the Neural Network program for prediction of the influent disturbances

Upon execution of the program, 7 figures are displayed (Figure 7.8). These include:

- the training phase showing the error being minimized;
- the scatter-plot of the network response to the training data;
- the scatter-plot of the network response to the test (validation) data;
- the instantaneous error between the desired target and the network response;
- the plot of the desired target and the NN response;
- regression plot for the training phase; and
- regression plot for the validation phase.





**Figure 7. 8:** The plots illustrate the training, validation, and NN response to training validation data, and network performance

When executing the script files in the MATLAB editor or workspace, the order of execution for the files is:

1. Open\_all\_data.m
2. Preprocess.m
3. raw\_norm\_inputs.m
4. Train\_tst\_data.m
5. Feedforward.m
6. Elman.m
7. RBF.m
8. Postprocess

The descriptions of the MATLAB script files for use in the implementation of the neural network prediction algorithms are summarized in Table 7.1. A selection of the implemented MATLAB software algorithms together with a description of variables used is presented in Appendix C. The flow chart describing the neural network software algorithm development is given in Figure 7.9.

**Table 7. 1: MATLAB script files and the functional description in the development of the NN program for the prediction of the influent disturbances of COD, TKN and influent Flow rate**

Filename	Description
Open_all_data.m	This file establishes communication between MATLAB and the MySQL database. All raw wastewater treatment influent plant and environmental weather variables are loaded into the MATLAB workspace. These include COD, TKN, flow rate, minimum and maximum temperature, rainfall, and wind speed.
Correlation_Coefficients.m	This file calculates the correlation coefficients between the different input variables of influent disturbances, COD, TKN, flow rate, minimum and maximum temperature, rainfall, and wind speed. This measure gives an indication of the relationships between the variables and aid in eliminating redundant variables.
Preprocess.m	In this file the raw data variables are normalized using a scaling method of dividing them by the maximum value. This ensures that all variables remain in more or less the same range so that and no one variable appears to be more important than another. This also ensures that all input variables to the NN are normalized to a value not greater than +1.
raw_norm_inputs.m	In this file the normalized data is divided into an input and target output set. Also the time-delayed inputs for influent variables are set up here.
Train_tst_data.m	The data is divided into two sets, a training and test (validation) set. Two thirds of the entire data set is used for training and one-third for validation. Every third sample is extracted for validation.
Feedforward.m	Here a feed-forward NN is designed with a specified number of parameters such as the number of hidden layers, neurons, and epochs, training algorithm, mean square error goal, and the activation functions for the hidden and output layers. The NN is trained and simulated with the training data. The validation phase follows and the NN is simulated with the test data.
Elman.m	The design of an Elman recurrent NN is implemented with a specified number of parameters such as the number of hidden layers, neurons, and epochs, training algorithm, mean square error goal, and the activation functions for the hidden and output layers. The NN is trained and simulated with the training data. The validation phase follows and the NN is simulated with the test data.
RBF.m	A radial basis function NN is designed where the sum of square error goal and the spread constant of the Gaussian activation function, is specified. The NN is trained and simulated with the training data. The validation phase follows and the NN is simulated with the test data.
Postprocess.m	All data variables are denormalized and the NN performance is calculated. The regression coefficient, determination coefficient, and absolute percentage error are calculated
Activation.m	This script file plots the various activation functions such as the hard limit, bipolar hard limit threshold, logarithmic sigmoid function, hyperbolic tangent function, radial basis function and the pure linear function.
Std_min_max.m	In this file the calculations for the minimum, maximum and standard deviation values, are performed.
Components.m	Calculates the COD and TKN influent components using the UCT model special coefficients
NN_GUI.fig	The GUI for the NN program. Three different NNs topologies are developed: MLP, ERNN and the RBFNN. Three NNs, one for each influent disturbance of COD, TKN and flow rate is also developed.

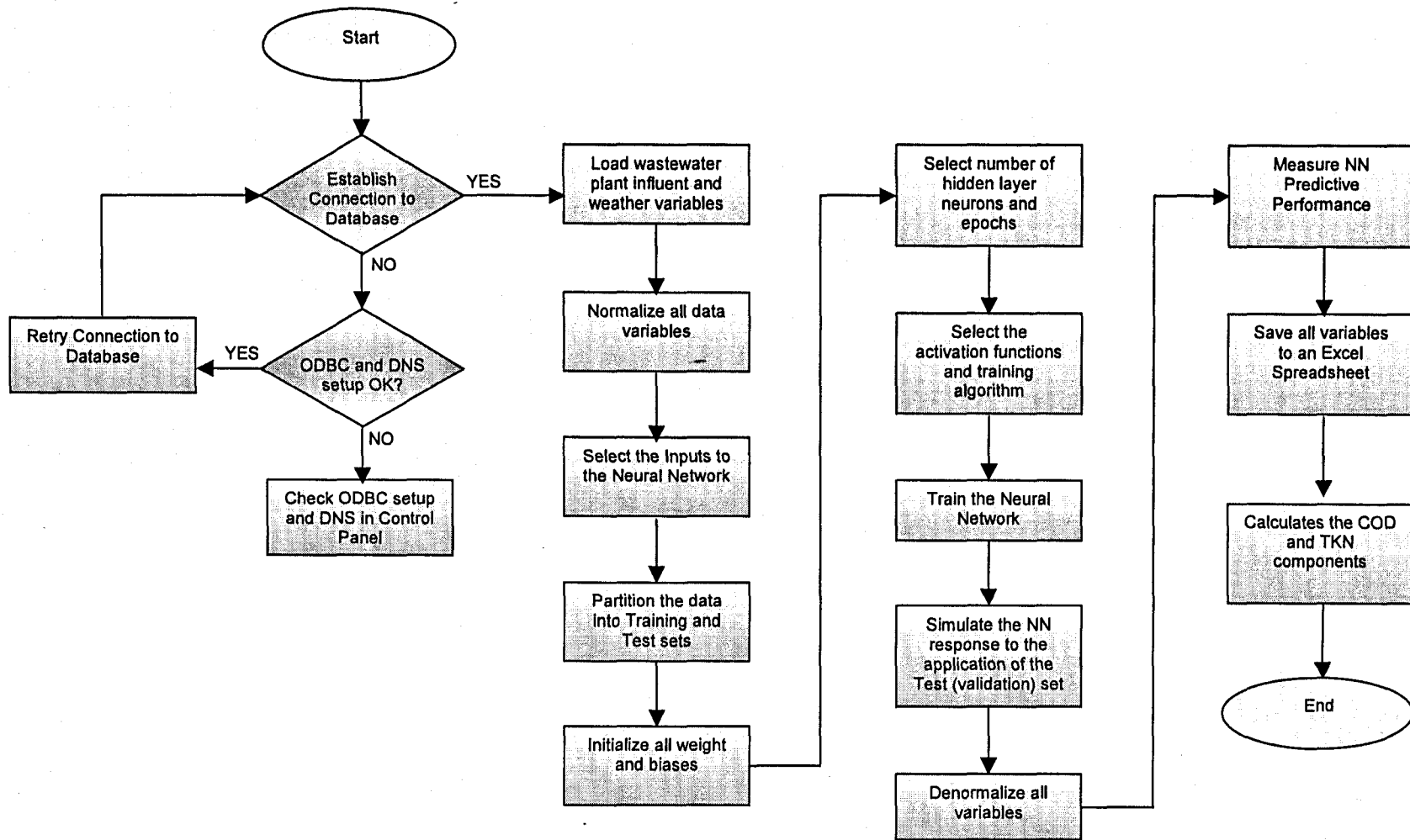


Figure 7. 9: The software flow diagram

## **7.8 Real-time implementation algorithm**

### **7.8.1 NN retuning and calculation of the new predicted values if COD, TKN, Flow rate**

The following procedure is implemented weekly:

1. The operator checks the database every Monday morning at 8h00 to ensure that the input data for the inflow COD, TKN and flow is saved.
2. The operator acquires the weather data and sends it to the database.
3. If the data is saved properly, the operator starts the GUI for prediction of the COD, TKN and flow for the next week.
4. The type of NN is selected.
5. The MATLAB program for training the NN is started.
6. The resulting weights are checked; if they differ +/- 10% more than the old weights, the old weights are selected from the database.
7. The prediction of the COD, TKN and flow rate is calculated using the selected weights.
8. The operator checks the results for predicted COD, TKN and flow. If they differ by more than +/- 10% from the corresponding measured values for the last 2 weeks, the old values are used.
9. The results are sent to the database and displayed on the Adroit window.
10. The procedure is repeated every week.

### **7.8.2 Calculation of the inflow COD and TKN components**

When the operator accepts the new predictions, the program for calculation of the COD and TKN is started. The obtained values of the concentrations of all biodegradable and non-biodegradable components are sent to the MySQL database. With this step the tasks for prediction of the influent disturbances are completed. The software for process simulation, state and parameter estimation, optimization and controller design then reads the obtained values.

In this way the repetitive calculation of the influent disturbances every week forms the basis for the adaptive optimal control strategy of the activated sludge process.

The period for prediction is one week, but it could very easily be made a couple of hours if the special model of the hourly, daily and weekly change of the inflow concentrations around the measured weekly average value is accepted (Olssen and Newell, 1999). Then the control strategy could be adjusted also according to the diurnal variations of the influent variables.

The real-time implementation algorithm is based on the old historical data with new values for the influent disturbances from the laboratory measurements and from the flow sensors, weather data from the weather bureau and the current month of the year, added every week. The new data is added to the training data set first and then to the test (validation) set. In this way the number of data points in the sets is growing which leads to better results from the prediction and better generalization when the number of data points reaches some limit (10 000). Some of the oldest data points are removed from both sets.

## **7.9 Summary of the chapter**

The chapter provides an overview of the real-time implementation strategy for the project. The pilot plant at CPUT is presented together with the PLC program for acquiring data from the process sensors. The SCADA system implemented in Adroit is coupled to the MySQL database using the ODBC protocol. The complete software algorithms are presented in Appendix C, while the connection and wiring diagrams are presented in Appendix D. Appendix E presents the conversion of the formats into the MySQL database format.

Chapter 8 provides the conclusion to this work and provides future direction to the project.

## CHAPTER EIGHT

### CONCLUSIONS AND FUTURE DIRECTION OF RESEARCH

#### 8.1 Conclusion

The problem statement spells out quite clearly the need for effective control of wastewater treatment plant processes particularly the activated sludge process. The effect that the influent disturbances of COD, TKN and influent flow rate have on effectively controlling the ASP is highlighted.

The problem that the implemented neural networks are required to solve is the prediction one step ahead of the influent disturbances of COD, TKN and influent flow rate on the basis of historical measurement of these disturbances, weather conditions and the month of the year. The data inputs are represented as time series in order to incorporate the dynamic character of the environment and influent behaviour. Three NN architectures are investigated to solve this problem, namely the multilayer perceptron with delayed inputs, the Elman recurrent NN and the radial basis function NN. The results of the performance of the neural networks are provided in Chapter 6.

The results show all three types of neural networks exhibited very similar forecasting behaviour, although the impression is obtained from the literature survey that one type of architecture is better than the other. One of the definite differences between the three types of neural networks is the speed at which the different networks are trained. Some types of neural networks are however suited to particular types of problems (Haykin, 1999) and perform better than others. The differences between the multilayer perceptron and the radial basis functions are noted in Chapter 5.

The inputs to all three types of NN are varied to include combinations of the influent disturbance, delayed influent disturbance, other influent disturbances, environmental variables of temperature, rainfall and wind speed, and month of the year data. The input combinations are reflected in Chapter 5, Table 5.1, Table 5.2 and Table 5.3 as Model 1 till Model 13.

The first set of results in section 6.2 pertains to those for the COD influent disturbance variable as the output. The results for the COD neural networks show that all three types of neural network are unable to respond well to the input training data, and the generalisation performance of the networks are also compromised. Upon application of the test set, the regression coefficient  $r$ , the coefficient of determination  $r^2$  and

the absolute percentage error for the difference between the desired target output and the NN output, indicate that the NN for COD at the output is not able to solve the given input-output mapping problem well. Despite rigorous experimentation by increasing the number of neurons gradually up to a maximum of 80, and the number of epochs up to a maximum of 10 000, the NN did not perform any better.

The second set of results in section 6.3 pertains to those for the TKN influent disturbance variable at the output. The results for the TKN neural networks show that all three types of neural network are again unable to respond well to the input training data, although the results are an improvement to that obtained for the COD neural networks. Once more the generalisation performance of the networks are poor as the NN gives a poor representation of the underlying training data.

The third set of results pertains to that of the influent flow rate variable at the output and is presented in section 6.3. This time there is a marked improvement in terms of the predictive performance of all three types of NN. The regression coefficients for the test or validation phase are on average throughout all the various input combinations at a value of approximately 0.85. Similarly the coefficient of determination has increased to a value of approximately 0.72. Although these figures are not too bad, the absolute percentage error is above 8%. This indicates that the neural network's generalisation ability although not poor when compared with the training regression figures, is still not acceptable. Additional experimentation on all three different architectures did not yield an improvement on performance for the neural networks for the influent flow rate variable.

The final set of results in section 6.4 is a comparison of the effect that increasing or decreasing the number of epochs and number of neurons in the hidden layer, has on the network performance. Also the effect that training a NN to convergence has on the generalisation ability of the network are presented here. Networks that are over-trained tend to learn the noise patterns in the training data set as well.

The fact that the neural networks could not solve the problem well indicates that:

- the problem is non-deterministic and the NN will never solve it; or
- the input data is not representative enough of the process.

These two reasons according to Masters (1993) are the only two in his opinion why neural networks are not successful. The first reason is the most likely option and is the experience of many NN practitioners whom the author has spoken to has been that the real-world applications are the most difficult to solve.

Additionally the quality of the wastewater treatment plant data may be another factor to consider. The experience of many plant operators and engineers has been that often plant equipment fails, and at times data is then erroneously reported, sometimes unknowingly. The problems encountered at wastewater treatment plants are reported in Chapter 1. The current situation in Cape Town, South Africa is that there are around 43% of the "manned" plants that have some sort of SCADA system as reported in Chapter 2. This advent of monitoring and control by introduction of an automation system might positively contribute to earlier fault detection and may decrease the incidence of faulty equipment and a lack of reliable reported data.

## **8.2 Thesis results and application of the results**

Scientific and application results are obtained as a result of the research done on the problem for development of a model for the dynamic behaviour of the wastewater treatment plant influent variables (COD, TKN, Flow rate) as a result of influence of the weather conditions (temperature, wind, rainfall) and the season of the year (month).

The scientific results can be stated as follows:

1. The wastewater treatment monitoring and control is considered and the role of the influent variables as the treatment process main disturbances is stated.
2. The problem for the development of the mathematical model for the dynamic behaviour of the wastewater treatment plant influent variables is analysed, and according to its characteristics a problem for prediction of the influent variables is formulated as a problem for development of a neural network type and its structure.
3. Analysis of the existing types and structures of the NNs is done according to the requirements of the formulated problem and 3 types of NN are proposed to be used:
  - Multilayer perceptron network with delayed inputs
  - Elman recurrent network
  - Radial basis function with delayed inputs
4. A procedure for preprocessing of data for the NN training and validation is developed on the basis of analysis of the existing approaches in the literature.
5. MLP with delayed data is developed for one-step ahead prediction of the influent COD, TKN and Flow rate. Thirteen (13) different variants / structures of the network are created on the basis of variation of the type and the number of inputs (Model 1 – Model 13).
6. Elman recurrent network is developed for on-step ahead prediction of the influent COD, TKN and Flow rate. Thirteen (13) different variants of the



- network are created on the basis of the same as in point 5 variants of the input data (Model 1 – Model 13).
7. Radial basis function network is developed for on-step ahead prediction of the influent COD, TKN and Flow rate. Thirteen (13) different variants of the network are created on the basis of the same as in point 5 variants of the input data (Model 1 – Model 13).
  8. The generalized conjugate gradient method is applied in an innovative way to organize the training process in all variants of the developed NNs.
  9. Different variants of the NNs are developed and compared with those previously developed according to the number of the hidden layers, number of hidden layer neurons, number of epochs, and training to convergence (over-training).
  10. MATLAB software is written to implement all tasks for pre-processing of data, training, validation, analysis of the solutions and graphical user interface.

The application results can be stated as follows:

1. Data from the Athlone wastewater treatment plant for the influent variables and from the weather bureau is gathered and processed.
2. Hardware structure of the control system for pilot wastewater treatment plant is developed where the physical connections between the components of the system (plant, sensors, transmitters, PLC, PC) and their calibration is done.
3. Data acquisition system for DO, pH, conductivity, temperature and turbidity is developed using the sensors, transmitters, PLC, PC and MySQL database.
4. Adroit SCADA system is developed for data acquisition, monitoring and calculation of the predicted values of the influent disturbances.
5. Communication between the software platforms Adroit, MATLAB and MySQL is established in order to solve the problems for the influent COD, TKN and Flow rate prediction in real time. A GUI is developed to implement the tasks for communication and prediction.

### **8.3 Application of the obtained results**

The developed NN models for prediction of the influent disturbances have the following positive characteristics:

- the structures of the NN are simple, with one hidden layer and do not require complex calculation;
- the time for the training of the networks is short;
- the scaled conjugate gradient method application leads to test and stable convergence to the minimum of the least squares error; and
- it is able to solve nonlinear real-world problems.

On the basis of the above the applicability of the results could be considered in the following directions:

- as a basis for further development of the theory for prediction of the influent variables for WWTP;
- as an expert system for the advice of the operator in the municipality WWTP;
- as a real-time toolbox of the existing SCADA systems in the WWTPs; and
- as a tool for future postgraduate student's projects at university.

Implementation of the developed software in the municipality SCADA systems can be done very easily because these SCADA systems are built using Adroit software and Modicon PLCs. Then the operation of the process done according to the predicted values of the influent disturbances will lead to adaptive control of the aeration, corresponding to these disturbances and as a result will reduce the power needed for control.

The solution of the problem for development of the mathematical model for dynamic behaviour of the influent disturbances according to the influence of the weather conditions and the season of the year is the first attempt in the scientific and research literature so far. The results give knowledge for the connection of these groups of variables and the methods and software for its acquisition.

#### **8.4 Future research**

The research has shown that neural networks are a viable option to consider for prediction of the influent disturbance variables to a wastewater treatment plant. However the effective implementation may warrant additional research. The literature research and the opinion of other more experienced neural network practitioners have indicated that a hybrid network might be the solution. This might include other soft computing options such as a fuzzy-neural network, or a neural network combined with a genetic algorithm.

#### **8.5 Publications related to the thesis**

1. Kriger, C., Tzoneva, R. 2005. "Neural Network Structures for Prediction". Postgraduate Control Systems Day, Pretoria.
2. Kriger, C., Tzoneva. R. 2007. " A Neural Network model for control of wastewater treatment processes, NOLCOS 2007, DVD ROM.
3. Kriger, C., Tzoneva. R. 2007. "Neural Networks for Prediction of Wastewater Treatment Plant Influent Disturbances", Conference Proceedings of 'IEEE AFRICON' Windhoek, Namibia, DVD ROM.
4. Kriger, C., Tzoneva. R. 2008. "Development of a NN software sensor for prediction of the wastewater treatment process variables", IEEE Control Systems Technology (submitted to journal).

# APPENDIX A

# APPENDIX A

## TYPES OF NEURAL NETWORKS

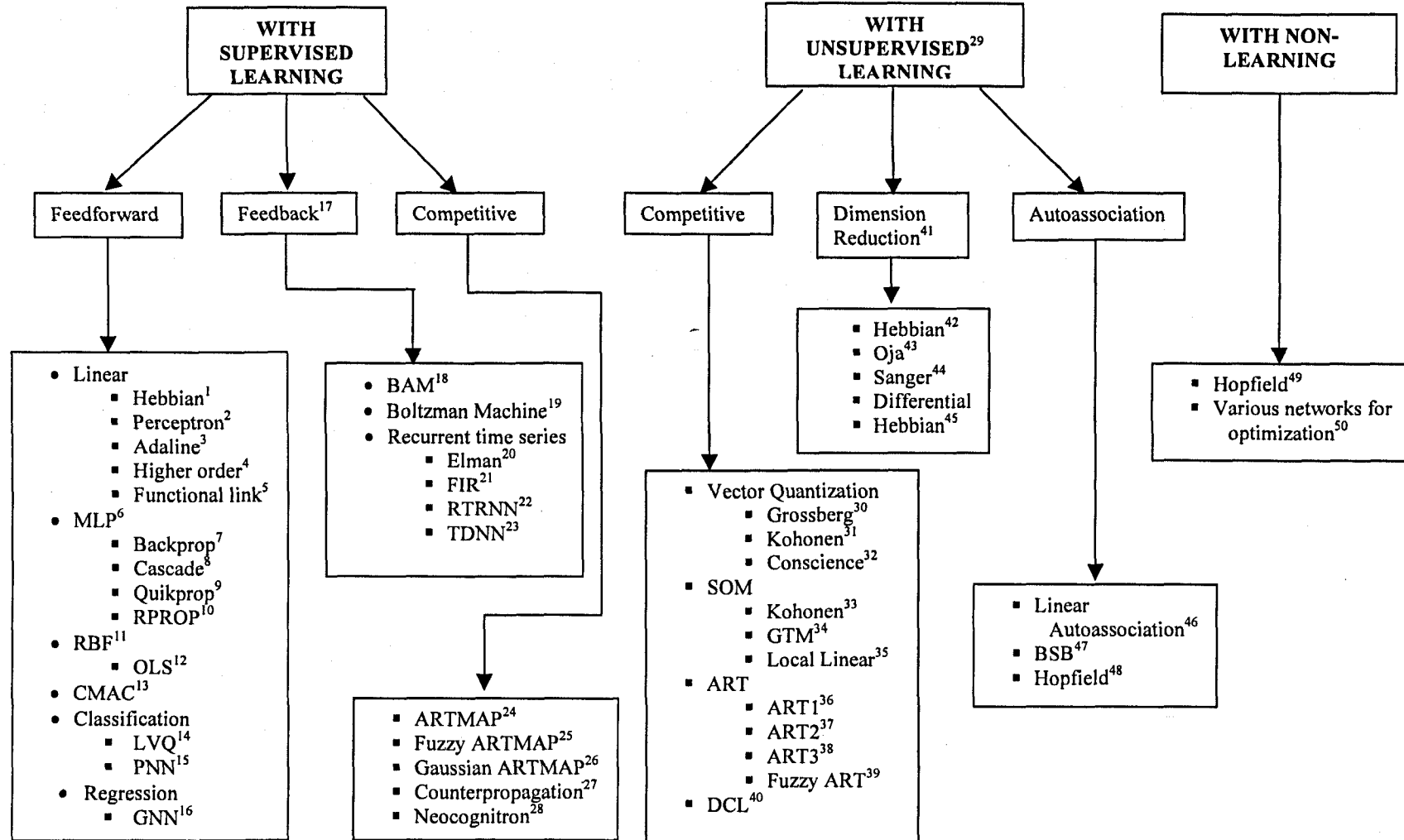


Figure A. 1: Well-known types of Neural Networks

1. Hebb (1949), Fausett (1994)
2. Rosenblatt (1958), Minsky and Papert (1969/1988), Fausett (1994)
3. Widrow and Hoff (1960), Fausett (1994)
4. Bishop (1995)
5. Pao (1989)
6. Bishop (1995), Reed and Marks (1999), Fausett (1994)
7. Rumelhart, Hinton, and Williams (1986)
8. Fahlman and Lebiere (1990), Fausett (1994)
9. Fahlman (1989)
10. Riedmiller and Braun (1993)
11. Bishop (1995), Moody and Darken (1989), Orr (1996)
12. Chen, Cowan and Grant (1991)
13. Albus (1975), Brown and Harris (1994)
14. Kohonen (1988), Fausett (1994)
15. Specht (1990), Masters (1993), Hand (1982), Fausett (1994)
16. Specht (1991), Nadaraya (1964), Watson (1964)
17. Hertz, Krogh, and Palmer (1991), Medsker and Jain (2000)
18. Kosko (1992), Fausett (1994)
19. Ackley et al. (1985), Fausett (1994)
20. Elman (1990)
21. Wan (1990)
22. Williams and Zipser (1989)
23. Lang, Waibel and Hinton (1990)
24. Carpenter, Grossberg and Reynolds (1991)
25. Carpenter, Grossberg, Markuzon, Reynolds and Rosen (1992), Kasuba (1993)
26. Williamson (1995)
27. Hecht-Nielsen (1987; 1988; 1990), Fausett (1994)
28. Fukushima, Miyake, and Ito (1983), Fukushima, (1988), Fausett (1994)
29. Hertz, Krogh, and Palmer (1991)
30. Grossberg (1976)
31. Kohonen (1984)
32. Desieno (1988)
33. Kohonen (1995), Fausett (1994)
34. Bishop, Svensen and Williams (1997)
35. Mulier and Cherkassky (1995)
36. Carpenter and Grossberg (1987a), Moore (1988), Fausett (1994)
37. Carpenter and Grossberg (1987b), Fausett (1994)
38. Carpenter and Grossberg (1990)
39. Carpenter, Grossberg and Rosen (1991b)
40. Kosko (1992)
41. Diamantaras and Kung (1996)
42. Hebb (1949), Fausett (1994)
43. Oja (1989)
44. Sanger (1989)
45. Kosko (1992)
46. Anderson et al. (1977), Fausett (1994)
47. Anderson et al. (1977), Fausett (1994)
48. Hopfield (1982), Fausett (1994)
49. Hertz, Krogh, and Palmer (1991)
50. Cichocki and Unbehauen (1993)

## APPENDIX B

## APPENDIX B

### SELECTED RESULTS OF INITIAL AND FINAL WEIGHTS AND BIASES

This section provides the initial and final conditions for the weights in the neural network for selected models. All weights are saved in Excel and are also imported into the database. For the real-time implementation, the current weights are examined for newly trained networks and the previous weights are considered before selecting a new predicted result.

The tables below all have the following abbreviations:

init inp weights	–	initial input weights
init out weights	–	initial output weights
Initial inp bias	–	initial input bias
Initial out bias	–	initial output bias
Final inp weights	–	final input weights (after training)
Final out weights	–	final output weights (after training)
Final inp bias	–	final input bias
Final out bias	–	final output bias
b(1)	–	input bias
b(2)	–	output bias

### B. 1: Feed-Forward Neural Network for COD - MODEL 1

The input for model 1 is COD(k) and the output is COD(k+1).

**Table B. 1: The initial inputs weights and biases for COD Model 1**

Hid neurons	Epochs	COD(k)		COD(k+1)		b(1)		b(2)		
		Init inp weights	Init out weights	Final inp weights	Final out weights	Initial inp bias	Final inp bias	Initial out bias	Final out bias	
1	500	3.195	5.251	0.165	0.112	-1.795	-1.662	0.018	0.385	
	5	15.975	16.579	-0.003	-1.004	-15.975	-15.729	0.332	0.343	
5		-15.975	-15.748	-0.419	-0.982	12.475	14.580			
		-15.975	-15.741	0.346	0.028	8.975	9.755			
		15.975	15.310	0.916	0.044	-5.475	-6.302			
		-15.975	-15.520	0.533	-0.044	1.975	3.972			
	500	31.951	33.003	0.828	-0.585	-31.951	-30.833	0.004	0.628	
		31.951	31.917	-0.543	0.659	-28.839	-28.974			
		-31.951	-31.809	0.724	0.166	25.728	25.899			
		-31.951	-31.233	0.313	-0.149	22.617	23.586			
		-31.951	-32.379	0.782	0.162	19.506	18.968			
		-31.951	-31.245	-0.024	-0.137	16.395	17.756			
10		-31.951	-32.184	0.985	-0.034	13.284	12.941			
		31.951	32.246	-0.253	0.020	-10.173	-9.201			
		-31.951	-31.766	0.063	-0.028	7.062	7.749			
		-31.951	-32.043	-0.637	0.213	3.951	3.286			
	1000	-3.195	-5.689	0.317	-0.106	1.795	1.840	0.727	0.389	
	5	1000	-15.975	-16.740	0.027	1.244	15.975	15.752	-0.184	0.337
			15.975	16.993	-0.574	1.212	-12.475	-15.666		
			-15.975	-15.946	-0.793	0.026	8.975	9.848		
			-15.975	-15.446	-0.685	-0.044	5.475	6.354		
			15.975	15.735	-0.185	0.043	-1.975	-4.030		
1000	31.951	33.199	0.122	-0.964	-31.951	-30.907	0.644	0.667		
		-31.951	-31.828	0.232	-1.029	28.839	29.147			
		-31.951	-32.007	0.324	0.251	25.728	25.640			
		-31.951	-30.986	0.233	-0.232	22.617	23.932			
		-31.951	-32.959	0.370	1.314	19.506	19.642			
		31.951	30.403	0.020	1.305	-16.395	-18.096			
		31.951	32.350	0.428	0.035	-13.284	-12.974			
		-31.951	-32.221	0.030	0.025	10.173	9.285			
		31.951	31.558	0.212	0.070	-7.062	-8.565			
		-31.951	-32.098	0.933	0.247	3.951	3.136			



## B. 2: Feed-Forward Neural Network for COD - MODEL 2

The inputs for Model 2 are COD(k) and COD(k-1) and the output is COD(k+1).

**Table B. 2: The initial inputs weights and biases for COD Model 2**

Hid neurons	Epochs	COD(k)	COD(k-1)	COD(k)	COD(k-1)	COD(k+1)	COD(k+1)	b(1)	b(1)	b(2)	b(2)		
		Init inp weights	Init inp weights	Final inp weights	Final inp weights	Init out weights	Final out weights	Initial inp bias	Final inp bias	Initial out bias	Final out bias		
1	500	-2.792	1.614	-0.763	-0.596	0.934	-0.827	0.688	-0.383	0.330	-0.182		
		5.817	-4.310	6.325	-1.720	0.688	0.161	-4.048	-4.257	-0.320	0.424		
		-3.651	-6.382	-3.530	-6.263	-0.652	0.040	7.096	7.373				
		4.186	-6.016	3.659	-6.263	-0.658	-0.153	0.929	-0.063				
		6.754	2.420	6.250	2.097	0.989	0.217	-3.549	-2.667				
5	500	-4.681	-5.608	-4.191	-5.881	-0.120	0.157	2.558	2.988				
		10	500	-8.257	6.051	-9.206	5.571	-0.052	-0.341	5.766	5.282	-0.943	-0.189
		-6.513	8.027	-6.973	7.121	0.818	0.349	2.725	2.748				
		-0.057	-10.499	-0.051	-10.676	0.192	-0.687	8.217	8.404				
		-2.804	-10.087	-2.354	-10.049	-0.342	0.649	8.552	9.250				
10	500	4.481	9.410	4.860	8.891	-0.044	0.028	-8.141	-8.209				
		9.634	3.163	9.636	3.161	0.194	0.023	-6.646	-6.657				
		-6.665	-7.890	-7.129	-7.922	-0.677	-0.058	6.572	5.475				
		7.031	-7.540	6.016	-7.662	0.659	0.131	2.621	3.167				
		7.516	-7.017	7.644	-6.543	0.912	0.678	3.047	3.872				
1	1000	9.644	-3.130	9.679	-2.942	0.191	-0.133	0.716	1.013				
		1.485	2.939	0.688	0.529	-0.531	1.323	-2.437	0.673	0.098	-0.665		
		5	1000	7.143	-0.154	8.161	-0.342	-0.371	0.115	-7.060	-6.527	0.175	0.460
		5.996	-4.037	6.227	-4.678	0.217	-0.258	-2.732	-1.609				
		-7.071	-1.061	-7.684	-1.879	-0.650	-0.028	4.551	3.001				
5	1000	-5.707	4.466	-5.891	4.405	0.242	-0.240	-0.795	1.092				
		-4.204	-6.002	-0.567	-6.820	-0.508	-0.073	2.505	3.322				
		10	1000	0.806	10.466	0.697	10.900	0.866	0.762	-10.587	-10.170	-0.428	-0.278
		4.409	9.447	3.145	10.603	0.427	-1.118	-11.072	-10.627				
		7.746	6.741	8.753	6.029	-0.544	0.127	-10.487	-10.606				
10	1000	-0.813	-10.465	-0.330	-10.494	-0.101	-0.697	7.639	8.028				
		-4.995	9.127	-4.618	8.353	-0.656	0.223	-1.679	-1.487				
		-8.017	6.390	-8.121	9.475	0.938	-0.194	0.528	-0.623				
		-1.189	10.426	-0.001	10.465	-0.289	-0.254	-6.494	-7.022				
		-4.378	-9.462	-2.482	-9.715	-0.902	-0.053	5.160	6.359				
10	1000	-7.033	7.538	-7.610	7.396	0.511	-0.861	-3.603	-4.082				
		10.084	0.657	10.077	0.875	0.790	0.086	-1.596	-0.768				

**B. 3: Feed-Forward Neural Network for COD - MODEL 3**

The inputs for Model 3 are COD(k), COD(k-1), COD(k-2), and the output is COD(k+1)

**Table B. 3: The initial inputs weights and biases for COD Model 3**

Hid neurons	Epochs	COD(k)	COD(k-1)	COD(k-2)	COD(k)	COD(k-1)	COD(k-2)	COD(k+1)	COD(k+1)	b(1)	b(1)	b(2)	b(2)
		Init inp weights	Init inp weights	Init inp weights	Final inp weights	Final inp weights	Final inp weights	Init out weights	Final out weights	Initial inp bias	Final inp bias	Initial out bias	Final out bias
1	500	1.035	-3.116	0.392	-0.966	-0.661	-0.705	-0.091	-0.654	0.893	-0.058	0.810	-0.075
	5	500	-2.072	5.053	1.427	-3.767	4.451	1.999	0.475	0.132	-0.018	-0.212	0.585
		-3.882	-0.380	3.947	-2.183	-0.148	2.554	0.733	-0.220	1.315	0.645		
		0.064	4.900	2.845	0.617	3.424	3.481	0.982	0.721	-4.344	-6.949		
		0.787	-0.367	5.564	1.158	0.850	5.749	0.008	0.120	-2.245	-2.218		
		3.026	-3.498	3.155	3.155	-3.907	1.681	0.258	0.059	0.787	-0.082		
10	500	4.140	-4.613	3.348	4.390	-3.776	3.735	-0.027	-0.488	-4.752	-4.959	0.891	0.121
		-3.158	-3.691	5.136	-3.271	-2.823	5.300	0.501	-0.137	3.179	3.783		
		-2.497	4.382	4.985	-1.646	4.515	5.269	-0.748	0.085	-2.178	-1.793		
		-3.221	0.339	-6.265	-4.453	-0.191	-4.738	-0.914	-0.456	6.233	6.797		
		5.121	-4.660	1.053	5.457	-4.121	1.798	-0.258	0.066	-1.277	-0.575		
		-0.273	3.303	6.291	1.391	4.351	4.953	0.387	-0.265	-5.601	-6.618		
		-1.602	-6.262	-3.008	-1.187	-6.392	-2.995	0.872	-0.084	5.039	5.120		
		6.405	2.594	0.366	6.590	2.614	0.698	-0.045	0.007	-3.548	-3.128		
		-1.718	6.637	1.966	-1.533	6.745	2.616	-0.742	0.154	-6.131	-5.825		
		6.484	-1.727	-1.655	6.506	-1.731	-1.723	-0.032	0.243	1.266	1.523		
1	1000	1.272	2.350	-1.922	0.842	0.572	0.604	-0.914	1.266	-0.891	0.506	-0.042	-0.675
	5	1000	-4.637	-1.998	2.223	-5.670	-2.177	2.432	0.594	-0.347	4.810	5.759	0.096
		2.897	2.215	4.241	-0.271	4.368	4.260	0.284	2.212	-6.472	-7.605		
		-0.967	4.455	3.347	2.018	4.057	3.297	-0.643	-0.312	-3.811	-6.453		
		3.014	2.555	3.957	0.320	2.330	2.152	0.059	0.185	-4.165	-2.075		
		3.155	-3.262	-3.267	3.656	-2.011	-2.577	-0.563	1.114	4.279	2.800		
10	1000	-5.825	1.724	3.373	-6.442	1.694	2.145	-0.079	-0.553	3.409	4.264	-0.695	1.757
		-4.280	5.081	2.341	-4.878	3.768	1.646	0.600	0.273	0.634	1.340		
		1.869	-4.790	4.908	1.614	-4.844	4.362	-0.421	0.093	-2.936	-3.945		
		-0.082	-4.727	-5.327	-0.181	-5.137	-5.209	0.390	-2.542	6.693	9.276		
		-3.818	3.747	4.589	-3.675	3.789	4.814	-0.481	-0.047	-2.202	-1.610		
		-4.194	3.780	-4.198	-3.699	4.336	-3.374	0.426	0.093	2.374	3.602		
		5.165	0.929	4.601	5.389	1.420	3.980	0.441	0.098	-5.049	-4.303		
		-3.263	4.698	4.163	-1.790	4.331	3.297	0.467	-1.403	-4.799	-5.609		
		-4.197	5.651	-0.452	-4.639	5.626	-0.207	0.245	0.128	-2.809	-2.481		
		-5.024	0.542	-4.823	-4.954	0.821	-5.100	0.980	-0.263	2.285	1.309		

### B. 4: Feed-Forward Neural Network for COD - MODEL 4

The inputs for Model 4 are COD(k), COD(k-1), COD(k-2), COD(k-3), and the output is COD(k+1).

**Table B. 4: The initial inputs weights and biases for COD Model 4**

Hid neurons	Epochs	COD(k)	COD(k-1)	COD(k-2)	COD(k-3)	COD(k)	COD(k-1)	COD(k-2)	COD(k-3)	COD(k+1)	COD(k+1)	b(1)	b(1)	b(2)	b(2)
		Init inp weights	Init inp weights	Init inp weights	Init inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Init out weights	Final out weights	Initial inp bias	Final inp bias	Initial out bias	Final out bias
1	500	-1.110	-2.269	0.037	2.052	-0.668	-0.435	-0.364	-0.351	0.231	-0.867	0.686	-0.249	-0.907	-0.235
	5	1.679	-3.462	1.270	2.719	2.369	-3.916	1.346	3.257	-0.728	-1.028	-3.407	-3.468	-0.981	1.072
		4.056	1.165	0.728	2.150	3.691	-0.818	1.479	3.565	0.510	0.765	-5.587	-6.056		
		2.373	2.986	-2.903	1.005	3.089	2.886	1.369	1.687	-0.374	0.129	-1.857	-3.190		
		-0.089	-3.001	2.459	2.965	-0.394	-3.891	1.122	2.471	-0.073	1.478	-2.440	-2.035		
		3.423	-1.890	-0.106	-2.791	3.253	-1.705	-0.269	-1.305	-0.321	0.521	2.830	2.146		
10	500	1.432	3.579	-3.511	2.603	2.033	4.085	-3.227	1.564	0.561	-0.377	-4.689	-5.211	-0.960	-0.054
		3.236	-1.993	3.280	2.832	3.593	-2.385	2.778	1.631	-0.347	-0.566	-6.145	-6.838		
		4.829	0.661	-2.993	-0.367	4.572	1.206	-3.377	-0.514	0.477	0.424	-2.529	-3.667		
		-0.995	2.539	1.611	-4.783	0.126	2.203	0.817	-4.257	-0.125	-0.382	1.765	3.351		
		-0.808	2.943	4.604	-1.917	-2.082	3.305	3.949	-1.260	0.184	0.150	-2.445	-0.732		
		3.776	-3.527	-1.334	2.198	4.452	-2.836	-1.923	2.668	-0.771	0.098	-0.389	-0.332		
		1.257	-4.408	-3.344	1.481	2.016	-4.485	-2.520	1.052	-0.362	-0.073	3.619	3.473		
		4.200	3.401	-1.322	1.511	3.347	3.838	-2.271	1.618	0.244	0.032	-2.920	-2.886		
		3.295	-2.722	3.923	0.301	3.829	-1.668	4.220	0.328	0.789	0.329	-0.856	0.038		
		2.444	-3.733	1.735	3.252	2.960	-2.668	2.160	3.623	0.927	-0.061	0.327	1.753		
1	1000	-0.306	-1.111	2.224	2.079	0.601	0.390	0.330	0.316	0.897	1.506	-1.670	0.602	-0.097	-0.863
	5	0.228	-4.584	1.869	0.167	-0.218	-4.146	1.645	1.118	0.192	0.754	-0.890	-1.151	0.604	1.138
		3.209	0.414	3.466	-1.037	5.328	0.236	0.833	-2.184	0.564	0.176	-4.486	-4.477		
		-0.535	4.157	-0.455	2.518	-1.636	5.374	-2.112	-0.148	-0.933	0.575	-3.119	-0.591		
		-2.860	2.602	-2.095	2.062	-2.930	6.513	-2.871	-0.275	0.714	-0.412	-0.813	-0.375		
		4.222	1.114	1.094	-1.650	3.506	0.316	3.109	0.397	0.365	0.111	-0.588	-1.705		
10	1000	1.757	-4.003	3.903	-0.197	2.787	-4.110	3.433	-0.657	-0.040	-0.003	-3.428	-3.197	0.039	0.723
		2.891	-2.723	-1.579	-3.836	4.464	-3.587	-1.007	-2.638	0.934	-0.247	0.988	-0.362		
		3.109	-2.496	3.293	-2.579	3.715	-2.117	2.959	-2.872	0.567	0.130	-2.214	-2.676		
		2.918	3.172	-1.542	-3.495	4.813	0.027	-2.694	-3.160	-0.181	0.274	-1.348	-1.941		
		0.441	-0.935	2.875	-4.848	2.445	-0.577	-1.102	-4.314	0.915	-0.668	1.056	4.395		
		-0.191	0.358	-4.647	-3.438	-0.463	0.425	-3.479	-3.755	-0.606	0.137	4.240	4.369		
		-0.514	3.815	-3.169	3.016	0.970	4.879	-2.018	2.382	0.059	0.086	-2.493	-2.873		
		2.259	-2.243	2.561	4.043	2.124	-4.208	2.040	3.190	0.008	0.090	-2.408	-1.314		
		-3.789	1.295	-3.764	1.747	-3.039	0.514	-3.098	3.105	-0.033	-0.131	0.669	0.143		
		0.453	-3.483	4.543	-1.201	0.008	-3.127	4.546	-0.037	-0.245	0.362	2.197	2.818		

### B. 5: Feed-Forward Neural Network for COD - MODEL 5

The inputs for Model 5 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k) and the output is COD(k+1).

Table B. 5: The initial inputs weights and biases for COD Model 5

Hid neurons	Epochs	COD(k)	COD(k-1)	COD(k-2)	COD(k-3)	TKN(k)	COD(k)	COD(k-1)	COD(k-2)	COD(k-3)	TKN(k)	COD(k+1)	COD(k+1)	b(1)	b(1)	b(2)	b(2)
		Init inp weights	Init inp weights	Init inp weights	Init inp weights	Init inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Final out weights	Final out weights	Initial inp bias	Final inp bias	Initial out bias
1	500	2.066	-2.485	-0.029	-0.388	-0.835	0.154	0.102	0.138	0.098	0.114	0.827	1.229	1.042	0.130	-0.680	-0.042
5	500	-2.910	-0.809	2.052	0.616	2.484	-3.080	-0.530	3.647	0.560	1.438	-0.241	0.740	1.184	1.441	0.640	0.983
		1.876	2.866	1.871	-1.823	-1.660	3.359	1.439	-0.909	-0.515	-2.117	0.806	1.045	-2.783	-3.373		
		2.521	0.852	0.283	1.557	-3.182	1.919	0.148	-0.259	2.677	-3.029	0.887	-0.086	-1.191	-1.911		
		-3.108	1.211	-2.742	0.326	1.224	-3.036	1.393	-3.863	0.244	2.135	-0.052	0.418	0.672	-1.002		
		-3.067	-0.368	-1.625	-2.708	-0.756	-4.215	0.569	-1.060	-1.969	0.416	-0.964	-0.166	2.882	1.827		
10	500	-3.112	-0.676	-0.916	2.189	-3.209	-2.392	-0.050	-0.474	2.627	-2.526	0.904	0.542	5.418	6.238	-0.423	0.087
		1.251	1.605	-3.178	-1.509	3.233	0.742	1.160	-3.472	-1.926	2.527	-0.919	-0.364	-2.611	-3.208		
		-1.245	-0.559	-2.940	-3.134	2.568	-0.147	0.062	-3.016	-3.112	2.564	0.643	0.374	4.231	3.950		
		0.364	-1.927	2.751	2.677	2.948	0.355	-2.506	1.920	2.234	3.005	-0.553	0.673	-4.481	-5.045		
		-0.880	1.171	0.134	-4.022	-2.917	-0.637	1.180	0.091	-4.120	-2.661	0.999	0.068	3.915	4.180		
		-3.072	3.257	-2.634	-1.025	-0.154	-2.959	2.750	-3.259	-1.513	-0.646	-0.479	-0.063	1.633	1.883		
		-3.589	3.519	-0.675	-0.119	-1.445	-3.788	2.607	-0.233	-0.842	-2.797	-0.539	-0.190	0.408	0.240		
		0.786	2.861	2.310	2.163	-2.986	0.199	2.628	1.578	1.021	-2.249	0.050	0.181	-1.756	-2.929		
		1.019	4.020	-2.933	-0.768	-1.650	1.024	3.759	-3.086	-0.249	-1.645	0.121	0.263	1.703	2.081		
		-2.863	2.626	-1.641	2.694	1.559	-2.810	2.347	-1.566	1.681	2.150	0.652	0.617	-3.720	-3.553		
1	1000	0.031	1.099	-1.693	-1.239	2.261	-0.128	-0.088	-0.115	-0.081	-0.095	-0.531	-1.341	-0.313	-0.047	-0.744	0.065
5	1000	0.954	1.486	-0.095	-2.717	-3.123	-0.835	1.835	-0.356	-3.464	-1.461	-0.589	-0.449	0.007	0.773	0.470	1.529
		-1.517	0.154	-1.217	4.033	0.670	0.246	-1.755	-0.735	2.753	1.436	0.721	-0.419	-0.307	0.088		
		-0.472	3.721	-0.051	2.056	1.947	-0.163	4.031	-0.839	0.505	2.446	-0.485	0.019	-4.222	-4.111		
		-1.220	-2.141	3.002	-0.986	-2.213	-2.423	-0.432	4.359	0.460	-1.653	-0.445	-0.080	1.181	-1.190		
		-0.714	0.683	3.579	2.590	0.607	-2.735	0.495	0.582	2.594	5.294	-0.424	1.189	-5.724	-7.369		
10	1000	0.964	3.107	3.456	1.334	2.000	-1.716	2.244	3.535	2.860	3.024	-0.648	0.753	-8.405	-8.958	0.513	2.598
		-3.379	-1.488	3.168	1.806	-0.169	-4.846	-0.637	1.552	1.927	-0.510	-0.864	-1.078	1.848	4.209		
		-2.663	3.078	3.352	0.277	0.497	-5.120	1.919	4.305	0.919	-2.239	-0.381	0.357	-1.389	1.432		
		-1.357	-1.051	-0.462	-4.271	2.353	-2.291	-1.564	-0.062	-3.770	2.729	-0.330	0.154	3.521	3.541		
		-1.600	3.303	-2.918	-2.431	0.787	-2.131	1.648	-3.022	-1.576	0.339	-0.248	-0.291	1.706	1.478		
		2.743	0.181	2.137	3.027	-2.292	1.388	0.469	2.492	3.203	-2.597	0.904	0.103	-3.022	-3.032		
		0.230	2.211	3.195	2.436	-2.552	-0.221	2.224	4.310	2.324	-2.191	0.439	-0.113	-2.429	-2.610		
		1.528	-0.887	-2.871	-3.790	1.233	0.333	0.121	-3.482	-2.671	-0.868	0.559	0.160	3.960	4.564		
		2.332	3.158	-1.609	-0.091	3.091	1.764	3.283	-3.279	-0.048	0.916	0.235	0.136	-2.291	-1.709		
		3.040	-3.231	2.664	-0.539	-0.962	2.859	-2.756	3.771	-0.873	-1.664	0.298	-0.901	1.853	2.214		

### B. 6: Feed-Forward Neural Network for COD - MODEL 6

The inputs for Model 6 are COD(k), COD(k-1), COD(k-2), COD(k-3), TKN(k), FLOW(k) and the output is COD(k+1).

**Table B. 6: The initial inputs weights and biases for COD Model 6**

Hid neurons	Epochs	COD(k)	COD(k-1)	COD(k-2)	COD(k-3)	TKN(k)	FLOW(k)	COD(k)	COD(k-1)	COD(k-2)	COD(k-3)	TKN(k)	FLOW(k)	COD(k+1)	COD(k+1)	b(1)	b(1)	b(2)	b(2)
		Init inp weights	Init inp weights	Init inp weights	Init inp weights	Init inp weights	Init inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Final inp weights	Init out weights	Final out weights	Initial inp bias	Final inp bias	Initial out bias
1	500	-1.902	1.099	-0.933	0.860	-1.662	1.330	0.221	0.118	-0.115	0.186	0.123	0.149	0.437	0.729	0.722	-0.143	0.741	0.316
		0.928	-0.400	-2.283	-0.609	-3.177	1.842	0.172	-0.047	-0.525	-1.648	-3.287	0.502	0.088	-0.303	0.419	0.527	-0.292	0.145
5	500	-0.172	2.464	-1.862	-0.015	-1.983	2.561	1.523	1.830	-2.003	1.833	-2.470	0.154	-0.387	0.069	0.387	-0.768		
		1.085	-2.624	-0.035	-2.082	2.232	1.375	2.904	-1.737	0.270	-1.340	2.399	1.494	0.345	0.293	0.069	0.334		
10	500	1.911	1.525	1.252	0.301	-1.998	-2.659	2.171	1.719	2.424	-0.176	-3.029	-4.471	-0.309	-1.001	0.631	4.214		
		-1.608	1.664	2.886	-0.342	2.630	-0.893	-2.486	1.347	2.431	0.230	2.501	-0.759	0.961	-0.763	-4.531	-4.552		
1	1000	1.396	2.665	-1.588	-2.155	2.400	-1.701	2.087	1.327	-2.243	-2.317	2.020	0.317	-0.637	-1.202	-2.714	-2.896	-0.165	0.230
		0.777	2.120	-1.571	-0.945	2.761	2.855	1.562	2.794	-0.998	-1.259	2.578	2.373	0.559	0.552	-5.019	-5.717		
5	1000	2.255	2.118	-1.148	1.267	2.189	-2.528	2.569	2.562	-1.256	-0.622	1.957	-2.826	0.619	0.729	-3.505	-4.225		
		-0.289	2.241	-0.223	-3.582	-2.451	-0.446	-0.502	2.124	0.107	-3.451	-2.865	-0.691	0.447	0.116	3.250	3.815		
10	1000	0.092	-0.533	-2.474	0.327	3.511	-2.502	0.774	-0.720	-1.563	0.587	3.798	-2.134	-0.909	0.094	0.773	1.070		
		-0.560	-0.521	4.348	0.450	3.044	0.901	-0.522	-0.698	4.050	0.257	2.693	0.812	0.227	0.028	-4.781	-5.077		
1	500	-0.598	-2.155	-1.745	1.253	-3.449	1.808	-0.560	-2.862	-1.526	1.416	-3.266	1.701	-0.330	0.044	2.307	1.916		
		-0.799	-3.830	2.531	0.624	-1.608	-1.569	-1.382	-2.851	2.824	-0.335	-2.226	-1.906	0.168	-0.052	1.518	1.933		
5	500	2.540	-0.975	3.585	1.380	-2.300	0.194	2.737	-0.914	3.619	1.321	-1.937	0.587	0.826	0.135	-0.990	-0.312		
		2.403	2.770	-2.325	2.056	0.834	-1.395	2.437	1.748	-3.123	1.968	1.271	-1.159	0.003	0.031	-0.347	-1.079		
10	500	1.487	-1.554	1.770	0.354	-1.804	-0.938	-0.162	-0.085	0.083	-0.136	-0.090	-0.109	-0.753	-1.050	0.381	-0.056	0.762	0.147
		1.302	-2.413	2.219	0.184	2.227	-1.788	-0.151	-1.426	2.416	-1.311	0.903	-2.289	0.525	0.242	-2.854	-1.872	-0.057	2.135
1	1000	-2.319	-0.036	3.311	-1.609	1.723	0.276	-2.428	-2.366	0.738	0.052	-0.120	1.900	-0.857	-0.451	-0.068	3.742		
		-1.040	0.684	2.933	-1.108	1.528	-2.704	-3.554	0.440	3.797	-1.762	-4.091	-0.752	0.436	-0.101	-0.389	1.118		
5	1000	-0.869	-0.752	-0.727	-2.745	2.258	-2.032	1.963	0.092	-3.006	-2.387	0.806	-2.810	-0.719	-0.140	1.813	1.057		
		1.478	-1.548	0.961	-2.402	-1.810	-2.072	3.022	-1.260	1.465	-0.651	-2.493	-5.745	0.231	-1.205	4.855	7.397		
10	1000	0.282	2.109	-2.197	3.996	0.254	0.623	0.273	2.454	-2.287	3.344	0.238	1.021	0.796	-0.467	-4.804	-5.017	0.974	0.073
		2.048	2.377	2.922	-0.210	2.518	1.351	2.292	2.796	3.136	0.182	2.457	1.664	0.506	-0.078	-8.072	-7.686		
1	500	-2.178	1.440	1.109	2.869	-2.321	1.444	-4.478	1.434	1.295	2.425	-1.270	0.570	-0.695	0.164	-0.226	0.854		
		1.477	1.721	3.889	1.153	-2.163	1.531	0.637	-0.303	5.029	-0.333	-1.238	3.177	0.431	-0.298	-5.191	-4.242		
5	500	-0.885	1.140	0.134	-3.712	0.542	-2.663	-1.843	-0.236	0.030	-1.125	-1.710	-2.068	0.683	0.450	3.165	4.519		
		2.848	-2.591	3.300	-1.110	-0.246	-0.609	1.851	-2.154	3.792	-1.302	-0.149	0.231	0.567	-0.342	-0.757	-0.547		
10	500	-2.444	-1.418	-1.974	-2.460	-0.630	-2.441	-2.877	-2.359	-1.427	-0.594	-1.502	-4.015	0.996	-0.665	5.833	8.529		
		-1.246	1.822	-1.726	-2.419	-2.584	2.037	-1.415	1.543	-2.567	-1.172	-3.196	2.182	0.172	-0.205	1.186	1.335		
1	1000	0.939	-0.802	3.816	2.038	-1.233	2.503	3.252	-0.810	4.073	-0.447	0.076	2.123	-0.278	0.442	-2.606	-3.908		
		1.373	1.341	2.021	1.012	3.741	-1.407	1.656	1.186	2.136	1.921	3.140	-2.041	-0.252	-0.222	-2.681	-2.804		

## **APPENDIX C**

## APPENDIX C

### SOFTWARE ROUTINES

This section from C.1 to C.11 presents all the implemented MATLAB software algorithms that are used in the development of the neural network prediction routines for prediction of the influent variables of COD, TKN and flow rate. The three NN topologies are the multilayer perceptron feed-forward NN, the Elman recurrent NN, and the radial basis function NN. The descriptions of the different script files are presented in Chapter 7, Table 7.2.

#### C. 1: MATLAB script – *Open\_all\_data.m*

```
% Establishes a connection to the MYSQL database and
% retrieves ALL plant and weather data.
% The DATABASE toolbox for use with MATLAB is used
%
% conn      - Returns a database connection object
% curs     - Returns a cursor object
% m        - Returns the data at the cursor position
% cod      - Chemical oxygen demand - WWTP variable
% tkn      - Total khejdahl nitrogen - WWTP variable
% flow     - Influent flow rate - WWTP variable
% mintemp  - Minimum temperature - weather variable
% maxtemp  - Maximum temperature - weather variable
% rain     - Rainfall - weather variable
% wind     - Wind speed - weather variable
% month    - Month data - weather variable
% database - MATLAB function to connect to an ODBC database
% exec     - MATLAB function to Execute a SQL statement
%          - and open the Cursor
% fetch    - MATLAB function to import data into MATLAB
% cell2mat - MATLAB function to convert the contents of a
%          - cell array into a single matrix
%
% Cape Peninsula University of Technology
% C.Kruger
% December 2007

% Establishing the connection with the DB
% Inserting the username and password for the DB
%
% Using the SQL script to select the COD data from
% a table named 'plant_weather_data'
conn = database('athlone', 'root', 'admin');
curs = exec(conn, 'select COD from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
cod = cell2mat(m); % Converting the cell array to a matrix
cod = cod';
clear m;
```

```

% Retrieve all TKN data from database
curs = exec(conn, 'select TKN from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
tkn = cell2mat(m);
tkn = tkn';
clear m;

% Retrieve all Flowrate data from database
curs = exec(conn, 'select Flow from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
flow = cell2mat(m);
flow = flow';
clear m;

% Retrieve all min_temp data from database
curs = exec(conn, 'select Mintemp from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
mintemp = cell2mat(m);
mintemp = mintemp';
clear m;

% Retrieve all max_temp data from database
curs = exec(conn, 'select Maxtemp from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
maxtemp = cell2mat(m);
maxtemp = maxtemp';
clear m;

% Retrieve all rain data from database
curs = exec(conn, 'select Rain from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
rain = cell2mat(m);
rain = rain';
clear m;

% Retrieve all wind data from database
curs = exec(conn, 'select Wind from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
wind = cell2mat(m);
wind = wind';
clear m;

% Retrieve all month of the year data from database
curs = exec(conn, 'select Sin,Cos from plant_weather_data');
curs = fetch(curs);
m = curs.Data;
month = cell2mat(m);
month = month';
clear m;

% Close all open database connections and clear variables
close (conn);
clear conn;
clear curs; % End

```



## C. 2: MATLAB script – *Correlation\_Coefficients.m*

```
% Calculation of the Correlation Coefficients between the input
% variables
%
%   corrcoef    -   MATLAB function to calculate the correlation
%                   coefficients
%   All variables are normalized (scaled). This does not affect
%   the
%   inherent relationship existing between variables
%
%   Cape Peninsula University of Technology
%   C.Kruger
%   Dec 2007

%Correlation between COD and TKN
cod_tkn_corr=corrcoef(cod,tkn)

%Correlation between COD and Flow
cod_flow_corr=corrcoef(cod,flow)

%Correlation between COD and maxtemp
cod_maxtemp_corr=corrcoef(cod,maxtemp)

%Correlation between COD and mintemp
COD_mintemp_corr=corrcoef(cod,mintemp)

%Correlation between COD and rain
COD_rain_corr=corrcoef(cod,rain)

%Correlation between COD and wind
COD_wind_corr=corrcoef(cod,wind)

%Correlation between TKN and Flow
tkn_flow_corr=corrcoef(tkn,flow)

%Correlation between TKN and maxtemp
tkn_maxtemp_corr=corrcoef(tkn,maxtemp)

%Correlation between TKN and mintemp
tkn_mintemp_corr=corrcoef(tkn,mintemp)

%Correlation between TKN and Rain
tkn_rain_corr=corrcoef(tkn,rain)

%Correlation between TKN and wind
tkn_wind_corr=corrcoef(tkn,wind)

%Correlation between FLOW and maxtemp
flow_maxtemp_corr=corrcoef(flow,maxtemp)

%Correlation between FLOW and mintemp
flow_mintemp_corr=corrcoef(flow,mintemp)

%Correlation between FLOW and Rain
flow_rain_corr=corrcoef(flow,rain)

%Correlation between FLOW and wind
```

```
flow_wind_corr=corrcoef(flow,wind)

%Correlation between maxtemp and mintemp
maxtemp_mintemp_corr=corrcoef(maxtemp,mintemp)

%Correlation between maxtemp and Rain
maxtemp_rain_corr=corrcoef(maxtemp,rain)

%Correlation between maxtemp and wind
maxtemp_wind_corr=corrcoef(maxtemp,wind)

%Correlation between mintemp and Rain
mintemp_rain_corr=corrcoef(mintemp,rain)

%Correlation between mintemp and wind
mintemp_wind_corr=corrcoef(mintemp,wind)

%Correlation between rain and wind
rain_wind_corr=corrcoef(rain,wind)

% End
```

### C. 3: MATLAB script – *Preprocess.m*

```
% Preprocessing of all inputs
% The month data is already processed by means of a sin/cos pair
% The values range from -1 to +1
%
%   Scaling of all input variables with their MAXIMUM value
%   COD      / 2964
%   TKN      / 116.5
%   Flow     / 200
%   mintemp  / 19.357
%   rain     / 21.443
%   wind     / 24.729
%
%   cod_norm      -   COD value scaled by its maximum value
%   tkn_norm      -   TKN value scaled by its maximum value
%   flow_norm     -   FLOW value scaled by its maximum value
%   mintemp_norm  -   Minimum temperature value scaled by its
%                   maximum value
%   rain_norm     -   Rainfall value scaled by its maximum value
%   wind_norm     -   Wind speed value scaled by its maximum
%                   value
%
%   Cape Peninsula University of Technology
%   C.Kruger
%   December 2007

cod_norm = cod/max(cod);
tkn_norm = tkn/max(tkn);
flow_norm = flow/max(flow);
mintemp_norm = mintemp/max(mintemp);
rain_norm = rain/max(rain);
wind_norm = wind/max(wind);

% End
```

#### C. 4: MATLAB script – raw\_norm\_inputs.m

```
% Determining the inputs to the NN
%
% 11 - Inputs - [COD, (COD-1), (COD-2), (COD-3), TKN, Flow,
% Month(sin/cos),
% Mintemp, Rain, Wind]
% 1 - target - [COD + 1]

% C.Kruger
% Cape Peninsula University of Technology
% December 2007

% COD-1 is declared (654)-start at index 3 ->654
g = 3;
for i =1:654
    cod_1(1,i) = cod_norm(g);
    i = i+1;
    g = g+1;
end

% COD-2 is declared (654)-start at index 2 ->654
j = 2;
for k =1:654
    cod_2(1,k) = cod_norm(j);
    k = k+1;
    j = j+1;
end

% COD-3 is declared (654)-start at index 1 ->654
l = 1;
for m =1:654
    cod_3(1,m) = cod_norm(l);
    l = l+1;
    m = m+1;
end

% Specifying the raw inputs which are used

rawinput = [cod_norm; tkn_norm; flow_norm; month; mintemp_norm; rain_norm;
wind_norm];

% The inputs are declared (654) - Start at index 4 -> 657
for a = 1:8
    h=4;
    for b = 1:654
        p(a,b) = rawinput(a,h);
        h = h+1;
    end
    a = a+1;
end

% Adding delayed inputs for COD
o = a;
for q = 1:654
    p(o,q) = cod_1(1,q);
```

```

    q = q+1;
end
o = a+1;
for q = 1:654
    p(o,q) = cod_2(1,q);
    q = q+1;
end
o = a+2;
for q = 1:654
    p(o,q) = cod_3(1,q);
    q = q+1;
end

% The target COD+1 is declared (654) - Start at index 5 ->658
d = 5;
for c = 1:654
    t(1,c) = rawinput(1,d);
    c = c+1;
    d = d+1;
end

% Clearing all variables not needed
clear a;
clear b;
clear c;
clear d;
clear e;
clear f;
clear g;
clear h;
clear i;
clear j;
clear k;
clear l;
clear m;
clear n;
clear o;
clear q;

% End

```

### C. 5: MATLAB script – *Train\_tst\_data.m*

```
% Partitioning the data into a TRAINING and TEST set
% Training set - 2/3 of total input data set
% Validation set - 1/3 of total input data set
%
% iitst - Test set generated by every 3rd sample
% iitr - Training set generated from the
%       1st and 2nd samples
% ptr - Input vector for the TRAINING data set
% ttr - Target vector for the TRAINING data set
% tst_p - Input vector for the TEST data set
% tst_t - Target vector for the TEST data set
%
% The TRAINING set has 436 data points
% The TEST set has 218 data points
%
% Cape Peninsula University of Technology
% C.Kruger
% December 2007

iitst = 3:3:654; % Test set generated from every 3rd sample
iitr = [1:3:654 2:3:654]; % Training set generated from 1st and 2nd
samples
% Training data randomized

ptr = p(:,iitr); % Input training set
ttr = t(:,iitr); % Output target set

tst_p = p(:,iitst); % Input test set
tst_t = t(:,iitst); % Output target set

% End
```

## C. 6: MATLAB script – *Feedforward.m*

```
% Design a feedforward NN with 1 input, 1 layer of hidden neurons
% with 3 neurons and 1 target output neuron
% Use a tan-sigmoid transfer function for the input and hidden
% layer and a pure linear transfer function for the output layer
% Train the network using the Scaled Conjugate Gradient algorithm

% C.Kruger
% Cape Peninsula University of Technology
% December 2007

neurons = 10      % Number of hidden layer neurons

net = newff([minmax(ptr)],[neurons 1 ],{'tansig','purelin'},'trainscg');
net.layers{1}.initFcn='initnw';
net.layers{2}.initFcn='initnw';
net = init(net);           % Initialization of network
net.performFcn='mse';
net.trainParam.epochs = 1000;   % Maximum no. of epochs
net.trainParam.show = 50;       % No. of epochs to display
net.trainParam.lr = 0.6;        % Learning rate
net.trainParam.mc = 0.5;       % Momentum term

init_in_weights = net.IW{1,1}; % Determining the initial input weights
init_out_weights = [net.LW{2,1}(:)]; % Determining the initial output
weights
init_in_bias = net.b{1};        % Determining the initial input bias
init_out_bias = net.b{2};       % Determining the initial output bias

% Training the network with Scaled conjugate gradient algorithm
[net,tr] = train (net,ptr,ttr);

% Saving of figure and training data
saveas(gcf, 'trainscg_err', 'fig');
save tr tr;

in_weights = net.IW{1,1};       % Determining the final input weights
out_weights = [net.LW{2,1}(:)]; % Determining the final hidden weights
in_bias = net.b{1};             % Determining the final input bias
out_bias = net.b{2};            % Determining the final output bias

% Simulating the network with the same inputs used to train the network
ttrsim = sim (net,ptr);

% Plotting input and target
figure(1)
hold on
title('Training inputs (normalized) and target output');
xlabel('Time in weeks');
ylabel('COD normalized');
plot(ttr, 'o', 'markersize',3);
plot(ttrsim, '*', 'markersize',3);
legend ('Training-Data', 'Network-Response');
hold off
saveas(gcf, 'Train_inp_targ', 'fig')
Y=[ttr(:),ttrsim(:)];
```

```

% Determining the minimum training error and epochs
min_tr_err=tr.perf(1,size(tr.perf));
min_tr_err = min_tr_err(1,2)

epochs=tr.epoch(1,size(tr.epoch));
epochs=epochs(1,2)

% Simulating by applying the test set
tstsim = sim (net,tst_p);

% Plotting test set input and test set expected outputs
figure(2)
hold on
title('Test inputs (normalized) and network response');
xlabel('Time in weeks');
ylabel('COD normalized');
plot(tst_t, 'o', 'markersize',3);
plot(tstsim, '*', 'markersize',3);
legend ('Test-Data', 'Network-Response');
hold off
saveas(gcf, 'Test_inp_net_resp', 'fig')
X=[tst_t(:),tstsim(:)];

% End

```



### C. 7: MATLAB script – RBF.m

```
% Design a RBF NN with 1 input, 1 layer of hidden neurons
% and 1 target output neuron.
% Use a Gaussian transfer function for the input and hidden layer
% and a pure linear transfer function for the output layer.
% Plot the inputs and targets for the TRAINING and TEST data sets
%
%
% goal          - Error goal for the SSE performance
%               function
% spread        - Spread constant for the Gaussian function
% net           - Created Radial basis function NN
% tr            - Training performance and epochs
% in_weights    - weights between the input and hidden layer
%               after training
% out_weights   - Weights between the hidden and output
%               layer after training
% in_bias       - Bias at the input after training
% out_bias      - Bias at the output after training
% ptr           - Input vector for the TRAINING data set
% ttr           - Target vector for the TRAINING data set
% ttrsimsim     - NN output after simulating with the
%               TRAINING inputs
% epochs        - Number of training iterations
% min_tr_err    - Minimum training error
% tst_p         - Input vector for the TEST data set
% tst_t         - Target vector for the TEST data set
% tstsim       - NN output after simulating with the TEST
%               inputs
%
% Cape Peninsula University of Technology
% C.Kruger
% December 2007

goal = 1.7;          % Error goal for SSE
spread = 0.5;       % Spread constant for Gaussian function

% Creating a new radial basis function network

[net,tr] = newrb(ptr,ttr,goal,spread);
saveas(gcf, 'trainscg_err', 'fig');
save tr tr;

in_weights = net.IW{1,1};          % Determining the final input weights
out_weights = [net.LW{2,1}(:)];    % Determining the final hidden weights
in_bias = net.b{1};                % Determining the final input bias
out_bias = net.b{2};                % Determining the final output bias

% Simulating the network with the same inputs used to train the network
ttrsimsim = sim (net,ptr);

%Plotting input and target
figure(1)
hold on
title('Training inputs (normalized) and target output');
xlabel('Time in weeks');
ylabel('FLOW normalized');
```

```

plot(ttr, 'o', 'markersize',3);
plot(ttrsim, '*', 'markersize',3);
legend ('Training-Data', 'Network-Response');
hold off
saveas(gcf, 'Train_inp_targ', 'fig')
Y=[ttr(:),ttrsim(:)];

%Determining the minimum training error and epochs
min_tr_err=tr.perf(1,size(tr.perf));
min_tr_err = min_tr_err(1,2)

epochs=tr.epoch(1,size(tr.epoch));
epochs=epochs(1,2)

% Simulating by applying the test set
tstsim = sim (net,tst_p);

% Plotting test set input and test set expected outputs
figure(2)
hold on
title('Test inputs (normalized) and network response');
xlabel('Time in weeks');
ylabel('FLOW normalized');
plot(tst_t, 'o', 'markersize',3);
plot(tstsim, '*', 'markersize',3);
legend ('Test-Data', 'Network-Response');
hold off
saveas(gcf, 'Test_inp_net_resp', 'fig')
X=[tst_t(:),tstsim(:)];

% End

```

## C. 8: MATLAB script – *Postprocess.m*

```
% Denormalize the inputs and perform a linear regression between
% inputs and expected target output

% C.Kruger
% Cape Peninsula University of Technology
% December 2007

net_out = tstsim*max(cod);           % Denormalizing) network output
expected_target_out=tst_t*max(cod); % Denormalizing expected target

%Absolute percentage error
APE=(abs(expected_target_out-net_out)/(expected_target_out)*100)

% Linear regression (Correlation) calculated and plotted
figure(3)
hold on
[m,b,r] = postreg(net_out,expected_target_out);
title('Linear correlation coefficient - Validation data ');
hold off
saveas(gcf, 'postregression', 'fig');

% Calculating the error on the test set and plotting it
err=(net_out-expected_target_out);

figure (4)
plot(err)
hold on
title('Test Set Error Plot');
hold off
avg_tst_err=mean(err)
saveas(gcf, 'Test_err', 'fig')

%Plotting net output and predicted output
figure (5)
set(gcf, 'color', 'white');
plot(net_out,'b.-','DisplayName', 'net_out', 'YDataSource', 'net_out');
hold on
title('Network response to Validation Set', 'FontSize', 12, 'FontWeight',
'Bold');
xlabel('Time in weeks', 'FontSize', 12, 'FontWeight', 'Bold');
ylabel('COD denormalized (mg/l)', 'FontSize', 12, 'FontWeight', 'Bold');
plot(expected_target_out, 'rx-', 'markersize',6, 'DisplayName',
'expected_target_out', 'YDataSource', 'expected_target_out');
legend('Actual Predicted out','Target output');
hold off
saveas(gcf, 'Test_pred_out', 'fig')

% Linear regression (Correlation) calculated and plotted
figure(6)
hold on
[m1,b1,r_tr] = postreg(ttr,ttrsims);
r_tr
r
title('Linear correlation coefficient - Training data ');
hold off
saveas(gcf, 'postregression_train', 'fig');
```

```

% Saving initial weights
save init_out_weights init_out_weights
save init_in_weights init_in_weights

%Saving initial bias
save init_in_bias init_in_bias
save init_out_bias init_out_bias

% Saving final weights
save in_weights in_weights
save out_weights out_weights

%Save final bias
save in_bias in_bias
save out_bias out_bias

% Saving all data variables to an Excel spreadsheet
txt=[char('Hid layers','Hid neurons','Train algorithm', 'Input
activation', 'Output activation','Epochs','Performance','Min train
err','Learn rate','Momentum','r_train',
'r_test','R_sq_train','R_sq_test','Avg_test_err','APE')];
txt = cellstr(txt)';

Data.hid=(net.numlayers-1);
Data.hid_neurons = neurons;
Data.train='trainscg';
Data.activ_in= 'tansig';
Data.activ_out='purelin';
Data.epoch=epochs;
Data.perf='MSE';
Data.min_err = min_tr_err;
Data.lr=net.trainParam.lr;
Data.mc=net.trainParam.mc;

outp_weights=[init_out_weights,out_weights];
Weights=[init_in_weights,in_weights,outp_weights,init_in_bias,in_bias];
Out_bias=[init_out_bias,out_bias];

txt1=[char('Init inp weights','Final inp weight','Init out weights','Final
out
weights','Init_in_bias','Fin_in_bias','Init_out_bias','Fin_out_bias')];
txt1 = cellstr(txt1)';

R_squared_train = (r_tr*r_tr);
R_squared_test = (r*r);

Result = [r_tr; r; R_squared_train; R_squared_test; avg_tst_err;APE]';

data=struct2cell(Data)';

SUCCESS = xlswrite('Data.xls',txt, 'Sheet1', 'A1');
SUCCESS = xlswrite('Data.xls',data, 'Sheet1', 'A2');
SUCCESS = xlswrite('Data.xls',Result, 'Sheet1', 'K2');
SUCCESS = xlswrite('Weights.xls',Weights, 'Sheet1', 'A2');
SUCCESS = xlswrite('Weights.xls',txt1, 'Sheet1', 'A1');
SUCCESS = xlswrite('Weights.xls',Out_bias,'Sheet1', 'AA2');

% End

```

## C. 9: MATLAB script – *Elman.m*

```
% Design an Elman RNN with 1 input, 1 layer of hidden neurons with
% 5 neurons and 1 target output neuron
% Use a tan-sigmoid transfer function for the input and hidden
% layer and a pure linear transfer function for the output layer
% Train the network using the Scaled Conjugate Gradient algorithm

net = newelm([minmax(ptr)], [5 1], {'tansig', 'purelin'}, 'trainscg');
net = init(net); % Initialization of network
net.performFcn='mse';
net.trainParam.epochs = 450; % Maximum no. of epochs
net.trainParam.show = 10; % No. of epochs to display
net.trainParam.lr = 0.001;
net.trainParam.lr_inc = 1;
net.trainParam.lr_dec = 0.1;
net.trainParam.mc = 0.5;

net.IW{1,1}=rand(size(net.IW{1,1})); %Random initialisation of input
weights
net.LW{2,1}=rand(size(net.LW{2,1})); % Random initialisation of hidden
layer1 weights

init_in_weights = net.IW{1,1}; % Determining the initial input weights
init_hid1_weights = net.LW{2,1}; % Determining the initial hidden weights

% Training the network with Scaled conjugate gradient algorithm
net = train (net,ptr,ttr);
saveas(gcf, 'trainscg_err', 'fig');
in_weights = net.IW{1,1}; % Determining the final input weights
hid1_weights = net.LW{2,1}; % Determining the final hidden
weights

% Simulating the network with the same inputs used to train the network
ttrsimsim = sim (net,ptr);

%Plotting input and target
figure(1)
hold on
title('Training inputs (normalized) and target output');
xlabel('Epochs');
plot(ttr, 'o', 'markersize',3);
plot(ttrsimsim, '*', 'markersize',3);
hold off
saveas(gcf, 'Train_inp_targ', 'fig')
Y=[ttr(:),ttrsimsim(:)];

R_train = corrcoef(ttr,ttrsimsim) % Linear regression (correlation)on
input set
[m,b,r_tr] = postreg(ttr,ttrsimsim);

% Simulating by applying the test set
tstsim = sim (net,tst_p);
```

```
% Plotting test set input and test set expected outputs
figure(2)
hold on
title('Test inputs (normalized) and network response');
xlabel('Data points');
plot(tst_t, 'o', 'markersize',3);
plot(tstsim, '*', 'markersize',3);
hold off
saveas(gcf, 'Test_inp_net_resp', 'fig')
X=[tst_t(:),tstsim(:)];

R_test = corrcoef(tst_t,tstsim)    % Linear regression on test set

% End of Elman Recurrent Neural Network program
```

### C. 10: MATLAB script – *Activation.m*

```
% The various activation functions are listed below
% together with a graphical plot.
%
% The functions "tansig, logsig, hardlim, hardlims and purelin"
% are MATLAB functions
%
% C.Kruger
% Cape Peninsula University of Technology
% December 2007

% Plot of the hyperbolic tangent function
figure (1)
x=linspace(-5,5,501);
y=tansig(x);
plot(x,y)
title('tansig')
xlabel('v')
ylabel('a')

% Plot of the logarithmic sigmoidal function
figure (2)
x=linspace(-5,5,501);
y=logsig(x);
plot(x,y)
title('logsig')
xlabel('v')
ylabel('a')

% Plot of the positive threshold or hard limit function
figure (3)
x=linspace(-5,5,501);
y=hardlim(x);
plot(x,y)
title('hardlim')
xlabel('v')
ylabel('a')

% Plot of the symmetrical or bipolar threshold or hard limit function
figure (4)
x=linspace(-5,5,501);
y=hardlims(x);
plot(x,y)
title('hardlims')
xlabel('v')
ylabel('a')

% Plot of the linear function
figure (5)
x=linspace(-5,5,501);
y=purelin(x);
plot(x,y)
title('purelin')
xlabel('v')
ylabel('a')
```

```
% Plot of the radial basis function
figure (6)
x=(-5:0.1:5);
y=radbas(x);
plot(x,y)
title('radbas')
xlabel('v')
ylabel('a')
```

```
% End
```



### C. 11: MATLAB script – *std\_min\_max.m*

```
% Calculation of standard deviation, min and max values
%
% minmax - MATLAB function to calculate the minimum and
%         maximum values from the rows of a matrix
% mean   - MATLAB function for calculation of the average or
%         mean value
% std    - MATLAB function to calculate the standard
%         deviation
%
% Cape Peninsula University of Technology
% C.Kruger
% Dec 2007

% Minimum and maximum value calculations
cod_minmax=minmax(cod)
tkn_minmax=minmax(tkn)
flow_minmax=minmax(flow)
maxtemp_minmax=minmax(maxtemp)
mintemp_minmax=minmax(mintemp)
rain_minmax=minmax(rain)
wind_minmax=minmax(wind)

% Mean calculations
cod_mean=mean(cod)
tkn_mean=mean(tkn)
flow_mean=mean(flow)
maxtemp_mean=mean(maxtemp)
mintemp_mean=mean(mintemp)
rain_mean=mean(rain)
wind_mean=mean(wind)

% Standard Deviation Calculations
cod_std=std(cod)
tkn_std=std(tkn)
flow_std=std(flow)
maxtemp_std=std(maxtemp)
mintemp_std=std(mintemp)
rain_std=std(rain)
wind_std=std(wind)

% End
```

## APPENDIX D

## APPENDIX D

### TECHNICAL WIRING OF THE PLC AND SENSORS

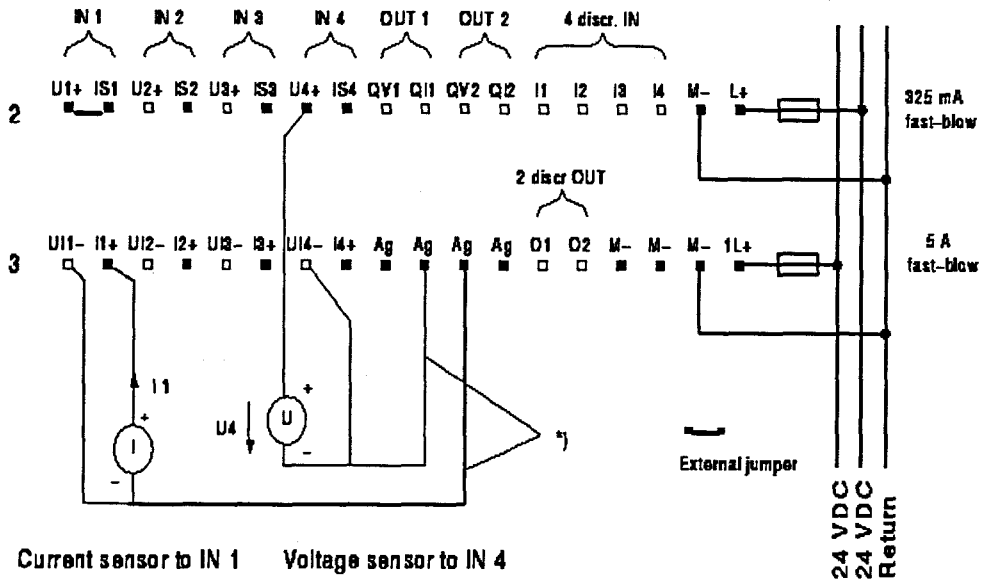
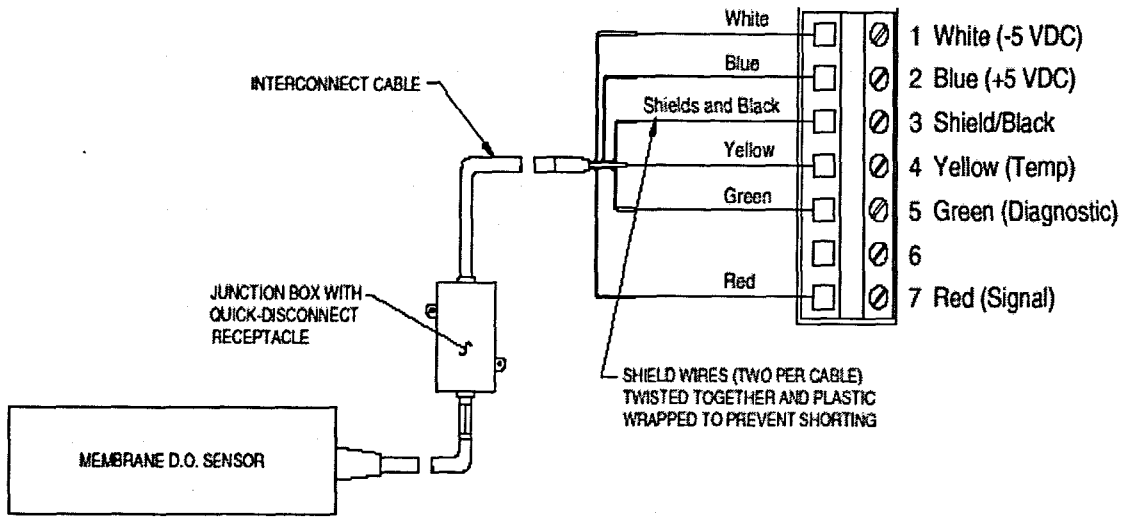


Figure D. 1: The Modicon PLC I/O Base wiring diagram

Description	Address Range
Digital input	10001 - 10016
Digital Output	00001 - 0016
Analogue Input	30001 - 30016
Analogue Output	40001 - 40016
Internal Coils 'Status Bits'	00017-01536
Internal Holding Registers	40017-41980



**Figure D. 2: The Dissolved Oxygen sensor wiring diagram**

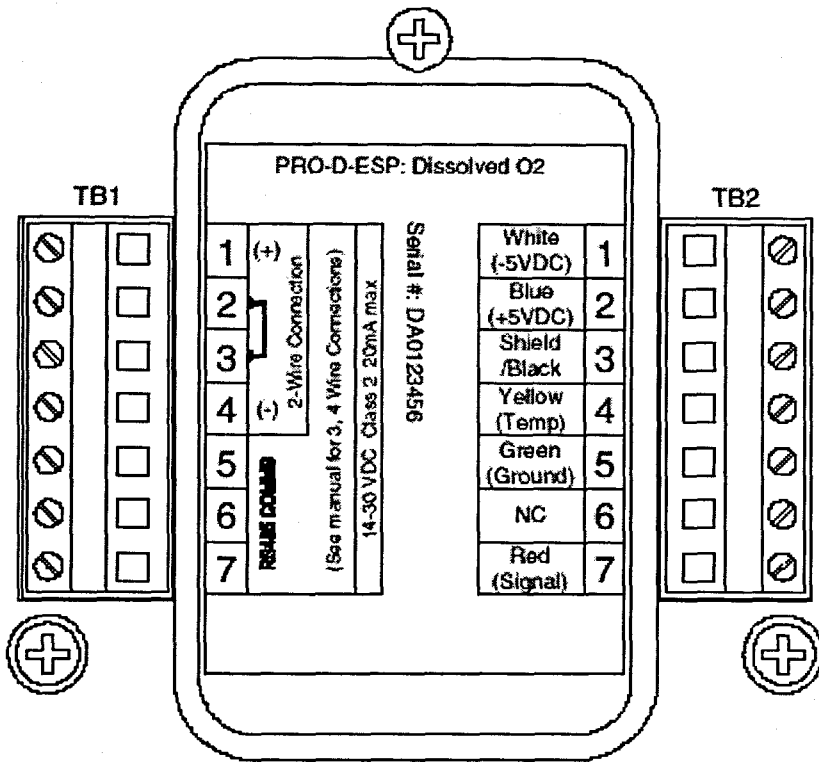


Figure D. 3: The Dissolved Oxygen transmitter wiring connector

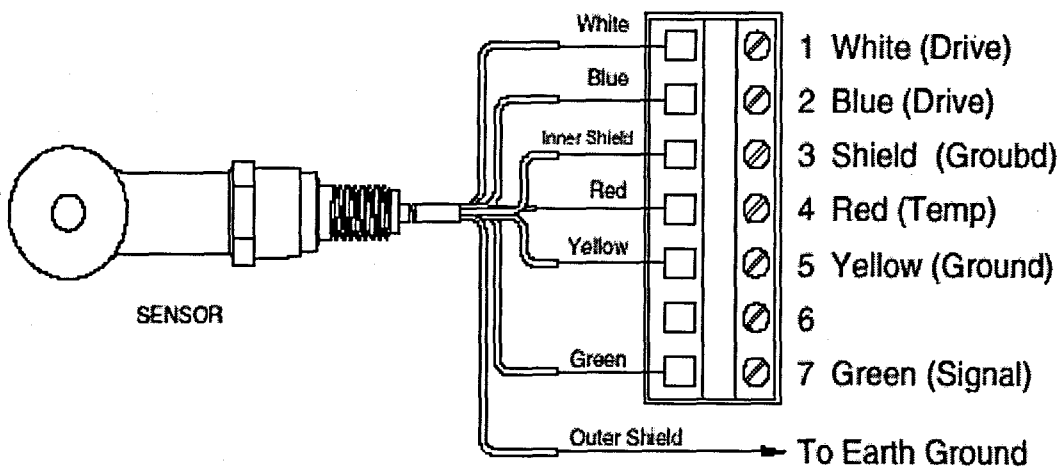


Figure D. 4: The pH sensor wiring diagram

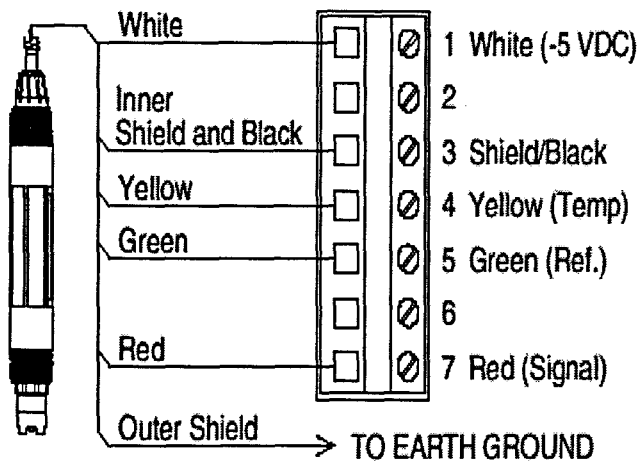


Figure D. 5: The Conductivity sensor wiring diagram

# APPENDIX E

## APPENDIX E

### DATABASE AND CONVERSION INTO MySQL FORMAT

This section presents the conversion of the ASCII text files and MS Excel files into the MySQL database format. The procedure for setting up of the MySQL database is also described.

#### 1 Conversion of the ASCII text files into MS Excel

The procedure describes how the conversion from ASCII text into MS Excel is done.

- Open MS Excel
- Click *File, Open*
- To display the text files select *All Files* at the bottom of Figure E.1.
- Select Open

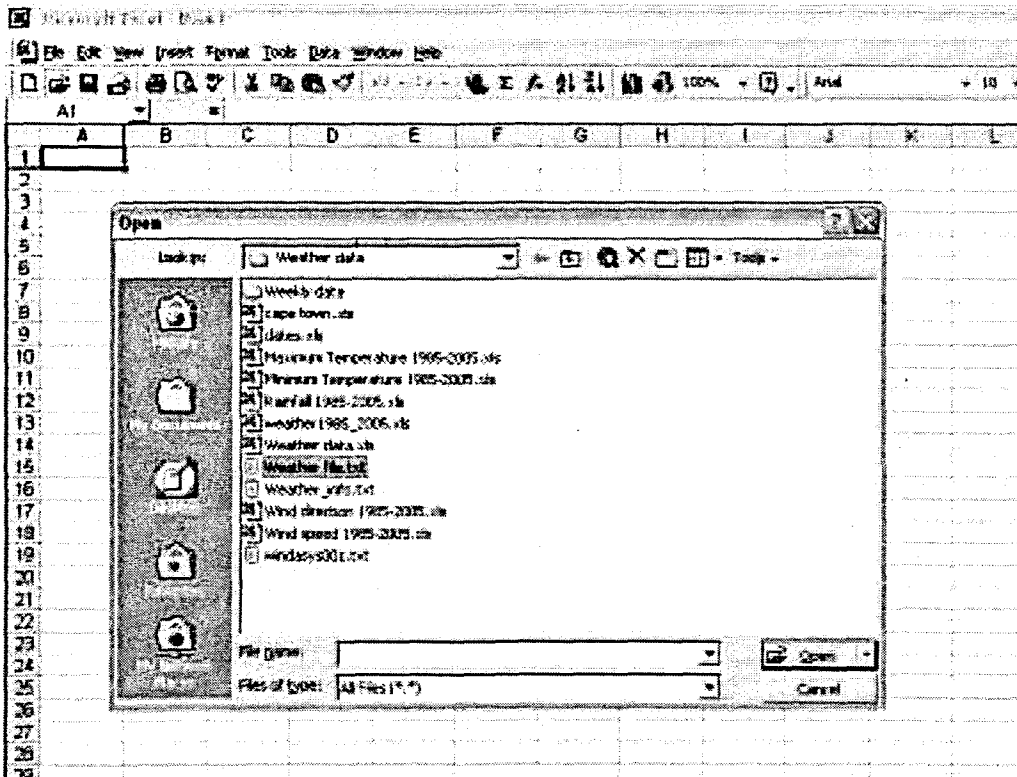


Figure E. 1: Load the text file into MS Excel

The Text Import Wizard window – Step 1 of 3 appears (Figure E.2)



- Select the *Delimited* radio tab
- Click *Next*
- Select the *Other* option and enter the ' | ' character in the space provided (Figure E.3)
- Click *Finish*

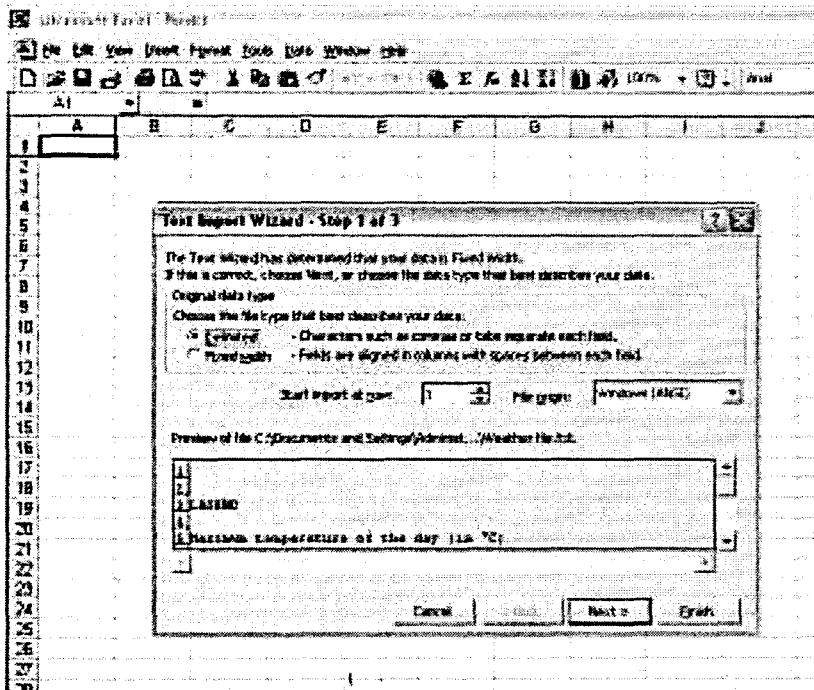


Figure E. 2: Text Import Wizard step 1

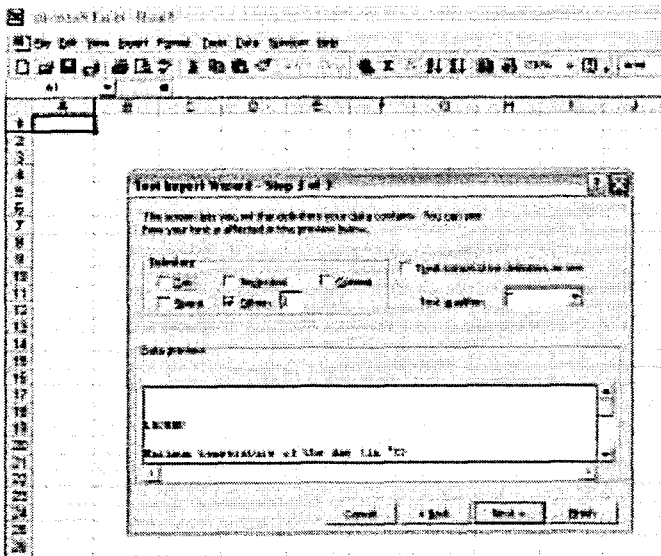


Figure E. 3: Text Import Wizard step 2

The text file is now converted and loaded in MS Excel (Figure E.4). Save the file with an appropriate filename.

Day	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
01	24.3	28.1	32.2	33.8	28.7	23.2	18.2	15.9	15.8	21.3	26.6	22.5
02	23.4	24.9	25.3	19.8	18.4	18.8	22.4	18.2	21	18.8	22.4	18.2
03	21.8	21.2	24.1	24.8	17.5	16.1	25.8	12.8	18.7	19.2	21.4	21.5
04	24.5	24.8	27.2	24.7	19.7	17.1	24.2	13.9	17.4	22.6	22.6	25.2
05	27.5	25.1	25.9	22.2	19.1	18.8	24.2	15.8	16.1	24.2	26.4	28.5
06	27.1	26.6	24.7	19.8	18.1	18.4	25.1	16.1	22.2	26.1	28.6	18.7
07	27.4	22.6	27.8	22.2	19.5	18.8	12.4	9	18.2	28.8	22.7	22.1
08	21.9	21.8	27.1	18.8	18.3	16.7	18.1	17.7	16.8	19	21.9	22.3
09	25.5	25.2	27.8	21.4	19.9	17.5	17.4	21.8	19.9	18.7	24.5	22
10	23	25.4	28.6	19.1	21.4	13.6	17.3	16.6	17.5	25.8	18	26.8
11	23.5	18.1	29	14	15.4	15.2	15.7	15.3	15.6	23.4	21.4	26.2
12	27.4	26.9	27.8	16.7	19.5	12.8	15	15.2	19.9	18.8	26.6	22.8
13	24.6	28.2	26	18.2	18.7	18.7	18.4	21	18.1	13.1	28.1	28.1
14	28.4	28.3	23.8	25	18.7	23.3	13.7	28.9	28.1	24.9	21.9	24.3
15	23.2	23.1	27.6	22	18.8	18.1	21.8	20.5	26.5	18.3	25.1	17.5
16	28.3	23.4	21.3	21.1	18.5	16.3	15.8	22	20	19	22.7	23.5
17	25.1	25.3	26.3	18.6	18.4	16.5	12.6	13.4	21.5	20.3	26	23.5
18	26.2	28.8	26.7	18.4	18.8	15.5	14	18.2	19.3	23	22.8	24
19	25.1	24.8	26.2	22.8	17.7	16.8	16.6	22.1	24.6	22	28.8	26.7
20	25.6	22.2	27.1	21.1	19.7	18.2	18.4	17.3	18.4	22.8	28.1	22.4
21	22.5	22.6	26.2	28.1	18.2	17.7	19.3	14.9	18.4	21.1	23.3	24.8
22	27.7	24.2	23.5	24.5	16.6	18.4	21	18.9	18.7	27.8	20	22.8
23	22.2	27.8	24.9	23.5	17.7	17.5	26.4	19.5	20.4	21.2	24.5	26.1
24	24.9	28.9	28.1	21.6	17.7	21.6	19.1	25.8	19.5	22.3	22.8	24.5
25	28.6	21.9	29.8	21.2	13.8	24	17.2	17.2	26.2	24.2	27	24.7
26	21.9	28.8	21.7	22	16.5	18.2	15.6	16.5	18.7	22.7	24.8	23.8
27	25.8	21	24.2	18.7	18.5	15.2	15.1	16.8	18.8	22	26.4	23.4
28	27.9	28.6	28.1	19.2	17.1	14.7	13.5	18.4	16.1	25.8	23.5	23.6
29	27.5	22.6	22.6	19.8	17.5	15.9	15.9	22.1	18.7	22.6	25.2	24.5
30	24.6	22.2	23.2	22.5	17.7	17.7	19.2	20	22.1	21.7	22.5	26
31	21.4	22.8	22.8	22.8	19.8	19.8	20	18.8	22.7	22.7	26.2	26.2

Figure E. 4: Weather file in MS Excel format

## 2 Conversion of the MS Excel into the MySQL database format

Open the MySQL DBManager by clicking on the icon on the desktop.

- Right-click on the *Databases* folder and select *Create Database* (Figure E.5)
- Create an appropriate name.
- Click on the plus sign next to the *Databases* folder and select the newly created database.
- Click on the *Tools* menu item and select *Data Management*, then select *Import Data (DAO)* in Figure E.6.
- Select the *Microsoft Excel* option from the *Data Wizard* window (Figure E.7).
- Click *Next*.
- Browse to the MS Excel file that is to be imported and select *Open* (Figure E.8).
- Click *Next*.
- Select *Sheet1\$* and click *Next*.
- Click *Next*.
- Select the correct database and click *Next*.
- Click *Finish*
- Click *OK*.
- Click *OK*.

The MS Excel file is now imported and can be seen by selecting the database and viewing the table with the exported data.

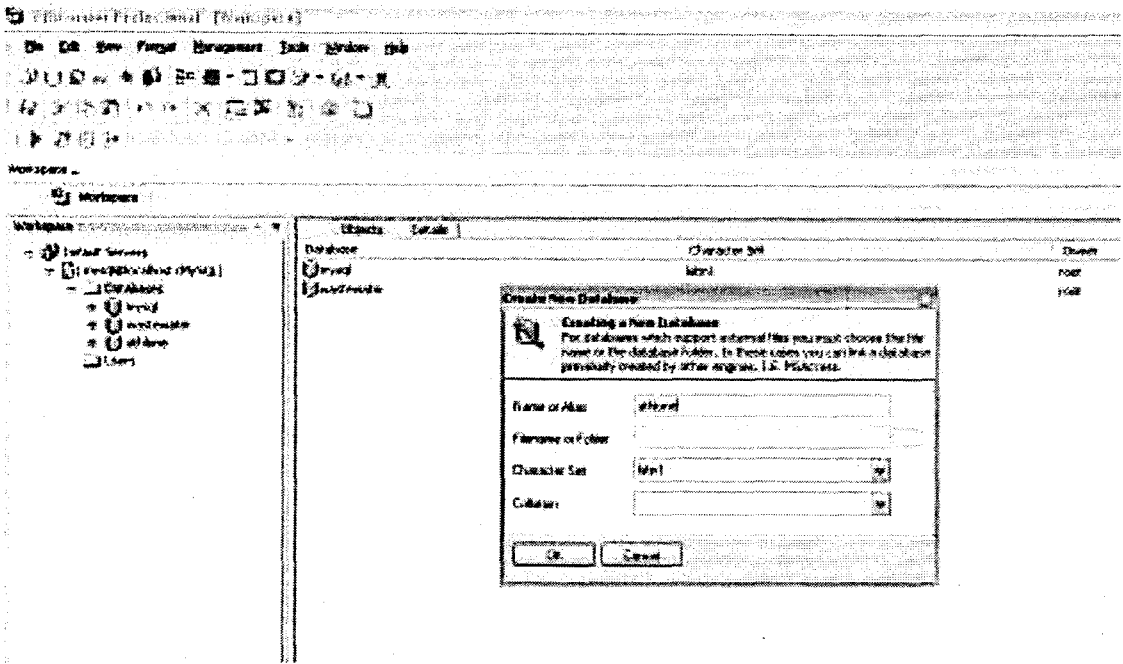


Figure E. 5: Creating a new database

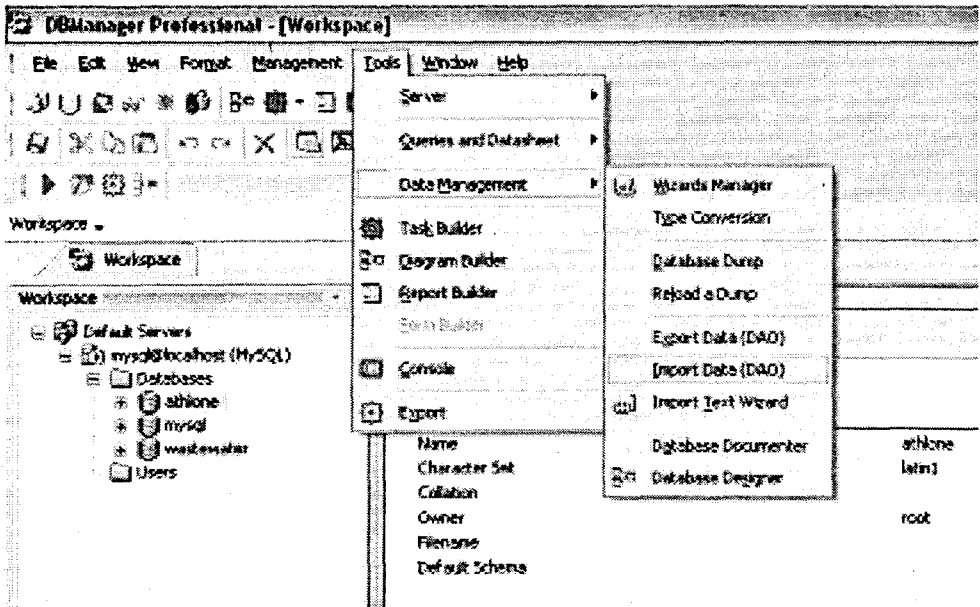


Figure E. 6: Selecting the menu item for importing the Excel data

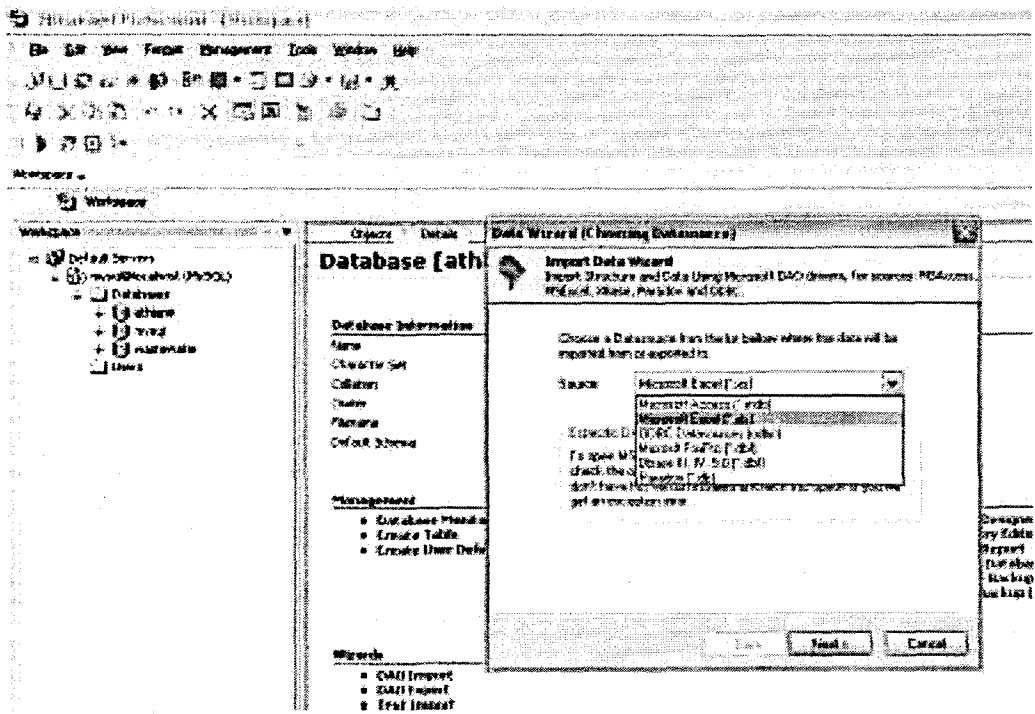


Figure E. 7: Selecting the MS Excel option from the Data Wizard window

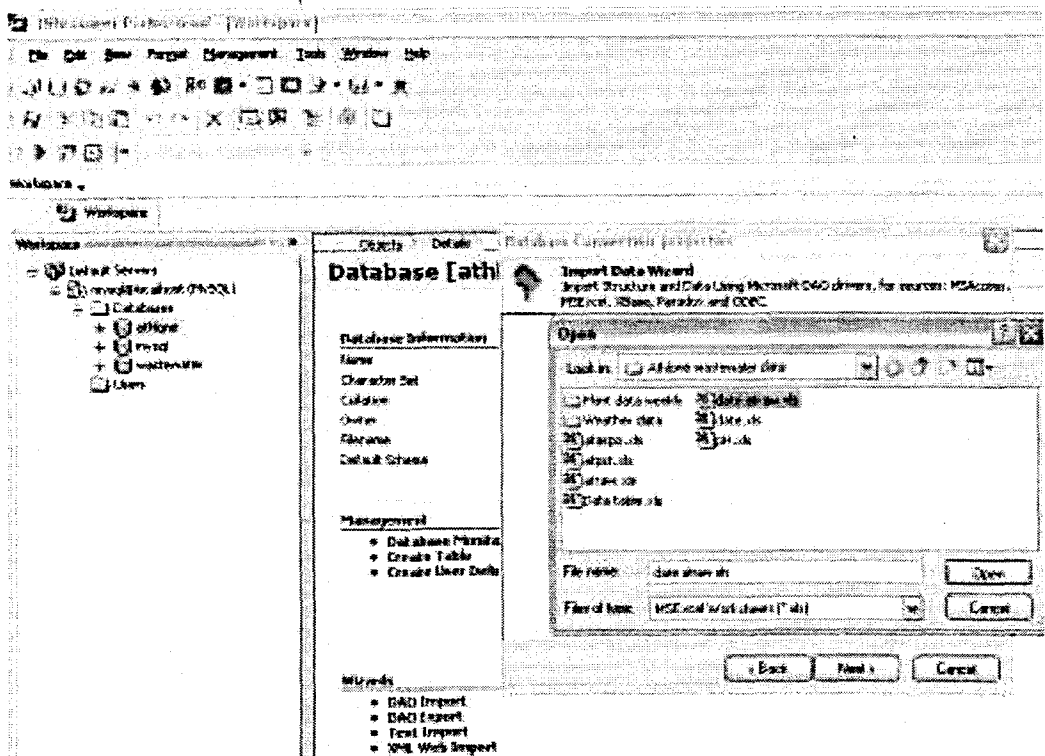


Figure E. 8: Selecting the MS Excel file that is to be imported

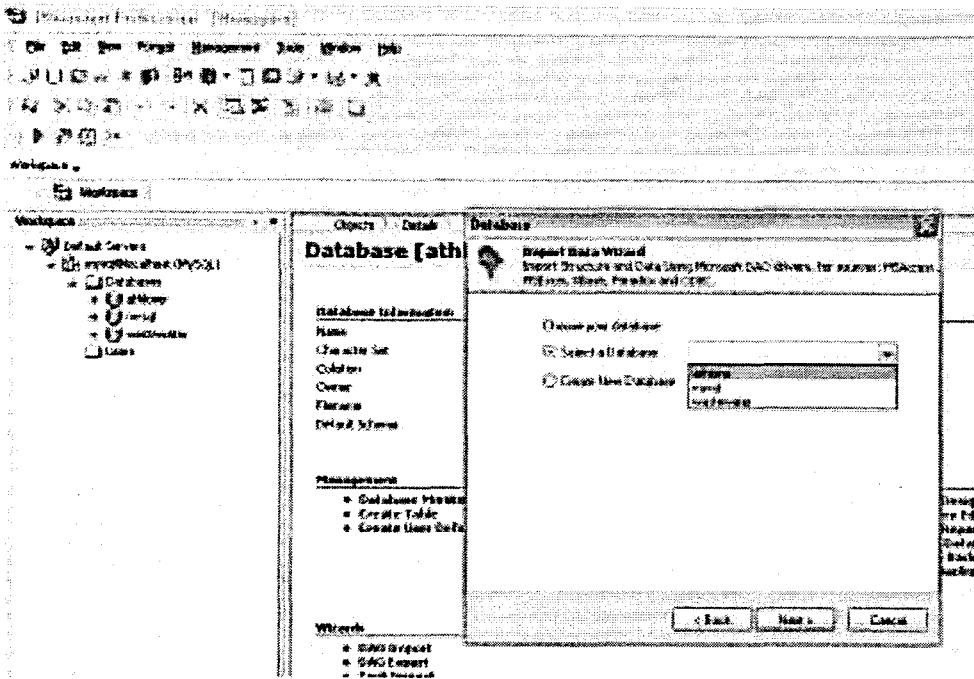


Figure E. 9: Selecting the database where the MS Excel file is to be imported

### 3 ODBC and Database setup

The ODBC properties are set up in the control panel. Please note that the following procedure will not work if the MySQL database software with all relevant components are not properly installed. The assumption is therefore made that the MySQL software is already installed. The following section presents a step-by step guide to setting up the ODBC connection to the MySQL database.

The first step is configuring the ODBC database connection

- In the Windows *Control Panel* window double click on the *Administrative Tools* icon (Figure E.10)
- Double click on the *Data Sources (ODBC)* icon in the *Administrative Tools* window. (Figure E.11)
- Click the *Add* button in the *ODBC Data Source Administrator* window (Figure E.12).
- Scroll down in the *Create New Data Source* window and select the *MySQL ODBC 3.51 Driver* (Figure E.13).
- The user can enter a unique *Data Source Name (DSN)*, *Server*, *Username*, *Password* and *Database* name in the following dialogue box. The successful connection to the database can be confirmed by clicking on the *Test* button in the *Connector/ODBC Add Data New Data Source* window (Figure E.14).
- Click OK.
- The unique DSN (athlone) should now appear together with the appropriate driver (MySQL ODBC 3.51 Driver) under the *User DSN* tab (Figure E.15).



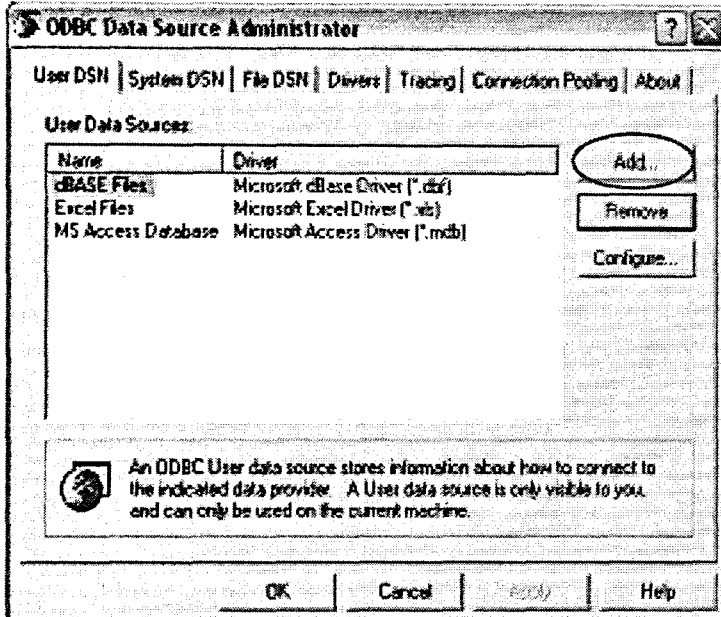


Figure E. 12: The ODBC Data Source Administrator Window

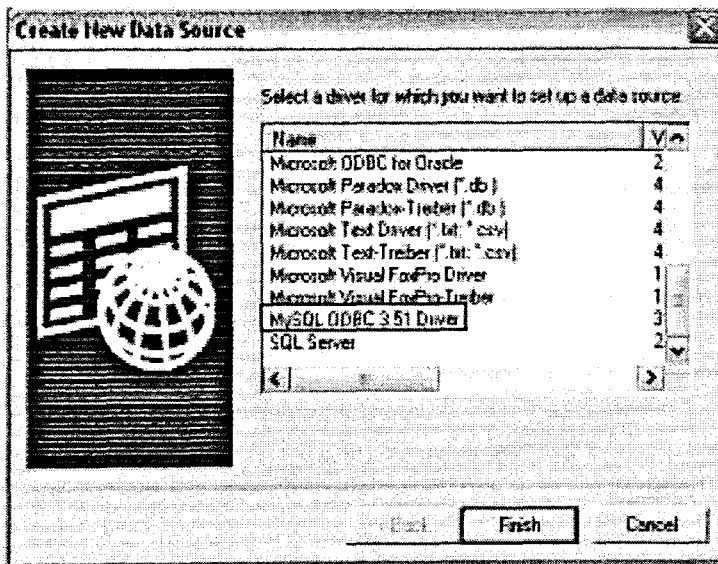


Figure E. 13: The Create New Data Source Window

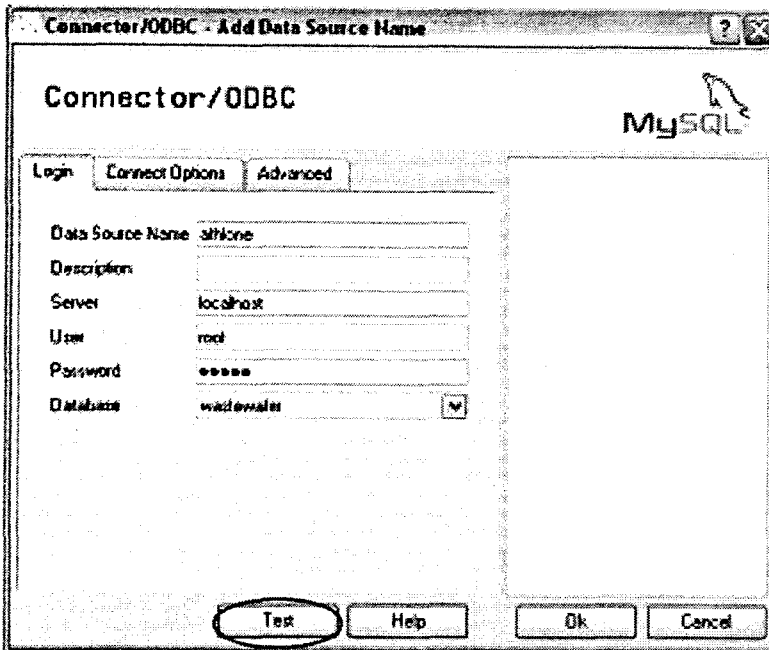


Figure E. 14: The Connector/ODBCAdd Data New Data Source Window

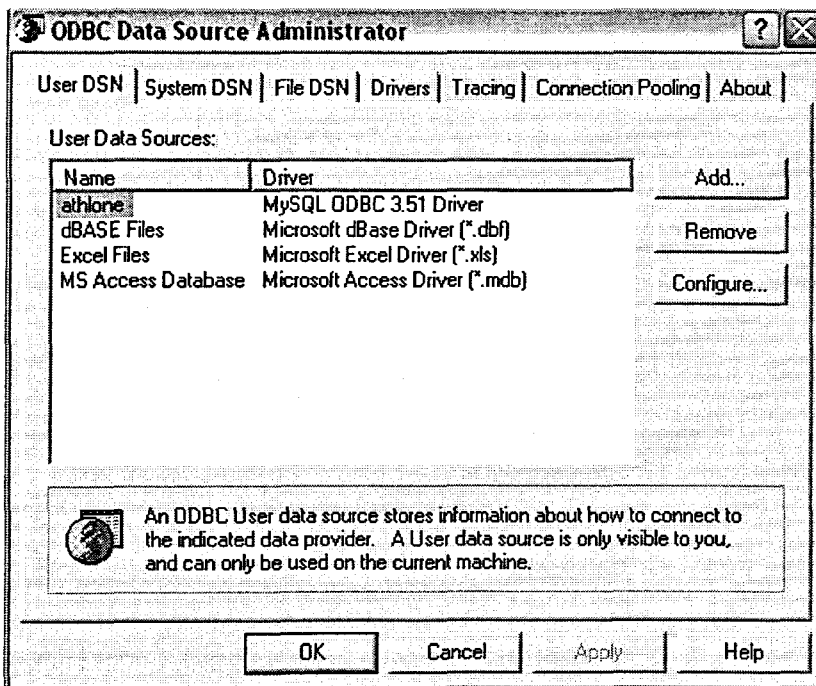


Figure E. 15: The ODBC Data Source Administrator Window showing the new MySQL driver



## REFERENCES

- Ackley, D.H., Hinton, G.F., and Sejnowski, T.J. 1985. "A learning algorithm for Boltzman machines," *Cognitive Science*, 9: 147-169.
- Adya, M., Collopy F. 1998. *How effective are Neural Networks at Forecasting and Prediction? A Review and Evaluation*. *Journal of Forecasting* pp 481-495.
- Ahmida, Z., Charef, A., Becerra, V. 2007. "Stable Nonlinear Receding Horizon Regulator using RBF Neural Network Models", Proc 14<sup>th</sup> Mediterranean Conference on Control and Automation.
- Albus, J.S 1975. "New Approach to Manipulator Control: The Cerebella Model Articulation Controller (CMAC)," *Transactions of the ASME Journal of Dynamic Systems, Measurement, and Control*, September 1975, 220-27.
- Alex, J., Ogurek, M., Jumar, U. 2005. *Formalised Model Representation for Wastewater Systems*. Proceedings of 16<sup>th</sup> IFAC World Congress in Prague DVD ROM.
- Anderson, J.A., Silverstein, J.W., Ritz, S.A., and Jones, R.S. 1977. "Distinctive features, categorical perception, and probability learning: Some applications of a neural model," *Psychological Review*, 84, 413-451. Reprinted in Anderson and Rosenfeld 1988.
- Anderson, J.A., and Rosenfeld, E., eds. (1988), *Neurocomputing: Foundations of Research*, Cambridge, MA: The MIT Press.
- Antognetti, P., Milutinović. 1991. *Neural Networks: Concepts, Applications, and Implementations*, Volume I, Prentice Hall, Englewood Cliffs, New Jersey.
- Arthur, R.M. 1982. "Application of On-line Analytical Instrumentation to Process Control". Proceedings of the First Annual Conference on Activated Sludge Process Control. Ann Arbor Science Publishers, Michigan.
- Barnett, M.W. 1999. *Neural nets take on a dirty job*, Gensym Corp. Cambridge, Massachusetts.
- Bartlett, R.E. 1971. Design in metric wastewater treatment, pp 24-37
- Bechmann, H., Nielsen, M., Madsen, H., Poulsen, N. 1998 "Control of sewer systems and wastewater treatment plants using pollutant concentration profiles", *Water Science Technology*, Vol 37 No.12, pp 87-93.
- Bellman, R. (1961), *Adaptive Control Processes: A Guided Tour*, Princeton University Press.
- Betteley, G., Mettrick, N., Sweeney, E., Wilson, D. 1994. *Using Statistics in Industry, Quality Improvement Through Total Process Control*, Prentice Hall International (UK) Ltd.

Bishop, C.M. 1995. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press.

Bishop, C.M., Svensén, M., and Williams, C.K.I 1997. "GTM: A principled alternative to the self-organizing map," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambridge, MA: The MIT Press, pp. 354-360. Also see <http://www.ncrg.aston.ac.uk/GTM/>  
Bitton, G.1999. *Wastewater Microbiology*, Second Edition. pp 56

Brown, M., and Harris, C. 1994. *Neurofuzzy Adaptive Modelling and Control*, NY: Prentice Hall.

Burton, M. 2006. "Neural Networks", Course Notes. Rhodes University Department of Mathematics.

Cannady J. 1998. "Artificial Neural Networks for Misuse Detection". National Information Systems Security Conference.

Carlsson, B., Lindberg C.F. 1996. "Some control strategies for the activated sludge process," Systems and Control Group Uppsala University. Revised 1997, 1998.

Carpenter, G.A., Grossberg, S. 1987a. "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, 37, 54-115.

Carpenter, G.A., Grossberg, S. 1987b. "ART 2: Self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, 26, 4919-4930.

Carpenter, G.A., Grossberg, S. 1990. "ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3, 129-152.

Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H., and Rosen, D.B. 1992. "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Transactions on Neural Networks*, 3, 698-713

Carpenter, G.A., Grossberg, S., Reynolds, J.H. 1991. "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," *Neural Networks*, 4, 565-588.

Carpenter, G.A., Grossberg, S., Rosen, D.B. 1991a. "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition," *Neural Networks*, 4, 493-504.

Carpenter, G.A., Grossberg, S., Rosen, D.B. 1991b. "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, 4, 759-771.

- Casey, M. 2004. Neural Networks CS365 Course - University of Surrey, Guildford.  
(see also  
<http://portal.surrey.ac.uk/pls/portal/docs/PAGE/COMPUTING/RESOURCES/L3/CS365/CS365%20INTRODUCTION.PDF>)
- Cavalletti, M., Ippoliti, G., Longhi, S. 2007. "Multiple Model Control Using Neural Networks for a Remotely Operated Vehicle", Proc 14<sup>th</sup> Mediterranean Conference on Control and Automation.
- Chapple, M., Database Fundamentals.  
<http://databases.about.com/od/administration/a/databasefund.htm> [July 2004].
- Chang, F., Liang, J., Chen, Y. 2001. "Flood forecasting using Radial Basis Function Neural Networks", IEEE Transactions on Systems, Man and Cybernetics, Part C Applications and Reviews, Volume 31 No. 4.
- Chen, S., Cowan, C.F.N., and Grant, P.M. 1991. "Orthogonal least squares learning for radial basis function networks," IEEE Transactions on Neural Networks, 2, 302-309.
- Cheng, T., Lewis, F., Abu-Khalaf, M. 2007. "A Neural Network Solution for Fixed-Final Time Optimal Control of Nonlinear Systems", Proc 14<sup>th</sup> Mediterranean Conference on Control and Automation.
- Cichocki, A. and Unbehauen, R. 1993. *Neural Networks for Optimization and Signal Processing*. NY: John Wiley & Sons, ISBN 0-471-93010-5.
- Cowan, J. (1967). "A Mathematical Theory of Central Nervous Activity", *unpublished PhD. Dissertation, University of London*.
- Danh N.T., Phien H.N., Gupta A.D. 1998. Neural Network models for river flow forecasting, Water SA, Vol 25(1), pp 33 - 39.
- Demuth, H., Beale, M., Hagan, M. 2004. Neural Network Toolbox For Use with MATLAB, Users Guide Version 4.
- Desieno, D. 1988. "Adding a conscience to competitive learning," Proc. Int. Conf. on Neural Networks, I, 117-124, IEEE Press.
- Dold, P., Wentzek, M., Billing, A., Ekama, G., Marais, R 1991. Activated Sludge Simulation Programs, Water Research Commission, South Africa.
- Diamantaras, K.I., and Kung, S.Y. 1996. *Principal Component Neural Networks: Theory and Applications*, NY: Wiley.
- Ekama, G.A. Marais, G. 1984. "Theory, design and operation of nutrient removal Activated Sludge Processes". *Co-ordinating Research and Development Committee for Sewage Purification*. South Africa. pp 9-34, pp 46-55.
- Elman, J.L. 1990. "Finding structure in time," Cognitive Science, 14, 179-211.

Fahlman, S.E. 1989. "Faster-Learning Variations on Back-Propagation: An Empirical Study", in Touretzky, D., Hinton, G, and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 38-51.

Fahlman, S.E., and Lebiere, C. 1990. "The Cascade-Correlation Learning Architecture", in Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*, Los Altos, CA:

Fausett, L. 1994. *Fundamentals of Neural Networks*, Englewood Cliffs, NJ: Prentice Hall.

Fruchard, M. Bernard, O. Gouze, J. 2002. Interval Observers with Guaranteed Confidence Levels Application to the Activated Sludge Process. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1

Fuente, M.J. Sainz Palmero, G.I. Pindado, D. 2002. Automatic Fuzzy Rule Generation in a Biotechnological Process. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 2

Fukushima, K., Miyake, S., and Ito, T. 1983. "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 826-834.

Fukushima, K. 1988. "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, 1, 119-130.

Gao, Y., Er, M. 2005. "Soft Computing Approach For Time-Series Prediction", *Proceedings of 16<sup>th</sup> IFAC World Congress in Prague DVD ROM*.

Garcia-Bartual R. 2002. *Short term river flood forecasting with neural networks*. In the proceedings of iEMSs 2002. Switzerland.

Garvey, E.B. 1997. On-line Quality Control of Injection Molding Using Neural Networks. Minor Thesis Department of Computer Science, Royal Melbourne Institute of Technology, Melbourne Australia.

Gernaey, K.2000. Sensors for nitrogen removal monitoring in wastewater treatment. *Laboratory for Microbial Ecology*. University of Gent. *Department of Applied Mathematics, Biometrics and Process Control (BIOMATH)*. University of Gent. pp 2

Grossberg, S. 1976. "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, 23, 121-134

Gutierrez, G. Vega, P. 2002. Integrated Design of Biological Processes and their Control System Including Closed Loop Properties for Disturbances Rejection. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1

Hand, D.J. 1982. *Kernel Discriminant Analysis*, Research Studies Press.

- Haykin, S. 1999. *Neural Networks, A Comprehensive Foundation*. Second Edition, Prentice-Hall, Inc, Ontario, Canada.
- Hebb, D.O. 1949. *The Organization of Behavior*, NY: John Wiley & Sons.
- Hecht-Nielsen, R. 1987. "Counterpropagation networks," *Applied Optics*, 26, 4979-4984.
- Hecht-Nielsen, R. 1988. "Applications of counterpropagation networks," *Neural Networks*, 1, 131-139.
- Hecht-Nielsen, R. 1990. *Neurocomputing*, Reading, MA: Addison-Wesley.
- Henze, M., Grady Jr. C.P.L., Gujer W., Marais G.R., and Matsuo, T. 1987. *Activated Sludge Model No.1*. IAWQ Scientific and Technical Report No. 1, IAWQ, London, UK.
- Henze, M., Gujer W., Takahasi, M., Matsuo, T., Wentzel, M.C., and Marais G.R. 1995. *Activated Sludge Model No.2*. IAWQ Scientific and Technical Report No. 3, IAWQ, London, UK.
- Hertz, J., Krogh, A., and Palmer, R. 1991. *Introduction to the Theory of Neural Computation*. Addison-Wesley: Redwood City, California.
- Hong, Y.-S., Rosen, M.R., and Bhamidimarri, R. 2003. Analysis of a municipal wastewater treatment plant using a neural network-based pattern analysis. *Water Research*, 37, 1608-1618.
- Hopfield, J.J. 1982. "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, 79, 2554-2558. Reprinted in Anderson and Rosenfeld 1988.
- Hornik, K., Stinchcombe, M., White, H. 1990. *Neural Networks*, pp2, 359.
- Husmeier, D. 1999. *Neural Networks for Conditional Probability Estimation, Forecasting Beyond Point Predictions*, Springer-Verlag London, Ltd.
- Jeppson, U. 1996. "A General Description of the Activated Sludge Model No. 1 (ASM1)" taken from the PhD thesis: "Modelling aspects of wastewater treatment processes. Lund Institute of Technology, ISBN 91-88934-00-4
- Kabundi, A.N. 2002. "Macroeconomic Forecasting: A Comparison between Artificial Neural Networks and Economic Models", Dissertation submitted for degree of Master of Commerce at Rand Afrikaans University, South Africa.
- Kasuba, T. 1993. "Simplified Fuzzy ARTMAP," *AI Expert*, 8, 18-25.
- Kendall, M., and Stuart, A. 1976. *The Advanced Theory of Statistics, Volume III*. Hafner Publishing Co., New York.
- Khan, R and Ondrůšek, Č. 2000. "Peak electric load forecasting using an artificial neural network", submitted for publishing in *Engineering Mechanics - International Journal of Theoretical and Applied Mechanics*, Technical University of Brno, Czech

Republic, 5<sup>th</sup> October 2000.

KİŞİ, Ö. 2005. Daily River Flow Forecasting Using Artificial Neural Networks and Auto-Regressive Models. *Turkish Journal of Engineering Environmental Science* pp9-20.

Kruger, C., Tzoneva. R. 2007a. " A Neural Network Model for Control of Wastewater Treatment Processes, Proceedings of International Conference NOLCOS 2007, DVD ROM.

Kruger, C., Tzoneva. R. 2007b. "Neural Networks for Prediction of Wastewater Treatment Plant Influent Disturbances", Proceedings of 'IEEE AFRICON' Conference, Windhoek, Namibia DVD ROM.

Kohonen, T. 1984. *Self-Organization and Associative Memory*, Berlin: Springer.

Kohonen, T. 1988. "Learning Vector Quantization," *Neural Networks*, 1 (suppl 1), 303.

Kohonen, T. 1995/1997. *Self-Organizing Maps*, Berlin: Springer-Verlag. First edition was 1995, second edition 1997.

Kosko, B. 1992. *Neural Networks and Fuzzy Systems*, Englewood Cliffs, N.J.: Prentice-Hall.

Kotsiantis, S.B., Kanellopoulos, D., Pintelas, P.E. 2006. "Data Preprocessing for Supervised Learning," *International Journal of Computer Science* Volume 1 Number 2 ISSN 1306-4428.

Kröse, B., van der Smagt, P. 1996. *An Introduction to Neural Networks*, Eighth edition, The University of Amsterdam.

Lang, K. J., Waibel, A. H., and Hinton, G. 1990. "A time-delay neural network architecture for isolated word recognition," *Neural Networks*, 3, 23-44.

Le Cun, Y. (1986). "Learning Processes in an Asymmetric Threshold Network", In Bienenstock, E., Fogelman-Souli, F and Weisbuch, G., eds. *Disordered Systems and Biological Organization*. NATO ASI Series, F20, Berlin: Springer-Verlag.

Levin, A Narendra, K. 1995: "Recursive identification using feedforward neural networks". *International Control*, Vol. 61, No.3, pp 533

Lilley, I.D., Pybus, P.J., Power, S.P.B. 1997. "Operating Manual for Biological Nutrient Removal Wastewater Treatment Works" Prepared for the Water Research Commission. WRC Report No. TT 83/97.

Lubenova, V. Simeonov, I. Queinnec, I. 2002. Two-Step Parameter And Estimation of the Anaerobic Digestion. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1-6.

Lukasse, L.J.S. 2001. *Control and Identification in Activated Sludge Processes*. PhD Thesis.

Malasri, K., Malasri, S. 2002. Backpropagation Networks for Time Series Forecasting: Case Studies in Data Modelling. Christian Brothers University. Mid-South Annual Engineering and Sciences Conference. *Proceedings of the MAEDC 2002 Conference*

Mandic, D.P., Chambers, J.A. 2001. *Recurrent Neural Networks for Prediction*, John Wiley & Sons, Ltd.

Mapisa, P. 2004. "Data acquisition system for monitoring of process variables in a biological nutrient removal of wastewater treatment plant", unpublished B-Tech Thesis, Cape Peninsula University of Technology

Masters, T. 1993. *Practical Neural Network Recipes in C++*, San Diego: Academic Press.

McCord, N.M., Illingworth W.T. 1991. *A Practical Guide to Neural Nets*, Addison-Wesley Publishing Company.

McCulloch, W.S. & Pitts, W. 1943. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5, 115-133. Reprinted in Anderson & Rosenfeld (1988).

Medsker, L.R., and Jain, L.C., eds. 2000. *Recurrent Neural Networks: Design and Applications*. Boca Raton, FL: CRC Press, ISBN 0-8493-7181-3.

Metcalf and Eddy. 1991. *Wastewater Engineering*, 3rd ed., McGraw-Hill, New York.

Mihai, G. 2002. Neural Network-based forecasting for electricity markets. *Telmark Discussion Forum 1: Technology* (2-4 Sep 2002).

Minsky, M.L., and Papert, S.A. 1969/1988. *Perceptrons*, Cambridge, MA: The MIT Press (first edition, 1969; expanded edition, 1988).

Moody, J. and Darken, C.J. 1989. "Fast learning in networks of locally-tuned processing units," *Neural Computation*, 1, 281-294.

Moore, B. 1988. "ART 1 and Pattern Clustering," in Touretzky, D., Hinton, G. and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, 174-185, San Mateo, CA: Morgan Kaufmann.

Morabito F.C. 1997. *Advances in Intelligent Systems*, IOS Press.

Muhammad R.K., Čestmír O. 2000. "Peak electric load forecasting using an artificial neural network", submitted for publishing in *Engineering Mechanics - International Journal of theoretical and applied mechanics*, Technical University of Brno, Czech Republic.

Mulier, F. and Cherkassky, V. 1995. "Self-Organization as an Iterative Kernel Smoothing Process," *Neural Computation*, 7, 1165-1177.

MySQL AB. Why MySQL?, (<http://www.mysql.com/why-mysql>)

Nadaraya, E.A. 1964. "On estimating regression", *Theory Probability. Application.* 10, 186-90.

Nadri, M. Hammouri, H. Fick, M.2002. Nonlinear Observers for State and Parameter Estimation in Biochemical Processes. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 2

Narendra, K.S., Parthasarathy, K. 1990. Identification and Control of Dynamical Systems using Neural Networks. IEEE Transactions on Neural Networks. Vol.1, pp 7-27.

Ng G.W. 1997. Application of Neural Networks to Adaptive Control of Nonlinear Systems, Research Studies Press Ltd, John Wiley and Sons Inc.Singapore

Neural Networks FAQ. 2005. <ftp://ftp.sas.com/pub/neural/FAQ.html>.

Nguyen, D., and B. Widrow. 1990 "Improving the learning speed of 2-layer Neural Networks by choosing initial values of the adaptive weights," *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, pp. 21–26.

Nyberg, U. Aspegren, H. Andersson, B. Jansen, J.la C. Villadsen, I.S. 1992. Full-scale application of nitrogen removal with methanol as carbon source. *Water Science Technology*, 26(5-6) pp 1077 – 1086.

Oja, E. 1989. "Neural networks, principal components, and subspaces," *International Journal of Neural Systems*, 1, 61-68.

Olssen, G., and Newell, B. 1999. *Wastewater Treatment Systems. Modelling, Diagnosis and Control*. IWA Publishing, London, UK.

Orr, M.J.L. 1996. "Introduction to radial basis function networks," <http://www.anc.ed.ac.uk/~mjo/papers/intro.ps> or <http://www.anc.ed.ac.uk/~mjo/papers/intro.ps.gz>

Ostafe, D. 2005. "Neural Network Hidden Layer Number Determination Using Pattern Recognition Techniques". 2<sup>nd</sup> Romanian-Hungarian Joint Symposium on Applied Computational Intelligence SACI 2005.

Pao, Y. H. 1989. *Adaptive Pattern Recognition and Neural Networks*, Reading, MA: Addison-Wesley Publishing Company, ISBN 0-201-12584-6.

Papliński, A.P. 2004. Lecture notes for Course 5301 "Neural Networks and Fuzzy Systems (Neuro-Fuzzy Computing)", Clayton School of Information Technology Monash University, Australia.

Park, M.H. 1996. Neural Network Control of a Chlorine Basin, University of California pp 20-52.



- Parker, D. 1985. "Learning Logic", Technical Report TR-87, Cambridge, MA: Center for Computational Resesarch in Economics and Management Science MIT.
- Pérez-Llera, C., Fernández-Baizán M.C., Feito J.L., González del Valle V. 2002. Local Short-Term Prediction of Wind Speed: A Neural Network Analysis. The International Environmental Modelling and Software Society.
- Pons, M. Potier., O. 2002. Control Strategy Robustness With Respect To Hydraulic Model Sophistication. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1
- Potocnik, P.2000: Adaptive self-tuning neurocontrol. *Faculty of Mechanical Engineering*. University of Ljubljana, Slovenia. pp 1-11.
- Rech, G. 2002. Forecasting with artificial neural networks. SSE/EFI Working Paper Series in Economics and Finance, No 491.
- Ripley B.D. 1996. Pattern Recognition and Neural Networks, Cambridge University Press.
- Reed, R.D., and Marks, R.J, II 1999. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.
- Riedmiller, M. and Braun, H. 1993. "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *Proceedings of the IEEE International Conference on Neural Networks 1993*, San Francisco: IEEE.
- Rosenblatt, F. 1958. "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological Review*, 65, 386-408.
- Rui, Y., El-Keib, A.A. 1995. A Review of ANN-Based Short-Term Load Forecasting Models, *Proc of 27th IEEE Southeastern Symposium on System Theory*, MSU, Mississippi.
- Rumelhart, D.E., Hinton, G.E., and Williams, R.J. 1986. "Learning internal representations by error propagation", in Rumelhart, D.E. and McClelland, J. L., eds. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1, 318-362, Cambridge, MA: The MIT Press.
- Sakai, M. Homma, N. Abe, K. 2002: A Statistical Approximation Learning Method for Simultaneous Recurrent Networks. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1-5.
- Samuelsson, P. 2001. "Modelling and Control of Activated Sludge Processes with Nitrogen Removal".
- Sánchez-Marrè, M., Cortés, U., Lafuente, J., Roda, I. R., Poch, M. 1995. "Knowledge-Based Techniques in Wastewater Treatment Plants Management". Submitted to *Encyclopedia of Life Support Systems (EOLSS)*. September,.

- Sanger, T.D. 1989. "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, 2, 459-473.
- Specht, D.F. 1990. "Probabilistic neural networks," *Neural Networks*, 3, 110-118.
- Specht, D.F. 1991. "A Generalized Regression Neural Network", *IEEE Transactions on Neural Networks*, 2, Nov. 1991, 568-576.
- StatSoft. 2003. *Neural Networks*, <http://www.statsoft.com/textbook/stneunet.html>.
- Tarassenko, L. 1998. "A Guide To Neural Computing Applications", John Wiley and Sons, New York.
- Teng, C.M. 1999. "Correcting Noisy Data" *Proceedings of 16<sup>th</sup> International Conference on Machine Learning*, pp 239-248. San Francisco.
- Teräsvirta, T., van Dijk, D. 2004. Linear models, smooth transition autoregressions, and neural networks for forecasting macroeconomic time series: A re-examination.
- Theilliol, D., Ponsart, J.C., Harmand, J., Join, C., Gras, P. 2003. "One-line estimation of unmeasured inputs for anaerobic wastewater treatment processes." *Control Engineering Practice*, vol. 11, pp. 1007-1019,
- Vahed, A. 2004. "Knowledge-Driven Training of Recurrent Neural Networks with Application to Time Series Modelling", PhD Thesis presented at the University of the Western Cape, South Africa
- Vanrolleghem, P.A. 2000. *Sensors for Anaerobic Digestion: An overview*. BIOMATH. University of Gent. pp 1-9.
- USGS, June 2006. *Wastewater Treatment Water Use*.  
<http://ga.water.usgs.gov/edu/wuww.html>
- VARGAS, A. MORENO, J. ZEITZ, M. 2002. Order Extension of Nonlinear Systems for Observer Design Under Reduced Observability. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1
- Vermaak J, Botha E.C. 1998. *Recurrent Neural Networks for Short-Term Load Forecasting*.
- Waibel A, Hanazawa T, Hinton G, Shikano K and Lang K. 1989. Phoneme recognition using time-delay neural networks. *IEEE Transactions Acoustics, Speech and Signal Processing* 37, 328-339.
- Wan, E.A. 1990. "Temporal backpropagation: An efficient algorithm for finite impulse response neural networks," in *Proceedings of the 1990 Connectionist Models Summer School*, Touretzky, D.S., Elman, J.L., Sejnowski, T.J., and Hinton, G.E., eds., San Mateo, CA: Morgan Kaufmann, pp. 131-140.
- WANG, J., HU, S. 2002. "A Multilayer Recurrent Neural Network For Real-Time Robust Pole Assignment in Synthesizing Output Feedback Control Systems",

*Proceedings of IFAC World Congress*, Barcelona, Spain. pp 2-4.

Watson, G.S. 1964. "Smooth regression analysis", *Sankhya - The Indian Journal of Statistics, Series A*, 26, 359-72.

Werbos, P.J. (1974/1994), *The Roots of Backpropagation*. NY: John Wiley and Sons.

Widrow, B., and Hoff, M.E., Jr. 1960. "Adaptive switching circuits," IRE WESCON Convention Record. part 4, pp. 96-104. Reprinted in Anderson and Rosenfeld (1988).

Williams, R.J., and Zipser, D. 1989. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, 1, 270-280.

Williamson, J.R. 1995. "Gaussian ARTMAP: A neural network for fast incremental learning of noisy multidimensional maps," Technical Report

Wu, H., Jagannathan, S. 2007. "Adaptive Neural Network Control and Wireless Sensor Network-based Localization for UAV Formation", Proc 14<sup>th</sup> Mediterranean Conference on Control and Automation.

Wu S-I 1995 Mirroring our thought processes. *IEEE Potentials* 14, 36-41.

Yoo, C. Lee, H. Lee, I. 2002. Comparisons of Process Identification methods and Supervisory DO Control in the Full -Scale Wastewater Treatment Plant. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 2-6.

Yu, D.I. Yu, D.W. Gomm, J.B. Williams, D.2002. Model Predictive Control Of a Chemical Process Based on an Adaptive Neural Network. *Proceedings of IFAC World Congress*, Barcelona, Spain. pp 1-6.

Živčák Dalibor, Electricity load forecasting using Artificial Neural Networks.  
<http://neuron.tuke.sk/competition/reports/DaliborZivcak.pdf>, 08-2004.

Zupan, J., Gasteiger, J. 1993. *Neural Network for Chemists*, VCH Publishers, New York, NY